Novel Directions in Network Steganalysis

Jun O Seo

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science, the University of Auckland, 2022.

Declaration

To the best of my knowledge and belief, this thesis contains no material previously published by any other person, except where acknowledged. The work presented in the thesis is the outcome of my study under the supervision of Dr Sathiamoorthy Manoharan and Dr Ulrich Speidel. This thesis, in whole or in part, has not previously been submitted or accepted for award of a degree or diploma in any other institutions.

July 21, 2022

Jun O Seo

Abstract

Network steganography is the art of exploiting network protocols or network flows to innocuously hide information. Network steganalysis is the study of analysing network traffic and flows to prevent any illicit use of steganography.

Using statistical metrics to compare malicious and matching benign flows has been a solid methodological approach in network steganalysis. This approach may not work in many practical situations, however, because it is difficult to acquire both malicious and matching benign flows. A fundamental question thus inspires this thesis: What if we only have the malicious flows on hand? That is, what if we do not have access to the matching benign flow so there is nothing to compare against? Moreover, while it is critical to detect the fraction of malicious flows with a steganalysis technique, there is a lack of measurement on how much damage malicious flows cause. This leads to another question: Can we estimate how much information a malicious flow contains, thereby indicating potential damage? This thesis investigates the use of complexity derivates and a re-embedding technique to answer the two fundamental questions posed above. The experiments presented here show that it is possible to detect and estimate the amount of malicious information accurately in a number of different scenarios. However, this method is semi-automatic and relies on a significant amount of manual work, making it impractical for large-scale networks that may generate a significant number of network flows. Therefore, this thesis investigates and proposes a number of approaches to fully automate the process.

Acknowledgements

First and foremost, I would like to express my deepest appreciation to my supervisors Dr Sathiamoorthy Manoharan and Dr Ulrich Speidel, for their support and guidance throughout my research. Mere words are not enough to describe the amount of invaluable advice and constructive criticisms that I received, and I would like to thank them from the bottom of my heart.

I would like to extend my sincere thanks to fellow PhD students, staff members of the Department of Computer Science, and the ICT team, especially Yu-Cheng Tu, for sharing exciting ideas and keeping me company. Special thanks to the Group Services team in the Department of Computer Science, especially Robyn Young, for all the administrative help. Thanks also to all my friends who made my life stimulating and enjoyable.

Last but not least, I would like to thank my family and extended family for their continuous support and encouragements. I must thank my parents, Gwonjong Seo and Jeongog Yang, for their unconditional love and trust.

Jun O Seo

Contents

1

Introduction

Steganography is the art of hiding a message in innocuous digital media. Such secret messages are known as *steganograms*. Network steganography allows one to hide steganograms using the computer network as a medium with the hope that adversaries cannot detect such communication, thus creating a *covert channel*. It appeals itself as a potential communication tool but this tool has evidently been used for malicious and illegal activities [1, 2, 3]. Thus makes *network steganalysis*, the science of detecting the presence of hidden secret messages in network flows, important.

Exploring steganalysis techniques has two strong motivations: (a) strengthening the corresponding steganography technique so that the secrecy it offers is not compromised, and (b) detecting and preventing illegitimate use of steganography.

This thesis focuses on network steganography. Compared to other forms of digital steganography, such as media steganography, the hiding medium (*cover*) offered by network steganography is generally not persistent, and in fact, short-lived. Ideally, any

steganalysis that attempts to uncover network steganography should therefore operate in a near-real-time.

It should be noted that covert timing channel and covert storage channel are timing-based steganography and storage-based steganography, respectively. These terms have been used interchangeably in the literature. Moreover, there appears no clear distinction between them. This thesis thus used both terms interchangeably.

1.1 Problem and motivation

There are two types of network traffic data one can acquire: (1) synthetic data and (2) real data measured at endpoints or intermediate nodes. Synthetic data has the advantage that it can be made to contain features that one would like to study. It may, however, miss some of the real-life features real traffic might contain if the synthesis process does not consider these features. On the other hand, while real data overcomes the latter problem, it may not always contain the features that one is interested in studying within a reasonable time period.

Comparison-based statistical measures have been a traditional approach in combating malicious network steganography uses. The technique compares benign and malicious flows based on their statistical values. In general, benign flows set a baseline and a range of decision boundaries that specify benign flows numerically. Hence, the comparison-based approach usually results in a binary classification: a flow either contains a steganogram or it does not. Such binary classification is arguably useful in assessing steganalysis performance, but we contend that more can be done through analytics. For instance, quantifying the size of the steganogram may be useful in hinting at the damage network steganography might have caused, e.g., compromised user credentials or disclosed confidential information. Furthermore, we view that the comparison-based statistical measures are generally not scalable and require frequent

maintenance. It is widely understood that two networks will never exhibit the same network behaviour because of the different environments and users. Baselines and decision boundaries calculated from one network may thus not apply to another network. The disparity may also happen on the same network [4, 5], requiring a frequent recalculation of the baselines and the decision boundaries. This leads to a secondary problem of identifying the frequency of the recalculation.

1.2 Research objectives

Comparison-based steganalysis techniques have practical limitations, such as scalability and maintainability. These limitations mostly arise from comparing flows with steganograms and *virgin data* (flows without steganograms). Hence, we will explore if steganalysis can work in the absence of virgin data. We would also like to quantify the size of the steganogram, thereby indicating the damage specific network steganography might have caused. On this basis, we formulated the following research questions to guide our research:

1. Is it possible to separate network flows that contain hidden data from flows that do not?

   (a) Which techniques could one employ for this binary classification?

   (b) How could we do steganalysis in the absence of virgin data – i.e., original flows with no steganograms – when we have access only to flows with steganograms?

2. Where a network flow contains steganograms, is it possible to identify the size of the steganograms (e.g. as a percentage of the flow data)?

3. How could one prevent network flows from carrying hidden data?

1.3 Thesis overview

The rest of the thesis is structured as follows:

We discuss the fundamental aspects of steganography in Chapter 2 and review some of the related work. Chapter 3 then introduces T-codes and their complexity derivates, as well as their potential uses in both steganography and steganalysis. In Chapter 4 we develop a storage-based steganography and detect it using a re-embedding steganalysis. Chapter 5 enhances the re-embedding steganalysis with an automatic detection capability. Chapter 6 experimentally evaluates the feasibility of other metrics besides T-entropy in re-embedding steganalysis. Chapter 7 evaluates the feasibility of a timing-based steganography over long paths. Chapter 8 summarises this thesis with our contributions and future research directions.

Listed below are the publications encompassing some of the work in this thesis:

- Jun O Seo, Sathiamoorthy Manoharan, and Aniket Mahanti. Network steganography and steganalysis – a concise review. In International Conference on Applied and Theoretical Computing and Communication Technology, pages 368–371, Bengaluru, India, July 2016. https://ieeexplore.ieee.org/document/7912025 [6]
- Jun O Seo, Sathiamoorthy Manoharan, and Aniket Mahanti. A discussion and review of network steganography. In IEEE Intl Conf on Dependable, Autonomic and Secure Computing, pages 384–391, Auckland, New Zealand, August 2016. https://ieeexplore.ieee.org/document/7588874 [7]
- Jun O Seo, Sathiamoorthy Manoharan, and Ulrich Speidel. Steganalysis of storage-based covert channels using entropy. In International Telecommunication Networks and Applications Conference, pages 1–6, Auckland, New Zealand, November 2019. https://ieeexplore.ieee.org/document/9078009 [8]
- Sathiamoorthy Manoharan, Giovanni Russello, Jun O Seo, Ulrich Speidel, and Asil

Stanikzai. On synthesizing network traces — case studies in network steganalysis and packet analysis. In IEEE Conference on Application, Information and Network Security, pages 47–52, Kota Kinabalu, Malaysia, November 2020. https://ieeexplore.ieee.org/document/9315093 [9]

- Jun O Seo, Sathiamoorthy Manoharan, and Ulrich Speidel. Automatic Detection of Storage-based Covert Channels. In International Conference on Electrical, Communication, and Computer Engineering, pages 1-7, Kuala Lumpur, Malaysia, June 2021. https://ieeexplore.ieee.org/document/9514168 [10]
- Jun O Seo, Sathiamoorthy Manoharan, and Ulrich Speidel. Feasibility Evaluation of Long-Distance Network Timing-based Covert Channels. In International Conference on Electrical, Communication, and Computer Engineering, pages 1-5, Kuala Lumpur, Malaysia, June 2021. https://ieeexplore.ieee.org/document/9514145 [11]

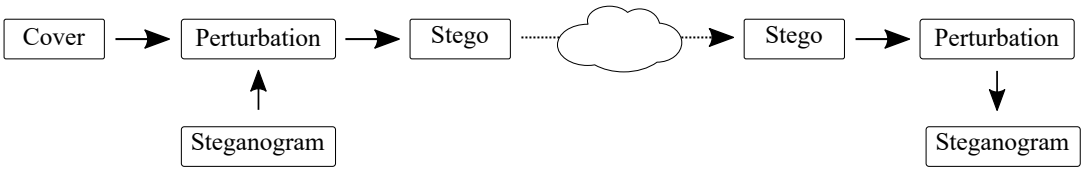
2
Background

This chapter provides a broad overview of network steganography. Sections 2.1 discusses a brief history and unique features of network steganography. Section 2.2 then discusses stakeholders of network steganography and their stance on network steganography. Sections 2.3 and 2.4 discuss design principles of network steganography and review existing network steganography, respectively. Section 2.5 reviews available countermeasures against network steganography and their limitations. Section 2.6 summarises this chapter.

2.1 History

The Histories, penned in 440 BCE, contains the earliest records of perturbing physical media to hide a secret message (*steganogram*) [12]. The hiding process perturbs an innocuous medium (*cover*), making the perturbed medium (*stego*) to almost appear

identical to its original. That is, the perturbed medium appears natural even after it goes through some form of perturbations. *The Histories* talks of shaving the head of a slave to tattoo a secret message onto the bald head then waiting for the hair to regrow before the slave travels to the message receiver. The hair naturally hides the tattooed message, rendering the secret message invisible. Once the slave reaches its destination, the message receiver shaves the slave's hair, retrieving the message. Most, if not all, of the existing steganography techniques work in the same principle, and Figure 2.1 illustrates this.



(a) Message Embedding (b) Message Extraction

Figure 2.1: Steganography process. The cover is a container that carries a secret message (steganograms). Perturbation technique embeds steganograms into the cover to generate stego. Stego travels to the message receiver, where the receiver reverses the perturbation steps to extract the steganograms.

Changes in the type of cover are one of the most noticeable shifts in steganography. In *The Histories*, humans used physical media as a cover. This trend shifted to digital media when computer systems emerged, encompassing image, voice, or video files as a cover [13]. Digital media steganography potentially offers robust and unsuspicious stego files. The steganograms, however, remain there in the cover file forever, leading to an eventual recovery of steganograms if a warden figures out the perturbation technique. Besides, the size of a cover file limits the extent of steganograms, i.e., the hiding capacity is upper bounded by the file size.

Network steganography naturally mitigates this issue, promoting itself as an attractive cover. We identify three favourable attributes: suitability, lifespan and flexibility.

Suitability as a two-way communication channel. Network steganography may be designed at any layers of the Internet protocol suite, known as a Transmission Control Protocol/Internet Protocol (TCP/IP) stack [14, 15], without exhibiting unusual network traffic. Voice over IP (VoIP), for example, is a widely used telephony application that enables voice data to flow bidirectionally for a flexible duration. Using such a trait, one may create a steganography-based two-way communication channel without raising much suspicion. HTTP (hyper-text transfer protocol) is a more widely used request-response protocol, and can be doctored to support two-way communication (for instance, using HTTP POST).

In media steganography, however, initiating two-way communication requires both parties involved in the communication to send an extraordinary number of media files back and forth, creating abnormal network traffic. As a result, media steganography may only be suitable for one-way communication.

Shorter lifespan of steganograms. Steganograms transferred over a network generally have a shorter lifespan than media steganography because the cover for network steganography is the network itself.

The steganograms inside the stego-packets reach the end of their lifespan once the *intrusion detection system* (IDS) dismisses the stego-packets from the quarantine and the receiver retrieves the steganograms.

Not all network steganography exhibit this characteristic, however, especially when steganograms persist in the network payloads. For example, if VoIP service provider archives the calls and if the calls contained steganograms, the stego-calls remain in the archive so long the provider holds onto it.

Flexible bandwidth. Media steganography bandwidth is upper-bounded by the size of a cover file. In contrast, the network bandwidth is upper-bounded by the network protocol and network environment. VoIP calls, for instance, tend to last tens of minutes or even hours, offering flexible bandwidth[1].

2.2 Stakeholders

Defining the stakeholders of steganography can can provide a number of insights – parties involved, their perspective, motivation for utilisation and detection.

Rezaei et al. categorise the stakeholders into three: government, organisation, and individuals [17]. The favourable aspect of utilising steganography channels is rather straightforward for all the stakeholders since it gives them a mean of clandestine communication channels. A few examples include:

- Transferring confidential information between trusted parties.
- Using it as a circumvention tool to prevent Internet censorship.
- Protecting privacy.

From a government's perspective, clandestine communications directly challenge their national security, e.g., terrorists may use steganography to scheme attacks [1]. Governments require steganography detection systems in place to know attacks in advance. Similarly, organisations would not want their competitors to acquire their confidential information unlawfully through industrial espionage. Likewise, individuals do not want a supposedly trusted application to compromise their sensitive information using a steganography channel. All the stakeholders would want a detection or prevention system to limit malicious activities through steganography channels, safeguarding their valuable assets and resources.

[1]There can be a constraint, however. If one utilises a G.711 VoIP codec, 50 packets travel every second, each packet containing 20 ms of voice data [16]. That is, 50 stego-packets is the bandwidth upper limit of G.711 VoIP codec.

An entity that monitors, detects or prevents such malicious activities can go by different names, but this thesis uses the word *warden* to refer to this. The art of probing techniques that the wardens use to detect steganography channels is known as *steganalysis*. A warden, however, must first understand the attack vector in detail before developing a steganalysis technique. The following section thus discusses the attacker's perspectives and the available tools at disposal in depth.

2.3 Design principles

We define a cyclic model which describes the process of designing a robust steganography technique. The cyclic model has three processes: *Location Identification*, *Steganogram Concealment*, and *Validation*, and Figure 2.2 illustrates this.

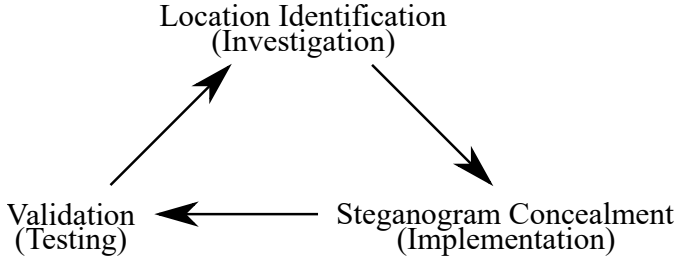
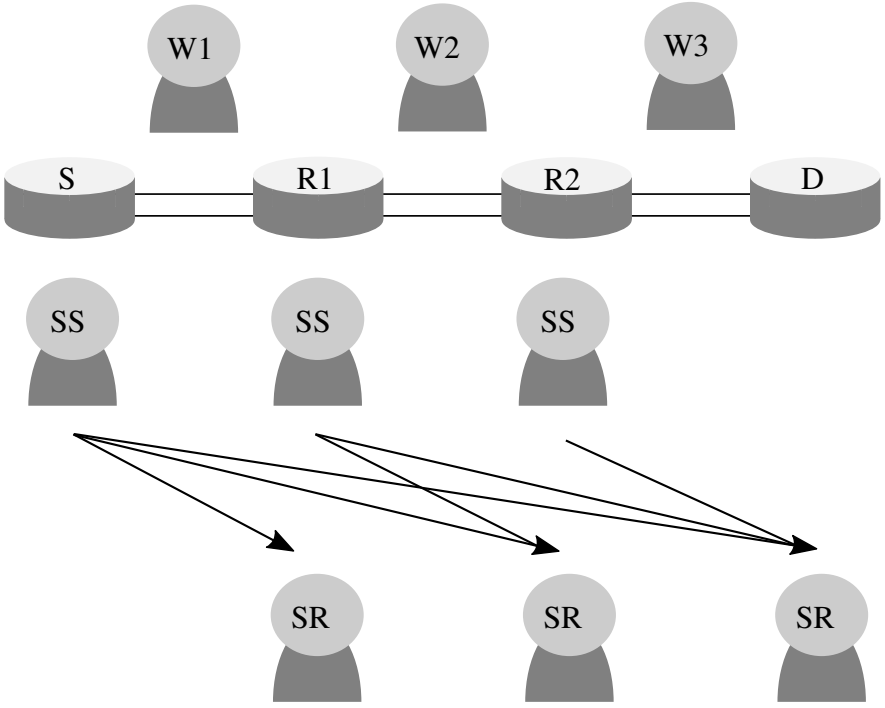


Figure 2.2: An attacker first investigates for a potential cover to hide steganograms. The attacker then selects a perturbation technique and implements their steganographic channel. The attacker validates the feasibility of their channel on its bandwidth, undetectability and robustness. If the channel does not meet the expectation, the process goes back into the investigation phase for possible improvement.

2.3.1 Location identification

Identifying an exploitable area is a primary step for designing an original steganography technique. This process requires one to have a detailed awareness of how an exploiting protocol operates. Otherwise, the resulting product may contain flaws where the detection can be straightforward.

Determining the network locations of a message sender and a receiver can provide valuable insights in designing a steganography channel. Lucena et al. analysed these locations [18], and Janicki et al. identified possible warden placements [19]. See Figure 2.3 which we merged and simplified these features. *S* denotes the source where a packet originates from, and *D* denotes its destination. *R1* and *R2* represent any number of intermediary points that could be located in-between. Any number of wardens, denoted as *W1*, *W2* and *W3*, may be present to detect malicious activities. *SS* denotes all possible locations where one could utilise steganography, and *SR* denotes all possible locations where a receiver could be.



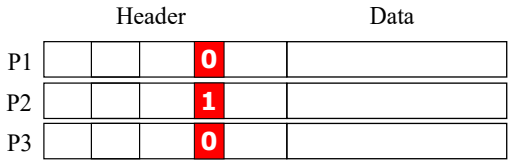
S = Source, R = Intermediary device, D = Destination, SS = Steganogram Sender, SR = Steganogram Receiver, W = Warden

Figure 2.3: Possible locations for network steganography and warden placements: A steganogram sender can be anywhere from the packet source to intermediary devices. Likewise, a steganogram receiver can be anywhere but has to be at least a hop away from the steganogram sender. A warden can be anywhere between the hops, monitoring ingress and egress packets.

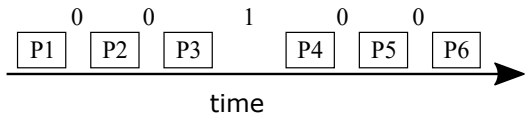
Different *W*, *SS*, and *SR* placements can impact the next process in the cyclic model because different placements suggest a different degree of requirements. Let us assume *SS* and *SR* are on *R1* and *R2*, respectively. If *W1* and *W3* monitor the network traffic, the wardens may notice a difference in the traffic, alerting the possibility of suspicious activity. A possible workaround an attacker may employ here is to reconstruct the original cover at the receiver to conceal the presence of the secret communication. As a side note, the reconstruction phase would be unnecessary if there was no *W1*, i.e., the *W3* does not have anything to compare against.

2.3.2 Steganogram concealment

There are two generic methods as well a hybrid of these two methods to exploit network protocols to conceal steganograms [20]. The first method, storage-based channels, encompass steganography techniques whereby steganograms are embedded directly into a cover, e.g., header fields of network packets or network payload bytes. On the other hand, the second method perturbs timing variations in network events to symbolically embed steganograms, e.g., modulating different interpacket delays. See Figure 2.4 which exemplifies this. A hybrid of the storage-based and timing-based methos is also possible.



(a) Steganography by alterations in data (here: embedded message = 010)



(b) Steganography by timing channel: message symbols map to different interpacket delays (here: embedded message = 00100)

Figure 2.4: Steganogram concealment methods. In general, embedding requires one to perturb a specific header field (a) or modulate different interpacket delays (b) in the network packets. It is usually a norm to directly hide information in storage-based channels, whereas information bits are symbolically hidden in timing-based channels.

Steganogram embedding mechanisms

Tweaking a message, or a plaintext, is possible to enhance a steganography technique, e.g., compressing, encrypting, or transcoding. Furthermore, a technique typically uses only a portion of the available cover space, potentially spreading steganograms across the cover.

Transmission alteration

Fraczek et al. have presented a Deep Hiding Technique (DHT) framework that can potentially be applied to all existing network steganography techniques to enhance its secrecy further [21]. DHT is divided into two sections: 'affect steganogram', and 'affect carrier'. As the name may imply, 'affect steganogram' covers different approaches to transmit steganograms from a sender to a receiver, e.g., "scattering" steganograms by using multiple protocols or recipients, and by "hopping" between different embedding techniques. Likewise, 'affect carrier' describes different methods of utilising carriers, e.g., utilising more than one layer of the TCP/IP stack, and using more than one embedding techniques.

2.3.3 Validation

Generally, steganography techniques are assessed using three attributes: bandwidth, undetectability and robustness [20].

The bandwidth attribute describes the number of steganograms that can flow from a sender to a receiver in a given timeframe, usually expressed as bits per second.

The undetectability attribute describes the capability of a steganography channel without alerting a warden. This attribute is usually subjectively biased towards the channel creator's opinion with their knowledge and belief because there usually is no objective way of measuring this. Janicki et al., however, demonstrate that objectively measuring the undetectability of VoIP-based steganography is possible [22]. Other steganography

techniques also need similar metrics to quantify the robustness objectively.

Lastly, the robustness attribute describes how much error a network steganography technique can withstand until steganograms cannot be retrieved. An acceptable range is usually around a 10% error rate. Suitable error correction techniques can be used to correct errors.

The three attributes, as a whole, assess and validate the feasibility of a proposed steganography technique. That is, the technique should withstand errors, be undetectable, and offer bandwidth that meets the requirements of its application. Another way of validating the technique is by carrying out steganalysis. Analysing the proposed technique from an attacker's perspective can help determine if the technique can bypass detection or prevention systems.

If a technique fails the validation process in the cyclic model, it needs to revisit the identification process to identify if further modifications to the processes can mitigate the problem.

2.4 Review and classification of steganography

As discussed in the previous section, steganography techniques are either storage-based or timing-based or a hybrid of the two (see Figure 2.4). Other classifications, such as one based on the TCP/IP stack (see Figure 2.5), are also possible. Table 2.1 summarises some of the notable techniques categorized using the TCP/IP stack classification.

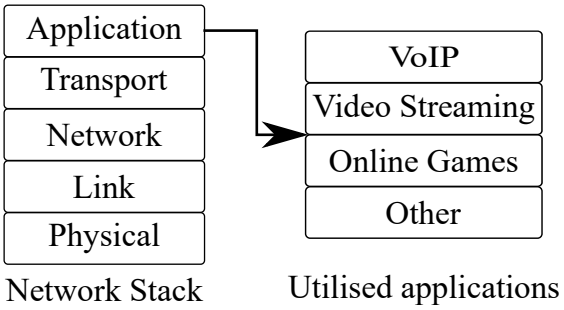


Figure 2.5: Classification of network steganography based on the TCP/IP network stack. Application layer can be further segmented into specific applications.

Table 2.1: Selected steganography techniques categorised based on the TCP/IP network stack. This table discusses each technique's embedding method, unique features, downsides and possible countermeasures.

Paper Title	Year	Embedding Method	Unique feature/s	Downside/s	Countermeasure/s
Physical/link Layer					
PHY Covert Channels: Can you see the Idles? [23]	2014	Modulating interpacket delays by manipulating the number of idle character, / I /	Utilising physical layer where current network cannot monitor its presence because of their limitation on timestamping	Would not function correctly under very congested network	Detection is possible with network equipment operating at nanosecond resolution
TCP/IP Layer					
A Novel High-Speed IP-Timing Covert Channel: Design and Evaluation [24]	2015	Timing covert channel where multiple paths convey steganograms	Overcomes limitations of timing based covert channel	No guarantee that steganograms convey the intended message	May be unnecessary until the technique becomes more stable

Network packet payload parity based steganography [25]	2013	Timing covert channel where different positions in packet payloads convey steganograms	No secret key involvement	Packet drops or out-of-order packets cause steganogram shifting, and requires exchange of agreed positions	Steganogram extraction is possible once agreed positions are compromised
Application Layer *VoIP*					
Using transcoding for hidden communication in IP telephony [26][22]	2014 2015	Storage covert channel Transcoding VoIP codec to lower quality	Transcoding to increase bandwidth with reasonable MOS-LQO degradation with acceptable increase in delay	Only specific pair of codecs can be utilised	Transcoding steganalysis [19]
An approach of covert communication based on the adaptive steganography scheme on Voice over IP [27]	2011	Adaptive embedding rate based on block smoothness	Dynamic bandwidth	Silent packets are not sent if the feature is enabled	Message extraction is possible

A packet loss concealment technique for VoIP using steganography [28]	2003 2004 2007	Two-side PWR with side information	VoIP protocol enhancement; PLC	N/A	N/A
Multimedia Streaming					
Hiding Data in Multimedia Streaming over Networks [29]	2010	Embedding steganogram in B frames of MPEG-2 streaming	Modifies B frames that have the most negligible impact on video playback	HTTP live streaming is replacing RTP based streaming	Steganogram extraction is possible
Online game					
Games Without Frontiers - Investigating Video Games as a Covert Channel [30]	2015	Common RTS elements	Development of framework which can be easily adapted to other games	Symbolic steganogram saved inside the map replay file	Government enforcement of banning replay system
Rook: Using Video Games as a Low-Bandwidth Censorship Resistant Communication Platform [31]	2015	Modulating game attributes based on statistical data gathered	Identifying immutable fields and building a symbol table	Symbol table removes outliers making range smaller	Detection is possible just by keep referring to mutable fields

Other					
Keyboards and covert channels [32]	2006	Physical device with passive time based covert channel	Exfiltrate keyboard activities without compromising host or its software	Have to place physical device	Entropy-based detection [33]

2.4.1 Physical/link layers

Typically, network steganography techniques are designed from layer three of the TCP/IP stack as it appears infeasible to implement at the lower layers. For example, the link layer header fields are responsible for the hop-to-hop delivery of frames. That is, any modification on the link layer header fields is retained for only one routing hop.

Lee et al., however, demonstrated that creating network steganography on the physical layer and preserving steganograms over a long distance is possible. Furthermore, detecting and disrupting such channel in real life poses some challenges with current network infrastructures [23]. Their prototype, called Chupja, modulates different inter-packet delays to convey binary bits. While this may sound like traditional timing-based network steganography, the technique involves modulating nanosecond delays by altering the numbers of idle characters, / I /, that only exist in the physical layer. The authors experimentally evaluated the feasibility of their channel by testing it on the National LambdaRail (NLR), a long-distance research network that was about 19000 km in length. Their result suggests that the nanosecond modulations preserved over the NLR with nine routing hops that usually had cross traffic of 10%~20% of the total channel capacity. Chupja exploits the fact that the current network timestamping mostly records in microsecond intervals, albeit hardware can handle nanosecond resolution. The constraints were in memory, storage and cost of maintaining the nanosecond resolution.

Chupja is not perfect, however, as it can only achieve an error rate of less than

10% when tuned to operate at 10% of the total channel capacity of 10 Gbps, and a large modulation amount of 2048 idle characters. Despite the 10% link utilisation, the network steganography still resulted in a data rate of 81 kbps.

2.4.2 TCP/IP layers

These layers are popular network steganography covers. Embedding steganograms in these layers are straightforward because steganograms can be effortlessly retained over multiple hops. A caveat here is that there were previous attempts to identify all possible steganography channels in these layers [34, 35]. It now requires an exceptionally profound hiding mechanisms.

Hovhannisyan et al. examined existing timing-based network steganography and attempted to address some common problems/limitations those channels share – low channel capacity, low covertness, and high error rate [24]. The authors proposed a multi-path embedding scheme to overcome these shortcomings, which essentially spreads out the steganograms. The authors proposed location or port-based methods to achieve this. The location-based system operates on having multiple receivers, whereas the port-based system utilises multiple network ports.

Abdullaziz et al. proposed two timing-based techniques that use User Datagram Protocol (UDP) payload [25]. The first technique where different lengths of payload to convey steganograms is easy to detect. Complete extraction of steganograms is possible by a warden. The second technique is more difficult to detect because a warden first needs to figure out the positions of the UDP payload bytes that the sender and receiver agreed to look up, called the agreed positions. The technique, however, has a few shortcomings: Firstly, packet drops or out-of-order packets cause steganograms to shift. Secondly, extracting steganograms is possible once the agreed positions are compromised. Thirdly, the sender and the receiver need to exchange the agreed positions before the actual communication, the extra overhead. Authors' proposal of bringing

compression or encryption to this channel as their future work may not be feasible unless the communication link is free of packet drops and out-of-order packets.

2.4.3 Application layers

These applications usually account for a considerable amount of Internet traffic around the world. Web browsing, VoIP and any entertainment-related applications such as video streaming and online games belong to this category. Koh classified past University of Auckland network traffic, identifying that the five most popular applications account for around 95% of the total network bytes [4].

VoIP

Janicki et al. demonstrate that one can measure the undetectability aspect of VoIP techniques by measuring the voice quality degradation amount [19]. One such metric is the Mean Opinion Score - Listening Quality objective (MOS-LQO) which objectively indicates the voice quality. Researchers can use this metric to quantify how much voice degradation their technique introduces and qualify on the undetectability aspect.

Stateless protocols operate on the principle of the best-effort delivery, i.e., no guarantee that network packets reach their destination. Conversely, *stateful protocols* validate data integrity, retrieving the same data again if missing or corrupted. While stateful protocols are reliable, they add overhead that can be non-trivial if quick data delivery is more important than data integrity. A stateless protocol, such as UDP, therefore, often encapsulates time-critical VoIP voice data. Packet Loss Concealment (PLC) in VoIP may minimise the possible adverse effects of the dropped packets. Unlike other researchers, Aoki explored the possibility of using network steganography for a good cause, i.e., enhance PLC by inserting the pitch variations of preceding packets as steganograms inside current packets [28]. One can debatably achieve the same outcome by allocating new network header fields to embed the same information but at the cost of transmitting

more data. The author eventually expands on their research, further improving the voice quality [36, 37].

Miao et al. identified that utilising the least significant bit (LSB) embedding scheme for VoIP degrades the speech quality, leading to increased detection rate [27]. The authors addressed this problem by factoring in the 'smoothness' of sample blocks and dynamically adjusting the embedding rate. The technique embeds more data to sharp blocks, i.e., high amplitude samples, than silent intervals with flat blocks. Their result suggests that their channel achieved a bandwidth of around 7.5 kbps, maintaining MOS-LQO degradation below 0.5.

Mazurczyk et al. explore the possibility of using *transcoding* to increase the bandwidth of VoIP steganography [26]. Transcoding is the process of converting one encoding to another, such as converting file format or character encoding. The technique chooses overt and covert codecs where covert codec disguises itself as overt codec, leaving some room for free storage of data. They selected G.711 (64 kbps) and G.726 (32 kbps) as the overt and covert codecs, respectively – 32 kbps of free message storage. Their result indicates that the MOS-LQO decreased from 4.46 to 3.834 with 0.39 ms of transcoding delay. The authors emphasised that detection of such channel is possible when the voice data are encapsulated in Real-time Transport Protocol (RTP) but poses challenges with Secure Real-time Transport Protocol (SRTP) because of encryption. Janicki et al. extend this research by examining all possible codec pair combinations, identifying G.711 and G.711.0 pair has costless conversion, i.e., no degradation of MOS-LQO [22].

Video streaming

Zhao et al. proposed using multimedia streaming over the network as a potential cover, specifically video streaming [29]. The technique stores steganograms into B frames of Group of Picture (GOP) and drop them purposefully at the receiver side to indicate the presence of steganograms. Overall, inspecting the contents of the dropped packets reveal the presence of steganography usage.

Online games

Hahn et al. proposed a steganography framework for Real-time Strategy (RTS) games. Their framework is applicable to all RTS games because the technique utilises common elements RTS games share: units, building, rally points, and replay logs [30]. Their prototype, Castle, achieved a bandwidth of around 50~200 Bps. A possible downside with their framework is that the replay log stores all the data. Getting a hold of a reply log thus would make extraction of steganograms possible.

Another game-based network steganography technique called Rook utilises First-person Shooter (FPS) games [31]. Rook is storage-based steganography, where it only modifies mutable fields to transfer steganograms. Rook first generates symbol tables, keeping a record of mutable field values. Rook goes through the pruning process during the symbol table generation to remove outliers in the lower region to make its length power of two such that it can send information symbols rather than the actual bits. The pruning process shifts data, potentially making steganalysis possible. Also, building a symbol table consists of the first one minute of the gameplay, which may not reflect the gameplay in other durations. A better approach would have been to teach Rook with other pre-recorded statistical data of other gameplays.

Other

JitterBug is a physical device that is placed in an input device to act as a keylogger to exfiltrate keystrokes out from trusted system [32]. Unlike other network steganography, Jitterbug uses passive timing-based channel from existing interactive connection, e.g. remote desktop/VoIP/SSH/Telnet.

2.5 Countermeasures and limitations

The previous section discussed some of the proposed network steganography techniques, showing the attacker's perspective. This section discusses resources available as a warden to combat such techniques.

2.5.1 Standards

United States Department of Defense (DoD) and National Institute of Standard and Technology (NIST) published a series of computer security guidelines and standards, widely known as Rainbow Series, where its nickname originated from the colourful document covers.

DoD Trusted Computer System Evaluation Criteria (TCSEC) or widely known as Orange Book, published in 1983, contains technical evaluation elements and methodologies to assess commercially available trusted computer systems [38]. The document hierarchically ranks protection requirements into four, called "division": Division A – verified protection, Division B – mandatory protection, Division C – discretionary protection and Division D – minimal protection. Each division may be further classified into "class", but they are not discussed further in this thesis.

Covert channels are ranked as division B, which implies that systems certified with B or above prevent the use of any covert channel. The document presents seven references on how covert channels were addressed in the past. However, as seen historically, steganography evolves rapidly, and the documentation does not cover covert channels in networks.

The next appearance of covert channels in Rainbow Series is from Covert Channel Analysis of Trusted Systems or widely known as Light Pink Book, published in 1993 [39]. As may be expressed in the title of the book, it is focused on a single topic of covert channel analysis. The scope of the book, however, mentions:

- "Not addressed are covert channels that only security administrators or operators can exploit by using privileged (i.e., trusted) software".
- "Although we do not explicitly address covert channels in networks ..., the issues we discuss in this guide are similar to the ones for those channels".

These points are only relevant to a certain degree now because portable devices can effortlessly join and leave a network. Portable device owners typically have full control of their devices, up to administrative privilege.

The last and most recent documentation, ISO/IEC 15408, is by Common Criteria (CC) [40], an international standard influenced by three different standards where one of them being TCSEC. CC's perspective on covert channels in networks is most realistic than previous standards since it places covert channels at Evaluation Assurance Level (EAL) 7, the highest level of assurance. Tipton et al. criticise in their book, however, that covert channels can also be considered as EAL 3 since they are "illicit information flows" [41]. Most Microsoft Operating systems (OS's) received EAL 4 rating means that they prevent any illicit information flows. Most individuals, however, have the administrator privilege to install applications, making EAL 4 certification questionable. Given the age of the book, we investigated if modern OS's received EAL 4 rating. As it turns out, EAL 4 rating applied only up to Microsoft Windows 7 OS products, and from Microsoft Windows 8 OS onward, no assurance levels are certified [42].

Overall, all standards mentioned earlier focused on 'trusted system' and did not identify or address covert channels in the network. From the government's perspective, this may be sufficient because it is only the administrator of a machine who has the privilege to utilise covert channels. However, a few questions remain: What if terrorists utilise network steganography to scheme their next attack? Are there any standards or guidelines that can detect network steganography? Furthermore, we are living in an era where a portable device can effortlessly join a network. How can a warden be certain that a portable device user has not been using network steganography to leak confidential information?

When we compare network steganography against cryptography, governments generally have more control since the presence and standards of cryptography is transparent. Thus, different countries have their own laws and regulations associated with exporting, importing, and key disclosure laws concerning cryptography [43, 44, 45]. Occasions where a government body intervenes with cryptography do occur, such as the Federal Bureau of Investigation (FBI) requesting Apple to assist to support the national security of the United States of America [46].

2.5.2 Traffic normaliser

raphy covers by reverting header fields back to their default values. Steganalysis literature often discusses traffic normalisation concepts because of their simplicity and effectiveness in preventing network steganography. *Traffic normalisation* is a technique to sanitise potential network steganography covers by reverting header fields back to their default values, overwriting steganograms. Steganalysis literature often discusses traffic normalisation concepts because of their simplicity and effectiveness in preventing network steganography.

Handley et al. scrutinised IPv4 header fields and identified a total of 73 possible traffic normalisations – 24 for Internet Protocol (IP), 2 for User Datagram Protocol (UDP), 38 for Transmission Control Protocol (TCP), and 9 for Internet Control Message Protocol (ICMP) [47]. Similarly, Lucena et al. identified possible normalisations of IPv6 header fields [35]. However, not every normalisation is a simple process of reverting to a default value, e.g., one must first validate semi-unused header fields conform to the protocol specifications. Moreover, traffic normalisation, i.e., essentially modifying header fields, require one to recompute checksums. These contribute to computation time.

Furthermore, prevention neither implies the detection of the secret channel nor the extraction of the message. While prevention mechanisms may fend off potential attacks, a warden may be unaware of the attacks or their detail. The timing channel prevention

technique, *Pump*, best exemplifies this as it outputs packets at an average 'trusted rate' to limit possible timing differences in the packets [48, 49]. Moreover, there appears to be no clear guideline on what happens after normalised traffic, leaving many unanswered questions in terms of analytics: When did the data breach occur? Where did it happen? What information was compromised? How much information was compromised? What was the carrier? Who were involved?

Last but not least, prevention only fuels the arms race because prevention raises the attacker's alertness. An attacker who realises the blocked communication will eventually design a more sophisticated technique that is potentially harder to detect. It may, therefore, be advisable to consider detection instead of or in addition to prevention. That way, analysing network flows or stego-packets can at least begin answering some of the questions posed above.

2.5.3 Steganalysis

Network steganalysis is the art of discerning the difference between stego-packets and non-stego-packets, thereby altering the presence of network steganography usage. Developing a steganalysis technique, however, poses challenges because of diversified embedding methods and covers (see Chapter 2.3). That is, there is a no-one-size-fits-all solution to detect all steganography technique.

A warden has to understand the embedding method before performing steganalysis. Therefore, a general trend in the literature is that a researcher first introduces their novel steganography technique. Possible steganalysis techniques are considered and discussed later on.

Researchers generally use *binary classification* to measure the detection accuracy (or robustness) of their steganalysis technique. The classification categorises observations into positive and negative outcomes. The positive outcome consists of correctly identified *true positive* (TP) and *true negative* (TN), whereas the negative outcome consists of

falsely identified *false positive* (FP) and *false negative* (FN). A steganalysis technique would ideally have no FP and FN, i.e., incorrectly identifying non-stego-packets as stego-packets and the other way around. A table that contains all of these measures is known as a 2×2 confusion matrix [50]. Based on these measures, one can then calculate other derivates, such as *positive predictive value* (PPV), *true positive rate* (TPR), and *true negative rate* (TNR).

PPV is defined as follows:

$$P = TP/(TP + FP),$$

where TP is the observed true positives, and FP is the observed false positives. PPV (also known as precision) measures the accuracy of a steganalysis technique in differentiating stego-packets from the normal-packets. A higher precision value indicates the better accuracy in differentiating stego-packets from normal-packets.

TPR is defined as follows:

$$TPR = TP/(TP + FN).$$

TPR (also known as recall) measures the accuracy of a steganalysis technique in identifying the stego-packets. A higher recall value indicates less chance of incorrectly categorising stego-packets as normal-packets.

TNR is defined as follows:

$$TNR = TN/(FP + TN).$$

TNR (also known as specificity) measures the accuracy of a steganalysis technique in identifying the non-stego packets. As such, a high specificity technique is less likely to miscategorise normal packets as stego packets.

2.5.4 Classification

There have been previous attempts to categorise existing steganalysis techniques [51, 52, 53].

Mazurczyk et al. highlight following eight statistical metrics that apply to steganalysis techniques: Welch's t-test, autocorrelation, chi-square test, regularity measure, Kolmogorov-Smirnov test (KS test), Kullback-Leibler (KL) divergence, entropy, and Conditional Entropy (CE) [51]. Archibald et al. further categorise a number of these metrics based on the detection methods [52]:

1. shape tests, e.g., KS test, Welch's t-test
2. entropy tests, e.g., first order entropy test, Corrected Conditional Entropy (CCE) test and KL divergence, and
3. regularity test, e.g., regularity and ϵ-Similarity.

Goher et al. categorise steganalysis techniques based on the steganogram concealment method: either storage or timing channels [53]. The authors discussed that most storage channel techniques are based on signatures, while the timing channel are based on traffic patterns. This categorisation, however, is broad.

Mazurczyk et al. propose another broad classification based on detection techniques [51]. The detection techniques are categorised as statistical or Machine Learning (ML), and the major difference between the two categories is the decision boundary that is used to differentiate 'normal' traffic from the stego-traffic. The decision boundary is automatically identified during the training phase for ML, while it is manually set in case of statistical approaches. The decision boundary is an important concept to explore because it can provide erroneous results if incorrectly set. For example, Fiore et al. experimentally showed that supervised ML will not perform well in a network that exhibits a different network flow than the training flow [54], thus highlighting the importance of the choice of decision boundaries.

Literature review

Many steganalysis attempts have been made in the past using different techniques [17, 34, 52, 55, 56, 57, 58, 59, 60, 61, 62]. Table 2.2 lists some of the selected techniques, assessing if the techniques may answer some of the research questions (Q1(a), Q1(b) and Q2).

The majority of the reviewed techniques require virgin data, i.e., imaginary data depicting 'normal' network traffic. This is especially the case for the techniques that use statistical metrics, and this is probably caused by how one designs and conducts the experiments. The following steps outline a typical execution of a statistical test-based steganalysis:

1. Statistical test on a virgin traffic flow without any steganograms
2. Statistical test on a stego-traffic flow
3. Identification of a baseline and a decision boundary based on steps 1 & 2
4. Generate a large number of stego-traffic flows and assess it using the baseline and the decision boundary identified in the previous
5. Present detection rate results (usually calculated using 2×2 confusion matrix)

A number of storage channel steganalysis work in the absence of virgin data [19, 63, 34]. Here, one usually knows the expected values – i.e., virgin values. Murdoch et al. have proved that this also applies to random number-based headers, e.g., Internet Protocol (IP) Identification (ID), Transmission Control Protocol (TCP) Initial Sequence Number (ISN), TCP timestamp [34]. The authors have demonstrated that all of these fields are pseudo-random and the values can be predicted based on the Operating System (OS) or libraries in use.

None of the reviewed techniques quantifies the size of steganograms. Quantifying the steganograms is important because it can indicate the damage specific network steganography could have caused.

Goher et al. look at number of existing covert channel detection methods and state

Table 2.2: Selected steganalysis techniques. It discusses each technique's detection domain, and whether it may answer some of our research questions.

Paper	Applies to	Q1(a)	Q1(b)	Q2
Steganalysis of transcoding steganography [19]	Storage channel based on VoIP Transcoding	MOS-LQO	√	X
Network steganalysis: Detection of steganography in IEEE 802.11 wireless networks [63]	Storage channel based on wireless 'Retry' and 'More Data' headers	—	√	X
Disrupting and Preventing Late-Packet Covert Communication Using Sequence Number Tracking [17]	Timing channel based on late arrival of RTP	MOS-LQO	√	X
Embedding covert channels into TCP/IP [34]	Storage channel specifically on TCP/IP stack	OS specific pseudorandom algorithm	√	X
A comparative analysis of detection metrics for covert timing channels [52]	Timing channel	Welch's t-test	X	X
Leveraging Statistical Feature Points for Generalized Detection of Covert Timing Channels [55]	Timing channel	autocorrelation	X	X
Real-Time Detection of Covert channels in Highly Virtualized Environments [64]	Storage (urgent field)	Chi-square test	X	X
IP covert timing channels: design and detection [57, 58]	Timing channels (including embedding variation and noisy channel)	regularity & ϵ-Similarity	√	X
Performance of selected noisy covert channels and their countermeasures in IP networks [59]	Timing channel	KS test	X	X
Design and analysis of a model-based Covert Timing Channel for Skype traffic [65]	Timing channel	KL test	X	X
Employing Entropy in the Detection and Monitoring of Network Covert Channels [61]	Storage channels (ICMPv4/v6 Code, TCP unused, UDP Length)	entropy	X	X
An Entropy-Based Approach to Detecting Covert Timing Channels [62]	Timing channels	conditional entropy	X	X

√ = answered, X = not answered

that covert storage and covert timing channels can generally be detected with misuse and network anomalies, respectively [53]. In misuse detection, one would examine if header fields align with the expected usage pattern. In network anomalies detection, one would monitor the packet arrivals, and comparison against historical network patterns are carried out. As for the application based covert channels the authors emphasised that this can be much more difficult since one needs a fair amount of knowledge of the protocol. Thus, it is usually based on behavioural analysis, and as an example, the author stated that half an hour long HTTP connection can be viewed as suspicious. However, this example is questionable since, for example, HTTP live streaming service is a popular streaming carrier.

Grabski et al. look at detection of steganography for 802.11 wireless networks that are based on 'Retry' and 'More data' fields [63]. However, since their technique is based on a naïve design, where these fields must be set and there must be a synchronisation bits to signal the start of a communication, these fields appear to be less favourable to be utilised as a covert channel.

Rezaei et al. designed an active warden for timing-based covert channels that intentionally use delayed packets that are discarded for late arrival [17]. According to the authors, their design specifically considers covert channels that utilise popular real-time protocol (RTP) which is used for video and audio transmission and they have achieved this by tracking sequence number. Their approach is to discard the packets at the intermediate device, e.g. firewall even before it reaches the destination by keeping the sequence number and see if it is within the receiver's jitter buffer window. The test has been carried out on VoIP which uses RTP. The result suggests that it can detect covert channel at a rate of 98% and disrupt the link while maintaining the quality of the MOS-LQO stable.

Murdoch et al. have scrutinised specifications of TCP/IP header fields and proved carrying out storage-based steganography is not feasible even if some of the fields,

such as IP ID, TCP ISN, and TCP timestamp appear random [34]. The authors have observed that all of the abovementioned fields are in fact pseudorandom, and values can be predicted depending on which OS is being used, i.e., different versions of Linux and OpenBSD have different algorithms to determine the pseudorandom value. The authors then designed a steganography scheme called 'Lathra' which takes into account their observation. This paper signifies that those who desire to design a covert channel need to have a profound knowledge of the protocol or medium they are going to be exploiting.

Janicki et al. specifically tackles detection of transcoding steganography [19], and results suggest that their detection rate ranged from 58 to 100% depending on codec pairs tested. The detection has been done by building models called Gaussian Mixture Model (GMM) by feeding it with mel-frequency cepstral coefficients (MFCC) parameters from different voice data from corpora. Comparison between original speech GMM and transcoded GMM is then carried out to justify if transcoding steganography can be detected.

As may be seen from each of the steganalysis techniques, there is no one-size-fits-all solution.

2.6 Summary

This chapter discussed a wide range of network steganography components by considering both the attacker's and the defender's perspectives. This chapter stressed the limitations of the countermeasures and the lack of standardisation in network steganography. This chapter reviewed the existing steganalysis techniques and their limitations on relying on virgin data, and inability to quantify steganograms.

The next chapter discusses T-codes and complexity derivates of T-codes, and their application in steganography and steganalysis.

3

T-codes and Complexity Derivates

This chapter introduces T-codes and complexity derivates of T-codes in Sections 3.1 and Section 3.2, respectively. Section 3.3 discusses applications of T-codes in steganography channel creation.

3.1 T-codes

T-codes are prefix-free variable-length codes (VLC) similar to Huffman codes, so no codeword in a T-code is the prefix of any other codeword in the code. Titchener introduced T-codes in the 1980s. The domain of knowledge about T-codes has progressively expanded over the years, moving away from compression applications into the area of information estimation via T-complexity, T-information and T-entropy, which have a number of practical applications [66, 67, 68, 69].

T-augmentation is a copy-and-append algorithm fundamental to the construction of

T-codes. T-augmentation makes k copies of an existing prefix-free code C and constructs another prefix-free code $C_{(p)}^{(k)}$ by prefixing each codeword in the i-th copy of C with i copies of the *T-prefix* $p \in C$, where $1 \leq i \leq k$. $C_{(p)}^{(k)}$ is the union of all the prefixed copies and the original code C, with all codewords of the form p^i removed. The latter is necessary to preserve the prefix-freeness of $C_{(p)}^{(k)}$. Note that, for further T-augmentation, p^{k+1} is in $C_{(p)}^{(k)}$, i.e., the prefix at each augmentation, is selected from the available codewords. k is known as the *T-expansion parameter* or *copy factor*. A code is a T-code if and only if it can be constructed from a finite alphabet A by zero or more T-augmentation steps with T-expansion parameters k_j and T-prefixes p_j, respectively, for $1 \leq j \leq n$ for some $n \in \mathbb{N}_0$. The resulting code may be denoted as $A_{(p_1,p_2,...,p_n)}^{(k_1,k_2,...,k_n)}$, and n is referred to as the code's *T-augmentation level*. This gives T-codes a recursive structure. Note that the system consisting of A, $(p_1, p_2, ..., p_n)$ and $(k_1, k_2, ..., k_n)$ is called the *T-prescription* of $A_{(p_1,p_2,...,p_n)}^{(k_1,k_2,...,k_n)}$. Also, note that A can have any cardinality $m \geq 2$, resulting in an m-ary tree.

Figure 3.1 illustrates the T-augmentation process leading to the T-code $A_{(0,1)}^{(1,2)}$ for a binary A. The binary alphabet is the base case of interest in this thesis, as steganograms are typically embedded in binary format. At T-augmentation level 1, the T-prefix and T-expansion parameter are $p_1 = 0$ and $k_1 = 1$, respectively. This results in $A_{(p_1)}^{(k_1)} = \{00, 01, 1\}$, where 00, 01, and 1 are the leaf nodes in the corresponding tree. The internal nodes of the tree are the prefixes of the leaf nodes and are therefore not valid codewords. The T-code at level 2 is augmented with $p_2 = 1$ and $k_2 = 2$, resulting in the code $A_{(0,1)}^{(1,2)} = \{00, 01, 100, 101, 1100, 1101, 111\}$.

3.1.1 Properties of T-codes

A code generated with T-augmentations cannot result in a perfect binary tree where all the leaf nodes are at the same level. This property guarantees that T-codes will always be variable-length codes (VLC) rather than fixed-length codes (FLC).

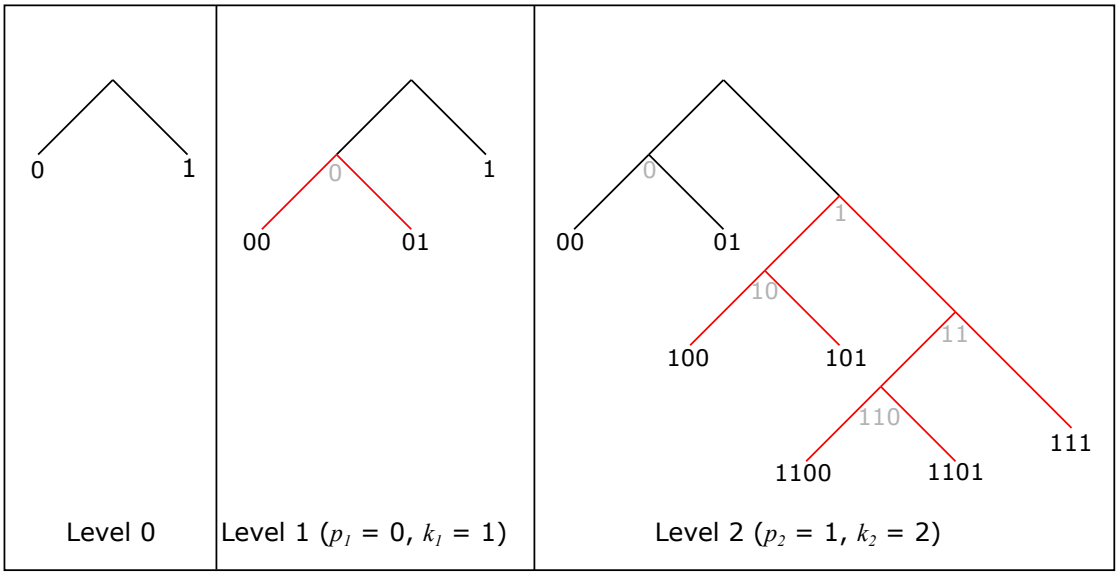


Figure 3.1: T-augmentation steps of $A^{(1,2)}_{(0,1)}$. T-prefixes and T-expansion parameters determine the way the T-code is augmented. The red lines in the figure indicate the T-augmentations. At level 1, one copy ($k_1 = 1$) of the previous level code (the alphabet $A = \{0, 1\}$) is attached to a leaf node, the T-prefix $p_1 = 0$, resulting in the T-code $A^{(1)}_{(0)}$. At level 2, two copies ($k_2 = 2$) of this level 1 T-code $A^{(1)}_{(0)}$ are attached to the original code via the chosen T-prefix of $p_2 = 1$, resulting in $A^{(1,2)}_{(0,1)}$. The shaded internal nodes represent the prefixes of the leaf nodes and do not represent valid codewords.

Bit insertion and bit deletion are conditions when a received bit stream appears to contain more or fewer bits, respectively, than the transmitter actually sent. As self-synchronising codes, T-codes are typically capable of synchronising within a few codewords after a bit insertion or bit deletion [70]. The proof that T-codes are statistically synchronisable is a simple proof by induction: Each codeword at T-augmentation level j is a combination of 1 or more codewords at level $j-1$, and any codeword boundary at level j that follows a codeword other than p_{j+1} is also a codeword boundary at level $j+1$. Consider a state machine where each T-augmentation level j corresponds to a state to which synchronisation has been established. Then, only an input of p_{j+1} causes a return to state (level) j, all other inputs cause a transition to state $j+1$. Following the principles of Markov chain state machines, a T-code decoder starting in state 0 and receiving anything but an infinite sequence of some p_{j+1} must therefore eventually progress to state n, which represents full synchronisation [71].

3.2 T-codes derivates

3.2.1 T-decomposition

T-decomposition reconstructs a T-Code set information from one of the longest codewords in the set. T-decomposition is discussed in further detail here because T-code derivates cannot be explained without it.

A T-code may have more than one T-prescription that leads to its construction, e.g., $A^{(1,2)}_{(1,11)} = A^{(5)}_{(1)}$, and it is possible to derive all possible T-prescriptions of a T-code from any of the T-prescriptions [72]. As may be seen from Figure 3.1, every T-augmentation introduces new longest codewords, i.e., codewords of lengths 2 and 4 at T-augmentation levels 1 and 2, respectively, in this case. Also, note that the new longest codewords are always the previous longest codewords prefixed with p^k for the p and k used in

the most recent T-augmentation. This implies the longest codewords in any T-code are unique to that code and only differ in the last symbol, due to the fact that the finite alphabet A is the T-code at level 0. Thus, the longest codewords are always of the form $p_n^{k_n} p_{n-1}^{k_{n-1}} \dots p_1^{k_1} a$ where $a \in A$, i.e., they contain the entire T-prescription information. The T-prefix portion of the longest codewords is known as the *T-handle*, and it is always possible to derive one of the T-prescriptions via the T-decomposition algorithm [72].

For example, the longest codewords in Figure 3.1 are $\{1100, 1101\}$, with the last symbols in the codewords being the respective symbols from the alphabet A. We can thus denote the general form of the longest codewords as $x\alpha = 110\alpha$ with $\alpha \in A$. The codeword boundaries are imposed by parsing of x into the available codewords at the current T-augmentation level and are specific to that level and marked by a subscript indicating the current level. At T-augmentation level 0, the only available codewords are the alphabet characters, i.e., $\{0, 1\}$, resulting in $x\alpha = {}_0 1_0 1_0 0_0 \alpha_0$. In this parsing, the T-decomposition algorithm looks for the *penultimate* codeword that appears, which is 0 in our case, and how often it appears (not counting the last codeword in the parsing of $x\alpha$). Here, the 0 appears only *once* in this position. Thus, we may identify $p_1 = 0$ and $k_1 = 1$. Once we T-augment level 0 with these parameters, our set of codewords becomes $A_{(0)}^{(1)} = \{00, 01, 1\}$. Using the new code, we identify the codeword boundaries again, i.e., our codeword boundaries now become $x\alpha = {}_1 1_1 1_1 0 \alpha_1$. Looking for the penultimate codeword again, we now find 1, however it appears in two consecutive instances here. This yields $p_2 = 1$ and $k_2 = 2$, resulting in $A_{(0,1)}^{(1,2)} = \{00, 01, 100, 101, 1100, 1101, 111\}$ as the level 2 T-code. The T-decomposition process ends here because the algorithm has reached the leftmost position in the string, which now parses as a single codeword from $A_{(0,1)}^{(1,2)}$ T-prescription.

3.2.2 T-complexity

Complexity determines the computational resources required in the construction of an object. Lempel-Ziv complexity measures the number of 'steps' required in parsing the string [73], for example. T-complexity aligns with this idea in that it counts the number of T-augmentation steps required to construct the T-code that contains a given string xa as the longest codeword, but it also weights each step with the binary logarithm of its T-expansion parameter [74], which makes T-complexity invariant under change to a different T-prescription for the same code. The T-complexity, $C_T(xa)$ is thus defined as:

$$C_T(xa) = \sum_{i=1}^{n} \log_2(k_i + 1), \tag{3.1}$$

where k_i are the T-expansion parameters obtained from the T-decomposition of xa.

For example, the T-complexity of $A^{(1,2,3)}_{(0,1,00)}$ is $\log_2(1+1)+\log_2(2+1)+\log_2(3+1) = \log_2(24) = 4.58$. Note that the number 24 also represent the number of internal nodes in the corresponding tree, i.e., $C_T(xa)$ represents the number of bits required to uniquely address all internal nodes.

3.2.3 T-information

Titchener's empirical study on T-complexity revealed that both Lempel-Ziv and T-complexity yield similar-looking graphs when complexity is plotted against the output of an information source [75]. The graphs also indicated that the number of 'steps' was comparatively higher for Lempel-Ziv – a possible indication to use T-codes in compression applications as the vocabulary is comparatively compact, an idea later pursued successfully by Hamano and Yamamoto [76]. Titchener's study further observed that both the average T-complexity of random strings and that of codewords constructed using the shortest available T-prefixes only closely follow the characteristic shape of the

logarithmic integral function [77], leading to the proposition that the inverse logarithmic integral of the T-complexity could be used to estimate total information content. Titchener thus defined T-information, denoted $I_T(xa)$, as:

$$I_T(xa) = \mathrm{li}^{-1}(C_T(xa)), \tag{3.2}$$

where li^{-1} is the inverse logarithmic integral.

Empirical tests on T-information suggest that it is suitable as an information content estimator as strings produced by stationary information sources exhibit a quasi-linear increase in observed T-information with length [75].

3.2.4 T-entropy

T-entropy derives itself from T-information as an information rate by considering the string length. The (instantaneous) T-entropy rate, H_T, is defined as:

$$H_T(x, L) = \Delta I_T(x, L)/\Delta L, \tag{3.3}$$

where $\Delta I_T(x)$ is the T-information added to a string x by the last ΔL symbols in the string. The average T-entropy $\bar{H}_T$ is the non-differential version of the formula above, i.e., $\bar{H}_T(x) = I_T(x)/|x|$.

Chapter 2 discussed some of the existing statistical network metrics used in steganalysis. These measures are usually reliant on the presence of virgin data. T-code derivates appeal in this regard because the T-decomposition process finds correlations in a bit stream without the need for virgin data. A previous application of T-code derivates in computer network environments by Speidel et al. used T-entropy to detect network events, specifically DDoS (Distributed Denial of Service) types of attacks [78] showed promising results. The authors conclude that T-entropy is suitable network event detec-

tion. The paper assesses Shannon's entropy as a less suitable measure to detect network events because it lacks sensitivity for correlation, i.e., it does not evaluate whether the next symbol is dependent on previous symbol(s). This is crucial because the behaviour of computer networks is not random. For instance, destination IP addresses of inbound packets at a border router are closely correlated as they are from a fixed subset.

3.3 Steganography channel creation use

T-code-encoded steganograms have the following benefits:

- Compression

  All codewords in fixed-length codes (FLC) have the same bit length, e.g., fixed 8-bit codewords in extended ASCII. In variable-length codes, such as T-codes, however, one may associate more frequent English alphabet characters with shorter codewords. While some of the codewords in T-codes may be longer than those of fixed-length codes, fewer bits are required for the whole steganogram.

- Bit slips (rate distortion)

  Bit slips result from a difference in clock frequency between transmitter and receiver and describe the situation where one or more bits are deleted or inserted during the steganogram reception. Facing such an issue in FLC leads to loss of synchronisation. T-code self-synchronisation can mitigate this problem to a certain degree.

- Bit flips

  In a FLC, a bit flip causes an erroneous codeword but does not lead to loss of synchronisation as in bit slips. In VLCs, bit flips often cause loss of synchronisation as they make the decoder go down a different (and possibly longer or shorter) branch of the decoding tree than the intended one. T-codes suffer from this as well, but self-synchronisation helps them recover.

While bit slips and bit flips are uncommon in storage-based steganography, we observe these effects in timing-based steganography (see Chapter 7). Compression of steganograms is a desirable feature for both storage and timing-based steganography.

3.3.1 Code design

A T-prescription determines a tree structure of a T-code set. The choice of T-prescription allows a T-code to be adapted to a source for compression purposes. As discussed in Section 3.1, one can choose any positive k_i and any of the codewords available at each T-augmentation level as p_i, providing considerable flexibility in constructing a T-code set.

Our T-code must meet two criteria: sufficient space and compression. Having sufficient space allows one to express all the required symbols, making communication possible. On the other hand, compression is a desirable feature that steganographers would consider utilising. Steganography can typically only use a small fraction of the medium in which it is embedded, and compressed steganographic messages make better use of the medium than uncompressed ones. Based on the two criteria, it appears that a Huffman code can be used instead of a T-code. We reiterate that the T-codes are self-synchronising codes that, by nature, excel in a lossy timing-based steganography. Arguably, some Huffman codes are self-synchronising codes. The construction of Huffman codes, however, is based on the source probabilities that does not guarantee the code will have the self-synchronisation property.

Cardinality of a T-code

The cardinality of a code describes the number of codewords in the code. The cardinality of a T-code is thus the number of codewords generated from its T-prescription, and is given by:

$$\#A^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)} = 1 + (\#A - 1)\prod_{i=1}^{n}(k_i + 1), \tag{3.4}$$

where $\#A$ is the number of alphabet symbols, and k_i is the T-expansion parameter at level i. Since our steganograms are in binary format, the formula can be simplified to:

$$\#\{0,1\}^{(k_1,k_2,\ldots,k_n)}_{(p_1,p_2,\ldots,p_n)} = 1 + \prod_{i=1}^{n}(k_i + 1). \tag{3.5}$$

As the formula indicates, only the T-expansion parameters k_i determine the number of symbols in a T-code set, which gives a simple criterion to assess whether a T-code contains sufficient codewords.

For example, if one requires a T-code set with $33 = 32 + 1$ symbols, then all the possible factorisations of 32 are: (32), (2, 16), (2, 2, 8), (2, 2, 2, 4), and (2, 2, 2, 2, 2). As each factor corresponds to a $k_i + 1$ for some i, the corresponding combinations of k_i are (31), (1, 15), (1, 1, 7), (1, 1, 1, 3), (1, 1, 1, 1, 1) and permutations thereof. The cardinality formula thus solves the problem of sufficient space exactly, but neither do any of these combinations guarantee optimal compression, nor can we determine compression efficiency unless we know which T-prefix one should choose at each augmentation level.

Compression performance

Optimal compression for a given source requires a search that may result in a code with more codewords than needed. A VLC offers better compression than another code when difference between the sum of the probability-weighted lengths of its codewords and the Shannon entropy of the source is smaller than for the other code. One can achieve this by adjusting the tree's growth in width and depth through an appropriate choice of p_i and k_i. Figure 3.2 illustrates this with a requirement of seven symbols with the following symbol probabilities: 0.25, 0.125, 0.125, 0.125, 0.125, 0.125 and 0.125. This source has a Shannon entropy of $0.25{*}2{+}0.125{*}3{+}0.125{*}3{+}0.125{*}3{+}0.125{*}3{+}0.125{*}3{+}0.125{*}3 = 2.75$. The left and right T-prescriptions result in seven and nine codewords, respectively.

Two codewords are unused in the right T-code, but it offers better compressibility. In terms of the average weighted codeword length, the left and right T-codes yield $0.25 * 1 + 0.125 * 2 + 0.125 * 3 + 0.125 * 4 + 0.125 * 5 + 0.125 * 6 + 0.125 * 6 = 3.5$ and $0.25 * 2 + 0.125 * 2 + 0.125 * 3 + 0.125 * 3 + 0.125 * 4 + 0.125 * 4 + 0.125 * 4 = 3$ bits, respectively. The right T-code is thus better suited for the source since its redundancy is lower than that of the left, i.e., $3.5 - 2.75 = 0.75 < 3 - 2.75 = 0.25$.

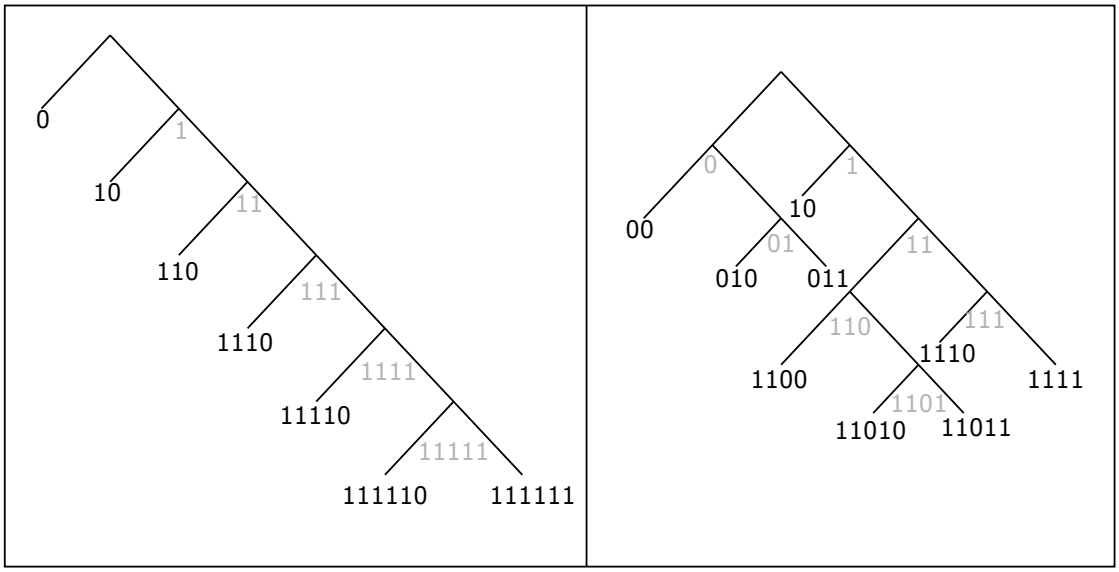


Figure 3.2: Comparing compressibility of $A^{(1,2)}_{(1,11)}$ (left) and $A^{(1,1,1)}_{(1,0,11)}$ (right). The source has a Shannon entropy of 2.75 assuming seven source symbol probabilities of 0.25, 0.125, 0.125, 0.125, 0.125, 0.125 and 0.125. The average weighted codeword lengths of the left and right T-codes are $0.25*1+0.125*2+0.125*3+0.125*4+0.125*5+0.125*6+0.125*6 = 3.5$ and $0.25 * 2 + 0.125 * 2 + 0.125 * 3 + 0.125 * 3 + 0.125 * 4 + 0.125 * 4 + 0.125 * 4 = 3$, respectively. The right T-code comes closer to the source entropy, offering better compressibility despite having two unused codewords.

Practical code design

Coding efficiency is not the main concern of this thesis, however. So, for simplicity, we restrict ourselves to $k_i = 1$, minimal-length p_i and thus *minimal T-augmentation*. Our T-code needs to support 65 symbols, i.e., the 26 English alphabet characters in lowercase,

uppercase and 13 punctuation characters. Using the cardinality formula (Equation 3.5), our choice of T-expansion parameters is as follows: $k_1 = 1$, $k_2 = 1$, $k_3 = 1$, $k_4 = 1$, $k_5 = 1$, $k_6 = 1$. Always choosing one of the shortest possible p_i to promote the tree's growth in width then leads to a code such as the one given by the T-prescription $A^{(1,1,1,1,1,1)}_{(1,0,11,00,10,011)}$. Figure 3.3 illustrates our encoder in the binary tree structure.

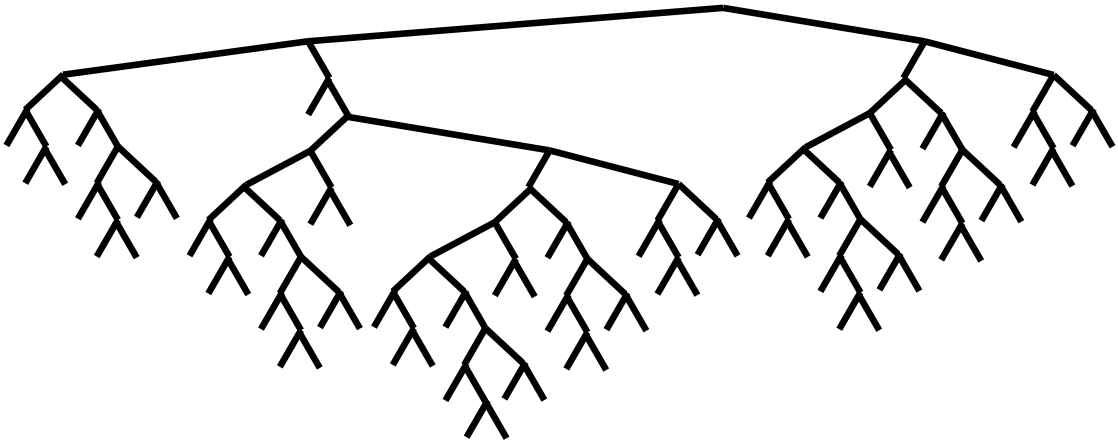


Figure 3.3: T-code encoder $A^{(1,1,1,1,1,1)}_{(1,0,11,00,10,011)}$. Constructed with the smallest possible k_i and p_i values to promote the tree's growth in width. This T-code set accommodates 65 symbols; 26 English alphabet characters in lowercase and uppercase, and 13 punctuation characters.

Enhancing T-codes

Cyclic equivalence (CE) consists of rotating a string of codeword N over alphabets A. A codeword is a periodic cyclic equivalence (PCE) class of a code when the codeword has N unique rotated strings. Otherwise, it is a non-periodic cyclic equivalence (NPCE). Removing PCE codewords from a T-code set, i.e., minimising consecutive T-prefixes improve the synchronisation delay a T-code decoder spends to recover from the unsynchronised state. Removing PCE codewords turns a T-code into a bounded synchronisation delay (BSD) code [79, 80]. Two finite strings are cyclically equivalent (CE) if they can be transformed into each other by removing a prefix from one of the

strings and attaching it as a suffix to the remainder (cyclic shift). Each finite string and its cyclic equivalents form a cyclic equivalence class. Cyclic equivalence classes of strings of length N with N distinct elements are called non-periodic cyclic equivalence (NPCE) classes, those with $< N$ elements are known as periodic cyclic equivalence classes (NPCE) their codewords are said to be *periodic*. In T-codes, periodic codewords consist of two or more copies of the same T-prefix. Removing periodic codewords from a T-code limits the number of consecutive T-prefixes a T-code decoder may encounter, therefore bounding and improving the code's synchronisation delay. Removing PCE codewords thus turns a T-code into a bounded synchronisation delay (BSD) code [79, 80].

Examples of PCE and NPCE, where $N = 3$:

- 000 - PCE, results in 000 under all cyclic shifts
- 001 - NPCE, {001, 010, 100}
- 011 - NPCE, {011, 110, 101}
- 111 - PCE, results in 111 under all cyclic shifts

Removing the four PCE codewords {0000, 1010, 1111, 011011} turns our initial T-code set into a BSD code at the expense of losing the ability to encode four punctuation characters. Figure 3.4 illustrates our final T-code tree.

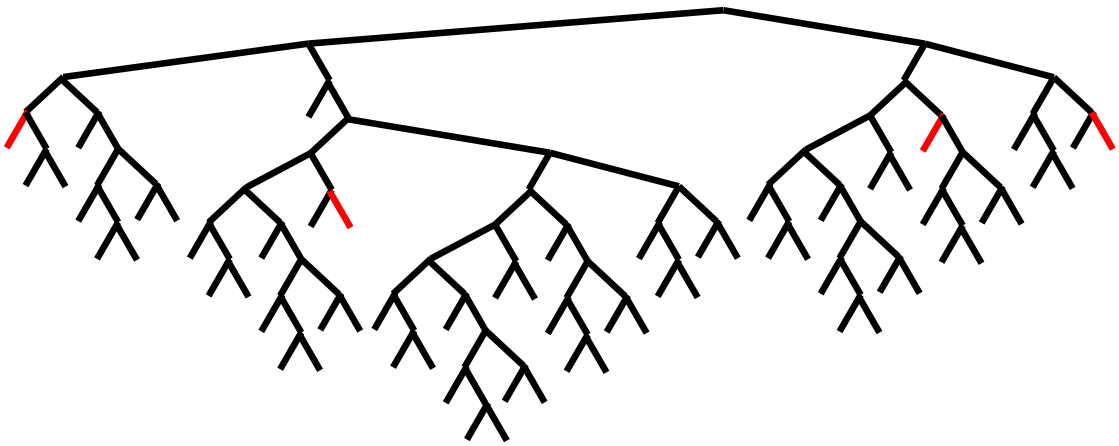


Figure 3.4: T-code $A^{(1,1,1,1,1,1)}_{(1,0,11,00,10,011)}$ turned into a bounded synchronisation delay code by removing the four periodic codewords (indicated with red branches).

Decoder model

A T-code decoder can work in a linear fashion, starting from the root of the coding tree, progressively increasing the number of bits to scan to find a matching codeword (leaf node). If no match is found even after it reaches the longest possible codeword, a bit deletion/insertion or a bit flip has occurred. The scan now returns to the root, and the same procedure repeats until the scan reaches the end of the input string.

This chapter briefly introduced T-codes and their derivates and finer details of our T-code set encoder and decoder. In the next chapter, we delve deeper into our storage-based steganography and steganalysis with T-entropy.

4

Network Storage-based Steganography

In this chapter, we describe our network storage-based steganography model and analyse it using a re-embedding steganalysis technique. Section 4.1 introduces our network storage-based steganography and re-embedding steganalysis. Section 4.2 discusses the background and a number of related works. Section 4.3 discusses our steganography design in detail and justifies our design decisions. Section 4.4 discusses our steganalysis in detail. Sections 4.5 – 4.8 discusses the results of re-embedding steganalysis. Section 4.9 discusses the implication of the results. Section 4.10 summarises the chapter.

4.1 Introduction

As discussed in Chapter 2, one can use different header fields in the network stack to embed secret data. This chapter focuses on TCP/IP header fields, i.e., Network Layer of the TCP/IP stack. TCP/IP network header fields are popular as a cover in network

storage-based steganography because steganograms can effortlessly be retained over multiple hops. As a rule of thumb, unused network headers are avoided as a cover. For instance, three reserved bits in the TCP layer need to remain untouched as zero. The reserved bits are intentional placeholders for possible future use. These bits are released from the reserved state only when it is needed, e.g., network specification change [81]. Thus, manipulating any of the reserved bits not only raises an immediate alarm to a warden but also enables the recovery of the most, if not all, of the secret data. In this regard, used header fields may appear as appealing steganography covers, but they pose compliance challenges. Such that, a manipulation introduced by a technique has to comply with the underlying network protocols to meet expected behaviours. Murdoch et al. empirically demonstrate an eventual discovery of a steganography channel with the compliance failure [34].

Semi-unused header fields are only utilised when certain conditions are met. The semi-unused header fields may thus be considered as compelling covers since the detection has to consider both scenarios of them being used and unused. Such characteristic may provide flexibility in steganography because both of the scenarios are viable options. In this chapter, we investigate the viability of semi-unused header fields, specifically IP flags field as a steganography cover and possible detections associated with it. We experimentally demonstrate that the creation and detection of IP flags steganography are possible. Re-embedding steganalysis on network storage-based steganography show promising results. The technique is not only able to detect the presence, but also quantify the size of the secret data. Only the techniques that use IP flags are experimented here but this does not imply the re-embedding steganalysis cannot be used on other header fields.

4.2 Background and related work

Fragmentation is a process of splitting a large chunk of network data into smaller chunks, called *fragments* so they can traverse a network and reach the final destination [15]. IP fragmentation is necessary because a *maximum transmission unit* (MTU), which defines the maximum packet size a receiving host accepts, can vary between the receiving hosts. Thus, when a packet size exceeds the MTU allowed on any of the receiving hosts that exist en route to the final destination, the packet has to be fragmented. The fragments are reassembled back to a packet once all the relevant fragments arrive at the receiving host. *Internet Protocol (IP) flags, IP identification and fragment offset* header fields aid the reassembly of the fragments.

Kent et al. discuss the disadvantages of the fragmentation [82]. They emphasise fragmentation process contributes to network overheads. A fragment, just like any other network packet, has to traverse a network to reach its destination. To enable such functionality, every fragment is encapsulated with the network headers – thereby increases *transmission delay*. Furthermore, these headers are required to be processed by the intermediary devices en route to the destination – thereby increases *processing delay*. In the worst case where a fragment does not reach a receiving host, then the whole packet is required to be re-fragmented and retransmitted. Fragmentation was initially designed to help heterogeneous networks to interoperate. However, as discussed previously, the consensus view on the fragmentation is not favourable. Thus, *Path MTU Discovery* (PMTUD) has been introduced as a workaround to prevent fragmentation [83]. PMTUD discovers a minimum MTU size from receiving hosts on a path by sending datagrams and listening for specific ICMP messages.

A routing path, however, can change during the packet traversal. Also, PMTUD can be blocked by a network administrator because the administrator may have a misconception that all ICMP packet are “harmful”. Luckie et al. empirically test 50,000

popular websites to measure PMTUD failure rates, and their experiments revealed that around 67% of the failed webservers also dropped ping requests [84]. A more recent paper from Göhring et al. revealed that the failure rate of PMTUD has declined from 2008 to 2016 [85]. The authors, however, ask a fundamental question of whether the benefit offered by PMTUD outweighs the possibility of exposing oneself to *degradation of service attacks*.

Although the adoption rate of PMTUD is increasing, fragmentation is not an obsolete phenomenon that is no longer observable. Fragmentation may be observed in *Session Initiation Protocol* (SIP), *Domain Name System Security Extensions* (DNSSEC), and *tunnelling protocols* [86, 87, 88, 89].

A number of papers looked at using fragmentation property to embed secret data [90, 91]. This can involve using ID field, offset field, reordering, and etc. Mazurczyk et al. propose a number of steganography models that are based on fragmentation [90]. One of their proposed models consists of conveying secret data by the number of the fragmented packets: the even and the odd number of fragments convey 0 and 1, respectively.

Three bits of IP flags field are *"evil"* bit, *don't fragment* (DF), and *more fragment* (MF) in the order of most significant bit (MSB) to lease significant bit (LSB). RFC 791 mandates the proper use of these flags [15]. In summary, three protocol-compliant bit combinations are as follows:

- 000 – the packet is not a fragment, and it can be fragmented.
- 010 – either the packet is not a fragment and it cannot be fragmented or the packet is the last fragment. Non-zero fragment offset field indicates the usage of the latter.
- 001 – the current and the following packets are fragments.

It should be noted that 010 is also used in PMTUD to signal PMTUD support.

Ahsan et al. propose to manipulate DF bit [92]. Their model generates 000 and 010 IP flags values to convey 0 and 1, respectively. Conflicting information can, however, be observed when their usages are reviewed. Bit combination 000 indicates potential

for fragmentation, whereas 010 indicates the opposite. While 000 and 010 are protocol-compliant IP flags values, they may not coexist. To validate our hypothesis, a residential network has been monitored with a network monitoring software, Wireshark [93]. The results reveal that 000 and 010 flags values are present in the monitored capture file. Furthermore, both the values can appear in a single network flow but from each side only. For instance, if a host decides 000 flags value for a flow, then the decision appears to remain throughout the flow. Thus, alternating between the two flags values may raise an immediate alarm to a warden. As discussed in section 2.5.2, traffic normalisation may prevent this technique. It should, however, be emphasised again that prevention neither implies the detection of the secret channel nor the extraction of the message. Thus, steganalysis is as important as prevention. Furthermore, the IP flags field is a semi-unused header field, so one has to consider both cases of it being used and unused. The next section describes our IP flags steganography models and the steganalysis associated with it.

4.3 Steganography models

4.3.1 Embedding modes

We use the three IP flags bits as follows:

- Compliant mode – utilising 000 or 010, and 001.
- Non-compliant mode – utilising all the possible three bits combinations, including the protocol-non-compliant bit combinations.

The compliant mode only use the valid IP flags values identified in the previous section. The non-compliant mode shows an example of utilising maximum allowed channel capacity. That is, it uses three bits of hiding space ranging from 000 to 111, albeit most bit combinations are non-compliant. This mode can be detected or prevented effortlessly. The purpose of this mode is to observe the impact it has on steganalysis.

4.3.2 Real vs. synthetic data

Network steganography experiments can generally be executed with the following two network data options: real and synthetic data. An experiment using real data is typically carried out as follows: 1) Generate a network flow of interest 2) Intercept and manipulate the flow according to a steganography model 3) The manipulated flow is released to a network link 4) Monitor and analyse incoming network flows at the destination. Using real data has an advantage of showing a network steganography's potential in real life scenario. It can indicate the feasibility of a developed network steganography because it managed to bypass wardens and reached the destination. A low-level permissions on OS is usually required to intercept and manipulate packets in real time, so the procedure can be problematic [26]. Also, because one has to intercept and manipulate a flow, it adds computation time; increased latency at the destination. Mazurczyk et al. use real data in their experiment, and measured average latency of their network storage-based steganography, TranSteg [26]. Their result suggests average latency changed from 0.85 ms to 1.24 ms – 0.39 ms increase. This is around 46% increase which may be regarded as a significant change in network steganalysis domain.

Furthermore, acquiring real data may not be ideal in many cases. Following are some of the reasons one may consider synthetic data [9]:

1. data associated with harsh environments
2. data from remote locations that are hard to access
3. data relating to rare or infrequent events
4. data that might be private, confidential, or sensitive
5. data that is impractical to obtain in large volumes
6. data that is impractical to gather over a long time period

Synthetic data is much more flexible because one only synthesises the features of interest. A major issue, however, with the data synthesis is that one cannot synthesise

something that one has not experienced. That is, one does not know if a synthesised packet conforms to the underlying network protocol, implying: Can the synthesised packets travel over a network like real data?

Our experiments use synthetic data to save the overhead in sending steganograms across a network. In timing-based steganography, timing manipulations have to withstand the possible network jitter in real life. In storage-based steganography, however, storage manipulations have to remain unchanged at the final destination.

To this end, we executed a validation experiment by manipulating IP flags. The eight packets from the non-compliant mode, which encompasses all the possible bit combinations, were sent from New Zealand to an Ohio Amazon Web Services (AWS) server. The packets were synthetically crafted and manipulated using WireEdit [94]. The crafted packets were then stored in a Packet Capture (PCAP) file. PCAP files are used in a number of network analysis applications [93, 95, 96, 97]. PCAP files have a simple structure as illustrated in Figure 4.1. A trace file starts with a global header containing common information such as a magic number, version, and capture time zone. The file then contains a series of packet dumps, each preceded by a packet record header describing each of the packets. The PCAP file containing the crafted packets are sent to the Ohio AWS server using Packet Player [98]. The packet monitoring software, Wireshark, at the server-side revealed that the packet manipulations remained unchanged.

Our experiment has been simulated under the assumption that the unchanging property holds for any number of subsequent packets. WireEdit is a useful tool in manipulating header fields with ease, thanks to its graphical interface. It may, however, not be suitable when a large number of packet manipulations are required. An XML-based specification can come in handy but can be tedious when every detail needs to be specified from the ground up [9]. To simplify the experiment, a pre-captured PCAP file has been manipulated with the help of a wrapper library. The PCAP file wrapper enables

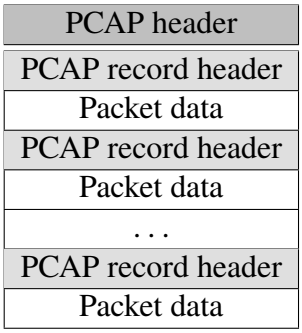


Figure 4.1: The format of a PCAP file. PCAP header contains a magic number and other necessary information to read the PCAP file. PCAP record header contains the timestamps and length of data bytes. Packet data contains the captured data.

one to parse a PCAP file and manipulate the contents inside, such as the header fields.

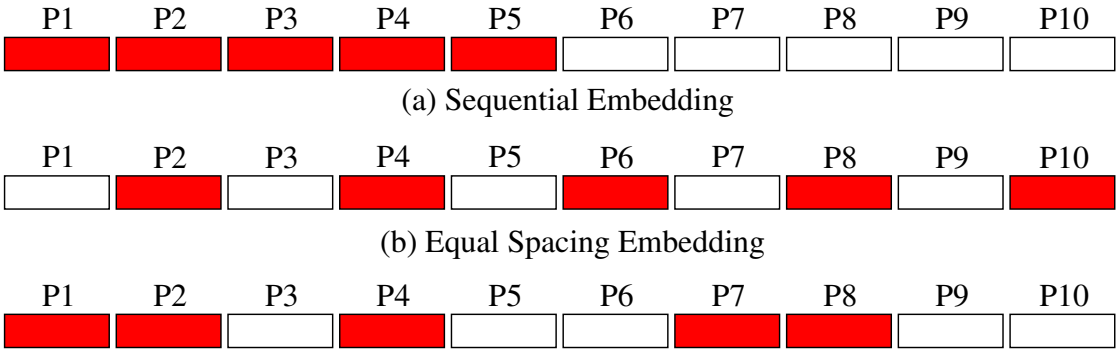
The pre-captured PCAP file has the following properties:

- Every packet's IP flags field is set as 010 by default, i.e., Don't fragment (DF) set.
- 2100 packets, in which, two IP addresses communicate in an alternating order.

If we assume a unidirectional communication, 1050 packets would be manipulated at 100% embedding level. Under compliant and non-compliant modes, one can hide secret data up to 1050 bits and 3150 bits, respectively. It should, however, be noted that one may not fully utilise all the capacity. This may be due to secret data being small or purposefully underutilising the full capacity. A purposeful underutilisation may be viewed as a security feature from an attacker's perspective because it may make the detection more difficult.

4.3.3 Embedding schemes

There are different ways how one can embed secret data. Three embedding schemes are considered and they are *sequential, equal spacing* and *random embeddings*. A sequential embedding approach manipulates consecutive cover packets. An equal spacing embedding spreads out a message by using every n^{th} packet. In other words, equal spacing embedding uses a stride of n while sequential embedding uses a stride of 1. Random



(a) Sequential Embedding

(b) Equal Spacing Embedding

(c) Random Embedding

Figure 4.2: Embedding 50% of a flow (containing 10 packets) using three different embedding schemes: a) Sequential, b) Equal spacing and c) Random. Packets used for embedding are marked in red.

embedding uses a pseudorandom number generator to find packets to hide messages randomly. Since the packets are streamed in order, typically the message is shuffled randomly and the message bits are embedded in the packets using a random stride. The receiver will need to know the pseudorandom number generator in order to decode the message. Figure 4.2 illustrates each of the embedding schemes at a 50% embedding level.

4.4 Steganalysis

Re-embedding steganalysis is a novel technique introduced in the context of media steganography [69]. The technique has proven its merit in media steganalysis, but its use in analysing network steganography had not been previously studied. This thesis therefore experiments with re-embedding steganalysis due to its ability to not only detect an attack, but also hint at the amount of secret data.

Re-embedding steganalysis is based on two observations. The first of these observations is: The T-entropy of a digital medium with hidden information is usually larger than that of the digital medium without any hidden information. The problem, however, is that one does not get to observe the same network trace with and without hidden

information.

Re-embedding steganalysis overcomes this problem with another observation: If there is information hidden in a digital medium and we hide further information in the same file, the original hidden data may be overwritten by the new. This means that hiding a similar amount of information in a file that already contained hidden information should not change the file's T-entropy by much.

Based on these observations, we can detect the presence and the amount of hidden information in a digital file. We hypothesise that network trace files will exhibit the same behaviour. The search for the presence of hidden information works as follows:

1. Hide δ random content in the trace, where δ ranges from 0 to 100% (of the hiding capacity).

2. For each of the cases above, calculate the T-entropy.

3. The δ value at which the T-entropy starts to increase is the estimate of the amount of hidden information (as a percentage of the hiding capacity).

Figure 4.3 shows this process for a sample trace file where hidden information occupies the first 20% of hiding capacity at the start of the file.

Figure 4.3 shows that the T-entropy does not increase as information is embedded in the trace – until the hidden information exceeds 20%. Beyond these 20%, the graph shows an almost linear increase in T-entropy. The turning point in the graph (at the 20% mark in this particular case) represents the amount of hidden information (as a percentage of the hiding capacity).

To find the effectiveness of re-embedding steganalysis, we conduct a series of experiments.

Firstly, we synthesise a number of network trace files with varying amount of hidden information embedded using different hiding techniques. The information is hidden by

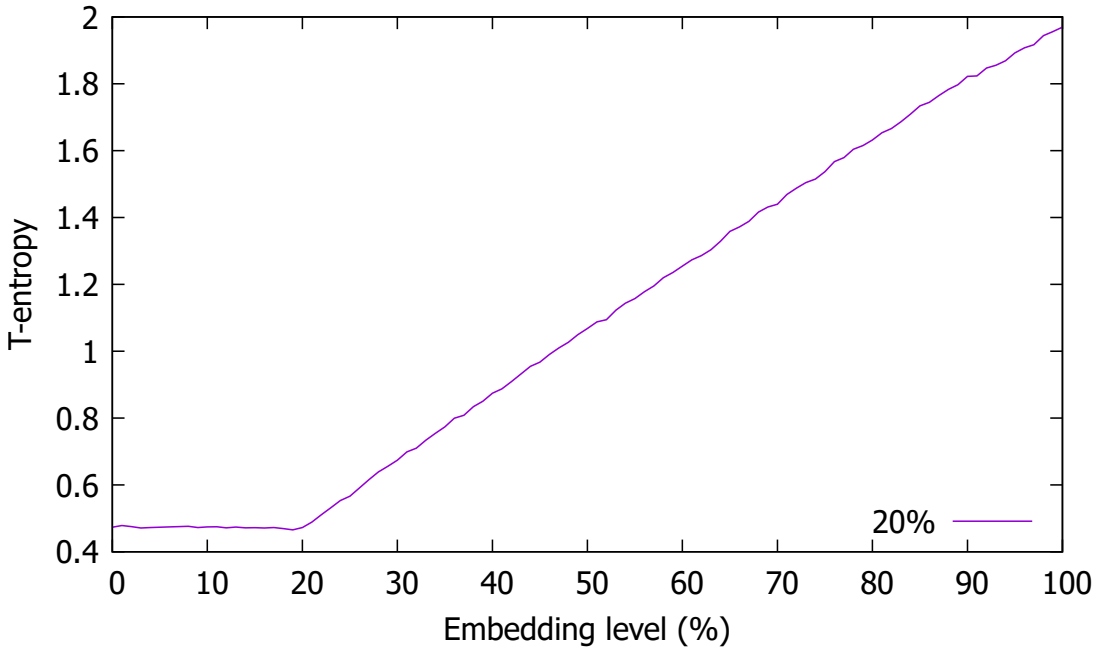


Figure 4.3: T-entropy vs. embedding level (i.e., new content embedded). Once the embedding level exceeds the amount of hidden information in the trace (20% in this case), the T-entropy value rises.

manipulating the IP flags field in the network packets. We call this first step source-embedding, since this is the embedding we strive to uncover.

For each of the generated traces, we follow the re-embedding steganalysis process as described earlier. This re-embedding process will vary the amount of hidden information step-by-step in the hope of finding the turning point in the T-entropy as shown in our sample plot of Figure 4.3. We would not know the hiding technique – sequential, equal spacing, or random – the source-embedding used, so we will have to try all possible hiding techniques in our re-embedding process.

The auto-generated “Lorem Ipsum” texts are used as the secret data to simplify the experiment and introduce randomness to the secret data. This randomness ensures that every secret data – for source as well as re-emdedding – is different.

Algorithm 1 shows the pseudocode of the experimental process. i and j parameters

define the percentages of the source and the trial embeddings, respectively. Algorithm 1 has been repeated with the different embedding schemes as defined in the previous subsection, 4.3.3. A tuple structure, (S, T) has been used to describe the possible combinations of the source and trial embeddings – S is the source embedding and T is the re-embedding. Both S and T can be any of these embedding schemes described earlier: sequential, random or equal spacing. Thus, all possible combinations are:

- (sequential, sequential)
- (sequential, equal spacing)
- (sequential, random)
- (equal spacing, sequential)
- (equal spacing, equal spacing)
- (equal spacing, random)
- (random, sequential)
- (random, equal spacing)
- (random, random)

4.5 Effects of different embedding schemes on T-entropy

We first experiment with the effects different embedding schemes have on T-entropy before assessing the capability of re-embedding steganalysis. To this end, following modifications are done to the Algorithm 1: *Trial embedding loop* is suppressed, *Source embedding loop* calculates the T-entropy, and i increments by 1 instead of 20. It is evident from the Figure 4.4, embedding scheme plays a large role because noticeable differences can be observed between the graphs. The sequential embedding scheme produced a straight-line graph where the T-entropy value and embedding level shows more or less a linear relationship. This indicates the T-entropy may be a reliable metric to detect a steganography that uses a sequential embedding scheme as the T-entropy responds relatively accurately to the amount of the secret data. The random embedding scheme resulted in a curved T-entropy graph. Moreover, this scheme usually produced

the highest T-entropy value out of all the schemes at different embedding levels. The uncertainty associated with the random embedding scheme would have attributed to such phenomenon. The equal spacing embedding scheme exhibits an overall increasing T-entropy with significant local fluctuations of unclear origin. Lastly, all the embedding schemes converged nearly at the same T-entropy value at 100% embedding level. This is an expected outcome as the same amount of data is embedded for each scheme.

Algorithm 1: Experimental methodology; shaded area indicates re-embedding steganalysis

Input : PCAP packets, Lorem Ipsum messages M, M_2

Output T-entropy values

:

for $i = 0$ *to* 100 **do** *Source embedding loop*

  Embed i% packets with the required initial portion of M ;

  for $j = 0$ *to* 100 **do** *Re-embedding loop*

    Re-embed j% packets with the required initial portion of M_2 ;

    Calculate T-entropy ;

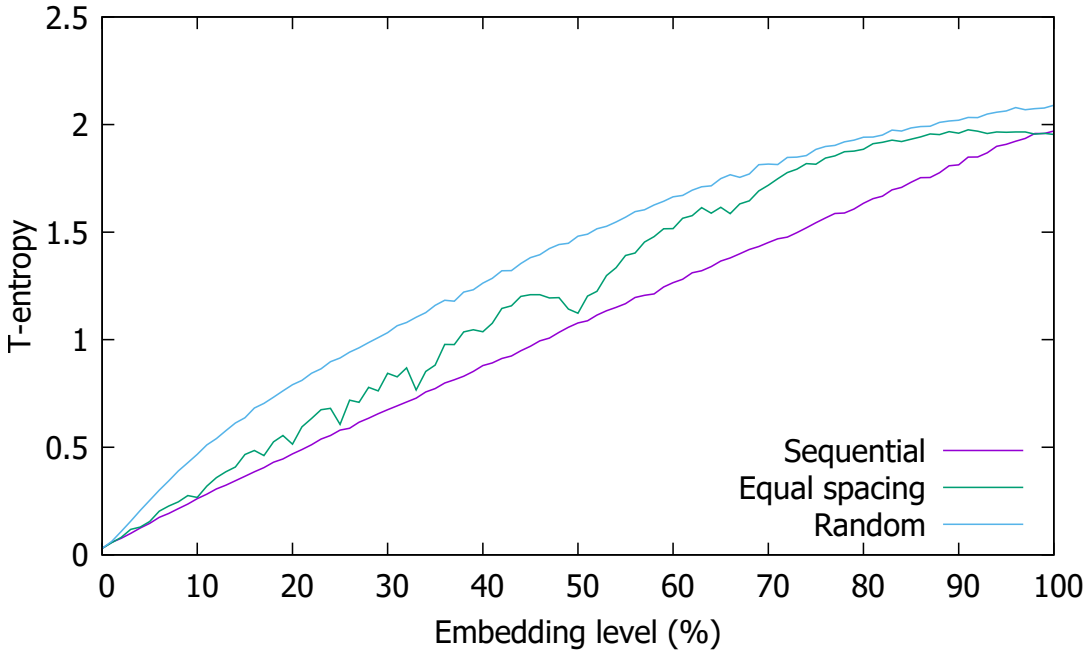
  end

  Clear embedded packets ;

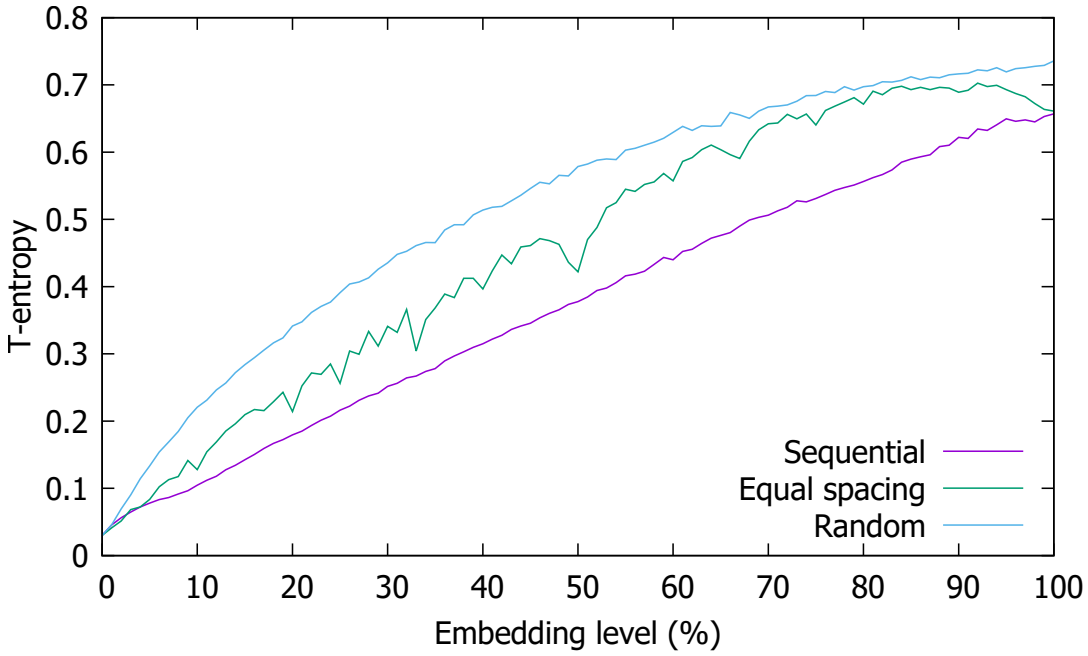
  Increment i by 20 ;

end

Two rather similar-looking graphs can be seen between the two embedding modes, but with the following two differences: First, the T-entropy values of compliant mode graphs are approximately a third of that of non-compliant mode. Second, the compliant mode graphs generally fluctuate more than the non-compliant mode equivalents. A correlation property of T-decomposition holds the answer to these observations. Since all three flags are used in T-entropy calculation, one can therefore expect to see strong correlation in the non-compliant mode. The compliant mode only exhibits a weak correlation because only one out of the three flags are used as a cover. The compliant



(a) Non-compliant mode where all of the bits of the Flags field are modified.



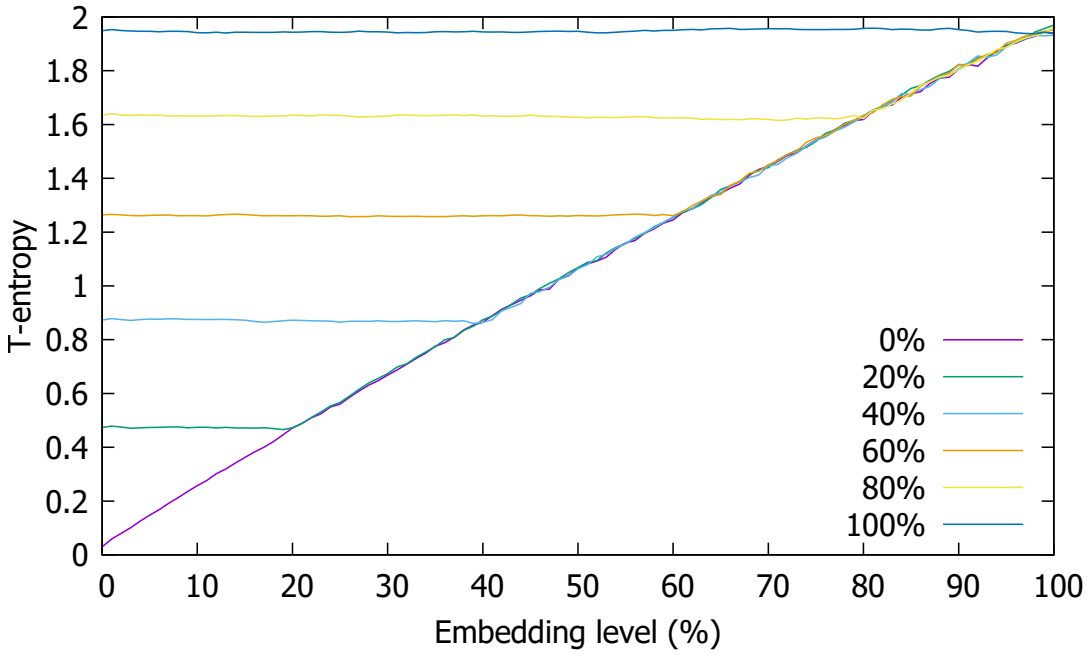
(b) Compliant mode where only the DF bit of the Flags field is modified.

Figure 4.4: The relationship between T-entropy and various embedding schemes. Both (a) non-compliant mode and (b) compliant mode generated similar-looking graphs, except T-entropy values on non-compliant mode were always higher than the compliant mode. The graphs indicate the embedding scheme has a significant impact on the T-entropy.

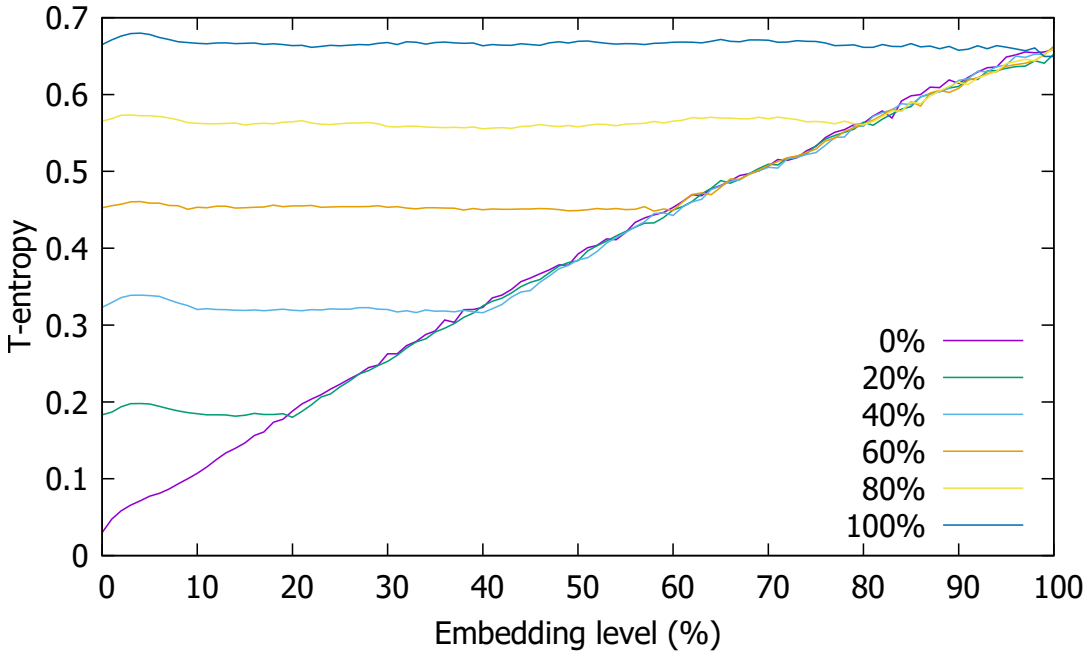
mode graphs, however, illustrate its accuracy is not significantly affected by the two irrelevant bits (*noise*) because the compliant mode graphs portray a story similar to the non-compliant mode graphs. The following steganalysis experiments will provide us an insight if the same can be said with the re-embedding steganalysis use case.

4.6 Steganalysis (sequential, x)

Figure 4.5 shows the result of the (sequential, sequential) tuple, and it shows a clear pattern: The T-entropy values remain almost constant until the trial embedding level exceeds the source embedding level. Once it exceeds the source embedding level, the T-entropy value starts to rise. This aligns with the findings in [69]. Figures 4.6 and 4.7 illustrate the results of (sequential, equal spacing) and (sequential, random) tuples, respectively. The figures, unfortunately, do not expose anything except the slopes of the lines due to the different initial embedding levels.

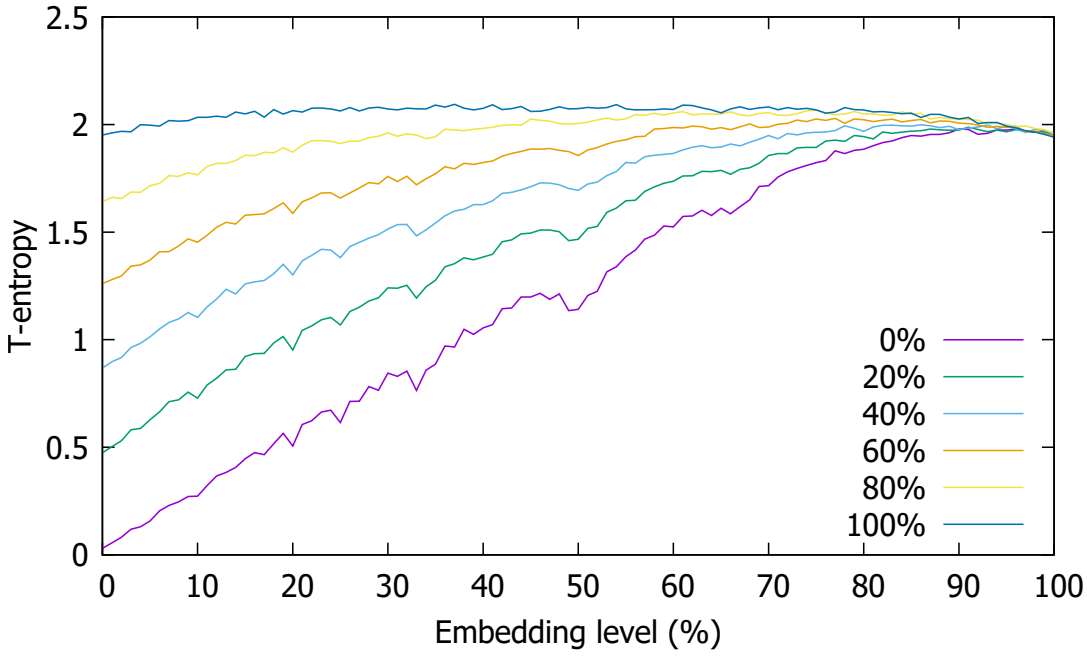


(a) Non-compliant mode where all of the bits of the Flags field are modified.

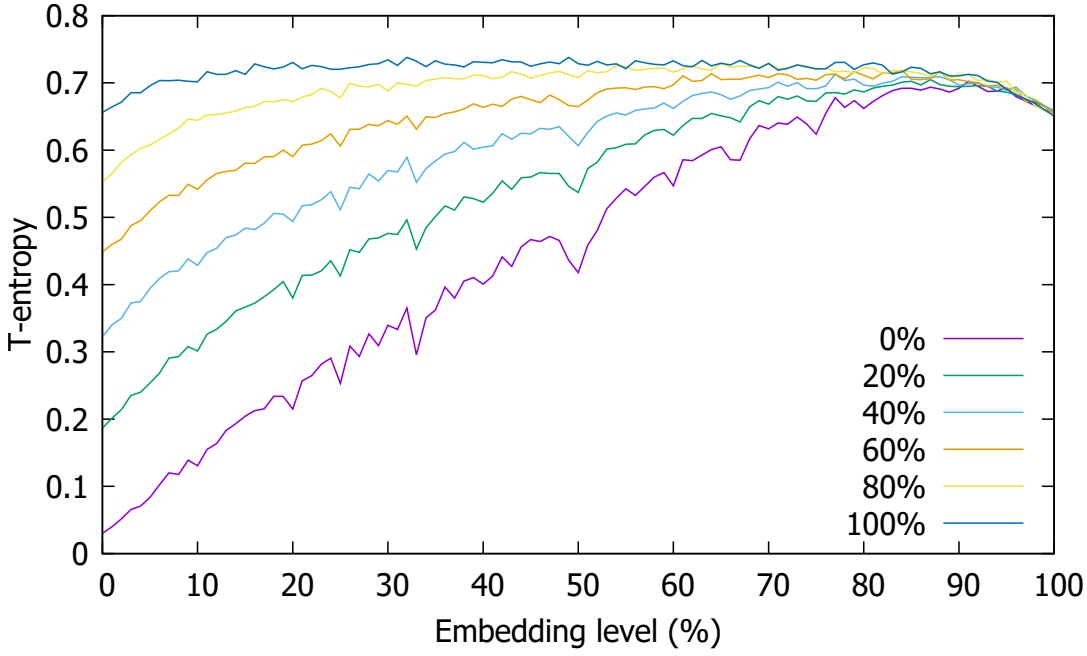


(b) Compliant mode where only the DF bit of the Flags field is modified.

Figure 4.5: (sequential, sequential) The effect of trial embedding a source embedded PCAP file. Both the source and trial embeddings used sequential embedding techniques. The T-entropy values start to rise once it exceeds its source embedding level.

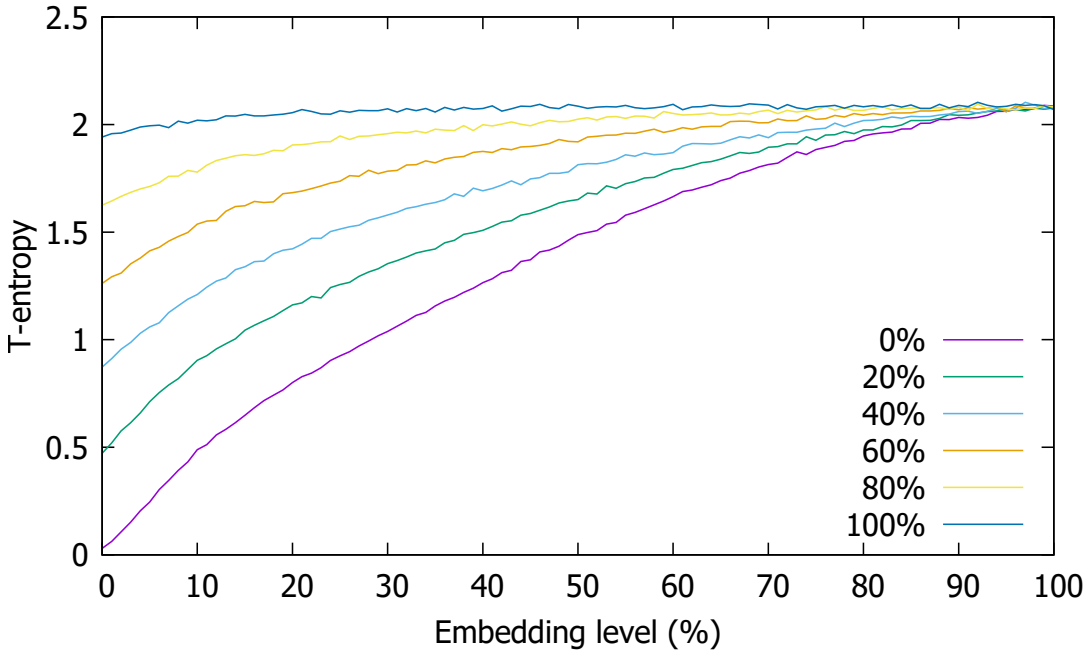


(a) Non-compliant mode where all of the bits of the Flags field are modified.

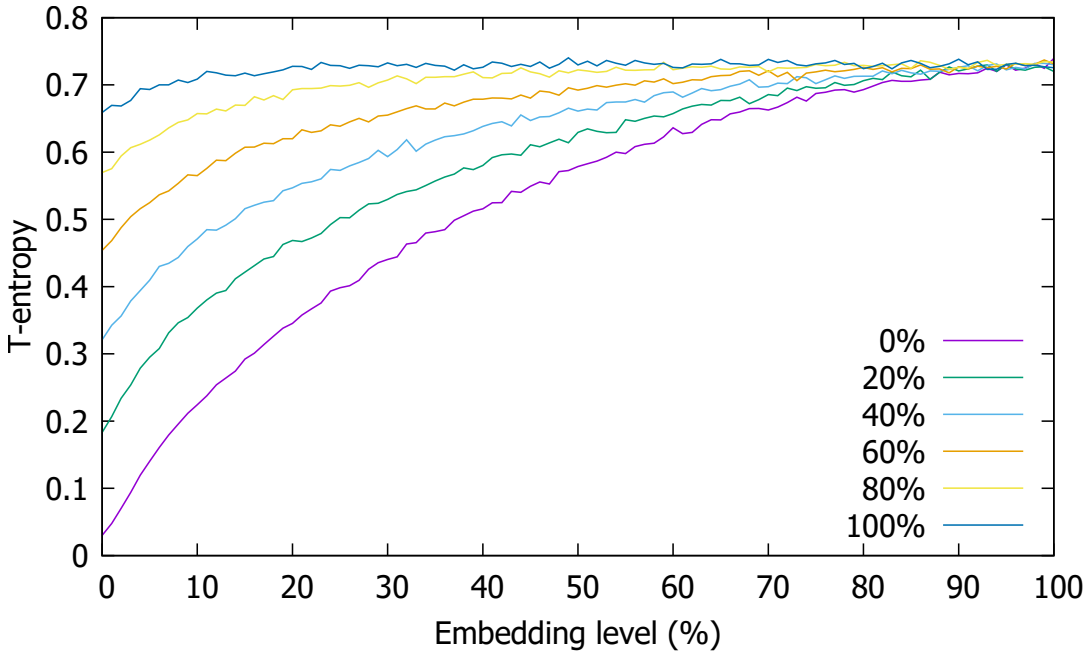


(b) Compliant mode where only the DF bit of the Flags field is modified.

Figure 4.6: (sequential, equal spacing) The effect of trial embedding a source embedded PCAP file. The source and trial embeddings used sequential and equal spacing embedding techniques, respectively. It appears this graph carries no significance except the high T-entropy values at the beginning.



(a) Non-compliant mode where all of the bits of the Flags field are modified.

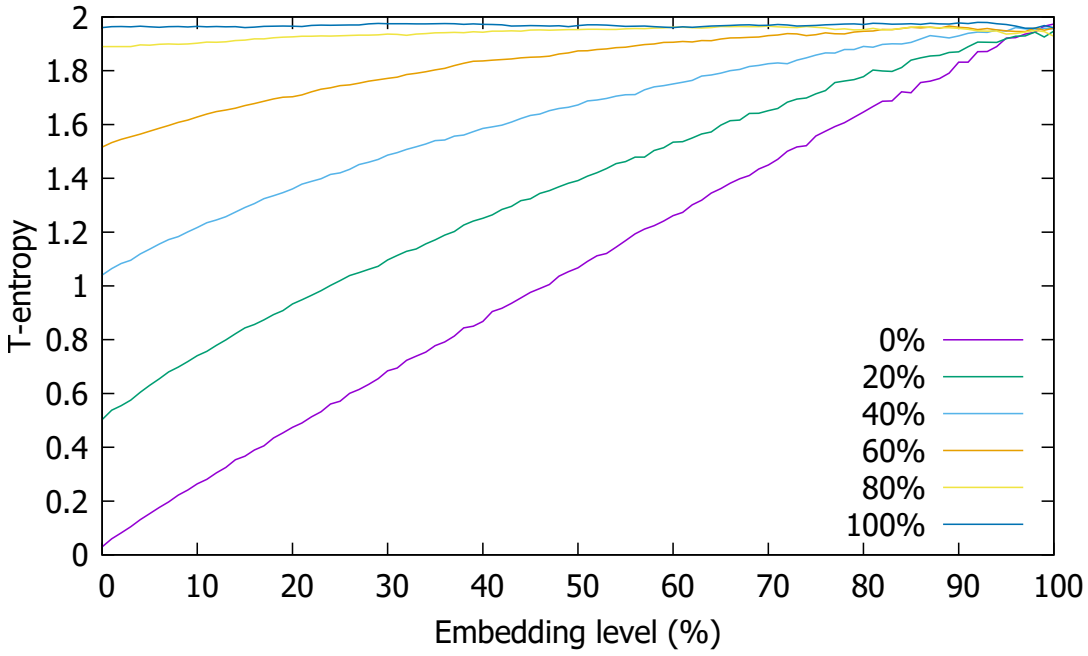


(b) Compliant mode where only the DF bit of the Flags field is modified.

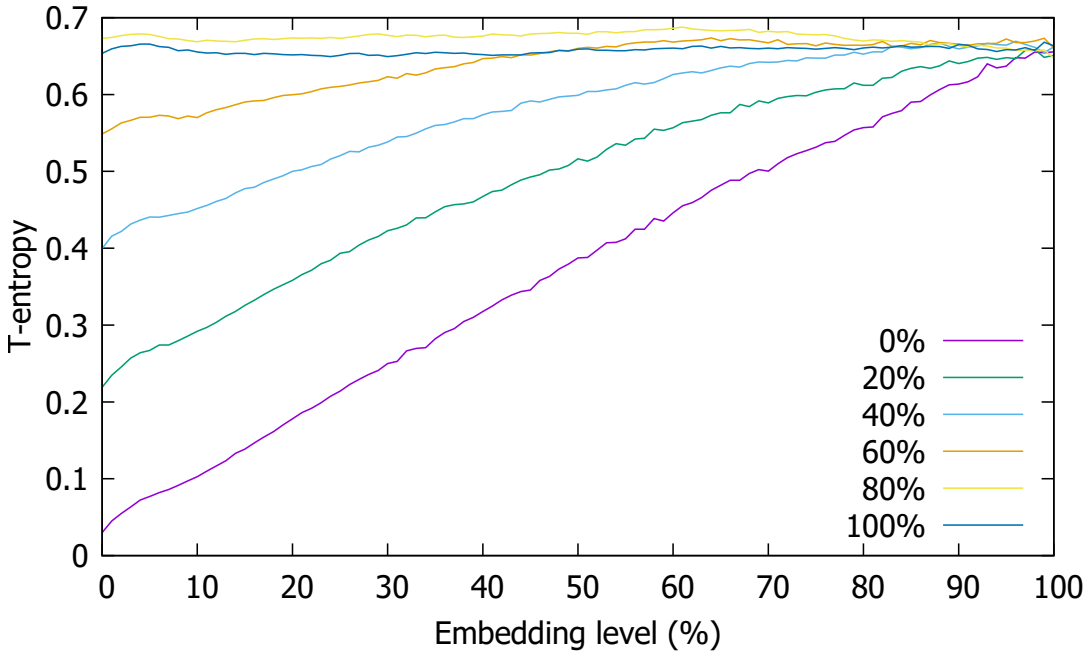
Figure 4.7: (sequential, random) The effect of trial embedding source embedded PCAP file. The source and trial embeddings used sequential and random embedding techniques, respectively. It appears this graph carries no significance except the high T-entropy values at the beginning.

4.7 Steganalysis (equal spacing, x)

Figures 4.8 and 4.10 show the results of (equal spacing, sequential) and (equal spacing, random) tuples, respectively. Non-matching embedding schemes, i.e., tuples with different source and trial embeddings, result in no significant outcome as also has been exhibited in the sequential tuples. Figure 4.9 shows the result of the (equal spacing, equal spacing) tuple, and strong fluctuations in the graphs can be observed. The T-entropy dips can be observed when the trial embedding reaches the same embedding level as the source embedding, as well as at power of 2 of that level. For example, if 10% of the embedding capacity were used during the source embedding, dips will occur at embedding levels of 10%, 20%, 40% and 80%. The reason behind this is that at these dips, the trial embedding modifies exactly the same positions as the source embedding. Thus, all the existing data is simply replaced with the new data.

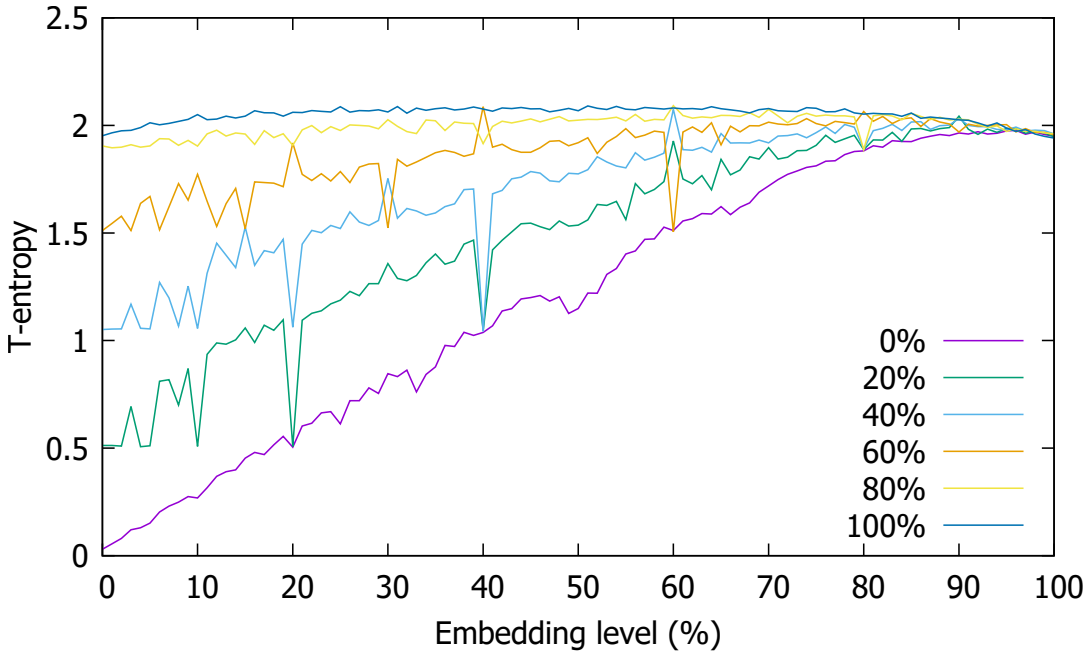


(a) Non-compliant mode where all of the bits of the Flags field are modified.

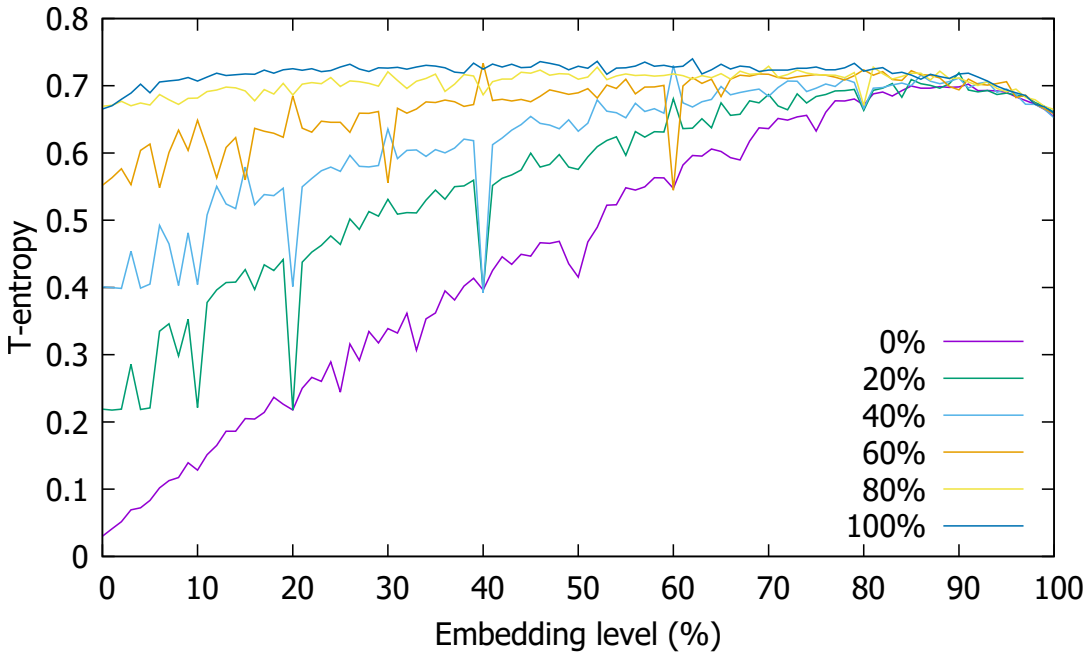


(b) Compliant mode where only the DF bit of the Flags field is modified.

Figure 4.8: (equal, sequential) The effect of trial embedding source embedded PCAP file. The source and trial embeddings used equal spacing and sequential embedding techniques, respectively. The graphs generally do not follow the trend of sequential embedding, thus it can indicate a presence of data.

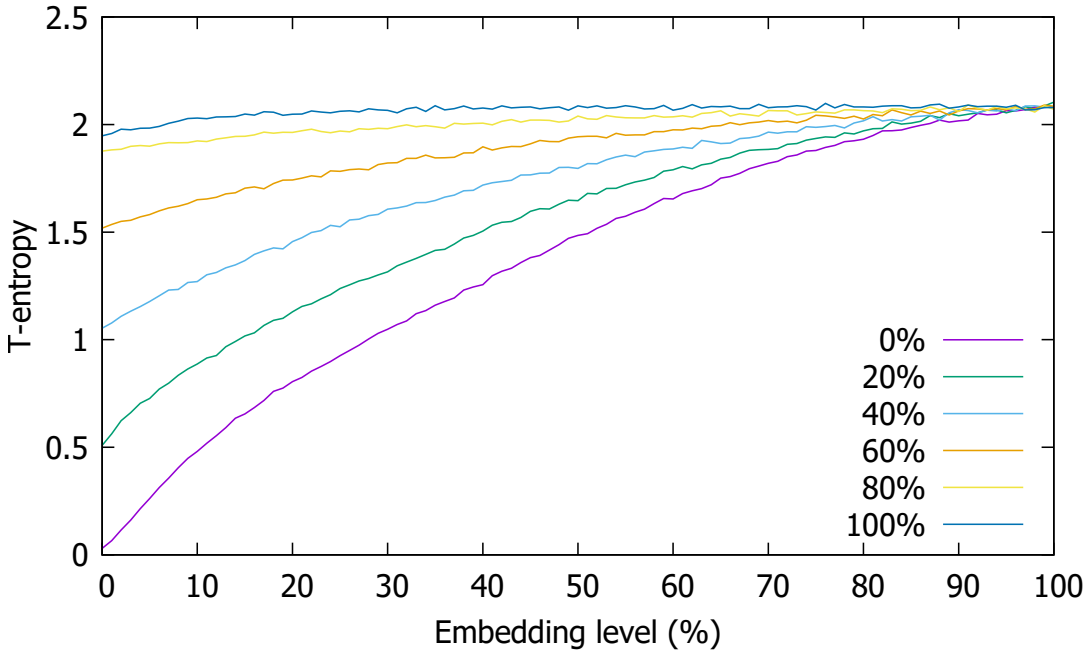


(a) Non-compliant mode where all of the bits of the Flags field are modified.

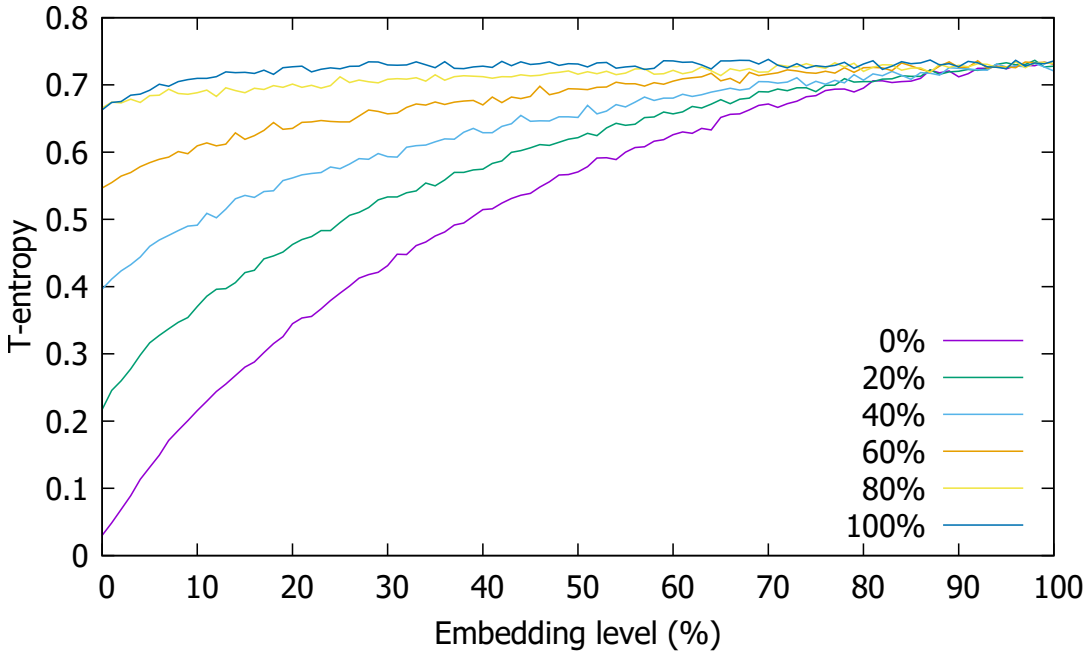


(b) Compliant mode where only the DF bit of the Flags field is modified.

Figure 4.9: (equal, equal) The effect of trial embedding source embedded PCAP file. Both the source and trial embeddings used equal spacing embedding techniques.



(a) Non-compliant mode where all of the bits of the Flags field are modified.



(b) Compliant mode where only the DF bit of the Flags field is modified.

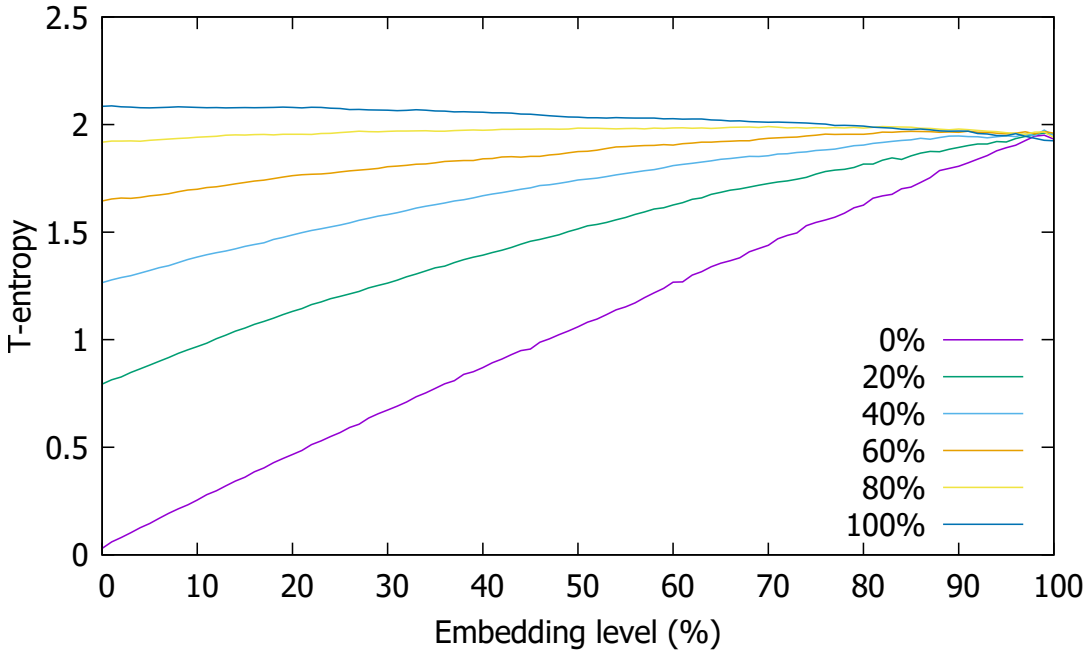
Figure 4.10: (equal, random) The effect of trial embedding source embedded PCAP file. The source and trial embeddings used equal spacing and random embedding techniques, respectively.

4.8 Steganalysis (random, x)

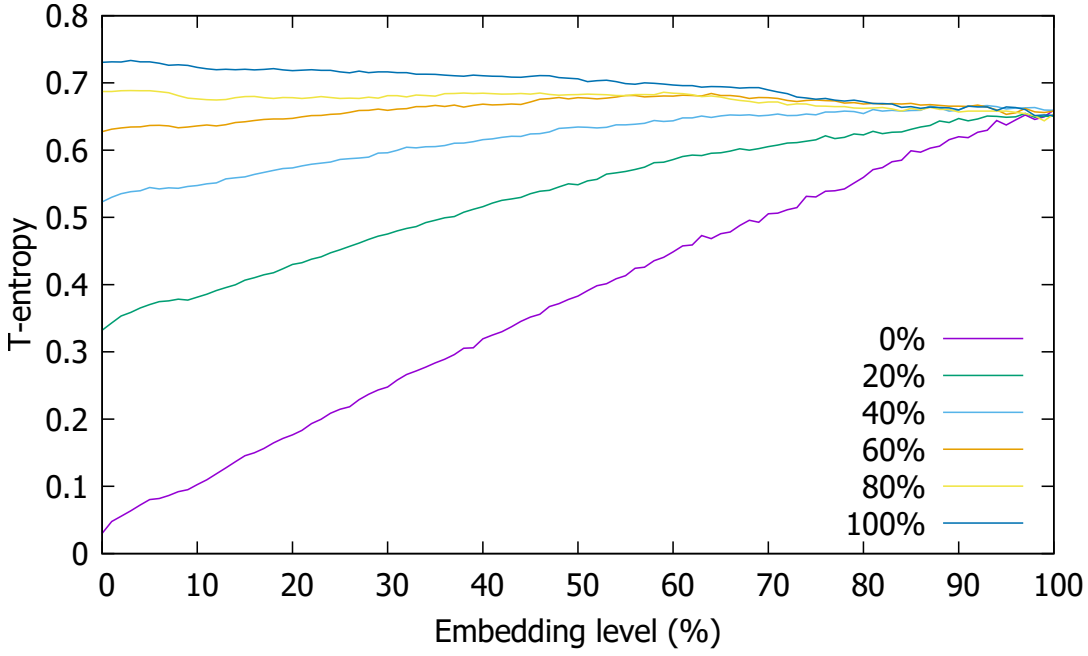
Figures 4.11, 4.12 and 4.13 show the results of (random, sequential), (random, equal spacing) and (random, random) tuples, respectively. The figures, unfortunately, do not expose anything extraordinary except the different slopes of the lines.

The (random, random) tuple which has a matching source and trial embedding schemes not exposing any trend is uncommon, as opposed to what has been observed in Steganalysis (sequential, x) and (equal spacing, x). This phenomenon may be explained with the lack of correlation in the random embedding. For the (sequential, sequential) tuple, one overwrites existing secret data with another Lorem Ipsum message. This has near no effect on T-entropy until the overwritten data size is greater than the source embedding. For the (equal spacing, equal spacing) tuple, one overwrites exact same packets when the source and trial embedding data capacity match. The (random, random) tuple, however, does not guarantee to manipulate the same packet positions as the source embedding.

Figure 4.14 illustrates the linear regression lines of the (random, random) tuple under the compliant mode. As may be seen from the figure, the higher the source embedding the flatter the graph becomes. A linear equation is defined as $y = mx + b$, where x and y are the coordinates, m is the coefficient and b is the y-intercept. The coefficient m determines the gradient of the curve, and m of the various embedding levels is as follows: 0.0063 (0% embedding), 0.0038 (20% embedding), 0.0021 (40% embedding), 0.0010 (60% embedding), 0.0003 (80% embedding), -0.00001 (100% embedding). The coefficient and y-intercept values follow a decreasing and increasing trends, respectively. It thus confirms a detection model based on the linear regression may be usable. It should, however, be noted that this is beyond the research interest of this thesis as discussed in Chapter 1.1. We reiterate that the comparison-based statistical measures are bound to face scalability and maintainability issues in the long run. Let us assume one observes

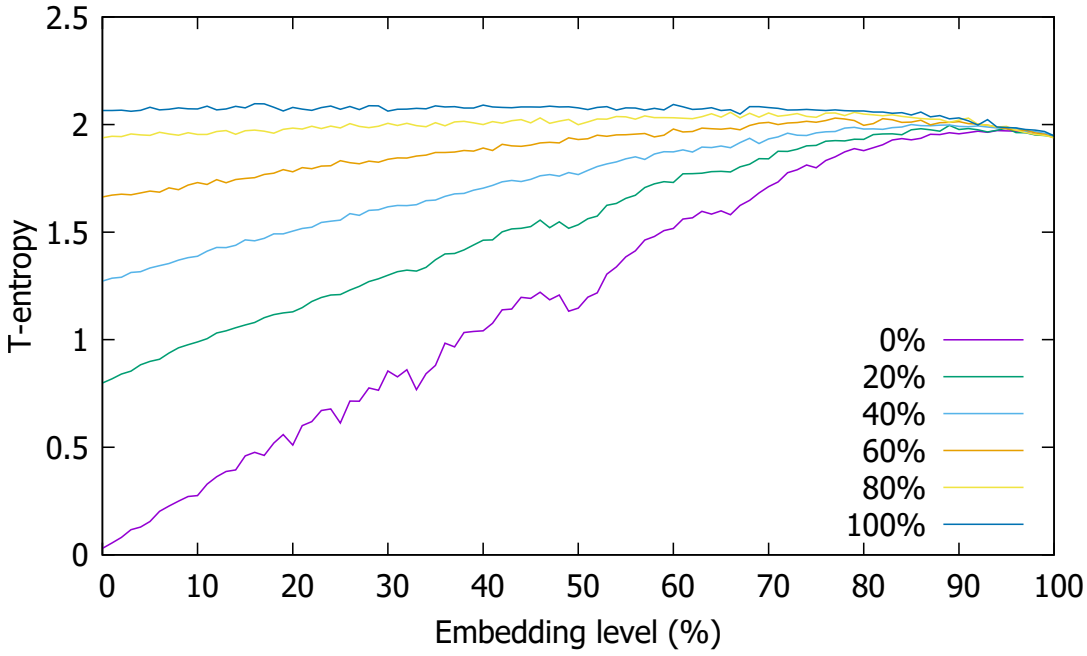


(a) Non-compliant mode where all of the bits of the Flags field are modified.

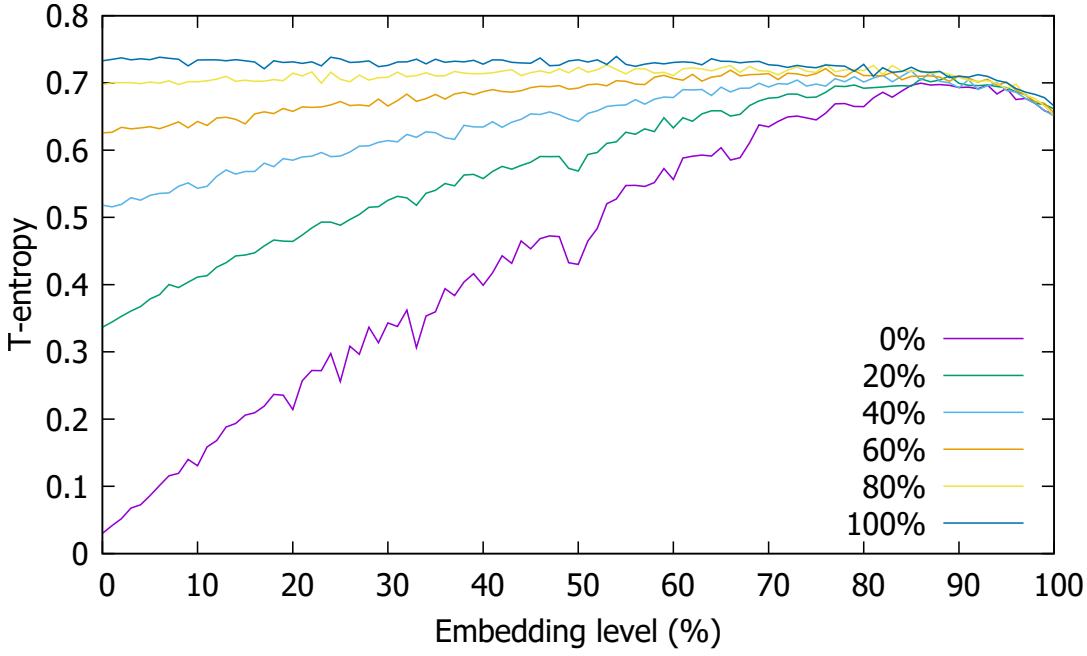


(b) Compliant mode where only the DF bit of the Flags field is modified.

Figure 4.11: (random, sequential) The effect of trial embedding source embedded PCAP file. The source and trial embeddings used random and sequential embedding techniques, respectively.

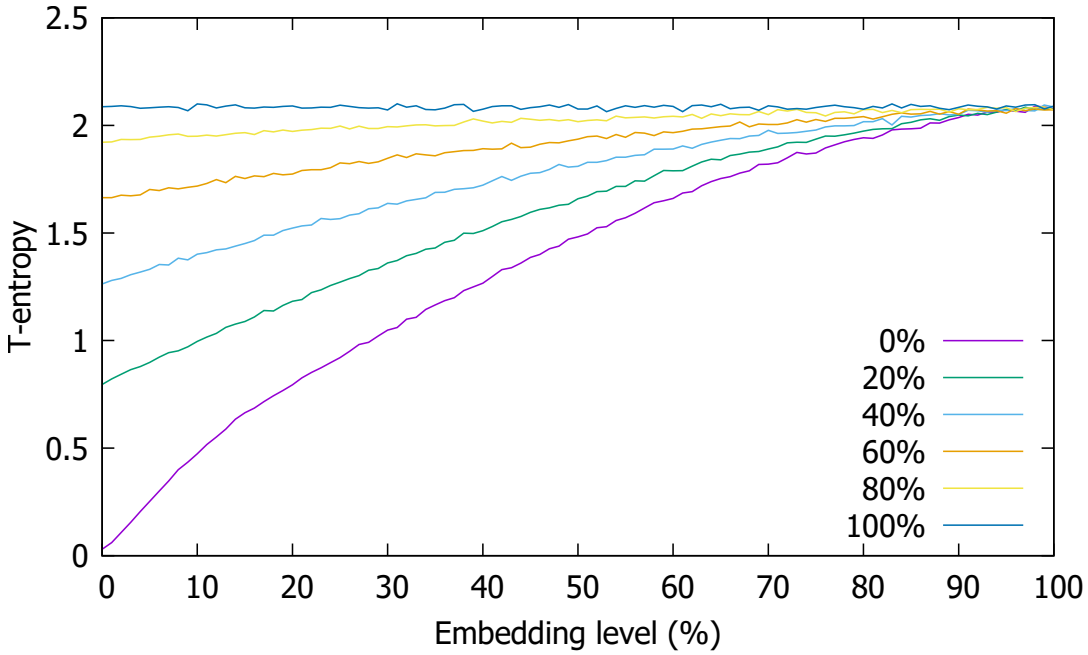


(a) Non-compliant mode where all of the bits of the Flags field are modified.

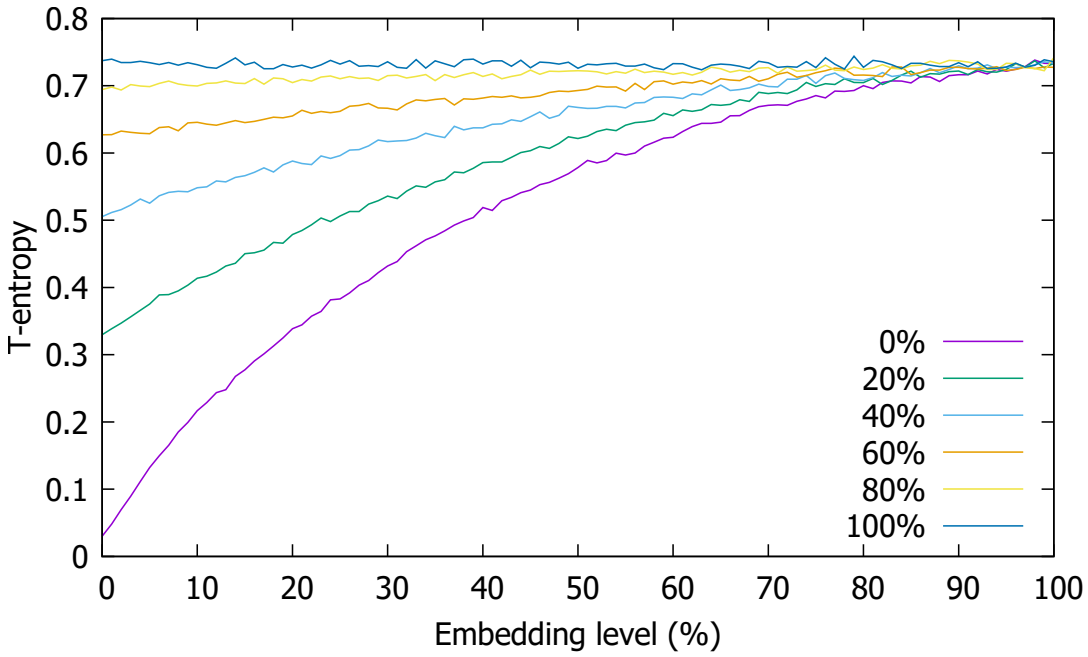


(b) Compliant mode where only the DF bit of the Flags field is modified.

Figure 4.12: (random, equal) The effect of trial embedding source embedded PCAP file. The source and trial embeddings used random and equal spacing embedding techniques, respectively.



(a) Non-compliant mode where all of the bits of the Flags field are modified.



(b) Compliant mode where only the DF bit of the Flags field is modified.

Figure 4.13: (random, random) The effect of trial embedding source embedded PCAP file. Both the source and trial embeddings used random embedding techniques.

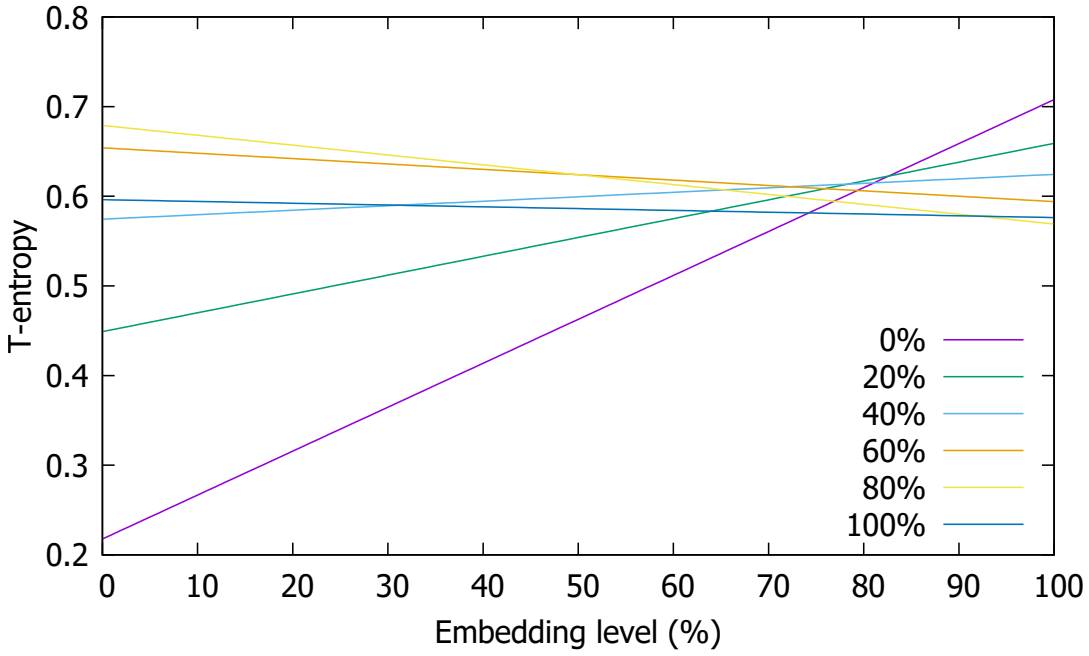


Figure 4.14: Linear regressions of Figure 4.13 (random, random) graphs. More source embedding generally lowers the coefficient (i.e., gradient) and increases the y-intercept.

a flag fields gradient of 0.0038 in their network flow and decides a decision boundary of 0.0005. Then any flow with the gradient between 0.0033 and 0.0043 is set as a benign flow. If the network condition changes and the number of the IP fragmentation is reduced, then one can expect false positive alarms. Likewise, a false negative alarm can be expected when an attacker utilises the channel capacity which happens to match with a predetermined gradient.

4.9 Discussion

The re-embedding steganalysis is able to detect and estimate the amount of secret data in the network storage channel. While we looked only at one semi-unused header field of IP flags, re-embedding steganalysis can undoubtedly be used for other semi-unused or unused header fields, and yield a similar result. T-entropy takes correlation into

account. Hence, as soon as the header fields that are supposedly unused or semi-unused are filled with correlated data, re-embedding steganalysis will excel at detecting it. It has been seen that T-entropy can withstand noise from uncorrelated data. We have noted that both the compliant and non-compliant modes of re-embedding steganalysis produce similar-looking graphs, thanks to T-entropy's ability to withstand noise.

Patterns from (sequential, sequential) and (equal spacing, equal spacing) tuples are useful in detecting the storage-based network steganography. In most of the other tuples, the slope of a line might be used in detecting network steganography. For instance, the gradient can be estimated via *linear regression*, and one might use this value to determine if certain flow should be viewed as legitimate or not.

4.10 Summary

In this chapter, we investigated experimentally the effectiveness of T-entropy in detecting storage-based network steganography. In addition, we tested the following three embedding schemes: sequential, equal spacing and random embeddings.

Our results show that the re-embedding steganalysis technique is not only effective for the detection but also for estimating the size of secret data. The same applies to the equal spacing embedding where the T-entropy dip can be used as an indicator. However, no observable pattern could be seen in case of random embedding. It should, however, be noted that the gradients of the graphs can be used for the detection but that is a comparison-based approach, which contradicts with what this thesis is trying to achieve.

While promising results can be seen from the re-embedding steganalysis graphs, manual human auditing is necessary. That is, a human has to manually identify a point in a graph that starts to increase or a dip is. The re-embedding steganalysis technique may, therefore, be unsuitable for network use case because the amount of network traffic that needs to be audited would be well beyond the speed one can manually audit. In the next chapter, we discuss the journey of building an automatic detector to address such limitation.

5

Automatic Detectors for Re-embedding Steganalysis

In the previous chapter, re-embedding steganalysis has shown its effectiveness in detecting both the presence and the amount of hidden information in storage-based steganography. The technique only draws a graph which later has to be manually inspected by a human on whether a flow is malicious or not. This approach is not practical, however.

This chapter thus discusses a journey of granting "intelligence" to the re-embedding steganalysis system. The system, therefore, is capable of forming similar judgements to the human. To this end, the system has to identify a major trend change in a graph, and consequently, this chapter experimentally evaluates a number of trend detection and smoothing techniques.

The rest of the chapter is organised as follows: Section 5.1 recapitulates the findings in the previous chapter, and defines specifications of the detector system. Section 5.2 presents selected related work in detection and smoothing techniques. Sections 5.3 – 5.7 share the journey of building the detector system for the (sequential, sequential) tuple. Section 5.8 uses the insight gained from the previous sections to build the detector system for the (equal spacing, equal spacing) tuple. The final section summarises the chapter.

5.1 Background

Re-embedding steganalysis re-embeds a suspicious cover with secret data at various capacity levels, and thereby attempts to arrive at the level of the source embedding. The previous chapter introduced the merits of re-embedding steganalysis with how its output, a graph, can be assessed by a human for potential steganographic channel usage. This approach of finding the amount of hidden information is impractical in a real world scenario, however, as networks can be busy with a number of network flows too large for human analysis.

As discovered in the previous chapter, (sequential, sequential) and (equal spacing, equal spacing) tuples showed noticeable trends. Figure 5.1 illustrates this. In the (sequential, sequential) tuple graph, the T-entropy does not increase as information is embedded in the trace – until the hidden information exceeds 20%. We refer to this section where the gradient of the graph stays quasi-constant at around 0 as the *horizontal section*. Beyond 20%, the graph shows an almost linear increase in T-entropy. We therefore refer to this section as the *increasing section*. The major turning point in the graph (at the 20% mark in this particular case) represents the amount of hidden information (as a percentage of the hiding capacity). This is also where the gradient transitions, so we refer to this point as a *trend transition point*. In the (equal spacing, equal spacing) tuple graph, the T-entropy value fluctuates during its trial embeddings. The T-entropy value dips when the trial embedding reaches the same embedding level as the source embedding, as well as at powers of 2 of that level.

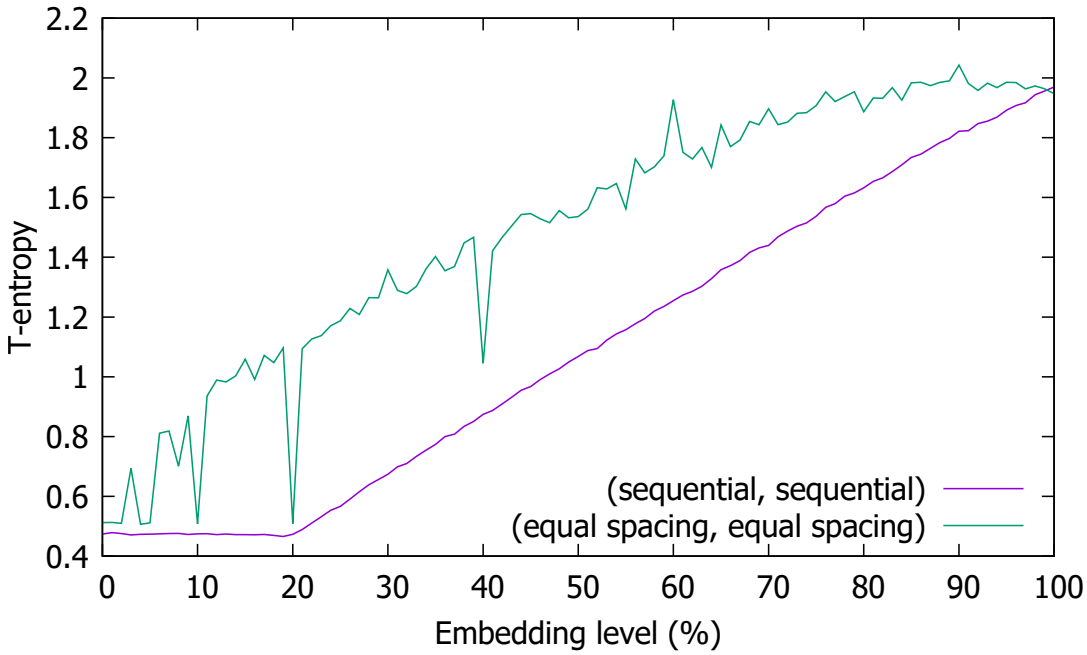


Figure 5.1: T-entropy vs. embedding level (i.e., new content embedded). Once the trial embedding exceeds the source embedding value of 20%, the T-entropy value rises in the (sequential, sequential) tuple. On the other hand, one of the dips indicates the source embedding level in the (equal spacing, equal spacing) tuple.

5.2 Related work

The change in trends described above can effortlessly be detected with human eyes, and we seek to develop an automatic detector that can achieve the same.

Identifying or predicting a turning point has a number of uses. In a volatile financial market, predicting correct turning points makes a significant difference. Li et al. demonstrate turning points of "chaotic characteristic in financial time series" can be identified using machine learning techniques [99]. Identifying trends in past rainfall data to forecast rainfall allows a country to efficiently manage water resources [100, 101]. This is a critically essential task to perform to survive through climate change.

Calculating a turning point can be simple. In differential calculus, if one knows a function of a curve, $f(x)$, one can calculate its first derivative, $f'(x)$ [102], which identifies the gradient of a curve at the specific data point. A significant change in the gradient marks the turning point.

Denoising a dataset may be a necessary step before further analysis can be undertaken. Kay et al. demonstrate how their model, GLMdenoise, improves cross-validation accuracy [103]. This leads to a better signal-to-noise ratio reading. Abdelhamed et al. discuss denoising techniques and demonstrate that when a Convolutional Neural Networks (CNN)-based method is trained with high quality datasets, it performs better than the alternatives [104].

5.3 Simple interpolation

This section discusses if simple interpolation can detect different embedding levels of (sequential, sequential) tuple graphs in Figure 5.2 by detecting the trend transition point.

We first explored the possibility of using interpolation to detect the trend transition. Given a set of finite data points, one can use interpolation to generate a curve and an

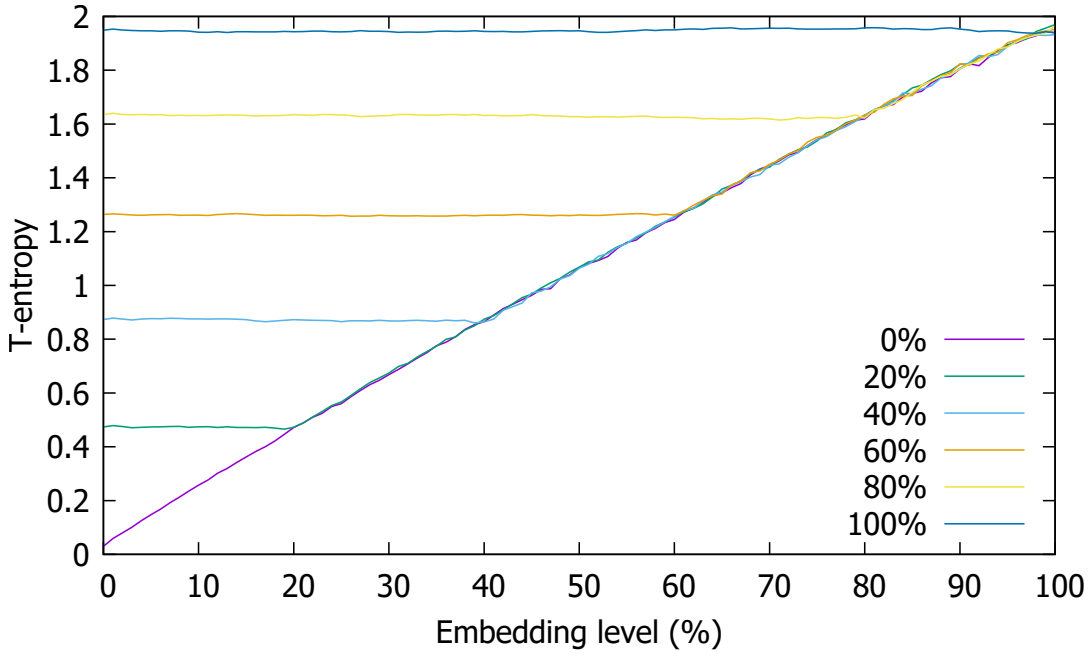


Figure 5.2: The effect of trial re-embedding a source-embedded packet capture file. The T-entropy values start to rise once the re-embedding exceeds the original embedding level in the source. Each curve represents an aggregation of 100 datasets.

underlying mathematical function $f(x)$ that describes the curve. The mathematical function can interpolate any point in the curve that was otherwise unavailable from the dataset. This is possible because the interpolation creates a continuous curve by connecting a set of data points. Typically, *linear*, *polynomial*, or *spline* functions are used for interpolation. With a mathematical function, one can also find a first derivative $f'(x)$ to obtain the gradient of a curve at a specific data point. In the ideal case, the gradient is expected to remain zero during the horizontal section, and then stay positive from the trend transition point onwards.

Figure 5.3 shows one of the 100 datasets from Figure 5.2, highlighting the fluctuations in the dataset. In statistics and signal processing, the term noise is often used to describe an additive component in the signal or error in the data whose instantaneous value cannot be described deterministically but whose statistical distribution or other properties may be known. The interpolation approach would not work because noise causes frequent short-range gradient changes and obscures the expected global trend transition behaviour. Moreover, the noise does not exhibit a noticeable pattern across different datasets (see Appendix B.1. This may prove to be a potential hurdle in building the automatic detector.

However, the trend persists, i.e., the curve appears to increase once the re-embedding level exceeds the source embedding level. Smoothing the dataset appears to be the logical step to take to reduce the noise. Two different smoothing techniques, Moving Average (MA) and Regression Analysis (RA), are therefore investigated and experimentally evaluated.

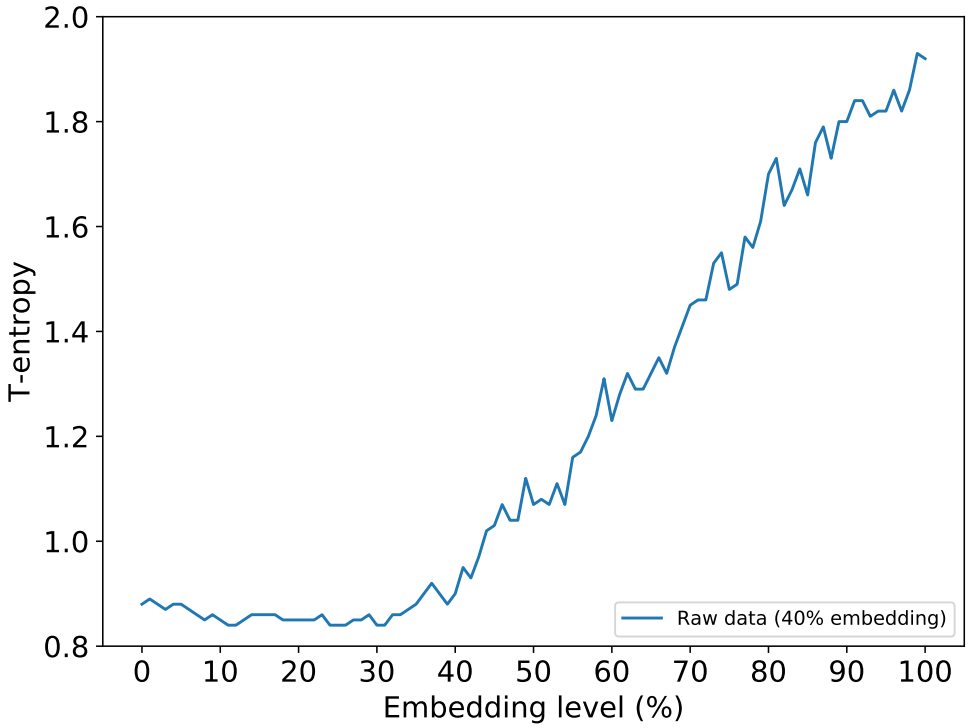


Figure 5.3: A single one of the 100 datasets with 40% embedding from Figure 5.2. Fluctuations are more pronounced, but the trend transition around the 40% mark is still evident.

5.4 Moving average

Moving Average (MA) is a smoothing algorithm to remove noise from a dataset. MA can either be weighted or unweighted; each with its own applications. Exponential Moving Average (EMA) is an example of a weighted MA algorithm where recent data points outweigh past data points. Stock markets are one of the applications of EMA where recent stock prices may be regarded as more relevant. On the other hand, Simple Moving Average (SMA) is unweighted and treats every data point equally. SMA is the obvious choice for T-entropy smoothing as there is no justification for weighting in this case. The *right-edged* SMA is defined as:

$$SMAr_p = \frac{x_p + x_{p-1} + \ldots + x_{p-(n-1)}}{n} \tag{5.1}$$

where x_p is the value of p-th element in the dataset and n is the window size.

Figure 5.4 shows the effect of various SMA window sizes on Figure 5.3. It shows that larger window sizes result in smoother curves. The noise in the dataset appears to be masked for $n \geq 10$, albeit at the cost of a lower average as the SMA window includes lower values to the left of x_p. A *centred* SMA window can mitigate such problems:

$$SMAc_p = \frac{x_{p-\frac{(n-1)}{2}} + \ldots + x_{p-1} + x_p + x_{p+1} + \ldots + x_{p+\frac{(n-1)}{2}}}{n}, \tag{5.2}$$

Figure 5.5 compares the smoothing effects of right-edged and centred SMA. The figure shows that the centred SMA appears to result in a curve that better represents the dataset. While the centred SMA with $n = 10$ appears to result in a satisfactory smoothing result for Figure 5.3, this may not be the case for other datasets.

Figure 5.6 illustrates this for a single dataset with 70% embedding level smoothed with $n = 10$ and 20. The figure shows that the noise around the horizontal section is marginally better suppressed when n is increased to 20 from 10. However, this is not

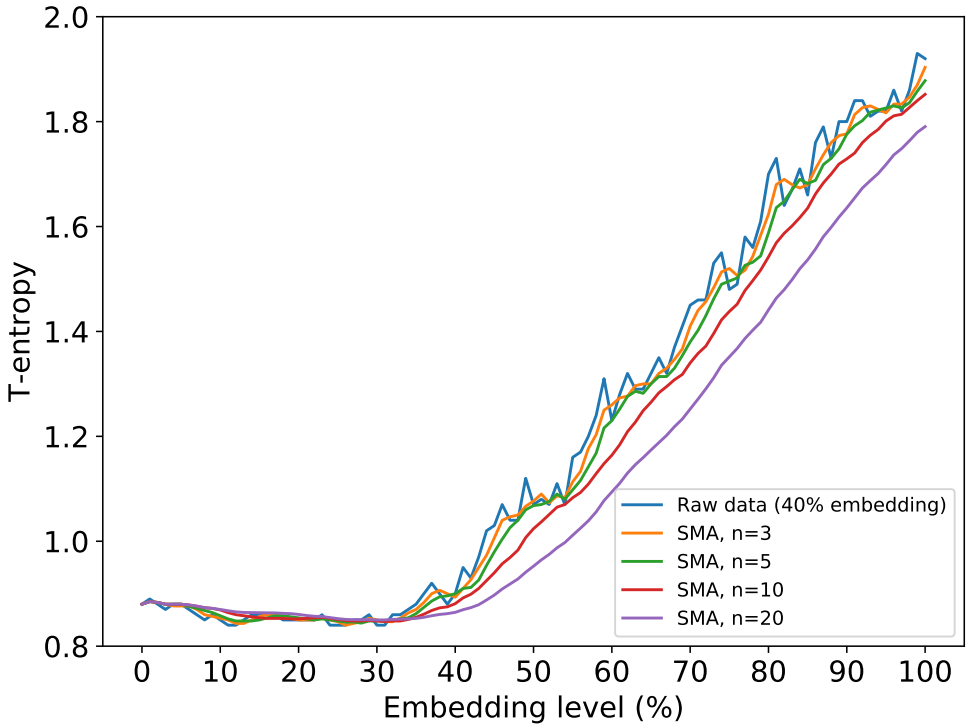


Figure 5.4: The effect of trial embedding a source embedded PCAP file with SMA smoothing of the data points. The window size n defines the degree of smoothing. Values of n below 10 appear to retain some of the noise.

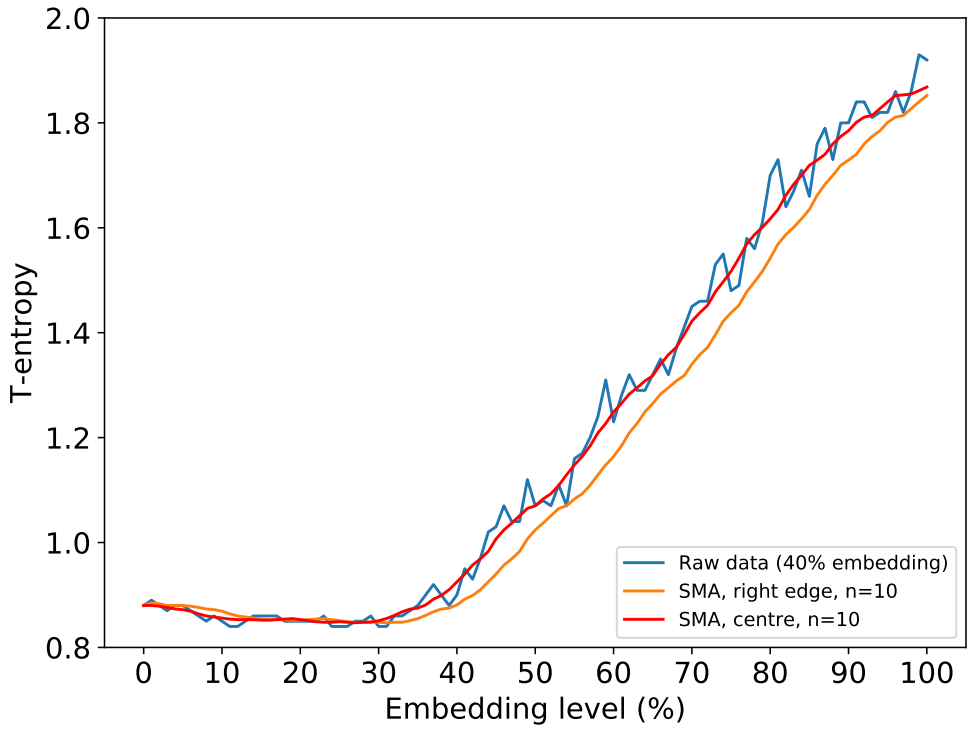


Figure 5.5: A centred SMA window mitigates the low average problem of right-edged SMA in Figure 5.4.

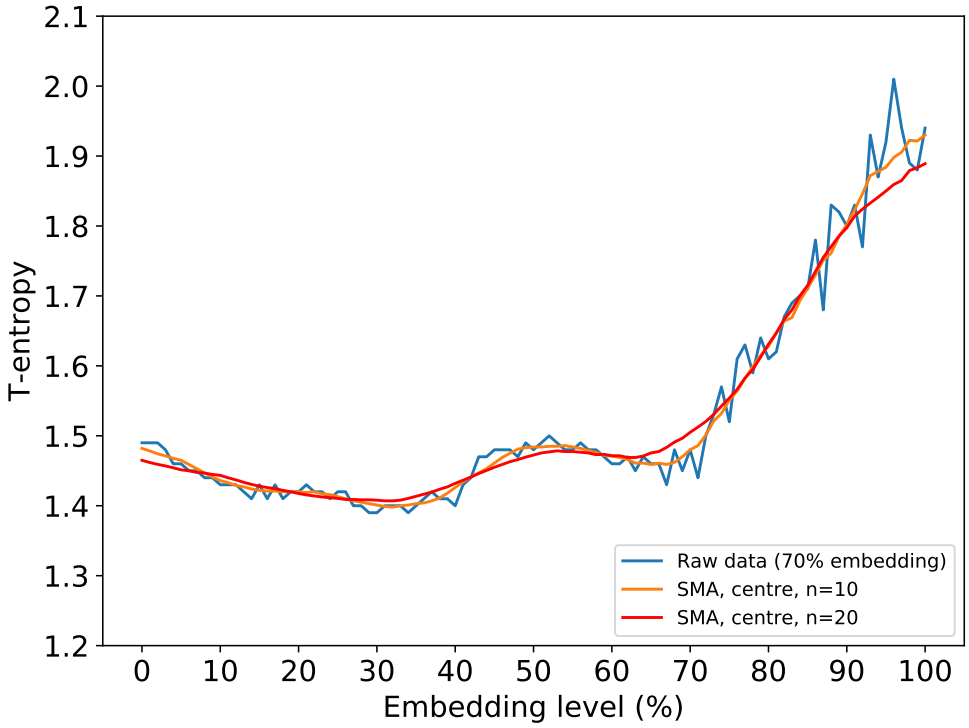


Figure 5.6: The effect of trial embedding a source embedded PCAP file. Both the source and trial re-embedding used sequential embedding techniques. The figure illustrates the impact that window size has on the curve. The window size of 20 appears to result in a marginally better curve than 10. horizontal section noise, however, is still preserved in both cases.

satisfactory because the horizontal section noise is still present. A fundamental question one may ask is whether there exists a way to quantify which n best represents a dataset. There is no definitive answer to this, and the datasets are at a risk of facing either *under-smoothing* or *over-smoothing*. Under-smoothing occurs when small values of n preserve too much detail, whereas excessive n may cause over-smoothing and a loss of the trend.

5.5 Regression analysis

SMA's difficulty in determining a useful value for n motivates a search for an alternative. Regression analysis (RA) shares some similarities with interpolation: It generates a curve and an underlying mathematical function. In RA, a curve, however, may not pass through data points because it generates a fitted polynomial curve, which represents an approximated model of the dataset. This may result in a certain degree of smoothing.

Figure 5.7 illustrates RA on Figure 5.3. The degree specifies the polynomial degree a dataset is fitted to. Selecting the correct degree avoids *underfitting* or *overfitting*, which occur as a result of excessive or insufficient generalisation of a model, respectively. Degree 1 is a straight line and demonstrates the effect of underfitting. The degree 1 subfigure shows that the trend transition is lost completely. The effect of overfitting may be observed in the degree 20 subfigure. While the trend transition is present here, the fit retains some of the noise around the increasing section. The degree 10 subfigure comes close to the best fit where the curve represents the dataset adequately. In summary, the following two points can be stated: Firstly, degree 1 RA must be avoided as it always destroys the trend transition. Secondly, it is not guaranteed that degree 10 will generate the best-fit curves for other datasets. Figure 5.8 illustrates this for a single dataset at 70% embedding level, where degree 10 retains the horizontal section noise while degree 3 masks it.

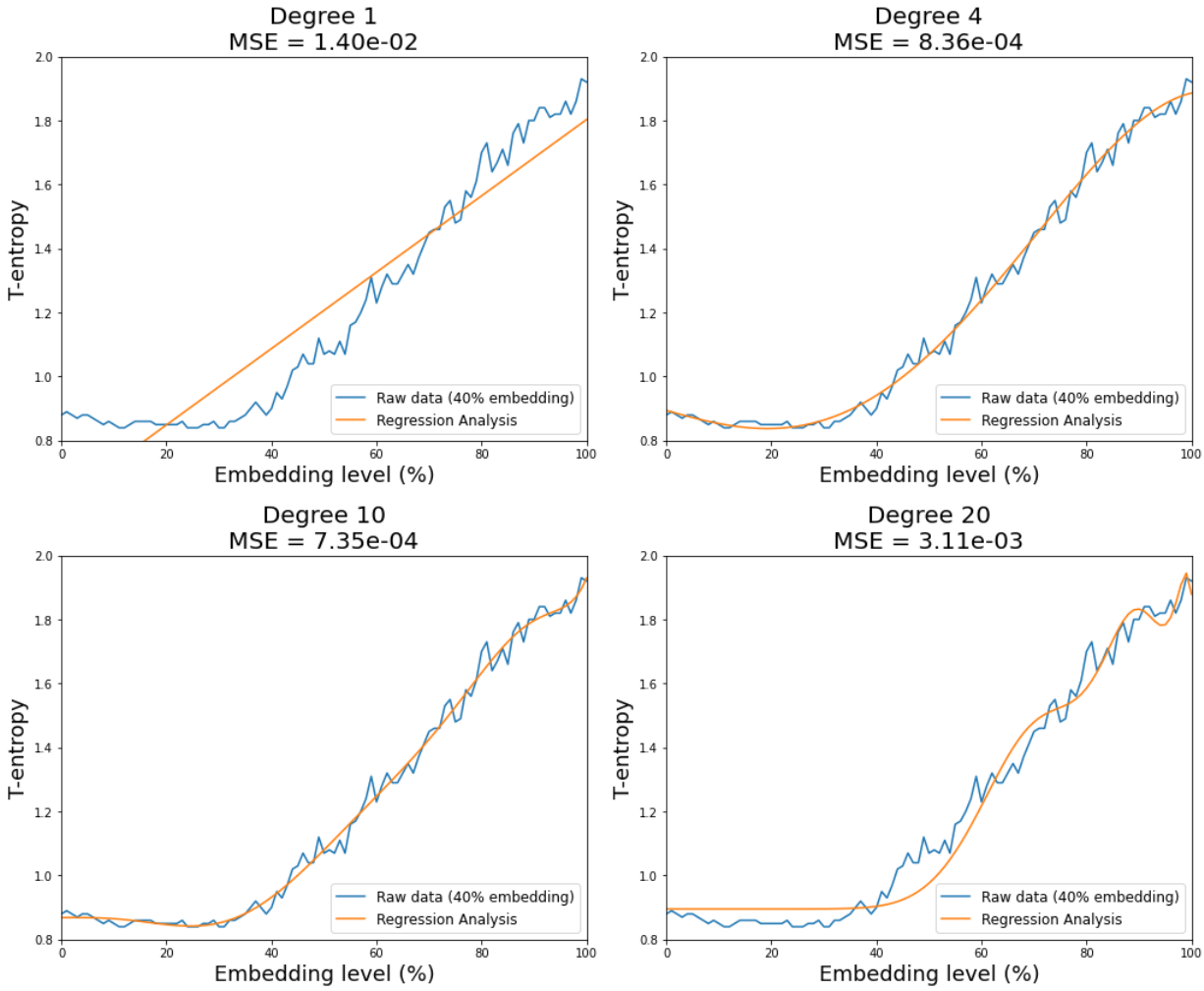


Figure 5.7: Regression analysis on Figure 5.3 with varying degrees. Degrees 1 and 20 show the effect of underfitting and overfitting, respectively. A good fit is indicated by a low mean squared error (MSE). Degree 10 has the smallest MSE, adequately suppressing noise in the horizontal section.

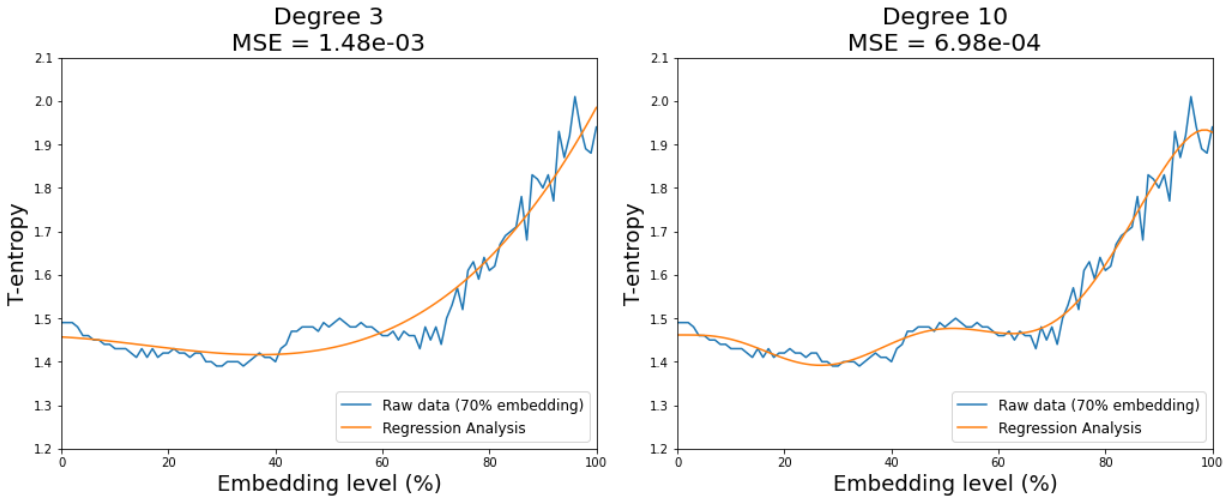


Figure 5.8: Degree 3 and degree 10 polynomial regressions on a single dataset with 70% embedding level: The degree 10 polynomial retains the noise in the horizontal section, while the degree 3 polynomial removes it.

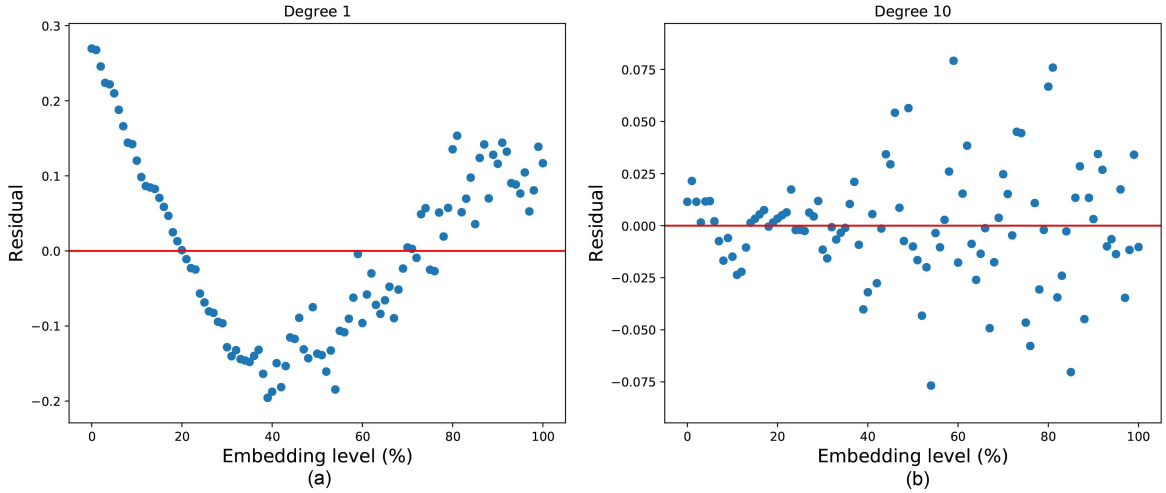


Figure 5.9: Residual plots of Figure 5.3 on (a) degree 1 and (b) degree 10 polynomials. The residual for the degree 1 polynomial exhibits declines and inclines because the linear polynomial cannot capture any trend in the dataset. On the other hand, residuals are scattered around the degree 10 polynomial, indicating a good fit.

Regression validation analyses a fitted model by assessing *goodness of fit*, which defines how well the model corresponds to the data points. The null and alternative hypotheses of goodness of fit are defined as:

$$\begin{aligned} H_0 &: M \text{ fits } D \\ H_1 &: M \text{ does not fit } D, \end{aligned} \tag{5.3}$$

where M is the fitted model and D is the dataset. However, for our purposes, H_0 needs to be altered because we require the fitted model to fit the dataset just enough to suppress the noise without losing the trend transition. Thus, our H_0 is defined as: M generalises D sufficiently without losing the trend transition. The model's goodness of fit can be assessed in different ways, but the common approach in regression validation uses the *residual*.

The residual is the difference between the data point and the corresponding fitted model value. Thus, the residual is defined as: $e_i = y_i - \hat{y}_i$, where y_i is the i^{th} data point in the dataset and $\hat{y}_i$ is the corresponding fitted model value. Figure 5.9 shows degree 1 and degree 10 polynomial residual plots of Figure 5.3. A model is considered a perfect fit if all residuals are 0. A perfect fit, however, may not be ideal because it signals the possibility of overfitting.

A sparse residual plot without any trend is ideal as a trend in the residual plot suggests the failure of the model. The degree 1 subfigure exhibits decline and incline trends, whereas the degree 10 subfigure shows no noticeable trend. The residual plot approach for finding the best-fit polynomial has a couple of drawbacks, however: Firstly, manually selecting the best random-looking plot among the different degrees of polynomial plots can be time-consuming; secondly, determining the trend of a plot is a subjective process. For instance, one might argue that the residual fans out in the degree 10 subfigure.

Alternatively, *error metrics* may be used to evaluate the goodness of fit. There are a number of different error metrics available – mean squared error (MSE), R-squared

Table 5.1: Experimenting with a range of error metrics with varying polynomial degrees from 1 to 15 on Figure 5.3. Degree 10 provides the logical best-fit as indicated by the blue font.

Degree	R^2	Mean Absolute Error	Explained Variance Score	Mean Squared Logarithmic Error	Max Error	Median Absolute Error	Mean Squared Error	Cross-Validation 10-folds
1	0.8964	0.1021	0.8964	0.0035	0.2695	0.0961	0.0140	0.0239
2	0.9795	0.0419	0.9795	0.0005	0.1380	0.0373	0.0028	0.0072
3	0.9916	0.0269	0.9916	0.0002	0.1014	0.0222	0.0011	0.0035
4	0.9938	0.0214	0.9938	0.0002	0.0894	0.0156	0.0008	0.0012
5	0.9940	0.0209	0.9940	0.0002	0.0850	0.0139	0.0008	0.0014
6	0.9940	0.0210	0.9940	0.0001	0.0844	0.0145	0.0008	0.0026
7	0.9940	0.0209	0.9940	0.0001	0.0840	0.0148	0.0008	0.0359
8	0.9943	0.0207	0.9943	0.0001	0.0854	0.0134	0.0008	0.0034
9	0.9945	0.0195	0.9945	0.0001	0.0806	0.0130	0.0007	0.0027
10	0.9946	0.0195	0.9946	0.0001	0.0791	0.0119	0.0007	0.0257
11	0.9941	0.0210	0.9941	0.0002	0.0810	0.0159	0.0008	0.2206
12	0.9939	0.0215	0.9939	0.0002	0.0814	0.0170	0.0008	0.4405
13	0.9940	0.0213	0.9940	0.0002	0.0786	0.0160	0.0008	0.4545
14	0.9940	0.0213	0.9940	0.0002	0.0755	0.0159	0.0008	0.1159
15	0.9936	0.0220	0.9936	0.0002	0.0809	0.0187	0.0009	0.3080

(R^2), and mean absolute error (MAE), to name a few. The error metrics generally use residuals in their calculation. For example, the MSE equation shown below illustrates its use of residuals:

$$MSE = \frac{1}{n}\sum_{n=0}^{n-1}(e_i)^2, \tag{5.4}$$

where n is the number of the data points in the dataset. Table 5.1 shows the result of experimenting with a range of different error metrics on Figure 5.3. As may be seen from the table, a polynomial degree of 10 appears to provide best fit for the figure because it usually resulted in a better fit than any other degrees for most of the metrics. In the following section, however, we show that this approach results in overfitting in our use case.

5.6 k-fold cross-validation

Using the same procedure as for Table 5.1 on the 70% embedding level dataset from Figure 5.8 indicates that degree 10 is still the best fit. Degree 3, however, is the better fit because it suppresses noise in the horizontal section. While the degree 10 polynomial model may best describe our dataset with the smallest errors, we require a lower degree, as our null hypothesis demands that M must fit D without losing the trend transition. *Cross-validation* (CV) allows us to meet this criterion.

CV is mainly used in machine learning where prediction performance is one of the major concerns. If a model overfits a training dataset, a poor prediction performance may be observed towards the *out-of-sample* (unseen data). There are different CV types one may use, but we specifically use k-fold CV. k-fold CV involves following five steps:

1. Partition a dataset into k subsets with equal cardinality.
2. Select a subset as a test set, and assign the remaining $k-1$ subsets as training sets.
3. Calculate an evaluation score using the previously identified testing and training sets.

4. Repeat 2. and 3. until all k subsets have been evaluated as a test set.
5. Average the k evaluation scores.

The value $k = 10$ has been selected as it is widely considered a good starting point. Furthermore, it is logical to partition our datasets into 10-folds because the embedding levels progressively increment by 10%.

With the help of CV, we are able to select the best polynomial degree that fits our dataset. However, one cannot expect to obtain a completely straight line in the horizontal section because of the polynomial regression. In order to detect the trend transition under such conditions, we use a threshold approach. We define the threshold as the maximum stationary value before the automatic detector flags more than one embedding level as a trend transition point. The threshold could be set at a 5% increment above an initial T-entropy value. However, this is not a reliable approach because the initial T-entropy value is affected by the initial embedding level. For instance, the T-entropy values for the 40% and 70% embedding level datasets range from 0.84 to 1.93 bits/byte (~230% increase) and 1.39 to 2.01 bits/byte (~145% increase), respectively. It thus seems more appropriate to use the overall percentage change in the calculation. The threshold is thus defined as:

$$T = \left(\frac{d_l}{d_f} 0.01\alpha + 1\right) d_f, \tag{5.5}$$

where α is the threshold percentage, and d_f and d_l are the first and last data points in the fitted curve, respectively.

Furthermore, the detector also needs to identify the 100% embedding level where the trend transition is not observable. The overall percentage change method, as seen from Equation 5.5, may once again be used to address this problem. That is, if the overall percentage change from start to end is less than 5%, we assume 100% embedding.

Figure 5.10 shows a performance analysis of the automatic detector using the aforementioned settings with $\alpha = 5\%$. Testing involved 100 datasets for each of the embedding capacities. As Figure 5.10 shows, the median values usually exceed the initial embed-

ding level. This is caused by the 5% threshold. It might appear as though subtracting the threshold value from the results might yield better results, but it is not a generic solution. A compact box indicates a more accurate detector than a tall box. Overall, the box plot range varies between the initial embedding levels, but the range is usually within 20%. Without the whiskers, this would bring down the range to $\approx 10\%$, but this only considers 50% of the datasets.

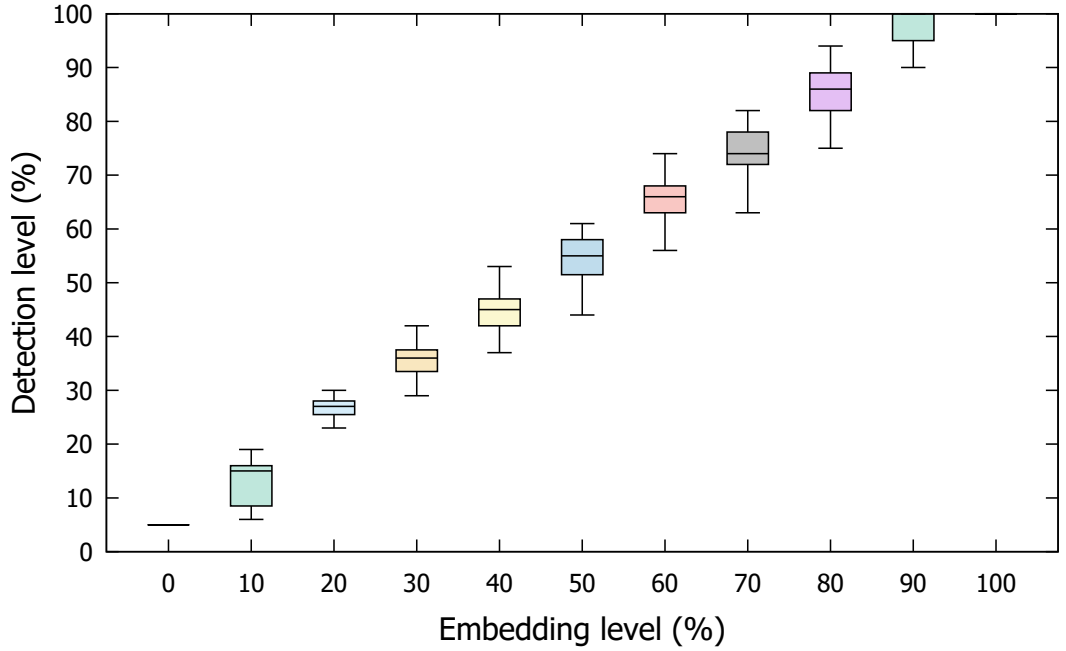


Figure 5.10: Performance analysis of an automatic detector based on polynomial regression with the $\alpha = 5\%$ threshold from Equation 5.5. Each box consists of 100 datasets for which the optimal degrees have been dynamically selected using 10-fold CV. The detector usually overestimates the embedding level because of the threshold.

5.7 Degree 1 residual approach

In RA section, Figure 5.9 (a), we criticised degree 1 in the goodness of fit test. A degree 1 polynomial is regarded as a linear regression, so it is of no surprise that our datasets will not fit well. A degree 1 residual plot, however, is special in its own right. As

Figure 5.9 (a) shows, connecting the dotted points results in a V-shaped curve, whose minimum marks the trend transition point, here shown for an initial embedding level of 40%. Appendix B.2 evidently illustrates that the V-shaped curve is not coincidental, and it can be observed from any of the 40% dataset. Figure 5.11 explains this phenomenon for the noise-free case where the stationary and increasing sections are both straight and the minimum residual occurs at the trend transition point at 40%.

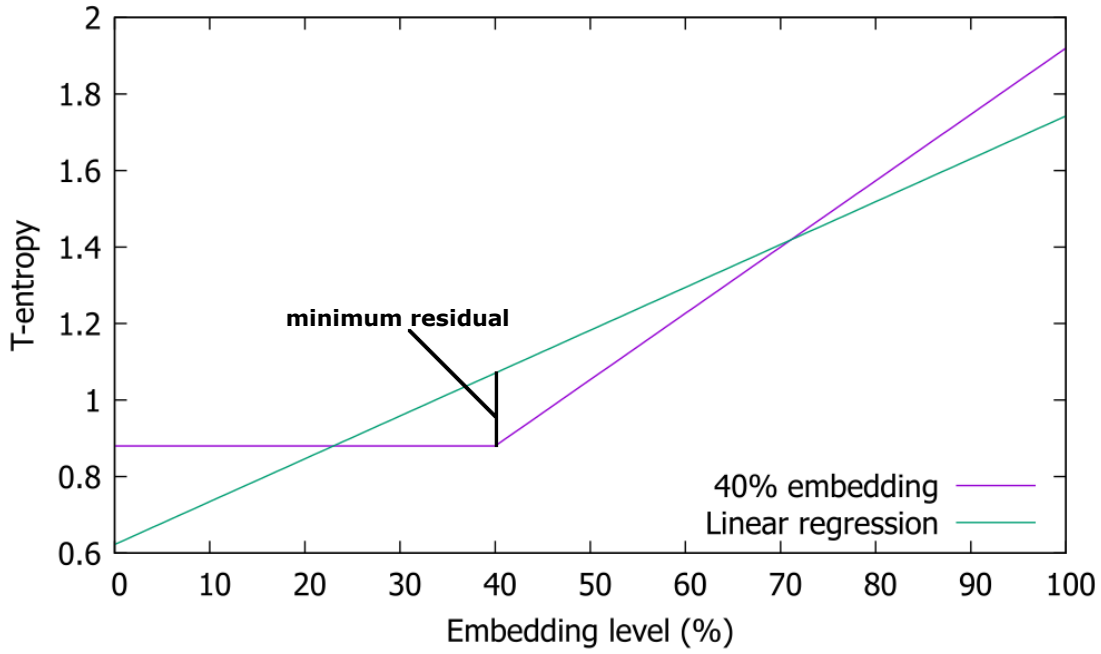


Figure 5.11: An ideal, noise-free dataset showing a 40% embedding level with straight linear and increasing sections. The green line shows the linear regression of the dataset. If one calculates residuals for all the data points, the trend transition point is the minimum residual.

We tested every dataset under the assumption that this property holds for all of the datasets. Figure 5.12 illustrates the results in box plots. The figure shows that selecting minimum residual values appears to supply relatively accurate trend transition points for most of the embedding capacities. This technique only fails for nearly unutilised or almost fully utilised channels. In the figure, these are the embedding levels 0%, 10%, 90% and 100%. That is, the V-shaped curves are not present in these embedding levels

(see Appendix B.3). The median values in the box plots almost align with the initial embedding levels. Moreover, the ranges of the box plots are comparatively shorter than those of the regression method in Figure 5.10.

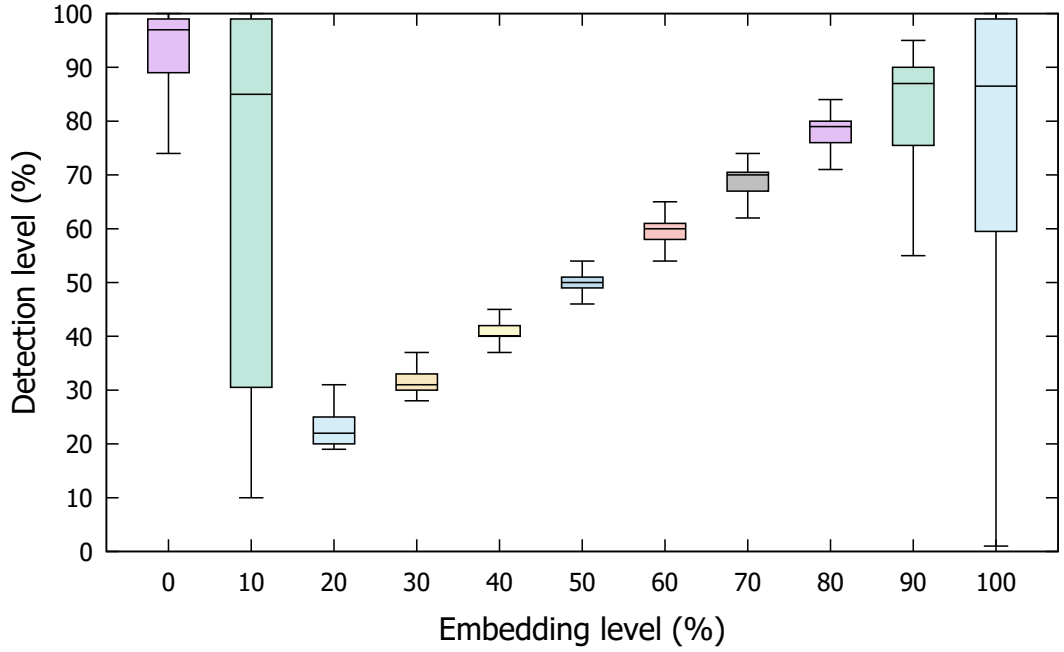


Figure 5.12: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots, yielding promising results for all but very small and very large embedding levels.

A fallback tactic can be used to mitigate the poor performance in estimating very low and very high embedding levels. The ranges of residual plots for the 0%, 10%, 90% and 100%embedding levels are shorter than other embedding levels (see Appendix B.3). When a short-range plot is detected, the detection method can be switched to the k-fold CV. This fallback tactic may be regarded as the combination of k-fold CV and degree 1 residual approaches because four-elevenths of the embedding levels rely on the k-fold CV detection. Figure 5.13 illustrates this. This approach produces more promising results, but at the cost of increased computation time.

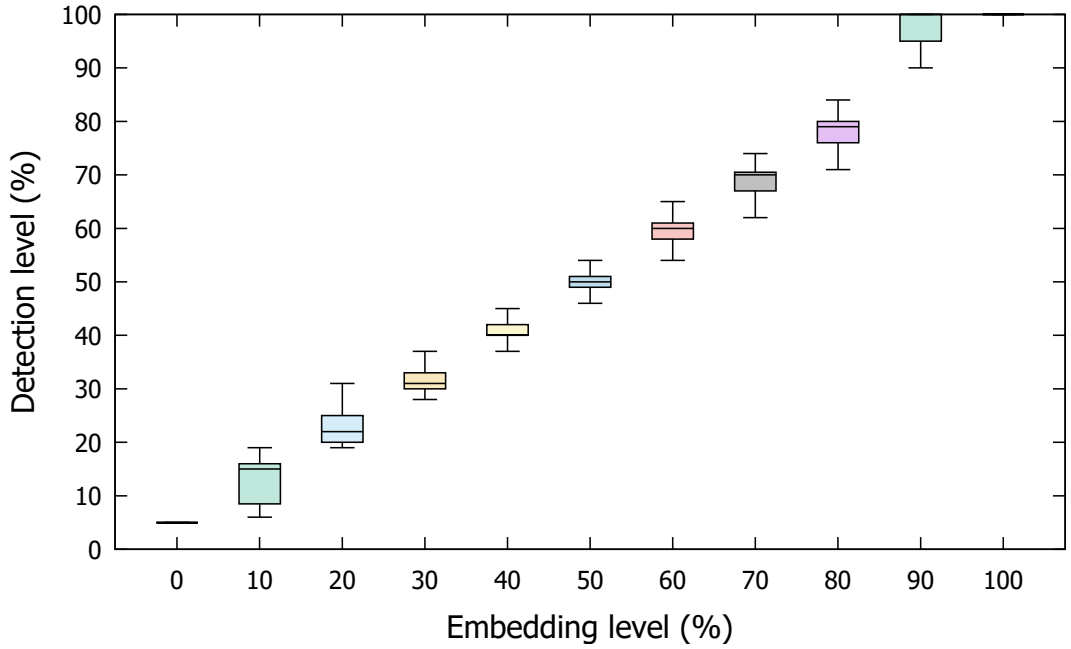


Figure 5.13: A hybrid technique that combines k-fold CV and degree 1 residual approaches. The technique produces more reliable results than the residual approach because the bottom and top 20% are remedied.

5.8 Automatic detection of the (equal spacing, equal spacing) tuple

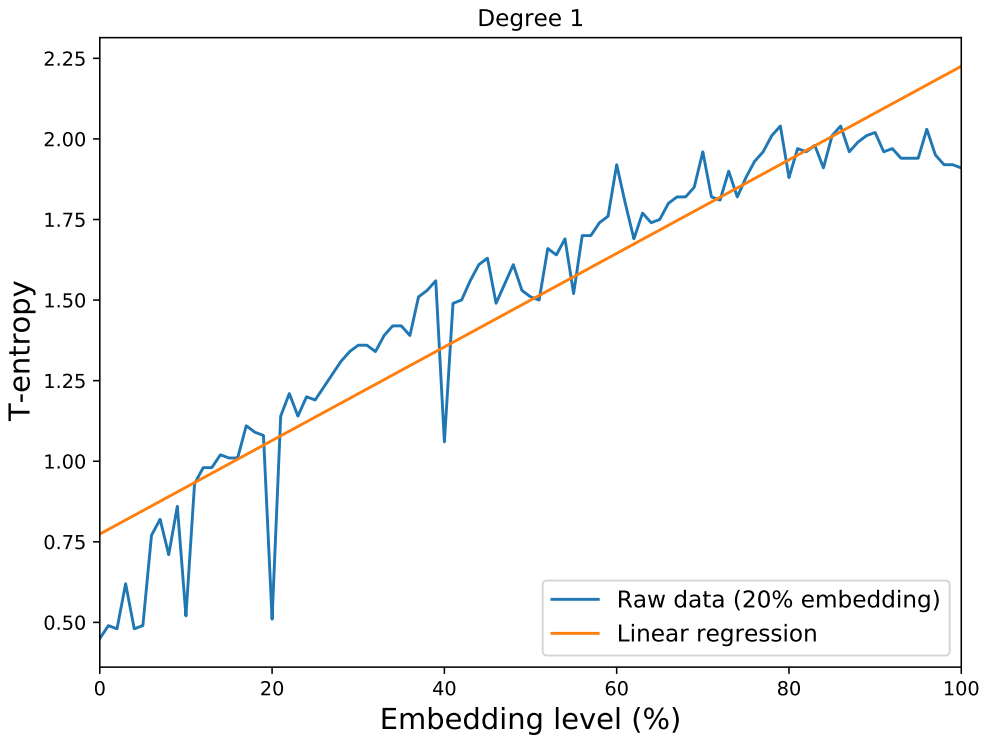


Figure 5.14: Linear regression on a 20% embedded (equal spacing, equal spacing) dataset. Drawing a residual plot may reveal that the minimum residual marks the deepest dip.

As discussed in Section 5.1, one of the dips in (equal spacing, equal spacing) tuple graph indicates the source embedding capacity. One can, however, misjudge an embedding capacity because there are multiple dips. For example, the noticeable dips of the 20% source embedding are at 10%, 20%, and 40% embedding levels. Upon closer inspection on the datasets, the deepest dip, i.e., a point that plummets and soars up the most, indicates the correct source embedding capacity.

We employ the same residual plot technique we had used for the (sequential, sequential) tuple. We hypothesise that the linear regression line will be drawn around the main curvature, therefore, the minimum residual will be the deepest dip. Figure 5.14 illustrates this on a 20% embedded dataset where the linear regression line is drawn around the main curvature. Figure 5.15 shows residual plot of Figure 5.14. Appendix B.4 shares more 20% datasets, illustrating this is not a coincidental behaviour.

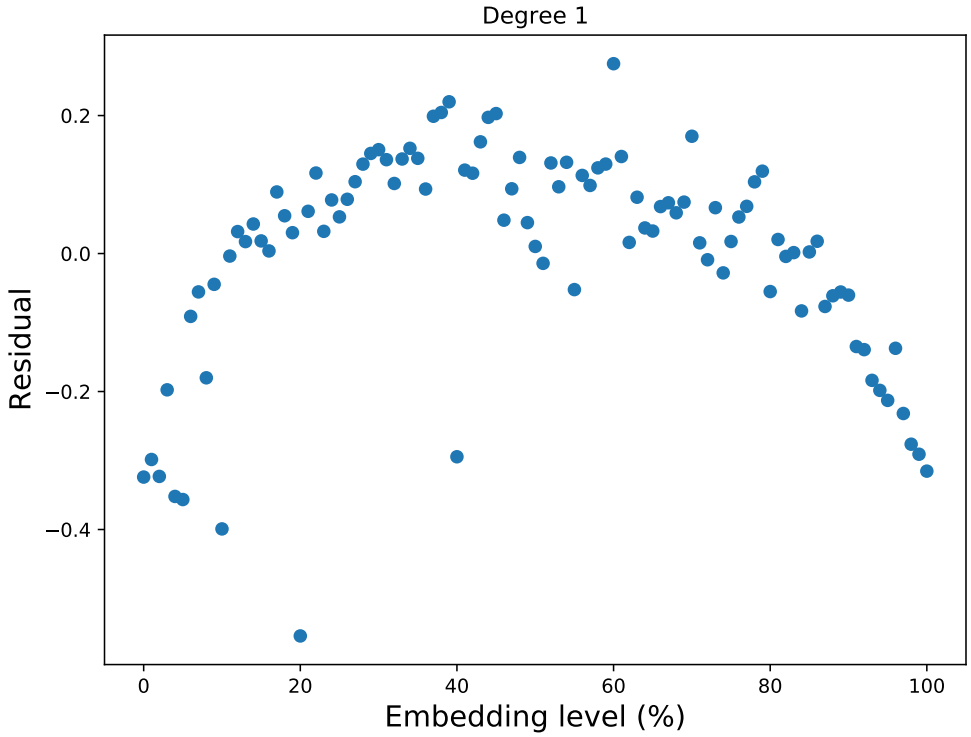


Figure 5.15: Degree 1 residual plot of Figure 5.14. This plot would be regarded as a bad fit because the data points show a pattern. For our purposes, at least, this plot shows a promising preliminary result because the minimum residual marks the source embedding level.

Figure 5.16 shows the results of using the degree 1 minimum residual approach. Satisfactory results may be seen from the figure beside 0%, 80%, 90% and 100%. In the case of 0% embedding, most of the minimum residual value comes out as 100%, whereas

it is complete opposite for 100% embedding (see Appendix B.4). A temporary fix of interpreting the 100% and 0% detected level as its opposite value can be considered here, but it is not a permanent fix. The results for 80% and 90% embeddings are unacceptable since the box plot ranges are too broad. On the other hand, the dips are accurately detected in all other embedding levels. The degree 1 residual plot technique also works here because of the same reason as described for the (sequential, sequential) tuple. Once a linear regression line is drawn, the deepest dip point is the minimum residual.

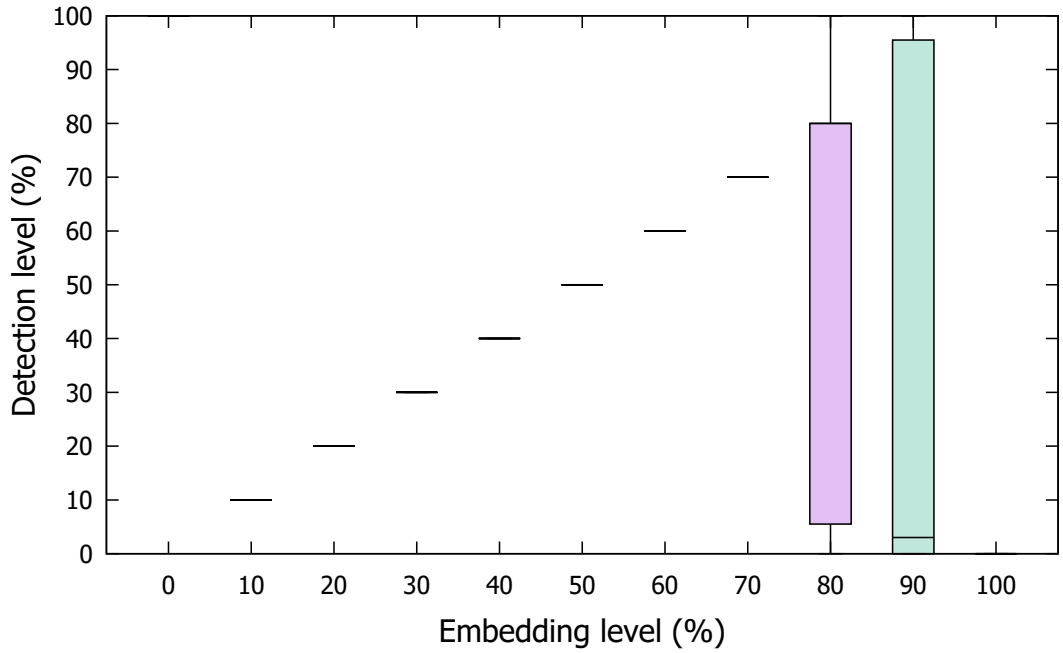


Figure 5.16: Performance analysis of an automatic detector that is based on residual plots. Minimum residual values from first degree polynomial residual plots are gathered in this approach. Promising result can be seen from all the embedding capacities except 0%, 80%, 90% and 100%.

5.9 Summary

This chapter discusses the journey of building an automatic detector capable of detecting a trend transition in a curve. In order to increase the detection accuracy, a number of dataset smoothing techniques were considered. We experimentally demonstrated that the traditional approach of using SMA is unsuitable here because it still retains noise. Furthermore, correct identification of a smoothing parameter proved problematic. Regression analysis was considered as an alternative, along with potential error metrics, which could be used to identify the polynomial degree. Our polynomial degree selector of choice is 10-fold CV, which meets all criteria. We then applied a threshold to the fitted curve to automatically detect the trend transition point. The detector's capacity estimation was generally off by $\approx 5\%$ due to the threshold. In terms of the detection accuracy, the range varied by 20%, which could be considered too broad. Linear regression was considered as an alternative as most of our datasets generate V-shaped residual plots under linear regression. This approach works for both (sequential, sequential) and (equal spacing, equal spacing) tuples, except for very low or very high embedding levels.

The next chapter experimentally evaluates the possibility of utilising other statistical metrics besides T-entropy in re-embedding steganalysis.

6

Re-embedding Steganalysis Using Other Statistical Metrics

Re-embedding steganalysis with T-entropy showed promising results in Chapter 4. The current chapter investigates the use of other statistical metrics besides T-entropy in re-embedding steganalysis. To this end, we experimentally evaluate only the matching source and trial embedding tuples and compliant mode datasets. This experimental setup showed the most promise for T-entropy and was robust against the noise therefore well suited for direct comparison between T-entropy and other metrics.

Steganalysis literature uses virgin and stego network packets as the two samples to utilise other statistical metrics [105]. We reiterate however that virgin data can be difficult to obtain, as already discussed in Section 2.5. Instead, we used the source and re-embedded network packets as our two samples. That is, we do not assume virgin data and use the source as it is.

The rest of the chapter is organised as follows: Sections 6.1–6.6 experimentally evaluate other metrics that appear in steganalysis literature. Section 6.7 summarises the findings. The final section summarises the chapter.

6.1 Welch's t-test

Welch's t-test statistically evaluates if two samples share a common mean in relation to the variation in the samples, and is defined as follows:

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}, \tag{6.1}$$

where $\overline{X_n}$, s_n^2 and N_n are the mean, variance and size of n, respectively. A positive t value indicates that the mean of the first sample is greater than that of the second sample. Whereas, a negative t value indicates the opposite.

In our re-embedding steganalysis, our null and alternative hypotheses are defined as:

$$\begin{aligned} H_0 &: \overline{X_1} = \overline{X_2} \Rightarrow E_1 = E_2 \\ H_1 &: \overline{X_1} \neq \overline{X_2} \Rightarrow E_1 \neq E_2, \end{aligned} \tag{6.2}$$

where $\overline{X_n}$ and E_n are the mean and embedding level of n, respectively.

Our null hypothesis is that when our two samples share a common mean, their embedding levels also match. We predict that the t-value of the Welch's t-test remain stationary until the re-embedding level is greater than that of the source.

Figures 6.1–6.3 show the results of using the t-value of the Welch's t-test in re-embedding steganalysis. The curves in the figures resemble those of the T-entropy, i.e., the trend transition and the deepest dip features are evident. There are however some noticeable differences: Firstly, t-values start at 0 for all the graphs because our

two samples are identical at 0% re-embedding level. Secondly, the graphs appear to be vertically inverted compared to those of the T-entropy, indicating that the mean of the second sample (re-embedded source) grows larger than that of the first sample (source embedding) as the embedding level increased. Swapping the two samples produces a graph similar to that of the T-entropy, but this carries no significance.

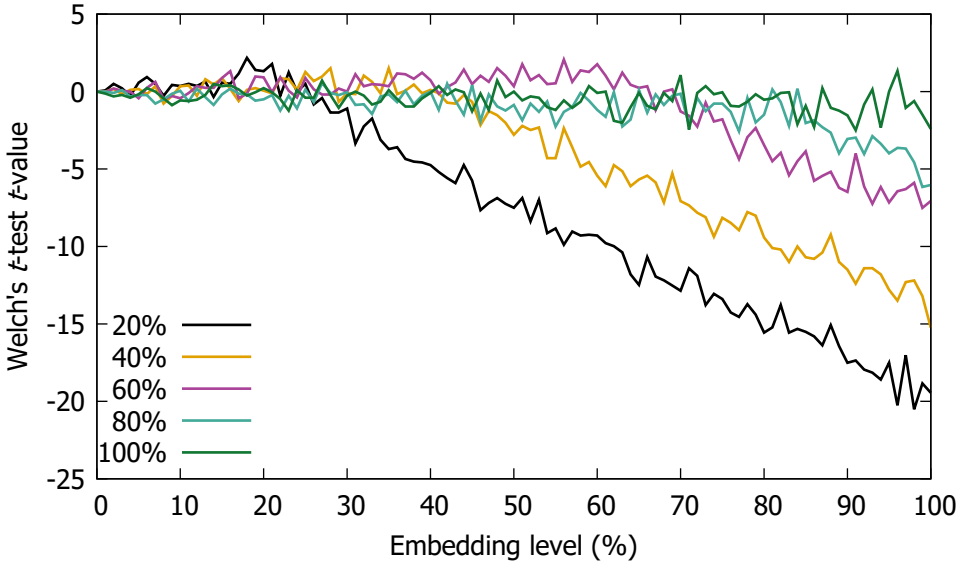


Figure 6.1: The effect of sequentially re-embedding a sequentially embedded source. The graphs in the figure resemble those of the T-entropy, except they are inverted.

One may, however, question that if the observation did not occur by a random chance, and this is where p-value comes helpful. The p-value refers to a predetermined probability distribution of the statistical test and provides the probability of the observed event. It is used as a piece of evidence to either reject or accept the null hypothesis. A high p-value favours the null hypothesis that the two samples share a common mean. Conversely, a low p-value favours the alternative hypothesis that the two samples do not share a common mean. A threshold value, or a significance level, of 5% is commonly used and is usually in decimal representation, e.g., 0.05.

Appendix C.1 illustrates the result of using the p-value. As may be seen from the

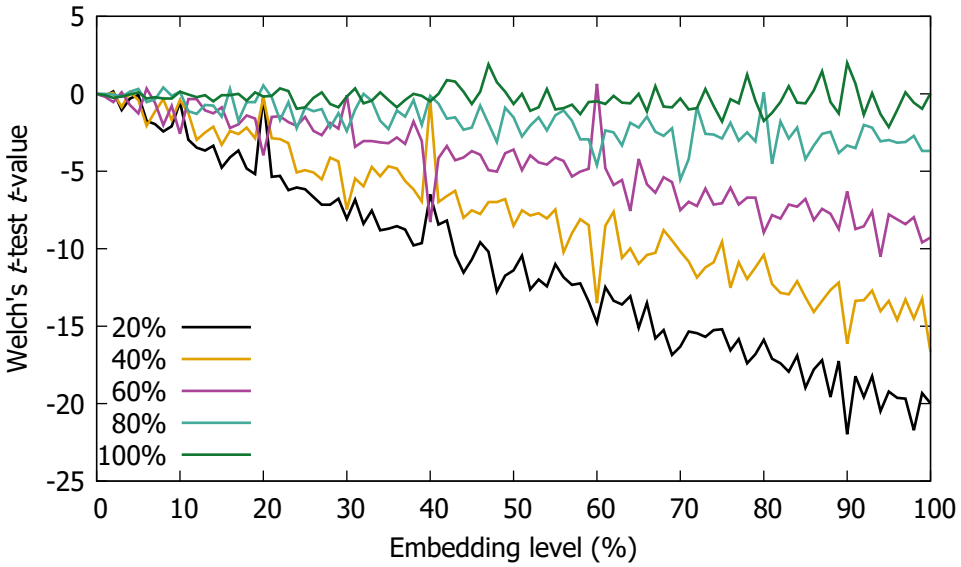


Figure 6.2: The effect of evenly re-embedding an evenly embedded source. The graphs in the figure resemble those of the T-entropy, except they are inverted.

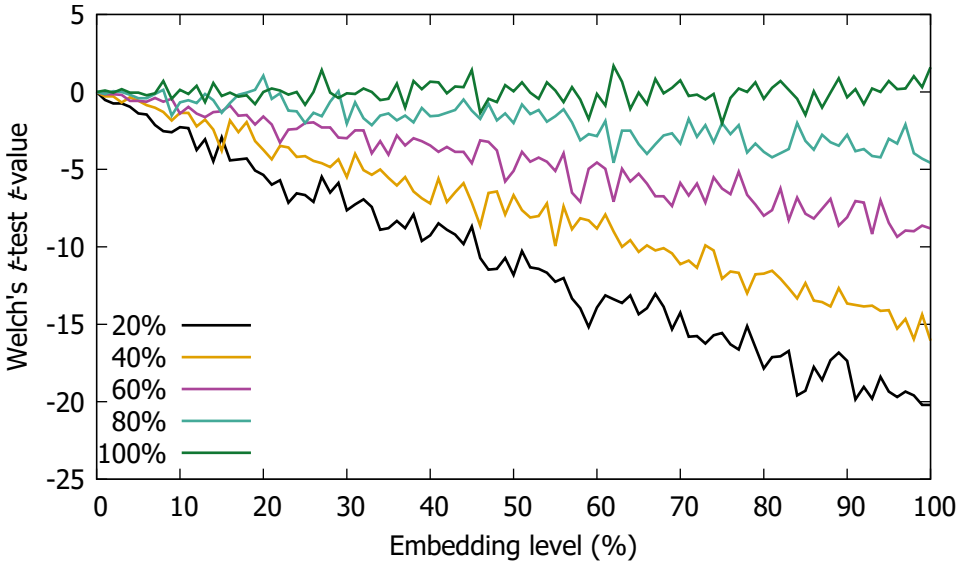


Figure 6.3: The effect of randomly re-embedding a randomly embedded source. The graphs in the figure resemble those of the T-entropy, except they are inverted (cf. Figure 4.13). The graphs show no significant trend.

figures, the p-values start at the peak value of 1 because the two samples are identical. The p-values fluctuate in all the embedding levels, and the highest peak in a graph does not necessarily indicate the source embedding level. However, the p-values generally stay above the 5% mark confidence level. The p-value results are therefore excluded from this point onward, and we focus on the test statistic values only.

Overall, the notable differences mentioned earlier about the re-embedding steganalysis figures do not prevent us from using the automatic detector developed in the previous chapter. Figures 6.4 and 6.5 show the automatic detection results on a hundred sequential and equal spacing embedding datasets, respectively. One caveat here is that the maximum residuals are gathered instead of the minimum residuals since the graphs are inverted.

As may be seen from Figure 6.4, the automatic detection accurately detects all the embedding capacities except 80%, 90%, and 100%. Furthermore, the ranges of the box plots on the other embedding capacities are shorter than those of the T-entropy (cf. Figure 5.12), implying that Welch's t-test may be better suited for re-embedding steganalysis in terms of its robustness.

As may be seen from Figure 6.5, the automatic detection accurately detects all the embedding capacities except 90% and 100%. Again, Welch's t-test performed better than T-entropy because it also detected the 80% embedding capacity which T-entropy was not able to.

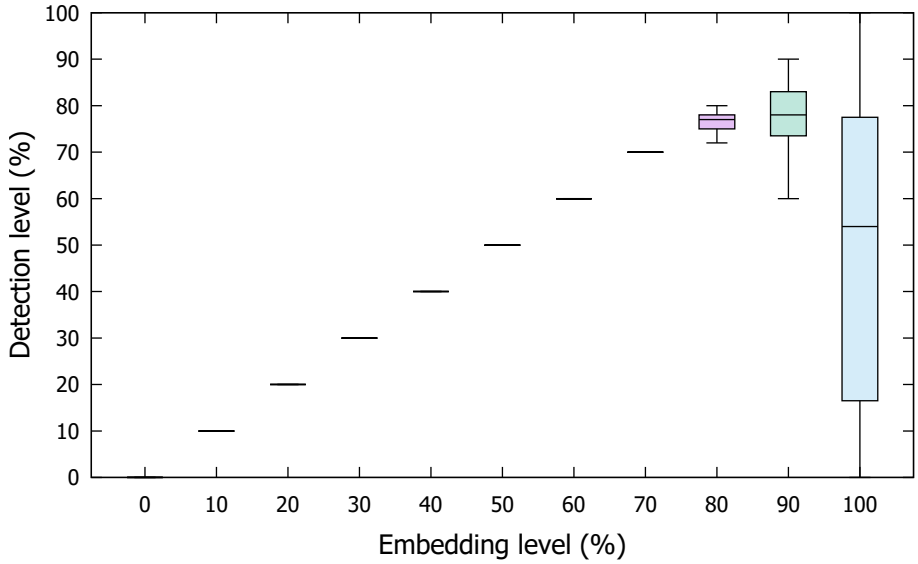


Figure 6.4: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of the sequentially re-embedded source. Promising results from all the embedding capacities except 80%, 90%, and 100%

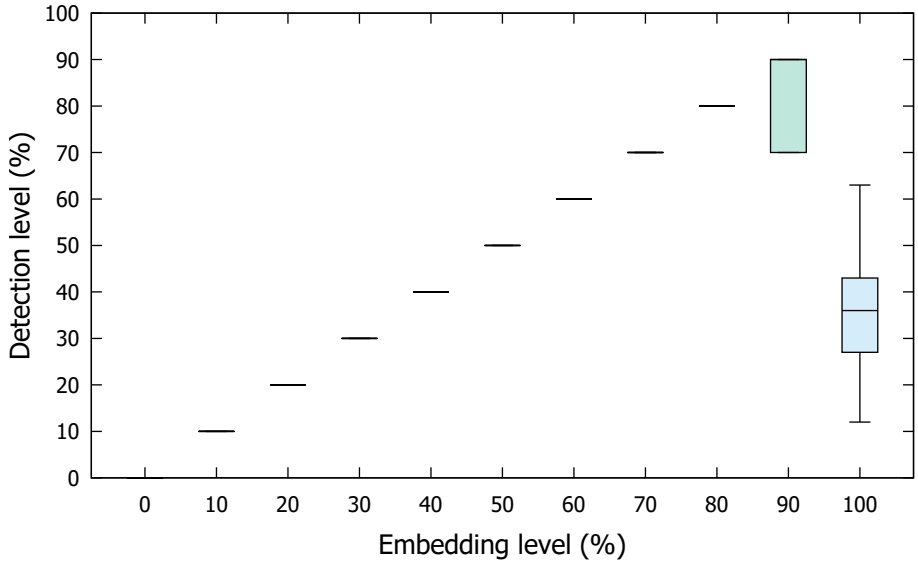


Figure 6.5: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of the evenly re-embedded source. Promising results from all the embedding capacities except 90% and 100%.

6.2 Chi-square test

Chi-square test statistically evaluates the independence between two samples. Unlike Welch's t-test, the chi-square test does not factor in the sample sizes and sample means in its calculation (cf. Equation 6.1). The chi-square test is defined as follows:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}, \tag{6.3}$$

where O_i is the observed value and E_i is the expected value at data point i.

Figures 6.6–6.8 illustrate the results of using the chi-square test in re-embedding steganalysis. Overall, we observe similar characteristics here as the T-entropy. There are however a couple of noticeable differences: Firstly, the graphs appear to reflected over the y-axis. Secondly, the curves in the graphs flatten as the source embedding level increases. The trend transition characteristic almost disappears from the 40% embedding level for the sequential embedding scheme in Figure 6.6. For equal spacing scheme, the deepest dip appears to disappear from the 60% mark in Figure 6.7.

Figures 6.9 and 6.10 show the results of using the automatic detection on a hundred sequential and equal spacing embedding datasets, respectively. Figure 6.9 illustrates that chi-square test fails to detect most embedding levels, indicating that chi-square test is not compatible with re-embedding steganalysis. The flattening characteristic in Figure 6.6 must have contributed to this phenomenon. Figure 6.10 shows a similar outcome where it detected only half of the embedding levels due to the flattening characteristic.

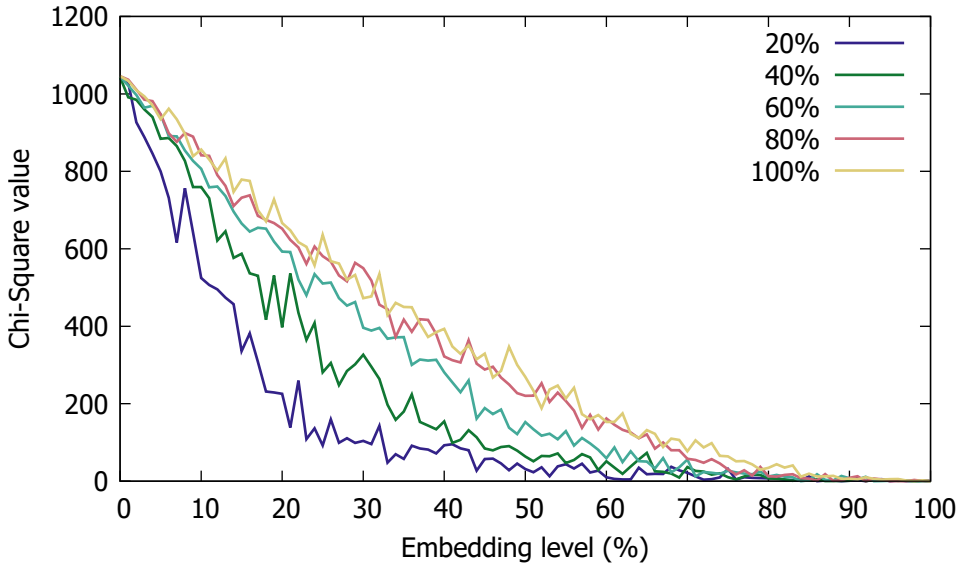


Figure 6.6: The effect of sequentially re-embedding a sequentially embedded source. The curves in the graphs resemble those for T-entropy, except that they are reflected over the y-axis. Furthermore, the graphs flatten as the source embedding level increases, losing the trend transition that is vital in re-embedding steganalysis.

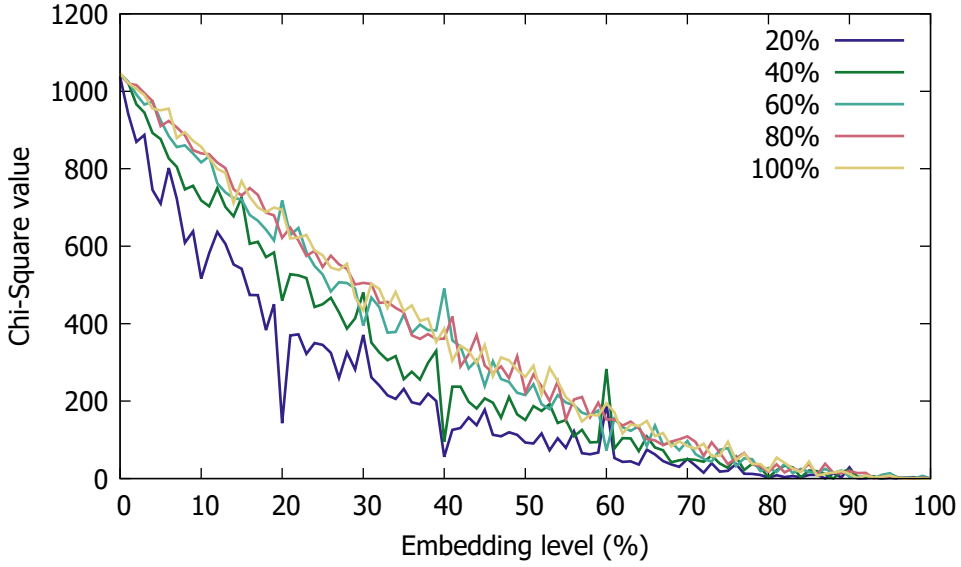


Figure 6.7: The effect of evenly re-embedding an evenly embedded source. The curves in the graphs resemble those for T-entropy. The graphs also flatten with the increasing source embedding, again indiciating that the chi-square test may not be compatible with re-embedding steganalysis.

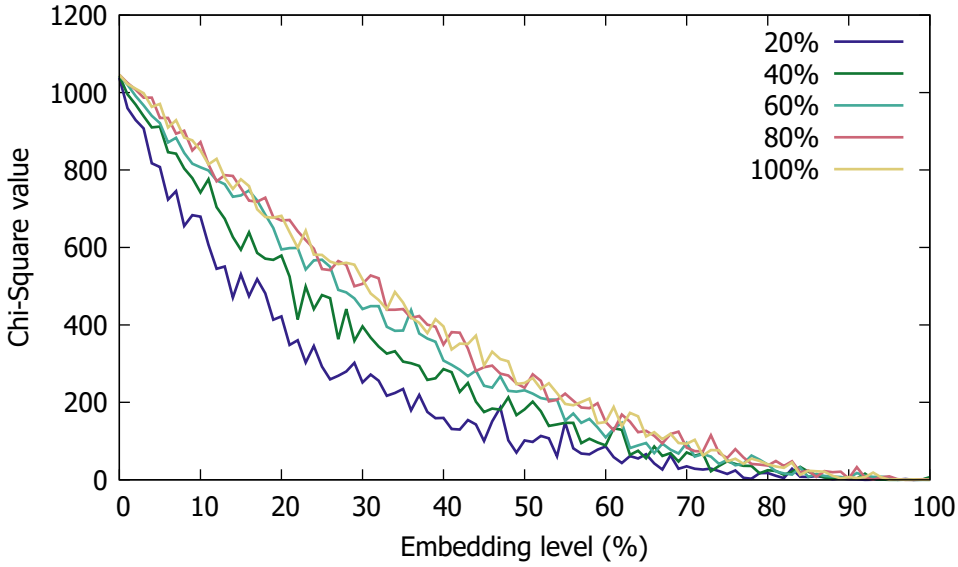


Figure 6.8: The effect of randomly re-embedding a randomly embedded source. The graphs show no significant trend between the source and the re-embedded part.

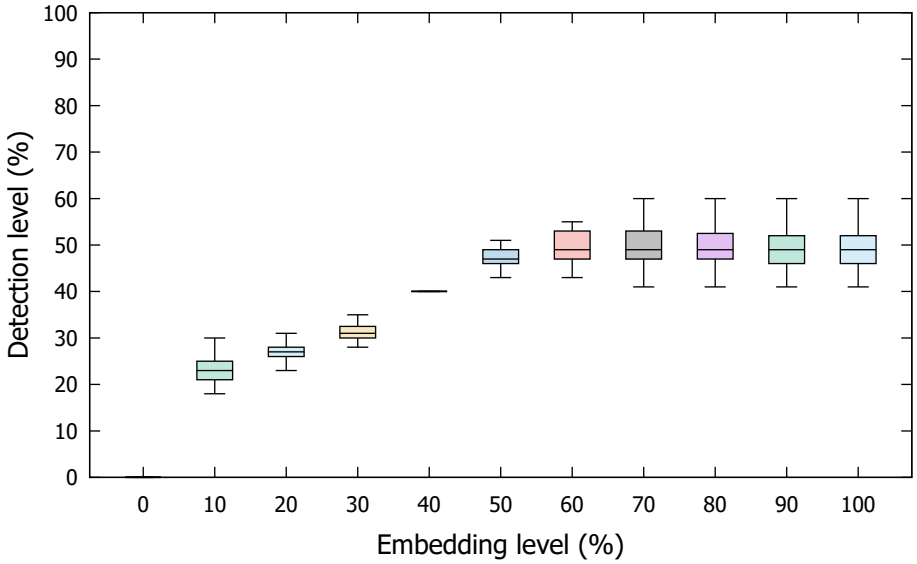


Figure 6.9: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of sequentially re-embedded source. Overall, chi-square is not a suitable metric to be used in re-embedding steganalysis because it detects only 0%, 30% and 40% embedding capacities. As illustrated in Figure 6.6, the curves flatten too quickly, losing the trend.

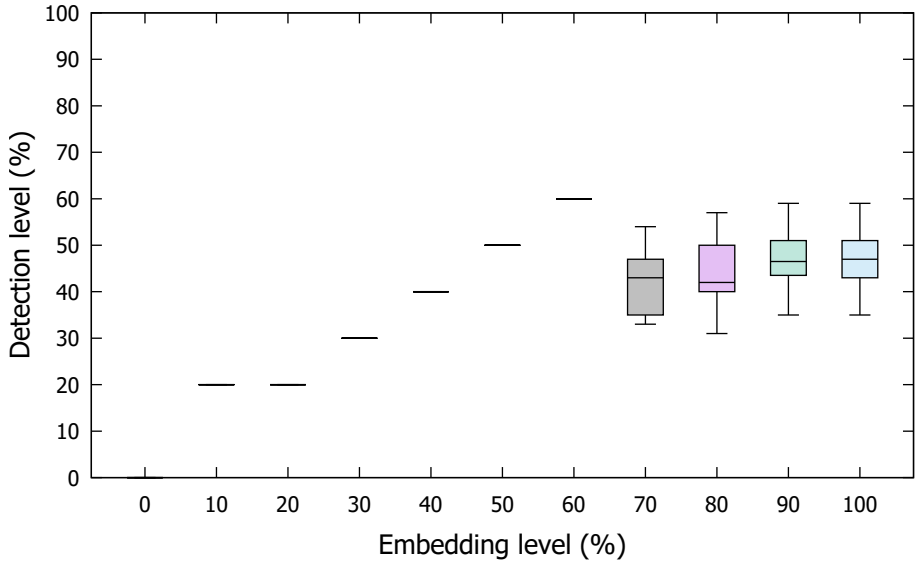


Figure 6.10: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of evenly re-embedded source. Overall, chi-square is not compatible with re-embedding steganalysis because it detects only 20%, 30%, 40%, 50% and 60% embedding capacities. As illustrated in Figure 6.7, the flattening characteristic destroys the necessary detection feature.

6.3 Kolmogorov-Smirnov test

The two-sample Kolmogorov-Smirnov (K-S) test measures the absolute supremum distance between the two samples' cumulative distribution functions (CDF). Low K-S test value means the two samples are likely from the same distribution, and vice versa. The two-sample K-S test is defined as follows:

$$K\text{-}S = \sup_x |F_1(x) - F_2(x)|, \tag{6.4}$$

where sup_x is supremum of the distances.

Figures 6.11–6.13 show the results, and the similar characteristics as that of the Welch's t-test are evident here: Firstly, the detection characteristics are present. Secondly, the initial K-S test values are 0.

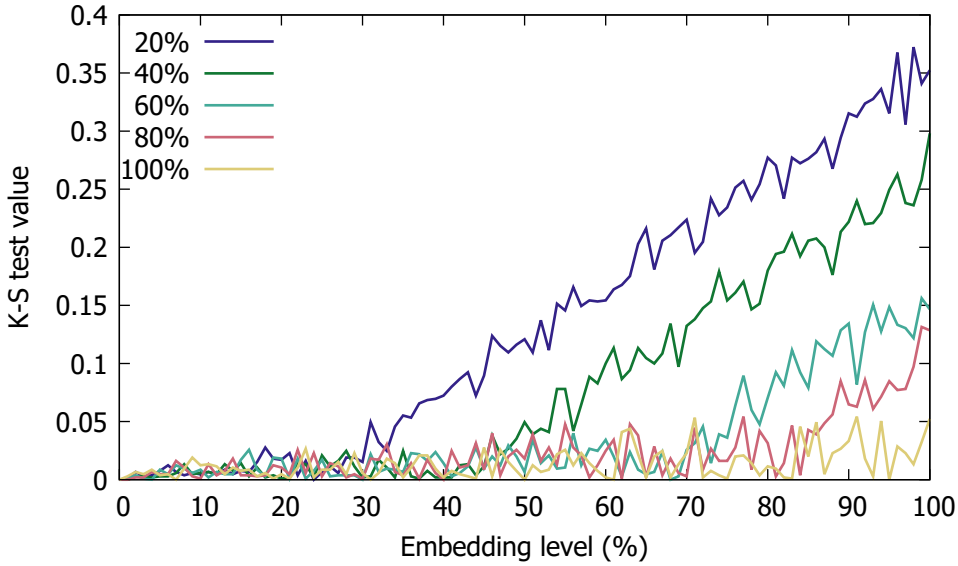


Figure 6.11: The effect of sequentially re-embedding a sequentially embedded source. The curves in the graphs resemble that of the T-entropy, except the initial test value of 0.

Figures 6.14 and 6.15 show the results of using the automatic detection on a hundred sequential and equal spacing embedding datasets, respectively. Figure 6.14 illustrates

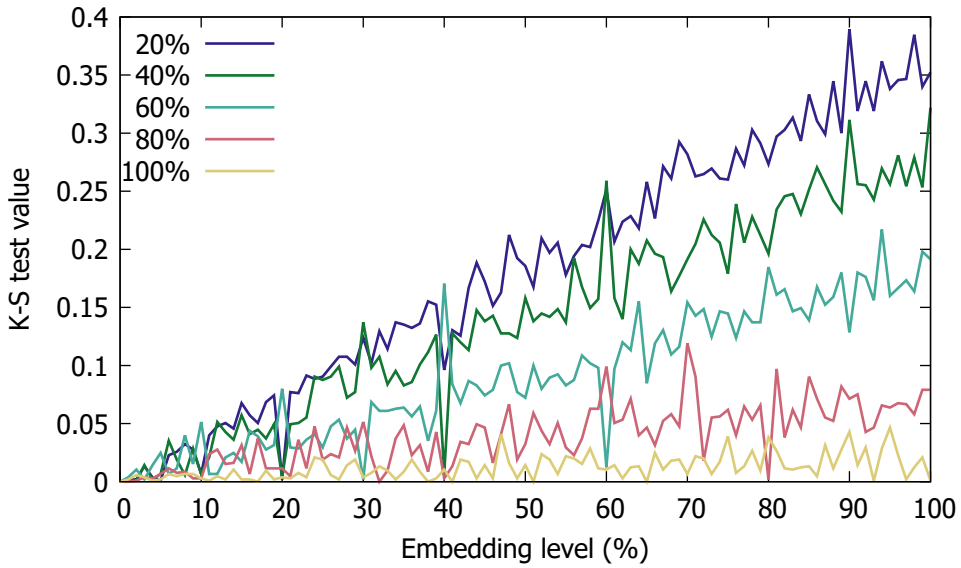


Figure 6.12: The effect of evenly re-embedding an evenly embedded source. The curves in the graphs resemble that of the T-entropy, except the initial test value of 0.

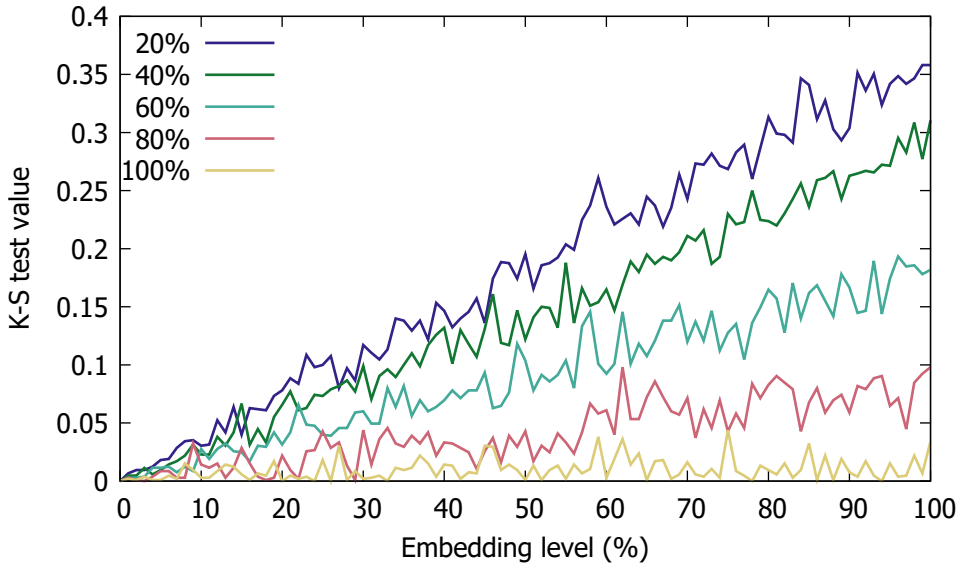


Figure 6.13: The effect of randomly re-embedding a randomly embedded source. The graphs show no significant trend, indicating automatic detection would not work.

that K-S detects all embedding levels except very low and very high embedding levels. Figure 6.15 shows similar behaviour. These results conclusively suggest high compatibility of K-S test in re-embedding steganalysis.

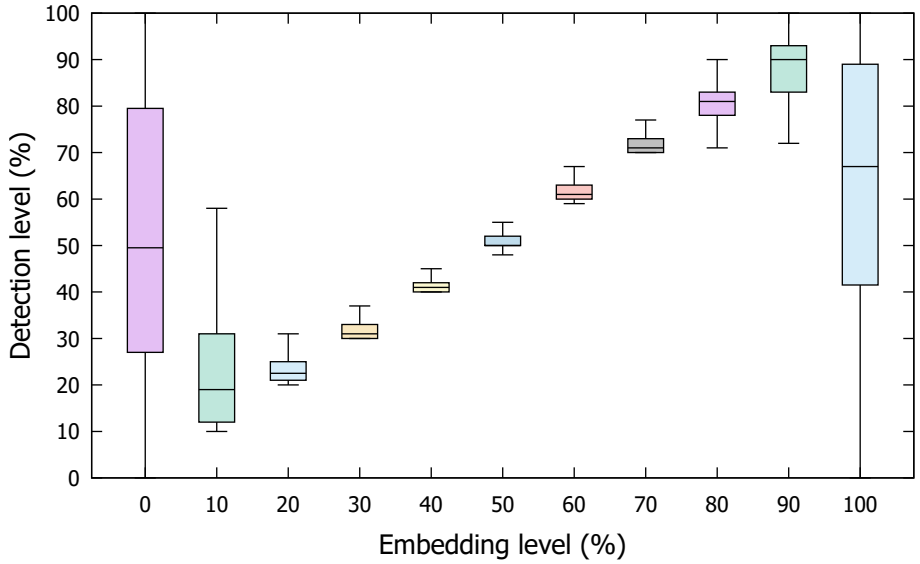


Figure 6.14: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of sequentially re-embedded source. Promising results can be seen from all the embedding levels except very small and very large embedding levels.

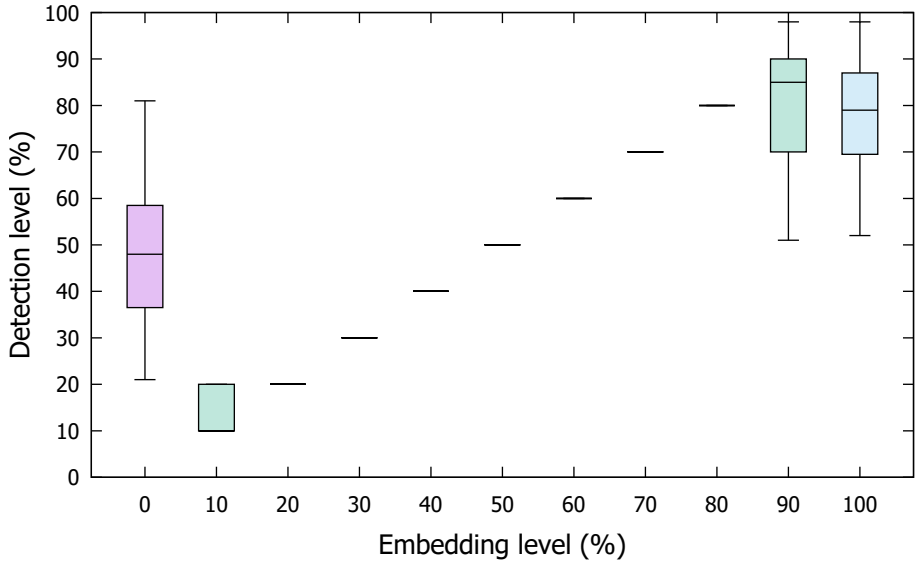


Figure 6.15: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of evenly re-embedded source. Promising results can be seen from all the embedding levels except very small and very large embedding levels.

6.4 Shannon entropy

Shannon entropy measures uncertainty by factoring the probability of symbol appearance [106]. A low entropy means highly patterned data, and vice versa. Shannon entropy is defined as follows:

$$H(X) = -\sum_{x} p(x) \log_2 p(x), \tag{6.5}$$

where $p(x)$ is the probability of symbol appearance.

Figures 6.16–6.18 show the results, and the graphs look similar to those of the T-entropy.

Figures 6.19 and 6.20 show the results of using the automatic detection on a hundred sequential and equal spacing embedding datasets, respectively. As may be seen from the figures, the detection accuracy is very accurate. The technique only fails at detecting

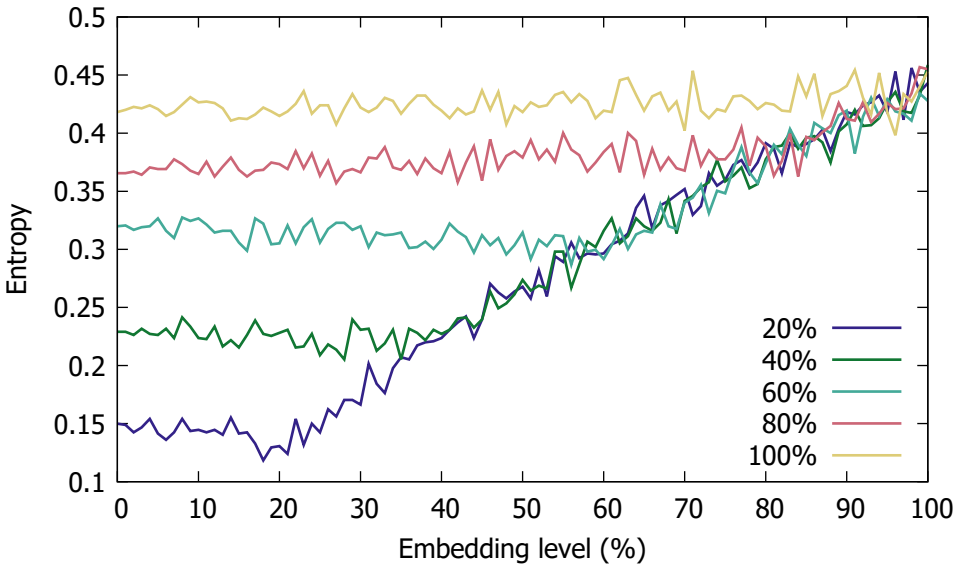


Figure 6.16: The effect of sequentially re-embedding a sequentially embedded source. The curves in the graphs resemble that of the T-entropy.

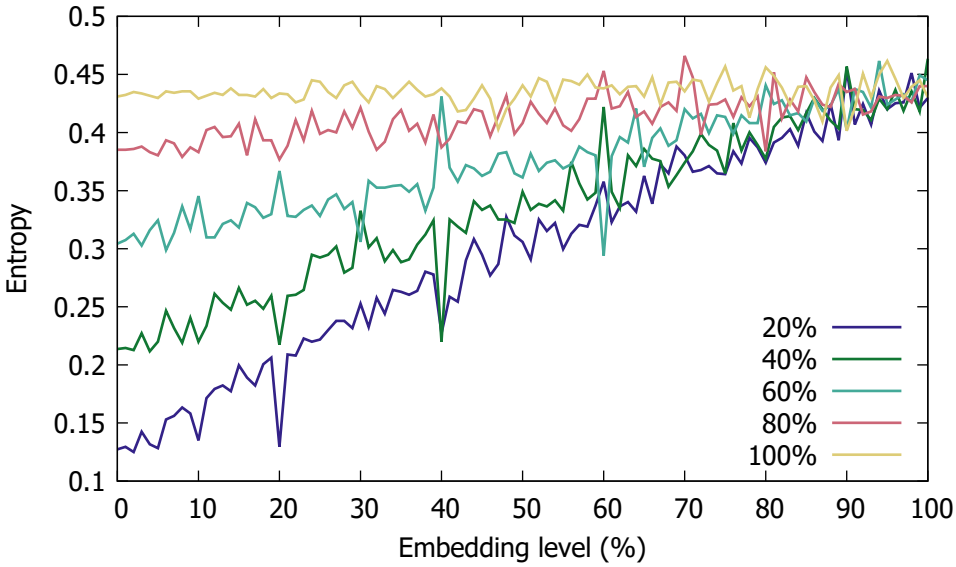


Figure 6.17: The effect of evenly re-embedding an evenly embedded source. The curves in the graphs resemble that of the T-entropy.

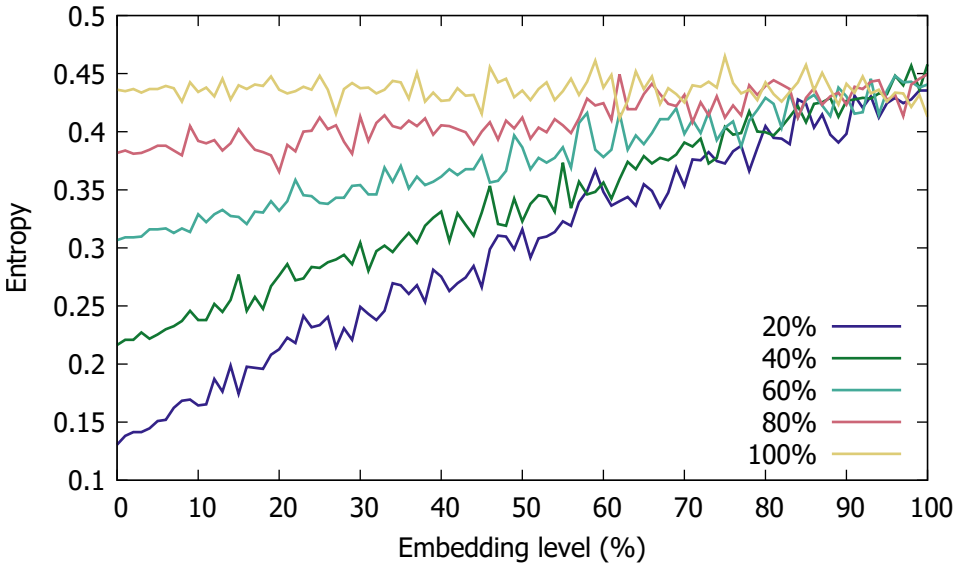


Figure 6.18: The effect of randomly re-embedding a randomly embedded source. The graphs show no significant trend.

very high embedding levels.

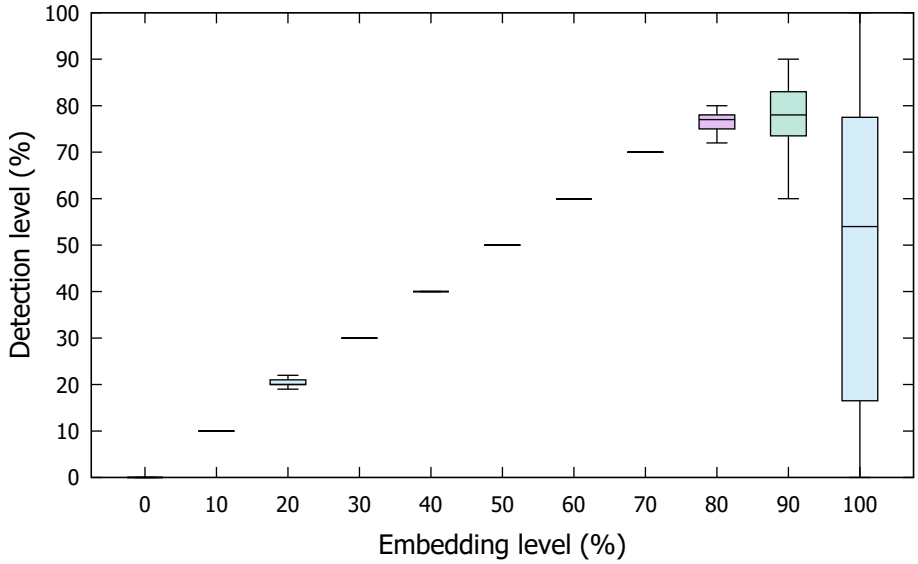


Figure 6.19: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of sequentially re-embedded source. Promising results can be seen from all the embedding capacities except 90% and 100%.

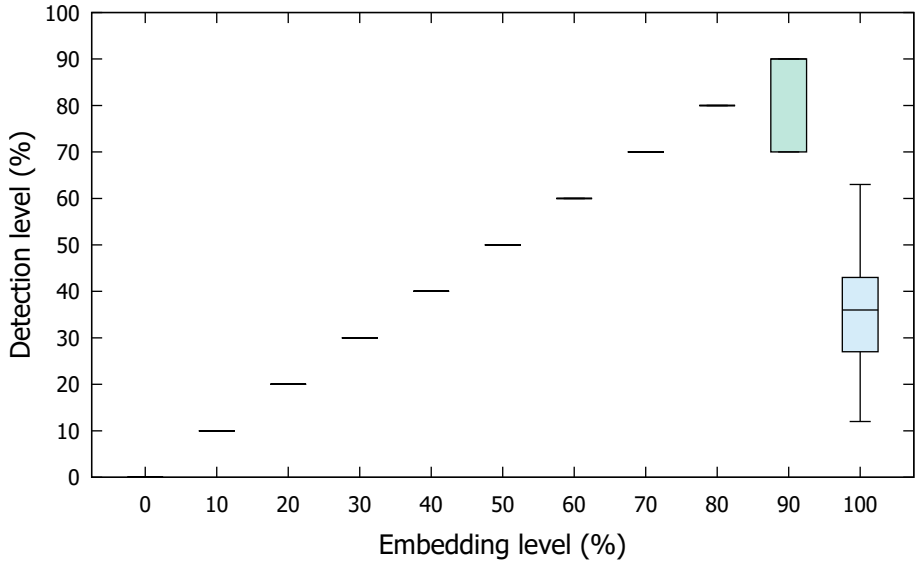


Figure 6.20: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of evenly re-embedded source. Promising results can be seen from all the embedding capacities except 90% and 100%.

6.5 Kullback-Leibler divergence

Kullbakc-Leibler divergence (KL), also known as relative entropy, measures the divergence of two probability distributions P and Q, where the expectation is based on the P. The greater the value, the greater the divergence exists between the distributions, and vice versa. KL is defined as follows:

$$D_{KL}(P\|Q) = \sum_{x \in X} P(x) \log_2 \left(\frac{P(x)}{Q(x)} \right). \tag{6.6}$$

Figures 6.21–6.23 show the results, and similar behaviour as K-S test has been observed here, i.e., the detection characteristics are present and the initial KL values are 0.

Figures 6.24–6.25 show poor performances in detecting both the sequential and equal spacing. In sequential and equal spacing embeddings, only two and four embedding levels are detected, respectively.

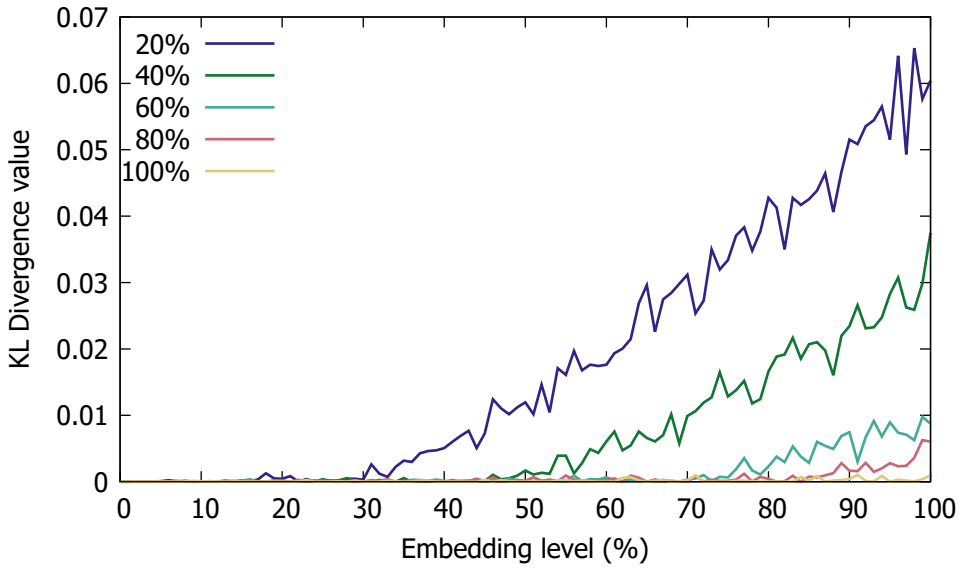


Figure 6.21: The effect of sequentially re-embedding a sequentially embedded source. The curves in the graphs resemble that of the K-S test.

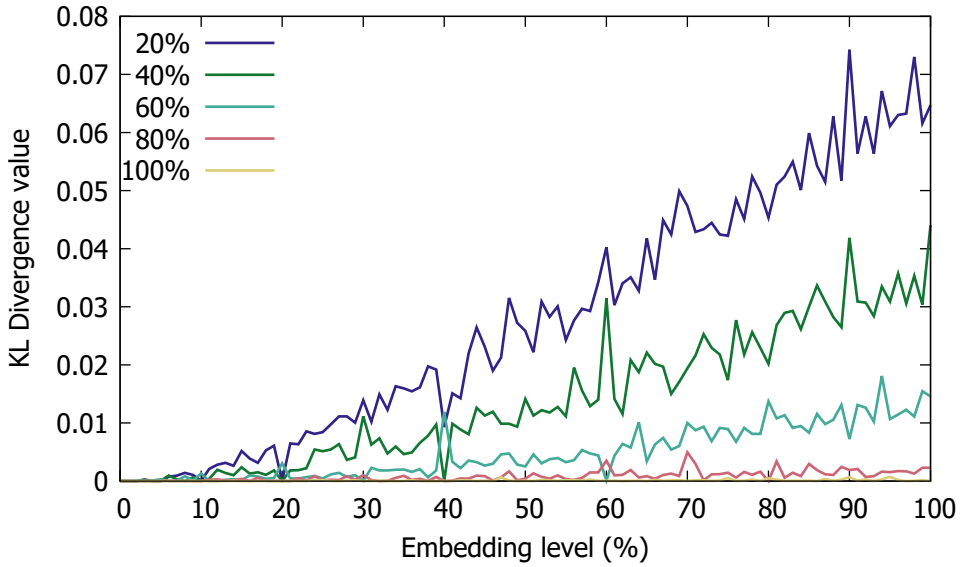


Figure 6.22: The effect of evenly re-embedding an evenly embedded source. The curves in the graphs resemble that of the K-S test.

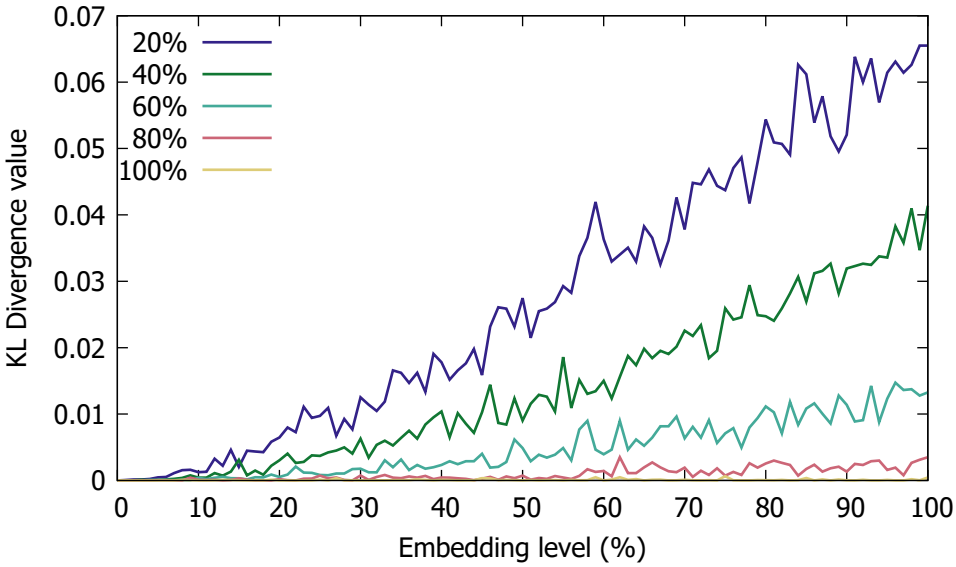


Figure 6.23: The effect of randomly re-embedding a randomly embedded source. The curves in the graphs resemble that of the K-S.

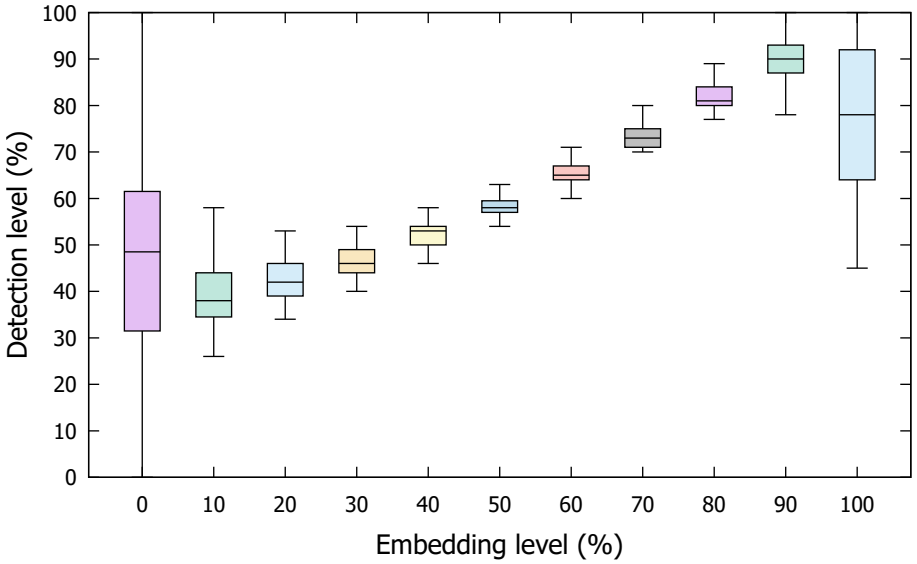


Figure 6.24: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of sequentially re-embedded source. Overall, K-L divergence is not a suitable metric to be used in re-embedding steganalysis because it detects only 50% to 80% embedding capacities.

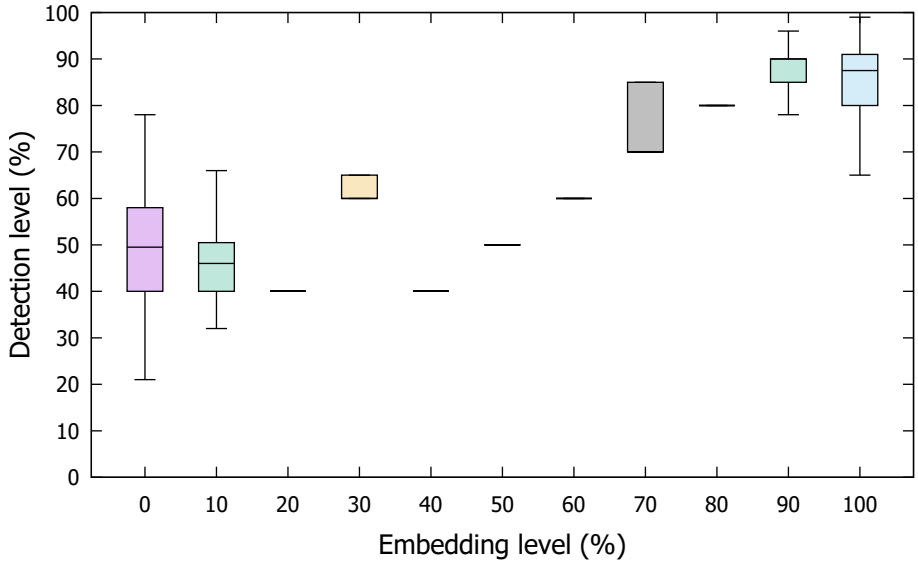


Figure 6.25: Performance analysis of an automatic detector that gathers minimum residual values from degree 1 polynomial residual plots of evenly re-embedded source. Overall, K-L divergence is not a suitable metric to be used in re-embedding steganalysis because it detects only 40%, 50%, 60% and 80% embedding capacities.

6.6 Autocorrelation function

Autocorrelation function (ACF) calculates data's correlation to itself by referring to its own delayed copy. ACF is defined as follows:

$$r_k = \frac{\sum_{t=k+1}^{T}(r_t - \bar{r})(r_{t-k} - \bar{r})}{\sum_{t=1}^{T}(r_t - \bar{r})^2}, \tag{6.7}$$

where k is the delay, and $\bar{r}$ is the mean of the dataset.

The delay is known as *lag*, and the function outputs a value that ranges from -1 (perfect negative correlation) to 1 (perfect positive correlation). Inventory stock management uses ACF, where stock control into the foreseeable future may be a crucial task. For example, winter coats are a seasonal item when the demands and sales volume may peak around the winter period. Calculating monthly ACF on the sales volume of winter coats

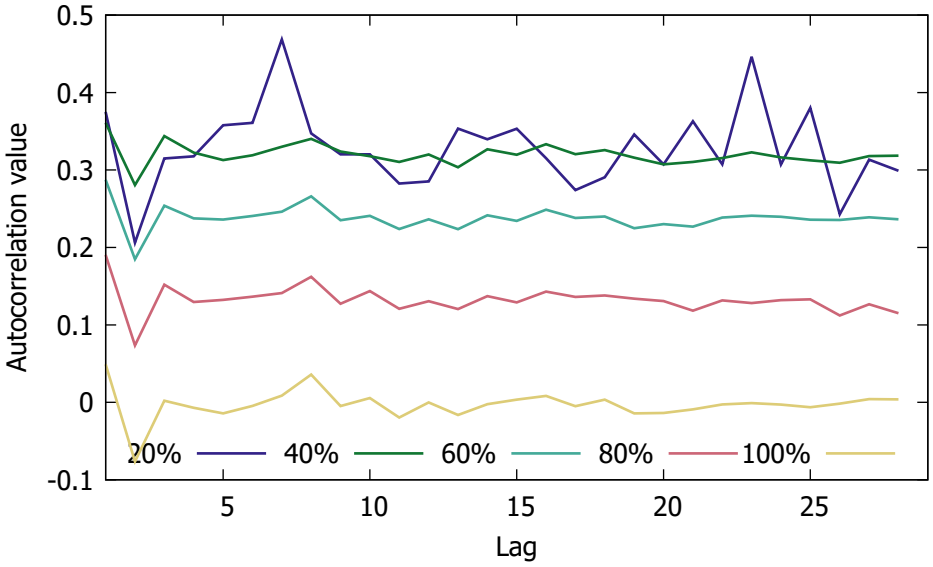


Figure 6.26: The effect of sequentially re-embedding a sequentially embedded source. The correlation becomes weaker (i.e., decrease in autocorrelation value) as the source embedding increases.

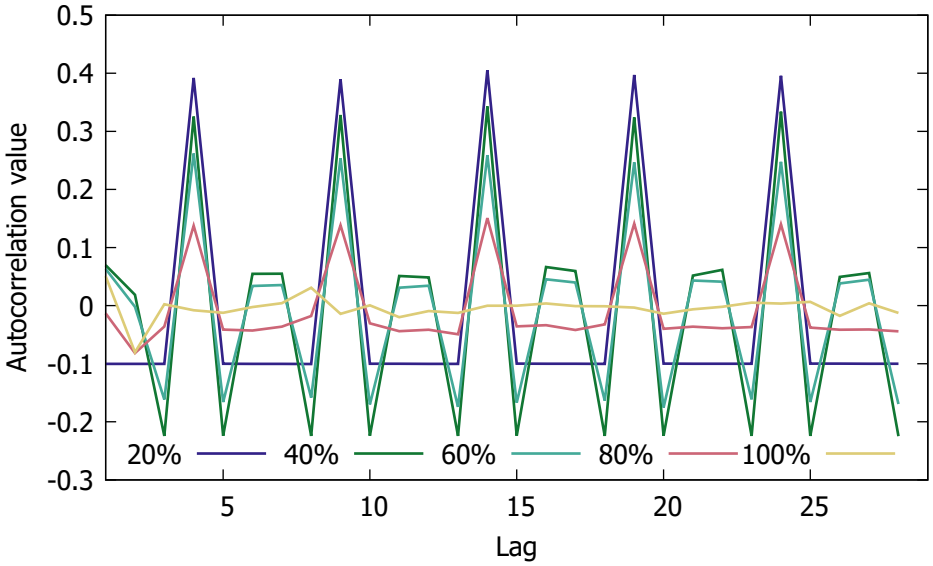


Figure 6.27: The effect of evenly re-embedding an evenly embedded source. Peaks can be observed from time lag multiples of 5, indicating autocorrelation can potentially detect steganography technique that spreads secret data which leaves a fixed number of gaps in-between.

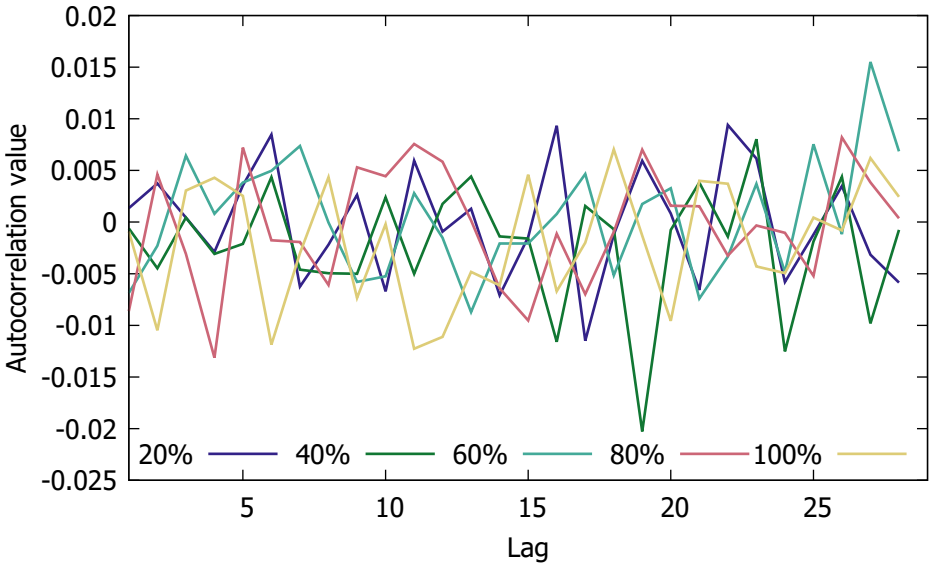


Figure 6.28: The effect of randomly re-embedding a randomly embedded source. No noticeable pattern can be seen from the graphs besides the vigorous fluctuations. Compared to Figures 6.26 and 6.27, the autocorrelation values are smaller. The values around 0 neither indicate the strong correlation nor the weak correlation in the dataset.

may reveal that a strong correlation exists with the lag of 6; every six months.

ACF shares a similarity with re-embedding steganalysis in that it does not rely on virgin data but only the source data. The metric, however, is not compatible with the re-embedding steganalysis because one cannot be fixated to a single lag value.

Figures 6.26, 6.27 and 6.28 illustrate the ACF on sequential, equal spacing and random embedding schemes, respectively. The time lag of 0, which yields the result of 1, has been omitted from the graphs to maintain the focus on the area of interest.

The sequential embedding figure shows an unfavourable steganalysis result. The figure shows correlation decreases as the source embedding increases. This is caused by a strong initial correlation. When nothing is embedded initially, i.e., all flags' fields are set as 000, ACF calculates a strong correlation between the flags. Once the embedding capacity fills up with secret data, the correlation weakens and worsens the more secret data are embedded. As may be seen from the figure, correlation generally drops after the

lag value of 1. The 20% source embedding fluctuates more vigorously when compared against other source embeddings, but this may not reveal anything significant when it is inspected on its own.

Curves soar and plummet in the equal spacing embedding at lag multiples of 5. Such a phenomenon may be explained by how equal spacing leaves gaps between its embedding. The number of gaps decreases as the embedding level increases, and the figure depicts this with the shorter peaks. The peaks and plummets may be helpful to determine the presence of equal spacing embedding but it may not help quantify the amount of the steganograms.

The curves in the random embedding figure fluctuate vigorously without showing any trend. The graphs span much narrower than the other two tuples, indicating weak correlation in the flags.

6.7 Discussion

This section summarises and compares the performance of the evaluated metrics in their robustness and time complexity. The robustness defines the accuracy of the automatic detector with an accompanied metric, whereas the time complexity describes the runtime of an algorithm (or an accompanied metric) using the Big O notation [107].

A constructive analysis of an algorithm is not possible without first considering a *cost model*. The cost model defines which of the operations in the algorithms are so essential that they contribute to a considerable portion of the overall computation time. For instance, partitioning, merging, selecting, and inserting are the core operations in sorting algorithms. A minor operation, such as assigning a partitioned array to two subarrays, is often not considered from the cost model.

The input size of N determines which cost model fits the purpose. In a real-life scenario of re-embedding steganalysis, one can expect the input (steganograms) to be a

finite length, using finite memory cells. A *uniform cost model*, therefore, is more suited rather than the *logarithmic cost model*.

Since the quantity we are dealing here is finite, elementary arithmetic operations (bit addition, subtraction, multiplication and division) on discrete values of integers that are encoded in certain number of bits are regarded as constant, $O(1)$, in the uniform cost model [108]. For example, Yang et al. evaluated that the T-entropy calculation of a non-finite input computationally takes $O(N \log N)$ [109] in logarithmic cost model but Rebenich demonstrate it is in the order of N in the uniform cost model [108].

The re-embedding steganalysis re-embeds the source 101 times, i.e., (0% to 100% re-embedding levels), and such operation may be regarded as linear. We however assign it as M because it is the core operation in re-embedding steganalysis that repeats the statistical metrics.

Table 6.1: Performance evaluation of various metrics in re-embedding steganalysis. Chi-square, Kullback-Leibler Divergence, and autocorrelation function fall behind in terms of robustness. Most metrics have the time complexity of $O(N)$.

Metric	Robustness (Sequential)	Robustness (Equal spacing)	Time Complexity
T-entropy	Cannot detect very low and very high embedding levels		$O(MN)$
Welch's t-test	Cannot detect very high embedding levels		$O(MN)$
Chi-square test	Cannot detect 8 out of the 11 embedding levels	Cannot detect half of the embedding levels	$O(MN)$
Kolmogorov-Smirnov test	Cannot detect very low and very high embedding levels		$O(MN)$
Shannon entropy	Cannot detect very high embedding levels		$O(MN)$
Kullback-Leibler Divergence	Cannot detect 9 out of the 11 embedding levels	Cannot detect 7 out of the 11 embedding levels	$O(MN)$
Autocorrelation Function	Cannot detect	May detect the presence but it cannot quantify the embedding level	$O(MN^2)$

Table 6.1 summarises the robustness and the time complexities of the evaluated metrics. Overall, three out of the seven evaluated metrics are not compatible with re-embedding steganalysis: Chi-square test, KL Divergence, and Autocorrelation function (ACF). These metrics were either inaccurate in estimating the size of the steganograms or could not detect the presence at all. In terms of the time complexity, all the evaluated metrics computes in the order of MN, except the ACF computes in the order of MN^2.

Re-embedding steganalysis using T-entropy and K-S share a similar result: they cannot detect very low or very high embedding levels. Being unable to detect very low and very high embedding levels accurately may be a critical flaw. Using gradients can fix such a flaw, though this approach encourages parameterisation. Welch's t-test and Shannon entropy performed the best where they are only incapable of detecting highly embedded sources. The default flags values of 000 contributed to this phenomenon. As

already discussed in Section 4.3, two of the three bits are noise in the compliant mode that can hinder the accuracy of the re-embedding steganalysis. In T-entropy, two noise bits are part of the secret data, affecting the accuracy of the detection. These noise bits, however, have no impact on the Shannon entropy and Welch's *t*-test performances because the probability and sample mean are not affected by the two 0s.

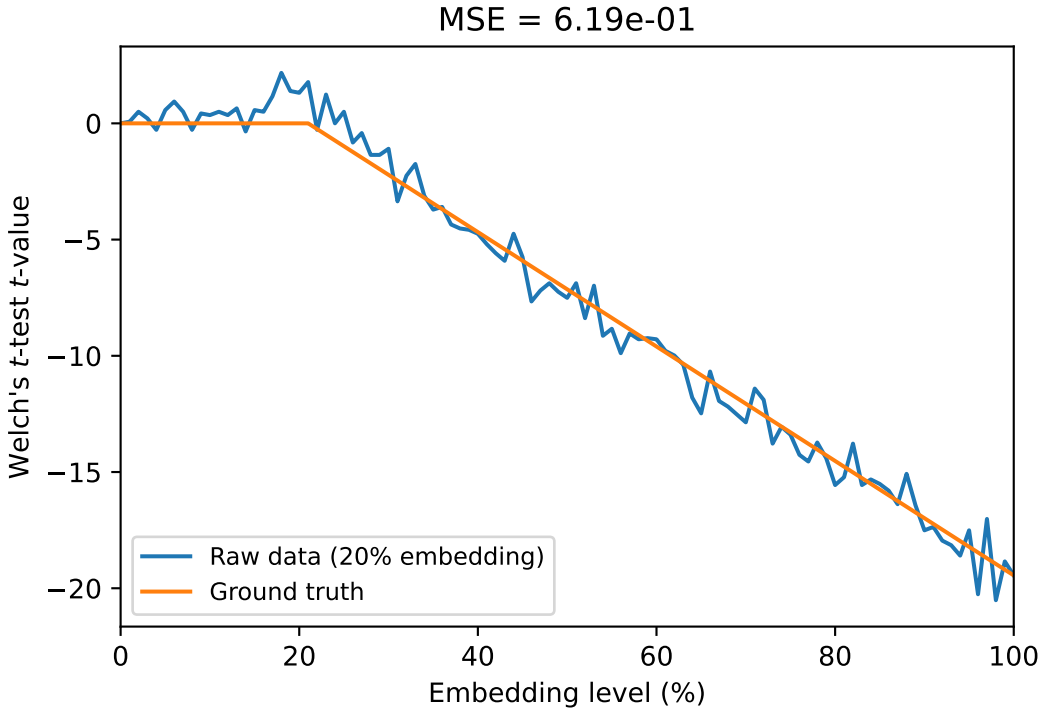


Figure 6.29: Calculating mean squared error (MSE) of raw data and ground truth data allows us to quantify noise. The noise level of Welch's *t*-test 20% embedding level is 0.619.

Overall, T-entropy appears to produce graphs with least noise out of all the metrics experimented with. We can measure noise by calculating mean squared error (MSE) of the metrics, i.e., how much the raw data deviate from the ground truth values. This is possible since an ideal case graph should remain horizontally constant until it reaches the embedding level, where it starts to increase linearly. Figure 6.29 illustrates this for

Welch's *t*-test 20% embedding level, showing a noise level of 0.619.

This approach, however, does not work across different metrics because they are in different scales, e.g., the noise level of K-S test embedded with 20% is 0.0002, though, fluctuations on K-S test and Welch's *t*-test appear to be almost similar (cf. Figure 6.1). Normalising our datasets, therefore, is a necessary step before noise levels can be compared directly.

Table 6.2: Noise evaluation of various metrics in re-embedding steganalysis. T-entropy and Kolmogorov-Smirnov test are the least and most noisy metrics, respectively. Noise generally increases as the embedding level increase.

Metric	20% Embedding	40% Embedding	60% Embedding	80% Embedding
T-entropy	4.97×10^{-4}	9.83×10^{-4}	7.41×10^{-4}	6.84×10^{-4}
Welch's *t*-test	1.54×10^{-3}	2.89×10^{-3}	7.58×10^{-3}	1.94×10^{-2}
Kolmogorov-Smirnov test	2.74×10^{-3}	3.95×10^{-3}	1.06×10^{-2}	2.23×10^{-2}
Shannon entropy	1.27×10^{-3}	2.25×10^{-3}	7.86×10^{-3}	2.06×10^{-2}

Table 6.2 shows the normalised noise levels of different metrics with varying embedding levels. The table only lists metrics that were compatible with re-embedding steganalysis. T-entropy performed the best, generating noise levels around the thousandths. On the other hand, other metrics hover around in the ranges of the tenths and hundredths. In general, noise level increases as the embedding level increases. T-entropy is the only exception here since its noise levels are already substantially low to begin with.

Welch's *t*-test and Shannon's entropy metrics have almost matching noise values for all the embedding levels. Moreover, the box plots of the two metrics almost resemble each other (see Figures 6.4 and 6.19). On the other hand, the noisier K-S test metric resulted in a box plot with wider ranges (see Figure 6.14). These observations may suggest that noise imapcts the performance of the automatic detector. On the contrary, T-entropy with the least noise did not perform the best. Again, default flags values of

000 may have contributed to this phenomenon, suggesting that future research should focus on a wider variety of experiments.

6.8 Summary

In this chapter, we experimentally demonstrated that re-embedding steganalysis technique can be utilised with other statistical metrics besides T-entropy. Four of the seven metrics were compatible with the re-embedding steganalysis. Promising results have been observed from the sequential and equal spacing embedding schemes, but none of the metrics were able to reveal any interesting detection features for the random embedding scheme.

In the next chapter, we shift our attention away from storage-based steganography and discuss timing-based steganography.

7

Network Timing-based Steganography

This chapter focuses on IPD-based timing-based steganography and its feasibility to be used in communication with nine different locations around the globe. Sections 7.1 and 7.2 discuss the scope of our timing channel and its related work, respectively, followed by a discussion of network delays and their impact on timing channel creation in Section 7.3. Section 7.4 then describes our model and experiments, whose results are presented in Section 7.5. The final section then summarises the chapter.

7.1 Introduction

Network timing characteristics can be used to hide information, an approach known as a covert timing channel (CTC). There are a wide range of CTC techniques available, but this chapter focuses on the technique that modulates the interpacket delay (IPD) [32, 57, 110].

This chapter first discusses a number of inherent network effects that may prevent or impair the use of CTCs. We then experimentally demonstrate the feasibility of an IPD-based CTC over long paths, followed by a discussion of the aforementioned factors and how they impact on CTC design and feasibility of IPD-based CTCs over long paths. To our knowledge, long distance CTCs between various locations across the world have not yet been experimentally studied in the literature. For our study, we created CTC channels between New Zealand and nine Amazon Web Service (AWS) instances around the globe. The channels were used to communicate a message at hourly intervals over a week's duration. We designed and studied three encoding methods: 1 bit per packet (BPP), 2 BPP, and 3 BPP using two, four, and eight nominal interpacket delays, respectively. The experiment results suggest that one can expect there to be around a 10% error rate regardless of the encoding scheme, and stress the need for error correcting codes in the CTCs.

7.2 Related work

Keyboard JitterBug is a physical interception device that spies on a protected network from within the network. It smuggles its captured data out of the protected network by modulating the timing of legitimate traffic exiting the network, thereby creating a CTC from within the protected network to a receiver outside the network. The encoding method in [32] uses a modulo operation on the IPDs and a window size. The window size is manually adjusted to suit the network environment, i.e., based on the amount of jitter already present in the network and on the maximum tolerable delay. Example: Assume that we wish to convey the bits 11001 with a 30 ms window size. JitterBug would then add delays to IPDs that when we reduce new IPDs modulo the window size results in 15, 15, 0, 0, and 15 ms delay, respectively. E.g., if the actual keystroke IPDs are 153 ms, 324 ms, 623 ms, 594 ms, and 891 ms, then the added delay would result in

IPDs of 165 ms (=5 x 30 ms + 15 ms), 345 ms (=11 x 30 ms + 15 ms), 630 ms (=21 x 30 ms + 0 ms), 600 ms (=20 x 30 ms + 0 ms) and 885 ms (=29 x 30 ms + 15 ms). The receiver, which only knows the windows size, decodes the message by performing the modulo operations. The window size plays vital role here because it accounts for the possible jitter there may be. For instance, the receiver may receive IPDs ± 5 ms from the intended IPD. Doing the modulo operation on IPD that conveys 1 will result in the values that would be within 10 to 20 ms. In the case of 0, the values would be within the ranges of 0 to 5 ms and 25 to 29 ms.

Cabuk et al. developed an on-off CTC [57]. A sender and a receiver agree on a time interval and a network protocol that is used. The sender remains silent or transmits a packet during the time interval to convey a bit 0 or 1, respectively. The time interval acts as a clock so a smaller time interval provides higher bit rate.

Gianvecchio et al. developed a model-based CTC [110] that first monitors a specific network traffic, gathering its IPDs. Using a fitter, the gathered IPDs are fitted to six probability distributions. The fit with the smallest root mean squared error (RMSE) gets selected as a model. Using the model, covert communication occurs mimicking the selected distribution. This approach is effective because IPD dsitribution of a covert communication would not deviate much from the "normal" traffic.

Ker et al. claim that most existing media steganography and steganalysis experiments are run in laboratory conditions under a number of assumptions and that any stressing of the steganography channel that is done is far from what one would expect in the real world [111]. One can argue that the investigation of covert timing channels might suffer from similar issues. In CTCs, critical information such as the network environment setup is often left out by authors [57, 112]. This not only prevents one from assessing the scope of the experiment but it also makes replication difficult.

7.3 Network delays

The round-trip time (RTT) is an important metric in CTC design as it measures the time until the message sender receives a reply from the recipient. In other words, it is the sum of the two network latencies, from sender to recipient and from recipient to sender. In general, the latency is calculated by factoring in network delays, i.e., processing, queueing, transmission and propagation delays. We review each of these delays below and discuss how they may influence the CTC design.

Processing delay

Processing delay is defined as the time intermediary devices such as switches and routers spend on processing a packet. The intermediary devices are mainly responsible for the next route identification by processing network headers, but other processing, such as integrity checks on packets or security features may also be included. Ramaswamy et al. demonstrate that there is a significant increase in overall network delay when an intermediary device has to execute a complex task [113]. As an example, the authors compare processing delays required in simple forwarding against encryption. They also claim that more complex processing is becoming a trend as there are increasing security demands in edge and access routers. One can, however, argue that routers also evolve to meet such demands. A fundamental question a CTC designer should ask is if there exists an accurate way of measuring exactly how many intermediary devices packets need to traverse to reach their destination and what the processing times in each of these heterogeneous intermediary devices are. A technique that may use two or more different network protocols to communicate may suffer from processing delay, e.g., a network protocol may happen to require a more strict security policy than the other protocols that are concurrently in use. Processing may not always result in the same delay, but it could be viewed as a fixed delay to a certain extent.

Queueing delay

Queueing delay is defined as the time intermediary devices hold onto a packet before it is processed and directed to the next device. An intermediary device, however, can only accommodate a certain number of packets at a given time due to limits in its buffer size. If an intermediary device is congested as a result of prolonged burst transmissions, all subsequent packets are discarded or redirected to another intermediary device until buffer space becomes available again. Large router buffer sizes may appear to solve the problem described above, but cause bufferbloat instead [114, 115]. Furthermore, there exist different queueing mechanisms and Quality of Service (QoS) implementations [116]. All of these factors contribute as uncertainty in the IPD calculation. Queueing delay is thus an uncontrollable variable delay.

Transmission delay

Transmission delay is defined as the time it takes to push an entire packet onto the medium. Its calculation is based on the maximum data link rate of a medium; but the perceived throughput may be lower where media contention is allowed for, e.g., on WiFi links. It is generally a variable delay because packets may have different sizes and the medium may be busy for a variable amount of time. If every packet is of the same size, then transmission delay may be regarded as fixed.

Propagation delay

The propagation delay is the time difference between the time that the start of a packet leaves the source and the time it reaches the next hop. If the distance a packet has to travel and the speed at which it travels are both fixed, propagation delay may be regarded as a fixed delay.

Of the delays above, queueing delay is the most problematic because it depends on uncontrollable variables. Transmission delay may be problematic if a sending host has an overwhelming amount of active sessions, causing packets to be queued on the host. Propagation delays only become problematic if the path is not stable, which can be in issue in wireless applications.

7.4 Methodology

This experiment modulated the IPDs of ICMP ping packets to convey a message. The message originated from New Zealand and was sent to nine different AWS instances across the globe, located in Sydney, Tokyo, Ohio, Oregon, Frankfurt, Ireland, São Paulo, Seoul, and London. These locations were selected to provide an insight into the effect of topological network distance on CTC. The message was encoded with a self-synchronising T-code to enable recovery from bit errors (see Section 3.1.1). The message size was 2994 bits and was sent to the aforementioned locations once per hour for a week.

As discussed in Section 7.2, there are a number of ways to modulate the IPD of packets. For this experiment, we determined the required IPD modulation amount from the network jitter. Absolute delay variations in the packets were used to calculate the network jitter [117]. In this experiment, the network jitter and the average latency were calculated every time before the actual message was sent. This was done by sending ten ICMP echo request packets from New Zealand to the respective AWS instance. Both the New Zealand host and the AWS instance each calculated the observed jitter and the average latency. The New Zealand host sent its measured jitter as a payload byte in the last ICMP request packet. The AWS instance compared its own jitter value against the one sent by the New Zealand host and chose the larger as the synchronised jitter value. The AWS instance then communicated this information to the New Zealand host as a

payload byte in the ICMP response packet, thus giving the New Zealand host and the AWS instance a common IPD modulation value. After this synchronisation, the New Zealand host sent the actual message of 2994 bits to the AWS instance by modulating the IPDs.

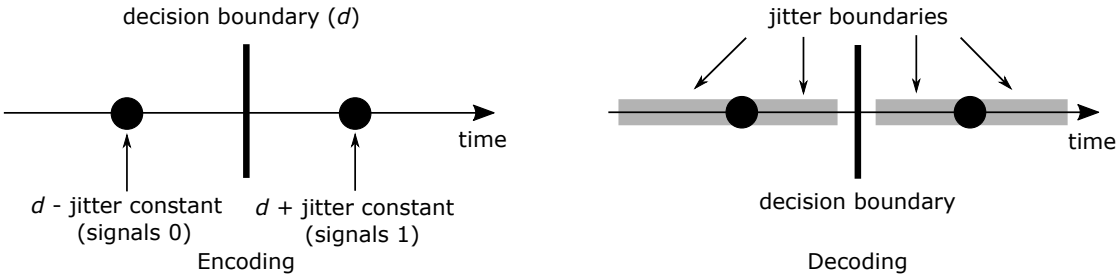


Figure 7.1: The encoder diagram (left) illustrates how a message sender can modulate IPDs to express the binary bits 0 and 1. The decoder diagram (right) illustrates a functional channel if the delays introduced by the sender are detectable by a receiver. The delay introduced by a sender should be large enough to take into account network jitter, i.e., the jitter boundaries for 0 and 1 should not overlap.

The three encoding methods used in this experiment are 1 bit, 2 bits, and 3 bits per packet (BPP). In the 1 BPP case, there is one decision boundary and two possible deviations for IPD modulation. Figure 7.1 illustrates this. The two IPD modulations are spaced out to account for the network jitter. Our preliminary experiment revealed that spacing the two IPD modulations by a factor of four of the observed jitter yielded a positive outcome. This creates a *jitter boundary* for each IPD modulations, i.e., a range a receiver uses to decode the different IPD modulations. Jitter boundary has to span from both sides of a modulated value because a packet can arrive earlier or later. Let us assume an average latency of packets and network jitter to be 200 ms and 2 ms, respectively. To convey message bits 0 and 1, the sending host modulates IPDs at 196 ms and 204 ms, respectively. A message receiver would decode IPDs between 192 ms to 200 ms as 0 and IPDs between 201 ms to 208 ms as 1.

In case of 2 BPP and 3 BPP, one needs four (00 to 11) and eight jitter boundaries (000 to 111), respectively. This in turn means that there exist three and seven decision boundaries for 2 BPP and 3 BPP, respectively. Figure 7.2 illustrates 2 BPP. Overall, the New Zealand host sends less packets if more bits are encoded per packet.

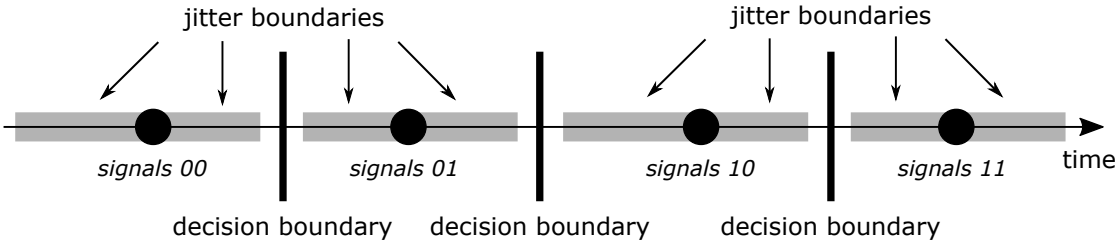


Figure 7.2: The figure shows 2 bits per packet (BPP) encoding method which consists of four jitter boundaries and three decision boundaries. Encoding more BPP has an advantage of transferring fewer packets to convey a message.

Table 7.1: Each experiment was run once an hour. Experiments were scheduled with different time offsets during each hour to minimise possible link and process interference between different experiments. For example, the Sydney 1 bit per packet experiment was executed hourly, at 1 minute past the hour. Each experiment was designed to be repeated for a week, i.e. have a total number of 168 repetitions.

Location	1 bit per packet	2 bits per packet	3 bits per packet
Sydney	01 min	21 min	41 min
São Paulo	02 min	22 min	42 min
Tokyo	03 min	23 min	43 min
Ireland	04 min	24 min	44 min
Oregon	05 min	25 min	45 min
Frankfurt	06 min	26 min	46 min
London	07 min	27 min	47 min
Ohio	08 min	28 min	48 min
Seoul	09 min	29 min	49 min

Overall, twenty-seven experiments were executed every hour to test nine AWS instances each with the three different encoding methods. To minimise possible network

Table 7.2: Average ping latency from our host in New Zealand to the nine AWS locations around the globe. Sydney, which is closest to New Zealand, had the lowest latency of 37 ms. São Paulo had the highest latency of 331 ms. Such locations were employed in an attempt to gain insight into the impact latency has on CTC.

AWS location	Latency(ms)
Sydney	37
Tokyo	151
Ohio	210
Oregon	168
Frankfurt	279
Ireland	300
São Paulo	331
Seoul	177
London	276

link congestion and processing delay, the experiments were staggered using the time epoch as shown in Table 7.1. All experiments were executed with the help of crontab and shell scripts. The table has been constructed by reflecting on the ping latencies between the New Zealand host and the nine locations. Table 7.2 shows the preliminary ping results. As may be seen from the table, the location which shows the highest latency is São Paulo with 331 ms. This means, one would spend 991 seconds or 16.5 minutes if one wanted to send 2994 bits of message to São Paulo using the 1 BPP encoding scheme.

7.5 Results

Figure 7.3 shows the average Bit Error Rate (BER) of the timing channel tested on the nine AWS instances. Decreasing trends were observed across all the AWS instances except Sydney, Ohio and London instances. Channel duration combined with IPD gap length may explain this phenomenon. The jitter value is calculated once at the start of a communication. Thus, if the channel lasts for a longer duration, jitter and average latency values would change, thereby making our CTC unstable. As mentioned in the

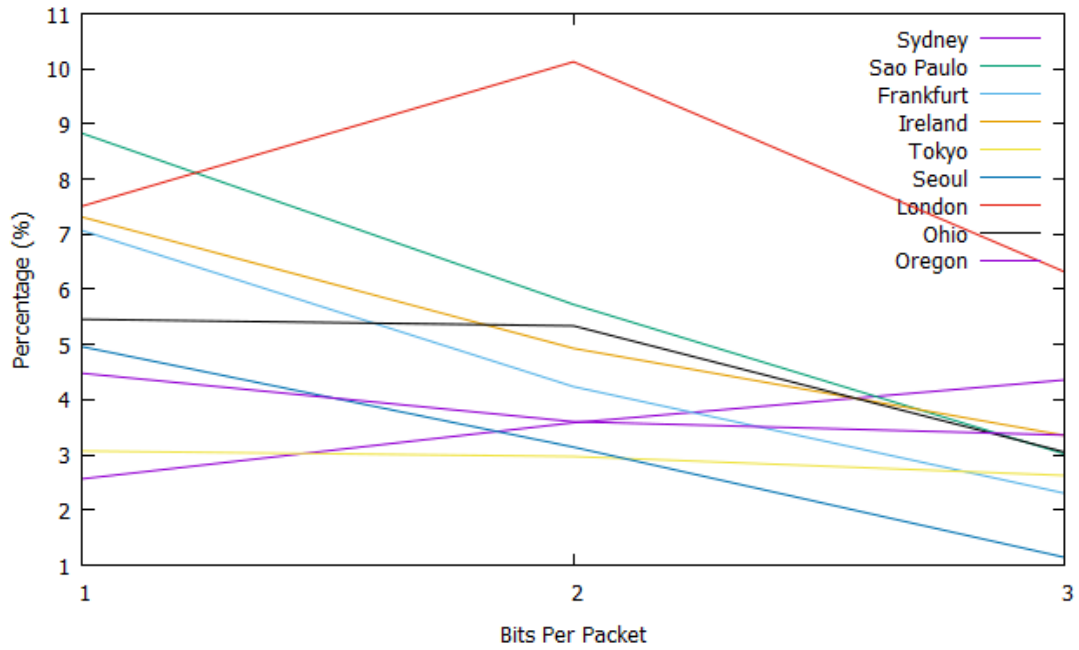


Figure 7.3: The average Bit Error Rate (BER) of the timing channel on the nine locations across the globe. Three different encoding methods were tested for each location; 1 Bits Per Packet (BPP), 2 BPP and 3 BPP. Overall, BER decreases with BPP, with the exceptions of Sydney, Ohio and London.

methodology section, fewer packets are sent when more bits are encoded per packet. This may explain why BER of 1 BPP is higher than 3 BPP. Lastly, ICMP packets are low priority packets that could be dropped in the event of network congestion. This may have contributed to increase in overall BER values. Sydney draws a counter-intuitive curve because rising BERs have been observed with higher BPP encodings. This is caused by IPD modulation amount strain into negative territory. According to Table 7.2 and Figure 7.5, the average latency and the median jitters of Sydney were 37 ms and 1.4 ms, respectively. There were instances where the jitter would go up to 4 ms. If the 3 BPP encoding method was used under this scenario, the IPD modulations are -19 ms and -3 ms to convey "000" and "001", respectively. A negative latency number will be treated as no delay so a packet will immediately be sent out one after the other. This in turn means that the error is introduced by the sender, not the network. Overall, none of the locations has a BER above 10%. The erroneous bits may be corrected with an error correcting code.

Figure 7.4 shows a bitwise difference of the nine locations in boxplots. This figure may be viewed as an expansion of the Figure 7.3. Figure 7.3 shows the average BERs, expressed in percentages, whereas Figure 7.4 shows the actual erroneous bits and their distribution. The median values are positioned toward the lower quartile. Two exceptions to this rule are the Sydney 3 BPP and the Seoul 3 BPP series. In the case of the Seoul 3 BPP, one could assume that the pipe was exceptionally stable and noise-free at the experiment time. The range of the Seoul 3 BPP boxplot seems to further support this conjecture since the range is much shorter than for other locations.

Figure 7.5 shows the jitter measurements of the nine locations in boxplots. The jitter measurements span for different for location, but it is generally within 3 ms to 4 ms. The box, which indicates the interquartile range (IQR), does not show significant difference between the AWS instances. The IQR of the boxplots generally have the size of 1 ms and tend to be placed toward the lower part of their range. Furthermore, the median values

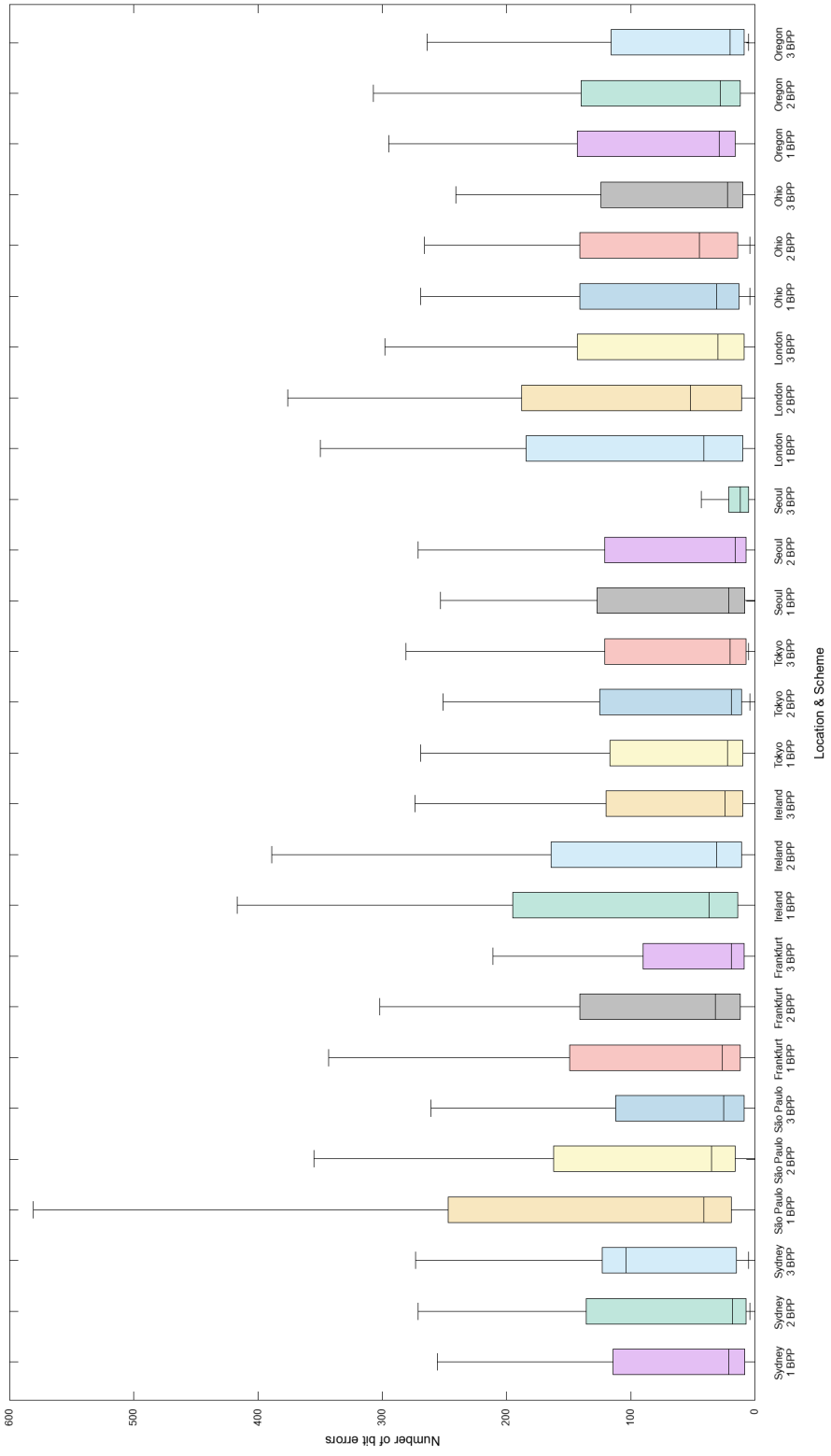


Figure 7.4: Bit errors for the nine locations. The *boxes* are generally placed on the lower part of the range, and so is the median of each respective box. The exceptions are the Sydney 3 BPP and Seoul 3 BPP series, where the median is placed in the upper middle part of its series box, respectively.

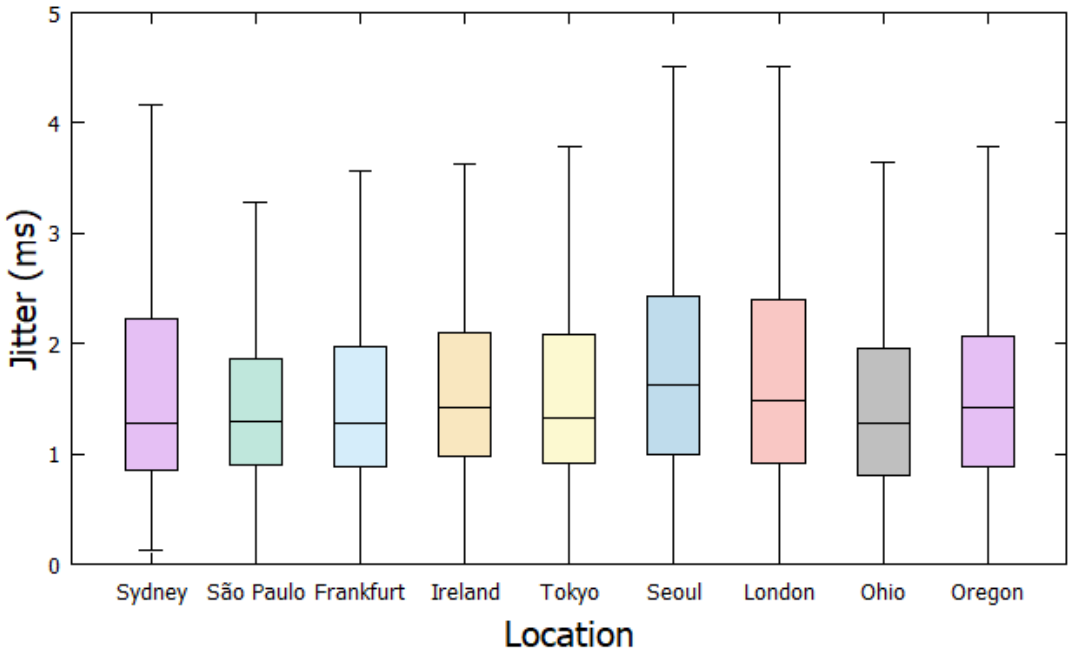


Figure 7.5: Jitters of the nine locations in boxplots. Ranges generally span from 3 ms to 4 ms, and the *boxes* are placed toward the lower half. There appear no significant differences in the median jitter values.

of boxplots appear to be around 1 ms. Similar box sizes, box placements, and median values all may be used to indicate the stability of the network links. Notwithstanding the fact that these locations are spread around the globe. In terms of our timing channel experiment, one could say there was no significant difference in jitter values.

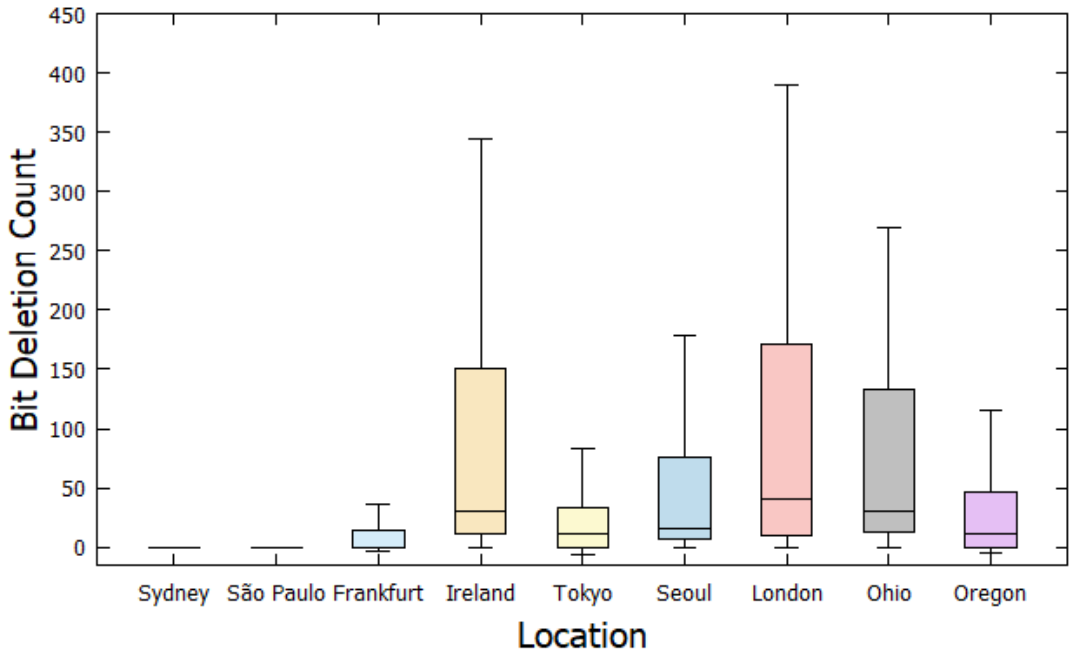


Figure 7.6: Bit deletions for the nine locations. This figure shows the distribution of the bits that did not reach the destination. Latency does not contribute to packet drops because Sydney and São Paulo had lowest and highest latency, respectively. However, both also had the least amount of dropped packets out of all the locations.

Figure 7.6 shows the bit deletion distributions of the nine AWS instances. Bit insertion is when an AWS instance receives more packets, i.e., more than 2994 message bits. These were also identified and they have been incorporated as data. However, bit insertions were rare, and they do not appear in the figure. Bit deletions are packet drops because it shows the actual number of bits an AWS instance has received from the 2994 message bits. One may assume a packet to be more susceptible to packet drops if it is exposed to a network longer. However, it appears latency does not contribute to dropped packets. As may be seen from the figure, both Sydney and São Paulo almost had no

bit deletions even though their ping latencies are lowest and highest among the AWS instances, respectively.

7.6 Summary

This chapter discusses the feasibility of using IPD-based CTC over long paths. Messages from New Zealand were sent to nine locations around the globe for a week at an hourly interval. Our results indicate that bit-perfect communication over long distances may not be possible with elementary CTC because of network jitter and packet loss. Bit deletions, which may be the result of buffer overflows, were observed for all the locations. The BER generally decreased when more bits were encoded per packet. Channel lifespan may have contributed to this because the channels require fewer packets when more bits are encoded per packet, decreasing the effects of jitter and packet loss. A jitter value determined at the beginning of the channel may also change during the channel's lifespan. Future work may investigate ways to adapt to changing jitter. Overall, the BER in our experiments remained below the 10% mark. Error correcting codes may be used to cope with erroneous and deleted bits. Future work should therefore focus on error correcting codes, and how they could be used to reduce bit errors in the long distance CTCs.

8

Conclusions

The first research question posed in this thesis was if it is possible to separate network flows that contain hidden data from flows that do not and, if so, which techniques one could employ for this binary classification. Additionally, this thesis also questioned whether the separation was even necessary in the first place. Chapter 2 investigated existing steganalysis techniques to conclude that using statistical measures to compare benign and malicious flows is a common practice. This thesis, however, contends that these approaches may not be suitable for network environments. Networks are heterogeneous and so are the users. A steganalysis technique that worked in a specific network is not guaranteed to work in the future or in other networks. Therefore, existing comparison-based approaches may not be scalable because frequent maintenance becomes inevitable. This problem stems from characterising benign traffic based on personal judgement and comparison with malicious traffic.

Chapter 4 demonstrated that network steganalysis does not need to rely on benign

traffic and that detection can be done without it. So the separation was not necessary. Furthermore, the approach proposed there can not only detect the presence of malicious flows but also estimate their size. This answers our second research question, whether it is possible to quantify the size of the malicious flows. Using a re-embedding steganalysis technique with a T-entropy measure sensitive to correlated data obtained positive results. With malicious traffic, the technique produces a graph with irregular patterns, i.e., a trend transition or a dip around the point when the re-embedding capacity matches the amount of secret data. Our results indicate that this observation was consistent throughout different embedding levels.

This is not to suggest that this approach is perfect, however. The technique works for equally spaced out and sequentially embedded data, but fails to detect randomly embedded data. This thesis suggests a gradient method that could circumvent the problem, but this introduces another layer of parameterisation which again requires frequent maintenance. Furthermore, since re-embedding steganalysis requires re-embedding at a range of different embedding levels, it is computationally more expensive than comparison-based approaches. This needs to be assessed in future work.

Another critical shortcoming identified during the experiments is that re-embedding steganalysis has limited practicability, especially in the network use case. The speed at which one can manually audit graphs falls well below the speed at which network traffic accrues. In order to address this problem, Chapter 5 developed a system that automatically detects unique characteristics in graphs, indicating the potential presence of malicious flows as well as their size. This system works both for equally spaced and sequentially embedded data, except for very low and very high embedding levels. Again, using gradients may address this problem, but their performance assessment and requirement for frequent maintenance remain as open problems. Chapter 6 further considered whether other metrics besides T-entropy could be used in re-embedding steganalysis. The chapter experimentally evaluated a number of such statistical metrics

used in the comparison-based steganalysis literature. The results show that T-entropy, Welch's *t*-test, Kolmogorov-Smirnov test and Shannon entropy are compatible with the system proposed in Chapter 5. This represents a substantial improvement over the existing techniques as it also provides information on the amount of secret data.

However, it can also be argued that a different storage-based steganography scheme could have yielded contradicting results because correlated traffic may have contributed to he aforementioned positive results. This remains an open problem, requiring one to re-assess the compatibility with other steganography schemes. Experimentally evaluating our system in different storage-based steganography schemes will validate which metric performs the best with re-embedding steganalysis.

This thesis then turned its attention to timing-based steganography and assessed its feasibility. The results in Chapter 7 indicate that T-code embedded secret message communication over timing channels is feasible. Based on the conjecture that the network environment might play a significant role, we tested this communication channel to different sites around the globe. This goes beyond some of the literature, where details of the network environment were not considered. The results showed that channel error rates remained below the 10% mark, generally decreasing with shorter communication time. Our channel design contributed to this phenomenon because our timing channel uses jitter measured right before the communication to account for possible random delay variations ("'timing noise"') on the link. This suggests that a regular exchange of jitter may result in a more error-free link. Arguably, more diverse communication links could have provided more compelling insights, i.e., mixed combinations of locations rather than a New Zealand host only to other locations around the globe.

Last but not least, this thesis also investigated network steganography prevention techniques as part of the last research question. Normalising traffic is one of the most effective ways of preventing steganography channels, but this thesis argues that prevention only fuels the arms race and many questions can go unanswered. An adversary

who notices a blocked steganography channel will resort to creating a more sophisticated channel that is harder to block. Furthermore, normalising traffic may remove an opportunity to gather valuable information, such as parties involved, compromised information content and amount. A detection system at least permits further analysis of a suspected channel.

In conclusion, the main contributions of this thesis were:

1. Employed T-entropy in re-embedding steganalysis within network environments with positive results. The re-embedding steganalysis technique not only works in the absence of benign traffic but also quantitatively indicates suspicious data.
2. Designed automatic detectors for re-embedding steganalysis, making the detection of storage-based channels more practical. Offloading the manual auditing step to the automatic detectors saves time and human resources.
3. Investigated re-embedding steganalysis using several different statistical metrics and studied their compatibility with the automatic detector. Not all statistical metrics were as suitable as T-entropy because many metrics do not account for correlation.
4. Introduced T-codes in timing-based steganography, showcasing them as a potential clandestine communication tool.

Future work should look at ways to detect the randomly embedded data, improve the automatic detectors and the timing channel.

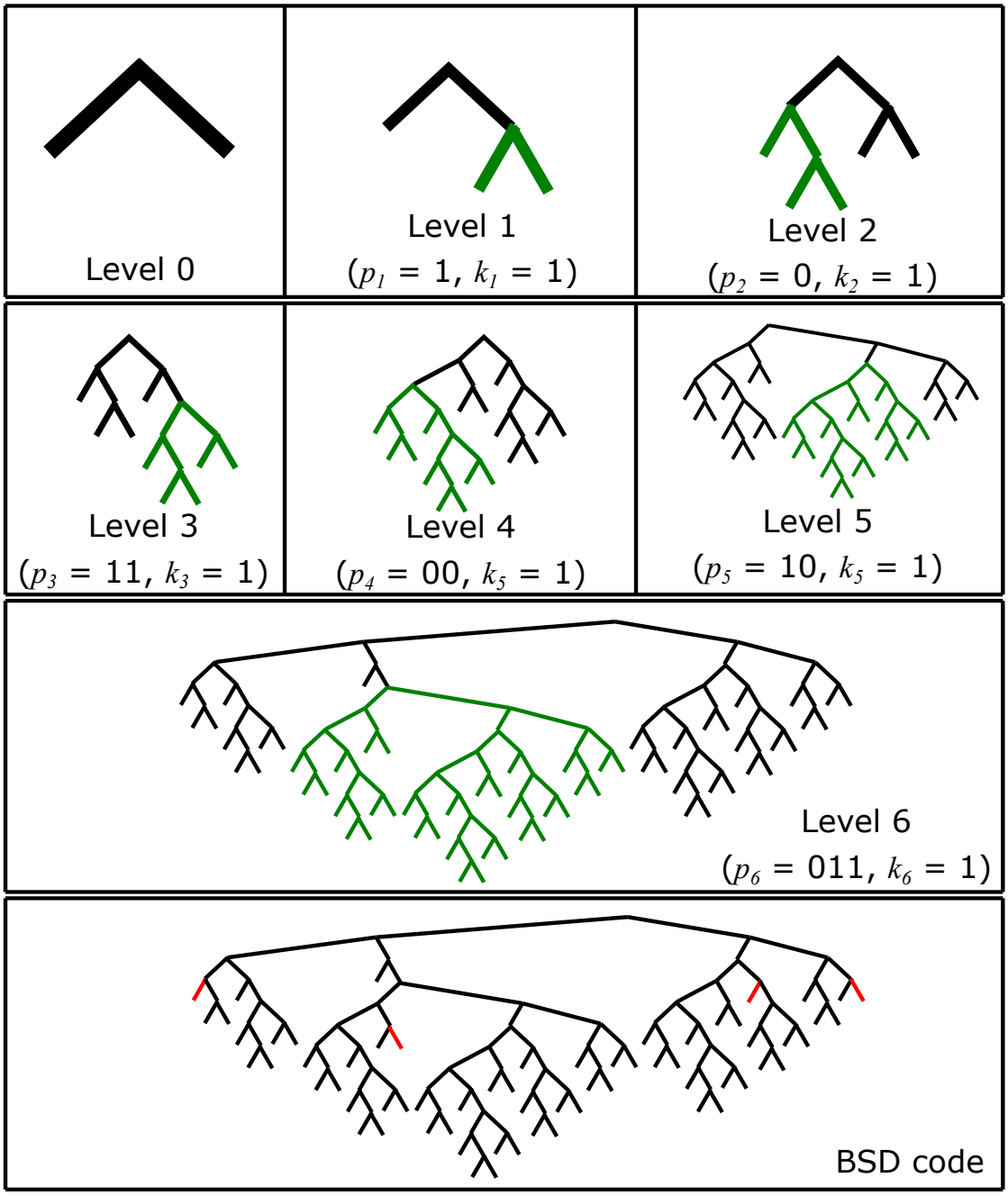
Appendices

A

Additional Details for the T-code tree

This section includes full details of the T-code set, $S^{1,1,1,1,1,1}_{1,0,11,00,10,011}$, that we used to encode our steganograms. Green lines indicate newly generated branches with T-augmentation, whereas red lines show the removed codewords to turn our T-code into a BSD code. In summary, T-expansion parameters were always 1 to minimise tree's growth in height. T-prefixes were the smallest possible value at each iteration.

- Level 1, smallest prefixes: {0, 1}, selected 1
- Level 2, smallest prefix: {0}, selected 0
- Level 3, smallest prefixes: {00, 10, 11}, selected 11
- Level 4, smallest prefixes: {00, 10}, selected 00
- Level 5, smallest prefix: {10}, selected 10
- Level 6, smallest prefixes: {010, 011}, selected 011
- BSD, removed PCE: {0000, 1010, 1111, 011011}


Level 0
Level 1
(p_1 = 1, k_1 = 1)
Level 2
(p_2 = 0, k_2 = 1)
Level 3
(p_3 = 11, k_3 = 1)
Level 4
(p_4 = 00, k_5 = 1)
Level 5
(p_5 = 10, k_5 = 1)
Level 6
(p_6 = 011, k_6 = 1)
BSD code

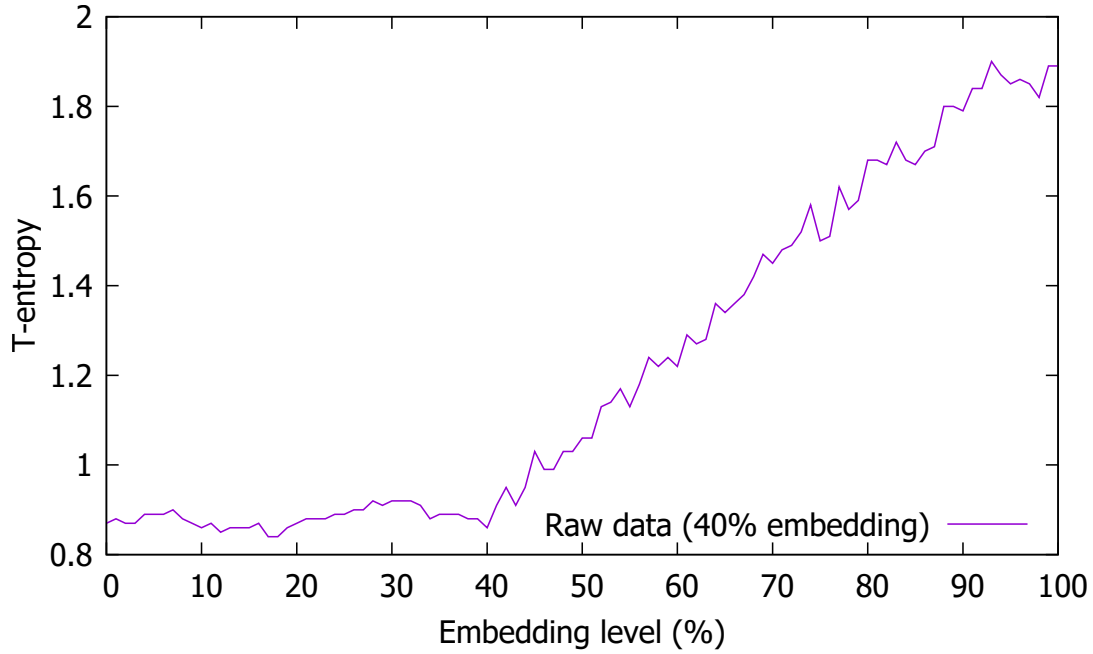
B

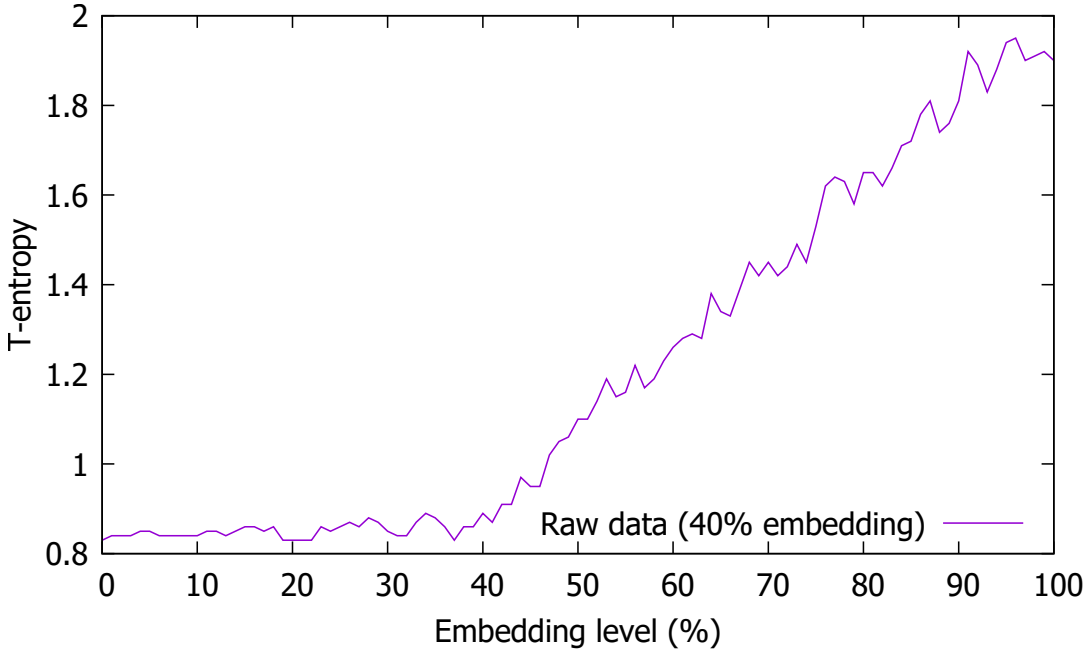
Additional Graphs for Automatic Detection

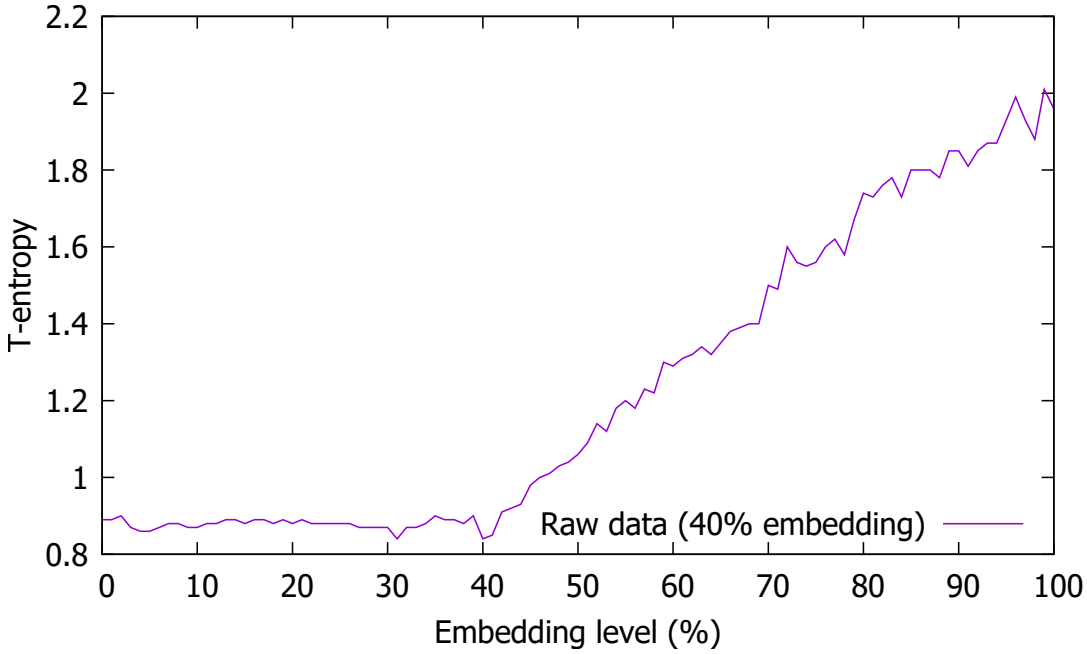
Most of the experimental results in Chapter 5 were based on one dataset, albeit 100 datasets were experimentally evaluated. This decision was made to enhance the readability and to preserve space. This appendix, therefore, presents additional graphs, showing wide variations in the graphs corresponding to the selected datasets. The variations in the graphs emphasise the difficulty of constructing an automatic detector.

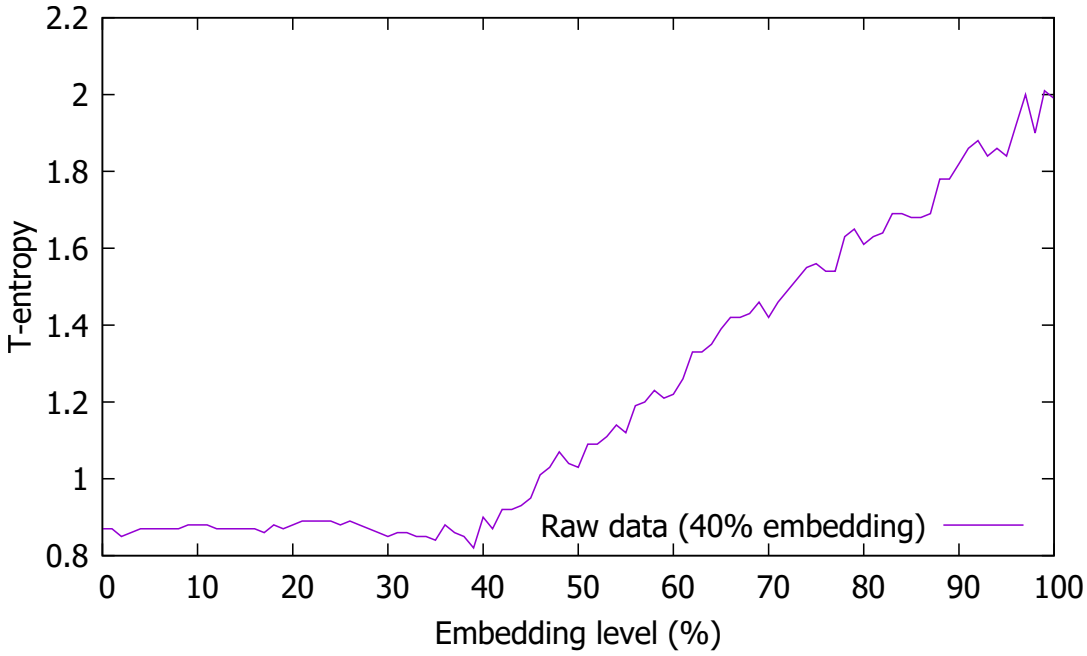
B.1 Collection of 40% embedded datasets

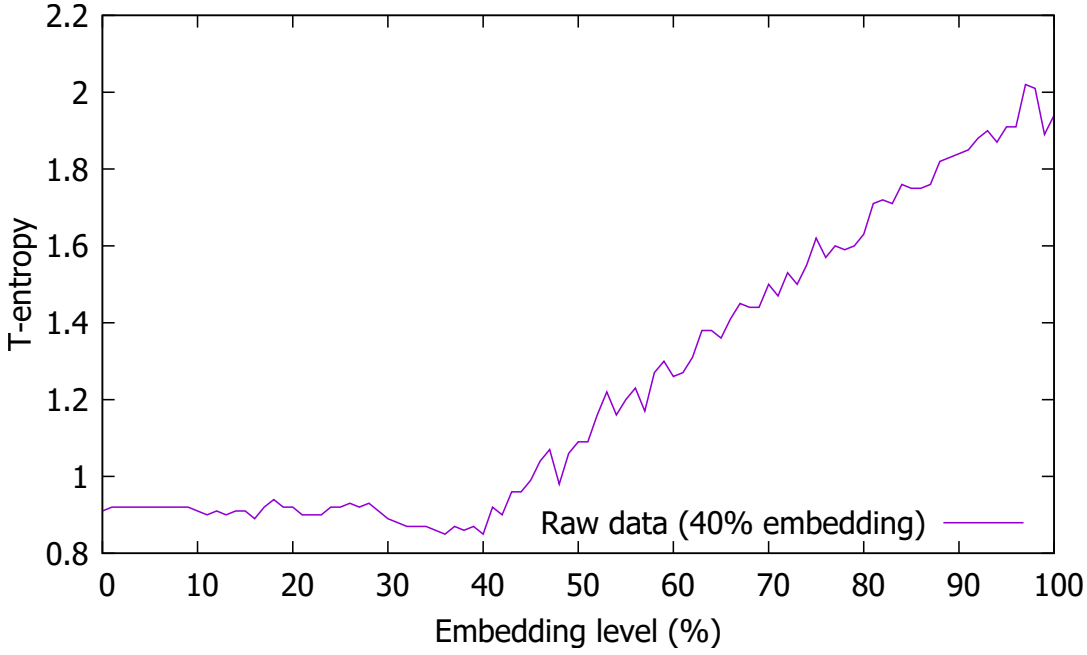
This section presents eight 40% embedded datasets to illustrate nonuniformities in the datasets. The figures reveal that the fluctuations (noise) are present in all the graphs but they do not share a common noise frequency or amount. The trend transition around the 40% mark is still evident from all the graphs.

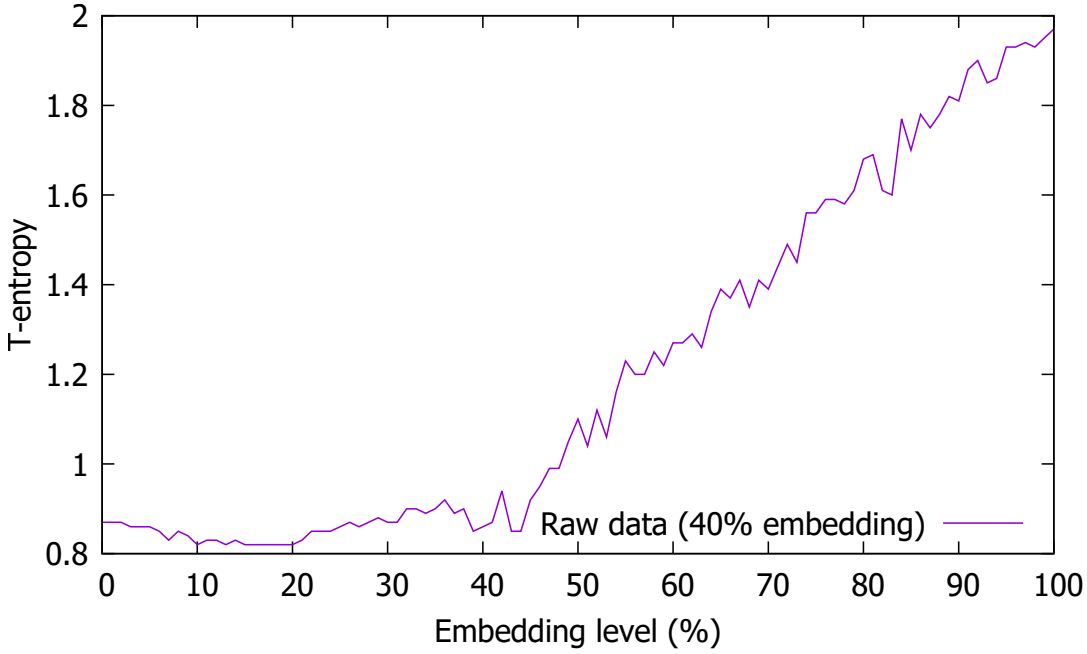


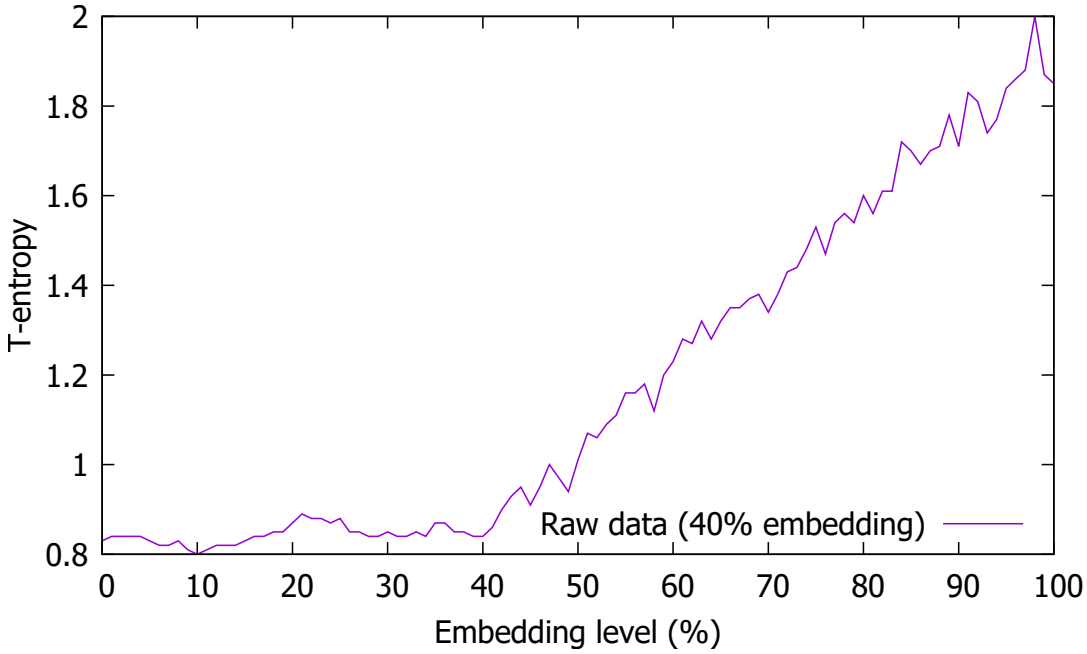

T-entropy
Embedding level (%)
Raw data (40% embedding)
2
1.8
1.6
1.4
1.2
1
0.8
0
10
20
30
40
50
60
70
80
90
100

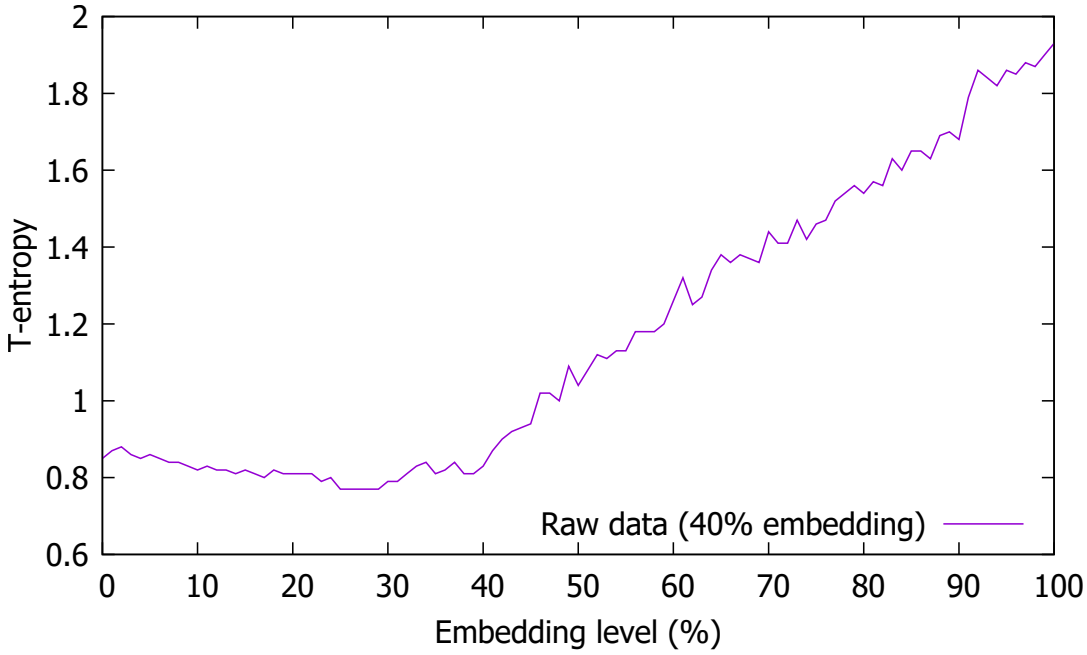

T-entropy
Embedding level (%)
Raw data (40% embedding)
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
10
20
30
40
50
60
70
80
90
100


T-entropy
Embedding level (%)
Raw data (40% embedding)
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
10
20
30
40
50
60
70
80
90
100


T-entropy
Embedding level (%)
Raw data (40% embedding)
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
10
20
30
40
50
60
70
80
90
100

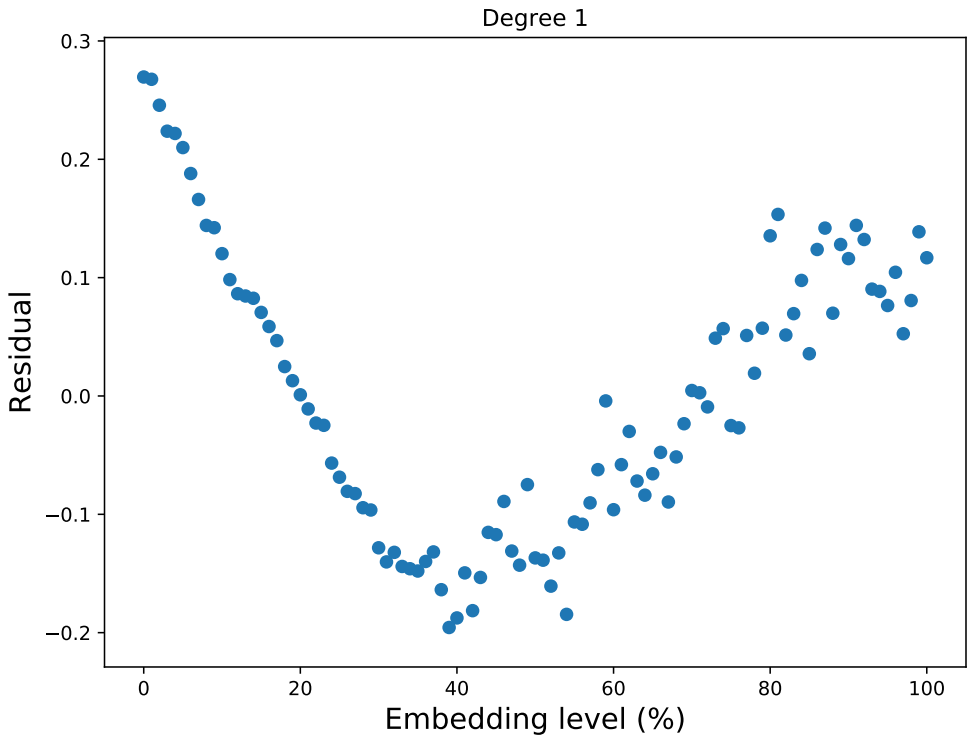

T-entropy
Embedding level (%)
Raw data (40% embedding)
2
1.8
1.6
1.4
1.2
1
0.8
0
10
20
30
40
50
60
70
80
90
100

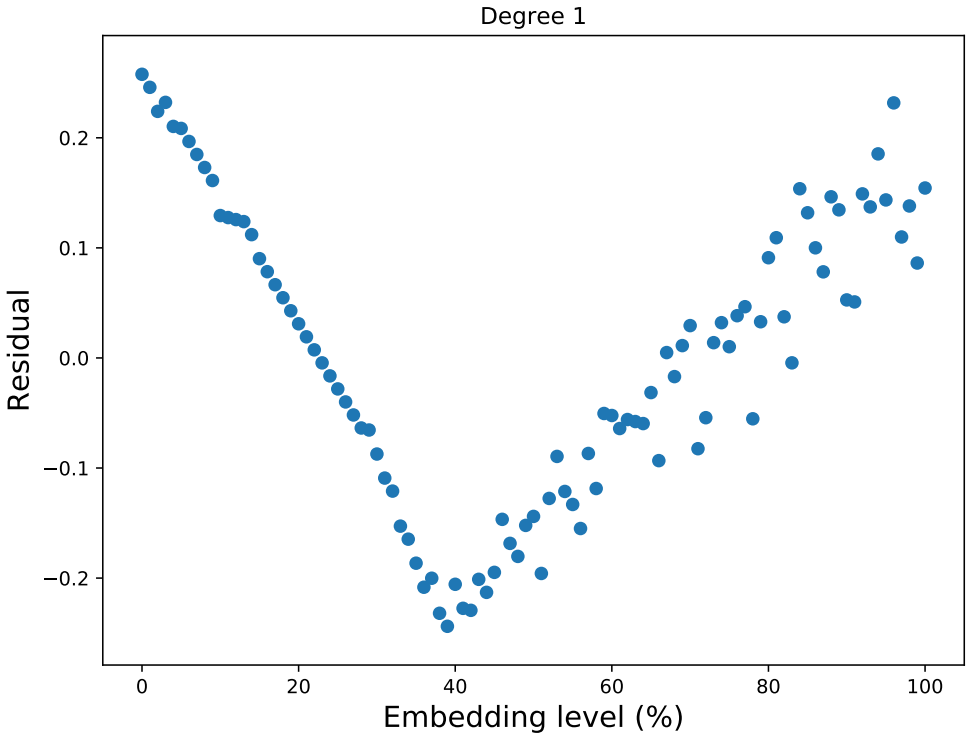

T-entropy
Embedding level (%)
Raw data (40% embedding)
2
1.8
1.6
1.4
1.2
1
0.8
0
10
20
30
40
50
60
70
80
90
100

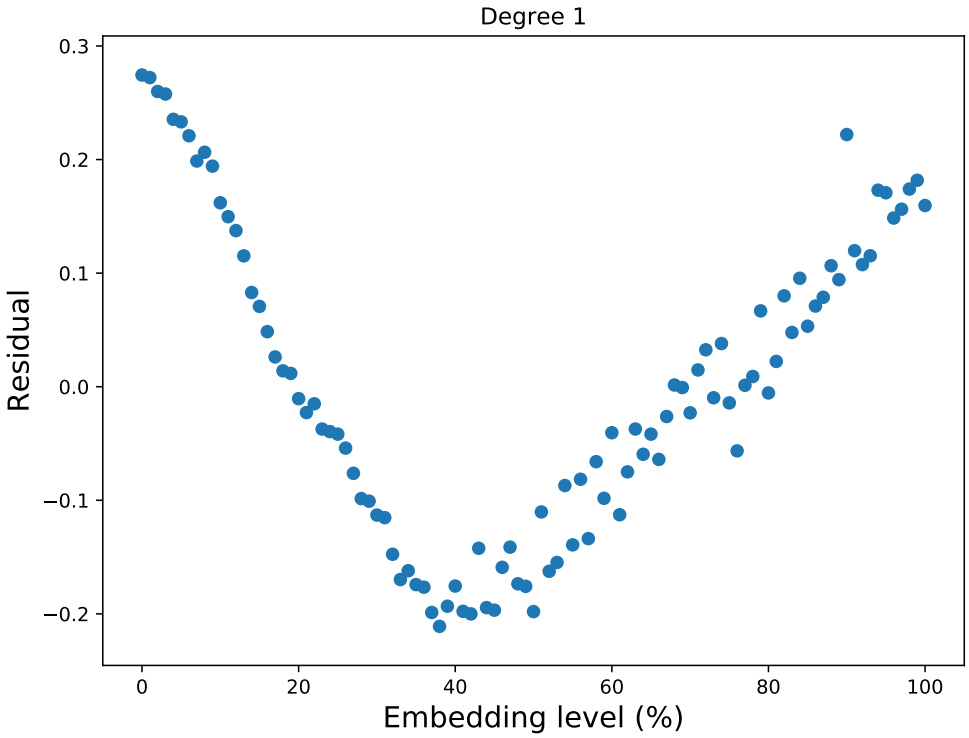

T-entropy
Embedding level (%)
Raw data (40% embedding)
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0
10
20
30
40
50
60
70
80
90
100

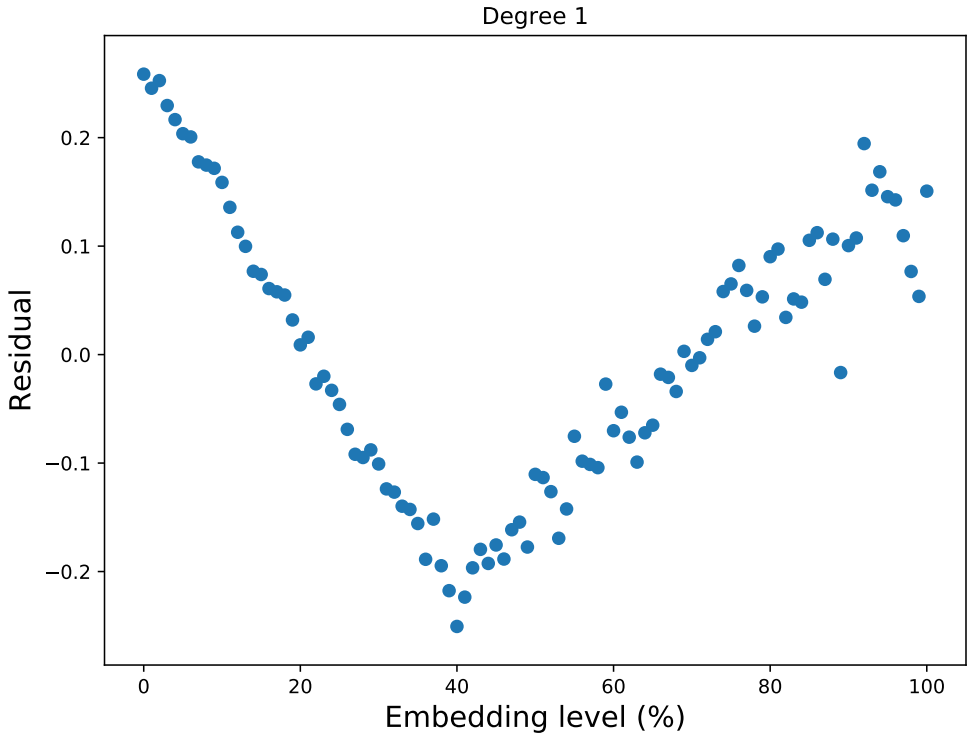
B.2 Degree 1 residual plots of 40% embedded datasets

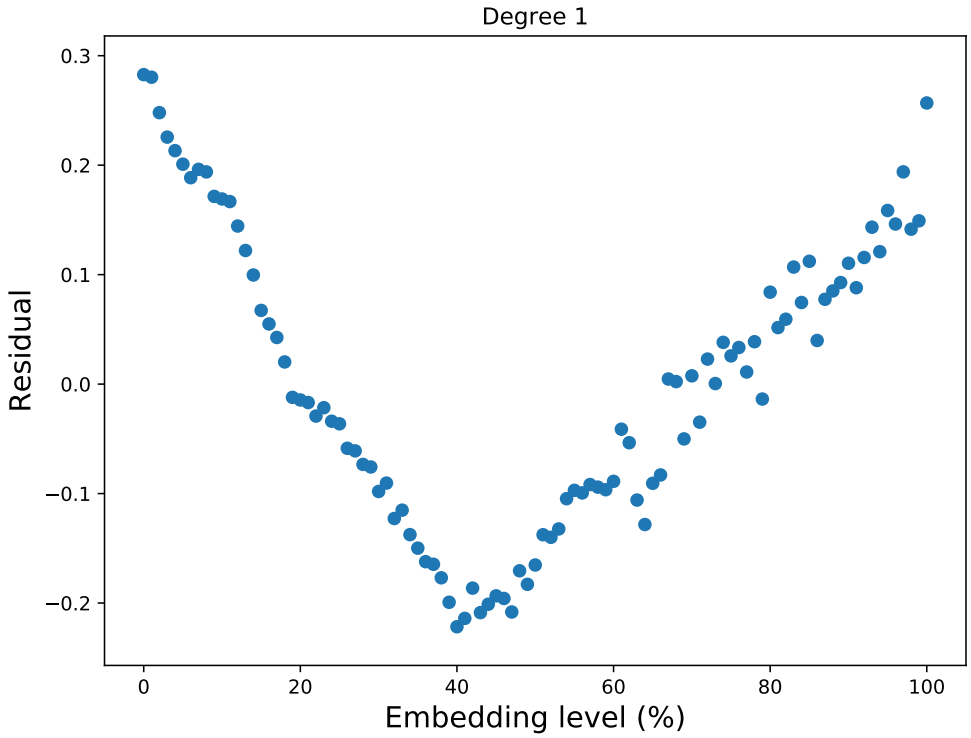
This section presents eight degree 1 residual approach results on 40% embedded datasets. The V-shaped curve is evident from all the datasets. As may be seen from each of the figures, the minimum residual value approximately marks the source embedding level.

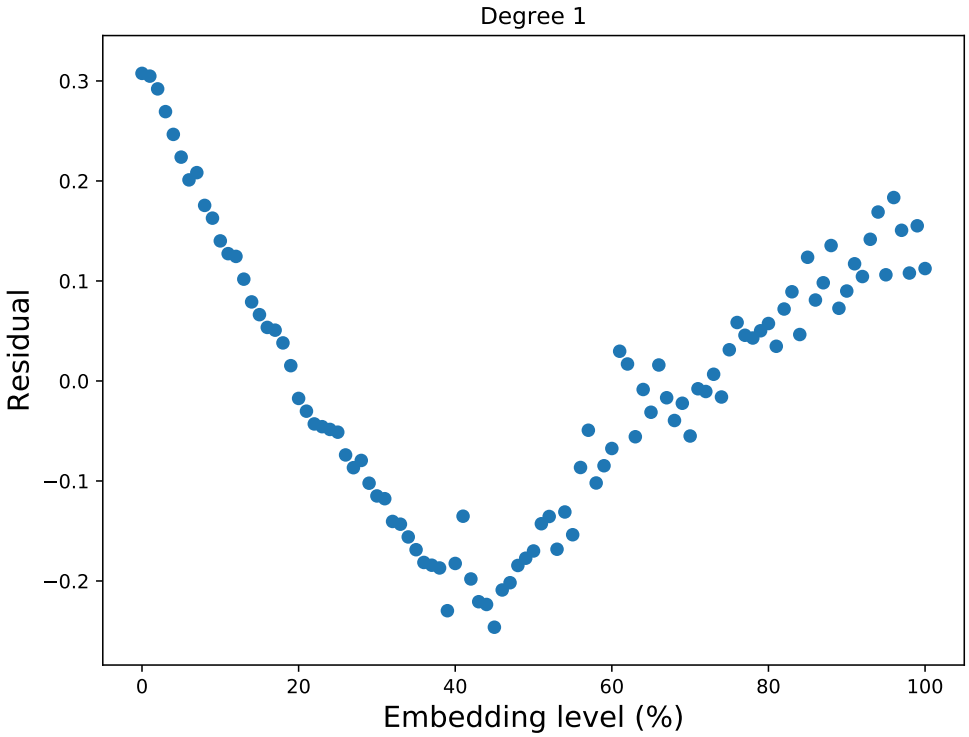


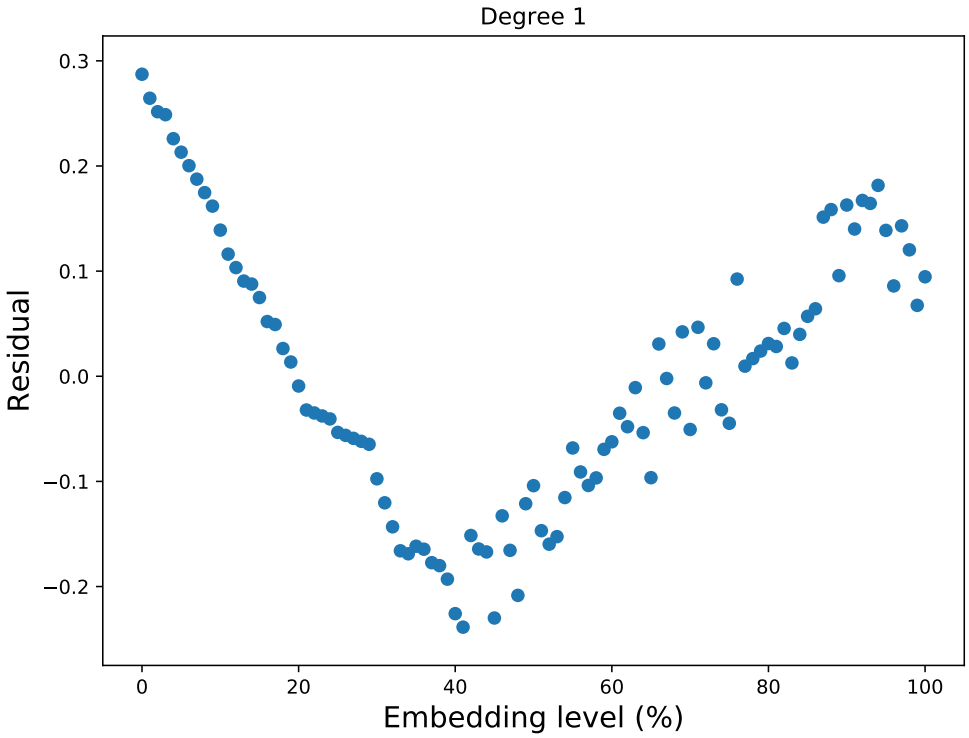

Degree 1
Residual
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

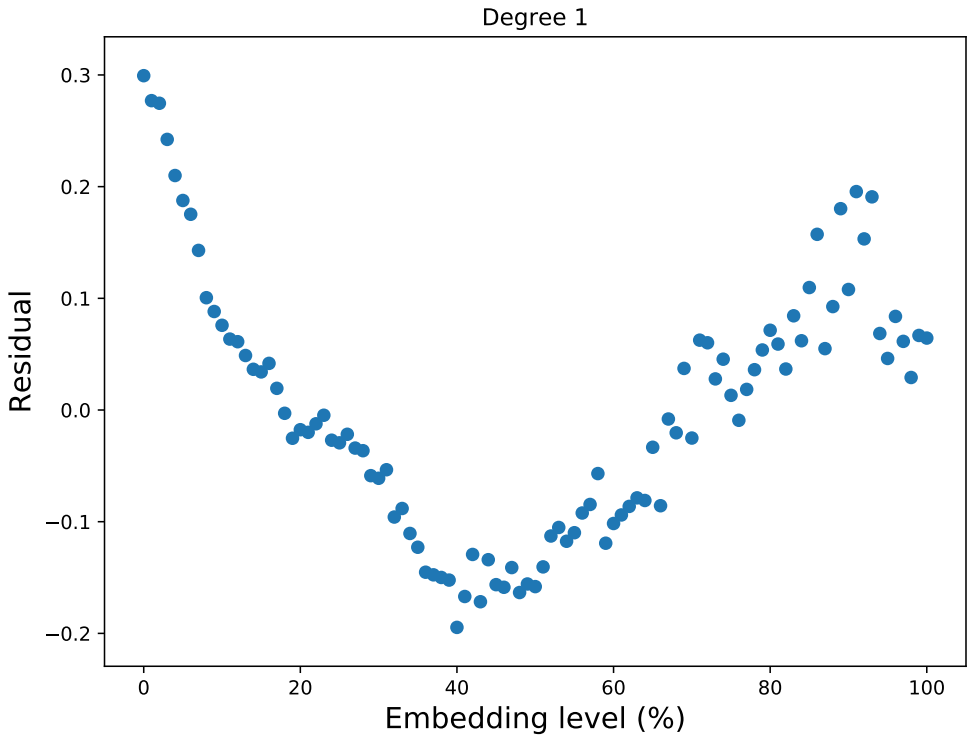

Degree 1
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)


Degree 1
Residual
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)


Degree 1
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

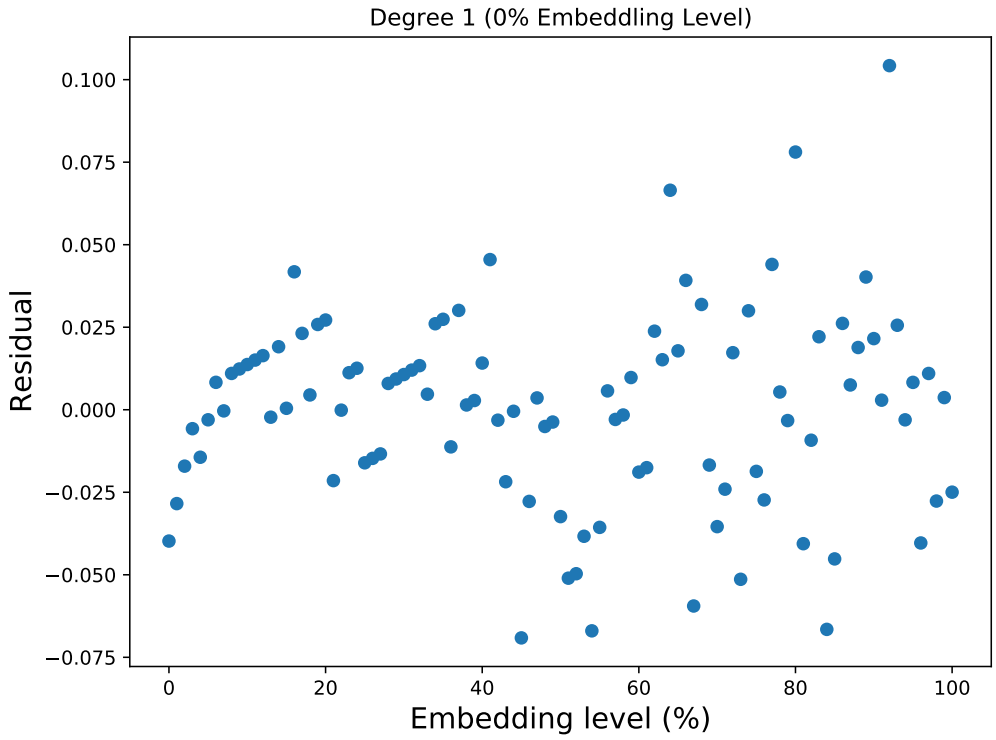

Degree 1
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

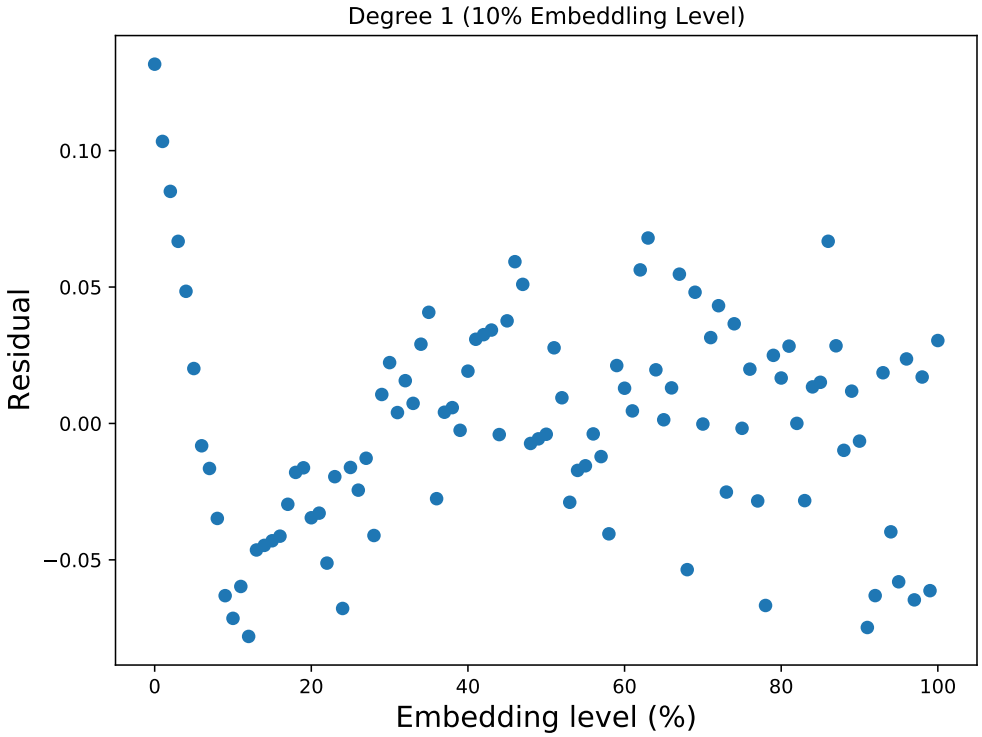

Degree 1
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

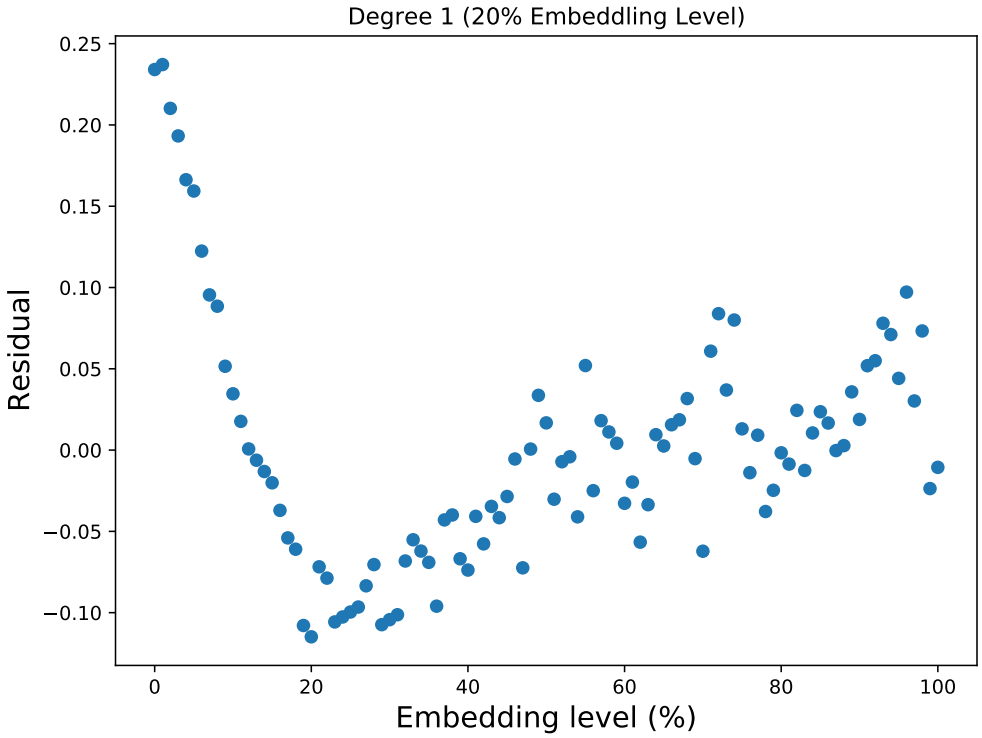

Degree 1
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

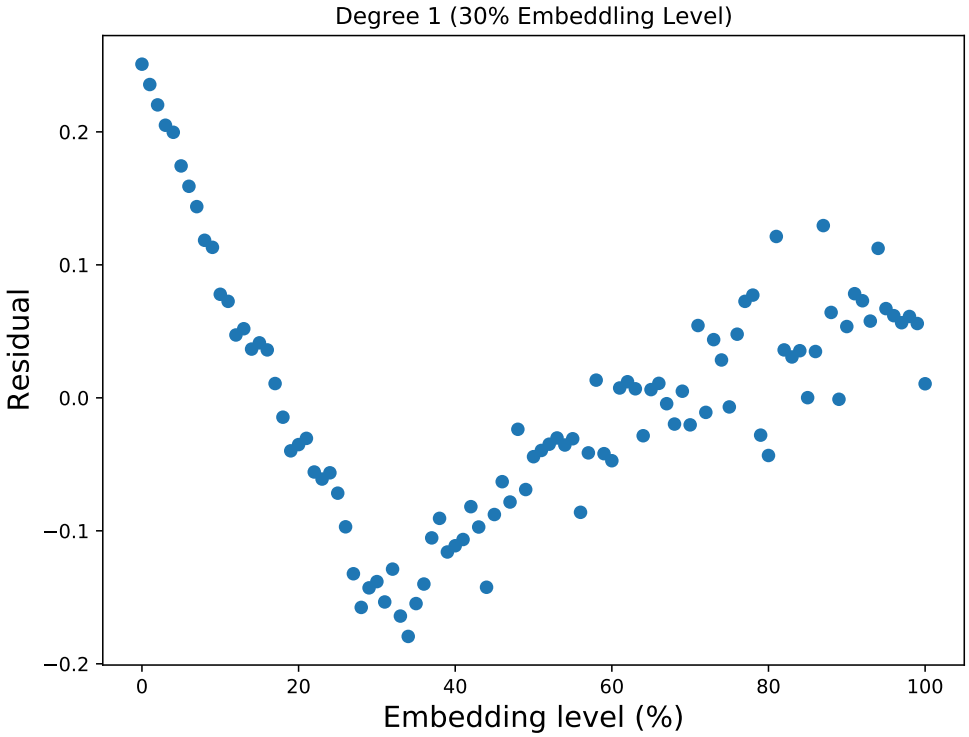
B.3 Degree 1 residual approach with varying embedding levels

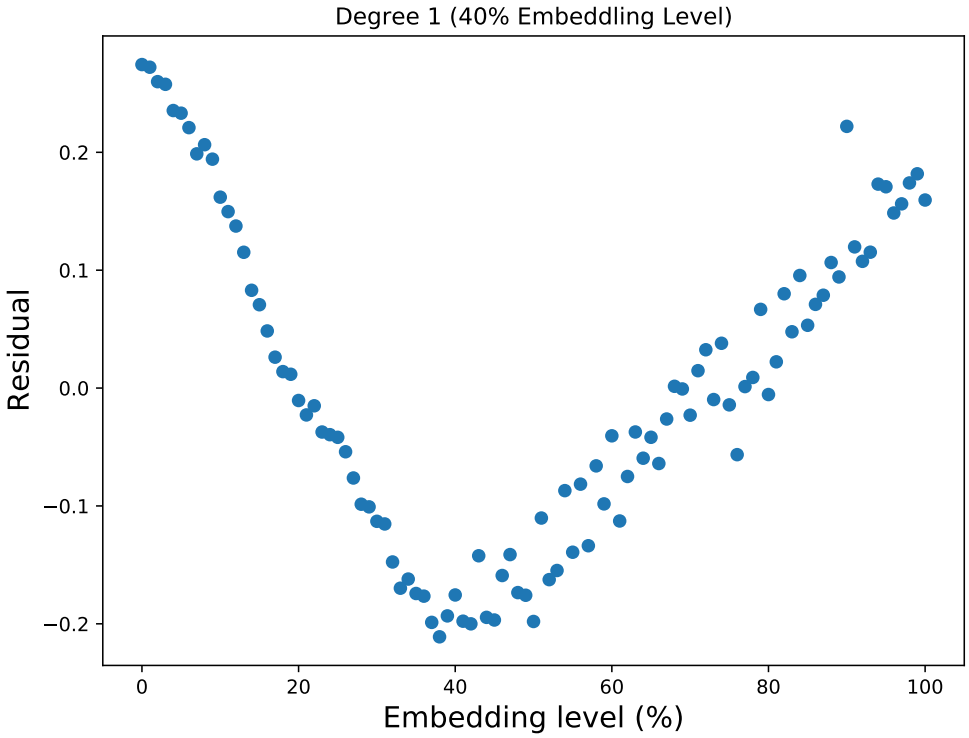
This section shows graphs with a degree 1 residual approach where embedding levels vary from 0 to 100. The V-shaped curves are not evident in low and high embedding levels. The V-shaped curve becomes more evident as the embedding level increases until it starts to disperse from the 80% embedding level.

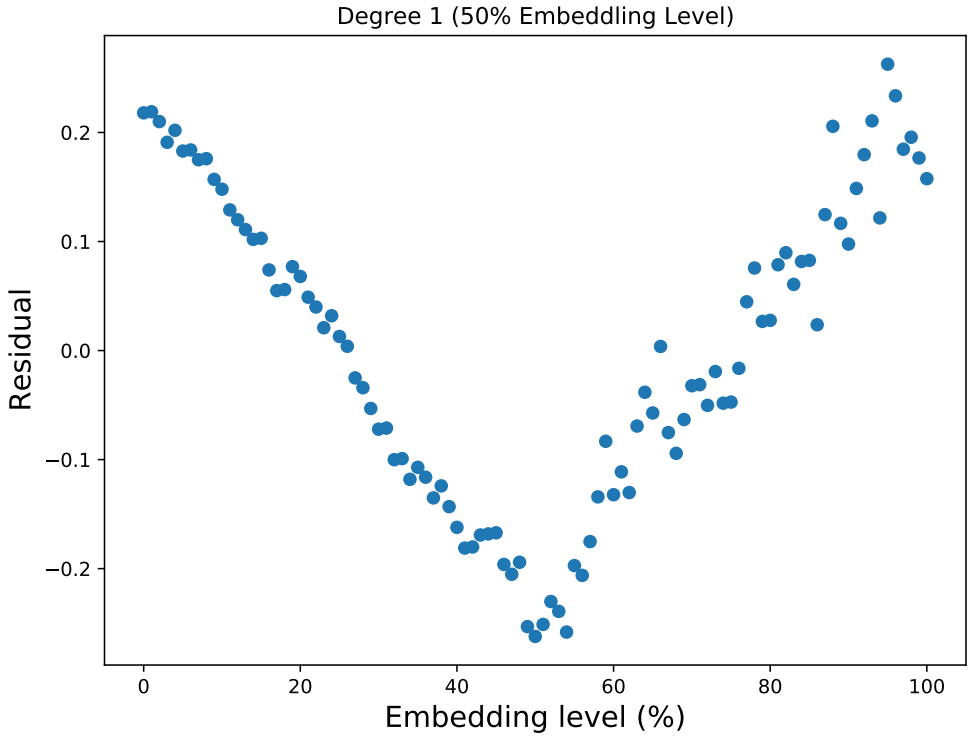


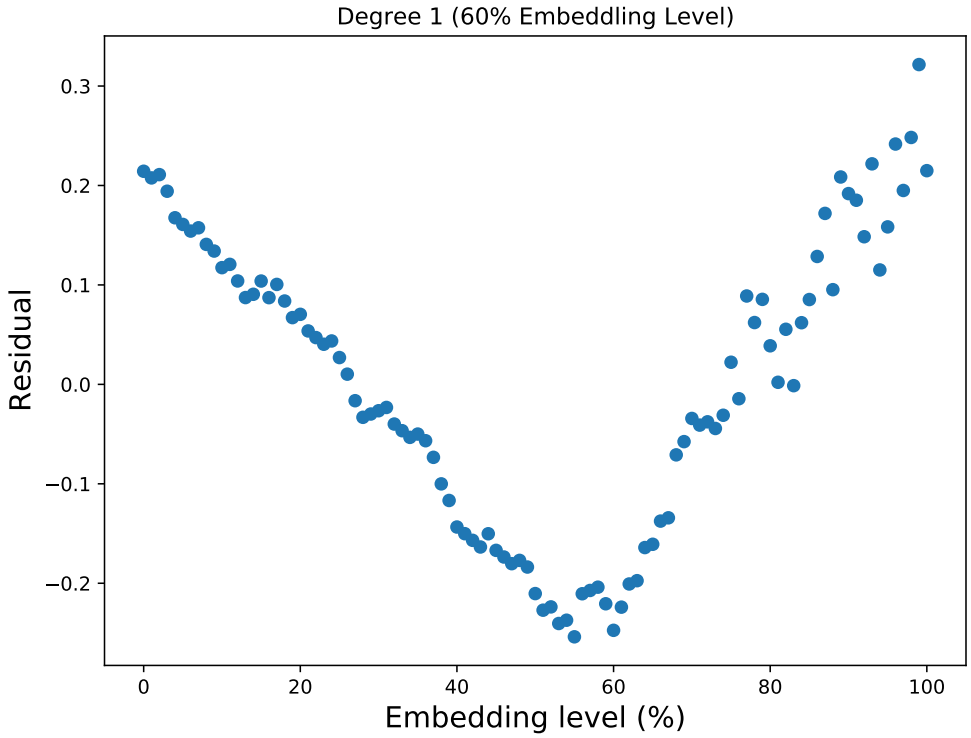

Degree 1 (10% Embeddling Level)
Residual
0.10
0.05
0.00
−0.05
0
20
40
60
80
100
Embedding level (%)

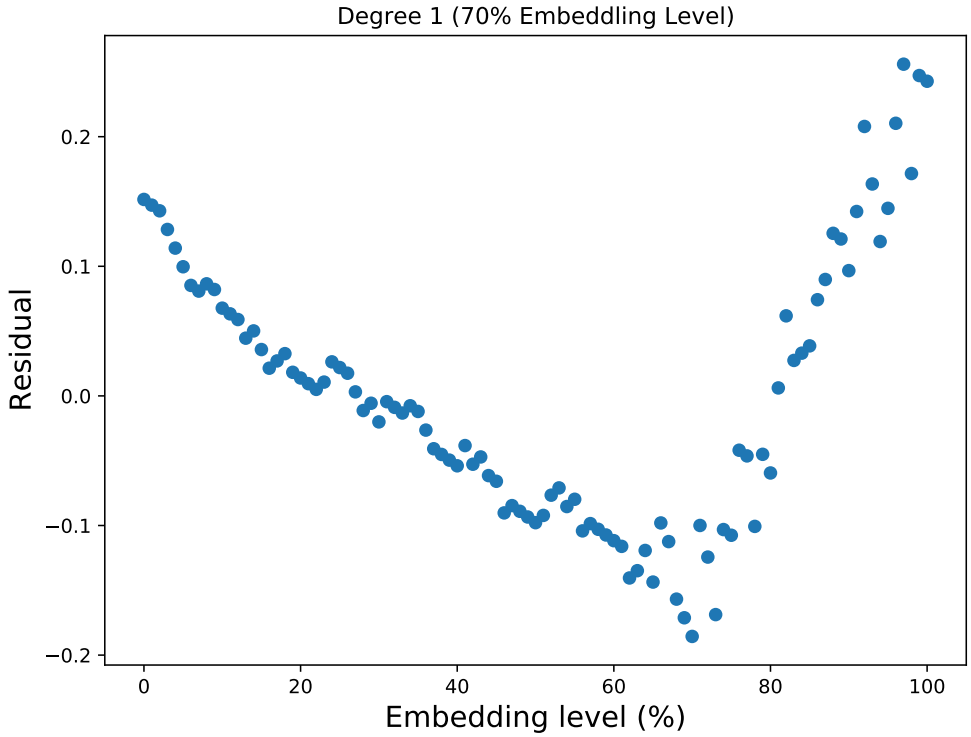

Degree 1 (20% Embeddling Level)
Residual
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
0
20
40
60
80
100
Embedding level (%)

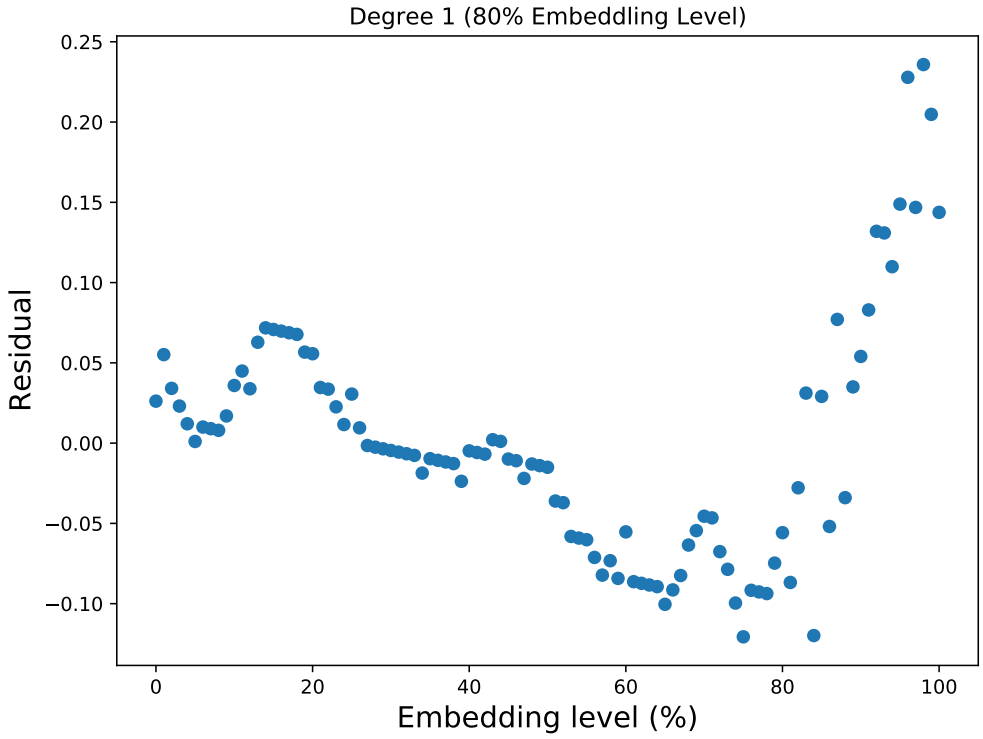

Degree 1 (30% Embeddling Level)
Residual
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

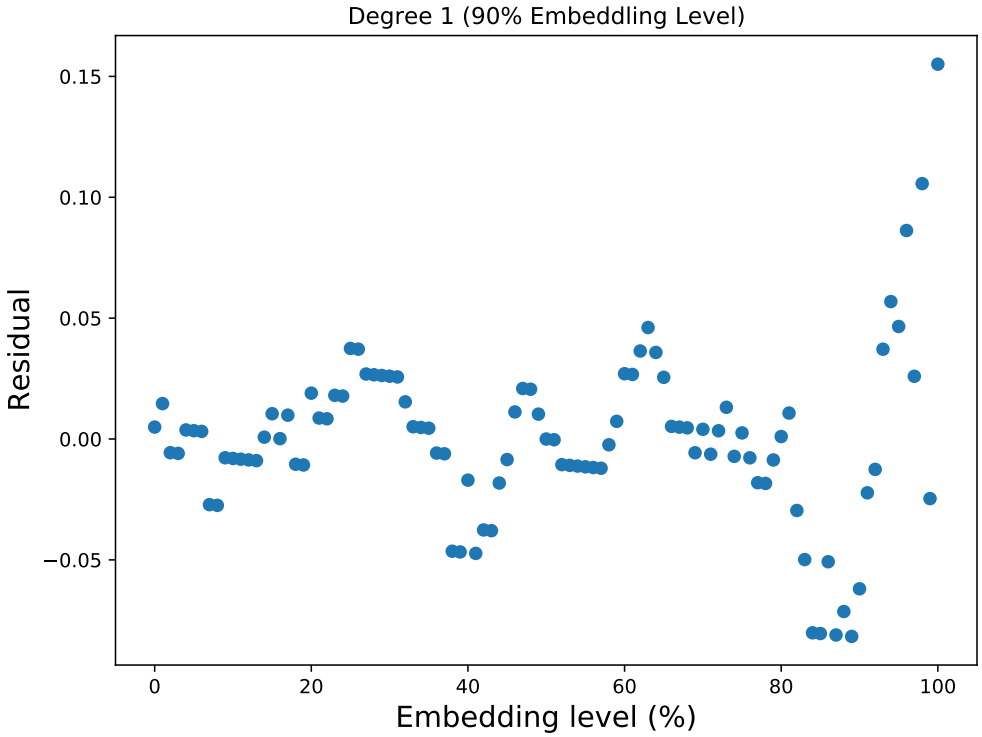

Degree 1 (40% Embeddling Level)
Residual
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

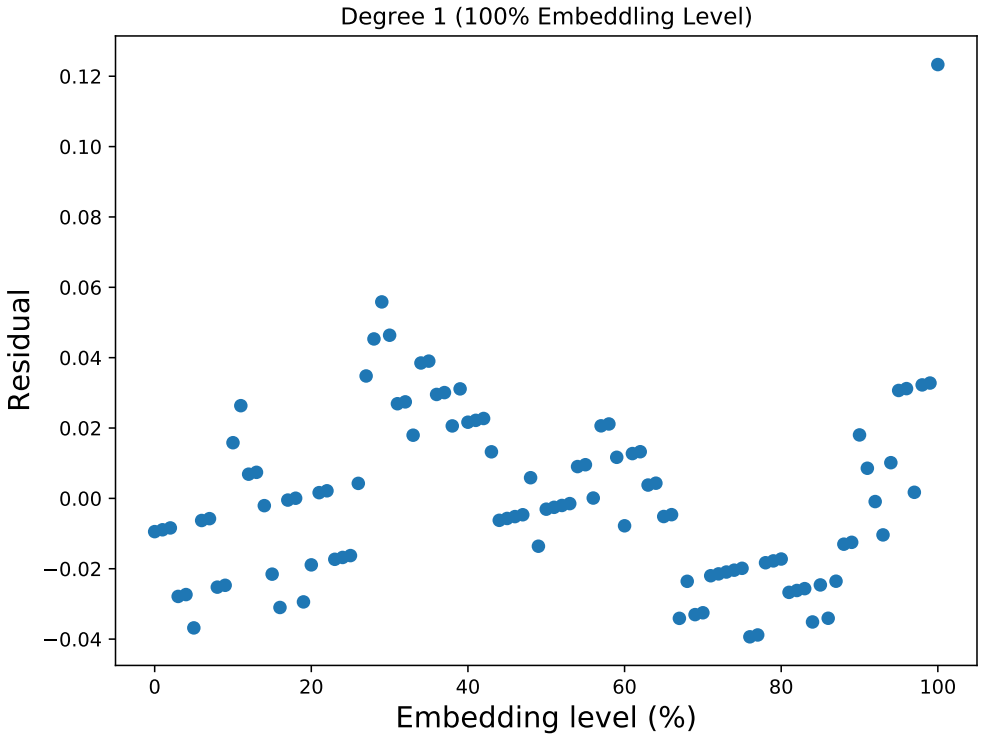

Degree 1 (50% Embeddling Level)
Residual
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)


Degree 1 (60% Embeddling Level)
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)


Degree 1 (70% Embeddling Level)
Residual
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

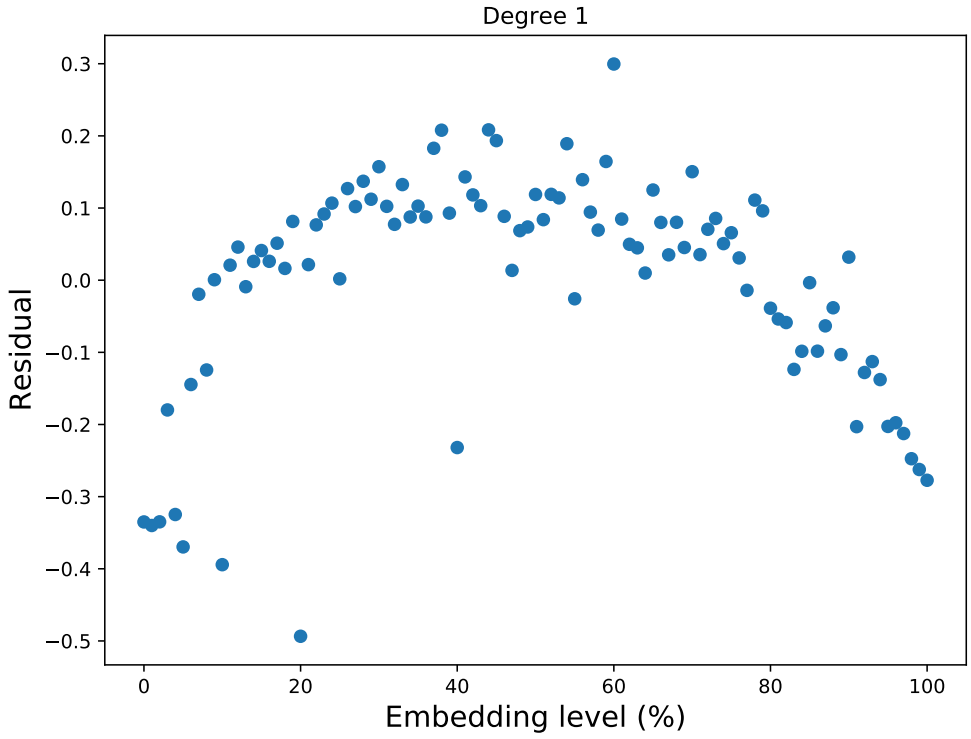

Degree 1 (80% Embeddling Level)
Residual
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
0
20
40
60
80
100
Embedding level (%)

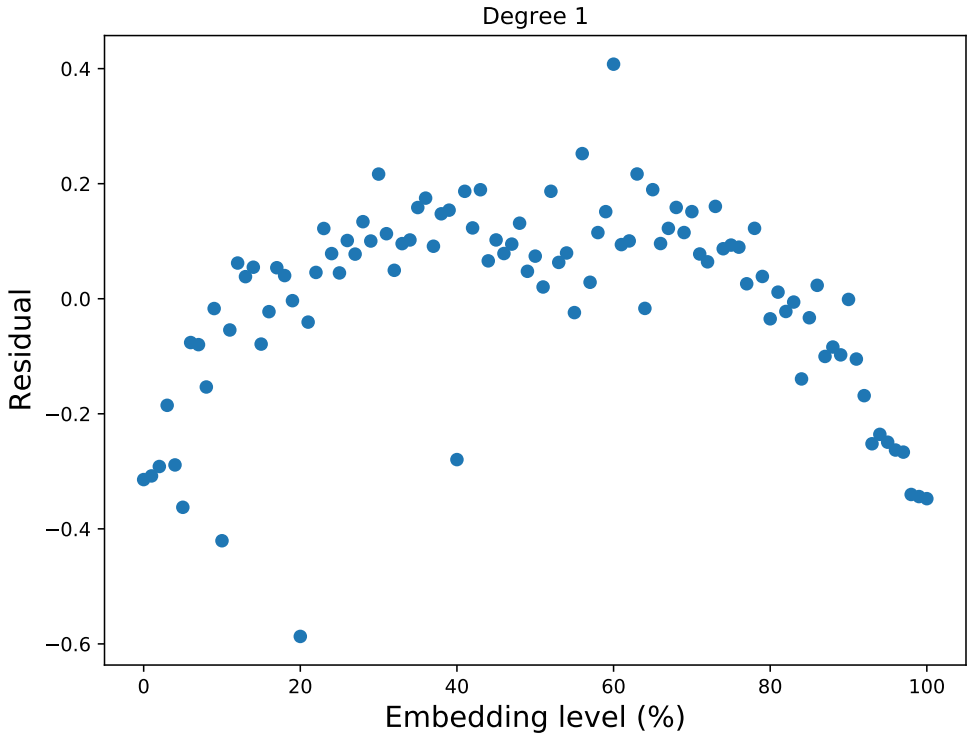

Degree 1 (90% Embeddling Level)
Residual
0.15
0.10
0.05
0.00
−0.05
0
20
40
60
80
100
Embedding level (%)

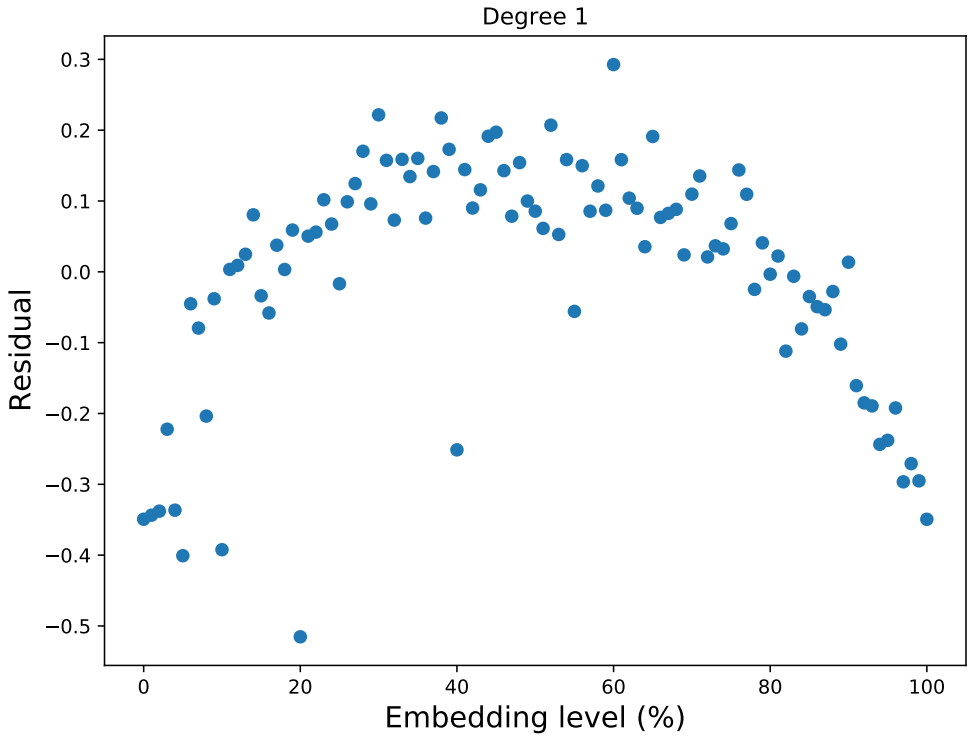

Degree 1 (100% Embeddling Level)
Residual
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
0
20
40
60
80
100
Embedding level (%)

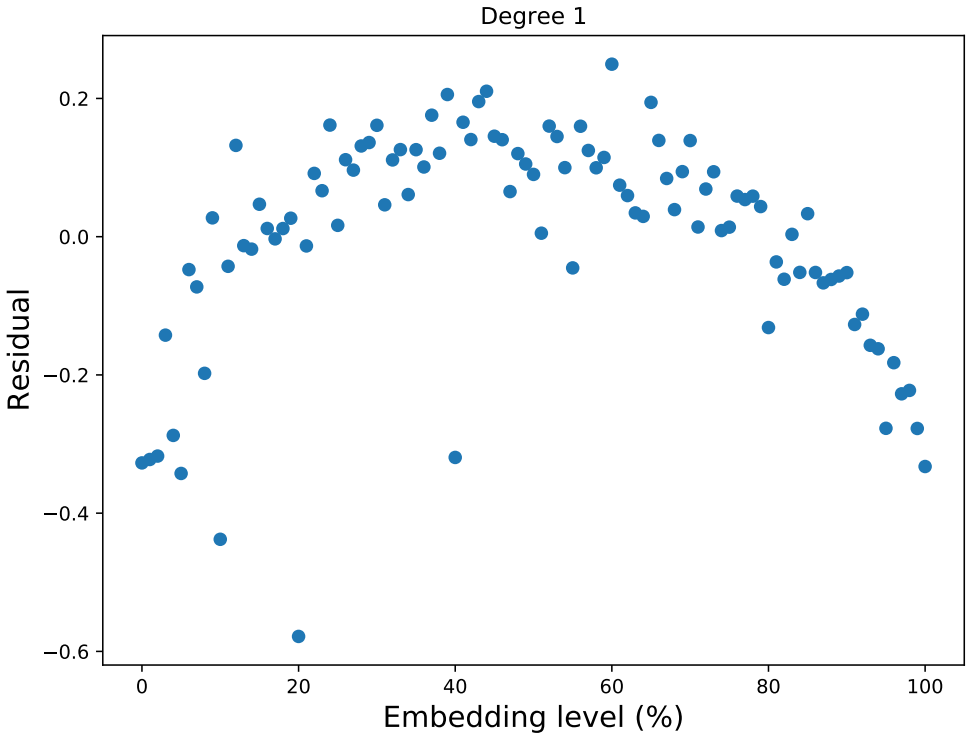
B.4 Degree 1 residual plots of 20% embedded (equal spacing, equal spacing) datasets

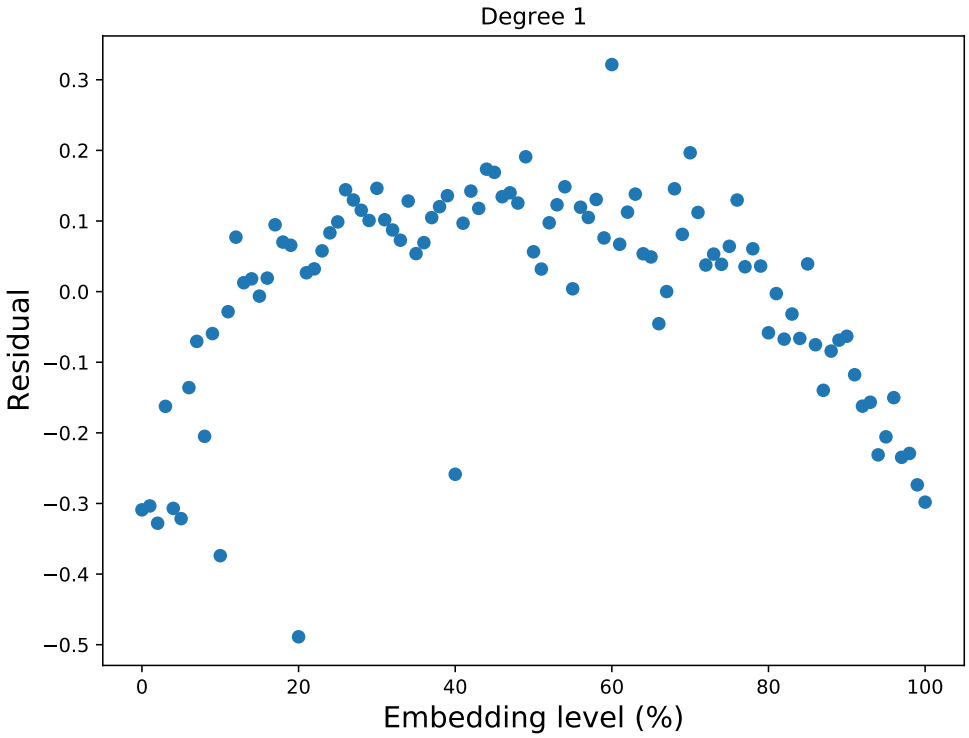
This section presents eight 20% embedded (equal spacing, equal spacing) datasets to illustrate the minimum residual marks the source embedding level.

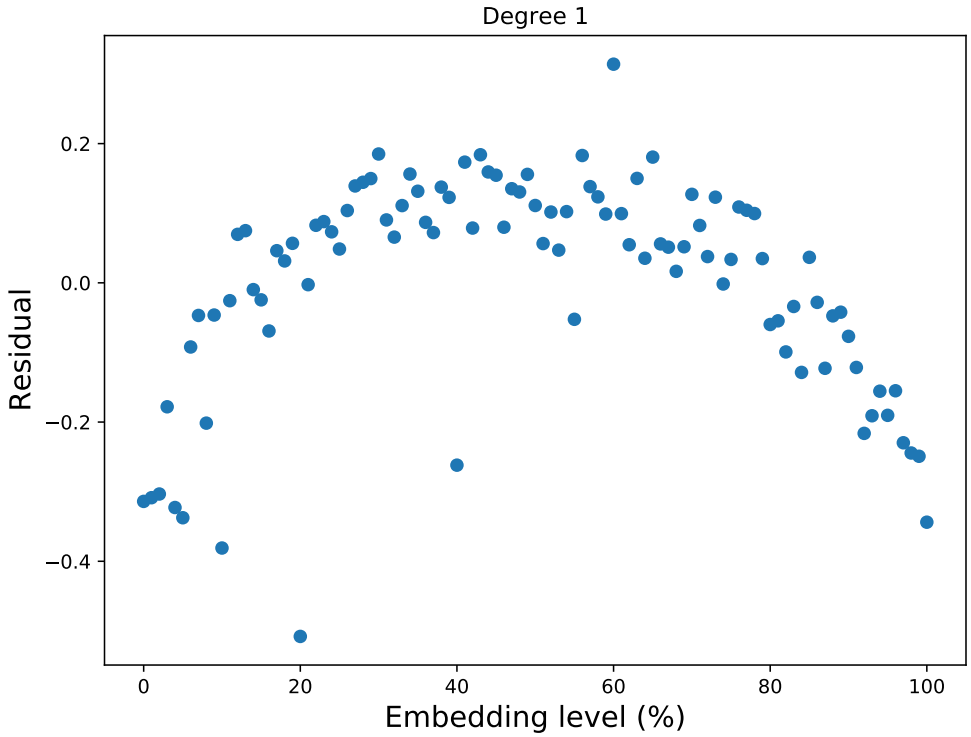


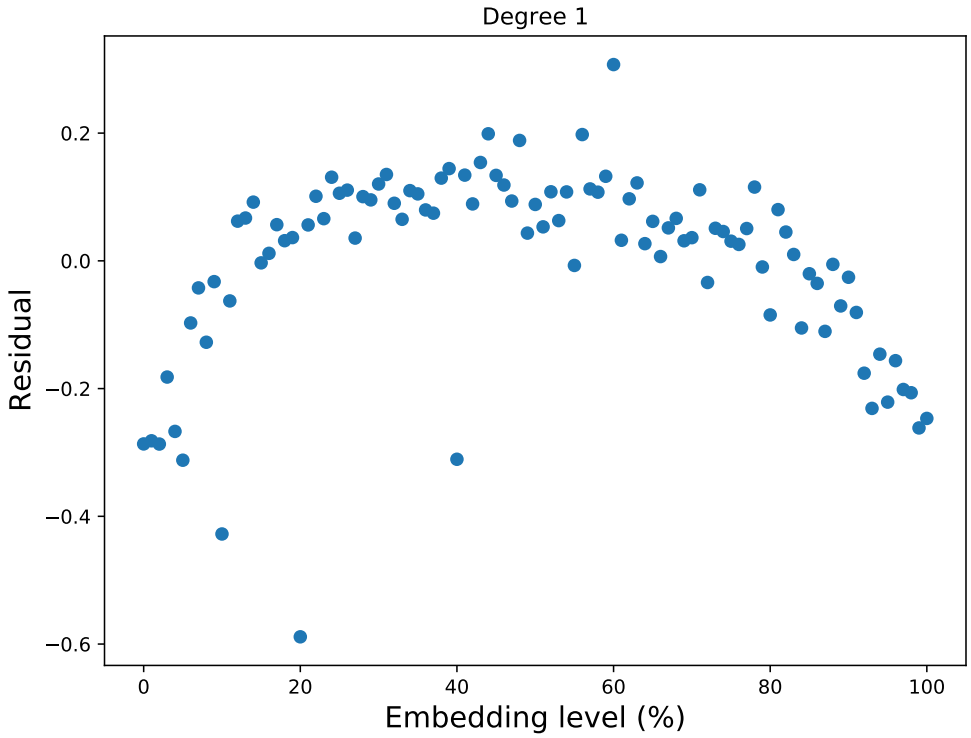

Degree 1
Residual
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
100
Embedding level (%)

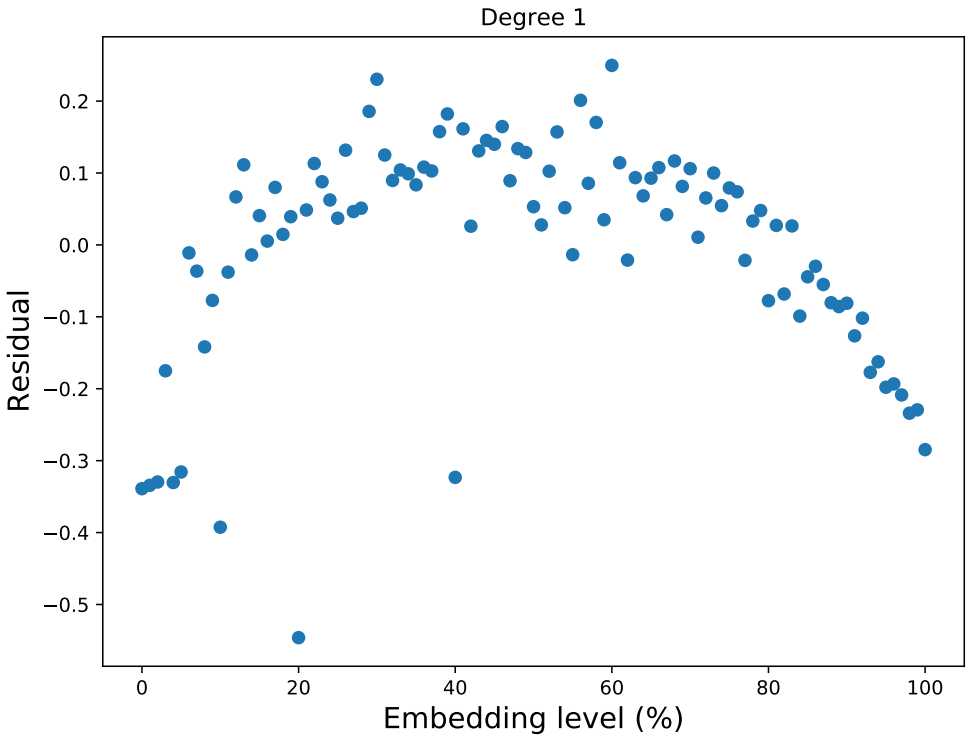

Degree 1
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
0
20
40
60
80
100
Embedding level (%)


Degree 1
Residual
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
100
Embedding level (%)


Degree 1
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
0
20
40
60
80
100
Embedding level (%)

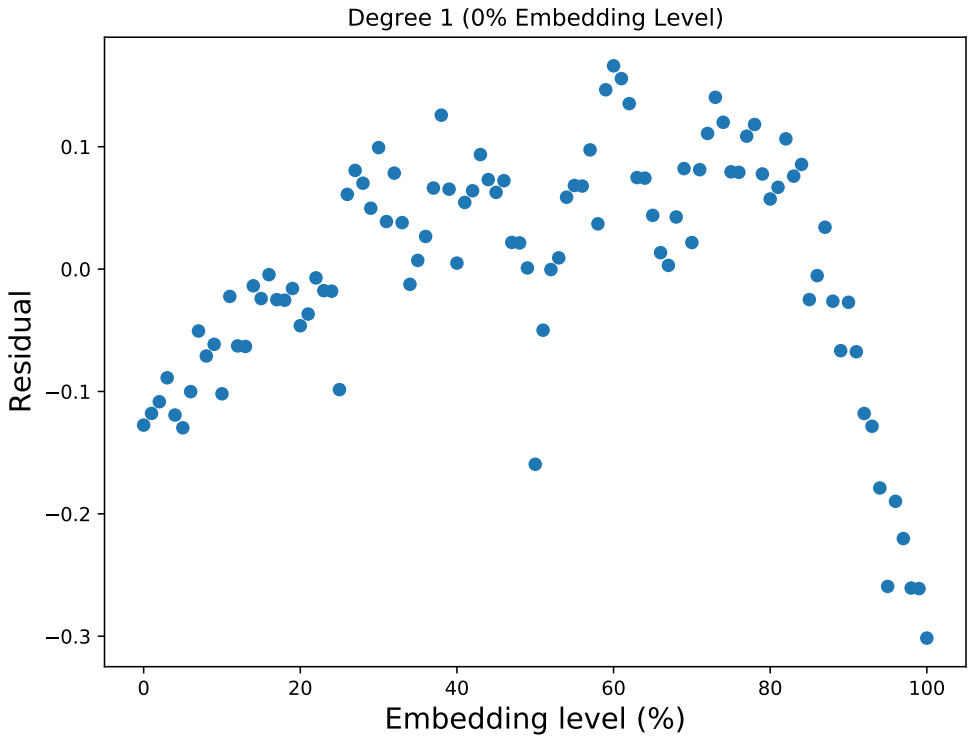

Degree 1
Residual
0.2
0.0
−0.2
−0.4
0
20
40
60
80
100
Embedding level (%)

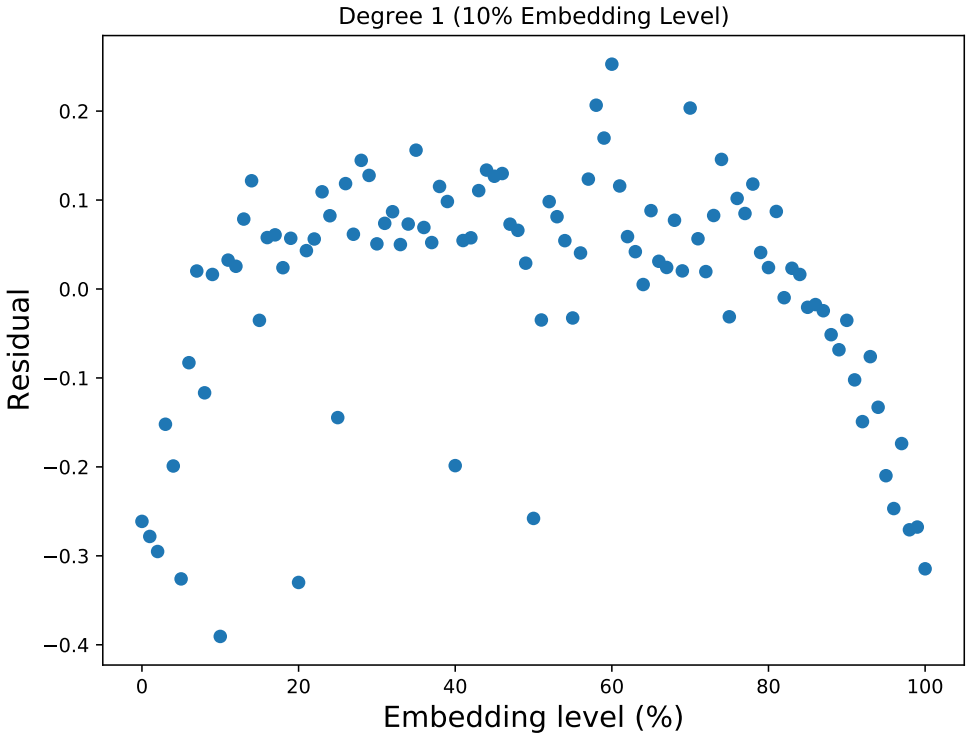

Degree 1
Residual
0.2
0.0
−0.2
−0.4
−0.6
0
20
40
60
80
100
Embedding level (%)

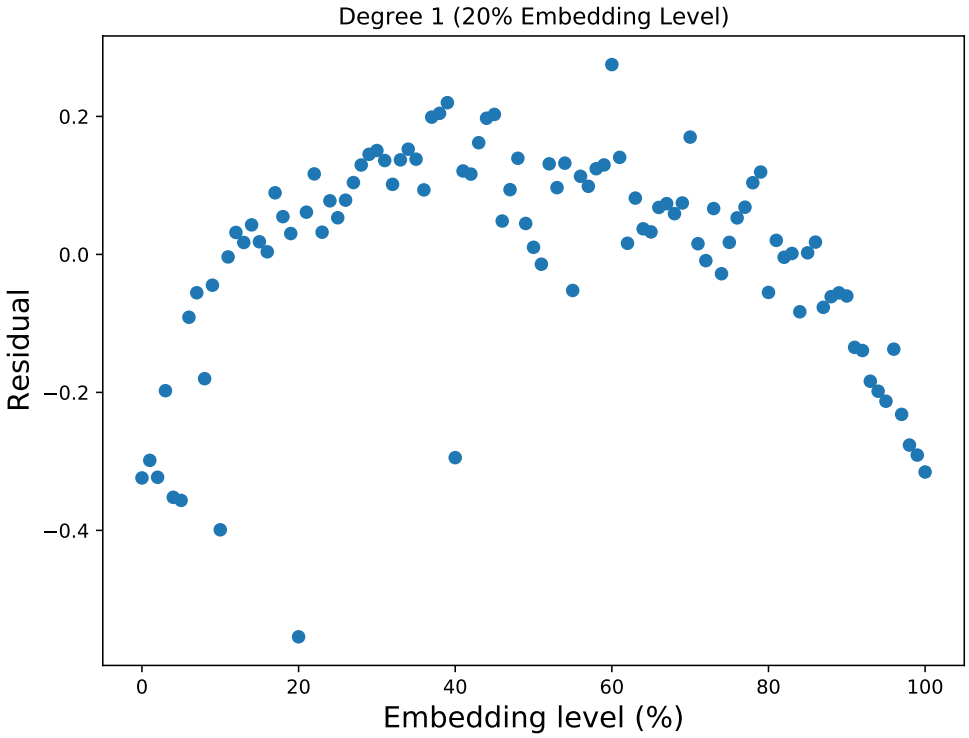

Degree 1
Residual
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
0
20
40
60
80
100
Embedding level (%)

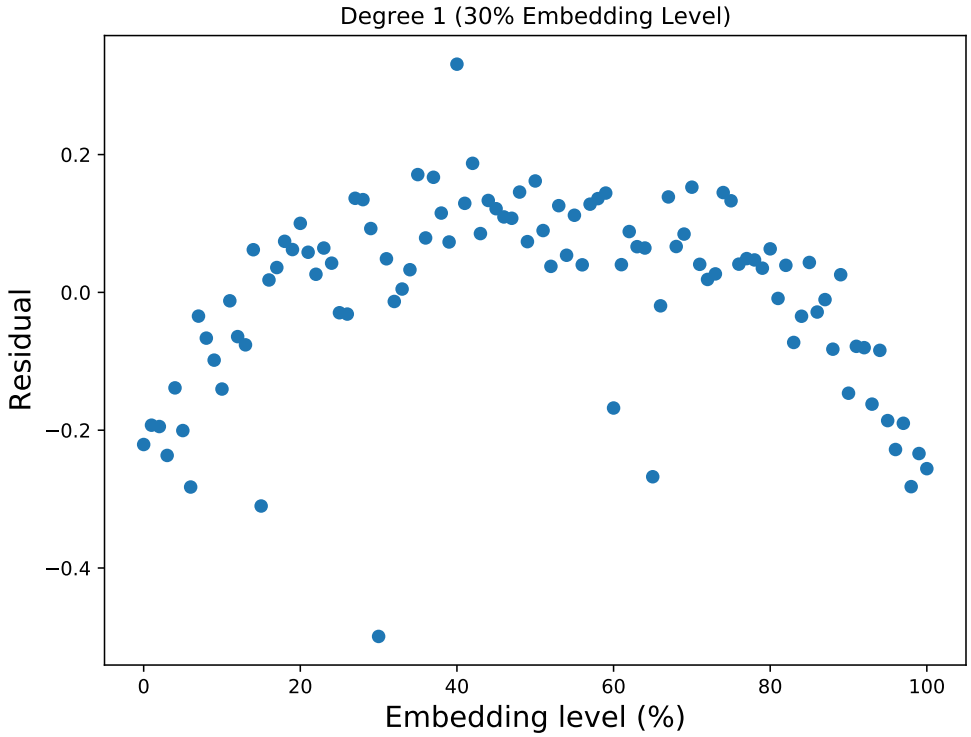
B.5 Degree 1 residual approach with varying embedding levels on (equal spacing, equal spacing) datasets

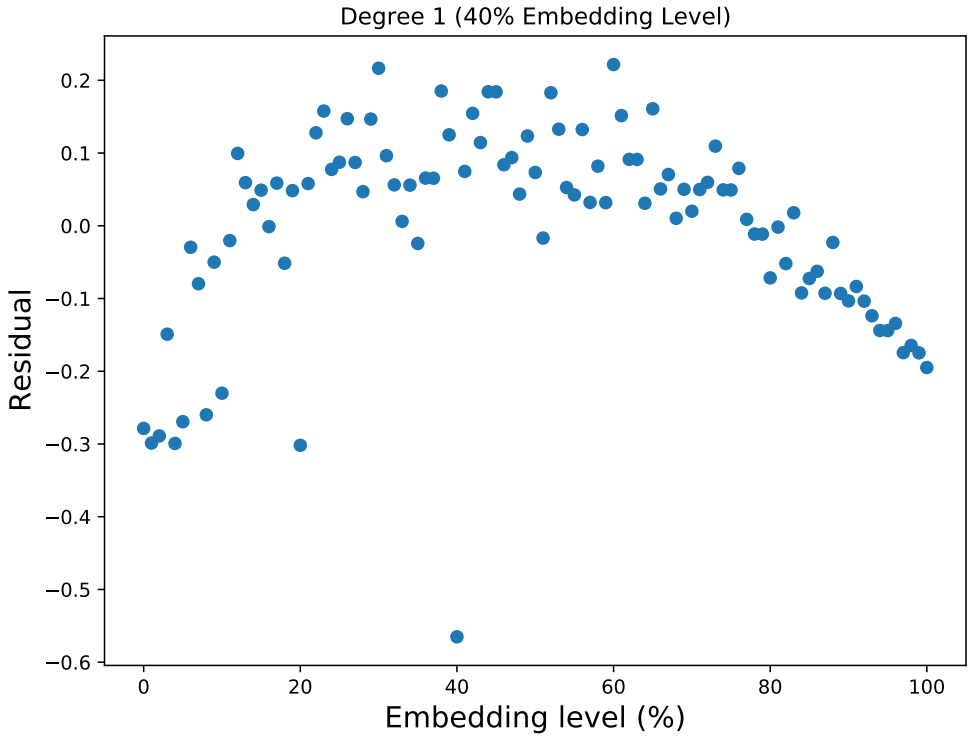
This section shows graphs with a degree 1 residual approach where embedding levels vary from 0 to 100. In many instances, minimum residual correctly indicates the source embedding level.

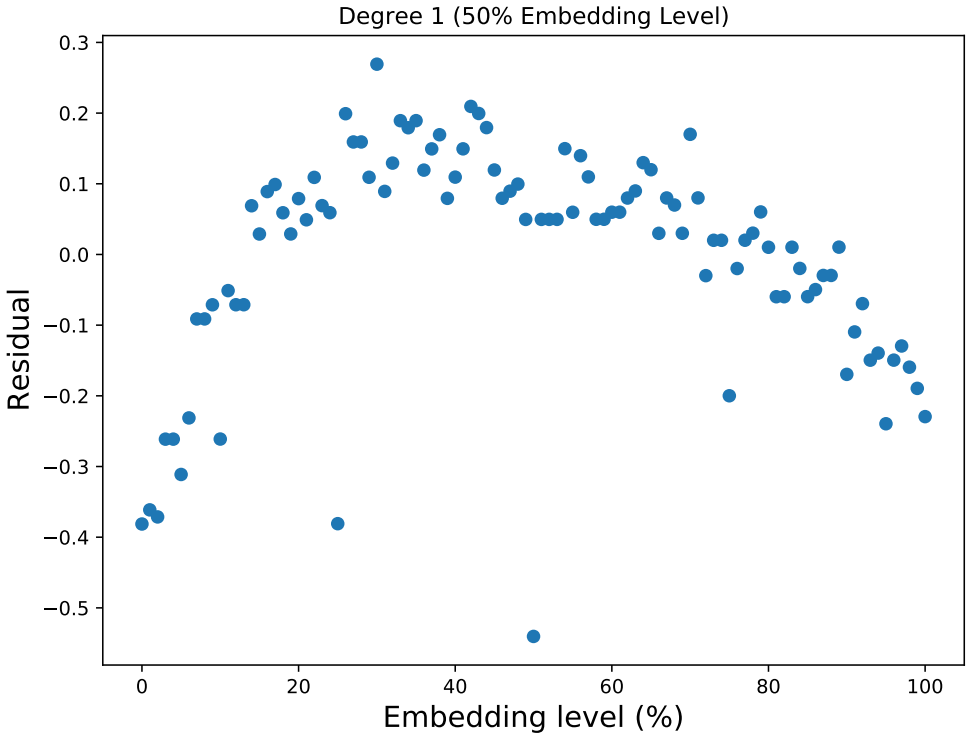


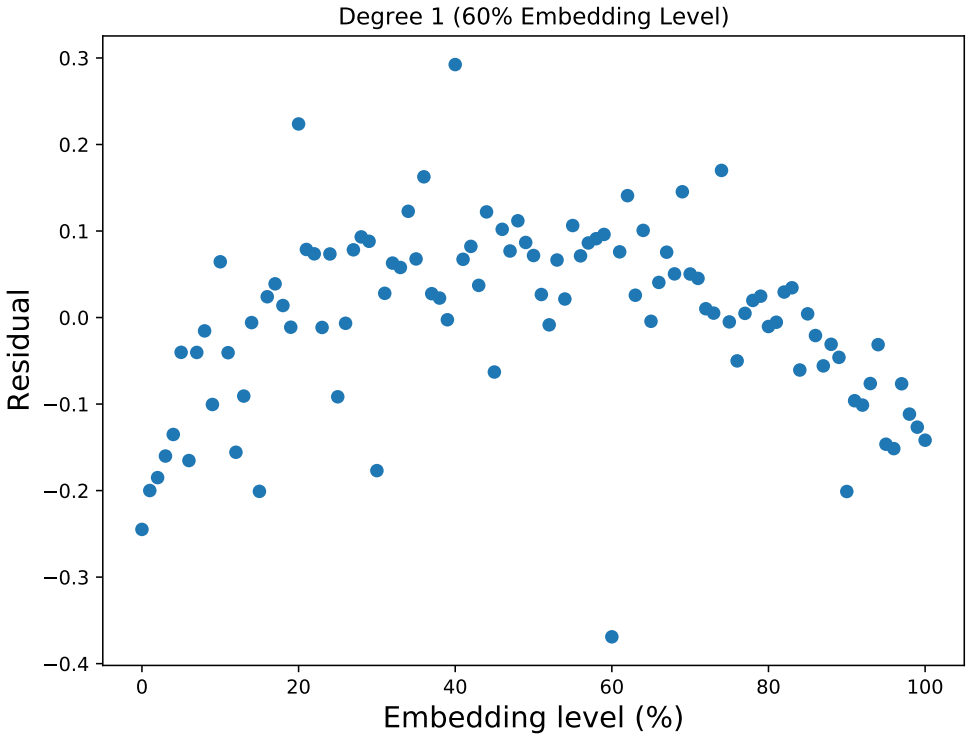

Degree 1 (10% Embedding Level)
Residual
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
20
40
60
80
100
Embedding level (%)

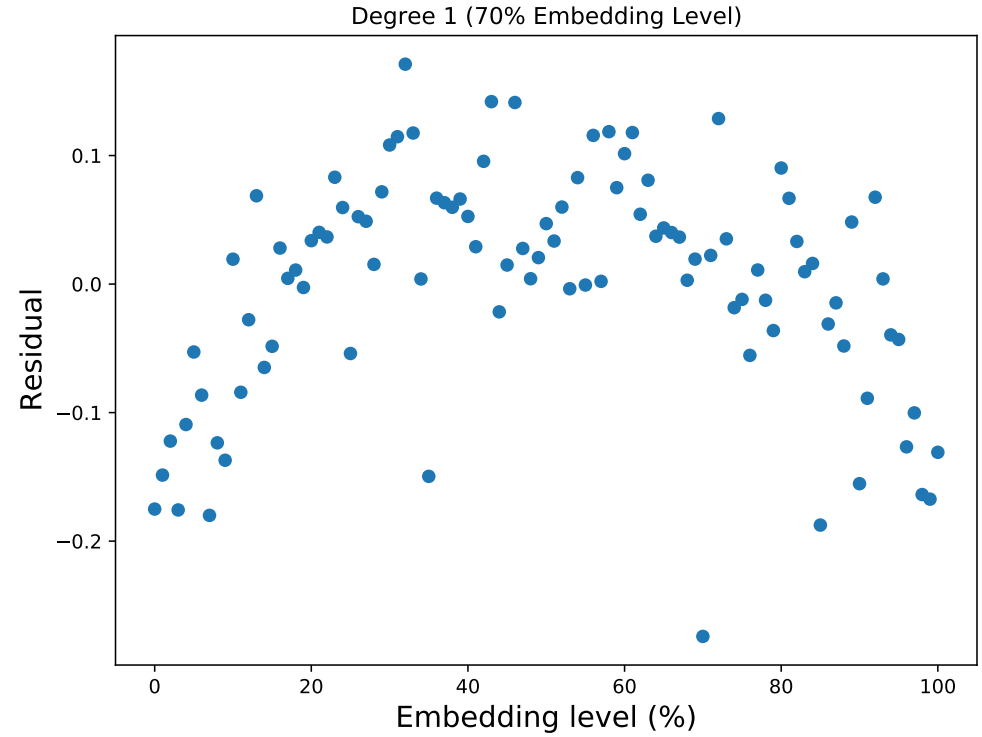

Degree 1 (20% Embedding Level)
Residual
0.2
0.0
−0.2
−0.4
0
20
40
60
80
100
Embedding level (%)

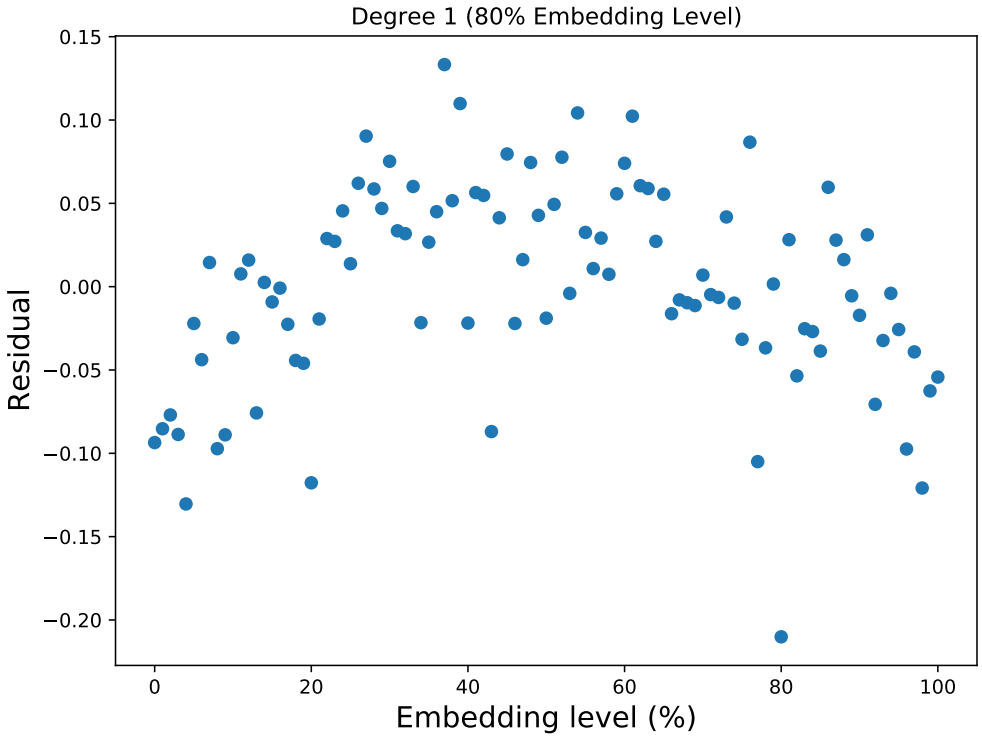

Degree 1 (30% Embedding Level)
Residual
0.2
0.0
−0.2
−0.4
0
20
40
60
80
100
Embedding level (%)

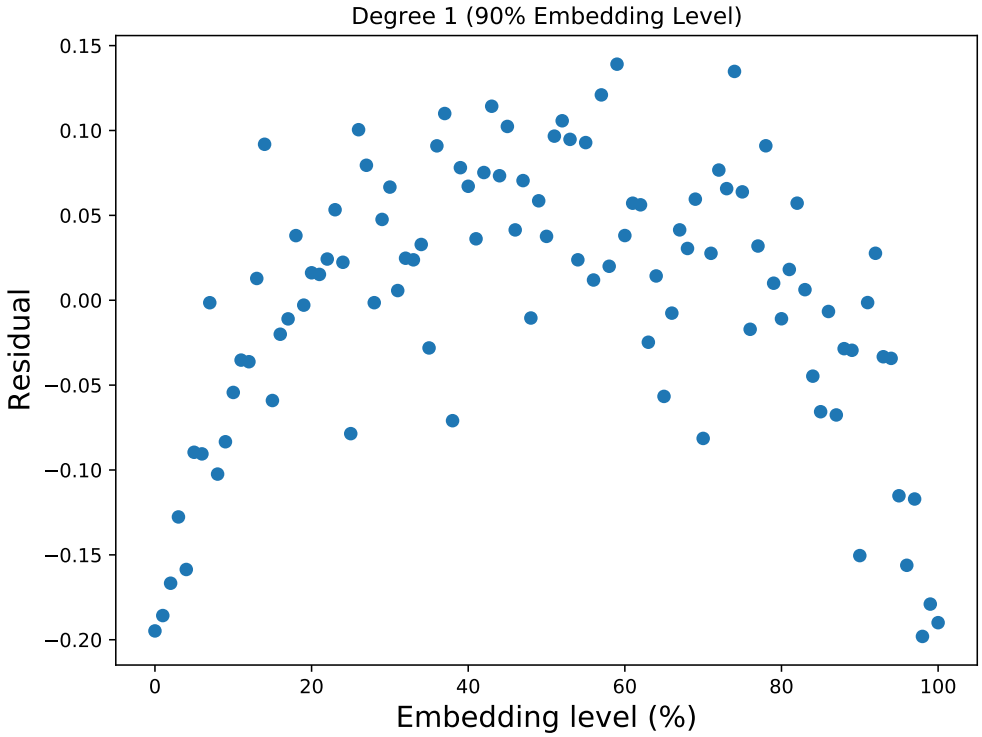

Degree 1 (40% Embedding Level)
Residual
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
0
20
40
60
80
100
Embedding level (%)

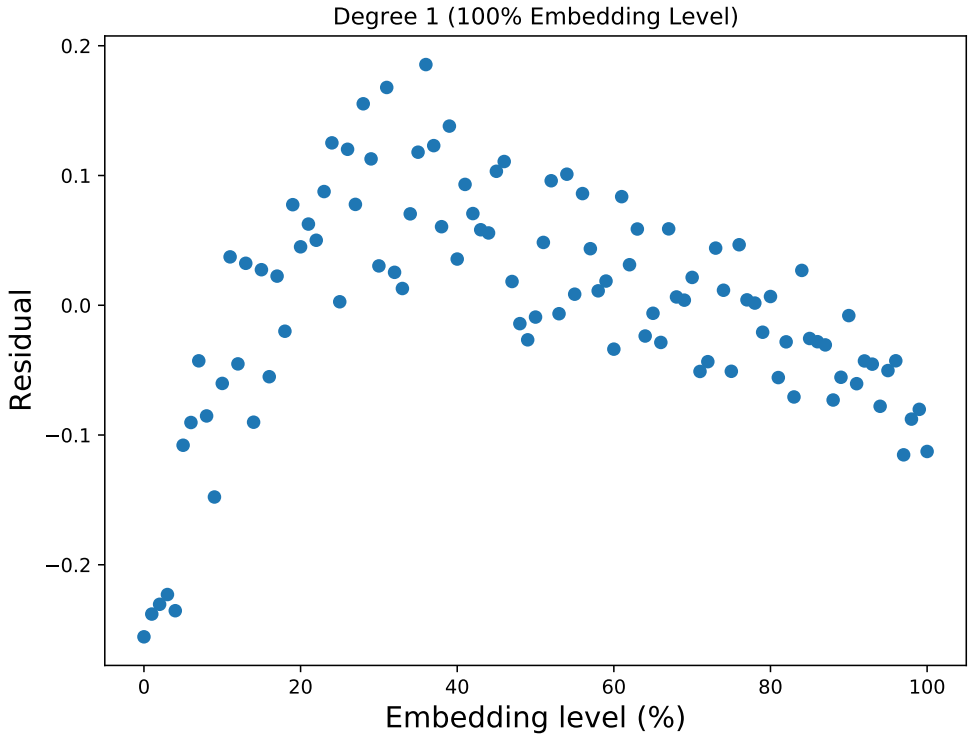

Degree 1 (50% Embedding Level)
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
0
20
40
60
80
100
Embedding level (%)


Degree 1 (60% Embedding Level)
Residual
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
20
40
60
80
100
Embedding level (%)


Degree 1 (70% Embedding Level)
Residual
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)


Degree 1 (80% Embedding Level)
Residual
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0
20
40
60
80
100
Embedding level (%)


Degree 1 (90% Embedding Level)
Residual
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0
20
40
60
80
100
Embedding level (%)

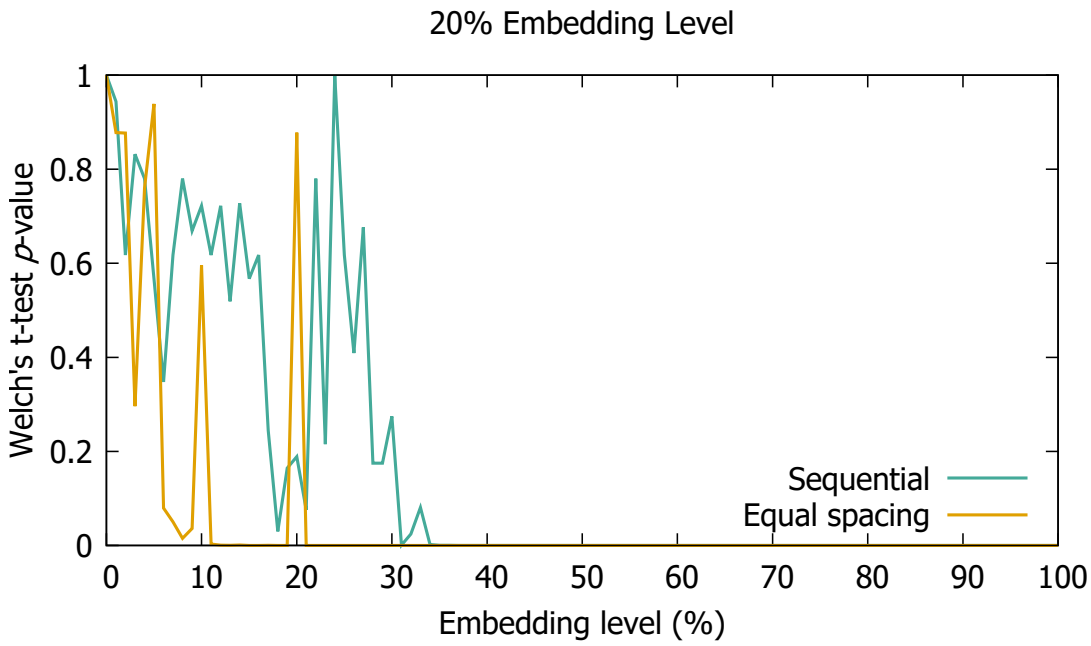

Degree 1 (100% Embedding Level)
Residual
0.2
0.1
0.0
−0.1
−0.2
0
20
40
60
80
100
Embedding level (%)

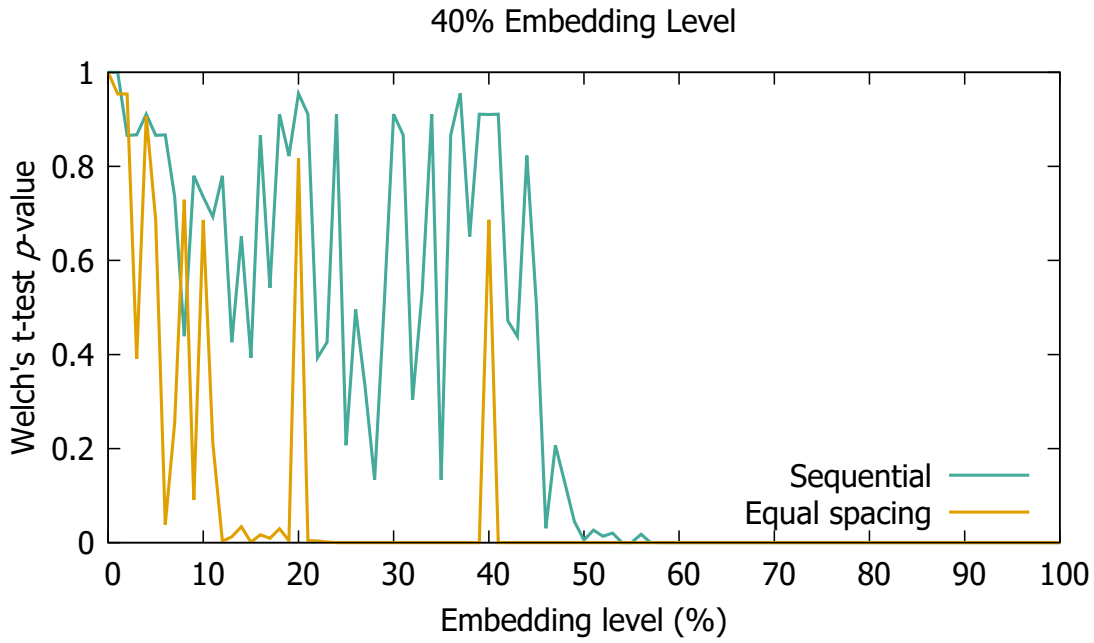
C

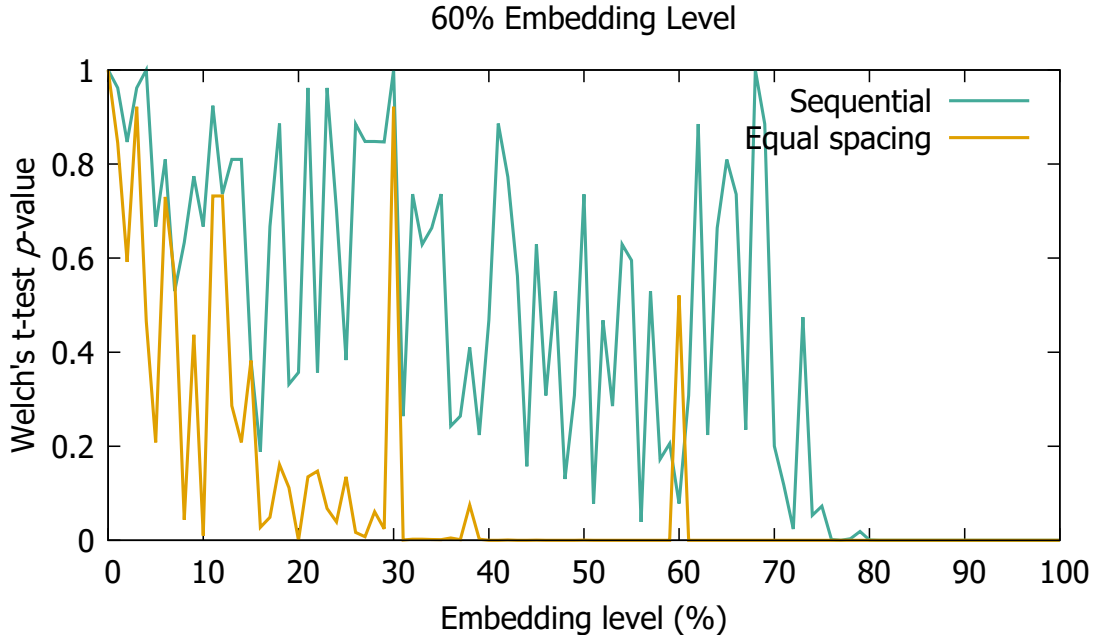
Additional Graphs for Other Metrics

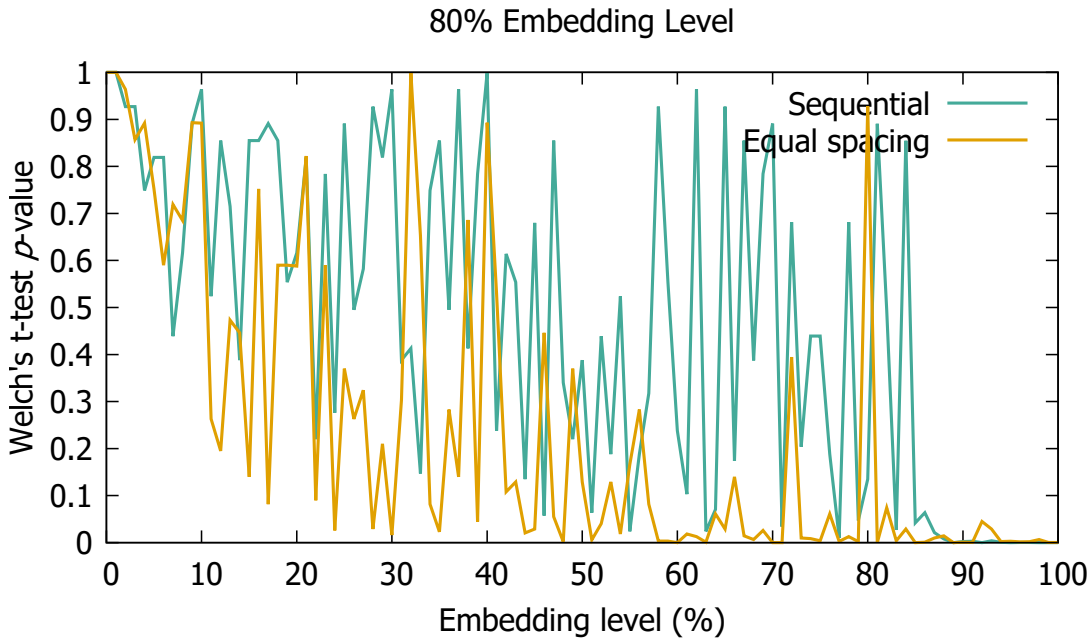
C.1 Welch's t-test p-value graphs

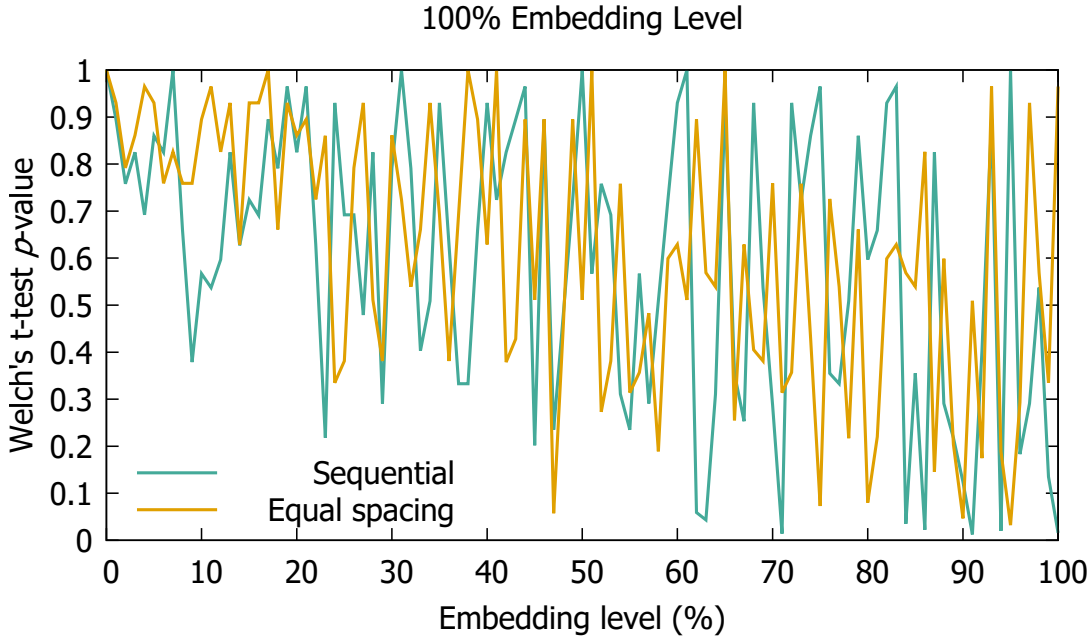
This section shows Welch's t-test p-value graphs where the embedding levels vary from 0 to 100. As may be seen from the graphs, the p-value spikes when the source embedding level matches with the re-embedding level. Every graph contains a number of peaks, however, and there is not enough information to suggest which peak marks the correct source embedding level.

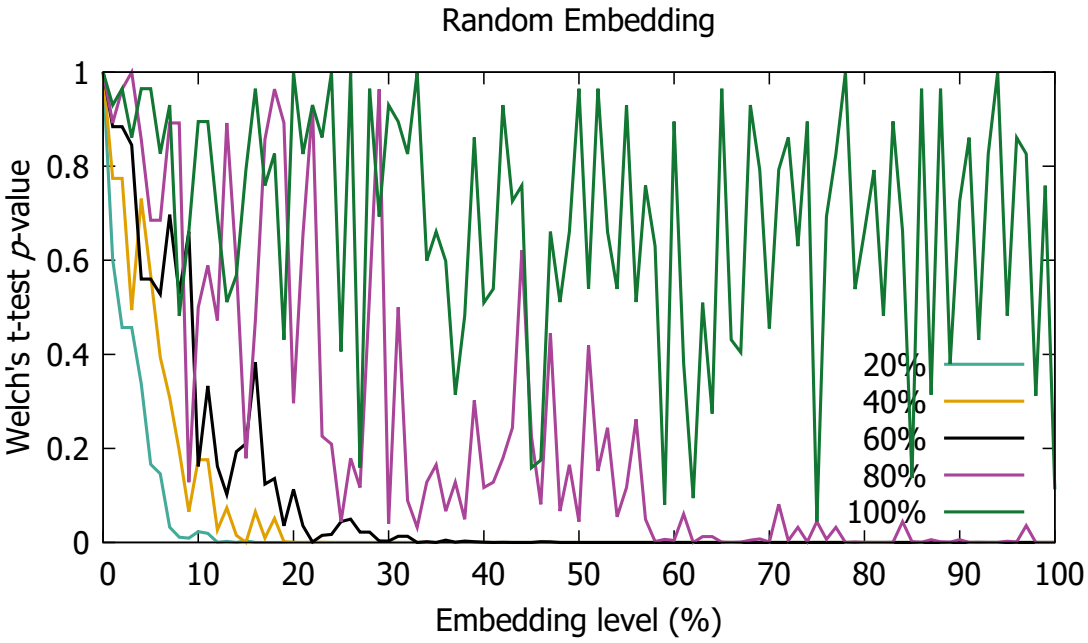

20% Embedding Level
Welch's t-test p-value
Embedding level (%)
Sequential
Equal spacing
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50
60
70
80
90
100


40% Embedding Level
Welch's t-test p-value
Embedding level (%)
Sequential
Equal spacing
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50
60
70
80
90
100


60% Embedding Level
Welch's t-test p-value
Embedding level (%)
Sequential
Equal spacing
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50
60
70
80
90
100


80% Embedding Level
Welch's t-test p-value
Embedding level (%)
Sequential
Equal spacing
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
10
20
30
40
50
60
70
80
90
100


100% Embedding Level
Welch's t-test p-value
Embedding level (%)
Sequential
Equal spacing
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
10
20
30
40
50
60
70
80
90
100


Random Embedding
Welch's t-test p-value
Embedding level (%)
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50
60
70
80
90
100
20%
40%
60%
80%
100%

Bibliography

[1] Kevin Maney. Bin Laden's messages could be hiding in plain sight. https://usatoday30.usatoday.com/tech/columnist/2001/12/19/maney.htm, Dec 2001. Accessed: 2022-07-21.

[2] ESET Research. Readers of popular websites targeted by stealthy stegano exploit kit hiding in pixels of malicious ads. https://www.welivesecurity.com/2016/12/06/readers-popular-websites-targeted-stealthy-stegano-exploit-kit-hiding-pixels-malicious-ads/, Dec 2016. Accessed: 2022-07-21.

[3] Check Point researchers. Hacked in translation – from subtitles to complete takeover. https://blog.checkpoint.com/2017/05/23/hacked-in-translation/, May 2017. Accessed: 2022-07-21.

[4] Shaun Koh. Traffic classification: Observing Internet traffic changes at Auckland. Master's thesis, The University of Auckland, 2016.

[5] Jie Li, Andreas Aurelius, Viktor Nordell, Manxing Du, Åke Arvidsson, and Maria Kihl. A five year perspective of traffic pattern evolution in a residential broadband access network. In *Future Network & Mobile Summit*, pages 1–9, Berlin, Germany, July 2012.

[6] Jun O Seo, Sathiamoorthy Manoharan, and Aniket Mahanti. Network steganography and steganalysis – a concise review. In *International Conference on Applied and Theoretical Computing and Communication Technology*, pages 368–371, Bengaluru, India, July 2016.

[7] Jun O Seo, Sathiamoorthy Manoharan, and Aniket Mahanti. A discussion and review of network steganography. In *IEEE Intl Conf on Dependable, Autonomic and Secure Computing*, pages 384–391, Auckland, New Zealand, August 2016.

[8] Jun O Seo, Sathiamoorthy Manoharan, and Ulrich Speidel. Steganalysis of storage-based covert channels using entropy. In *International Telecommunication Networks and Applications Conference*, pages 1–6, Auckland, New Zealand, November 2019.

[9] Sathiamoorthy Manoharan, Giovanni Russello, Jun O Seo, Ulrich Speidel, and Asil Stanikzai. On synthesizing network traces – case studies in network steganalysis and packet analysis. In *IEEE Conference on Application, Information and Network Security*, pages 47–52, Kota Kinabalu, Malaysia, November 2020.

[10] Jun O Seo, Sathiamoorthy Manoharan, and Ulrich Speidel. Automatic detection of storage-based covert channels. In *International Conference on Electrical, Communication, and Computer Engineering*, pages 1–7, Kuala Lumpur, Malaysia, June 2021.

[11] Jun O Seo, Sathiamoorthy Manoharan, and Ulrich Speidel. Feasibility evaluation of long-distance network timing-based covert channels. In *International Conference on Electrical, Communication, and Computer Engineering*, pages 1–5, Kuala Lumpur, Malaysia, June 2021.

[12] Fabien Petitcolas, Ross Anderson, and Markus G. Kuhn. Information hiding – a survey. *Proceedings of the IEEE*, 87(7):1062–1078, Jul 1999.

[13] Elżbieta Zielińska, Wojciech Mazurczyk, and Krzysztof Szczypiorski. Trends in steganography. *Communications of the ACM*, 57(3):86–95, March 2014.

[14] Jon Postel. Transmission control protocol. RFC 793, RFC Editor, September 1981. `http://www.rfc-editor.org/rfc/rfc793.txt`.

[15] Jon Postel. Internet Protocol. RFC 791, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc791.txt.

[16] Michael Ramalho, Paul Jones, Noboru Harada, Muthu Perumal, and Lei Miao. RTP payload format for G.711.0. RFC 7655, RFC Editor, November 2015. http://www.rfc-editor.org/rfc/rfc7655.txt.

[17] Fahimeh Rezaei, Michael Hempel, Dongming Peng, and Hamid Sharif. Disrupting and preventing late-packet covert communication using sequence number tracking. In *IEEE Military Communications Conference*, pages 599–604, San Diego, CA, USA, November 2013.

[18] Norka B. Lucena, James Pease, Payman Yadollahpour, and Steve J. Chapin. Syntax and semantics-preserving application-layer protocol steganography. In Jessica Fridrich, editor, *Information Hiding*, pages 164–179, Berlin, Heidelberg, June 2005.

[19] Artur Janicki, Wojciech Mazurczyk, and Krzysztof Szczypiorski. Steganalysis of transcoding steganography. *Annals of Telecommunications*, 69(7):449–460, 2013.

[20] Wojciech Mazurczyk. VoIP steganography and its detection—a survey. *ACM Computing Surveys*, 46(2), December 2013.

[21] Wojciech Fraczek, Wojciech Mazurczyk, and Krzysztof Szczypiorski. How hidden can be even more hidden? In *International Conference on Multimedia Information Networking and Security*, pages 581–585, Shanghai, China, November 2011.

[22] Artur Janicki, Wojciech Mazurczyk, and Krzysztof Szczypiorski. Influence of speech codecs selection on transcoding steganography. *Telecommunication Systems*, 59(3):305–315, July 2015.

[23] Ki Suh Lee, Han Wang, and Hakim Weatherspoon. PHY covert channels: Can you see the idles? In *USENIX Symposium on Networked Systems Design and Implementation*, pages 173–185, Seattle, WA, USA, April 2014.

[24] Hermine Hovhannisyan, Kejie Lu, and Jianping Wang. A novel high-speed IP-timing covert channel: Design and evaluation. In *IEEE International Conference on Communications*, pages 7198–7203, London, UK, June 2015.

[25] Osamah Ibrahiem Abdullaziz, Vik Tor Goh, Huo-Chong Ling, and KokSheik Wong. Network packet payload parity based steganography. In *IEEE Conference on Sustainable Utilization and Development in Engineering and Technology*, pages 56–59, Selangor, Malaysia, May 2013.

[26] Wojciech Mazurczyk, Paweł Szaga, and Krzysztof Szczypiorski. Using transcoding for hidden communication in IP telephony. *Multimedia Tools and Applications*, 70(3):2139–2165, June 2014.

[27] Rui Miao and Yongfeng Huang. An approach of covert communication based on the adaptive steganography scheme on Voice over IP. In *IEEE International Conference on Communications*, pages 1–5, Kyoto, Japan, June 2011.

[28] Naofumi Aoki. A packet loss concealment technique for VoIP using steganography. In *Proceedings of International Symposium on Intelligent Signal Processing and Communication Systems*, pages 470–473, Awaji Island, Japan, December 2003.

[29] Hong Zhao, Yun Q. Shi, and Nirwan Ansari. Hiding data in multimedia streaming over networks. In *Annual Communication Networks and Services Research Conference*, pages 50–55, Montreal, QC, Canada, May 2010.

[30] Bridger Hahn, Rishab Nithyanand, Phillipa Gill, and Rob Johnson. Games without frontiers: Investigating video games as a covert channel. In *IEEE European Sym-*

posium on Security and Privacy, pages 63–77, Saarbruecken, Germany, March 2016.

[31] Paul Vines and Tadayoshi Kohno. Rook: Using video games as a low-bandwidth censorship resistant communication platform. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, pages 75–84, New York, NY, USA, 2015.

[32] Gaurav Shah and Andres Molina. Keyboards and covert channels. In *USENIX Security Symposium*, Vancouver, B.C. Canada, July 2006.

[33] Steven Gianvecchio and Haining Wang. Detecting covert timing channels: An entropy-based approach. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 307–316, New York, NY, USA, 2007.

[34] Steven J. Murdoch and Stephen Lewis. Embedding covert channels into TCP/IP. In *Proceedings of the International Conference on Information Hiding*, pages 247–261, Berlin, Heidelberg, 2005.

[35] Norka B. Lucena, Grzegorz Lewandowski, and Steve J. Chapin. Covert channels in IPv6. In George Danezis and David Martin, editors, *Privacy Enhancing Technologies*, pages 147–166, Berlin, Heidelberg, 2006.

[36] Naofumi Aoki. VoIP packet loss concealment based on two-side pitch waveform replication technique using steganography. In *IEEE Region 10 Conference*, volume C, pages 52–55 Vol. 3, Chiang Mai, Thailand, November 2004.

[37] Naofumi Aoki. Potential of value-added speech communications by using steganography. In *Conference on Intelligent Information Hiding and Multimedia Signal Processing*, volume 2, pages 251–254, Kaohsiung, Taiwan, November 2007.

[38] National Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD edition, December 1985.

[39] National Computer Security Center. *Covert Channel Analysis of Trusted Systems*, NCSC-TG-030 edition, November 1993.

[40] Common Criteria. Information technology – Security techniques – Evaluation criteria for IT security. ISO ISO/IEC 15408, International Organization for Standardization, 2012.

[41] Harold F. Tipton and Micki Krause. *Information Security Management Handbook, Volume 2*. CRC Press, 2008.

[42] Common Criteria. Certified products. `https://www.commoncriteriaportal.org/products`, 2021. Accessed: 2022-07-21.

[43] Wassenaar Members. Wassenaar Arrangement. `http://www.wassenaar.org`. Accessed: 2022-07-21.

[44] Electronic Privacy Information Center. *Privacy and Human Rights Report 2006: An International Survey of Privacy Laws and Developments*. Electronic Privacy Information Center, 2007.

[45] Tom Pullar-Strecker. Customs downplays password plan. `http://www.stuff.co.nz/technology/digital-living/67449940/customs-downplays-password-plan`. Accessed: 2022-07-21.

[46] Lev Grossman. Inside Apple CEO Tim Cook's fight with the FBI. *TIME*, March 2016.

[47] Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalizationand end-to-end protocol semantics. In *Proceedings*

of the Conference on USENIX Security Symposium - Volume 10, Berkeley, CA, August 2001.

[48] Myong H. Kang, Ira S. Moskowitz, and Daniel C. Lee. A network version of the pump. In *Proceedings IEEE Symposium on Security and Privacy*.

[49] Myong H. Kang, Ira S. Moskowitz, Bruce E. Montrose, and James J. Parsonese. A case study of two NRL pump prototypes. In *Proceedings Annual Computer Security Applications Conference*, pages 32–43, San Diego, CA, USA, December 1996.

[50] Karl Pearson. *On the theory of contingency and its relation to association and normal correlation*. Dulau and Company, 1904.

[51] Wojciech Mazurczyk, Steffen Wendzel, Sebastian Zander, Amir Houmansadr, and Krzysztof Szczypiorski. *Network Steganography Countermeasures*, pages 207–242. 2016.

[52] Rennie Archibald and Dipak Ghosal. A comparative analysis of detection metrics for covert timing channels. *Computers & Security*, 45:284–292, 2014.

[53] S. Zerafshan Goher, Barkha Javed, and Nazar Abbas Saqib. Covert channel detection: A survey based analysis. In *High Capacity Optical Networks and Emerging/Enabling Technologies*, pages 057–065, Istanbul, Turkey, December 2012.

[54] Ugo Fiore, Francesco Palmieri, Aniello Castiglione, and Alfredo De Santis. Network anomaly detection with the restricted Boltzmann Machine. *Neurocomput.*, 122:13–23, December 2013.

[55] Pradhumna Lal Shrestha, Michael Hempel, Fahimeh Rezaei, and Hamid Sharif. Leveraging statistical feature points for generalized detection of covert timing

channels. In *IEEE Military Communications Conference*, pages 7–11, Baltimore, MD, USA, 2014.

[56] Cai Zhiyong, Shen Ying, and Shen Changxiang. Detection of insertional covert channels using chi-square test. In *International Conference on Multimedia Information Networking and Security*, volume 1, pages 432–435, Wuhan, China, November 2009.

[57] Serdar Cabuk, Carla E. Brodley, and Clay Shields. IP covert timing channels: Design and detection. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 178–187, Washington DC, USA, 2004.

[58] Serdar Cabuk, Carla E. Brodley, and Clay Shields. IP covert channel detection. *ACM Trans. Inf. Syst. Secur.*, 12(4):22:1–22:29, April 2009.

[59] Sebastian Zander. *Performance of Selected Noisy Covert Channels and Their Countermeasures in IP Networks*. PhD thesis, Swinburne University of Technology, May 2010.

[60] Shankar Sadasivam, Pierre Moulin, and Sean Meyn. A universal divergence-rate estimator for steganalysis in timing channels. In *IEEE International Workshop on Information Forensics and Security*, pages 1–6, Seattle, WA, USA, December 2010.

[61] Chaim Sanders, Jacob Valletta, Bo Yuan, Daryl Johnson, and Peter Lutz. Employing entropy in the detection and monitoring of network covert channels. In *Proceedings of the International Conference on Security and Management*, Las Vegas, NV, USA, July 2012.

[62] Steven Gianvecchio and Haining Wang. An entropy-based approach to detecting covert timing channels. *IEEE Transactions on Dependable and Secure Computing*, 8(6):785–797, 2011.

[63] Szymon Grabski and Krzysztof Szczypiorski. Network steganalysis: Detection of steganography in IEEE 802.11 wireless networks. In *Ultra Modern Telecommunications and Control Systems and Workshops, International Congress on*, pages 13–19, Sept 2013.

[64] Anyi Liu, Jim Chen, and Li Yang. Real-time detection of covert channels in highly virtualized environments. *Critical Infrastructure Protection V*, pages 151–164, 2011.

[65] Rennie Archibald and Dipak Ghosal. Design and analysis of a model-based covert timing channel for Skype traffic. In *IEEE Conference on Communications and Network Security*, pages 236–244, Florence, Italy, September 2015.

[66] Alvis C. M. Fong, G. R. Higgie, and Bernard Fong. Multimedia applications of self-synchronizing T-codes. In *Proceedings International Conference on Information Technology: Coding and Computing*, pages 519–523, Las Vegas, NV, USA, April 2001.

[67] Kenji Hamano and Hirosuke Yamamoto. Data compression based on a dictionary method using recursive construction of T-codes. In *2010 Data Compression Conference*, pages 531–531, Snowbird, UT, USA, March 2010.

[68] Ulrich Speidel, T. Aaron Gulliver, and Thokozani Shongwe. Multicarrier error correction using T-codes. In *Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 877–880, Victoria, BC, Canada, August 2011.

[69] Sathiamoorthy Manoharan, Aquib Master, and Ulrich Speidel. Complexity-based steganalysis. In *International Symposium on Information Theory and its Applications*, pages 40–44, Victoria, BC, Canada, October 2014.

[70] Mark R. Titchener. Character-error bound for the T-code synchronisation process. *IEE Proceedings E - Computers and Digital Techniques*, 134(3):155–158, May 1987.

[71] Ulrich Günther. *Robust source coding with generalised T-codes*. PhD thesis, University of Auckland, 1998.

[72] Radu Nicolescu. *Uniqueness theorems for T-codes*. Tamaki report series ; no. 9. University of Auckland, Tamaki Campus, Auckland, N.Z., September 1995.

[73] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

[74] Mark R. Titchener. Deterministic computation of complexity, information and entropy. In *Proceedings. IEEE International Symposium on Information Theory*, pages 326–, Cambridge, MA, USA, August 1998.

[75] Mark R. Titchener. A measure of information. In *Proceedings Data Compression Conference*, pages 353–362, Snowbird, UT, USA, March 2000.

[76] Kenji Hamano and Hirosuke Yamamoto. Data compression based on a dictionary method using recursive construction of T-Codes. In *Data Compression Conference*, pages 531–531, Snowbird, UT, USA, March 2010.

[77] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. US Government Printing Office, 1948.

[78] Ulrich Speidel, Raimund Eimann, and Nevil Brownlee. Detecting network events via T-entropy. In *International Conference on Information, Communications Signal Processing*, pages 1–5, Singapore, December 2007.

[79] T. Aaron Gulliver, Isaiah Makwakwa, and Ulrich Speidel. On the generation of aperiodic and periodic necklaces via T-augmentation. *Fundamenta Informaticae*, 83:91–107, 01 2008.

[80] Solomon W. Golomb and Basil Gordon. Codes with bounded synchronization delay. *Information and Control*, 8(4):355–372, 1965.

[81] Stig Venaas and Alvaro Retana. PIM message type space extension and reserved bits. RFC 8736, RFC Editor, February 2020. `http://www.rfc-editor.org/rfc/rfc8736.txt`.

[82] Christopher A. Kent and Jeffrey C. Mogul. Fragmentation considered harmful. *SIGCOMM Computer Communication Review*, 25(1):75–87, January 1995.

[83] Jeffrey Mogul and Steve Deering. Path MTU discovery. RFC 1191, RFC Editor, November 1990. `http://www.rfc-editor.org/rfc/rfc1191.txt`.

[84] Matthew Luckie and Ben Stasiewicz. Measuring Path MTU Discovery behaviour. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*, pages 102–108, Melbourne, Australia, November 2010.

[85] Matthias Göhring, Haya Shulman, and Michael Waidner. Path MTU Discovery considered harmful. In *IEEE International Conference on Distributed Computing Systems*, pages 866–874, Vienna, Austria, July 2018.

[86] Stephen Deering and Robert Hinden. Simple Internet Protocol (SIP) specification. RFC 8507, RFC Editor, December 2018. `http://www.rfc-editor.org/rfc/rfc8507.txt`.

[87] Gijs Van Den Broek, Roland Van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC meets real world: dealing with unreachability caused by fragmentation. *IEEE Communications Magazine*, 52(4):154–160, 2014.

[88] Catherine Wu and Eric Vyncke. Resolve IPv4 fragmentation, MTU, MSS, and PMTUD issues with GRE and IPsec. Technical report, Cisco, Jan 2019.

[89] Pekka Savola. MTU and fragmentation issues with in-the-network tunneling. RFC 4459, RFC Editor, April 2006. http://www.rfc-editor.org/rfc/rfc4459.txt.

[90] Wojciech Mazurczyk and Krzysztof Szczypiorski. Evaluation of steganographic methods for oversized IP packets. *Telecommunication Systems*, 49:207–217, 02 2012.

[91] Deepa Kundur and Kamran Ahsan. Practical Internet steganography: data hiding in IP. *Proc. Texas wksp. security of information systems*, 2003.

[92] Kamran Ahsan and Deepa Kundur. Practical data hiding in TCP/IP. In *Multimedia and Security Workshop – ACM Multimedia*, Juan-les-Pins, France, December 2002.

[93] Wireshark. Wireshark. https://www.wireshark.org. Accessed: 2022-07-21.

[94] Omnipacket. WireEdit. https://www.omnipacket.com/wireedit. Accessed: 2022-07-21.

[95] Open Information Security Foundation. Suricata. https://suricata-ids.org. Accessed: 2022-07-21.

[96] TCPDUIMP. TCPDUMP. http://www.tcpdump.org. Accessed: 2022-07-21.

[97] Gianluca Costa and Andrea de Franceschi. Xplico. https://www.xplico.org. Accessed: 2022-07-21.

[98] Colasoft. Packet Player. https://www.colasoft.com/packet_player. Accessed: 2022-07-21.

[99] Xiuquan Li and Zhidong Deng. A machine learning approach to predict turning points for chaotic financial time series. In *IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 331–335, Patras, Greece, 2007.

[100] Bushra Praveen, Swapan Talukdar, Shahfahad, Susanta Mahato, Jayanta Mondal, Pritee Sharma, Abu Reza Md. Towfiqul Islam, and Atiqur Rahman. Analyzing trend and forecasting of rainfall changes in India using non-parametrical and machine learning approaches. *Scientific reports*, 10(1):1–21, 2020.

[101] Ansoumana Bodian, Lamine Diop, Geremy Panthou, Honoré Dacosta, Abdoulaye Deme, Alain Dezetter, Pape Malick Ndiaye, Ibrahima Diouf, and Théo Vischel. Recent trend in Hydroclimatic conditions in the Senegal river basin. *Water*, 12(2):436, 2020.

[102] Peter J. Hilton. *Differential Calculus*. Routledge, 1958.

[103] Kendrick N. Kay, Ariel Rokem, Jonathan Winawer, Robert F. Dougherty, and Brian A. Wandell. GLMdenoise: a fast, automated technique for denoising task-based fMRI data. *Frontiers in neuroscience*, 7:247, 2013.

[104] Abdelrahman Abdelhamed, Stephen Lin, and Michael S. Brown. A high-quality denoising dataset for smartphone cameras. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1692–1700, Salt Lake City, UT, USA, 2018.

[105] Artur M. Brodzki and Jędrzej Bieniasz. Yet another network steganography technique based on TCP retransmissions. In *International Conference on Frontiers of Signal Processing*, pages 35–39, Marseille, France, 2019.

[106] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1998.

[107] Donald E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, April 1976.

[108] Niko Rebenich. Fast low memory T-Transform string complexity in linear time and space with applications to android app store security. 2012.

[109] Jia Yang and Ulrich Speidel. A T-decomposition algorithm with O(n log n) time and space complexity. In *Proceedings. International Symposium on Information Theory*, pages 23–27, Adelaide, SA, Australia, September 2005.

[110] Steven Gianvecchio, Haining Wang, Duminda Wijesekera, and Sushil Jajodia. Model-based covert timing channels: Automated modeling and evasion. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection*, pages 211–230, Berlin, Heidelberg, 2008.

[111] Andrew D. Ker, Patrick Bas, Rainer Böhme, Rémi Cogranne, Scott Craver, Tomáš Filler, Jessica Fridrich, and Tomáš Pevný. Moving steganography and steganalysis from the laboratory into the real world. In *Information Hiding and Multimedia Security*, pages 45–58, New York, NY, USA, 2013.

[112] Amir Houmansadr and Nikita Borisov. Coco: Coding-based covert timing channels for network flows. In *Information Hiding*, pages 314–328, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[113] Ramaswamy Ramaswamy, Ning Weng, and Tilman Wolf. Characterizing network processing delay. In *IEEE Global Telecommunications Conference*, volume 3, pages 1629–1634 Vol.3, Dallas, TX, USA, November 2004.

[114] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proceedings of the Conference on Applications, Technologies, Architecture-sand Protocols for Computer Communications*, pages 281–292, Portland, Oregon, USA, 2004.

[115] Jim Gettys. Bufferbloat: Dark buffers in the Internet. *IEEE Internet Computing*, 15(3):96–96, 2011.

[116] Artem Proskochylo, Mikhail Zriakhov, and Artem Akulynichev. The effects of queueing algorithms on QoS for real-time traffic in process of load balancing. In *International Scientific-Practical Conference Problems of Infocommunications. Science and Technology*, pages 575–580, Kharkiv, Ukraine, October 2018.

[117] Carlo Demichelis and Philip Chimento. IP packet delay variation metric for IP performance metrics (IPPM). RFC 3393, RFC Editor, November 2002. `http://www.rfc-editor.org/rfc/rfc3393.txt`.