Information & Security

M. Bajor & M. Niemiec
vol. 47, no. 2 (2020): 261-275
https://doi.org/10.11610/isij.4718

Published since 1998

ISSN 0861-5160 (print), ISSN 1314-2119 (online)

Research Article

A New Steganographic Algorithm for Hiding Messages in Music

Michał Bajor (✉), *Marcin Niemiec*

AGH University of Science and Technology, Krakow, Poland
https://www.agh.edu.pl/

ABSTRACT:

Steganography in audio files usually revolves around well-known concepts and algorithms, least significant bit algorithm to name one. This paper proposes a new, alternative approach where steganographic information is connected with the medium even more – by using the medium itself as the information. The goal of this paper is to present a new aspect of steganography, which utilizes machine learning. This form of steganography may produce statistically indeterminable steganographic files which are immune to brute force attempts at trying to retrieve the hidden messages. Then the proposed solution is verified against statistical analysis and brute force attacks with promising results.

ARTICLE INFO:

RECEIVED: 04 JUNE 2020
REVISED: 24 AUG 2020
ONLINE: 22 SEP 2020

KEYWORDS:

steganography, audio, MIDI format, machine learning

Introduction

Steganography is often discussed along with cryptography. It is not surprising – both are related to confidentiality. There is no need to explain why keeping certain information confidential is important and most often this is achieved by encrypting such information. However sometimes it is crucial to hide the very fact of communication. It seems that the general tendency of steganography leans towards hiding messages in network traffic or inside image files. However, one, sometimes neglected, branch of steganography revolves around audio

✉ Corresponding Author: Tel.: +48 126173937; E-mail: bajor.michal98@gmail.com

files. Among those kinds of files, one of the best mediums for hiding messages are music files.[1]

This paper focuses on MIDI (Musical Instrument Digital Interface) files used as the medium to hide confidential messages. This format was used mostly due to less complicated nature of creating and detecting sounds. However, the general idea presented and verified in the following sections should be applicable to any audio file format.

In the next section current state of work regarding steganography in music (MIDI files) is examined. In section three steganography in general is discussed. Then a new steganography technique is proposed and explained. Section five is dedicated to the verification of proposed method. The last section concludes this paper.

Related Work

Hiding messages in audio files is a broad subject, however there are not many research papers which focus on MIDI files. Most common algorithms for hiding messages in MIDI files were described in[2]. However, there seem to be multiple branches of steganography in MIDI files.

One branch revolves around indistinguishable changes in velocity of sounds.[2; 3; 4] The first one describes simple LSB (Least Significant Bit) algorithm, however the latter two take a more sophisticated approach. Velody [3] uses the first note's velocity as the reference value and then encodes secret message in following notes' velocities by either adding or subtracting one from reference velocity value. Wu et al.[4] took even more complicated approach using velocities in sequences of either non-increasing or non-decreasing pitches. The idea similar to the one presented in this paper was briefly mentioned in [3] and described as utilization of natural properties of music and binary substitution (choosing two notes, each representing one binary value), however it was not investigated further in that paper.

Another manner of steganography in MIDI files is hiding messages using note duration.[5] General idea here is very similar to velocity-based MIDI steganography. Instead of altering velocity value of a note, duration is slightly changed. The method proposed in [6] uses every two adjacent delta time values, but hides secret information in the first delta time (in consequence increasing it). Second delta time from the pair is modified to mitigate the changes done to the first one so that total value of delta times for a given pair remains the same.

Quite interesting, and alternative, approach was presented in StegIbiza.[7] It used a music tempo as the message carrier and proved that tempo change of 1% or lower were undetectable even by people with musical background and working as professional musicians.

Steganography

As mentioned in the Introduction section, steganography is the art of hiding messages in plain sight. To hide the message a medium is needed – it might be a real painting, piece of paper, digital image file, audio file, etc. Only the medium should be visible to a person for whom the message was not intended. That is

the greatest problem in steganography – convincing a third party that there is no additional information in a given medium. At the same time even for a curious user, who would like to investigate the file, it should be impossible (or at least very challenging) to notice anything out of the ordinary. Quite often steganographic algorithms exploit flaws in human senses to achieve this effect.

Historically speaking, steganography is as old as cryptography, if not older. Both were used in VII B.C. in Sparta, in form of scytale. Considering the characteristics of steganography, it is not surprising that it was often used in both military (microdots [1]) and criminal (hiding malware in files) appliances. Steganography is also well known by children who sometimes play with lemon juice and use it as a sympathetic ink.

Nowadays the most frequent implementation of steganography is using metadata or noise in a file. For instance, least significant bit (LSB) algorithm is widely used in image and audio files. This algorithm directly exploits flaws in human senses. Changing least significant bit of red channel (or any other channel for that matter) in RGB image file is indistinguishable for the human eye. Similarly changing the least significant bit of sound velocity in an audio file is unnoticeable to human ear. In some cases changing even three least significant bits may not be detected by human senses[2]. A very thorough description of steganography and most common use cases with examples of techniques was presented by Judge.[8]

On the other hand, some form of steganography is often used as a mean to protect data from malicious activities, such as counterfeit or illegal copying. This is better known as watermarking and may be both visible and invisible. The first one is more common; an example might be a company logo in the background of a pdf file. The idea here is that this watermark cannot be removed without visually disturbing the file it marks. However, it is the latter way that is more connected to steganography. For example, changing a specific number of pixel data in an image file (for example using LSB algorithm) might be enough to then determine if the file has not lost its integrity. Of course, invisible watermarking needs to be done in a robust way, so that even a slightest change in a file will affect the watermark.

Steganography in general has one big drawback – it is highly inefficient. Quite often a lot of background information is needed to hide even short messages. For instance, LSB algorithm in the best-case scenario needs eight times more medium data than useful information.[2] Furthermore, it often requires some offline information to be distributed beforehand. At least the recipient should know that he or she should expect a steganographic message.

---

[1] A method of scaling down an image into the size of a dot used in a text (for example above letter ‘i’ or at the end of the sentence. For instance, those scaled down images might have been pictures taken from a reconnaissance plane.

[2] Assuming that least significant bit of every byte of the medium is used to hide secret message.

To avoid any suspicion that something is hidden in a given music file it is crucial not to leave any data that might lead to that conclusion. In the era of digitization, it is equally important, if not more, that the computer will not be able to notice the existence of a hidden message as much as a human being. In order to achieve that a message cannot be hidden in any form of metadata associated with a specific file. One might argue that LSB algorithm does just that – value changes are imperceptible for human senses and since the distribution of '1's is usually about the same as the distribution of '0's in any meaningful data the computers also have hard time spotting it. However, the popularity of LSB is its greatest flaw – this is usually the first thing that an analyst will look for. This is why LSB algorithm is often used alongside another measure. This measure would be responsible for getting the value which then is modified using LSB. This approach proves to be working and gives pretty decent results; however, it also makes the algorithm more complicated.

A New Algorithm

This paper introduces a new steganography technique, which uses relatively simple principles as basis. The solution assumes that a confidential message should be hidden as the music itself, not as something that was added later. Instead of changing metadata or noise in a file, using actual sounds as the information carrier minimizes the risk associated with leaking information whether something is hidden in a specific file or not. If a message is embedded into already existing file, then it is possible that a comparative analysis of those files will reveal steganographic information. Creating a file purely for steganographic purposes mitigates that risk.

Unfortunately, this idea introduces new problems, main one being that the process of creating music might take a long time even without any restrictions related to attempting to hide anything. Additionally, the whole music piece needs to sound 'good', i.e.: a listener should simply enjoy listening to it. It would be possible to hire a professional musician who would compose a new song; however, it would also prove to be highly costly and time ineffective.

The solution proposed in this paper uses a different approach. Machine learning model is used to create music.[9] That model was trained on piano music mostly consisting of music from Final Fantasy games. It is a recurrent neural network, i.e.: neural network which uses its computation results to calculate next output. A thorough description of the model was given by its author in [10]. The general idea is that the model first randomly picks a sequence of 100 notes, which will be used as first input (they are needed because model needs a fixed input length). Then each calculated output will be put in front of that sequence, which then will be clipped to 100 notes again. Only the calculated notes are used to create MIDI files.

Algorithm

The high-level idea of the whole algorithm is presented in Figure 1, however in this subsection the whole process is explained with greater detail. It is worth mentioning that this paper focuses purely on hiding plain-text ASCII messages.

It is possible to encrypt the message before hiding it, however it is not in the scope of this paper.

1. Musical notes are split into two sets – one is representing a binary '1' and the tother – a '0'. It does not matter which notes are in which set, however to ensure better results sizes of those sets should be the same or very close.
2. User needs to provide a secret message (ASCII string) and a positive number (called depth_level in Figure 1) corresponding to how often a steganographic sound will occur in the output musical file. Please note, it does not mean that every depth levelth sound will be a steganographic note – it means that from a certain point every depth levelth note will be a steganographic one.
3. Secret message provided by user is converted into a stream of '1's and '0's.
4. Results are stored in a list which will be converted to a MIDI file. The music generation process starts and works according to following rules.
    - Chords are added to the final list without any changes. This algorithm does not use them as the information carrier.
    - First 2*depth_level notes are also added without any changes. As the model used for this implementation is a recurrent neural network, this rule allows the model to warm up.
    - If total number of notes (notes_number) modulo depth level is equal to 0 then this is a note that will reflect next '0' or '1' from a converted secret message. In other words – every depth level note is a steganographic note.
    - If a note is not a steganographic note then it is added to the final list without any changes.
    - If a note is a steganographic note:
        - it is added to the list if it already is present in a correct notes set (i.e.: if bit to be hidden is a '1' and note is in a set corresponding to '1' then it is added).
        - if it is not in a correct notes set then the next best note is taken (from the model's prediction) and these rules are applied once again.
5. Once all '1's and '0's representing secret message were projected into their corresponding notes the generation process ends and the list is converted into a MIDI file.

Implementation

Algorithm described in this section was implemented using Python programming language (version 3.7). For model prediction it uses tensorflow and keras modules. Most of number operations use numpy module. Both creation and analysis of MIDI files were performed using music21 module. Figure 2 presents graphical user interface, which was written using kivy module. For the graphical interface it is possible to specify output filename, however it has nothing to do

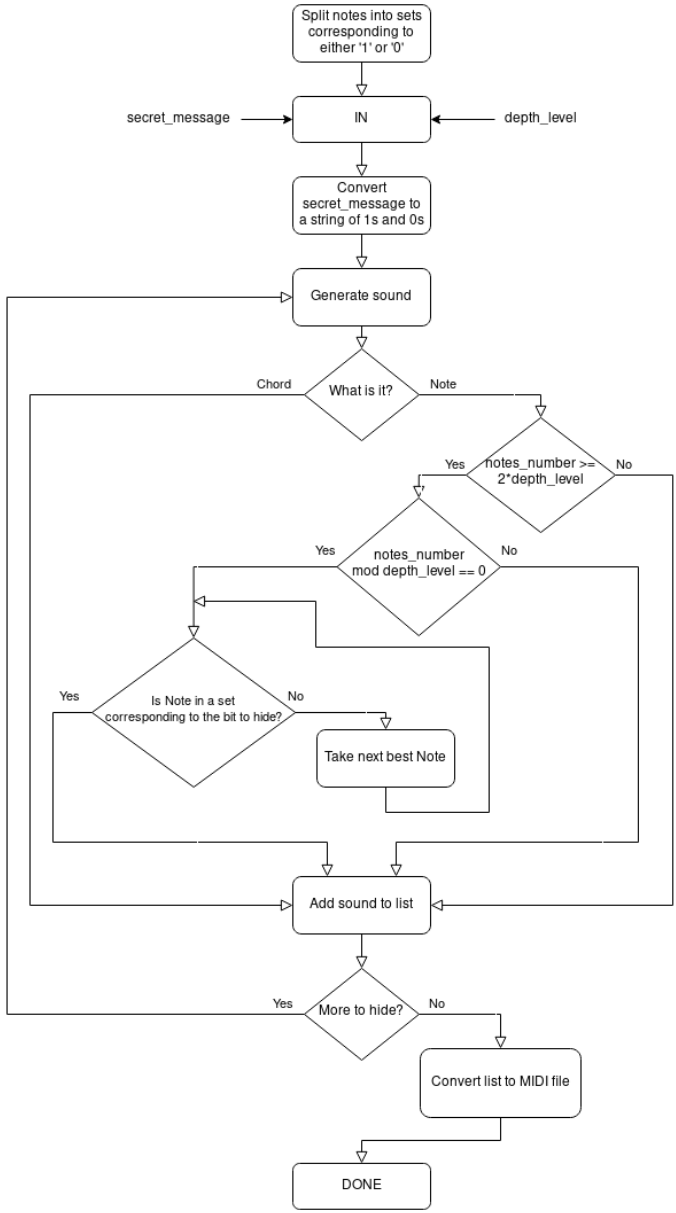

Figure 1: Diagram representing high-level overview of proposed algorithm for hiding messages.

with algorithm itself. As seen in Figure 2, in order to extract the secret message only the hiding depth is needed, as a proof of correctly performed extraction the secret message fetched from a file is displayed.

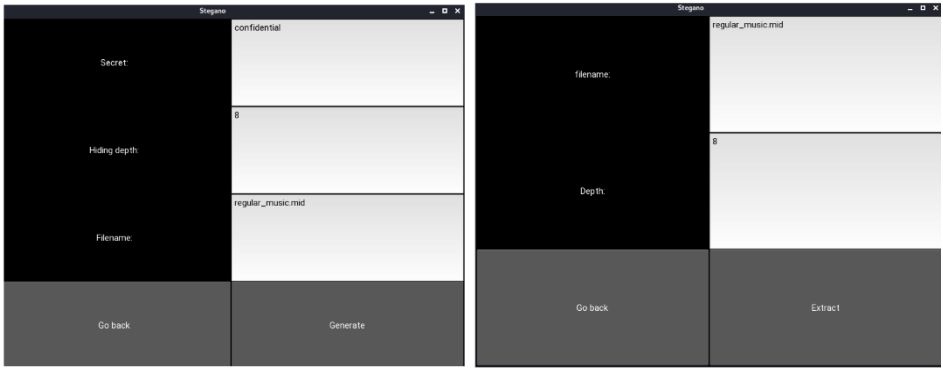

Figure 2: Creating steganographic file (left) and extracting secret message from a file (right).

Verification

In order to verify the functionality and performance of the proposed solution a series of MIDI files were created using the machine learning model. Then using the algorithm described in previous section a series of steganographic MIDI files were created (with varying secret messages and hiding depths). First the files were subjected to purely statistical analysis, then they were analysed in terms of music theory. Lastly brute-force attacks were considered.

Statistical Analysis

Statistical analysis was supposed to check whether it would be possible to determine if a given file might contain something more than just music. Initial tests, separated from theory of music, were designed to see if there are any visible differences between regular MIDI file and one with secret message hidden among the notes.

Figures 3 and 4 show distributions of notes can be found for regular MIDI file and steganographic MIDI file respectively. Notes found in the x-axis of those figures are not binding – those are only examples. Due to random aspect of model predictions each generated file (regardless of secret message presence) is different. However, the distributions always look very similar to those seen in Figures 3 and 4.

Following the information found in Figures 4 and 5 it is visible that steganographic notes do not contribute significantly to overall number of occurrences for a specific note. Due to splitting notes into two sets of a similar size, statistical characteristics of a steganographic MIDI files and normal ones are similar (Figures 3 and 4). Hence a protection against statistical analysis was amplified. This is very similar to the homophonic cipher principle of operation – many possibilities for encoding single character (either ’1’ or ’0’ in this case) flattens the statistics. In result the fewer the number of times a specific note is used for steganographic purposes the less likely it is to be recognized as such.

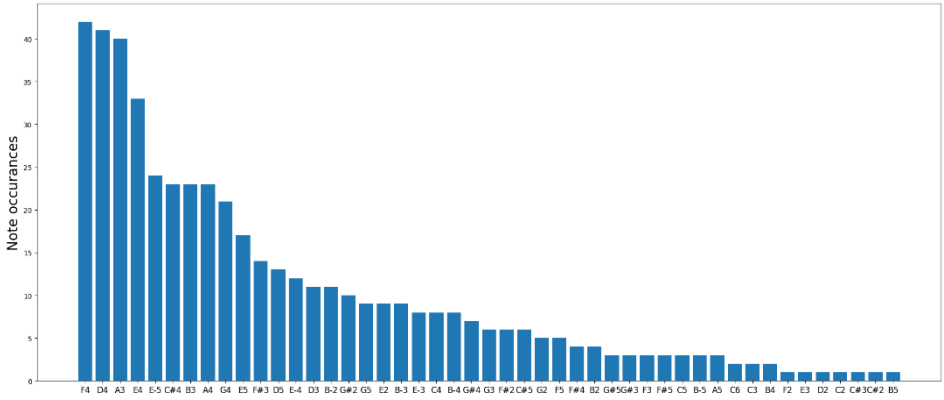

Figure 3: Example of notes distribution in a generated MIDI file without hidden message.

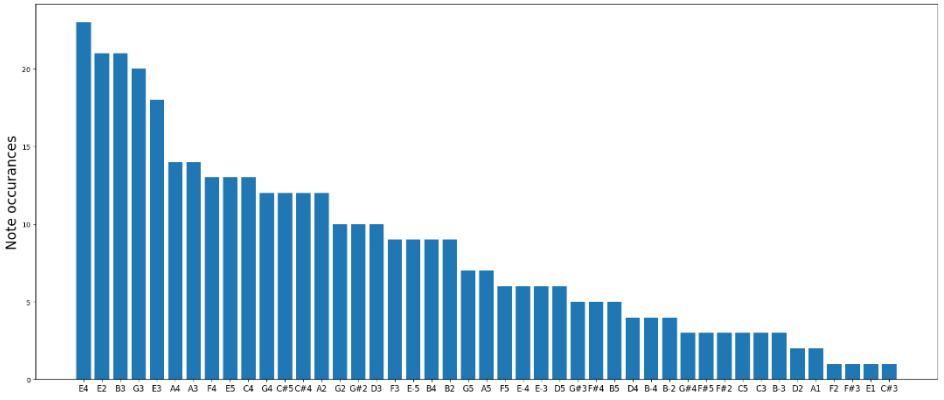

Figure 4: Example of notes distribution in a steganographic MIDI file (hiding depth value set to 8).

For example, note E2 was used 21 times in total, however only once to hide a message (Figures 4 and 5).

Useful steganographic information to carrier information ratio (referred to as 'SCR' – Steganography-Carrier Ratio) can be defined as the total number of notes that were used to hide secret message divided by the total number of sounds in the given MIDI file. Implementation of the algorithm proposed in this paper takes only one, excluding the secret message, parameter – hiding depth. Essentially it has an impact on how often a steganographic note occurs in the file. The relationship is rather simple – increasing the value of hiding depth results in longer output file and in consequence lower SCR. Refer to Table 1 for average values of SCR in regards of *hiding_depth* value. Hence SCR value can be controlled to some extent by user.

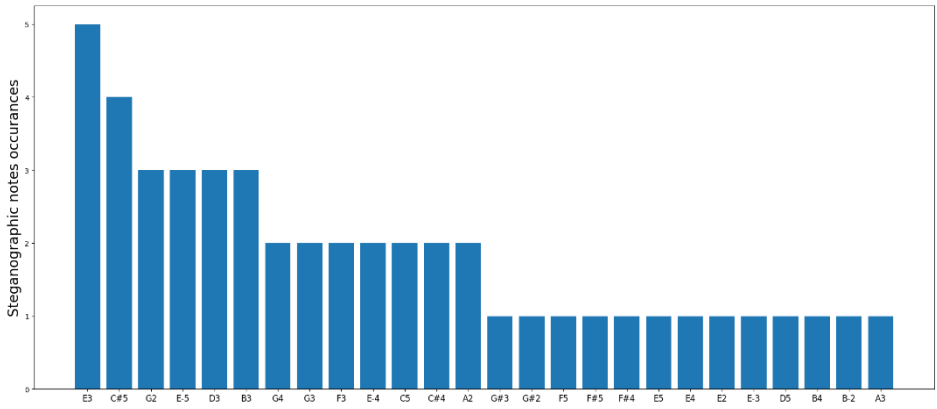

Figure 5: Notes that were used to hide secret message, same file as in Figure 4.

Table 1. Average value of SCR in regards of different hiding depth values.

Hiding depth	Average SCR value	Standard deviation σ
7	0.139	0.003
8	0.122	0.002
12	0.0813	0.0009
16	0.0610	0.0009
20	0.0488	0.0005
24	0.0408	0.0004

Statistical Analysis with Music Theory

To further analyse this solution the concept of musical scales needs to be introduced. The simplified description for a scale is a group of notes that generally sound good together. Though it is not the rule that a single song has to utilize only one scale it is safe to assume that in general the more a scale (any scale) fits the song the more likely it is to sound pleasing.[3] Once again a series of tests were performed to see how this steganographic algorithm operates regarding musical scales.

The model used for generating sound sequences was not trained with musical theory and specifically scales in mind, but on examples instead. Thus it is not the case that every song without a hidden message can be assigned fully to a

[3] However, if a song would consist of only three notes from the same scale just playing constantly in the same order then it would look like the song should be enjoyable to listen to, but in fact it would be the opposite case.

scale. This can be seen in Figure 6. For reference – the closer the value is to one the more fitting the scale is. Most of the regular files generated by the model lie in the fitness of around 0.2 for the best fitting scale. However as seen in Figure 6 there are cases when fitness is much higher (even equal to maximal value).

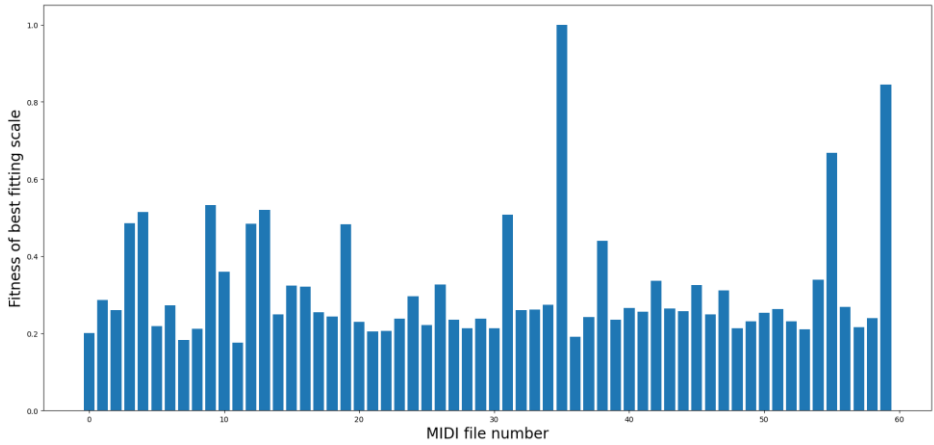

Figure 6: Plot describing fitness of best fitting scale for a given MIDI file.

After analysing 80 MIDI files it turned out that, if the best fitting scale has the fitness of about 0.5 or higher then odds are it will be enjoyable. Although cases with best fitness below 0.5, which also sounded pleasurable have been encountered.

On the other hand analysing steganographic files regarding musical scales revealed a flaw in this implementation. Fitness for most cases is still oscillating around 0.2, however the highest fitness is usually around 0.5 (Figures 7 and 8). It is worth noting that this is only the disadvantage of the implementation – the model used for generating sounds was not trained with steganography as its purpose. Nevertheless, even in this implementation it should be possible to achieve higher fitness results for steganographic MIDI files. In theory if the model always predicted next note as steganographic note every hiding depth note (i.e.: no changes would be needed) then fitness should be higher, although during data collection for this paper it did not occur.

Brute Force Attacks

Algorithm, apart from the secret message, takes only one parameter – in this paper it is referred to as 'hiding depth'. Essentially, this parameter controls how often steganographic note occurs in an output MIDI file, which directly maps to the output file's length. Of course, the length of the secret message is also a contributing factor in the final length of the output file, however for the same secret message it is the hiding depth that controls it.

Regarding immunity to the brute-force attacks it would be helpful to consider worst case scenario. An adversary knows the algorithm and can spend as much

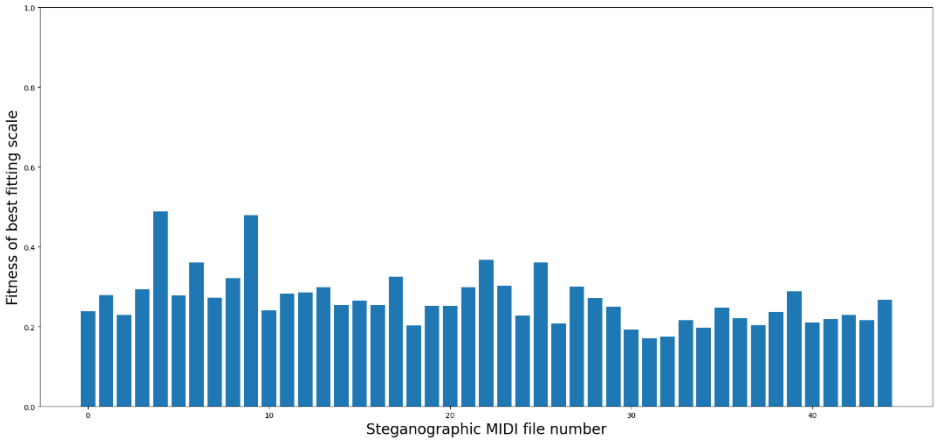

Figure 7: Plot describing fitness of best fitting scale for a given steganographic MIDI file (hiding depth set to 6).

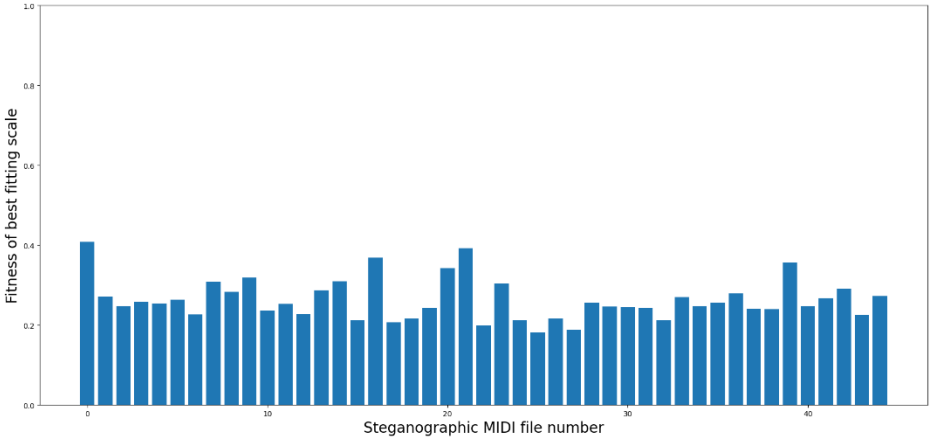

Figure 8: Plot describing fitness of best fitting scale for a given steganographic MIDI file (hiding depth set to 16).

time trying to brute-force the file as he or she wants. Additionally, the adversary knows how notes were split into '1'- and '0'-sets. However, he or she does not know what is the hiding depth value thus all possible values need to be checked. In summary, initial conditions are such that an adversary knows everything about the algorithm. All that he or she will need to correctly retrieve the secret message is hiding depth value. Brute-force attack will require to only check all possible values of hiding depth – so $h \in \langle 1, \frac{N_n}{2} \rangle$, where h is the hiding depth and N_n is the total number of notes in the MIDI file. The average number of

notes for a given hiding depth value and secret message can be found in Tables 2 and 3.

Tables 2 and 3 also prove that total number of notes in a steganographic MIDI file is linearly dependent on both hiding depth and secret message length (length of word 'confidential' is twice as big as length of word 'secret').

Checking all possible values of hiding depth for a MIDI file of total number of notes of 1487 takes nearly five minutes.[4] However as seen in Tables 2 and 3 actual value of hiding depth is much lower than the total number of notes. Therefore, using brute-force with full algorithm knowledge is a trivial task.

Table 2. Average number of notes in a steganographic MIDI file for a given hiding depth and secret message of 'confidential'.

Hiding depth	Average number of notes	Standard deviation σ
7	647	10
8	745	14
12	1123	15
16	1487	32
20	1862	23
24	2262	26

Table 3. Average number of notes in a steganographic MIDI file for a given hiding depth and secret message of 'secret'.

Hiding depth	Average number of notes	Standard deviation σ
7	326	5
8	375	7
12	559	8
16	752	18
20	938	17
24	1131	30

As it was already mentioned in section 5.1 by default algorithm has only one parameter, however it is possible to introduce one additional configurable attribute. As explained in section 4.1, algorithm requires two sets of notes to map a sound to the corresponding binary value. All of data collected and presented in this paper used one, arbitrarily chosen, division of notes into sets. Those sets were of size differentiated by one (due to uneven set of all notes) and that

[4] Tested in Python 3.7 on Intel Core i7-8550U (1.9-4.0 GHz) processor, 16 GB RAM.

specific arrangement of notes was chosen to be as much statisticallyneutral as possible. With that in mind – it does not mean that there is not any other arrangement that would also fit given restrictions.

Assuming that the goal is to have as much protection against brute-force attacks let us consider the situation where user could also specify the note sets used by algorithm. Initial conditions of that brute-force attack would be similar to one presented previously. An adversary knows how algorithm works however this time he or she does not know both, how the notes were split into '1'- and '0'-sets and what was the hiding depth value. Since for a given '1'- and '0'-sets finding the correct hiding depth is a trivial task, analysis of this scenario comes down to solving a simple combination without repetition problem. There are 77 individual notes (used for hiding messages in this implementation). Creating two sets is simply choosing the number of notes to be in one set and calculating how many ways there are to choose that many notes from 77 notes, the rest will automatically end up in the second set. Keeping the sizes of those sets the same as they are in current implementation means that total number of ways to split the notes is the sum of total number of ways to choose '0'-set of length 38 and of length 39 - which is essentially multiplying one of those numbers by two, i.e.:

$$2 * \binom{77}{38} = 2.7217014868 * 10^{22}$$

This is about as much as there are stars in our Universe.[11] The number would be even bigger if different sizes of sets were considered. Of course, on average the correct sets would be found after checking about half of all possibilities, which potentially reduces the number of checks that would have to be done. This, however, does not change the order of magnitude of that number, so for obvious reasons it would be impossible to measure how long it would take to find the correct note sets. However, it only proves that in theory the algorithm is resistant to brute-force attacks even if an adversary knows how it operates, provided that set configuration is used. In a real scenario not all of the notes in the given set are used, which might limit the number of possibilities. This behaviour is not deterministic due to random aspect of music generation in the model used for that purpose. It would be possible to manually influence the performance of the algorithm to make it use more notes from the sets. This would result in an even flatter statistical analysis, which is desirable. On the other hand, it might also worsen the music scale fitness, which is not beneficial.

Introducing set configuration also leads to one new problem. Note sets are crucial components of proposed algorithm, hence the creator and recipient of the secret message should have the same sets configured during embedding and extracting the message. There are many possible solutions for that problem. Steganography is not used for interactive conversation and rather for unidirectional type of communication. Also considering the characteristics and use cases of steganography it is not crucial to have a real-time note sets distribution.

For example, participating parties could agree beforehand on the sets used in the next message even verbally.

Summary

In this paper a new technique of hiding messages in music files was proposed. It utilizes machine learning model to mitigate drawbacks of human-composed music. Algorithm described in this paper is based on a set of simple principles which combined produce complicated structures. Series of MIDI files (both with and without embedded hidden message) were analysed in order to verify the performance of that solution. It proved to be immune to statistical analysis concluding from similar notes distribution for all generated files.

There are some discrepancies considering music theory. Although in most cases the best fitness oscillates around 0.2 for all files (regardless of secret message presence), the best encountered fitness values are much different. For cleanly generated music files (no secret message) the best possible fitness of 1.0 was encountered. Best calculated fitness for steganographic files was below 0.5. Such phenomenon corresponds directly to level of enjoyment associated with a given piece of music. Best fitness value below 0.5 most often means that given music file will not be enjoyable, however it is only a rule of thumb.

Analysis from brute-force perspective gave promising results. Proposed algorithm introduces enough degrees of freedom so that it is impossible to perform brute-force attack in a reasonable period of time. Unfortunately, the more resistant to brute-force attempts a file is, the more likely it is to cause suspicion due to decrease in scale fitness. Thus, it appears that level of brute-force immunity is the matter of compromise with the level of enjoyment associated with listening to a file. That is the case for the analysed implementation.

The biggest defect of this particular implementation is that due to changing the natural behaviour of a model and its randomness, it might take more time until the file that will satisfy user's subjective demands is generated.

Potential future work may include developing custom model for music generation, which will be designed with that specific appliance in mind. It may help with musical scale fitness results and in consequence it might be easier (and faster) to create good-sounding steganographic file.

Acknowledgements

This work was supported by the ECHO project which has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement no. 830943**.**

References

[1] Fatiha Djebbar, Beghdad Ayad, Karim Abed Meraim, and Habib Hamam, "Comparative Study of Digital Audio Steganography Techniques," *EURASIP Journal on Audio, Speech, and Music Processing* 1 (2012), article 25, https://doi.org/10.1186/1687-4722-2012-25.

[2] A. Adli and Z. Nakao, "Three Steganography Algorithms for Midi Files," *2005 International Conference on Machine Learning and Cybernetics*, Guangzhou, China, vol. 4, 2005, pp. 2401–2404, https://doi.org/10.1109/ICMLC.2005.1527346.

[3] Camilla Vaske, Mattias Weckstén, and Eric Jarpe, "Velody — A Novel Method for Music Steganography," *2017 3rd International Conference on Frontiers of Signal Processing (ICFSP)*, Paris, France, 2017, pp. 15–19.

[4] Da-Chun Wu, Chin-Yu Hsiang, and Ming-Yao Chen, "Steganography via Midi Files by Adjusting Velocities of Musical Note Sequences with Monotonically Non-increasing or Non-decreasing Pitches," *IEEE Access* 7 (2019): 154056–154075.

[5] Kotaro Yamamoto and Munetoshi Iwakiri, "A Standard Midi File Steganography Based on Fluctuation of Duration," *2009 International Conference on Availability, Reliability and Security*, Fukuoka, Japan, 2009, pp. 774–779.

[6] Da-Chun Wu and Ming-Yao Chen, "Reversible data hiding in standard midi files by adjusting delta time values," *Multimedia Tools and Applications* 74, no. 21 (Nov. 2015): 9827–9844.

[7] Krzysztof Szczypiorski, "Stegibiza: New Method for Information Hiding in Club Music,"*2016 2nd International Conference on Frontiers of Signal Processing (ICFSP),* 2016, pp. 20–24.

[8] James Judge, "Steganography: Past, present, future," Technical Report, 2001.

[9] Sigurður Skúli, "Classical piano composer," 2019, accessed: 15.04.2020, https://github.com/skuldur/classical-piano-composer.

[10] Sigurður Skúli, "How to Generate Music Using a lstm Neural Network in Keras," 2017, accessed April 15, 2020, https://towardsdatascience.com/how-to-generate-music-using-a-lstmneural-network-in-keras-68786834d4c5.

[11] ESA, "How Many Stars Are There in the Universe?" accessed: May 25, 2020, https://www.esa.int/Science_Exploration/Space_Science/Herschel/How_many_stars_are_there_in_the_Universe.

About the Authors

Michał **Bajor** is an ICT student at the AGH University of Science and Technology, Poland. Apart from pursuing his university degrees (with focus on cybersecurity) he works as a penetration tester.

Marcin **Niemiec** was awarded his PhD and PhD Hab. in telecommunications in 2011 and 2019 respectively. Currently he works as assistant professor at the AGH University of Science and Technology, Poland. His research interests focus on cybersecurity. He has been involved in numerous European projects (FP6, FP7 and H2020). He is the co-author of over 80 publications.