

Generative Music with Stochastic Diffusion Search

Asmaa Majid Al-Rifaie¹ and Mohammad Majid al-Rifaie²

¹ Department of Computing
Goldsmiths University of London
International Programme
as.majid.as@gmail.com

² Department of Computing
Goldsmiths, University of London
London SE14 6NW, United Kingdom
m.majid@gold.ac.uk

Abstract. This paper introduces an approach for using a swarm intelligence algorithm, Stochastic Diffusion Search (SDS) – inspired by one species of ants, *Leptothorax acervorum* – in order to generate music from plain text. In this approach , SDS is adapted in such a way to vocalise the agents, to hear their “chit-chat” . While the generated music depends on the input text, the algorithm’s search capability in locating the words in the input text is reflected in the duration and dynamic of the resulting musical notes. In other words, the generated music depends on the behaviour of the algorithm and the communication between its agents. This novel approach, while staying loyal to the original input text, when run each time, ‘vocalises’ the input text in varying ‘flavours’.

Keywords: swarm intelligence, stochastic diffusion search, generative music, nature-inspired algorithm

1 Introduction

Nature inspired algorithms have been the source of many inspirations in arts and sciences. Swarm intelligence algorithms as one category of nature-inspired algorithms have been used increasingly to solve various optimisation problems. The behaviour exhibited by swarm intelligence techniques are inspired by the interaction of social animals and insects in nature: fish schooling, bugs swarming, birds flocking, ant colonies foraging, bacterial growth, animal herding, brood sorting by ants, etc. Examples of swarm intelligence algorithms are Particle Swarm Optimisation [1], Genetic Algorithm [2] and Ant Colony Optimistion [3].

Many techniques derived from swarm intelligence algorithms have been used to produce generative music. In the last several years, the development of compositional computer programs have been attracting several artists, musicians and researchers. While these approaches are mostly driven forward primarily from

academic theory without the involvements of many composers, generative music has been a thriving field of research [4].

This work presents a novel approach utilising a swarm intelligence algorithm's optimisation capabilities in order to introduce a method for generating music.

In this paper, a swarm intelligence algorithm, Stochastic Diffusion Search, is explained, followed by details on how this algorithm is used to generate music based on an input text. Then a few examples of the generated music are presented and discussed. Conclusion and possible future research are included at the end of the paper.

2 Stochastic Diffusion Search

Stochastic Diffusion Search (SDS) [5, 6], first introduced in 1989, belongs to the extended family of Swarm Intelligence algorithms to solve best-fit pattern recognition and matching problems. SDS has a strong mathematical framework, which describes the behaviour of the algorithm by investigating its resource allocation, convergence to global optimum, robustness and minimal convergence criteria and linear time complexity.

In order to introduce SDS, the Mining Game metaphor is presented (for more details please see [5]):

- At the start of the mining process each miner is randomly allocated a hill (his hill hypothesis, h).
- Then each miner selects a random region on his hill to mine.
- Whether the miner is happy or not depends on whether he finds gold.
- At the end of the day the miners congregate and over the evening each miner who is unhappy selects another miner at random to talk to. If the chosen miner is happy, he happily shares with his colleague the location of his hill (that is, he communicates his hill hypothesis, h , which thus both share and each picks a random region within the shared hill). Conversely, if the chosen miner is unhappy he says nothing and the selecting miner is once more reduced to selecting a new hill (or hypothesis) – identifying the hill he is to mine the next day – at random.

In any SDS search, each agent maintains a hypothesis, h , defining a possible problem solution. SDS has two phases:

- Test Phase (testing gold availability)
- Diffusion Phase (congregation and exchanging of information)

It is shown that using this algorithm, after few days, the miners will be able to find the hill where there is the maximum amount of gold available.

2.1 Method of Communication

Communication is important in all swarm intelligence algorithms, including SDS. In one species of ant, *Leptothorax acervorum*, a tandem calling mechanism (one-to-one) is used for communication. In this method, the ant that finds the resource

location, recruits a single ant on its return to the nest, therefore the location of the resource is physically publicised [7]. In SDS, direct one-to-one communication (which is similar to tandem calling recruitment) is used. However the behaviour of the agents in SDS is simpler than the recruitment behavior of real ants.

2.2 SDS Search Example

In order to show how SDS works one search example will be discussed. The search example here shows how to find a set of letters within a larger string of letters. The goal is to find a 3-letter model (Table 1) in a 13-letter search space (Table 2). For simplicity purposes only three agents are assumed for this example.

Table 1. Model

| | | | |
|-------|---|---|---|
| Index | 0 | 1 | 2 |
| Model | N | O | T |

Table 2. Search Space

| | | | | | | | | | | | | | |
|--------------|---|---|---|---|---|---|---|---|---|---|----|----|----|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Search Space | T | O | B | E | O | R | N | O | T | T | O | B | E |

In this example, a hypothesis, which is a potential problem solution, identifies three adjacent letters in the search space (e.g. hypothesis ‘2’ refers to B-E-O, hypothesis ‘10’ refers to O-B-E). At first each agent initially randomly picks a hypothesis from the search space (Table 3)

- The first agent points to the 5th entry of the search space; in order to partially evaluate this entry, it randomly picks one of the letters (e.g. the first one, R): **R|N|O**
- The second agent points to the 9th entry and randomly picks the third letter (B): **T|O|B**
- The third agent refers to the 1st entry in the search space and randomly picks the third letter (E): **O|B|E**

Table 3. Iteration 1

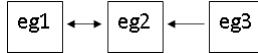
| Agent Num: | 1 | 2 | 3 |
|----------------|-------------------|-------------------|-------------------|
| Hypothesis: | $\frac{5}{R-N-O}$ | $\frac{9}{T-O-B}$ | $\frac{1}{O-B-E}$ |
| Letter picked: | 1 st | 3 rd | 3 rd |
| Status: | X | X | X |

The letters picked are compared to the corresponding letters in the model, which is N-O-T.

- First letter from the first agent (R) is compared with the first letter from the model (N), because they are not the same, the agent is set inactive.
- For the second and third agents, letters ‘B’ and ‘E’ are compared against ‘T’ from the model. Since none of the letters correspond to the letter in the model, the status of the agents are set inactive

In the next step (diffusion phase), each inactive agent randomly selects another agent. If the selected agent is active, the inactive agent adopts the hypothesis of the active agent. If the selected agent is inactive, the selecting agent generates a random hypothesis. As hinted earlier, communications between agents occur during the diffusion phase. Assume that the first agent chooses the second one; since the second agent is inactive, the first agent must choose a new random hypothesis from the search space. Figure 1 shows the communications between agents. The process is repeated for the other two agents. As the agents are

Fig. 1. Agents Communication



inactive, they all choose new random hypotheses (see Table 4)

Table 4. Iteration 2

| Agent Num: | 1 | 2 | 3 |
|----------------|-------------------|--------------------|-------------------|
| Hypothesis: | 6 <i>N-O-T</i> | 10 <i>O-B-E</i> | 0 <i>T-O-B</i> |
| Letter picked: | 2 nd | 3 rd | 1 st |
| Statuse: | ✓ | X | X |

In Table 4, the second and third agents do not refer to their corresponding letter in the model, so they become inactive. The first agent, with hypothesis ‘6’, chooses the 2nd letter (O) and compares it with the 2nd letter of the model (O). Since the letters are the same, the agent becomes active. The same process is repeated for the other two agents, and since the letters do not match the letters in the model, they are set inactive. This process is repeated until all agents are active. It is important to note that the number of agents is irrelevant to the number of letters in the model (e.g. it is possible to have 5 agents and a two-letter model).

3 Generative Music

In generative music, various computational techniques are used; some are closely related to swarm intelligence and some have been generated using other techniques. Generative music based on swarm intelligence uses the dynamic properties of the swarms; these properties are tightly linked to the communication and therefore movement of the swarms throughout the possible search space.

Using these properties, scientists and artists develop creative music. In one such paper, “Music Composition with Interactive Evolutionary Computation” [9] the authors describe a new approach to the music composition by means of interactive evolutionary computation (IEC) which discusses the interactive musical composition system. It is claimed that system can generate musical phrases by combining genetic algorithms and genetic programming.

Another attempt and more recently, “Experiments with Particle Swarm Optimization” [10] uses Particle Swarm Optimisation (PSO) [1] which is a swarm intelligence and evolutionary computation technique, developed in 1995, and is inspired by the social behaviour of bird flocking, to generate music. In particle swarms, members of the swarm neither have knowledge about the global

behaviour of the swarm nor global information about the environment, the local interactions of the swarms result in a complex collective behaviour, such as flocking, herding, schooling, exploration and foraging. In this work, particles are made to follow a hypothetical point (focal point, fp); each agent selects a random point between A and B (fp) and moves to it until they reach the target. This example uses PSO to develop continuous music so swarms have to continue their movement; at the time that any agent's fitness is below a predefined threshold, focal point randomly moves to a new position in a search space and the particles search begins.

For a more detailed account of other related works in generative music using various computational techniques the readers are referred to "Evolutionary computer music" [4].

There are other works related to sonification of text. In one such work [11] a platform is proposed that allows the creation of user-generated mapping for the sonification of text messages and arbitrary clients to sonify text messages using a web-based API. While there are other works generating melodies from text input, SDS generated music is unique in its ability to generate non-identical musics from one input text.

4 Generating Music with SDS

A sentence is formed of few words and each word is comprised of letters. Each letter has its own tone and every word has its individual concept and meaning. With these concepts, humans aim to communicate with each other just like the agents in SDS algorithm. Humans, among other ways, communicate through text and the agents of SDS (as shown in the section 2.2) communicate with each other through the words and letters.

The aim of this paper is to represent an input text as the sound they create; the output sound is based on letters, words and ultimately a longer string of characters. In other words, SDS is adapted in such a way to vocalise the agents, to hear their "chit-chat!" while communicating with each other throughout the search space.

The task of generating music is guided by taking the three below-mentioned parameters into account:

1. Pitch
2. Note Duration
3. Dynamic (or volume)

These values are determined by the input text as well as the behaviour of the algorithm while processing the input through its test and diffusion phases.

The next part explains the link between the above-mentioned parameters, the input text and SDS. Afterwards the process through which SDS is tasked to generate the music is explained.

Table 5. Letters Frequency

| | |
|------------------|---|
| Letter Frequency | ETAON, RISHD, LFCMU, GYPWB, VKJXQ, Z |
| Pairs of Letters | TH HE AN RE ER IN ON AT ND ST ES EN OF TE ED OR TI HI AS TO |
| Doubled Letters | LL EE SS OO TT FF RR NN PP CC |

The relation between pitches and letters Each letter (or pair of letters) is mapped onto a musical note (an individual pitch which has its own MIDI number). In order to assign a MIDI number to a character or pair of characters, letter frequency will be used. Herbert S. Zim [12], in his classic introductory cryptography text “Codes and Secret Writing”, gives the English letter frequency sequence as well as the most common pair of letters and the most common doubled letters (see Table 5).

Music is made of a set of 12 notes and each one of these notes has its individual MIDI numbers. The set of letters in table 5 are divided into 12 separated sets where each will be associated with one of the 12 notes (see Table 6). One of the topics for future research is to conduct an investigation to find a better way in which letters are assigned to their possibly corresponding musical notes.

Using Table 6, an example is given on how to map a simple text (e.g. ‘Hello World’) into the corresponding MIDI numbers (see Table 7).

Note Duration and Dynamic Duration refers to a certain amount of time or a particular time interval which may be described as short/long or with varying duration of time. This property plays a crucial role in forming one of the bases of rhythm within music. In music, dynamic refers to the volume of a sound or a note. Both of these parameters (i.e. duration and dynamic) will be defined using SDS parameters individually; more details are provided below in section 4.2.

4.1 The parameters of SDS

Each agent in SDS has a status which is a boolean value; this entails that an agent can be either active/inactive, true/false, or happy/unhappy. For SDS to generate music, three parameters are used. These parameters are based on the global number of agents (in all iterations) in each of the following categories:

1. Number of Lucky agents (l_g)
2. Number of Happy agents or (h_g)
3. Number of Unhappy agents or (u_g)

Table 6. Frequencies and Notes

| Notes Num | Notes | Sets Freq | MIDI Num |
|-----------|-------|----------------|----------|
| 1 | C | ETAON | 72 |
| 2 | C# | RISHD | 73 |
| 3 | D | LFCMU | 74 |
| 4 | D# | GYPWB | 75 |
| 5 | E | VKJXQ | 76 |
| 6 | F | Z | 77 |
| 7 | F# | TH TE TI TO RE | 78 |
| 8 | G | AN AT AS HE HI | 79 |
| 9 | G# | ON OF OR ND ST | 80 |
| 10 | A | ER ES EN ED IN | 81 |
| 11 | A# | LL EE SS OO TT | 82 |
| 12 | B | FF RR NN PP CC | 83 |

Table 7. Example: Converting each letter or pair of letters to the corresponding musical note and MIDI number

| hello world | | |
|-------------|-------|------------|
| Characters | Notes | MIDI |
| he | G | 79 |
| ll | A# | 82 |
| o | C | 72 |
| | F | space = 77 |
| w | D# | 75 |
| or | G# | 80 |
| I | D | 74 |
| d | C# | 73 |
| | F | space = 77 |

A lucky Agent is an unhappy agent randomly picking an agent whose status is true (i.e. happy). Deciding which agent is lucky happens during the Diffusion phase, whereas determining whether an agent is happy or unhappy occurs during the Test phase.

The following should hold regarding the above-mentioned parameters:

$$NP = h_g + u_g \quad (1)$$

$$l_g \leq u_g \quad (2)$$

where NP is the population size.

In this paper, the population size is set to 20. Given that SDS iterates 10 times, the maximum number of unhappy agents in each iteration is 20; therefore the maximum number of unhappy agents over the whole iterations is 200.

4.2 The relation between SDS and music

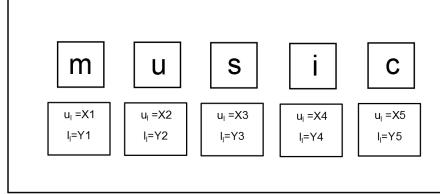
In SDS, a model (or goal) is needed which can be one word. Initially the first word is set as a model, which is searched in the search space using SDS and the above mentioned three values are calculated and used subsequently to generate the duration and dynamic values.

Given that each character or pair of characters are to be converted into a musical note, each word is assigned as a model n times (where n is the length of the model according to table 6; therefore a pair of letters will be considered as one). This process is repeated until reaching the end of the search space.

Searching for each model in the search space results in different values of SDS parameters. The number of lucky agents and unhappy agents will change from word to word. Additionally, even if each word is searched twice, due of the nature of the swarm intelligence algorithm, there is no guarantee that these figures stay the same in each run. In order to generate a musical note for each character in the search space, the mining process has to run as many times as the number of the corresponding musical notes (see Table 7 for an example).

Consider the model to be ‘music’. The number of corresponding musical notes in this word is 5. Therefore, the mining process (i.e. test and diffusion phases) has to run five times, each time generating the necessary information (duration and dynamic) for that particular musical note (e.g. the first run will result in generating relevant values for the duration and dynamic of the musical note ‘m’;

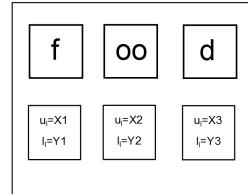
Fig. 2. SDS parameters for ‘music’



the second run, for the musical note ‘u’ and so forth). This way, each character (or pair of characters) has separately generated values for the number of local unhappy agents (u_l) and the number of local lucky agents (l_l) which will be used to calculate the duration and dynamic at a later stage.

See Figure 2 which shows that u_l , l_l values need to be generated for each character (or pair of characters) within the model. As shown in Figure 2, each character of the word ‘music’ is separated, this is because this word does not contain any of the pairs of letters shown in Table 6. However, when the word ‘food’ becomes the model of the algorithm (see Figure 3), the mining process has to run only three times. This is due to the presence of the pair ‘oo’ which forms one musical note (see Table 6).

Fig. 3. SDS parameters for ‘food’



4.3 How SDS generates music

Algorithm 1 explains the process through which SDS processes the input text which will then get converted into music.

Algorithm 1 Generating music with SDS

```

FOR I = 1 TO N
    Choosing I'th word as model

    FOR J = 1 TO M
        Run SDS for T iterations
        Store the numbers of local unhappy and lucky agents
    END J

    Store the numbers of global unhappy and lucky agents

END I

- N is the number of words in the search space.
- M is the number of musical notes in the current model.
- T (the total number of iterations) is set to 10.
- SDS population size is set to 20 (NP = 20).

```

As mentioned earlier, the number of lucky agents impacts the duration of the corresponding note. In other words, a character (or pair of characters) with more lucky agents is “luckier” and plays for longer. The following formulas are used to calculate the duration and dynamic of each note:

$$l_n = \frac{l_g \times S_r}{n} \quad (3)$$

l_n is the normalised lucky agents; l_g is the number of global lucky agents in all iterations; S_r indicates that agents are averaged over 5 (i.e. $S_r = 5$); and n is the total number of notes in the input text.

After normalising the lucky agents, the following is used to determine the duration:

$$du = \frac{\alpha \times l_l}{l_n} \quad (4)$$

where du is the duration; l_l is the number of lucky agents for each note; and α is a constant value that adjusts the duration of the note and is set to 2.

On the other hand, the number of unhappy agents has effect on the volume of the note, which is calculated using the formulas below:

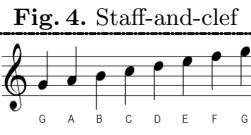
$$u_n = \frac{u_g \times S_r}{n} \quad (5)$$

$$dy = \frac{\beta \times u_l}{u_n} \quad (6)$$

where dy is the dynamic; u_n is the normalised unhappy agents; u_g is the total number of unhappy agents for all the notes; u_l is the number of unhappy agents for each note; and β is a constant value which adjusts the volume of the machine’s speaker and is set to 10000.

4.4 Music Sheet

For each generated music in this paper a music sheet or score is presented. Therefore, the musical notes are placed on the staff according to Table 6. See Figure 4 which illustrates where each note should be positioned on the staff.



Given each note has its corresponding duration (see section 4.2), this needs to be reflected in the generated music as well as the score. Having used the time signature of 4:4 here, whenever the duration of a particular note towards the end of a bar exceeds this figure, a silent note is placed and the note is moved after the bar. The link between the notes and their corresponding time duration is shown in Figure 5.

Fig. 5. Note Duration

| Note Name (US name) | Note Shape | Time Duration |
|---------------------|------------|---------------|
| Whole Note | ○ | 2 (sec) |
| Half Note | ♩ | 1 (sec) |
| Quarter Note | ♪ | 1/2 (sec) |
| Eighth Note | ♫ | 1/4 (sec) |
| Sixteenth Note | ♬ | 1/8 (sec) |
| Thirty Second Note | ♬ | 1/16 (sec) |

4.5 Sample Set of Generated Music

The scores in Figure 6 illustrate two system runs, using a sample input text: “hello music sds welcome to the reality”. The details of musical notes, the corresponding dynamics and durations and the algorithm’s generated values are presented in Table 8. The recorded musics³, correspond to the music sheets and the values in the table. The music library imported to generate the output is Sound Cipher [13], where piano and guitar effects are also added to the music.

The generated musics shown here have a noticeable similarity⁴ both while listening to the music or by looking at the music sheets. This is due to using the same seed (input text) for all three runs. These musics, while exhibiting loyalty towards the input text, have their own unique ‘swarmic flavours’ which distinguish them from one another; this is because each time SDS algorithm is run to process the input text, it returns varying dynamics and durations. This difference in dynamics and duration in each run is the reflection of the searching behaviour of the swarms.

In other words, the generated musics from different inputs have their own musical features; equally, those musics generated from the same seed, demonstrate ‘loyalty’ to the input text while at the same time exhibit their unique ‘swarmic flavours’. Figure 7 shows three runs off the same text.

5 Conclusion

This paper introduced a music generating algorithm based on Stochastic Diffusion Search (SDS) which is a swarm intelligence algorithm mimicking the behaviour of one species of ants, *Leptothorax acervorum*. The input to the system is a plain text which also forms the search space for the swarm intelligence algorithm. Each letter or pair of letters are allocated a musical note; this process is based on the English letter frequency sequence as detailed by Herbert S. Zim [12],

³ Follow this link to listen to three runs of the music generated by the algorithm based on the input text ‘hello music sds welcome to the reality’:
<https://www.dropbox.com/s/bh4icqsdlpz04re/SDSMusic.zip?dl=0>

⁴ While different, the similarities between all three music sheets are evident. In every run, the differences in note values and rest values are noticeable (i.e. by comparing all the first bars of all the three runs with each other, you can see how the note values are different and also there is one rest value in the third run)

Fig. 6. Music sheets showing two runs



in his classic introductory cryptography text “Codes and Secret Writing” where each letter as well as the most common pair of letters and the most common doubled letters are highlighted. The swarm intelligence algorithm then generates the dynamic and duration of each note. This process leads to generating a music each time the system is run.

While the final generated musics from the same input have resemblance with each other (representing the original input text and the corresponding musical notes which are determined by the English letter frequency sequence), due to varying dynamic and duration, which are dependant on the searching behaviour of the swarm intelligence algorithm in each run, the output musics have a unique ‘swarmic flavour’. In other words, while the musics generated from one input text are ‘loyal’ to their input, the behaviour of the swarms induces enough ‘freedom’ to ensure originality in the each resulting music.

Among the topics for future research is to investigate and find a better approach in which letters are assigned to their corresponding musical notes. Additionally rhythm is yet to be fully implemented in the system.

References

1. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science. Volume 43., New York, NY, USA: IEEE (1995)

Fig. 7. Music sheet for the third run



Table 8. Values generated by SDS while processing the input text

| | | First Run | | | | Second Run | | | | Pitch | |
|----|---------|----------------------|---------------------|-----------|----------|----------------------|---------------------|-----------|----------|--------|----|
| | | Before Normalization | After Normalization | Dynamic | Duration | Before Normalization | After Normalization | Dynamic | Duration | | |
| 1 | Model : | hello he | 70 | 18 | 1261 | 0.387 | 48 | 19 | 905 | 0.431 | 79 |
| 2 | | hello ll | 111 | 24 | 2000 | 0.516 | 100 | 23 | 1886 | 0.522 | 82 |
| 3 | | hello o | 81 | 21 | 1459 | 0.451 | 126 | 32 | 2377 | 0.727 | 72 |
| 4 | | hello | 89 | 30 | 1603 | 0.645 | 79 | 21 | 1490 | 0.477 | 77 |
| 5 | Model : | music m | 47 | 18 | 846 | 0.387 | 148 | 18 | 2792 | 0.409 | 74 |
| 6 | | music u | 31 | 15 | 558 | 0.322 | 52 | 20 | 981 | 0.454 | 74 |
| 7 | | music s | 84 | 22 | 1513 | 0.473 | 86 | 19 | 1622 | 0.431 | 73 |
| 8 | | music i | 77 | 22 | 1387 | 0.473 | 82 | 24 | 1547 | 0.545 | 73 |
| 9 | | music c | 134 | 20 | 2414 | 0.430 | 99 | 20 | 1867 | 0.454 | 74 |
| 10 | | music | 114 | 19 | 2054 | 0.408 | 43 | 14 | 811 | 0.318 | 77 |
| 11 | Model : | sds s | 80 | 15 | 1441 | 0.322 | 75 | 20 | 1415 | 0.454 | 73 |
| 12 | | sds d | 143 | 21 | 2576 | 0.451 | 60 | 27 | 1132 | 0.613 | 73 |
| 13 | | sds s | 45 | 17 | 810 | 0.365 | 53 | 18 | 1000 | 0.409 | 73 |
| 14 | | sds | 130 | 20 | 2342 | 0.430 | 44 | 18 | 830 | 0.409 | 77 |
| 15 | Model : | welcome w | 62 | 21 | 1117 | 0.451 | 104 | 16 | 1962 | 0.363 | 75 |
| 16 | | welcome e | 124 | 24 | 2234 | 0.516 | 61 | 17 | 1150 | 0.386 | 72 |
| 17 | | welcome l | 47 | 15 | 846 | 0.322 | 80 | 24 | 1509 | 0.545 | 74 |
| 18 | | welcome c | 134 | 19 | 2414 | 0.408 | 50 | 17 | 943 | 0.386 | 74 |
| 19 | | welcome o | 58 | 25 | 1045 | 0.537 | 97 | 22 | 1830 | 0.5 | 72 |
| 20 | | welcome m | 50 | 19 | 900 | 0.408 | 143 | 20 | 2698 | 0.454 | 74 |
| 21 | | welcome e | 144 | 21 | 2594 | 0.451 | 99 | 23 | 1867 | 0.522 | 72 |
| 22 | | welcome | 139 | 29 | 2504 | 0.623 | 68 | 22 | 1283 | 0.5 | 77 |
| 23 | Model : | to to | 94 | 29 | 1693 | 0.623 | 148 | 26 | 2792 | 0.590 | 78 |
| 24 | | to | 125 | 39 | 2252 | 0.838 | 88 | 21 | 1660 | 0.477 | 77 |
| 25 | Model : | the th | 87 | 23 | 1567 | 0.494 | 85 | 20 | 1603 | 0.454 | 78 |
| 26 | | the e | 61 | 19 | 1099 | 0.408 | 109 | 19 | 2056 | 0.431 | 72 |
| 27 | | the | 101 | 21 | 1819 | 0.451 | 75 | 22 | 1415 | 0.5 | 77 |
| 28 | Model: | reality re | 187 | 10 | 3369 | 0.215 | 190 | 2 | 3584 | 0.045 | 78 |
| 29 | | reality a | 193 | 2 | 3477 | 0.043 | 188 | 8 | 3547 | 0.181 | 72 |
| 30 | | reality l | 181 | 10 | 3261 | 0.215 | 192 | 2 | 3622 | 0.0454 | 74 |
| 31 | | reality i | 189 | 8 | 3405 | 0.172 | 183 | 11 | 3452 | 0.25 | 73 |
| 32 | | reality t | 191 | 5 | 3441 | 0.107 | 184 | 7 | 3471 | 0.159 | 72 |
| 33 | | reality y | 188 | 6 | 3387 | 0.129 | 180 | 7 | 3396 | 0.159 | 75 |
| 34 | | reality | 184 | 10 | 3315 | 0.215 | 185 | 5 | 3490 | 0.113 | 77 |
| | | | Total:3775 | Total:637 | | | Total:3604 | Total:604 | | | |

2. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (1989)
3. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. Computational Intelligence Magazine, IEEE 1(4) (2006) 28–39
4. Miranda, E.R., Al Biles, J.: Evolutionary computer music. Springer (2007)
5. al-Rifaie, M.M., Bishop, M.: Stochastic diffusion search review. In: Paladyn, Journal of Behavioral Robotics. Volume 4. Paladyn, Journal of Behavioral Robotics (2013) 155–173
6. Bishop, J.: Stochastic searching networks, London, UK, Proc. 1st IEE Conf. on Artificial Neural Networks (1989) 329–331
7. Möglich, M., Maschwitz, U., Hölldobler, B.: Tandem calling: a new kind of signal in ant communication. Science 186(4168) (1974) 1046–1047
8. Blackwell, T.: Swarming and music. (2007)
9. Tokui, N., Iba, H.: Music composition with interactive evolutionary computation. In: Proceedings of the 3rd international conference on generative art. Volume 17. (2000) 215–226
10. Herber, N.: Experiments with particle swarm optimization, <http://www.x-tet.com/pf2004-10/psos.html> (2004-2011)
11. Alt, F., Pfleging, B., Schmidt, A.: Sonify-a platform for the sonification of text messages. In: Mensch & Computer. (2013) 149–158
12. Zim, H.S.: Codes and secret writing. W. Morrow (1948)
13. Brown, A.R.: Sound Musicianship: Understanding the Crafts of Music. Volume 4. Cambridge Scholars Publishing (2012)