Intelligent Exploration of Sound Spaces Using Decision Trees and Evolutionary Approach

Gordan Kreković
Faculty of Electrical Engineering and Computing,
University of Zagreb, Croatia
gordan.krekovic@fer.hr

Davor Petrinović
Faculty of Electrical Engineering and Computing,
University of Zagreb, Croatia
davor.petrinovic@fer.hr

ABSTRACT

This paper describes Synthbee, an assistive tool for sound design which enables musicians to achieve desired sounds without managing parameters of a sound synthesizer manually. The system allows musicians to specify desired sound characteristics using attributes and explore the space of producible sounds by controlling the interactive evolutionary algorithm extended to take into account specified attributes. Using the interactive evolutionary approach, musicians can recombine and mutate patches towards a satisfactory result. While performing recombination of patches, the algorithm tries to maintain values of synthesis parameters which are relevant for achieving desired sound characteristics. Synthbee thereby enables efficient creation of novel sounds which possess characteristics described by input attributes. The method for finding and maintaining relevant synthesis parameters during an interactive exploration is our original algorithm which uniquely combines machine learning techniques with evolutionary computing. The results of the initial subjective evaluation of Synthbee showed that the users were generally satisfied with generated sounds, but also indicated some opportunities for improvement.

1. INTRODUCTION

Modern sound synthesis technology opens innumerable possibilities for creating desired sonorities. Musicians have detailed control over the synthesis process what gives them freedom to be more innovative and ambitious expressing their ideas. To provide such a level of flexibility, most commercial software synthesizers have complex architectures and offer a large number of controllable parameters. As a result, the task of sound synthesis becomes difficult and time consuming what negatively affects inspiration and productivity of musicians. One approach to overcome this problem is to use artificial intelligence techniques for automatic selection of synthesis parameters based on musician's requirements or guidance [1].

Since last few decades, this problem was addressed in many previous studies. Several authors focused on using timbral attributes for controlling a sound synthesizer. Based on specified attributes, the goal was to automatically find appropriate synthesis parameters which produce a sound with described characteristics [2-5]. For example, a musician could specify that the sound is expected to be *metallic*, *bright*, and *harsh*, while the system should synthesize such a sound. Controlling a sound synthesizer using timbral attributes is a challenging problem for two reasons. The first one is a lack of theoretical and notational support related to timbre [6]. While other characteristics such as pitch and rhythm have more formal notations, timbral attributes are not standardized. The second reason is complexity and ambiguity of mapping between verbal descriptions and synthesis parameters.

Existing works include several attempts at synthesizing sound specified by timbral attributes. Miranda used a machine learning algorithm based on decisions trees to induce relations between quasi-timbral attributes and synthesis parameters [2]. A research conducted by Gounaropoulos and Johnson employed a neural network to learn relations between adjectives and audio features of a sound characterized by those adjectives [3]. For generating synthesis parameters based on given adjectives, they used a modified version of the backward-propagation algorithm on the same neural network.

Besides using timbral attributes, there are other approaches to generating desired sounds. The most common approach is target matching, i.e. finding the parameters which produce the most similar sound to the given target audio sample. Evolutionary computing techniques such as genetic algorithms were employed to achieve target matching for specific synthesis techniques [7-11].

Another notable approach is an interactive exploration through a sound space of a certain synthesis process [12-14]. The parameters are managed by a genetic algorithm which takes musician's personal judgments as the fitness measure. Using a rich graphical user interface, musicians can control the evolutionary process to produce novel patches in each iteration of the algorithm. The interaction is intuitive, because it consists of browsing and rating the patches.

Motivated by the mentioned publications, we have designed and developed Synthbee, a system which introduces a new way of employing artificial intelligence techniques in sound design. Synthbee allows musicians to specify characteristics of a target sound using textual descriptions and perform a guided exploration of the sound space towards a satisfactory result.

For each possible input attribute, the machine learning algorithm is trained to determine which synthesis pa-

rameters and their values which are relevant for that attribute. The induced relations between input attributes and synthesis parameters are later used in the interactive evolutionary algorithm for narrowing the exploration towards the target sound.

The interactive search starts from an initial set of existing patches allowing musicians to choose favorite patches for genetic recombination. The reproduction process is adapted to maintain parameters relevant for the given input attributes. During the exploration process, novel patches will be generated in every iteration, but their desired characteristics described by input attributes will be maintained. This is radically different from the basic exploration, because the musician can use textual descriptions to direct the exploration process and reach the desired sound more efficiently.

2. SYNTHBEE

2.1 System overview

The system consists of two modules which are independent of an underlying synthesis technique and can work with patches for any synthesizer which supports the *Virtual Studio Technology* (VST) standard. The term patch stands for a sound setting which is represented by a vector of synthesis parameter values. The high-level architecture of Synthbee is shown in Figure 1.

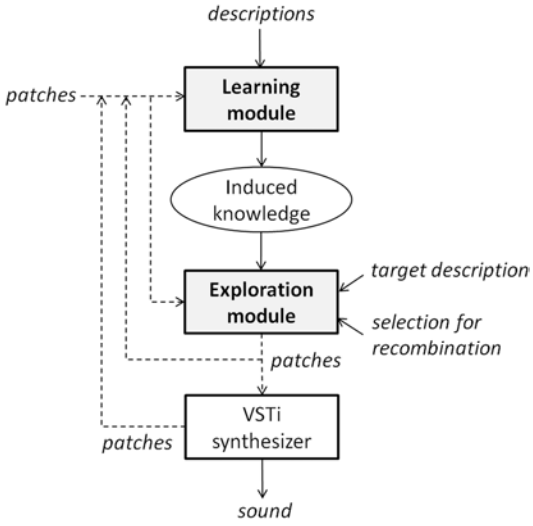

Figure 1. The high-level architecture of Synthbee.

The purpose of the learning module is to detect relations between all available attributes and synthesis parameters. For each attribute, the module learns which synthesis parameters are relevant and what values they should have so that the synthesized sound fits that attribute. For example, if the sound is labeled as *monophonic*, the parameter "unison" should have the value "on". However, relations are usually not as simple as in this example, since most of parameters are continuous and there can be several parameters relevant for one attribute. In some cases it is even impossible to detect relevant parameters for certain attributes, because they can be used inconsistently, or without a sufficient number of training examples.

The exploration module recombines patches selected by a musician to generate novel sounds. During that process, the exploration module uses the knowledge induced by the learning module to maintain relevant parameters during the process of interactive evolution. As an additional feature, the user can assign attributes to any new patch and add it to the pool of learning examples in order to increase the size of the training set and eventually improve accuracy of the learning module.

2.2 Learning module

2.2.1 Attributes

The learning algorithm employed in Synthbee relies on the simplified assumption that the attributes are independent of each other. Miranda devoted a significant part of his work to the problem of the layered organization of attributes [2]. He pointed out that people tend to assemble perceptual qualities of the sound in the more abstract concepts. For example, one could associate the sound of thunder with attributes such as loud amplitude, sharp attack, noisy, low pitch, and medium duration. People are prone to group this information and recall it simply as thunder instead of listing all the attributes separately. Whilst Miranda concentrated on relations within sets of attributes, we focused on relations between attributes and synthesis parameters. For that reason, Synthbee treats all attributes without hierarchical relations and for each attribute the system identifies relevant synthesis parameters in the same way.

A patch description consists of an arbitrary number of nominal attributes. A musician can simply list any attributes which he associates with the sound. For example, a patch can be described with attributes like *soft*, *monophonic*, *metallic*, *slow release*, and so on. Perceptual and taxonomic descriptions are treated in the same way by the learning module. In that sense, a musician can combine perceptual attributes with those from a particular taxonomy (family of musical instruments, instruments names, or sound effects category).

Such flexibility can cause difficulties for the learning algorithm, because a musician can be inconsistent in describing sounds. As a solution to this problem, Synthbee has an option to disable adding new attributes in the system by users. In this mode, musicians can build input descriptions by selecting attributes from the predefined set. With dozens of attributes prepared in the system, musicians still have an expressive vocabulary for forming their requirements, but the results are expected to be significantly better. Another benefit of this approach is that musicians are able to share training sets among them by using a common vocabulary.

2.2.2 Learning algorithm

In order to identify relevant synthesis parameters and their values for each input attribute, the learning module relies on principles of machine learning. For each attribute, the learning module creates a classifier which can determine whether a given patch produces a sound which

suits that attribute or not. The purpose of classifiers in the learning module is not to classify unknown patches, but to reveal relations between the attribute and synthesis parameters.

For example, a correctly trained classifier for the attribute *monophonic* employs the following classifying rules:

- if the parameter "unison" has the value "on", the sound is monophonic,
- if the parameter "unison" has the value "off", the sound is not monophonic.

Those rules are then used by the exploration module in order to fetch relevant synthesis parameters and their values. In this example only the "unison" parameter appears, because other parameters are not relevant for a *monophonic* sound.

The intended purpose of the classification algorithm imposes that classifying rules have to be expressed explicitly and based on relevant synthesis parameters. Time efficiency is not of paramount importance, because the learning module does not need to work in real time. Considering these aspects, decisions trees come as a reasonable choice for such binary classifiers [15].

Decisions trees in Synthbee were designed so that internal nodes represent tests performed on synthesis parameters, branches represent possible outcomes, and leaf nodes specify the overall classification result. As an input, a decision tree accepts a patch which consists of synthesis parameters ranging from 0 to 1 according to the VST standard. An example of a decision tree built by the learning module is shown in Figure 2.

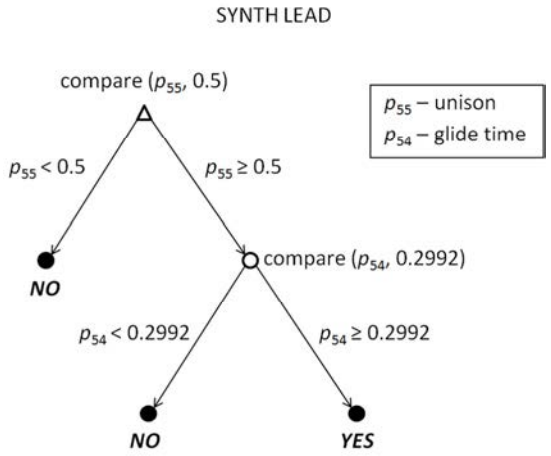

Figure 2. The decision tree for the attribute *synth lead*.

The learning module induces decision trees using the Classification and Regression Tree algorithm, CART [16]. Selection of splitting values in internal nodes is based on Gini index. The first step of the learning process is the construction of complete trees without pruning. Such trees can be too complex and over-fitted to training samples. Therefore, they have to be optimized by cutting off insignificant nodes and subtrees. This is done in the second step by pruning based on 10-fold cross-validation. The concrete implementation of the learning module in the initial version of Synthbee relies on existing functions from Matlab Statistics Toolbox.

A learning set for each tree is formed of all available patches. In our case trees are binary classifiers, so learning samples must be labeled with either a positive or negative goal predicate. Namely, for each patch, there should be the label "yes" if the patch satisfies the attribute for which we are building the tree or the label "no" otherwise. For that reason, patches containing that attribute in their descriptions are labeled as positive samples, while the others are considered as negative. For example, to train a decision tree for the attribute *synth lead*, the patches described with the *synth lead* attribute are taken with the positive goal predicate, while the others with the negative one. This example is shown in Figure 3.

Labeled patches

	Parameters	Attributes
1	0.0211, 0.1234, 0.3593, ..., 0.500	*synth lead, mono, buzzy*
2	0.1245, 0.0000, 0.7734, ..., 0.000	*mellow, soft, pad, slow release*
3	1.0000, 0.2364, 0.2345, ..., 0.500	*synth lead, dark, mono, simple*
4	0.0000, 0.7352, 0.5266, ..., 0.000	*bright, brass, vibrato*

Training set for the attribute *synth lead*

	Parameters	Goal
1	0.0211, 0.1234, 0.3593, ..., 0.500	*Yes*
2	0.1245, 0.0000, 0.7734, ..., 0.000	*No*
3	1.0000, 0.2364, 0.2345, ..., 0.500	*Yes*
4	0.0000, 0.7352, 0.5266, ..., 0.000	*No*

Figure 3. A simple example of forming a training set.

2.3 Exploration module

2.3.1 Target description

The purpose of the exploration module is to recombine patches selected by the user in a way that desired characteristics given by input attributes are maintained in new patches. When specifying attributes, the user can choose among those for which decision trees have been built successfully. As explained before, the learning algorithm may not be able to build a decision tree for every attribute.

The exploration algorithm always uses the current input attributes, so the target description can be changed at any moment during the exploration. This way, musicians can narrow or change direction of the exploration in every iteration.

2.3.2 Exploration process

An exploration process consists of sequential reproductions and mutations following the basic principles of genetic algorithms [17]. There is no automatic fitness measurement, since musicians manually choose patches for reproduction. The seed patches from the initial population are chosen from the set of labeled patches so that they meet the target description.

The unique feature of the exploration algorithm in Synthbee is that relevant parameters and their values are maintained in the reproduction process. In analogy to the

biological evolution, relevant parameters can be considered as dominant genes.

This extended exploration algorithm can be described by the following steps:

1. generate two offspring using the standard genetic recombination of selected parent patches,
2. optionally apply the mutation operator to the offspring to increase differentiation from the parent patches,
3. check whether the offspring fit the input description; if not, take values of relevant parameters from the parent that fits the description.

The third step is the most innovative part of this algorithm and will be explained in more details later.

2.3.3 Reproduction types

A reproduction operator generates two offspring for a couple of parent patches by recombining vectors of synthesis parameters. The exploration module supports two types of reproduction operators.

The first recombination type uses one crossover point randomly selected somewhere within a patch using the uniform probability distribution. The crossover point splits the parent patches in two parts. The first child is formed from the first part of the parent A and the second part of the parent B. Similarly, the second child is formed from the first part of the parent B and the second part of the parent A as shown in Figure 4. This recombination type preserves clusters of parameters and is expected to provide more predictable results than the second recombination type.

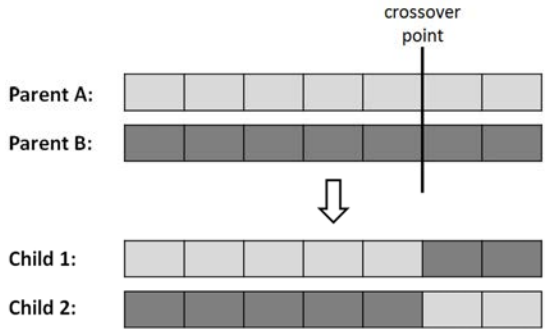

Figure 4. An example of genetic recombination which uses the first type of reproduction technique.

The second recombination type uses two randomly chosen crossover points which split the parents into three parts. Initially, two offspring are produced as parent clones. Parts before the first and after the second crossover point stay the same as in the parents, while each of parameters between the crossover points is overwritten by a corresponding parameter from the randomly selected parent. The first child has the parameters of the parent A before the first and after the second crossover point, but has mixed parameters in between as shown in Figure 5. The second has the parameters from the parent B where the first child has the parameters from the parent A. In other words, at this moment there is no parameter which both of the offspring inherit from the same parent. This recombination technique may disrupt clusters of parameters and result with radically different and sometimes inaudible or noisy sounds.

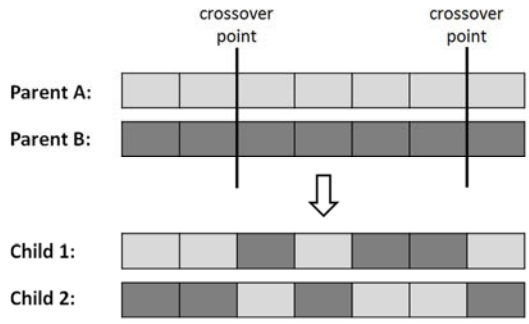

Figure 5. An example of genetic recombination which uses the second type of reproduction technique.

2.3.4 Mutation

In the process of genetic recombination, the offspring inherit parameters from parents. If new patches are not added during the exploration process, the algorithm will just recombine the same limited set of parameter values. To introduce new genetic material, the mutation operator can be applied to offspring patches. In Synthbee each parameter is mutated by adding a Gaussian random variable with zero mean and variance of 0.05.

2.3.5 Maintaining relevant parameters

The children patches generated by the standard recombination process might not fit the target description. In such cases, the algorithm will make a correction by taking values of relevant parameters from the parent which possesses appropriate values. For example, if the target sound has to be monophonic, but the child patch inherited the *unison* parameter with the value "off" from the polyphonic parent, this algorithm will check if the other parent is monophonic and then take its value. The purpose of the algorithm for maintaining relevant parameters is to find a subset of parameters which should be taken from the different parent so that the final patch fits the target description.

This original method for correcting results of genetic recombination relies on the knowledge induced by decision trees. Information about relevant parameters is stored along paths which start with the root of the tree and end with a positive leaf. Nodes on those paths represent conditions which a given patch needs to fulfill in order to be classified positively according to that attributes. For the *synth lead* attribute shown in Figure 2, there is only one positive path which contains conditions on parameters p_{55} and p_{54}. If the parameters of a given patch fulfill those conditions, the patch is classified as a synth lead.

Generally, there can be more than one positive leaf node, so the algorithm has to consider all positive paths when searching for relevant parameters in the decision tree. For example, the decision tree for the attribute *short* shown in Figure 6 has two positive leaves. To achieve the attribute *short*, either the parameter p_{11} must be lower than 0.0472, or the p_{11} must be equal or higher than that value, while the p_{32} is lower than 0.0236.

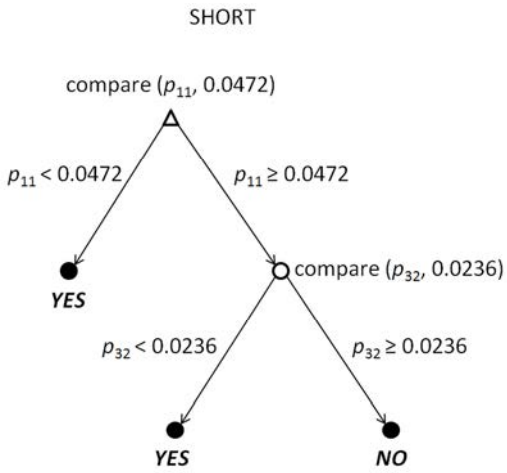

Figure 6. The decision tree for the attribute *short*.

In order to make appropriate corrections in offspring patches, the algorithm needs to check all positive paths and find the one which requires the minimal number of corrections. A correction of a parameter means taking the parameter value from the different parent. The more parameters are corrected in a child patch, the more it will be similar to the other patch generated from the same parents, because they will have more parameters with the same values. In order to keep differentiation within the offspring, the algorithm should change the minimal number of parameters, but still correct the patch to satisfy input attributes.

To find the optimal set of corrections for the offspring we designed a specific method which starts from a leaf node and travels to the root counting needed corrections. After all the positive paths are considered, it is known whether the correction is possible and what parameter values should be corrected in the offspring patch P_o. This method consists of the following steps:

1. set the current node to a leaf node,
2. set the number of replacements n_r to zero,
3. if the current node is the root, finish the algorithm and return the number n_r,
4. read the condition from the branch which connects the current node with its parent node,
5. check whether the patch P_o fulfills that condition,
   5.1. if the condition is fulfilled, go to the step 6,
   5.2. if the condition is not fulfilled, check both of the parent patches,
   5.3. if one of the parents fulfills the condition, remember its parameter, increase the number of replacements n_r by one and continue with the step 6,
   5.4. if none of the parents fulfills the condition, set the number of replacements n_r to infinity and exit the algorithm,
6. set the parent node to be the new current node,
7. go to the step 3.

The result of running these steps from all leaf nodes will be a number of replacements and a list of corrected parameters for each positive path. If the minimal number of replacements is greater than the length of the longest positive path in the tree, it is not possible to fulfill the target description by replacing certain parameters with those from the different parent. That happens when none of the parent patches has the desired characteristic.

When there are more viable positive paths, the algorithm chooses the one with the smallest number of corrections. Lists of corrections are memorized during the algorithm for each positive path and the shortest list is applied to the child patch.

This principle will be illustrated by a simple example. The only input attribute in the target description for this example is the attribute *short*. The first patch chosen for reproduction has the parameters p_{11} = 0.0324 and p_{32} = 0.8456, while the other one has p_{11} = 0.5186 and p_{32} = 0.0000. Obviously, both of the patches match the attribute *short* according to the decision tree for that attribute (see Figure 6). After genetic recombination, a child patch, for instance, inherits the parameter p_{11} from the second parent and the parameter p_{32} from the first parent. Such a patch does not fulfill the target description, so the parameters have to be corrected. There are two possibilities: to take p_{11} from the first parent, or to take p_{32} from the second parent. The both possibilities require one replacement, so they are equally acceptable. After replacing the parameter value, the corrected patch will satisfy the target description.

It is important to notice that this algorithm corrects eventual damage done by mutation. In the mutation process an eligibly inherited parameter can be changed, so it should be set back to the original value. The algorithm natively takes care of such cases, because both of the parent patches are checked in the step 5.2.

2.3.6 Multiple attribute descriptions

The algorithm explained so far works for input descriptions which consist of a single attribute. This section describes how the algorithm is extended to accept descriptions formed of multiple attributes. The goal of this extension is to correct a patch so that it satisfies as many attributes from the input description as possible, while the number of corrected parameters is the secondary criterion. The main principle of this extended algorithm is based on generating corrections for each attribute separately and then merging all the corrections.

In the first step, the offspring patches are corrected for each attribute using the method described in the previous section. However, instead of saving only patches with the minimal number of corrections, the algorithm has been modified to memorize all of the corrected patches. The result is a set of corrected patches C_i for each attribute from the input description.

In the second step, the algorithm chooses one patch from each set C_i and forms the so-called merging set M. All the patches from the merging set are then combined so that the resulting patch satisfies as many attributes as possible.

When forming the merging set M, the selection of patches from the sets C_i is very important. In the merging set there should the minimal number of patches for which corrections have been performed on the same parameters. The optimal situation is the one in which each candidate has corrections of different parameters. In that case,

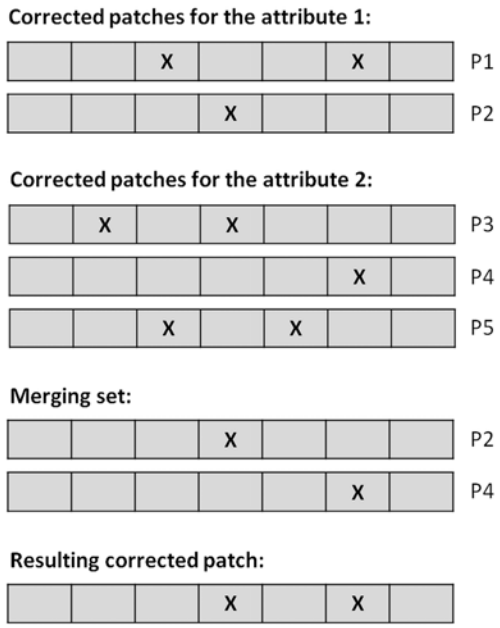

Figure 7. An example of forming a merging set and applying the corrections to the resulting patch. Parameters with corrected values are marked with an X.

merging is very simple, since all corrected parameters can be taken from their patches unambiguously. The resulting patch will maintain all relevant parameter values and satisfy all the input attributes. This situation is illustrated in Figure 7.

To achieve the minimal number of overlapping corrections in the merging set M, before selecting patches from the sets C_i, our algorithm checks all possible combinations. The complexity of this operation is proportional to the product of the cardinalities $| C_i |$. Those sets are not expected to be large due to small decision trees, so time efficiency should not be a problem. After checking all the combinations, the algorithm chooses the one with minimal overlapping of corrected parameters.

If the best combination still has patches whose corrected parameters overlap, for each such parameter, the algorithm fetches conditions from decision trees. Then it tries to find a value which satisfies all conditions if possible. For example, if the parameter p_{62} must be greater than 0.1 to satisfy the first attribute and smaller than 0.4 to satisfy the second attribute, the algorithm will take some value between 0.1 and 0.4. This example is illustrated in Figure 8. If conditions are conflicting, the parameter will not be corrected and it will maintain the value obtained after the recombination and mutation.

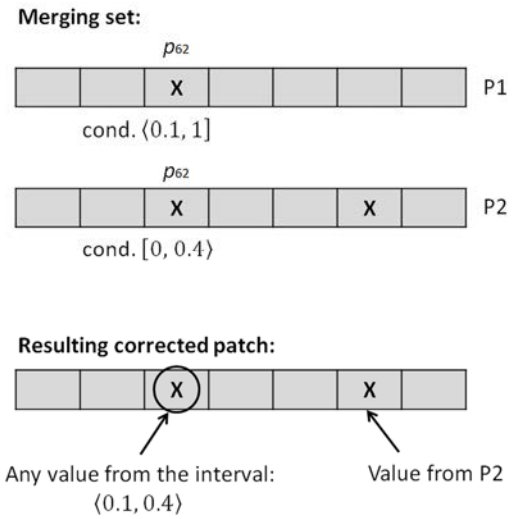

Figure 8. An example of resolving overlaps in the merging set.

3. EVALUATION

To ascertain the effectiveness of Synthbee in practical tasks of sound design, we conducted a survey among six amateur musicians. They were asked to grade how well patches generated by the system matched the input attributes and to which extent they differed from their parents. Additionally, participants answered a set of general questions about using attributes for controlling a sound synthesizer. This chapter describes technical details of the system setup and the evaluation procedure.

3.1 Synthesizer

For the evaluation purposes we used a subtractive VST synthesizer consisted of: two oscillators, a white noise generator, a low frequency oscillator, a filter with its envelope generator, an amplitude envelope generator, and a delay effect. These elements have totally 69 controllable parameters allowing users precise control over the synthesized sound. The synthesizer comes with a wide variety of predefined patches covering sounds from strings, leads, pads, and bells to electric percussions.

3.2 Attributes

Before the first use of Synthbee, we manually labeled 201 patches by assigning attributes to each of them. The attributes were chosen from different taxonomies and had different levels of abstraction. Some examples are: *dark*, *mellow*, *short*, *noisy*, *resonating*, *strings*, *aggressive*, etc. There were totally 27 different attributes used to describe patches for the training set.

3.3 Learning module

After running the learning algorithm, 11 out of 27 decision trees were successfully created. The remaining trees were reduced to the root by pruning, because each of their attributes appeared in less than 10% of the training samples, so there were too few positive samples to build reliable classifiers.

The average accuracy for the 11 properly induced trees was 90.88%. The accuracy was calculated using a 10-fold cross-validation after the pruning. The average number of nodes after pruning was 4.83, while the average height was 1.83. In most cases it was sufficient to test just one or two synthesis parameters to make a plausible decision whether a patch satisfied the attribute or not. The list of all attributes with properly induced decision trees is shown in Table 1.

Attribute	*N* before pruning	*N* after pruning	Accuracy
Synth lead	21	5	91.04%
Delay	27	5	81.59%
Mono	15	3	94.03%
Synth bass	11	5	93.53%
Vibrato	19	3	88.55%
Slow variation	21	3	87.56%
Modulation	13	3	90.05%
Sequencer	3	3	100.00%
Deep	9	5	97.01%
No sustain	9	5	95.02%
Glide	11	7	95.02%
No velocity	25	11	77.14%

Table 1. This table enlists all the properly induced decision trees with number of nodes before pruning, after pruning, and the accuracy of the tree calculated using cross-validation.

3.4 Survey preparation

To create a set of patches for the survey, 20 pairs of parent patches were randomly selected from the initial learning set. For each pair, we prepared a target description which is expected to be satisfied by their offspring. Every target description was intentionally composed only of attributes selected from the union of the parents' attributes. This way, we eliminated situations in which offspring cannot inherit parameters relevant for a desired attribute. This setup also corresponds to the real scenario of using Synthbee, since initial patches, which do not match the target description, are automatically filtered out at the beginning.

The average number of attributes in sound descriptions was 2.5. After applying the reproduction algorithm with recombination, mutation, and parameter correction on parents, they generated 40 offspring which were used for this survey.

In order to evaluate both of the recombination types (explained in Chapter 2.3.3), the first half of the offspring patches was reproduced using the first type of recombination with one crossover point, while the other half was reproduced using the second type of recombination with two crossover points.

For each of the 40 generated patches, participants were asked to provide their subjective opinion on how well does the patch fit the given attributes. Participants rated patches using the scale with 5 available options ranging from very poor (1) to very well (5). Additionally, for each patch, participants were asked to assess how different the patch is from its parents using the scale ranging from very similar (1) to very different (5).

At the end of the survey participants were informally asked to answer these general questions regarding automation in sound design:

1. How clear were the given descriptions based on attributes? (1 - very unclear, 5 - very clear)
2. According to your opinion, are descriptions based on attributes intuitive for musicians? (1 – not intuitive at all, 5 - very intuitive)
3. How helpful is the system for intelligent exploration of sound spaces? (1 - very little, 5 - very much)

4. RESULTS

The average grade for the questions concerning how well the generated sounds met the given descriptions was 3.96 and the median grade was 4. These statistical values were calculated taking into account answers from all six participants for all 40 assessed sounds. There was no significant correlation between grades and numbers of attributes in descriptions. The average grade for the patches produced using the first recombination type was 3.92, while the average grade for the patches produced by the second recombination type was 4.01. Distribution of all these grades is shown in Figure 9.

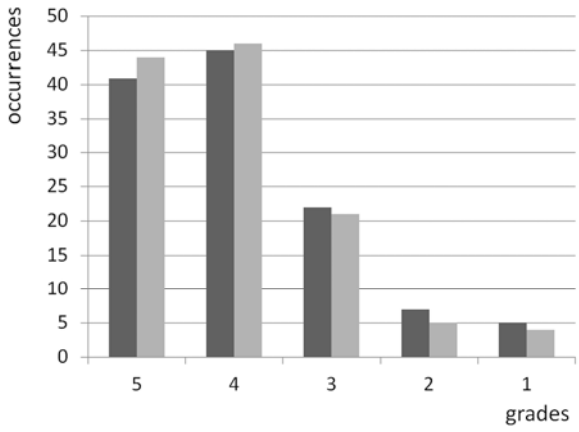

Figure 9. This chart shows how many times each grade appeared in the answers from all participants. The darker bars represent grades for the patches generated using the first recombination type, while the lighter ones represent grades for patches generated using the second recombination type.

The average grade for the questions concerning differences between patches and the parents was 2.66. The patches generated using the first recombination type were averagely rated with higher grades (2.84) than the patches generated using the second recombination type (2.51). This means that the latter patches were unexpectedly rated as more similar to their parents.

The negative correlation of -0.38 was observed between grades from the first and the second question. The patches, which are more similar to their parents, match the target description slightly better.

The average grades on last three questions were 3.37, 4.00, and 4.33 respectively for clarity of such descriptions, intuitiveness of using attributes, and potential usefulness of our approach.

5. DISCUSSION

The results of the evaluation have shown that users are generally satisfied with sounds generated by Synthbee. An interesting finding is that the recombination with two crossover points produces offspring which are more similar to parents than those produced by the recombination with one crossover point. This is contrary from what we expected and what was indicated in other papers [12]. Apparently, for this particular sound synthesizer, there are no many clusters of interdependent parameters, so mixing adjacent parameters from different parents did not

seriously affect the synthesized sound. Using this recombination type, each offspring inherits more parameters from one parent what turned out to be more important for similarity then maintaining clusters of parameters together.

Another important finding is that the free choice of attributes for labeling training samples and creating target descriptions may have a negative consequence. In our case, only 11 of 27 attributes resulted with a properly built decision trees. A few metonyms appeared in the learning set, and for several attributes there was insufficient number of positive training samples. We believe that using a carefully predefined set of available attributes could lead musicians to use them more consistently and with better understanding of their meanings. At the same time we do not expect that limiting the vocabulary would seriously affect its expressiveness.

The answers on the last three questions in the survey suggest that the problem of synthesizing sounds using attributes and interactive exploration is relevant and interesting for musicians.

6. CONCLUSIONS

The results of this study indicate that this unique combination of machine learning and interactive evolutionary techniques may yield some valuable tools for sound design. Using descriptions of desired sound characteristics, musicians could direct the exploration of sounds and make it more efficient.

In the context of interactive evolutionary techniques applied to practical tasks, it is important to balance between introducing diversity among suggested results and maintaining the desired characteristics. This is generally difficult to achieve, since our research showed a negative correlation between these two aspects. However, by tuning parameters of the algorithm and employing the appropriate recombination method, we believe that the algorithm can be adapted for a specific application.

The overall results are encouraging and indicate that combinations of machine learning and evolutionary technique could open a new direction in automation of sound design tasks.

7. REFERENCES

[1] E. Miranda, *Computer Sound Design: Synthesis Techniques and Programming*, second edition, Focal Press, Oxford, UK, 2002.

[2] E. Miranda, "An artificial intelligence approach to sound design", Computer Music Journal, The MIT Press, vol. 19, no. 2, pp. 59-75, 1995.

[3] A. Gounaropoulos, C. Johnson, "Synthesizing timbres and timbre changes from adjectives/adverbs" in P. Collet et al. *Applications of Evolutionary Computing*, Springer-Verlag, Berlin, 2006.

[4] R. Ethington and B. Punch, "SeaWave: a system for musical timbre description", Computer Music Journal, vol. 18, no. 1, pp. 30-39, 1994.

[5] A. Pošćić, G. Kreković, "Controlling a sound synthesizer using timbral attributes", In Proceedings of the International Computer Music Conference, Stockholm, Sweden, 2013.

[6] T. Wishart, *On Sonic Art*. Taylor & Francis Group, 1996.

[7] A. Horner, J. Beauchamp, L. Haken, "Genetic Algorithms and Their Application to FM, Matching Synthesis", Computer Music Journal, vol. 17, no. 4, pp. 17–29, 1993.

[8] A. Horner, "Envelope matching with genetic algorithms", Journal of New Music Research, vol. 24, no. 4, pp. 318–34, 1995.

[9] J. Riionheimo, "Parameter Estimation of a Plucked String Synthesis Model via the Genetic Algorithm", MSc Thesis, Helsinki University of Technology, Finland, 2004.

[10] M. Chinen, N. Osaka, "Genesynth: Noise band-based genetic algorithm analysis/synthesis framework", Proceedings of the International Computer Music Conference, Copenhagen, Denmark, 2007.

[11] M. J. Yee-King, M. Roth, "SynthBot – An Unsupervised Software Synthesizer Programmer", Proceedings of the International Computer Music Conference, Belfast, N. Ireland, 2008.

[12] J. Mandelis, "Genophone: An Evolutionary Approach to Sound Synthesis and Performance", Proceedings of the Artificial Life Models for Musical Applications Workshop, Prague, Czech Republic, 2001.

[13] C. Johnson, "Exploring the Sound-Space of Synthesis Algorithms Using Interactive Genetic Algorithms", Proceedings of the AISB Workshop on Artificial Intelligence and Musical Creativity, 1999.

[14] J. McDermott, Evolutionary computation applied to the control of sound synthesis, PhD thesis, University of Limerick, Ireland, 2008

[15] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, 2003.

[16] L. Breiman, J. Friedman, R. Olshen, C. Stone, *Classification and Regression Trees*, CRC Press, Boca Raton, 1984.

[17] D. E. Goldberg, *Genetic Algorithms in search, optimization and machine learning*, Addison-Wesley, Reading, Massachusetts, 1989.