International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 10, No.1 (Feb-2021)

http://dx.doi.org/10.12785/ijcds/100138

Top 10 Artificial Intelligence Algorithms in Computer Music Composition

Nermin Naguib J. Siphocly[1], El-Sayed M. El-Horbaty[1] and Abdel-Badeeh M. Salem[1]

[1]Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

Received 02 Sep. 2020, Revised 28 Dec. 2020,, Accepted 24 Jan. 2021, Published 8 Feb. 2021

Abstract: Music composition is now appealing to both musicians and non-musicians equally. It branches into various musical tasks such as the generation of melody, accompaniment, or rhythm. This paper discusses the top ten artificial intelligence algorithms with applications in computer music composition from 2010 to 2020. We give an analysis of each algorithm and highlight its recent applications in music composition tasks, shedding the light on its strengths and weaknesses. Our study gives an insight on the most suitable algorithm for each musical task, such as rule-based systems for music theory representation, case-based reasoning for capturing previous musical experiences, Markov chains for melody generation, generative grammars for fast composition of musical pieces that comply to music rules, and linear programming for timbre synthesis. Additionally, there are biologically inspired algorithms such as: genetic algorithms, and algorithms used by artificial immune systems and artificial neural networks, including shallow neural networks, deep neural networks, and generative adversarial networks. These relatively new algorithms are currently heavily used in performing numerous music composition tasks.

Keywords: Computer Music Composition, Machine Learning Techniques, Artificial Intelligence, Music Composition Tasks

1. Introduction

Thanks to the advancements in computer music composition, it is of no surprise today to find non-musicians composing very nice music, even on-the-go. The desire to compose music with the aid of computers returns back to the early days of computer invention. A common belief is that the first musical notes produced by a computer were heard in the late 50's (Illiac Suite) by Hiller et al. [1] through ILLIAC I computer at the University of Illinois at Urbana–Champaign. However, a recent research by Copland et al. [2] shows that musical notes produced from computers were heard even earlier, in the late 40's, by Alan Turing; the father of modern computer science himself. Although not intended primarily to compose music, Turing emitted repeating clicks from the loudspeaker of his Manchester computer with certain patterns; which were interpreted by human ears as continuous sound or musical notes. Building on this and using the same Manchester computer, Christopher Strachey, a talented programmer, succeeded in 1951 to develop a program that plays Britain's national anthem "God Save the King" [3] along with other melodies, which were then recorded and documented by the BBC [4]. It is still undeniable that Hiller's research [1] results attracted researchers more to the field of computer music composition.

Music composition is the task of devising a new musical piece that contains three main components: melody, accompaniment, and rhythm. Fig. 1 labels the main musical tasks lying under the computer music composition umbrella, further with their types. Melody generation is devising the musical notes pitch; "melody" is the group of consecutive notes forming the musical piece while a note "pitch" [5] is the human interpretation of the note's frequency that distinguishes it from other notes. Timbre [6], another aspect of the musical piece's melody, is defined by the American Standard Association (ASA) to be "That attribute of sensation in terms of which a listener can judge two sounds having the same loudness and pitch are dissimilar" [7]. Thus, timbre generation helps in the interpretation of musical instruments that play the melody. Music accompaniment has four types: counterpoint, chorales, chord, and bass. Counterpoint [8] is a special type of music accompaniment that represents the harmony between multiple accompanying voices (typically two to four) generated by a set of strict rules. Chorale [9] accompaniment is formed of four-part music lines; soprano and three other lower voices. Chord [10]

E-mail: nermine.naguib@cis.asu.edu.eg, shorbaty@cis.asu.edu.eg, absalem@cis.asu.edu.eg

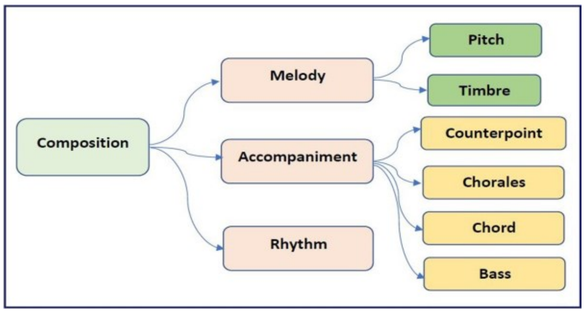


Figure 1. Computer Music Composition Tasks

accompaniment is a prominent type of harmony where a chord is the group of multiple harmonic notes that sound agreeing when heard. Closely related to chord accompaniment generation is bassline [11] generation. Musical piece's rhythm [12] controls its speed and style; it is the beat of the piece.

Computers can aid, either fully or partially, in each of the aforementioned musical tasks. Along the years, research has been carried out for developing algorithms that automates each of these musical tasks, which lead to the term "algorithmic composition". Artificial Intelligence (AI) had a great share of the research in algorithmic composition since teaching computer the various music composition tasks needs high levels of creativity. Computer music composition is all about emulating human creativity in music and that specifically is the challenge of AI [13].

The machine learning field is a subset of AI that is concerned by how computers learn from the given data. Instead of explicitly instructing computers how to perform tasks step-by-step, machine learning techniques enables computers to interpret relationships from the given data and accordingly perform tasks such as classification, clustering, regression, and prediction.

In this paper, we study the top ten AI algorithms used in computer music composition with their applications. Also, the most recent machine learning techniques used in music composition for automating the various musical tasks are discussed. We first give an overview of each algorithm; its description, technical background, or pseudocode as needed. Then, we discuss its applications in the field of music composition. Our main focus is on the applications that have been developed in the last ten years. We consider very few older papers due to their high impact in the field. Finally, we list the strengths and weaknesses of each algorithm. Kindly note that we only focus on the music composition field; there are other fields of computer music generation that are not of interest in this work; such as improvisation (where the computer plays on-the-go harmonic music with human players) and expressive performance (which is concerned with simulating the personal touch of music players).

The rest of this paper is organized as follows: Sections 2 to 11 list each of the algorithms under study along with their recent applications in computer music composition. More specifically: Section 2 describes rule-based systems, Section 3 discusses case-based reasoning systems, Section 4 describes Markov chains generally focusing on the hidden Markov model. Section 5 describes generative grammars focusing on the Chomsky hierarchy. Section 6 describes linear programming. Section 7 elaborates on genetic algorithms. Section 8 discusses artificial immune systems. Sections 9 and 10 cover shallow and deep artificial neural networks respectively. Finally, Section 11 sheds the light on one of the most recent and promising machine learning techniques used in music composition which is the use of generative adversarial networks. We compare the presented algorithms and discuss their merits in Section 12. Finally, we conclude our survey in Section 13.

2. Rule-Based Systems

Rule-Based systems (sometimes known as knowledge-based or expert systems) are means of capturing human knowledge in a format comprehensible by computers. Rule-based systems mainly aim to aid humans in decision making; for example, medical expert systems help doctors in reaching the right diagnosis for patients based on the given symptoms.

2.1 Overview and Description

A rule-based system has three main components:

1. Knowledge Base (KB): set of IF-THEN rules representing the captured knowledge. The IF part of a rule is called antecedent and the THEN part is called consequence. The KB might also contain facts (known assertions).
2. Inference Engine (IE): component responsible for inferring or deducing new information from the knowledge base according to the system input. It matches the rules in the knowledge base with the current state of the world present in the working memory.
3. Working Memory (WM): storage holding temporary data (assertions) about the current state.

```
While TRUE do
    Find all matches between KB and WM
    for each match M do
        if M does not change WM then
            Mark M as defunct
        end if
    end for
    if Number of unmarked matches = 0 then
        TERMINATE
    end if
    if Number of unmarked matches > 1 then
        Call Resolve_Conflict
    end if
    Fire matched rule
    Assert rule consequence in WM
end while
```

Figure 2. Forward Chaining Algorithm

The inference engine can infer data through either forward or backward chaining. Pseudocode of the forward chaining inference algorithm is listed in Fig. 2. The idea behind forward chaining is simply to find all the possible matches from the knowledge base that are relevant to the current state of the working memory. The forward chaining algorithm then makes sure that a matched rule only fires if its consequences will affect the working memory (change its current state). After marking all the defunct rules; which are rules that will not affect the working memory, the algorithm will continue only if the number of unmarked matches is not equal to zero. A conflict resolution technique might be needed if there are more than one unmarked match. Conflict resolution helps to select only one matched rule to fire. The selected rule then fires, and its consequences are asserted in the working memory. The process repeats until no more matches will affect the working memory. Forward chaining is known as data-driven approach since it starts from the already given facts to extract more information. Thus, it is seen as a bottom-up approach.

On the other hand, backward chaining aims to prove whether a goal is true or not. Backward chaining pseudocode is listed in Fig. 3. The algorithm first matches the goal with the knowledge base facts; if a match is found, it returns true. If no fact matches the goal, all rules consequences are checked whether they match the goal. If the goal does not match any of the rules' consequences, the algorithm returns false. Else, backward chaining is called recursively on all the matched rules antecedents keeping track of all the bound variables in the recursion process. If all the recursive calls return true, the algorithm is said to succeed and returns true. Backward chaining is called goal-driven inference approach since it starts with the goal and searches for a possible match in the knowledge base, which makes it a top-down approach.

2.2 *Rule-Based Systems in Algorithmic Composition*

Rule-based systems are convenient for counterpoint type of harmony since it is based on rigid rules for generating harmonic voices. Aguilera et al. [14] coded counterpoint rules with the help of probabilistic logic

```
Match Goal with KB
if a match is found then
    return TRUE
end if
Match Goal with rules consequences
if Match(es) is/are found then
    for each matched rule antecedent A do
        Call Backward Chaining(A)
    end for
    if all recursive calls return TRUE then
        return TRUE
    else
        return FALSE
    end if
else
    return FALSE
end if
```

Figure 3. Backward Chaining Algorithm

using Derive 6 software which is a computer algebra system. Navarro-Cáceres et al. [10], encoded rules about chord construction and progression in a penalty function. Then, they employed an Artificial immune system (discussed in Section 8) to suggest the next chord in a given sequence such that this chord minimizes the penalty function.

3. Case-Based Reasoning

The most common definition for Case-Based Reasoning (CBR) is that it is an approach for solving problems by means of previous experiences. CBR aims for simulating human reasoning and learning from past experiences. Hence, it is a method of retrieving and reusing successful solutions from past problems. Unlike rule-based systems that keep general knowledge about the problem domain, CBR employs the knowledge of previous solutions for specific problems in the domain. Moreover, CBR systems keep adding new knowledge from learned experiences of newly solved problems.

3.1 *Overview and Description*

A CBR system keeps information about previous problems in what is called a case base. A "case" has the description of a previously solved problem along with its solution. The three main operations in the CBR cycle are: case retrieval, adaptation, and storing. Fig. 4 shows a diagram for the CBR cycle. When a new problem is presented to the system, the similar cases to that problem are retrieved from the case base. The next step is to adapt

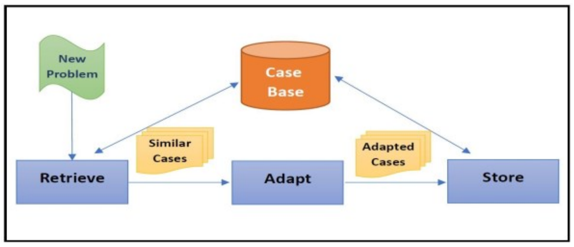


Figure 4. CBR cycle diagram

the retrieved cases and their solutions to meet the demands of the new problem. Finally, the newly solved problem is stored in the case base for future retrievals.

Design decisions of CBR systems include how to represent the case, which also affects the retrieval quality. Case representation includes mainly some parameters or descriptors for the state of the world when the problem happened accompanied by the solution that worked for solving the problem. Another design issue in the retrieval process is the matching algorithm for comparing the new problem with the previous cases. The simplest way of matching is to apply nearest neighbor between each attribute in the new problem and its corresponding in the current case from the case base, then the cases having the largest weighted sum of all attributes, are retrieved. The case adaptation technique is chosen according to how close the matched case is to the current problem.

3.2 CBR in Algorithmic Composition

CBR is a very suitable approach for melody generation systems as these mainly aim to emulate the experience of human composers. CBR systems give the ability to apprehend previous composition experiences to be utilized in the generation of new compositions of the same style. To our knowledge, the recent applications of CBR in music composition are few. Instead, CBR is applied extensively in the field of expressive performance which is outside the scope of this paper as previously mentioned.

Ribeiro et al. [15] developed a system that generated melody lines given a chord sequence as an input. Each case in the case-base contains a chord along with its corresponding melody line and rhythm. Case matching is through comparing chords based on "Schöenberg's chart of the regions". The system enables a set of transformations to modify the output melody before adapting the case and storing the produced solution in the case base. Navarro-Cáceres et al. [5] developed a system whose purpose is to assign probabilities for given notes following the last note of the melody. Their system was built upon a CBR architecture accompanied by a Markov model. The case base holds past solutions and melodies and cases are retrieved according to the user's choices of the desired style, composer, etc. The adaptation step consists of training the Markov model with the retrieved cases. The adapted case is then evaluated and stored in the case base for future retrievals. Within the case adaptation phase, their system depended on user feedback to guide pitches and note duration. Generally speaking, CBR needs external guidance to successfully compose the desirable melodic pieces.

4. MARKOV CHAINS

Before introducing Markov chains, we need to define stochastic processes and chains. A **stochastic process** describes a sequence of events that depend on the time parameter t. The set of events is called the "state space", and the set of parameters is called the "parameter space". **Stochastic chain** refers to a stochastic process that consists of a sequence of a countable number of states.

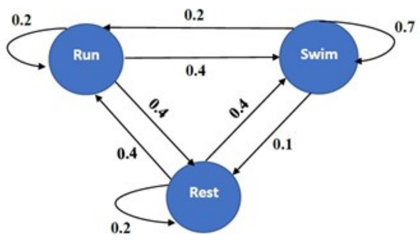
4.1 Overview and Description

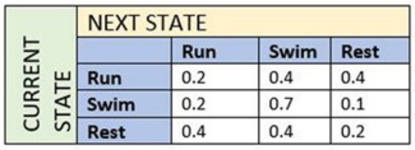
A **Markov chain** can be defined as a special case of stochastic chains in which the probability of a future event X(t+1) (random variable X at time t+1) depends on the current state X(t) according to the following equation:

$$P(X_{t_{m+1}} = j | X_{t_m} = i) = p_{ij}(t_m, t_{m+1})$$

This expression represents the transition probability of state X_{t_m}=i at a given time t_m to the state $X_{t_{m+1}}$=j. A higher order Markov chain is a Markov chain where each state depends on more than one past state such that the order number indicates how many past states affect the current state. A Markov chain can be represented as a state transition graph (also known as a Markov state space graph), where each edge shows the transition probability as shown in Fig. 5(a). Note that the sum of all probabilities on the edges leaving each state is always one. Equivalently, a Markov chain can be represented by a transition matrix as shown in Fig. 5(b).

Both figures describe the daily sportive activities of a man. He randomly spends his daily workout running or swimming and rests on some other days. By observing his sportive patterns, it was found that if he spends his workout on a certain day running, it is unlikely to see him run on the following day (with probability 0.2). Instead, it is more





CURRENT STATE	NEXT STATE		
	Run	Swim	Rest
Run	0.2	0.4	0.4
Swim	0.2	0.7	0.1
Rest	0.4	0.4	0.2

Figure 5. (a) Markov state space graph example - (b) Markov transition matrix example

likely to see him either swim or rest on the following day (both with an equal probability of 0.4). On the other hand, if he spends his workout swimming on a certain day, it is very probable to see him swim again on the following day (with probability 0.7). It is very unlikely to see him run or rest (with probabilities of 0.2 and 0.1 respectively). After a day of rest, he will probably spend the next day running or swimming (both with an equal probability of 0.4). He will hardly rest the next day (with probability 0.2).

An advanced type of Markov chains is the Hidden Markov Model (HMM) that allows us to predict a sequence of unknown "hidden" variables from a set of observable variables. That is to say that the set of observable output symbols of the Markov model are visible, but their internal states and transitions are not. Each state generates an emission probability. An HMM represents a coupled stochastic process due to the presence of transition probabilities in addition to the state-dependent emission probabilities of observable events.

Any HMM can be formally described by:

$S = S_1, S_2, ..., S_N$. Set of N states
$\pi = \pi_1, \pi_2, ..., \pi_N$. Initial state probabilities
$A = a_{ij}$. The transition probabilities from state i to state j
$O = o_1, o_2, ..., o_T$. The set of T observable outputs belonging to the set of output symbols $O_t \in v_1, ..., v_M$
$B = b_{it}$. The probability that observable output o_t is generated from state i

There are three main algorithms for training HMM:

1 **Forward algorithm:** the probability of the observed sequence is computed while all parameters are known. The opposite to the forward is the backward algorithms that computes probability in the opposite direction.

2 **Viterbi algorithm:** the most likely sequence of hidden path (Viterbi path) is calculated based on the given observable sequence.

3 **Baum-Welch algorithm:** calculates the most likely parameters of HMM based on the given observable sequence.

Fig. 6, Fig. 7, and Fig. 8 - adapted from [16] - show simple pseudocodes for the aforementioned algorithms.

4.2 Markov Chains in Algorithmic Composition

Markov chains have been extensively used in music composition since the early research attempts in the field. As previously mentioned, for HMM there is a given set of observable events and we are to calculate the most probable states leading to these events. Similarly, in computer music, we have a set of desirable notes sequence forming a nice musical piece and we try to figure out the possible paths leading to this piece to produce similar pieces in the future.

```
forward[i,j] ← 0 for all i,j
forward[0,0] ← 1.0
for each time step t do
    for each state s do
        for each state transition s to s′ do
            forward[s′,t + 1]+ = forward[s,t] ×
                                 a(s,s′) × b(s′,o_t)
        end for
    end for
end for
return Σforward[s,t_final+1] for all states s
```

Notes
1. $a(s,s')$ is the transition probability from state s to s'
2. $b(s',o_t)$ is the probability of state s' given observation o_t

Figure 6. Forward Algorithm

```
viterbi[i,j] ← 0 for all i,j
viterbi[0,0] ← 1.0
for each time step t do
    for each state s do
        for each state transition s to s′ do
            newscore ← viterbi[s,t] ×
                       a(s,s′) × b(s′,o_t)
            if newscore > viterbi[s′,t + 1] then
                viterbi[s^0,t + 1] ← newscore
                maxscore ← newscore
            end if
            save maxscore in a queue
        end for
    end for
end for
return queue
```

Notes
1. $a(s,s')$ is the transition probability from state s to s'
2. $b(s',o_t)$ is the probability of state s^0 given observation o_t

Figure 7. Viterbi Algorithm

```
Initialize HMM Parameters
Iterations = 0
repeat
    HMM' ← HMM ; Iterations ++
    for each training data sequence do
        Execute forward algorithm
        Execute backward algorithm
        Update HMM Parameters
    end for
until |HMM − HMM'| < delta OR iterations > MAX
```

Figure 8. Baum-Welch Algorithm

As for the applications of Markov chains in melody pitch generation; as aforementioned in Section 3.2, María Navarro-Cáceres et al. [5] combined a Markov model with case-based reasoning in the adaptation step in order that the Markov model is trained by the retrieved cases to predict the successive notes probabilities. Markov models also aided in counterpoint accompaniment generation. Padilla et al. [17] developed a recent emulative system that generates two-voice counterpoint based on Palestrina-style. Being trained on limited data, the Markov model was not enough for capturing all the musical features found in counterpoint. Hence, Padilla's system worked on discovering repeated patterns in each piece of the corpus (template) from different viewpoints. The discovered patterns were then fed into the Markov model for generating a two-voice counterpoint in terms of those patterns.

Wassermann and Glickman [18] have recently developed two novel approaches for harmonizing chorales in Bach style. In the first approach, the primary input into a chorale-harmonization algorithm is the bass line, as opposed to the main melody. Their second approach combines music theory and machine-learning techniques in guiding the harmonization process. They employ a hidden Markov model in learning the harmonic structure and apply a Boltzmann pseudolikelihood function optimization for determining individual voice lines. They incorporate musical constraints through a weighted linear combination of constraint indicators. They evaluated their model through a group of test subjects who could not easily distinguish between the generated chorales and the original Bach chorales. The bass-up approach outperformed the traditional melody-down approach in out-of-sample performance.

5. Generative Grammars

Language, in its simplest definition, is a set of symbol strings (sentences) whose structure abides to certain rules. A formal language is that provided with mathematical description of both its alphabet symbols and formation rules. A grammar describes how a sentence is formed and distinguishes between well-formed (grammatical) and ill-formed (nongrammatical) sentences. A **generative grammar** is a recursive rule system describing the generation of well-formed sentences (expressions) for a given language. String sequences in a generative grammar are formed through rewriting rules where symbols on the right-hand side replace symbols on the left-hand side of a rule.

5.1 Overview and Description

In the following we give an example of generative grammar which can be used to derive a simple English sentence. The grammar rules are as follows:

$$\begin{array}{lll} S & \rightarrow NP & VP \\ NP & \rightarrow Det & NP \\ NP & \rightarrow Adj & N \\ VP & \rightarrow V & Adv \end{array}$$

where S refers to "sentence" NP refers to "nominal phrase", VP refers to "verbal phrase", Det is for article (determiner), Adj is for adjective, N is for noun, and Adv is for adverb. Fig. 9 shows the derivation of a sample sentence using this generative grammar. The last row of the derivation holds the terminal symbols that cannot be further rewritten in contrast to the non-terminal symbols. Terminal symbols (in our example: *The*, *scared*, *boy, runs*, and *furiously*) are to be found in the language lexicon.

Noam Chomsky [19] classified generative grammars into four types according to their restriction level from type-0 (unrestricted) to type-3 (highly restricted). This classification is now known as "the Chomsky hierarchy".

Fig. 10 shows a visual representation of the Chomsky hierarchy. From the figure it is clear that each type is contained in the less restrictive types. In other words, a type-3 grammar is also a type-2, a type-1, and a type-0 grammar. Similarly, a type-2 grammar is also a type-1 and a type-0 grammar.

Table 1 compares the four types of grammars defined by Chomsky. As we go down in the Chomsky hierarchy (from type-0 to type-3), complexity and generative capacity decrease while restrictions increase. Type-0 grammars produce the recursively enumerable languages recognized by a Turing machine. It has the lowest level of restriction and the highest generative capacity.

Deciding whether a string belongs to such a language is an undecidable problem. The rules of type-0 grammars are in the form: $\alpha \rightarrow \beta$ such that α and β are any sets of terminal and non-terminal variables.

Type-1 grammars define context-sensitive languages. In this case, the number of symbols on the left-hand side of a rule must be less than or equal to the number of those on its right-hand side. Type-1 grammars are also highly generative, of exponential complexity, and recognized by linear bounded automata. Type-2 grammars, on the other hand, describe context-free languages that are recognized by pushdown automata which can make use of stack in determining a transition path. They are of medium generative capacity and of polynomial complexity. Finally, Type-3 grammars or regular grammars are the least generative and the most restrictive of all the four types. Its rules are restricted to only one non-terminal on the left-hand side and only one terminal on the right-hand side possibly followed by a non-terminal. Type-3 grammars are of linear complexity and is recognized by deterministic or non-deterministic finite state automata (DFA or NFA).

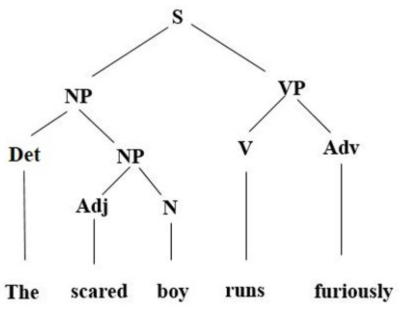


Figure 9. Derivation of a generative grammar

On the other hand, a renowned grammar system is Lindenmayer System or L-System [20] which is famous for parallel rewriting. Parallel rewriting is finding all the possible derivations for a given rule simultaneously. These are especially suited for representing self-similarity. L-Systems were used primarily in microbial, fungal, and plant growth portrayal.

5.2 Generative Grammar in Algorithmic Composition

Forming music through grammars can be achieved by replacing strings with music elements such as chords, notes pitch, and duration. The major design concern in composing music with grammars is the formulation of the set of grammar rules itself, most of the time this step is carried out manually then rules are fed into the system. However, there exists other software where grammar rules are extracted (inferred) from a previously composed music corpus; a process that is called "grammatical inference". During the music composition process, a human composer converges from the main piece theme towards its individual elements and notes. Likewise, music composition by grammar converges from level to level in conjunction with the derivation of symbols and musical elements.

One early work that extremely influenced music composition by grammars, is the book named "A Generative Theory of Tonal Music" [21]. The book describes the relation between tonal music and linguistics where the authors grammatically analyze tonal music. The book was not intended for computer music composition, nonetheless, the concepts within the book were further utilized in this research field. Melody morphing by

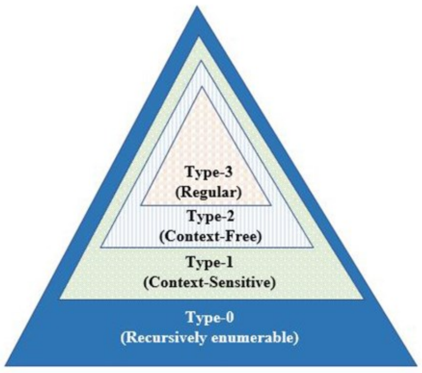


Figure 10 - Chomsky Hierarchy

Hamanaka et al. [22] is an example of the works inspired by the book.

Cruz-Alcázar et al. [23] developed a grammatical inference system for modeling a musical style which was then used in generating automatic compositions. They expanded on their work in [24] adding more comparisons between different inference techniques and music coding schemes (absolute pitch, relative pitch, melody contour, and relative to tonal center).

A basic approach for employing L-Systems in music composition applications was to interpret the graphical representations produced by L-Systems into musical notes such as in [25]. Kaliakatsos-Papakostas et.al. [12] modified finite L-Systems to generate rhythm sequences. Quick and Hudak [26] presented a novel category of generative grammars in their work called Probabilistic Temporal Graph Grammars (PTGGs), that handles the temporal aspects of music and repetitive musical phrases. Melkonian [27] expanded on Quick and Hudak's [26] probabilistic temporal graph grammars in order to include generating of melody and rhythm in addition to generating harmonic structures.

6. LINEAR PROGRAMMING

Linear programming or linear optimization is a subset of mathematical programming (optimization) where an optimal solution is found for a problem with multiple decisions about limited resources. In linear programming, a problem is formulated as a mathematical model with

TABLE 1. COMPARISON OF GENERATIVE GRAMMARS

Type	Language	Automaton	Complexity	Generative Capacity	Production Rule
Type-0	Recursively enumerable	Turing machine	Undecidable	Very high	$\alpha \rightarrow \beta$
Type-1	Context sensitive	Linear-bounded automaton	Exponential	High	$\alpha A\beta \rightarrow \alpha\gamma\beta$
Type-2	Context-free	Push-down automaton	Polynomial	Medium	$A \rightarrow \gamma$
Type-3	Regular	DFA or NFA	Linear	Low	$A \rightarrow b$ or $A \rightarrow bC$

linear relationships.

6.1 Overview and Description

In order to solve a problem using linear programming the following steps are involved:

1. Modeling the problem mathematically
2. Examining all the possible solutions for the problem
3. Finding the best (optimal) solution out of all the possible ones

A linear programming model can be described as follows:

1. Set of decision variables $X = \{x_1, x_2, \ldots, x_n\}$
2. Objective function $Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$ or
$$Z = \sum_{j=1}^{n} c_j x_j$$
3. Set of constraints:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m ,$$

where all elements in $X \geq 0$

Finding a solution for a two-variable linear programming model is simple; it can be solved graphically via drawing straight lines that correspond to each constraint in a two-dimensional space. The area covered by each straight line contains the values of its possible solutions and the area covered by all the lines (area of intersection) represents the "feasible region" or "feasible solution space". The shaded area in the graph in Fig. 11 is an example of a feasible region; where Con1, Con2, and Con3 model the linear equations of each constraint.

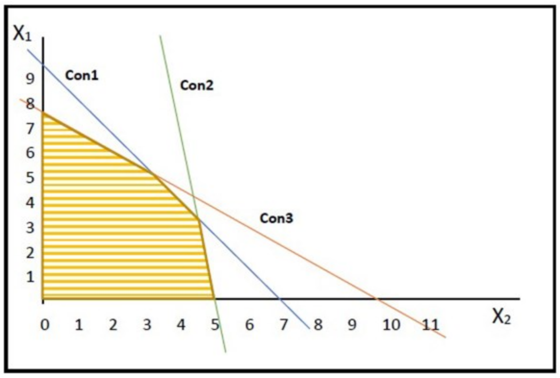


Figure 11. Graph of Linear Programming Model for a Two Decision Variables Problem

```
Locate a starting extreme point EP
while TRUE do
    for all the edges containing EP do
       find the edge E that provides the greatest
       rate of increase for the objective function
    end for
    if E = NULL then
       RETURN EP  % no more increasing edges found
    end if
    Move along E to reach the next extreme point
    if a new extreme point is found EP(new) then
       Let BFS = BFS(new)
    else
       RETURN FAIL   % The edge is infinite;
                       no solution found
    end if
end while
```

Figure 12. Simplex Algorithm (Basic version)

However, when there are more than two decision variables, the "simplex method" is adopted for finding the problem solution. Instead of exploring all the feasible solutions, the simplex method deals only with a specific set of points called the "extreme points" which represents the vertex points of the convex feasible region containing all the possible solutions. Fig. 12 is a basic version of the simplex method described geometrically, for more mathematical details please check [28] (P. 864 - 878).

The simplex algorithm starts by locating an extreme point in the feasible region. Among all the edges connected to the extreme point, the algorithm searches for the edge with the highest rate of increase in favor of the objective function. It then moves along this edge until it reaches the next extreme point. The aforementioned step might have two results; either a new extreme point is found, or the edge turns out to be infinite which means that this problem is unbound and has no solution. This algorithm repeats until no more increasing edges are found.

6.2 Linear Programming in Algorithmic Composition

Linear programming has been excessively employed in timbral synthesis; the main idea is to distribute sounds in what is called a timbral space. When the user enters specific descriptors to describe the desired timbre properties, these descriptors are given numerical values and represented as linear equations in the timbral space. The generated linear equations represent the boundaries of the region containing the desired timbre, hence, solving them using linear programming results in the optimal sound.

Timbral synthesis has earlier been based on verbal descriptions. However, Mintz [29] built his timbral synthesis on a more standard format which is MPEG-7. In his method, users can define the timbre they want using standardized descriptors. His model contains a timbral synthesis engine that turns these descriptor values into control envelopes producing synthesis equations. The coefficients of these equations are mapped as specific points in the timbral space. Seago et al. [6] also worked with timbral spaces. They proposed a timbre space search strategy, based on Weighted Centroid Localization (WCL). They expanded on their work in [30]. However, the authors pointed out that this method suffers from slow convergence.

7. Genetic Algorithms

Genetic Algorithms (GAs) are a class of evolutionary algorithms, they are also considered as stochastic search techniques. GAs tend to emulate the natural system of evolution on computers. Natural evolution is based on the fact that organisms produce excess offspring than that could survive. The large offspring compete over limited resources; thus, only those individuals who are best suited to their environment (best fit) will be able to survive. The surviving offspring reproduces and transfers its traits to its offspring creating a more fit generation each time.

7.1 Overview and Description

GAs portray natural evolution by working on a population of artificial chromosomes. Each artificial chromosome is formed of a number of genes, each is represented by a "0" or "1". Hence, mapping any problem to be solved by GAs involves, first, encoding individuals (representing potential solutions) into chromosomes. A fitness function decides how fit the individuals in each generation are. Genetic operations (crossover and mutation) on the best fit chromosomes evolve a new generation.

The steps of GA are demonstrated in more detail in Fig. 13. The very first step is to formulate chromosomes' binary genes. The population size is defined from the beginning, same for the crossover and mutation probabilities, Pc and Pm respectively. These probabilities determine the applied ratio of the crossover and mutation operations in each generation. A fitness function is then defined according to the satisfying criteria of describing a "fit" chromosome. Next, one generation of chromosomes is generated (the first generation). The fitness function is then applied to each chromosome to return its fitness ratio. According to the returned fitness ratios, the best fit chromosomes are **selected,** and genetic operators are applied to each pair of them. The result is a new generation of chromosomes that passes through the same process over and over again until the desired fitness is attained or until the maximum number of generations is reached.

```
Represent chromosomes in binary format
Set Size of population to $N$
Set mutation probability to $Pm$
Set crossover probability to $Pc$
Define the fitness function: $Fitness\_Fn()$
Produce the first generation of chromosomes
$X_1, X_2, \ldots, X_N$
repeat
  for each $i$ do
    Call $Fitness\ Fn(i)$ to GET the fitness ratio $F_X$
  end for
  repeat
    Choose a pair of chromosomes $X_i$ and $X_j$
    according to their fitness ratios $F_{X1}$ and $F_{X2}$
    Apply genetic operators on $X_i$ and $X_j$ according
    to $Pc$ and $Pm$
    Add new $Xi'$ and $Xj'$ to the new generation
  until number of chromosomes of the new
        generation = $N$
  Replace old generation with the new one
until a chromosome with satisfying fitness is
      found or the maximum number of generations
      is reached
```

Figure 13. The Genetic Algorithm

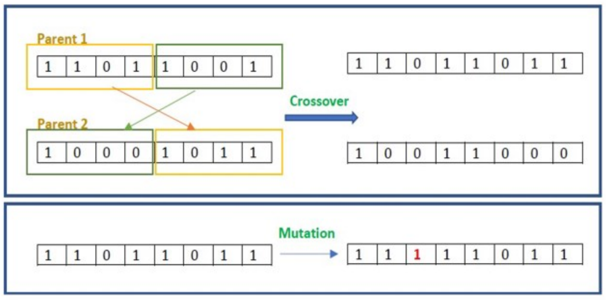


Figure 14. Examples on mutation and crossover

Other than **selection**, a GA mainly relies on the **crossover** and **mutation** operators. Mutation is the flipping of one randomly selected gene in the chromosome, while crossover involves splitting a pair of chromosomes at a randomly selected crossover point and exchange the resulting chromosome sections. Fig. 14 gives an example of each operation.

7.2 GAs in Algorithmic Composition

GAs have been broadly used in the field of algorithmic composition. GAs are suited for music composition applications due to the following [31]:

1. They help in generating many segments (generations) to form larger musical pieces.
2. Musical segments are continuously generated and examined, which generally complies with the composition process and concepts.
3. The generated music is always evaluated through fitness metrics which improves the quality of the generated music.

TABLE 2. COMPARISON BETWEEN ABSOLUTE AND RELATIVE PITCH

Absolute Pitch	Relative Pitch
When a chromosome is modified the following sequence stays intact.	When a chromosome is changed all the following notes are affected.
Preferred if transpositions apply to one voice of polyphonic movement.	Allows for transposition for whole segment.
Mutation produces larger modification.	Mutation causes less modification.

4. The generated results are also affected by how music is encoded.

Music is encoded either in the form of absolute or relative pitch; absolute pitch encoding specifies the concrete note pitch value in binary, while relative pith encodes the distance between two consecutive notes. Table 2 compares between absolute and relative pitch.

Genetic operators are twisted to be applied to music; for example, mutation and crossover are changed into mirror and crab. However, these twists can be applied to western music only where the distance between notes in the scale are constant. These transformations, on the other hand, are supposed to work on musical segments primarily musically related. This is, nonetheless, problematic in GAs due to the continuous generation of new fragments through rearranging chromosomes, and thus, losing their musical structure in the process. Large musical fragments are problematic because the musical context is completely changed with every genetic operator, resulting in a possibly undesirable abrupt modulation. To overcome this issue, we can use rule-based techniques to control and guide the operators according to the musical domain modeled in the rules. Moreover, since we are dealing with context-dependent information, Markov chains and generative grammars can be used as promising potential aids to GAs for better handling such information.

Fitness evaluation also represents a challenge for GAs in music. At first, human user evaluation has been adopted for GAs; this however, caused the algorithm to be delayed waiting for the user input. Consequently, techniques such as rule-based and neural networks were utilized in fitness functions as a weak alternative to human evaluation. Nonetheless, research involving more than one fitness function proved to produce better results.

There are plenty of applications of GAs in the different music composition tasks. As for melody generation, the work of Pedro J. Ponce de León et al. [32] enhanced the selection process through developing a multi-objective fitness function. Moreover, they proposed a new data structure; "Melodic Trees" for chromosomes representation. For the task of timbre synthesis, Carpentier et al. [33] developed an evolutionary algorithm that, not only discovers optimal orchestration solutions, but also indulges in the examination of non-spontaneous mixtures of sounds.

GAs were also employed in chord generation, such as the polyphonic accompaniment generation (formed of main, bass, and chord) system of Liu et al. [34]. In their system they implemented a fitness function that is built upon evaluation rules inspired by music theory. Later, they enhanced their system in [35] by mining and extracting chord patterns from specific composer's music to be introduced as genes in the GA. In addition to chords, Liu's work included bassline and rhythm generation, through the merging of GAs and data mining.

Recently, R. De Prisco et al. [36] developed an algorithm for automatic music composition using an evolutionary algorithm. They work on chorales of four voices. Their algorithm takes one of the voices as input and produce the rest of the four voices as output. they aim to finding both; suitable chords, in addition to the melodic lines. They not only proposed a novel representation for the chromosomes, but also, they enhanced the quality of the new generations through customizing operators to make use of music theory and musical statistical analysis on a Bach's chorales corpus. As for the fitness function, the authors used a multi-objective fitness function dealing with both the harmonic and the melodic aspects.

Abu Doush and Sawalha [37] combined GAs and neural networks for composing music. They implemented a GA to generate random notes and used neural networks as the fitness function for that algorithm. The authors compared between four GAs with different combinations of parameters such as; tournament and roulette-wheel for the selection phase and one-point and two-point crossovers. Their experiments showed that using tournament selection and two-point crossover generate music compositions of higher quality.

8. ARTIFICIAL IMMUNE SYSTEMS

Researchers developed Artificial Immune Systems (AIS) aspiring to find solutions for their research problems based on concepts inspired by the biological immune system. The biological immune system protects our bodies from pathogen attacks (harmful microorganisms that stimulate immune response). Unlike the centralization of the neural system that is controlled by the brain, the biological immune system is decentralized and distributed throughout the body.

The immune system has the advantage of being robust, self-organized, and adaptive. It has pattern-recognition and anomalies detection capabilities, and it keeps track of previous attacks for better future responses. When a body is attacked by pathogens, the immune system detects them and instantiates a primary defense response. If the attack repeats later, the immune system remembers that past experience and consequently launches a secondary response quicker than the primary one. The immune system has the ability to differentiate between self (those belong to the body) and non-self cells (invaders). The term “B cell” refers to a part of the immune system that produces antibodies targeting the pathogens to be diminished from the body. Antibodies are produced when B cells clone and mutate after a process of recognition and stimulation.

8.1 Overview and Description

Prior to solving problems by AIS, the “antigens” and “antibodies” need to be defined in terms of the problem domain, and then encoded in binary format. An important design choice is the “affinity metric” (also called the “matching function”) which is pretty similar to the “fitness function” in GAs. Selection and mutation operations also need to be determined (mutation is also very similar to that in GAs, based on flipping bits). When all the above is well defined the algorithm can then be executed.

The most famous selection algorithms in AIS are “negative selection” and “clonal selection” algorithms. The negative selection algorithm proposed by Forrest et al. [38], is shown in Fig. 15 reproduced from [39]. Its idea is to differentiate between self and non-self cells and to react differently to them. The input to this algorithm is a set of self strings that are stored and marked as friendly normal data. The first phase of the algorithm is the generation of string detectors. Detectors are generated as random strings and matched with the list of self strings keeping only those that do not match. The second phase is to monitor the system for detection through continuously matching the input strings with the detector strings and streaming out those that match.

```
input: S = set of self strings characterizing
           friendly, normal data.
output: A = Stream of non-self strings detected.
Create empty set of detector strings D.
Generate random strings C.
for all random strings c ∈ C do
    for all self strings s ∈ S do
        if c matches s then
            Discard c
        else
            Place c in D
        end if
    end for
end for
while there exist protected strings p to check do
    Retrieve protected string p
    for all detector strings d ∈ D do
        if p matches d then
            Place p in A and output.
        end if
    end for
end while
```

Figure 15. Negative Selection Algorithm (Reproduced from [39])

```
input: S = a set of antigens, representing data
           elements to be recognized.
output: M = set of memory B-cells capable of
            classifying unseen data elements.
Generate set of random specificity B-cells B.
for all antigens ag ∈ S do
   Calculate affinity of all B-cells b ∈ B with ag.
   Select highest affinity B-cells, perform
   affinity proportional cloning, place clones
   in C.
   for all B-cell clones c ∈ C do
      Mutate c at rate inversely proportional to
      affinity.
      Determine affinity of c with ag.
   end for
   Copy all c ∈ C into B.
   Copy the highest affinity clones c ∈C into
   memory set M.
   Replace lowest affinity B-cells b ∈ B with
   randomly generated alternatives.
end for
```

Figure 16. Clonal Selection Algorithm (Reproduced from [39])

Clonal selection is based on the idea of cloning the B cells that are proved to be of the highest match with the antigens (highest affinity). The cloned B cells act as an army for defending the body against antigens; because they have the correct antibodies inside them. The clonal selection algorithm is shown in Fig. 16 reproduced from [39]. The first step in the algorithm is to generate a random group of B cells. The affinity is then calculated between each antigen and all the B cells. The B cells of the highest affinity are cloned proportionally to the affinity measure. The cloned cells are mutated with a probability that is inversely proportional to the affinity measure. The affinity of the mutated clones, with respect to the antigen, is then calculated. The B cell clones of higher affinity replace the B cells of lower affinity in the old generation. Furthermore, a copy of the clones with the highest affinity is kept in memory.

8.2 AIS in Algorithmic Composition

As previously mentioned in Section 2.2, AIS has been used for chord generation by Navarro-Cáceres et al. [10] to provide a recommendation for the following chord in a sequence. AIS is unique in that it provides more than one suggested solution (chord) because multiple optima are found in parallel. In the case of chord generation, the suggested multiple optima need to be filtered to offer a threshold for generating the good chords only. Navarro-Cáceres et al. expanded on their work in [40] and enhanced their AIS so as to optimize an objective function that encodes musical properties of the chords as distances in the so called Tonal Interval Space (TIS). Chord selection is viewed as a search problem in a the TIS geometric space in which all chords are represented under certain constraints. Navarro-Cáceres' work is centered about generating the next candidate chord given the previous two chords as an input; thus, their system captures short-term dependencies only and need enhancements to generate a chord that depends on the whole music context rather than the previous few chords only.

For the task of computer aided orchestration, Caetano et al. [41] developed a multi-modal AIS that comes up with new combinations of musical instrument sounds as close as possible to the encoded sound in a penalty function. In contrary to chord generation, the nature of the orchestration problem is that it may hold more than one possibility, hence the aim of Marcelo's system was to maximize the diversity in the solution set of the multi-modal optimization problem. This approach led to the existence of various orchestrations for the same reference sound and actually embraced the multiple optima phenomena in AIS.

Navarro-Cáceres et al. [42] have recently introduced an interesting application for chords generation based on a neurological phenomenon called Synesthesia. In this phenomenon, the stimulation of one sensory, results in automatic, involuntary experiences in a second sensory. Inspired by this phenomenon, the authors extract sound from colors for chord progressions generation utilizing an AIS. They extract the main colors from a given image and feed them as parameters to the AIS. They developed an optimization function to come up with best candidate chord for the progression, according to its consonance and relationship with the key and the previous chords in the progression.

9. ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANNs) aim for simulating the biological neural system controlled by the brain. A biological neural system is composed of small interconnected units called neural cells or neurons. A biological neuron is a special type of cells that processes information. A neuron has four parts: Dendrites, Soma, Axon, and Synapses. Dendrites help the neuron receive information from other neurons. Soma is the cell body that is responsible for information processing. A neuron sends information through Axon. Synapses help a neuron to connect with other neurons in the network.

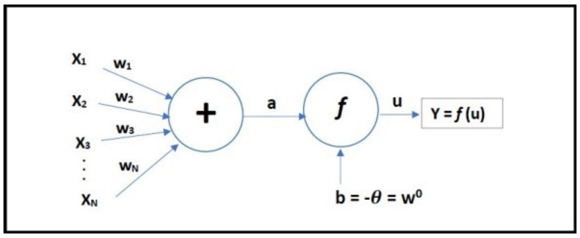


Figure 17. Artificial neuron (Adapted from [43])

9.1 Overview and Description

An Artificial Neuron (AN) models the biological neuron; it also has inputs, a node (body), weights (interconnections) and an output. Fig. 17 adapted from [43] shows a diagram of an artificial neural.

The variables in the figure are:

1. Input variables $\{x_1, x_2, \ldots, x_N\}$: features or attributes coming from other neurons connected to the current neuron.
2. Weights $\{w_1, w_2, \ldots, w_N\}$: factors multiplied by the inputs to control how much each input is affecting the result.
3. Summation result a: It is the weighted sum of all inputs.
4. Bias b: A constant affecting the activation function $f(u)$ where $u = a + b = \sum_{i=1}^{N} w_i x_i + b$
5. Error threshold $\theta = -b$ which is applied to the neuron output to decide whether the neuron will fire or not.
6. Neuron output $Y = f(u)$
7. Activation Function $f(u)$: It is the function applied to u to determine the final output Y. f can be a linear, step, or sigmoid function, among others.

One of the building blocks of neural networks design is the "network topology" which is how the neurons are organized in the network. There are two main types of topology:

1. Feedforward networks: In a feedforward network signals move in one direction from input to output. A feedforward network can either be:
   (a) A single layer feedforward network (single layer perceptron): A network having only one layer of nodes connected to the input layer. This type of network is typically used to solve linearly separable classification problems.

(b) A multilayer feedforward network: A network having one or more layers between the input and the output layer. Multilayer networks are fully connected which means that each neuron in one layer is connected to all the neurons in the next layer. This type of networks is typically used to solve non-linearly separable classification problems.

2. Feedback Networks: In a feedback network signals can flow in both directions such as in the case of Recurrent Neural Networks (RNNs) which have closed loops in their architecture. They have connections from units in a layer leading to the same layer or to previous ones through what is called "context neurons". This kind of networks is dynamic and keeps changing until it reaches an equilibrium state. RNNs provide a means of storing information from previous epochs (training cycles) and using them in future ones, i.e. they support long-term dependencies. For even more support of long-term dependencies, Long Short-Term Memory (LSTM) networks add memory block units to recurrent networks.

ANNs are trained (they learn) to perform the desired tasks. There are three ways an ANN can learn with:

1. Supervised learning: A dependent type of learning in which the output of the network is compared with the desired output. According to the difference between the actual output and the desired output, the weights of the network are updated until the neuron's output match the target output. Supervised learning examples include: the *delta rule* for training single layer perceptrons and the *backpropagation algorithm* to train multilayer networks.
2. Unsupervised learning: An independent type of learning typically used for clustering input data of similar types or for dimensionality reduction.
3. Reinforcement learning: A semi-dependent type of learning based on reward and punishment. Without labeled data items, the network receives a feedback from the environment as an evaluation to its performance.

Since backpropagation is one of the most powerful learning algorithms that are widely used in different types of ANNs, we provide a simplified version of this algorithm in Fig. 18 (adapted from [44]), assuming the network has only one hidden layer. The input training samples (or vectors) are fed into the network and transferred to the hidden layer as weighted sums. The hidden layer units apply an activation function to these sums, then they transfer the results to the output layer units as another set of weighted sums. The output of the

```
Initialize weights (small random values)
repeat
 for each training sample do
   -FEEDFORWARD-
    Each input unit receives an input signal $X_i$
    $\in \{x_1, x_2, ..., x_n\}$ and broadcasts it to all
    the hidden units in the next layer
    Each hidden unit $Z_j \in \{Z_1, Z_2, ..., Z_p\}$ sums the
    weighted input signals:
              $z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$
    then applies the activation function to the
    output $z_i = f(z\_in_j)$ then broadcast its
    signal to all the units in the output layer
    Each output unit $Y_k \in \{y_1, y_2, ..., y_m\}$ sums the
    weighted input signals:
              $y\_in_k = w_{0k} + \sum_{j=1}^{p} z_i w_{jk}$
    then applies the activation function to
     the
    output    $y_k = f(y\_in_k)$

   -BACKPROPAGATION-
    For each output unit $Y_k \in \{y_1, y_2, ..., y_m\}$, the
    error $\sigma_k$ is computed between the output
    signal and the target sample corresponding
    to the
    training input sample
              $\sigma_k = (t_k - y_k) f'(y\_in_k)$
    where $f^0$ is the derivative of the
    activation function. The weight correction
    term is then
    calculated
              $\Delta w_{jk} = \alpha \sigma_k z_j$
    and the error $\sigma_k$ is transferred to all the
    hidden units in the previous layer
    For each hidden unit $Z_j \in z_1, z_2, ..., z_p$ the
    weighted sum of the transferred error is
    calculated
              $\sigma\_in_j = \sum_{k=1}^{m} \sigma_k w_{jk}$
    The error information is then calculated:
              $\sigma_j = \sigma\_in_j f'(z\_in_j)$
    and the weight correction term is
    calculated:
              $\Delta v_{ij} = \alpha \sigma_j x_i$   where $\alpha$ is the
                                 Learning rate
    Weights of the hidden layer are updated
              $v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$
    Weights of the output layer are updated
              $w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$

  end for
until Error threshold is reached
```

Figure 18. Backpropagation Algorithm (Adapted from [44])

network is the result of applying the activation function in the output layer. The difference between the output and the corresponding target sample data is calculated which serves as the error factor. The weighted error sum is propagated backward from the output layer to the hidden layer and from the hidden layer to the input layer. Meanwhile, the weight correction factor is calculated for each unit (in terms of the error factor) which is then used to update the network weights. The whole process is repeated until the error becomes less than a given threshold.

9.2 ANNs in Algorithmic Composition

There are many examples of using ANNs in musical tasks. For chorale music, Hadjeres et al. [45] aimed for imitating Bach's chorales in their system using a dependency network and pseudo-Gibbs for the music sampling. On the other hand, Yamada et al. [9] set up a comparison between adopting Bayesian Networks (BNs) and recurrent neural networks in chorale music generation to show the strengths and weaknesses of each.

Recent research about chord generation using ANNs include that of Brunner et al. [46] and of Nadeem et al. [47]. The former's system produces polyphonic music based on combining two LSTMs. The first LSTM network is responsible for chord progression prediction based on a chord embedding. The second LSTM then uses the predicted chord progression for generating polyphonic music. The latter's system produces musical notes accompanied by their chords concurrently. They use a fixed time-step with a view to improve the quality of the music generated. To produce new music, chords and notes networks are trained in parallel and afterwards their outputs are combined through a dense layer followed by a final LSTM layer. This technique ensures that both inputs, notes and chords, are being dealt with along all the steps of generation, and thus, become closely related. As mentioned earlier in Section 7.2, Abu Doush and Sawalha [37] employed neural networks to compute the fitness function for a GA. They were trained to learn the regularity and patterns of a set of melodies.

10. DEEP NEURAL NETWORKS

Deep Neural Networks (DNNs) are distinguished from single hidden layer ANNs by their depth. The network's depth means the number of layers that an input has to pass through until it reaches the output layer.

10.1 Overview and Description

A famous type of deep neural networks is the Convolutional Neural Network (CNN). CNNs apply the convolution operation instead of general matrix multiplication (weighted sum) in at least one of its layers. Fig. 19, adapted from [48], shows the general architecture of a CNN. The main building blocks of a CNN are the input, output, convolutional, pooling, and fully connected layers. The idea behind CNNs is to decompose a given problem into smaller ones and work on solving each. the

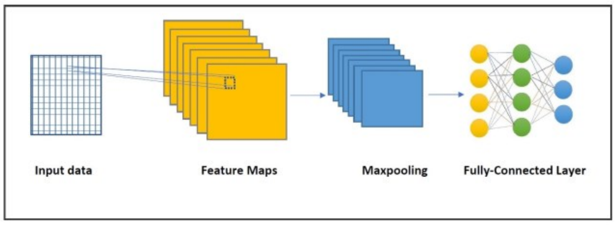


Figure 19. Convolutional Neural Network Architecture (Adapted from [48])

convolution layer in a CNN applies several filters on the input data such that each filter extracts a specific feature from it. The maxpooling layer performs dimension reduction by keeping only data items of highest values within the pooling size. The output from maxpooling is then flattened to be introduced to a fully connected neural network which in turn produces the final output.

The convolutional layer performs the computation defined by the following mathematical equation (the dot product between the input data and a given filter):

$$(I * F)(x, y) = \sum_{v=y-h}^{y+h} \sum_{u=x-w}^{x+w} I(u, v)F(x - u, y - v)$$

where I is the input, F is the filter of width 2w + 1 and height 2h + 1. F is defined over [−w,w] × [−h,h]. A simple pseudocode for the convolution process is described in Fig. 20 adapted from [49]. This can be interpreted visually as a window scanning the input data from left to right and from up to down moving one cell at a time. At each step, a dot product is computed between the window's values and the current values of the input data that corresponds in position to the window. These steps repeat until the algorithm finishes scanning the input data. Training the CNN can be done through applying the backpropagation algorithm on the features map produced from the convolution process in the fully-connected layer.

10.2 DNNs in Algorithmic Composition

```
Given the filter array $F$ of size $F_w \times F_h$ and the
input data array $I$ of size $I_w \times I_h$
for $y$ from 0 to $I_h$ do
    for $x$ from 0 to $I_w$
      do $sum = 0$
      for $j$ from 0 to $F_h$ do
         for $i$ from 0 to $F_w$ do
            $sum$ += $I[y + j][x + i] * F[j][i]$
         end for
      end for
      $C[y][x] = sum/(F_w * F_h)$
    end for
end for
return $C$
```

Figure 20. Convolution Pseudocode (Adapted from [49])

CNNs were employed by Huang et al. [8] for composing music in a nonlinear fashion closer to human composers' style of composition, rather than the chronological style. Their CNN was trained to generate partial musical scores. Moreover, they introduced the use of blocked Gibbs sampling as an equivalent to music rewriting.

Deep RNNs implemented by Colombo et al. [50] for the purpose of melody generation, had the ability to capture the long range temporal structure of music. Temporal dependencies were also modeled in Benjamin Smith's [51] work through the help of a Convolutional Restricted Boltzmann Machine (CRBM). They could achieve full reconstructions of musical pieces just from a starting seed note. BachBot is a recent research project by Liang et al. [52] that imitates Bach's chorales style through a deep LSTM generative model. Inspired by CNNs, Johnson [53] developed two deep networks architectures for polyphonic music composition; Tied Parallel LSTM-NADE (TP-LSTM-NADE) as well as Bi-Axial LSTM (BALSTM). Johnson designed his models so as to be transposition invariant; i.e., making the training independent from the musical key. In brief, he divides the process of music generation into a set of tied parallel RNN networks with tied weights between them such that each network is responsible for a single note prediction. Since all the networks use the same procedure to calculate their outputs, a shift in the input will cause an equal amount of shift in the output; thus, the process is transposition invariant. The name Bi-axial comes from the combination of LSTMs with recurrent connections along two different axes: First along the time axis and then along the note axis. This architecture eliminated the need for windowing the input notes before introducing them to the networks, and thus, the captured dependencies between notes became not limited to the bunch of notes inside the window. Mao et al. [54] enhanced the biaxial model developed by Johnson [53] so as to include musical styles and music dynamics. They based their architecture on the bi-axial model nonetheless they added conditioning for style enforcement at every layer. However, both systems [53] and [54] lacked long term structure between notes and lacked central themes in the generated music.

One of the powerful recent applications of DNNs in music is WaveNet [55] which is a deep neural network for generating raw audio waveforms. Although principally developed for speech synthesis purposes, WaveNet was able to produce reasonable musical waveforms when tested on musical piano pieces. The problem with WaveNets is that they are rather slow; producing 0.02 seconds of audio in one second of time, and thus, not suitable for real time applications. Vasquez and Luis [56] developed an audio generative model based on WaveNet in the frequency domain. They work on spectrograms and their model is based on autoregression. Accordingly, it estimates a distribution element-by-element over the time and frequency. Their results were very promising on a wide range of applications not only music generation and the music generated by their system was of a rather high quality.

Cífka et al. [57] have recently presented a one-shot style transfer approach. The authors clarify the meaning of one-shot learning to be the process of learning the concept of a class from a single example. They target accompaniment style transfer in their work. They introduce an encoder-decoder neural network that consists of two encoders one for content and one for style in addition to a decoder for the output generation.

One year after developing WaveNet, the authors; Oord et al. [58], developed a much more advanced version of WaveNet. The new version uses *probability density distillation* which is a new method for training a parallel feed-forward network. It takes the new WaveNet only one second to produce twenty seconds of audio which is a thousand times faster than the original version. The new WaveNet has a reference (teacher) network that works correctly but slowly and is another (student) network that tries to mimic the teacher network but more efficiently. This architecture has somehow similar vibes as the generative adversarial networks described in the next section.

11. Generative Adversarial Networks

Generative Adversarial Networks (GANs) were invented by Ian Goodfellow [59] in 2014. They are designed to put two ANNs in a competition with each other. GANs aims for imitating any distribution of data.

11.1 Overview and Description

The two main components of a GAN are a generator network G and a discriminator network D. Fig. 21, adapted from [60], shows the basic architecture of a GAN. The generator takes as an input randomly sampled data taken from a previously generated distribution and generates a fake sample. The input to D is either a real sample (taken from the prior distribution) or a fake sample (generated from G). The networks are trained such that D learns to distinguish between real and fake samples while G learns to deceive D. The objective function of a GAN

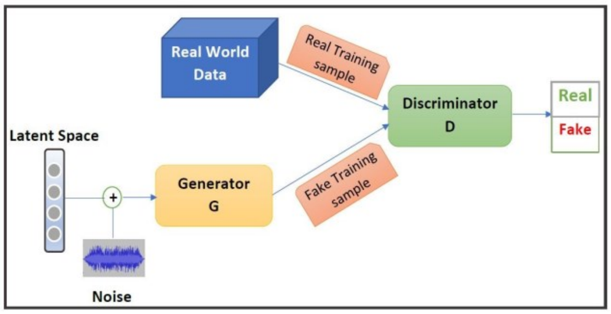


Figure 21. Generative Adversarial Network Architecture (Adapted from [60])

is:

$$min_G max_D V(D,G) = \mathbb{E}_{x\sim Pdata(x)}[logD(x)] + \mathbb{E}_{z\sim pz(z)}[log(1 - D(G(z)))]$$

This formulation, in brief, suggests that G and D are playing a minmax game over an objective; that is to support D to assign the label 'one' to the real data and 'zero' to the fake data. This is the objective that G aims to minimize, while D tends to maximize. G and D in the first GAN version were multilayer perceptron neural networks, nonetheless, these networks were later on replaced by convolutional deep neural networks. Two particularly interesting variations of GANs are the pix2pix [61] and cycleGAN [62]. These GANs generate images based on condition images rather than on random noise. Pix2pix and cycleGANs are used for supervised and unsupervised image style transfer respectively.

11.2 GANs in Algorithmic Composition

The field of computer music generation, among other fields, has excessively benefited from the invention of GANs. Yang et al. [63] were able to generate melody employing GANs; however, they worked on MIDI sequences in the symbolic musical space rather than working on waveforms. Dong et al. [64] succeeded in developing a multi-track music generation system utilizing GANs. They worked with piano-rolls music representation format which encodes music as binary-valued time-pitch matrices. While convolutional GANs normally produce real values, they subsequentially enhanced their GAN, in [65], to overcome the problem of the resulting real-valued piano-rolls by incorporating binary neurons. They introduced a refiner network that is applied to the generator's output. The output layer of the refiner network is formed of binary neurons. This approach achieved better results than the naïve binarization post-processing of the network's output. Oza et al. [66] expanded on Dong's work [65] by applying progressive training such that new layers are successively added to the pretrained, already converged, network. They start their training by small time-step values and pitch range, then they progressively enlarge these by adding more layers to the network. The total number of layers in their network after all the progressive training phases is 12 layers in the shared generator and discriminator network and 8 layers in the refiner network. Progressive training in [66] lead to musically better results than [65] having lesser fragmentation of the notes and improving periodicity and melodic perception.

Brunner et al. [67] developed a system for music style transfer utilizing cycleGANs. They perform music transfer between Classic, Jazz, and Pop styles. They trained their network with binary piano-rolls; however, they compressed all music tracks into one instrument removing drums track. Their network contained multiple discriminators in order to improve the fidelity of the input music to the input music structure. Their system generates music that is harmonic in general; nonetheless, this method is limited to musical files having few tracks and still does not capture notes velocities, correct duration, and instruments' variations.

Most recently, Jin et al. [68] developed a style-specific music generation system inspired from GAN and based on reinforcement learning. Their system consists of a generator, a control network, and a probability network (discriminator). The novelty of their system lies in the addition of a control network mediating between the generator and the discriminator, in addition to taking advantage of LSTM in the generator. There are three main tasks for the control network: 1. Introducing music rules to restrict the generated music to a specified style, 2. Calculating a loss function according to music theory constraints to generate high-quality music, and 3. Assigning a scoring to the music generated by the generator network by means of a reward function algorithm. Their system generated music that was evaluated to be of high quality but needed more enhancement to match the specified style.

All the GAN applications mentioned so far in this section train on piano-rolls (binary-valued time-pitch matrices). Liu et al. [69] developed a GAN music generator that works on mel-spectograms, which they enhanced in [70]. In their system, the generator takes a variable-length sequence of noise vectors as input and generates variable-length mel-spectrograms.

12. Comparison and Discussion

Table 3 compares all top ten AI algorithms we covered. The table lists the recent applications of the algorithms for each computer music composition task. The table also highlights the strengths and weaknesses of each algorithm. In this section, we delve into a deeper discussion of their challenges and future directions.

A. Rule-Based Reasoning

Our analysis shows that rule-based systems are suitable for representing music theory and rules. Adopting rule-based systems in music composition is straightforward due to the nature of music theory that is composed of rules.

Challenges: In real life, it is not enough to have knowledge about music theory for composing music, yet musical skills and experiences are essential to accomplish the task. Similarly, the rule-based approach cannot be used alone for producing the desired musical compositions. Instead, they would achieve better results if used to assist other algorithms.

TABLE 3. COMPARISON BETWEEN THE TOP 10 AI ALGORITHMS IN COMPUTER MUSIC

Algorithm	Composition Task	Reference(s)	Strengths	Weaknesses
(1) Rule-Based	Counterpoint	[14] (2010)	Simple to implement because musical rules are already available, can serve as a guidance for other techniques.	Insufficient alone for getting acceptable results.
	Chord	[10] (2015)		
(2) Case-Based Reasoning	Melody Pitch	[15] (2002), [5] (2017)	Excellent for learning by example.	Insufficient for obtaining pleasing results. Requires external guidance or association with other techniques. Lacks creativity.
(3) Markov Chains	Melody Pitch	[5] (2017)	Predicts new successive notes based on previous knowledge.	Predicts one note at a time. Lacks long term musical dependencies. Not suitable for polyphonic music.
	Counterpoint	[17] (2018), [18] (2020)		
(4) Generative Grammars	Melody Pitch	[22] (2008), [23] (2003), [24] (2008), [25] (2009), [27] (2020)	Fast compositions. Compositions conform to musical rules.	Sometimes grammatical rules derivation is done manually. Various grammatical inference techniques affect the produced music quality.
	Accompaniment	[26] (2013)		
	Rhythm	[12] (2012), [27] (2020)		
(5) Linear Programming	Timbre	[29] (2007), [6] (2010), [30] (2013)	More successful than earlier verbal descriptors. Offers standardization through timbral spaces.	Does not perform well in high dimensional timbral spaces. Slow convergence.
(6) Genetic Algorithm	Melody Pitch	[32] (2016) [37] (2020)	Simulates the natural composition process. Aids in concurrently executing multiple musical composition tasks.	Single fitness function is insufficient for getting satisfying musical results, while multi-objective fitness functions are sometimes contradictory and need optimization techniques to be combined.
	Timbre	[33] (2010)		
	Chord	[34] (2012), [35] (2015)		
	Bass	[34] (2012), [11] (2015)		
	Rhythm	[35] (2015)		
	Chorales	[36] (2020)		
(7) Artificial Immune System	Timbre Orchestration	[41] (2019)	Suggests more than one solution.	Might need a means of filtration if fewer solutions are needed, captures short-term dependencies.
	Chord	[10] (2015), [40] (2019), [42] (2020)		
(8) Artificial Neural Networks	Chorale	[45] (2017), [9] (2018)	Can produce polyphonic music and four-part harmonization. RNNs and CRBM can capture long term dependencies in music.	Contradicting results when adopting different networks for the same musical task. Dependency networks (among other types) are time consuming.
	Chord and Pitch (Polyphonic)	[46] (2017), [47] (2019) [35] (2015)		
	Melody	[37] (2020)		
(9) Deep Neural Networks	Pitch	[50] (2016), [51] (2017)	Shown to produce appealing music. Capable of training on large musical corpus. Performs multiple compositional tasks concurrently. Excels in musical features extraction from data, enhancement of generated compositions through revisiting (in case of CNNs).	As networks get deeper, the results get better at the cost of computational power. Some DNNs lack long-term relative dependencies between notes.
	Chorale	[52] (2017)		
	Counterpoint	[8] (2019)		
	Chord and pitch (polyphonic)	[53] (2017), [54] (2018), [54, 56] (2016, 2017), [56] (2019)		
	Accompaniment	[57] (2020)		
(10) Generative Adversarial Networks	Pitch	[63] (2017), [63, 64] (2018), [66] (2019)	The generated music is voted for as high quality.	Consumes high energy rates. Needs special handling for music data. Still not perfect in capturing notes velocities and duration.
	Polyphonic	[67] (2018), [68] (2020)		

Future Directions: Rule-based systems can serve as pre- or post-processing tools within other music generation systems either for data selection or enhancement. We already gave an example for rule-based utilization in the evaluation function of AIS [10]. It would be interesting to study rule-based fitness functions in GAs as well. Basically, a study is needed to define the musical rules that can be used to compute a chromosome's fitness. Additionally, rule-based algorithms can be integrated with neural networks, whether shallow or deep, to guide the overall convergence of the network according to musical guidelines. The same idea can also be applied to GANs where extra penalties can be added to the loss functions to generate images that are musically correct.

B. Case-Based Reasoning

CBR systems are very good for resembling human's way of learning from previous musical experiences.

Challenges: CBR alone is rather limited and not sufficient for generating pleasing compositions. When applying CBR in music composition, the resulting musical pieces lack creativity as they are copies of previously composed musical pieces stored in the case-base.

Future Directions: CBR can assist other algorithms in music composition. We already gave an example combining CBR and Markov chains [5]. Further study is needed to combine CBR with other algorithms such as GAs and AIS in the fitness evaluation. It would also be of interest to incorporate CBR with GANs. For example, GANs would be responsible for generating melody lines and CBR would be responsible for fetching suitable accompaniments for the generated melodies. This can produce more creative musical pieces. Additionally, CBR needs to be further exploited in the automation of music composition tasks other than melody pitch generation, such as accompaniment music or rhythm generation.

C. Markov Chains

Markov chains were proven to be suitable for melody generation as they excel in predicting new notes from previous knowledge.

Challenges: Markov models can only predict one note at a time. As such, they lack long term dependencies; a key feature of musical pieces. On the other hand, Markov models are more suitable for composing monophonic music rather than polyphonic ones. This is because Markov models grow significantly in complexity as the number of musical voices increase.

Future Directions: Markov models might be a suitable candidate to replace or assist the generator network of a GAN since it has the ability to predict the next musical notes.

D. Generative Grammars

They provide for fast composition of musical pieces that conform to music rules.

Challenges: Generative grammars need wise decisions for each design issue such as how to formulate grammar rules, which grammatical rules to apply, etc. Most of the time the composition grammatical rules are formulated manually and fed into the system. This manual formulation of rules constitutes a severe overhead. On the other hand, grammatical inference techniques adopted produce rules of diverse quality potentially harming the quality of the produced music.

Future Directions: Further research is needed in the automation of generative grammars derivation for music. Since current grammars are restricted to melodic and harmonic structures, it is encouraged to devise grammars for style and genre specific music generation.

E. Linear Programming

Linear programming aids a lot in timbre synthesis over the early verbal descriptors due to the idea of timbral spaces.

Challenges: linear optimization works better for small dimensional spaces. In [6] it is stated that the WCL method performs significantly better in relatively simple three-dimensional spaces (in this case the formant space and the SCG-EHA spaces) than in spaces where the dimensionality is greater (the MDS space).

Future Directions: research is needed for speeding up the convergence process in the WCL method. Another research direction would be to consider applying linear programing for different music composition tasks other than timbre synthesis devising spaces having the same style as timbral spaces but for notes or chords. Linear programming might also be a good candidate for being embedded in fitness evaluations for GA and AIS.

F. Genetic Algorithms

GAs already provide enhancements in various music composition tasks such as melody generation, timbre synthesis, chord, bass, and rhythm generation. GAs are pretty similar to the natural process of music composition as quoted in [71]: "*For composers, it provides an innovative and natural means of generating musical ideas from a specifiable set of primitive components and processes. For musicologists, these techniques are used to model the cultural transmission and change of a population's body of musical ideas over time*".

Challenges: GAs have challenges such as the challenge of designing the most convenient fitness function. Due to the multiple featured nature of music, single fitness functions are often not sufficient for a proper evaluation. This brought up the need for multi-objective fitness functions; however, this approach excels only if suitable optimization is achieved among all the (sometimes contradictory) fitness criteria.

Future Directions: In GAs, there is a need for research in fitness function enhancement to cope with the nature of musical notes and generate music from different genres. Additionally, the door is open for more research in applying other algorithms in the fitness computation, such as linear programming and rule-based algorithms. We already gave an example for employing ANNs to do so [37]. Further research can include experimenting with different types and architectures of ANNs and comparing the results.

G. Artificial Neural Networks

Among the strengths of ANNs is the ability to generate polyphonic music [46] as well as four-part chorales [9, 45]. RNNs and CRBMs are excelling in music composition applications due to their capability of capturing long-term dependencies as is the case between musical notes. Thus, these types of networks succeed in generating notes that are consonant with their previous ones.

Challenges: most challenging design feature in any ANN music composition application is the network architecture that best suits the musical task to be performed. This is sometimes problematic and even contradictory; as shown in the comparison between RNNs and BNs for chorale generation in [9]. The comparison could not favor one network over the other since the used BN failed to produce smooth basslines despite producing consonant harmonies in general, while the used RNN produced almost monotonous alto and tenor lines, but produced smooth basslines. Another challenge for ANNs is the type of data they deal with. Cottrell et al. [72] devised a study that gives insight and hints about how ANNs can deal with complex data. Another problem with using ANNs in music composition is the large energy and time consumption of some network types such as dependency networks [45].

Future Directions: The design architecture challenge opens the door for more research comparing the effect of different types of networks when implemented in each musical task. On the other hand, ANNs are already employed for GA's fitness, and thus, they are worth exploring for AIS evaluation as well.

H. Deep Neural Networks

Similar to shallow ANNs, DNNs were shown to produce appealing music to the ears of human listeners to a great extent [46, 55]. Furthermore, DNNs excel in extracting musical features only from the provided (training) data without any prior musical knowledge, thus offering a successful data-driven model. DNNs prevail to produce compositions of the same style based on the extracted musical features, hence providing more generalization [49, 50, 51]. Additionally, CNNs in particular provide a realistic simulation for the whole composition process; revisiting and enhancing previously generated melody partials [8].

Challenges: In addition to ANN challenges, the main challenge of DNNs is that as they get deeper, they achieve better results at the cost of time and power consumption [51].

Future Directions: A hot and promising research direction in DNNs for music composition would be using "*transfer learning*" [73]. This concept of machine learning is based on reusing a model, that has been pre-trained to perform a certain task, to perform another task. This method of using a pre-trained model as the starting point for another model, saves a lot of time and power. This method is popular in computer vision and language fields. It is of great interest to explore transfer learning in computer music composition field.

I. Generative Adversarial Networks

State-of-the-art GANs offer promising quality and appealing music. GANs aim to generate realistic musical pieces from complete random noise.

Challenges: GANs need suitable optimization procedures and a lot of parameters tuning in order to coordinate the work between the different elements of the network and consequently achieve the best musical results. GANs are still not perfect in capturing notes velocity and duration [67]. We cannot deny the high computational power needed to work with GANs. Since GANs were primarily developed to work on image data, they need special adaptation to be able to work on musical data.

Future Directions: The door is wide open for research and experimentation with GANs in the field of computer music composition. Representing music as images for GANs to train on, is a promising research area. Specifically, it is interesting to devise creative representations for music into images other than the traditional symbolic representation to capture more musical data within them. It is also worth further experimenting with increasing the number of discriminators such as the example in [67] (to handle more musical properties), or adding extra networks for musical quality control and even changing in the network architecture such as in [68]. It is also needed to handle more musical features in GANs such as velocities, note durations, and instrumentation.

13. CONCLUSION

In this paper we presented an informative survey about the most important algorithms that are frequently used in the field of computer music composition research. The top ten algorithms mentioned by order are: rule-based, case-based reasoning, Markov chains, generative grammars, linear programming, biologically inspired algorithms such as; genetic algorithms, artificial immune systems, artificial neural networks, deep neural networks, and finally, generative adversarial networks.

We started our survey by introducing the field of music composition, highlighting the main tasks involved in it. We then explored each of the aforementioned algorithms giving an overview and a description for each, supplying the needed explanatory diagrams and pseudocodes. We also focused on the application of each algorithm in the field of computer music composition. Moreover, we provided an insightful discussion and comparison between the presented algorithms shedding the light on their strengths, weaknesses, challenges, and future research directions.

Our study aims for guiding researchers to the best research paths in the field, paving the way for more innovation. Our work highlights the most suitable algorithm or technique for performing each music composition task. In summary: Rule-based systems are perfect for music theory representation. Case-based reasoning encapsulates musical experiences in the case-base. Markov chains excel in predicting new musical notes given a previous one. Generative grammars enable fast generation of musical pieces that adhere to music rules. Linear programming is used for timbre synthesis. The biologically inspired algorithms enhance the quality of generated music and can apply various musical styles to it.

REFERENCES

[1] L. Hiller and L. Isaacson, "Machine models of music," S. M. Schwanauer and D. A. Levitt, Eds. Cambridge, MA, USA: MIT Press, 1992, pp. 9–21.

[2] B. J. Copeland and J. Long, Eds., *Turing and the history of computer music*. Springer, Cham, 2017.

[3] N. Foy, "The word games of the night bird (interview with Christopher Strachey)," *Comput. Eur.*, pp. 10–11, 1974.

[4] Dean and R. T. (ed.), Eds., *The Oxford handbook of computer music*. New York: Oxford University Press, 2009.

[5] M. Navarro-Cáceres, S. Rodríguez, D. Milla, B. Pérez-Lancho, and J. M. Corchado, "A user controlled system for the generation of melodies applying case based reasoning," in *Case-Based Reasoning Research and Development*, 2017, pp. 242–256.

[6] A. Seago, S. Holland, and P. Mulholland, "A novel user interface for musical timbre design," in *Audio Engineering Society Convention 128*, May 2010.

[7] "Acoustical terminology," New York: American Standard Association, 1951, p. 25.

[8] C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. C. Courville, and D. Eck, "Counterpoint by convolution," *CoRR*, vol. abs/1903.0, 2019.

[9] T. Yamada, T. Kitahara, H. Arie, and T. Ogata, "Four-part harmonization: comparison of a Bayesian network and a recurrent neural network," in *Music Technology with Swing*, 2018, pp. 213–225.

[10] M. Navarro-Cáceres, M. Caetano, G. Bernardes, L. N. de Castro, and J. M. Corchado, "Automatic generation of chord progressions with an artificial immune system," in *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, 2015, pp. 175–186.

[11] K. Kunimatsu, Y. Ishikawa, M. Takata, and K. Joe, "A music composition model with genetic programming - a case study of chord progression and bassline," in *International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'15*, Jul. 2015, pp. 256–262.

[12] M. A. Kaliakatsos-Papakostas, A. Floros, and M. N. Vrahatis, "Intelligent generation of rhythmic sequences using finite L-systems," in *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Jul. 2012, pp. 424–427.

[13] R. L. de Mántaras, "Making music with AI: some examples," in *Rob Milne: A tribute to a Pioneering AI Scientist, Entrepreneur and Mountaineer*, A. Bundy and S. Wilson, Eds. IOS Press, 2006, pp. 90–100.

[14] G. Aguilera, J. L. Galán, R. Madrid, A. M. Martínez, Y. Padilla, and P. Rodríguez, "Automated generation of contrapuntal musical compositions using probabilistic logic in Derive," *Math. Comput. Simul.*, vol. 80, no. 6, pp. 1200–1211, 2010.

[15] P. Ribeiro, F. C. Pereira, M. Ferrand, and A. Cardoso, "Case-based melody generation with MuzaCazUza," in *AISB'01 Symposium on Artificial Intelligence and Creativity in Arts and Science*, 2002, pp. 67–74.

[16] D. Jurafsky and J. H. Martin, *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall, 2008.

[17] V. Padilla and D. Conklin, "Generation of two-voice imitative counterpoint from statistical models," *Int. J. Interact. Multimed. Artif. Intell.*, vol. 5, pp. 22–32, 2018.

[18] G. Wassermann and M. E. Glickman, "Automated harmonization of bass lines from Bach chorales: a hybrid approach," *Comput. Music. J.*, vol. 43, no. 2–3, pp. 142–157, 2020.

[19] N. Chomsky, *Aspects of the theory of syntax*. MIT Press, Cambridge, Mass., 1965.

[20] A. L. P. Prusinkiewicz, "The algorithmic beauty of plants," *Springer 1990. Springer-Verlag.*, pp. 101–107.

[21] F. L. and R. Jackendoff, "A generative theory of tonal music.," *Cambridge, Mass. MIT Press*, 1983.

[22] M. Hamanaka, K. Hirata, and S. Tojo, "Melody morphing method based on GTTM," in *International Computer Music Conference (ICMC)*, 2008, pp. 155–158.

[23] P. P. Cruz-Alcázar and E. Vidal-Ruiz, "Modeling musical style using grammatical inference techniques: a tool for classifying and generating melodies," in *Proceedings Third International Conference on WEB Delivering of Music*, 2003, pp. 77–84.

[24] P. P. Cruz-Alcázar and E. Vidal, "Two grammatical inference applications in music processing," *Appl. Artif. Intell.*, vol. 22, no. 1–2, pp. 53–76, Jan. 2008.

[25] L. F. R. Ralha and C. P. Brand, "L-systems, scores, and evolutionary techniques," in *Proceedings of the SMC 2009 - 6th Sound and Music Computing Conference*, Jul. 2009.

[26] D. Quick and P. Hudak, "Grammar-based automated music composition in Haskell," in *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling & Design*, 2013, pp. 59–70.

[27] O. Melkonian, "Music as language: putting probabilistic temporal graph grammars to good use," in *Proceedings of the 7th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design*, 2019, pp. 1–10.

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms, third edition*, 3rd ed. The MIT Press, 2009.

[29] D. Mintz, *Toward timbral synthesis: a new method for synthesizing sound based on timbre description schemes*. University of California, Santa Barbara, 2007.

[30] A. Seago, "A new interaction strategy for musical timbre design," in *Music and Human-Computer Interaction*, London: Springer London, 2013, pp. 153–169.

[31] G. Nierhaus, *Algorithmic composition: paradigms of*

automated music generation, 1st ed. Springer Publishing Company, Incorporated, 2008.

[32] P. J. P. de León, J. M. Iñesta, J. Calvo-Zaragoza, and D. Rizo, "Data-based melody generation through multi-objective evolutionary computation," *J. Math. Music*, vol. 10, no. 2, pp. 173–192, 2016.

[33] G. Carpentier, G. Assayag, and E. Saint-James, "Solving the musical orchestration problem using multiobjective constrained optimization with a genetic local search approach," *J. Heuristics*, vol. 16, no. 5, pp. 681–714, Oct. 2010.

[34] C.-H. Liu and C.-K. Ting, "Polyphonic accompaniment using genetic algorithm with music theory," *2012 IEEE Congr. Evol. Comput.*, pp. 1–7, 2012.

[35] C.-H. Liu and C.-K. Ting, "Music pattern mining for chromosome representation in evolutionary composition," *2015 IEEE Congr. Evol. Comput.*, pp. 2145–2152, 2015.

[36] R. De Prisco, G. Zaccagnino, and R. Zaccagnino, "EvoComposer: an evolutionary algorithm for 4-voice music compositions," *Evol. Comput.*, vol. 28, no. 3, pp. 489–530, 2020.

[37] I. A. Doush and A. Sawalha, "Automatic music composition using genetic algorithm and artificial neural networks," *Malaysian J. Comput. Sci.*, vol. 33, no. 1, pp. 35–51, 2020.

[38] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, May 1994, pp. 202–212.

[39] M. Read, P. S. Andrews, and J. Timmis, "An introduction to artificial immune systems," in *Handbook of Natural Computing*, G. Rozenberg, T. Bäck, and J. N. Kok, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1575–1597.

[40] M. Navarro-Cáceres, M. Caetano, G. Bernardes, and L. N. [de Castro], "ChordAIS: an assistive system for the generation of chord progressions with an artificial immune system," *Swarm Evol. Comput.*, vol. 50, p. 100543, 2019.

[41] M. Caetano, A. Zacharakis, I. Barbancho, and L. J. Tardón, "Leveraging diversity in computer-aided musical orchestration with an artificial immune system for multi-modal optimization," *Swarm Evol. Comput.*, 2019.

[42] M. Navarro-Cáceres, J. A. Castellanos-Garzón, and J. Bajo, "An intelligent system to generate chord progressions from colors with an artificial immune system," *New Gener. Comput.*, vol. 38, no. 3, pp. 531–549, 2020.

[43] Z. Waszczyszyn, "Fundamentals of artificial neural networks," in *Neural Networks in the Analysis and Design of Structures*, 1999, pp. 1–51.

[44] L. Fausett, Ed., *Fundamentals of neural networks: architectures, algorithms, and applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

[45] G. Hadjeres, F. Pachet, and F. Nielsen, "DeepBach: a steerable model for Bach chorales generation," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 2017, pp. 1362–1371.

[46] G. Brunner, Y. Wang, R. Wattenhofer, and J. Wiesendanger, "JamBot: music theory aware chord based generation of polyphonic music with LSTMs," in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, Nov. 2017, pp. 519–526.

[47] M. Nadeem, A. Tagle, and S. Sitsabesan, "Let's make some music," in *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, Jan. 2019, pp. 1–4.

[48] V. Pavlovsky, "Introduction to convolutional neural networks." 2017.

[49] B. van Werkhoven, J. Maassen, H. E.Bal, and F. J. Seinstra, "Optimizing convolution operations in CUDA with adaptive tiling," *Futur. Gener. Comput. Syst.*, vol. 30, pp. 14–26, Jan. 2014.

[50] F. Colombo, S. P. Muscinelli, A. Seeholzer, J. Brea, and W. Gerstner, "Algorithmic composition of melodies with deep recurrent neural networks," in *the First Conference on Computer Simulation of Musical Creativity (CSMC 2016)*, 2016.

[51] B. D. Smith, "Musical deep learning: stylistic melodic generation with complexity based similarity," in *Musical Metacreativity Workshop at the Eighth International Conference on Computational Creativity*, 2017.

[52] F. T. Liang, M. Gotham, M. Johnson, and J. Shotton, "Automatic stylistic composition of Bach chorales with deep LSTM," in *18th International Society for Music Information Retrieval Conference (ISMIR)*, 2017.

[53] D. D. Johnson, "Generating polyphonic music using tied parallel networks," in *Computational Intelligence in Music, Sound, Art and Design*, 2017, pp. 128–143.

[54] H. H. Mao, T. Shin, and G. W. Cottrell, "DeepJ: style-specific music generation," *CoRR*, vol. abs/1801.0, 2018, [Online]. Available: http://arxiv.org/abs/1801.00887.

[55] A. van den Oord *et al.*, "WaveNet: a generative model for raw audio," *ArXiv*, 2016.

[56] S. Vasquez and M. Lewis, "MelNet: a generative model for audio in the frequency domain," *CoRR*, vol. abs/1906.0, 2019, [Online]. Available: http://arxiv.org/abs/1906.01083.

[57] O. Cífka, U. Şimşekli, and G. Richard, "Groove2Groove: one-shot music style transfer with supervision from synthetic data," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 28, pp. 2638–2650, 2020.

[58] A. van den Oord *et al.*, "Parallel WaveNet: fast high-fidelity speech synthesis," *CoRR*, vol. abs/1711.1, 2017, [Online]. Available: http://arxiv.org/abs/1711.10433.

[59] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *NIPS*, 2014.

[60] A. Damien, "Generative adversarial network example." .

[61] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CoRR*, vol. abs/1611.0, 2016, [Online]. Available: http://arxiv.org/abs/1611.07004.

[62] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros,

"Unpaired image-to-image translation using cycle-consistent adversarial networks," *CoRR*, vol. abs/1703.1, 2017, [Online]. Available: http://arxiv.org/abs/1703.10593.

[63] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "MidiNet: a convolutional generative adversarial network for symbolic-domain music generation," in *International Society of Music Information Retrieval (ISMIR)*, 2017.

[64] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *AAAI Conference on Artificial Intelligence*, 2018.

[65] H.-W. Dong and Y.-H. Yang, "Convolutional generative adversarial networks with binary neurons for polyphonic music generation," *CoRR*, vol. abs/1804.0, 2018, [Online]. Available: http://arxiv.org/abs/1804.09399.

[66] M. Oza, H. Vaghela, and K. Srivastava, "Progressive generative adversarial binary networks for music generation," *ArXiv*, vol. arXiv:1903, 2019.

[67] G. Brunner, Y. Wang, R. Wattenhofer, and S. Zhao, "Symbolic music genre transfer with CycleGAN," *CoRR*, vol. abs/1809.0, 2018, [Online]. Available: http://arxiv.org/abs/1809.07575.

[68] C. Jin, Y. Tie, Y. Bai, X. Lv, and S. Liu, "A style-specific music composition neural network," *Neural Process. Lett.*, 2020.

[69] J.-Y. Liu, Y.-H. Chen, Y.-C. Yeh, and Y.-H. Yang, "Score and lyrics-free singing voice generation." arXiv:1912.11747, 2020.

[70] J.-Y. Liu, Y.-H. Chen, Y.-C. Yeh, and Y.-H. Yang, "Unconditional audio generation with generative adversarial networks and cycle regularization." arXiv:2005.08526, 2020.

[71] E. Miranda and J. Biles, Eds., *Evolutionary computer music*. Berlin: Springer, 2007.

[72] M. Cottrell, M. Olteanu, F. Rossi, J. Rynkiewicz, and N. Villa-Vialaneix, "Neural networks for complex data," *KI - Künstliche Intelligenz*, vol. 26, 2012.

[73] J. Torrey, L., & Shavlik, "Transfer learning," *Handb. Res. Mach. Learn. Appl. trends algorithms, methods, Tech. IGI Glob.*, pp. 242–264, 2010.

Nermin Naguib J. Siphocly: M.Sc. (2013) and B.Sc. (2008) in Computer Science from Ain Shams University, Cairo, Egypt. Teaching assistant since 2008 till now at the Faculty of Computer and Information Sciences, Ain Shams University. Taught various computer science courses. Participated in several projects at the same faculty. Published multiple papers in the areas of GPU Computing and Intelligent Computer Music Generation.

Professor El-Sayed M. El-Horbaty: received his Ph.D. (1985) in Computer science from London University, U.K., his M.Sc. (1978) and B.Sc. (1974) in Mathematics from Ain Shams University, Egypt.
His work experience includes 46 years as an academic in Egypt (Ain Shams University), Qatar (Qatar University), and Emirates (Emirates University, Ajman University, and ADU University).
Prof. El-Horbaty's current areas of research are Distributed and Parallel Computing, Cloud Computing, Edge Computing, Mobile Cloud Computing, e-health Computing, IoT, and Optimization of Computing Algorithms.
His work appeared in many journals such as: Parallel Computing, International Journal of High performance and Grid Computing, International Journal of Information Security, Advances in Intelligent System and Computing, Inter. J. of Mobile Network Design and Innovation, Inter. J. of Bio-Medical Information and e-Health.

Professor Abdel-Badeeh M. Salem: Professor of Computer Science at Ain Shams University, Cairo, Egypt, since 1989. Founder of the Artificial Intelligence and Knowledge Engineering Research Labs, Ain Shams University. Chairman of Working Group on Bio-Medical Informatics, ISfTeH, Belgium. Published around 600 papers (105 of them in Scopus). Has been involved in more than 600 international conferences and workshops as: keynote and plenary speaker, member of program committee, workshop/session organizer, session chair, and tutorials. Member of the editorial board of 50 international and national journals. Member of many international scientific societies and associations. Elected member of Euro Mediterranean Academy of Arts and Sciences, Greece. Member of Alma Mater Europaea of the European Academy of Sciences and Arts, Belgrade and European Academy of Sciences and Arts, Austria.