# EvoComposer: An Evolutionary Algorithm for 4-Voice Music Compositions

**R. De Prisco** robdep@unisa.it
Dipartimento di Informatica, University of Salerno, Fisciano (SA), 84084, Italy

**G. Zaccagnino** zaccagnino.gianluca@gmail.com
Dipartimento di Informatica, University of Salerno, Fisciano (SA), 84084, Italy

**R. Zaccagnino** rzaccagnino@unisa.it
Dipartimento di Informatica, University of Salerno, Fisciano (SA), 84084, Italy

**Abstract**

Evolutionary algorithms mimic evolutionary behaviors in order to solve problems. They have been successfully applied in many areas and appear to have a special relationship with creative problems; such a relationship, over the last two decades, has resulted in a long list of applications, including several in the field of music.

In this article, we provide an evolutionary algorithm able to compose music. More specifically we consider the following 4-voice harmonization problem: one of the 4 voices (which are bass, tenor, alto, and soprano) is given as input and the composer has to write the other 3 voices in order to have a complete 4-voice piece of music with a 4-note chord for each input note. Solving such a problem means finding appropriate chords to use for each input note and also finding a placement of the notes within each chord so that melodic concerns are addressed. Such a problem is known as the *unfigured harmonization problem*.

The proposed algorithm for the unfigured harmonization problem, named *EvoComposer*, uses a novel representation of the solutions in terms of chromosomes (that allows to handle both harmonic and nonharmonic tones), specialized operators (that exploit musical information to improve the quality of the produced individuals), and a novel *hybrid* multiobjective evaluation function (based on an original statistical analysis of a large corpus of Bach’s music). Moreover EvoComposer is the first evolutionary algorithm for this specific problem. EvoComposer is a multiobjective evolutionary algorithm, based on the well-known NSGA-II strategy, and takes into consideration two objectives: the harmonic objective, that is finding appropriate chords, and the melodic objective, that is finding appropriate melodic lines. The composing process is totally automatic, without any human intervention.

We also provide an evaluation study showing that EvoComposer outperforms other metaheuristics by producing better solutions in terms of both well-known measures of *performance*, such as hypervolume, $\Delta$ index, coverage of two sets, and standard measures of *music creativity*. We conjecture that a similar approach can be useful also for similar musical problems.

**Keywords**

Evolutionary algorithms, automatic music composition, evolutionary music.

## 1 Introduction

Since the invention of the computer, both computer scientists and artists have been considering the use of computers for the production of artifacts. Music is one of the arts that has most benefited from such a possibility. *Computer music* is as old as the computer itself: the ILLIAC suite described in Hiller and Isaacson (1958) is the first piece of music composed by a digital computer. The field of computer music encompasses many aspects across music and computer science, including sound synthesis, real-time performances, digital instruments, music notation software, and much more. In this article, we are concerned with the aspect of automatic music composition; that is, we focus our attention on the problem of composing music by means of a computer program. Computer-assisted composition is an active area of musical, technical, and humanistic research. The user interaction that computer-assisted music composition systems offer ranges from marginal help to fully automated systems (in which the user provides only the input). In this work we focus on *algorithmic* music composition, that is, the production of music through an algorithm that takes an input and, without any human intervention, produces new music. As an historical note, we remark that even "automatic composers" dating back to well before the computer era can be considered algorithmic (see, for example, Mozart's dice game[1]).

In the design of a music composition algorithm we need to take into account the fact that music composition is a process that must satisfy a set of *functional* (music theory/harmonization rules, genre-stylistic restrictions) and *aesthetic* (preferred patterns, style, motifs, etc.) goals. The functional goals are related to the specific music genre that the composition follows (classic, jazz, pop, etc.) and can be coded as formal constraints (e.g., parallel fifths are not allowed in classical music). The aesthetic goals are related to specific composers, and sometimes can violate functional goals. For example, despite the fact that parallel fifths are forbidden in classical music, Bach sometimes deliberately used this type of voices movement in his compositions. As we will see, in our approach we strive for both functional and aesthetic goals; the former are taken into account by explicitly coding the formal rules while the latter derive from the use of weights obtained through a statistical analysis of Bach's chorales.

Automatic composition of music could be seen, assuming the existence of a precisely defined evaluation metric, as a *combinatorial problem* that consists in placing a finite number of notes in a finite grid (the music sheet). Unfortunately (or fortunately?) such a metric does not exist, but we can exploit harmonic and melodic rules to evaluate a piece of music. Combinatorial problems often are characterized by an immense search space: many algorithms that work quite well in other settings, are not helpful since exploring the search space would take too long a time. Moreover, providing clever ways in order to guide an algorithm toward a good solution, can be difficult due to the nature of the problem. In fact, a small perturbation can lead to a wrong part of the search space completely disorienting the algorithm (e.g., Jacob, 1994). Adding heuristics that can improve the efficiency by restricting the exploration in significantly smaller search spaces (see, for example, Gartland-Jones and Copley, 2003).

Apart from locating a single optimum solution in a complex search space, an algorithm designed to solve creativity problems has an even greater task to undertake: to locate multiple solutions where "optimum" can be represented by terms like "interesting," "satisfying," and so on. In most traditional problems of physics or mathematics, this would make no sense, but from the aesthetics point of view, one single, unique, and global optimum might not exist; Bentley (2000) states the following: "The goal is to identify new and interesting solutions, normally more than one is desirable, and these

[1]For an explanation see https://en.wikipedia.org/wiki/Musikalisches_Würfelspiel.

solutions must be good. However, finding global optima may be undesirable, impractical, or even impossible.”

In this context, evolutionary computation (EC), and specifically evolutionary algorithms (EA), appear to work quite well and present an opportunity to explore extremely large search spaces in order to locate multiple optimal solutions (Papadopoulos and Wiggins, 1998) related to the problem domain (Goldberg, 2002), while combining exploration and optimization (Goldberg, 1989, 2002; Holland, 1975). According to Goldberg (2002), this is achieved primarily by a *continual innovation*, as the result of the evolutionary operators (selection, crossover, mutation) and their impact on the population.

## 1.1 Contributions of This Work

In this article, we focus on the *unfigured harmonization problem*. A related harmonization problem is the *figured* harmonization problem for which the input contains also the chords that have to be used. The figured harmonization problem is quite easier because the algorithm has only to find the voicing (relative positions of the notes of the chord) in each chord.

In this section, we provide a discussion about the motivations; then we give an overall description of the proposed strategy, and we explain the contribution of this article, discussing also its significance in the general context of EC and music.

**Motivations** The unfigured harmonization problem is a generalization of the so-called unfigured *bass* harmonization problem. In the traditional unfigured bass harmonization problem the composer is given a bass line as input and it has to compose other 3 voices to make a complete 4-voice piece of music (for further details refer to Buck, 1922 and Piston and DeVoto, 1987).

Such a problem is very important in classical music education. It is often given as a basic exercise in music composition classes. Formidable examples of 4-voice chorales are those composed by J.S. Bach, about three centuries ago. These chorales play a very important role in classical music education because they have served and continue to serve today as models for the study of harmony and counterpoint. They demonstrate Bach’s incomparable mastery in combining harmony and melody in a coherent and beautiful whole.

Here we consider a generalization of the unfigured bass problem in the sense that the input can be any of the four voices, not just the bass line. This generalization derives from the observation that the composition can start from other lines too, notably from the melodic one[2] (Buck, 1922).

**Proposed strategy** We present a novel evolutionary algorithm, *EvoComposer*, for the unfigured harmonization problem. The main novel technical aspects of the proposed algorithm are the following (see Section 4):

- we use an *hybrid multiobjective* approach that allows to consider simultaneously both the harmonic and the melodic aspects;
- we introduce specialized operators that exploit musical information to improve the quality of the produced compositions;
- we introduce a representation that allows handling also of nonharmonic tones;

[2]Bach himself used to start the composition of a chorale from the melody.

- we use an evaluation function based on weights derived from a statistical analysis of the corpus of Bach's chorales. As far as we know, a statistical analysis of Bach's chorales is not available in the literature and so, in order to obtain such a statistical analysis, we had to write a program able to analyze chorales. This tool was partially developed in Parisi (2013). Using this tool we derived tables of weights for chords and tonality changes.

We remark that the algorithm manages the four voices equivalently and thus it can compose taking as input any of the four voices. This might seem surprising at first glance, especially if the bass line contains more information about the composition (e.g., the bass often plays the root of the chord). However, since in our approach we do not have, and thus do not use, such extra information, the four voices are equivalent and the outputs are similar in terms of harmonic and melodic quality.

To find good solutions, the algorithm considers two *objectives*: the *harmonic* objective, that is, to find chords and the *melodic* objective, that is, to find an appropriate placement of the notes within each chord. It is important to find a good compromise between these objectives, so we use a *multiobjective* approach.

**Our contributions** A distinctive characteristic that makes EvoComposer different from other algorithms is the use of specialized variations of the well-known NSGA-II strategy. As detailed in Section 4, such variations concern: (1) a novel representation of the solutions in terms of chromosomes that allows handling of both harmonic and nonharmonic tones, (2) novel specialized operators that exploit musical information to improve the quality of the produced individuals and in particular (3) the definition of a novel *hybrid* multiobjective evaluation function.

In fact, to evaluate the produced music, usually, automatic composition systems use an objective function, according to either *existing rules from music theory* (e.g., Assayag et al., 1999; Geis and Middendorf, 2007; Herremans and Sörensen, 2013) or by *learning from a corpus of existing music* (e.g., Moray and Christopher, 2004). Although every musical genre has its own rules, these are usually not formally defined, and this represents a serious obstacle to the applicability of the first approach (e.g., Moore, 2001). The second strategy can be used to overcome such a problem, by automatically learning stylistic rules from existing music. In this work, we propose a novel *hybrid* approach: we define an evaluation function that, on one hand, adheres to classical music rules, and on the other hand, extracts statistical information from a corpus of existing music, to assimilate a specific composer's style. Specifically, we extracted statistical information from a large corpus of J.S. Bach's chorales and used it to guide the harmonic aspect of the composition process.

The main contribution of our work to the files of EC and music is that EvoComposer represents one of the first attempts to show that an evolutionary strategy based on a hybrid evaluation function and on customized operators based on musical information, allows the definition of, in a "very simple" way, an automatic music composition process able to satisfy both functional and aesthetic goals.

Thus, this work opens up new perspectives on the use of customized EC techniques in the field of musical creativity (and, maybe, also in other fields).

Concerning the evaluation of the algorithm, we compared the music produced by EvoComposer, with its predecessor EvoBassComposer (De Prisco, Zaccagnino et al., 2010) and with music produced by several well-known metaheuristics strategies, which are intelligent strategies used to design and improve general heuristic procedures with a high performance, and which have been already used for several other problems (e.g., Melián et al., 2003). As described in Section 5, in order to use other metaheuristics we have applied, when possible, similar customization in terms of representation of the solutions, evaluation function, and so on. The result is that our composer outperforms them by producing better solutions in terms of both well-known measures such as hypervolume, $\Delta$ index, coverage of two sets, and measures of music creativity obtained by interviewing music experts (Section 5). Furthermore, we also carried out an evaluation of the similarity of the compositions produced by EvoComposer with the corpus of Bach's chorales used to extract statistical information. We remark that the aim of this work is to produce "novel" music, not necessarily as close as possible to that of Bach. The analysis of the output, however, opens new perspectives for future work about music styles reproduction.

EvoComposer is totally automatic since it does not require any human intervention.

Finally, we remark that the description of EvoComposer provided in this article is very detailed and includes an in-depth description of all the aspects of the genetic algorithm (pseudocode, chromosome representation, crossover and mutation operators, fitness function and weights used, initial population, and configurations of the experiments). This allows any interested reader to actually implement the algorithm: this, usually, is not the case with other similar papers that provide evolutionary algorithms for music problems.

## 2 Related Work

The last decade has seen impressive advances in artificial intelligence (AI). Research studies have produced powerful applications that have a wide practical impact. *Machine creativity* is a recent trend in AI: it allows the design of improved engineering solutions at a lower cost. In this context, EC is likely to play a central role. A good recent survey of problems solved with evolutionary creativity is provided by Lehman et al. (2018). We refer the reader to such a survey for further references. All these successful applications of evolutionary creativity suggest that the approach can be useful also for "artistic" problems, as, for example, problems in the field of music. In this section, we review work concerning evolutionary music creativity.

This section is organized into two subsections. In the first one, we describe the use of EC in the field of music and arts in general, while in the second one, we consider the specific problem of music harmonization, which is the subject of this article. In each of the subsections we first discuss related papers and then we describe the approach followed in this article, highlighting the differences with previous works.

### 2.1 Music Problems and EC

#### 2.1.1 State of the Art

Several recent papers deal with creativity and EC and are focused on *evolving art*. Although they concern other domains, such as painting, some of these overlap, with respect to the creative aspect, with our music context (see Nguyen et al., 2015). Many of these works focus on evolving artificial intelligence and study how evolution produced complex, intelligent, diverse life on this planet by trying to computationally recreate it. A major focus is on evolving large-scale, structurally organized neural networks (networks with millions of connections that are modular, regular, and hierarchical). Some of these also investigate other bio-inspired AI techniques, such as deep learning. From the point view of the art, most of them concern the automatic production of images and games (e.g., Hastings et al., 2009, Donnelly and Sheppard, 2011, and Risi et al., 2015).

In the last years, the interest of composers and musicologists for *evolving music* has considerably increased. Indeed, it provides way to investigate musical ideas which is new but which is also "natural" in the sense that it follows the "evolutionary nature" of the compositional process, which, similarly to the evolutionary paradigm, goes through a generation of musical ideas and a selection of the most promising one for further iterated refining (Miranda and Biles, 2007). Moreover, music problems have intrinsically huge solutions spaces, which makes EC a natural approach.

Several EA have been proposed in this context and some of these consider different music problems (not the 4-voice harmonization). As an example, in Hofmann (2015) the authors proposed an automatic composer inspired by genetic programming which uses a tree-based domain model of compositions. The purpose is to overcome some limitations inherent in genetic representations of previous works in which the evolution is focused on only one or few musical aspects, such as pitch or rhythm. The model represents musical pieces as a set of constraints changing over time, creating musical contexts that allow us to compose, reuse, and reshape musical fragments.

Jeong et al. (2017) present a multiobjective evolutionary approach for automatic melody composition in order to produce a variety of melodies at once. To this end, two fitness evaluation functions are defined to evaluate a melody: *stability* and *tension*.

Lopes et al. (2017) present evolutionary methods for automatic melody composition based on two fitness functions, one defined using Zipf's law (1949) and the other defined using Fux's rules (Fux and Mann, 1965). The first one is defined using 14 phenomena (explained in Jensen, 2011) that represent different ways to evaluate music and calculate a fitness value as follows: (1) count how many times an event of that phenomenon occurs and order each pair *(event, occurrence)* by the number of occurrences in ascending order obtaining a list of pairs, (2) calculate the mean square error of the linear regression of the corresponding list of points divided by the amount of different events, and (3) sum all such values (multiplied by $-1$). Instead, the second one, that is the fitness function based on the Fux's rules, is a simple sum of seventeen values derived from counterpoint rules.

Scirea et al. (2016) describe MetaCompose, a music composition framework based on an evolutionary technique combining a feasible/infeasible two-population method (Kimbrough et al., 2008) and a multiobjective optimization. The composition generator produces music in multiple steps: (1) create a chord sequence, (2) evolve a melody fitting this chord sequence, and (3) create an accompaniment for the melody/chord sequence combination. Given a chord sequence, melodies are produced by generating a variable number of notes for each chord. These notes will evolve without duration information, according to several features divided into *constraints* and *objective* functions. Once the sequence of notes is created, it generates the duration of the notes randomly.

Liu and Ting (2015) explore the composition styles by miming music patterns of a specific composer. The patterns are used as genes and the composition styles are used for the generation of new music. To smooth the progression between phrases, a fitness function based on music theory is used.

### 2.1.2 EvoComposer

Obviously, the main difference between this article and the work cited earlier is the specific problem considered. Jeong et al. (2017) and Lopes et al. (2017) compose only melodies (one voice). The music produced by MetaCompose (Scirea et al., 2016) consists of a sequence of chords, a melody, and an accompaniment; that is, the composer does not produce a complete score. So, it does not provide a solution to the unfigured harmonization problem. The work of Liu and Ting (2015) considers pop music and handles only chords indication and the melody; again this is not what is required in the unfigured harmonization problem. The composer described in Hofmann (2015) targets general music so it is not a composer of chorale music.

Our approach aims at producing 4-part harmonizations in the style of Bach's chorales and exploits well-known music rules and also statistical information extracted from Bach's chorales. The papers mentioned earlier do not consider any constraint on style, genre, or theoretical rules in order to produce music that is coherent with a given style (except for the work of Liu and Ting (2015) for which patterns from songs of a specific pop singer are used).

For example, in Hofmann (2015), the authors define a list of statistical (Table 1 of Hofmann, 2015) and structural (Table 2 of Hofmann, 2015) fitness functions, on the basis of *subjective* considerations, such as dissonance, duration of notes, and pauses, using criteria defined by the authors.

Also in Scirea et al. (2016), the constraints and objective functions are based on subjective choices. For example, there are three constraints which are not related to any style or music theory: a melody should (i) not have leaps bigger than a fifth, (ii) contain at least a minimum amount of leaps of a major second, and (iii) each note pitch should be different than the preceding one.

In Liu and Ting (2015), the fitness function is based on evaluation rules and weights which represent personal choices of the authors (Table V of Liu and Ting, 2015). The weights we use in our fitness functions are derived from a statistical analysis of Bach's chorales.

In Jeong et al. (2017), the fitness functions measure the psycho-acoustic effects of the produced music, in terms of sound stability and tension, judged on a subjective basis. The fitness functions used in our work, instead, measures the produced music with respect to music composition rules and adherence to the chorale style.

In Lopes et al. (2017), the authors do use a subset of formal counterpoint rules to define the fitness function. However, the goal is not that of producing (polyphonic) music in counterpoint style. The produced music is monophonic (only one voice) and its quality is measured through a questionnaire about its pleasantness.

## 2.2 Music Harmonization

### 2.2.1 State of the Art

The specific problem of producing music harmonization has been faced in several ways and music genres. Automatic, or almost automatic, music composers can be found, for example in Cope (1996, 2000); Cope and Hofstadter (2004); Chuan and Chew (2007); Miranda (2000, 2003); Gimenes et al. (2005); Anders and Miranda (2008, 2011); De Prisco et al. (2010); and De Prisco et al. (2011a, 2011b). Among the techniques used in these works there are formal grammars, cellular automata, fractals, neural networks, Markov chains, evolutionary algorithms, and evolutionary music (DNA and protein-based music). The book by Miranda (2001) contains a good survey.

In the specific context of 4-voice music composition several automatic composers, that do not use an evolutionary approaches, have been proposed. For example Ebcioglu (1986) and Schottstaedt (1984) describe automatic composers based on rules and expert-systems. Another system capable of composing 4-voice chorales (and not only) is the EMI system by Cope (1996); the EMI system uses a combination of various techniques, such as formal grammars, rules, music analysis, and pattern matching. Gang et al. (1997) use an approach based on neural networks. Phon-Amnuaisuk and Wiggins (1999) investigated the use of heterogeneous cellular automata. Evans et al. (2014) described a system, AutoChorusCreator, capable of producing, in real time, a variety of 4-part harmonies starting from an input voice line *and* from the chords to be used; that is, the problem considered is actually the *figured* harmonization problem. The system is based on a approach of generating 4-part music with variations by incorporating two techniques, one based on statistical rules and another based on dynamic programming.

There are very few evolutionary algorithms that compose 4-voice music. For example, Jacob's (1994), which uses an interactive evolutionary approach (i.e., it needs human intervention) is based on evolutionary agents: the composer, the ear, and the arranger modules. The composer module produces music, the ear module filters out unsatisfactory material, and the arranger module imposes an order on whatever is left. The human operator judges the agents on their ability to produce pleasing music, and recombines successful agents to produce better agents.

Horner and Ayers (1995) have published a two-page abstract announcing an evolutionary algorithm to compose 4-voice music. No details are given, but from the overall description we understand that they approach the problem in two steps: in the first one, all possible chords for each note of the input are identified (using an enumeration) and then, in the second step, a genetic algorithm is used to find the "best" chord sequence. The fitness function used is very simple: it just counts the number of violations of the leading voice constraints (the set of constraints considered is not specified). The fitness function is the only detail provided about the genetic algorithm.

In Munoz et al. (2016), the authors use an adaptive multiagent memetic platform comprising various metaheuristics to tackle the unfigured harmonization problem. The platform is set up with the chromosome representation, the operators and the fitness functions, including the tables of weights, originally given in De Prisco, Eletto et al. (2010) and Zaccagnino (2012); a collection of experimental studies on Bach's 4-voice chorales showed that the cooperation among different optimization strategies yields improved *performance* over the conventional and hybrid evolutionary algorithms. Notice that the evaluation is solely about the *performance of the algorithm*, and it does not consider at all the quality of the produced music.

## 2.3 EvoComposer

EvoComposer is the result of a project that we started several years ago: in De Prisco and Zaccagnino (2009) we provided the "first version" of EvoComposer.[3] In that paper, the algorithm was able to solve the figured bass problem: that is, the input already contains the chords to be used and, thus, the algorithm has to find only the position of the voices for each chord in the input. In a PhD thesis, Zaccagnino (2012) tackled the more general problem of inputs without chords indications, that is the unfigured harmonization problem. His work produced the "second version" of the algorithm; a two-page extended abstract announcing the algorithm (without details) was published in De Prisco, Zaccagnino et al. (2010). Such a second version of the algorithm used tables of weights for chords and tonality change with arbitrary and only partially justified weights. This is a major drawback of the second version that (along with other aspects) we have improved in the version presented in this article.

[3]For previous versions of the algorithm we have used the name "EvoBassComposer." We have called this new version EvoComposer because while previous versions used the bass line as input, this new version is able to compose taking as input any of the four voices.

Comparing EvoComposer with other algorithms for 4-voice harmonization based on evolutionary techniques, we notice a big difference with the work of Jacob (1994): our composer is completely automatic while the other needs a substantial interaction with the user.

With respect to the work of Horner and Ayers (1995) the main difference is due to the use of musical information in the evolution process. Our chromosome representation, operators, and fitness function exploit specific musical information (rules from music theory, statistical data from Bach's music) while the approach of Horner and Ayers (1995) uses the standard evolutionary technique. Unfortunately, we cannot make a direct comparison with their algorithm. However, we remark that we ended up exploiting musical information because without it, the outputs of the composer were poor. In other words, in the very first attempts of this work we have tried with the standard approach and later on we have augmented it with customized choices based on musical information. We also remark that the published abstract in Horner and Ayers (1995) is very short and no details are provided. For example, it is not clear what the individuals of the population represent, nor what are the crossover and mutation operators (not mentioned).

With respect to AutoChorusCreator (Evans et al., 2014) there is a huge difference in the problem solved: EvoComposer solves the unfigured harmonization problem while AutoChorusCreator solves the figured harmonization problem. Moreover, AutoChorusCreator uses a set of heuristic rules which seems derived more from subjective musical taste rather than from standard classical rules or from statistical analysis of existing music in the intended style, as done, instead, for EvoComposer. We also remark that the paper does not provide many details (for example only 2 out of the 13 heuristic rules are explained) and that we have tackled basically the same problem, the *figured bass problem*, in an earlier paper (De Prisco and Zaccagnino, 2009).

With respect to Munoz et al. (2016), a big difference is the use of the statistical information extracted from Bach's chorales. We notice that this is a fundamental difference. Indeed the algorithm in Munoz et al. (2016) is basically equivalent to EvoBassComposer (De Prisco, Zaccagnino et al., 2010) because it uses exactly the *same* approach (chromosome representation, fitness functions, including the tables of weights, and the operators). As we show later in the evaluation section, the music produced by EvoComposer is by far better than that produced by EvoBassComposer; indeed the algorithm presented in this article is the result of many improvements made over the version presented in De Prisco, Zaccagnino et al. (2010) while the work of Munoz et al. (2016) is heavily based on De Prisco, Zaccagnino et al. (2010) and Zaccagnino (2012). In Section 5.2.2, we report the comparison with EvoBassComposer.

## 3 Background

In this section, we briefly recall the needed background to understand the rest of the article. We provide a brief music background section that recalls the needed music concepts (tonality, chords, harmony rules, etc.). Then we provide a brief section about the basic concepts of multiobjective evolutionary algorithms. We assume that the reader is already familiar with evolutionary algorithms.

### 3.1 The Unfigured Bass Harmonization Problem

We focus our attention on music composed in four voices; unparalleled examples of such type of compositions are J.S. Bach's chorales. A chorale consists of four independent voices: *bass*, *tenor*, *alto*, and *soprano*. Music is written in a sequence of measures,

Table 1: Chords considered and examples.

| Chord | Root note | Set of notes |
|---|---|---|
| Major triad | C | C, E, G |
| Minor triad | C | C, Eb, G |
| Major seventh | C | C, E, G, B |
| Minor seventh | C | C, Eb, G, Bb |
| Dominant seventh | C | C, E, G, Bb |
| Half-diminished seventh | C | C, Eb, Gb, Bb |

each consisting of a given number of beats. In each beat the four voices play (or sing) a note. It is also possible that in a beat a voice plays more than one note; in this case, the first note is typically the one that is part of the chord, while the other notes are passing tones. Thus we always have four notes in each beat that make up a chord. The chords that we consider are described in Table 1 (for each chord we give one example with C as root note for each chord).

The sequence of chords used to compose music is obviously decided by the composer. However, there are rules established by the theory of harmony. According to these rules there are some combinations of chords that work better than others, and there are particular chord sequences (e.g., cadences) that have specific musical functions.

Besides rules about chord sequences, there are also rules about melodic lines. Such rules aim at avoiding single- and two-voice errors. The rules can refer to the movement of a single voice (for example, normally a jump bigger than an octave is not allowed; jumps within the notes of the scale are preferred to jumps to notes not part of the scale) or also to the movements of two voices (for example, two voices that proceed by parallel fifths are not allowed).

Chords are built on the degrees of the scale of the tonality used. For example, if the tonality is D Major, which has two sharps, the scale is D,E,F♯,G,A,B,C♯,D. To abstract from the specific tonality used, the degrees of the scale are indicated with roman numerals: I,II,III,IV,V,VI,VII. On each degree we can build a chord. It is common notation to use such numerals to indicate the corresponding chord using capital letters for major chords, and small letters (i,ii,iii,iv,v,vi,vii) for minor chords.

The interested reader should refer to a standard textbook on harmony, like Piston and DeVoto (1987), for more information about music concepts, notation, and further explanations.

The specific composition problem that we consider in this article is the following: the composer is given an input line (voice) and has to compose the other three voices. An example is provided in Figure 1. Part *a* of the figure provides the input: only a bass line; part *b* provides a solution: the same bass line with the other three voices. As explained before, EvoComposer works also if the input line is the tenor, the alto, or the soprano line (obviously, it composes the missing three voices).

## 3.2 Multiobjective Evolutionary Algorithms

Most optimization problems encountered in practice involve multiple criteria that need to be considered. These so-called *objectives* are often conflicting and no single solution is usually simultaneously optimal with respect to all of them (usually trying to maximize one of the functions will inevitably lower the value of the other functions). Multiple objectives, thus, can produce a set of optimal solutions instead of a single optimal

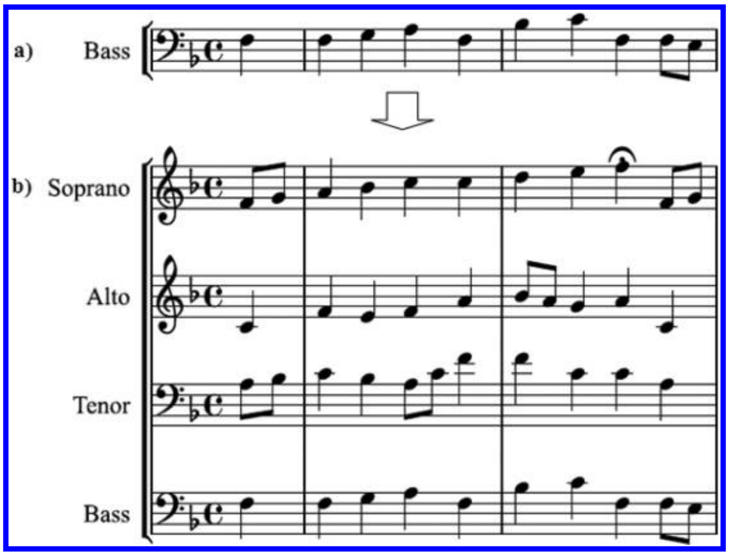


Figure 1: Unfigured bass harmonization example.

solution, because no single solution can be optimal for multiple conflicting objectives. A solution $s_1$ that is better than a solution $s_2$ with respect to all the objectives is clearly preferable. Such a solution $s_1$ "dominates" solution $s_2$ (and vice versa $s_2$ is dominated by $s_1$). A nondominated solution is an optimal solution. The set of nondominated solution is called the Pareto-front (see Fonseca and Fleming, 1995, 1993). Multiobjective evolutionary algorithms are a class of search methods that approximate the Pareto-front; that is, instead of guaranteeing to find the Pareto-optimal solutions, they aim at finding solutions as close as possible to the optimal ones. Once multiple such solutions are found, usually, higher-level decision-making strategies are adopted to pick a single solution from the Pareto-front.

In EvoComposer, we measure the goodness of a solution using a novel *harmonic objective*, to evaluate the sequence of chords, and a novel *melodic objective*, to evaluate the melodic lines. Hence we get a 2-objective evolutionary algorithm. As we already said, it might be impossible to maximize both objectives simultaneously. Hence we will have to find solutions that achieve good compromises amongst the objectives.

## 4 The EvoComposer Algorithm

In this section, we present our algorithm that we call EvoComposer. The algorithm takes as input a voice line and produces a complete 4-voice harmonization of the input line.

### 4.1 Algorithm Description

In this section, we describe the details of the algorithm. We start by providing high-level motivations and an overall description and then proceed to explain the details.

#### 4.1.1 Choices and Motivations

To develop EvoComposer, first we have chosen a reference multiobjective evolutionary strategy, and then we have customized the strategy to work with specific musical choices regarding the chromosome representation, the operators, and the fitness function. Thus, the evolutionary process (composition process) is obtained by running the chosen algorithm, with the addition of our customized musical choices.

The strategy chosen as reference is NSGA-II (Deb et al., 2002). We made such a choice for several reasons. From a *practical* point of view these reasons are (1) its popularity among practitioners and the vast available body of peer-reviewed material about it, (2) its proven efficiency on both benchmarks and real-word engineering problems, and (3) its clear algorithmic design and ease of implementation. NSGA-II has the advantage to incorporate techniques that could reflect in a natural way some music composition habits: the *elitism* and *diversity maintenance*. Elitism consists in maintaining an external population of all nondominated solutions; these are individuals with characteristics that we wish to keep in the final solution. As we will see, in the specific context of the unfigured harmonization problem it is important to preserve solutions having specific music harmonic and melodic properties, and use them during the composition process. Diversity is an important aspect of an evolutionary multiobjective optimization since it improves the coverage of the search space and allows for the exploration of different evolutionary paths leading to the trade-off surface. In the specific music composition context, the diversity of compositions used for producing new music allows avoidance of harmonic and melodic flattening and, so, to obtain always new ideas during the composition process.

Furthermore, to better shape the compositional music process as an evolutionary process we have defined a chromosome representation which, as we will see, reflects the structure of a chorale, and then we have added specialized operators guided by a fitness function built using statistical information extracted from a corpus of music written by Bach.

**Pseudocode** We assume that the reader is already familiar with NSGA-II concepts which are the basis of EvoComposer. Further details can be found in Deb et al. (2002). Here we provide a brief description.

Algorithm 1 shows the pseudocode for EvoComposer. The algorithm takes as input a voice line $line$, the size of the population $p_{size}$, the crossover and mutation probabilities $p_c$, $p_m$, and the maximum number of generations $max_{gen}$. It returns a set of solutions $P$, where each solution is a 4-voice chorale.

The first step is that of creating the initial population and this is done in line 1, function InitializePopulation(). Section 4.6 provides the details of this step.

Line 2, function Evaluate(), assigns to each element of the initial population its fitness values; this step is accomplished by applying the harmonic and melodic evaluation function that will be explained later in Sections 4.3–4.5.

The third step, in line 3, FastNonDominatedSort(), orders the elements of the initial population with respect to the nondominated relation.

At this point the evolution of the population starts. In each generation, that is each cycle of the **for** loop, the following happens. In line 5, a new population *Mate* is obtained from $P$ using the usual binary tournament selection, function SelectParentsByRank().

Then, in lines 6–7, we apply the evolutionary operators, which will be explained in Section 4.7, on the selected population of parents *Mate*.

In line 8 the set $U$ is constructed by simply merging the two populations $P$ and *Mate*.

Line 9 generates the set $F = (F_1, F_2, \ldots)$ that consists of the nondominated fronts of the set $U$; this is computed by the standard crowded-comparison operator, function FastNonDominatedSort().

**Algorithm 1:** EvoComposer pseudocode

```
    Input  : $line$,$p_{size}$, $p_c$, $p_m$, $max_{gen}$.
    Output: $P$.
 1  $P \leftarrow$ InitializePopulation($line, p_{size}$);
 2  Evaluate($P$);
 3  FastNonDominatedSort($P$);
 4  for $i = 1$ to $max_{gen}$ do
 5      Mate$\leftarrow$ SelectParentsByRank($P$);
 6      EvoCrossover(Mate, $p_c$);
 7      EvoMutation(Mate, $p_c$);
 8      $U \leftarrow$ Merge($P$,Mate);
 9      $F \leftarrow$ FastNonDominatedSort($U$);
10      $P \leftarrow \emptyset$;
11      $F_L \leftarrow \emptyset$;
12      foreach $F_j \in F$ do
13          CrowdingDistanceAssignment($F_j$)
14          if size($P$)+size($F_j$) $> p_{size}$ then
15              $F_L \leftarrow F_j$;
16              break;
17          else
18              $P \leftarrow$ Merge($P, F_i$);
19      if size($P$) $< p_{size}$ then
20          $F_L \leftarrow$ SortByRankAndDistance($F_L$);
21          $P \leftarrow$ FillRemaining($F_L$);
22      Evaluate($P$)
23  return $P$;
```

In the **foreach** loop, at lines 12–18, elements of the sets $F_j$ are sorted according to the the standard function CrowdingDistanceAssignment() and then inserted, if possible, in the new population $P$.

Finally, in lines 19–21, for the set $F_L$ that could not be inserted in its entirety, only the best solutions are inserted into the new population $P$.

The above is the standard NSGA-II strategy. Next we describe the specific choices that we have made for EvoComposer: (1) chromosome and gene representation in Section 4.2, (2) multiobjective evaluation function in Sections 4.3–4.5, (3) initial population procedure in Section 4.6, and (4) evolutionary operators in Section 4.7.

## 4.2 Chromosome and Gene Representation

The population in our evolutionary algorithm is made up of individuals (chromosomes) that are harmonizations of the given input line. As we can see in Figure 2, each chorale is organized in a sequence of measures, and each measure is organized in a sequence of beats. We consider as a basic time unit the length of the beats, and for each beat we have the set of notes for each voice in such a beat.

A chromosome, that is an individual of the population, is represented as a sequence of genes. Each gene represents a beat of the music and contains the notes used by the voices in that beat; we remark that since we consider also nonharmonic tones, each voice corresponds to one or more notes. To handle nonharmonic tones each gene is actually represented as a matrix of 4 rows and a fixed number $N$ of columns: the columns

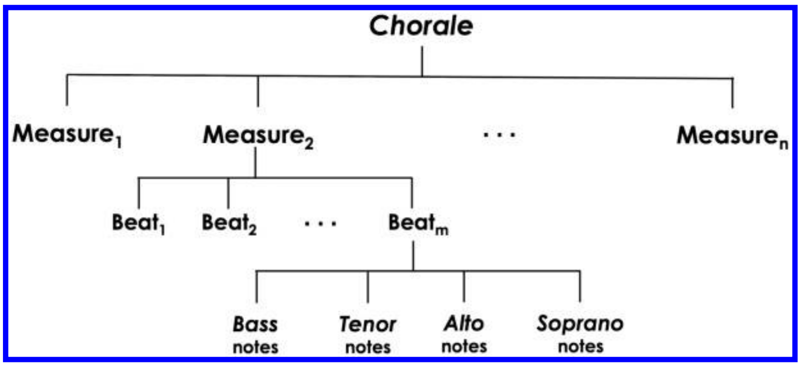


Figure 2: Chorale structure.

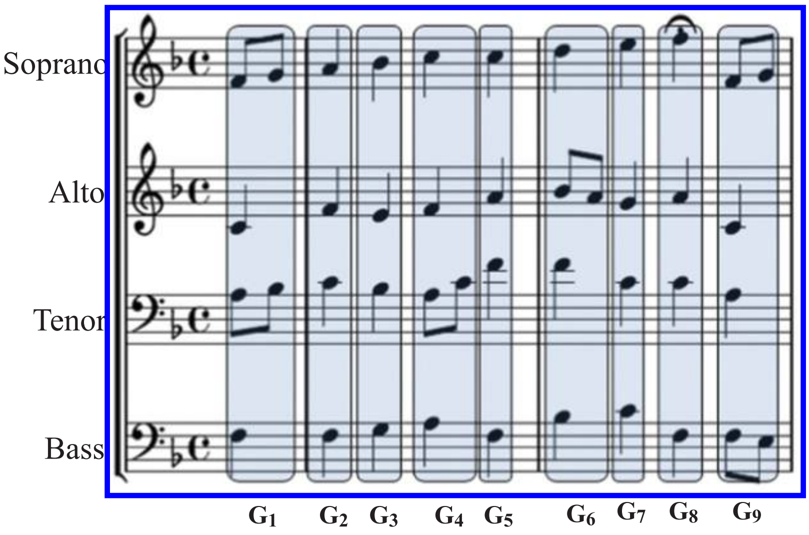


Figure 3: Chromosome example.

correspond to the notes. Thus an entire chromosome is an array $C$ with 4 rows and $N \cdot V$, where $V$ is the number of beats of the voice line (given in input) and $N$ is a parameter that determines the maximum number of notes that can be placed in each beat (we set $N = 16$). When a gene contains more than one note for a specific voice, only one of the note, typically the first one (see Figure 3), is part of the chord (harmonic tone); other notes are non-harmonic notes.

Formally, a chromosome $C$ is a sequence $C = [G_1, \ldots, G_V]$ where each $G_i$ is a gene and comprises the set of notes for the $i^{th}$ beat for each voice. We denote with $B_i$, $T_i$, $A_i$, and $S_i$ the list of notes of the bass, tenor, alto, and soprano, respectively. Figure 3 shows the genes for the harmonization provided in part b of Figure 1.

### 4.3 The Multiobjective Evaluation Function

EvoComposer uses two objective functions: an harmonic objective function $f_h$ and a melodic objective function $f_m$. Both functions have the following general form:

$$f = \sum_i a_i^h w_i^h, \tag{1}$$

Table 2: Typical harmonic chord passages in a major tonality (see Piston and DeVoto, 1987). The symbol – means that there is no chord in the corresponding class.

| Major | Cadence | | | |
|---|---|---|---|---|
| *Degree* | *often* | *sometimes* | *seldom* | *never* |
| $I \rightarrow$ | $I, IV, V$ | $vi$ | $ii, iii$ | $vii^\circ$ |
| $ii \rightarrow$ | $ii, V$ | $IV, vi$ | $I, iii$ | $vii^\circ$ |
| $iii \rightarrow$ | $iii, vi$ | $IV$ | $I, ii, V$ | $vii^\circ$ |
| $IV \rightarrow$ | $IV, V$ | $I, ii$ | $iii, vi$ | $vii^\circ$ |
| $V \rightarrow$ | $I, V$ | $IV, vi$ | $ii, iii$ | $vii^\circ$ |
| $vi \rightarrow$ | $ii, V, vi$ | $iii, IV$ | $I$ | $vii^\circ$ |
| $vii^\circ \rightarrow$ | $I, iii, vii^\circ$ | $vi$ | $ii, IV, V$ | – |

where $a_i^h$ are the coefficients and $w_i^h$ are weights. This is a general approach that many evolutionary algorithms use. As examples, we cite the fitness function that evaluates harmonized choral melodies and instrumental solos in Wiggins et al. (1998) or the fitness function that evaluates the aesthetic qualities of a wide range of melodies in Towsey et al. (2001).

The weights, $w_i$, are used to express the objective part of the evaluation while the coefficients are used to express a subjective component. The coefficients $a_i$ are normally obtained with a statistical analysis of known data.

As both functions require a considerable amount of description, we provide them in separate subsections.

### 4.4 The Harmonic Function

The harmonic function $f_h(C)$ is defined as follows:

$$f_h(C) = \sum_{i=1}^{n-1} a_i w_i, \tag{2}$$

and evaluates the harmonic quality of a chorale $C = (c_1, \ldots, c_n)$ by considering all pairs of consecutive chords $c_i, c_{i+1}$. The objective is to maximize $f_h(C)$. In our approach the coefficients represent the style of Bach while the weights represent well-known rules from the theory of harmony. More specifically we consider the following three possible cases:

1. $c_i$ and $c_{i+1}$ are chords in the same *major* tonality
2. $c_i$ and $c_{i+1}$ are chords in the same *minor* tonality
3. $c_{i+1}$ is identified as a modulation change, that is, $c_i$ belongs to the previous tonality while $c_{i+1}$ belongs to a new tonality (regardless of the mode, major or minor).

For each of these cases, we have defined a set of coefficients and weights.

- *Weights.* We relied upon well-known rules from the theory of harmony. We used as reference the description of the major harmonic progressions given by (Piston and DeVoto, 1987, page 17). Table 2 summarizes the "rules" for chord passages in a major tonality.

Table 3: Weights for chord passages in a major tonality.

| Major | Degree | | | | | | |
|---|---|---|---|---|---|---|---|
| *Degree* | $I$ | $ii$ | $iii$ | $IV$ | $V$ | $vi$ | $vii^\circ$ |
| $I \rightarrow$ | 0.266 | 0.025 | 0.025 | 0.266 | 0.266 | 0.150 | 0.002 |
| $ii \rightarrow$ | 0.025 | 0.400 | 0.025 | 0.075 | 0.400 | 0.075 | 0 |
| $iii \rightarrow$ | 0.016 | 0.016 | 0.400 | 0.150 | 0.016 | 0.400 | 0.002 |
| $IV \rightarrow$ | 0.075 | 0.075 | 0.025 | 0.400 | 0.400 | 0.025 | 0 |
| $V \rightarrow$ | 0.400 | 0.025 | 0.025 | 0.075 | 0.400 | 0.075 | 0 |
| $vi \rightarrow$ | 0.050 | 0.266 | 0.075 | 0.075 | 0.266 | 0.266 | 0.002 |
| $vii^\circ \rightarrow$ | 0.400 | 0.016 | 0.016 | 0.400 | 0.016 | 0.150 | 0.002 |

To compute the weights, $w_i$, we need to define a probability distribution for the classes "often," "sometimes," and "seldom" appearing in Table 2. These weights are subjective parameters; in order to make an educated choice we have tried several possibilities. More specifically, we tried values for $X_{often} \in \{60, 65, 70, 75, 80, 85, 90\}$, $X_{sometimes} \in \{10, 15, 20, 25, 30, 35, 40\}$, $X_{seldom} \in \{5, 10\}$, with the obvious constrain that the sum be 100. For each of such choices we have run several experiments, by fixing the number of iterations $k \in \{10, 50, 100, 500, 750, 1000, 2500, 5000, 7500, 10000\}$, and the max size $p_{max} \in \{10, 50, 100, 500, 750, 1000\}$. For each experiment, that is a combination of $(X_{often}, X_{sometimes}, X_{seldom})$ and a specific choice of $k$ and $p_{max}$ we ran 5 executions of EvoComposer and observed the structure of the generated solutions. We say that a chorale is *well-formed* if it starts with a $I$ degree and ends with the cadence $V - I$. This is a typical condition of well-composed chorales. Thus, we computed the average number of well-formed music compositions, obtaining the following results: for $d = (80, 15, 5)$, 87% of the solutions were well-formed; for $d = (85, 10, 5)$, 81%; for $d = (70, 20, 10)$, 76%; for $d = (60, 25, 15)$, 73%; for $d = (50, 40, 10)$, 69%. So, we set $(X_{often}, X_{sometimes}, X_{seldom}) = (80, 15, 5)$ as the probability distribution used for the experiments which will be described in Section 5.

Notice that since there can be more than one possible ending chord in each class, we need to split up the percentage of the class among the chords. For example, if $c_i$ is chord $I$, then the coefficient for $c_{i+1}$ will be $0.8/3 \simeq 0.26$ for each one of $I, IV$ and $V$, it will be about 0.26 for $vi$ and $0.05/2 = 0.025$ for $ii$ and $iii$. Table 3 shows all the weights for chord passages within a major tonality.

Similarly, we obtained the weights for chord passages within a minor tonality. Table 4 summarizes the typical chord passages suggested by Piston and DeVoto (1987). The passages are very similar to the ones in a major tonality with some differences due to the possibility of using the $VII$ chord in a minor tonality. Table 5 shows the corresponding weights. Notice that for the row of chord $vii^\circ$, there are no chords in the "sometimes" class; hence for this row we use a split of 95%–5% among the "often" and the "seldom" class.

To assign the weights for consecutive chords when a change of tonality occurs, we used the distance in the circle of fifths (see Figure 4), between the starting and the ending tonality. More specifically the weight $w_{Modulation}(S, E)$

Table 4: Typical harmonic chord passages in a minor tonality (see Piston and DeVoto, 1987). The symbol $-$ means that there is no chord in the corresponding class.

| Minor | Cadence | | | |
|---|---|---|---|---|
| *Degree* | *often* | *sometimes* | *seldom* | *never* |
| $i \rightarrow$ | $i, iv, V$ | $VI$ | $ii^\circ, III, vii^\circ$ | $VII$ |
| $ii^\circ \rightarrow$ | $ii^\circ, V$ | $IV, VI$ | $i, III$ | $vii^\circ$ |
| $III \rightarrow$ | $III, VI$ | $iv$ | $i, ii^\circ, V$ | $vii^\circ$ |
| $iv \rightarrow$ | $iv, V$ | $i, ii^\circ$ | $III, VI$ | $vii^\circ, VII$ |
| $V \rightarrow$ | $i, V$ | $IV, VI$ | $ii^\circ, III$ | $vii^\circ, VII$ |
| $VI \rightarrow$ | $ii^\circ, V, VI$ | $III, iv$ | $i$ | $vii^\circ, VII$ |
| $vii^\circ \rightarrow$ | $i$ | $-$ | $vii^\circ$ | $-$ |
| $VII \rightarrow$ | $III, VII$ | $VI$ | $iv$ | $i, ii^\circ, V, vii^\circ$ |

Table 5: Weights for chord passages in a minor tonality.

| Minor | Degree | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Degree* | $i$ | $ii^\circ$ | $III$ | $iv$ | $V$ | $VI$ | $vii^\circ$ | $VII$ |
| $I \rightarrow$ | 0.266 | 0.016 | 0.016 | 0.266 | 0.266 | 0.150 | 0.016 | 0 |
| $ii^\circ \rightarrow$ | 0.025 | 0.400 | 0.025 | 0.075 | 0.4 | 0.075 | 0 | 0 |
| $III \rightarrow$ | 0.012 | 0.012 | 0.400 | 0.150 | 0.120 | 0.400 | 0 | 0.120 |
| $iv \rightarrow$ | 0.075 | 0.075 | 0.025 | 0.400 | 0.400 | 0.025 | 0 | 0 |
| $V \rightarrow$ | 0.400 | 0.025 | 0.025 | 0.075 | 0.400 | 0.075 | 0 | 0 |
| $vi \rightarrow$ | 0.050 | 0.266 | 0.075 | 0.075 | 0.266 | 0.266 | 0 | 0 |
| $vii^\circ \rightarrow$ | 0.950 | 0 | 0 | 0 | 0 | 0 | 0.050 | 0 |
| $VII \rightarrow$ | 0 | 0 | 0.400 | 0.050 | 0 | 0.150 | 0 | 0.400 |

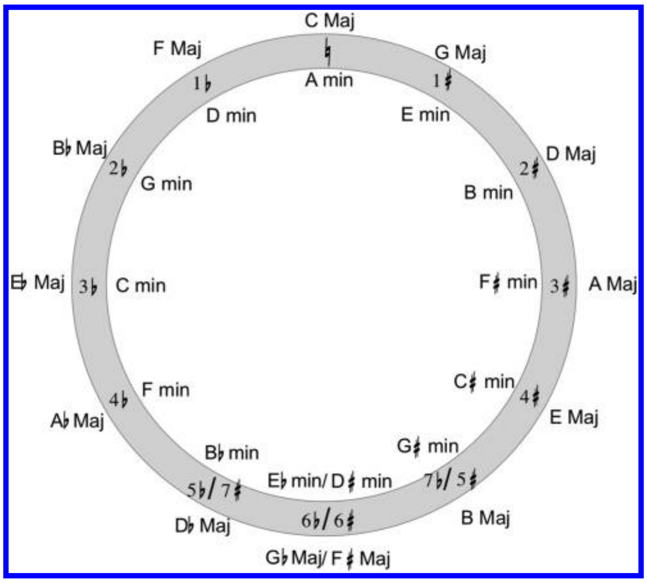


Figure 4: Circle of fifths.

Table 6: Coefficients for consecutive chords in the same major tonality.

| Major | Degree | | | | | | |
|---|---|---|---|---|---|---|---|
| *Degree* | $I$ | $ii$ | $iii$ | $IV$ | $V$ | $vi$ | $vii^\circ$ |
| $I$ | 8.1 | 2.5 | 0.5 | 0.7 | 18.5 | 1.7 | 0.3 |
| $ii$ | 0.9 | 0.6 | 0.4 | 0.2 | 10.4 | 0.2 | 0.2 |
| $iii$ | 0.9 | 0.3 | 0.6 | 0.5 | 0.8 | 0.6 | 0.1 |
| $IV$ | 0.6 | 0.5 | 0.1 | 0.3 | 5.3 | 0.1 | 0.2 |
| $V$ | 22.5 | 0.6 | 1.5 | 0.8 | 10.8 | 2.0 | 0.1 |
| $vi$ | 0.4 | 0.6 | 0.6 | 0.4 | 2.2 | 0.7 | 0.1 |
| $vii^\circ$ | 0.3 | 0.0 | 0.0 | 0.1 | 0.1 | 0.1 | 0.1 |

is the length of the shortest path from the starting tonality $S$ to the ending tonality $E$. For example, the distance between D major and C minor is 5 because we can go from $D$ major to C minor either counterclockwise using 5 steps or clockwise using 7 steps. As before, the weights for the chord passages are computed by defining a distribution for the classes "often," "sometimes," and "seldom," and so the values are always between 0 and 1. The weights for the modulations, instead, are computed as the distance on the circle of fifths of the tonalities. Thus, in order to maintain such weights in a comparable range with the weights for chord passages (between 0 and 1), we normalize the above distance over the maximum possible value, that is 6. For example, the (normalized) harmonic distances from C to G, D and A are, respectively 1/6, 2/6, and 3/6; the (normalized) harmonic distances from C to B♭, E♭ and A♭ are, 2/6, 3/6, and 4/6 respectively; the (normalized) harmonic distances from A♭ to B♭ minor, D♯ minor and G♯ minor are, 1/6, 2/6, and 3/6, respectively.

- *Coefficients.* The coefficients have been obtained by performing a statistical analysis over a large corpus of Bach's chorales. More precisely, we have written a program that analyzes chorales and that extracts information from the used harmonization. In particular, we looked for adjacent chords and we counted the percentage of passages from one chord to the subsequent one. We analyzed a corpus of Bach's chorales, ranging from chorale BWV 253 through chorale BWV 306 and from chorale BWV 314 to chorale BWV 438, for a total of 177 chorales.

  Tables 6, 7, and 8 summarize the result of the analysis. Notice that the sum of all the percentages in Tables 6 and 7 is smaller than 100 because there have been cases where our program was not able to identify the chords; those cases have not been classified, but simply ignored. In Table 8, many entries are zero because those specific changes of tonality were not encountered.

  As an example, let us consider the music fragment shown in Figure 1. The chromosome that represents the music fragment, which is in the tonality of F major, is $C =$ I-I-V-I-I-IV-I-I-I. Thus, the harmonic value: $f(C) = f_h(I, I) + f_h(I, V) + f_h(V, I) + f_h(I, I) + f_h(I, IV) + f_h(IV, I) + f_h(I, I) + f_h(I, I) = 8.1 * 0.266 + 18.5 * 0.266 + 22.5 * 0.4 + 8.1 * 0.266 + 0.7 * 0.266 + 0.6 * 0.075 + 8.1 * 0.266 + 8.1 * 0.266 = 2.1546 + 4.921 + 9 + 2.1546 + 0.1862 + 0.045 + 2.1546 + 2.1546 = 22.7706$.

Table 7: Coefficients for consecutive chords in the same minor tonality.

| Minor | Degree | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Degree* | $i$ | $ii^\circ$ | $III$ | $iv$ | $V$ | $VI$ | $vii^\circ$ | $VII$ |
| $I$ | 17.1 | 0.7 | 1.4 | 6.4 | 15.6 | 1.4 | 2.1 | 1.4 |
| $ii^\circ$ | 0.0 | 0.0 | 0.0 | 0.0 | 3.3 | 0.0 | 0.0 | 0.0 |
| $III$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $iv$ | 2.3 | 0.2 | 0.2 | 1.4 | 9.5 | 0.4 | 0.3 | 0.3 |
| $V$ | 22.6 | 0.0 | 0.0 | 0.8 | 8.0 | 0.0 | 0.0 | 0.1 |
| $VI$ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $vii^\circ$ | 0.5 | 0.0 | 0.1 | 0.2 | 0.4 | 0.0 | 0.0 | 0.0 |
| $VII$ | 1.1 | 0.0 | 0.0 | 0.0 | 0.2 | 0.0 | 0.0 | 0.0 |

Table 8: Coefficients $a_{Modulation}$ for change of tonalities. Major tonalities are shown in boldface.

| | **C** | **C#** | **D** | **D#** | **E** | **F** | **F#** | **G** | **G#** | **A** | **A#** | **B** | *Cm* | *C#m* | *Dm* | *D#m* | *Em* | *Fm* | *F#m* | *Gm* | *G#m* | *Am* | *A#m* | *Bm* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **C** | - | | 0.1 | | | 2.95 | | 4.6 | | 0.1 | | | | | | | | | | | | 1.48 | | |
| **C#** | | - | | | | | | | | | | | | | | | | | | | | | | |
| **D** | | | - | | | | | 3.2 | | 2.4 | | | | | | | | | | | | | | 1.15 |
| **D#** | | | | - | | | | | | | 1.85 | | | | | | | | | | | | | |
| **E** | | | | | - | | | | | 1.9 | | | | | | | | | | | | | | |
| **F** | | | | | | - | | | | | 2.7 | | | | | | | | | | | | | |
| **F#** | 2.4 | | | | | | - | | | | | | | | 1.1 | | | | | | | | | |
| **G** | 4.5 | | 2.9 | | | | | - | | | | | | | | | 1.65 | | | | | | | |
| **G#** | | | | | | | | | - | | | | | | | | | | | | | 1.2 | | |
| **A** | | | 2.87 | | | | | | | - | | | | | | | | | | | | | | |
| **A#** | | | | | | 2.5 | | | | | - | | | | | | | | | 1.6 | | | | |
| **B** | | | | 1.9 | | | | | | | | - | | | | | | | | | | | | |
| *Cm* | | | | | | | | | | | | | - | | | | | | | | | | | |
| *C#m* | | | | | | | | | | | | | | - | | | | | | | | | | |
| *Dm* | | | | | | | | | | | | | | | - | | | | | | | | | |
| *D#m* | | | | | | | | | | | | | | | | - | | | | | | | | |
| *Em* | | | | | | | | 1.7 | | | | | | | | | - | | | | | | | |
| *Fm* | | | | | | | | | | | | | | | | | | - | | | | | | |
| *F#m* | | | | | | | | | | | | | | | | | | | - | | | | | |
| *Gm* | | | | | | | | | | | 1.4 | | | | | | | | | - | | | | |
| *G#m* | | | | | | | | | | | | | | | | | | | | | - | | | |
| *Am* | 1.5 | | | | | | | | | | | | | | | | | | | | | - | | |
| *A#m* | | | | | | | | | | | | | | | | | | | | | | | - | |
| *Bm* | | | 1 | | | | | | | | | | | | | | | | | | | | | - |

## 4.5 The Melodic Function

The melodic function $f_m(C)$ evaluates the melodic quality of a chromosome $C$ by performing an "exception analysis" to identify stylistic anomalies and errors. We decided to classify the exceptions in two severity classes: "warning" and "error." A warning exception is intended to highlight a feature that might be stylistically unusual; an error exception indicates a problem that should be corrected. We assign the weights to an exception on the basis of its severity level: 2 for errors and 1 for warnings.

Table 9: Single voice and two voices errors: weights and coefficients.

| Exception Class | $w^m$ | $a^m$ | Meaning |
|---|---|---|---|
| Motion exceptions | | | |
| parallel octaves | 2 | 0.05 | consecutive octaves, same two voices |
| direct octaves | 1 | 0.01 | octaves by similar leap, any two voices |
| parallel fifths | 2 | 0.14 | consecutive fifths, any two voices |
| direct fifths | 1 | 0.02 | fifths by similar leap, any two voices |
| parallel unisons | 2 | 0.01 | consecutive unisons, any two voices |
| direct unisons | 1 | 0.01 | unisons by similar leap, any two voices |
| Voice exceptions | | | |
| voice jump | 1 | 6.9 | average motion, any voice |
| voice crossing | 1 | 0.19 | voice crossing, any pair |
| voice overlap | 1 | 0.04 | voice overlap, any pair |
| voice range | 1 | 0.08 | voice out of normal vocal range |
| voice spacing | 1 | 0.14 | wider that an octave, upper voices |

We consider two exception classes: motion exceptions and voicing exceptions. For each class we define a certain number of subtypes (see Table 9). We remark that many other categories and subtypes exceptions are possible and the rule system adopted by our algorithm can be easily expanded.

The coefficients have been obtained with a statistical analysis of a large corpus of Bach chorales, as done for the harmonic fitness function.

Given a chromosome $C$, its melodic value is given by

$$f_m(C) = \sum_i a_i^m w_i^m,$$

where the index $i$ runs over all errors and the values of $a_i$ and $w_i$ are reported in Table 9.

The objective is to minimize $f_m(C)$. Notice that, in order to make uniform the type of objective functions, in term of maximization or minimization, we consider $f'_m(C) = \frac{1}{f_m(C)}$, which is a function to maximize. Thus, we have to maximize $f_h(C)$ and $f'_m(C)$.

Let's consider an example and again take the one presented in Figure 1. There is a direct fifth in the first measure, a direct fifth and a direct octave in the second measure. So $f_m(C) = 1 \times 0.01 + 1 \times 0.01 + 1 \times 0.02 = 0.04$. So $f'_m(C) = \frac{1}{0.04} = 2.5$.

## 4.6 Initial Population

We start from an initial population $P$ of $p_{size}$ individuals. We have done experiments using two approaches for the initial population:

1. **Random individuals:** we obtain $p_{size}$ chromosomes by selecting a random chord for each input note and choosing random positions for the notes in each chord.
2. **In-tonality individuals:** we select $p_{size}$ individuals by choosing again random chords but with the constraint that the chords be chosen only in the starting tonality. If for some input note it is not possible to select such a chord, then we choose a random chord without restriction.

Recall that $B_j$, $T_j$, $A_j$, $S_j$ are the lists of notes for $G_i$ and that one of them (e.g., $B_j$) is fixed by the input. Initially the voices not fixed by the input (e.g., $T_j$, $A_j$, and $S_j$, if the bass line is the input) contain only one note and the duration of each note is assumed to be equal to that of a beat; that is, the note fills the entire beat. The mutation operator can introduce additional notes within each beat.

We have run tests using both choices. The tests with in-tonality initial populations provided better solutions. The data we report have been obtained using in-tonality initial populations.

## 4.7 Evolutionary Operators

We remark that the objective functions for our specific problem are defined on sequence of genes, that is, are calculated by checking the harmonic and melodic features of the genes of a chromosome. This property allows us to locate for each chromosome, the areas in which the characteristic of the genes are not good. We use this aspect to guide the evolutionary operators by fitness information: the operators, instead of operating in randomly selected points of the chromosomes, carefully choose such points to be the "worst areas" of the chromosome. This approach differentiates EvoComposer from many evolutionary algorithms, but reflects a natural way of approaching the music composition task. In fact, when musicians compose music, they usually operate in the worst points of their compositions, making changes, replaying patterns and so on, until they feel they have achieved a satisfying result.

It's common to fix the number of crossover points at a very low constant value of 1 or 2. We use a double-point crossover during the reproduction phase. In order to let the population evolve, we apply the following evolutionary operators: harmonic crossover and mutation operators and melodic crossover and mutation operators. Moreover, in order to implement the *elitism technique* we select at each generation those chromosomes that have at least one of the following characteristics: (1) first and last chords are in the starting tonality, (2) there exists at least one subsequence of chords that matches an harmonic "favored" sequence. The particular set of favored sequences that we have used in our algorithm is V-I, I-IV, II-V-I, IV-V-I, I-II-III, III-II-I, V-VI-VII-I. These sequences are very common. Notice that since we use a multiobjective fitness function the best individuals are given by those in the Pareto front, that is, the set of all nondominated chromosomes. Thus, among all individuals in the Pareto front we choose the solution which has the best harmonic value. This is justified by the fact that, being forced to make a choice, the harmonic aspect can be considered more important.

In the following, we describe the details of the crossover and mutation operators. We remark that the harmonic and melodic crossover and mutation operators were introduced in our previous paper on EvoBassComposer (De Prisco, Zaccagnino et al., 2010) while the classic and rhythmic melodic mutation are introduced in this article.

**Harmonic and Melodic Crossover** Let us consider the harmonic crossover. For each chromosome $C_1 = [G_1, \ldots, G_V] \in \mathit{Mate}$ selected for recombination, let $(G_i, G_{i+1})$ and $(G_j, G_{j+1})$, with $1 \leq i < j < V$, the two points of $C_1$ such that the values $f_h(G_i, G_{i+1})$ and $f_h(G_j, G_{j+1})$ are lowest among all pairs. These points are the crossover points considered for the harmonic recombination operator. Then, the algorithm looks at the chromosome $C_2 = [G'_1, \ldots, G'_V] \in \mathit{Mate}$ with the best value for $f_h(G'_i, G'_{i+1})$ and $f_h(G'_j, G'_{j+1})$ among all chromosomes in *Mate*. So we apply to $C_1$ and $C_2$ the harmonic crossover operator to produce a new chromosome $C_3 = [G_1, \ldots, G_i, G'_{i+1}, \ldots, G'_j, G_{j+1}, \ldots, G_V]$.

The melodic crossover works similarly (just substitute $f_h$ with $f_m$).

**Harmonic and Melodic Mutation** Let us consider the harmonic mutation. In the following, we use the notation $key(G)$ to indicate the tonality of chord $G$, $scale(G)$ to indicate the scale associated to $key(G)$, $position(n, scale(G))$ to indicate the degree of the note $n$ in the $scale(G)$, $pitch(n)$ and $duration(n)$ to indicate the pitch value and the duration of $n$, respectively; so $key(G') = key(G'')$ means that $G'$ and $G''$ are chords in the same tonality. For each chromosome $C \in$ *Mate* selected for mutation, we consider the pair $(G_i, G_{i+1})$ of genes with the worst value of $f_h(G_i, G_{i+1})$. Given a such pair $(G_i, G_{i+1})$, the harmonic mutation operator generates four new individuals:

1. Generate a new pair of genes $(G'_i, G'_{i+1})$ such that $f_h(G'_i, G'_{i+1}) > f_h(G_i, G_{i+1})$. Then a new chromosome $C'$ is generated by replacing $(G_i, G_{i+1})$ with $(G'_i, G'_{i+1})$;
2. Generate a new gene $G'_i$ such that $f_h(G'_i, G_{i+1}) > f_h(G_i, G_{i+1})$ and $key(G'_i) = key(G_{i+1})$. Then a new chromosome $C'$ is generated by replacing $(G_i, G_{i+1})$ with $(G'_i, G_{i+1})$. This mutation is biased toward keeping the tonality of $C_{i+1}$;
3. Generate a new gene $G'_{i+1}$ such that $f_h(G_i, G'_{i+1}) > f_h(G_i, G_{i+1})$ and $key(G'_{i+1}) = key(G_i)$. Then a new chromosome $C'$ is generated by replacing $(G_i, G_{i+1})$ with $(G_i, G'_{i+1})$. This mutation is biased toward keeping the tonality of $C_i$;
4. Generate a new pair of genes $(G'_i, G'_{i+1})$ such that $f_h(G'_i, G'_{i+1}) > f_h(G_i, G_{i+1})$ and $key(G'_i) = key(G'_{i+1})$. Then a new chromosome $C'$ is generated by replacing $(G_i, G_{i+1})$ with $(G'_i, G'_{i+1})$. This mutation is biased toward keeping the two chords in the same tonality.

For the melodic mutation, we consider two types of operators given a chromosome $C = [G_i, \ldots, G_V] \in$ *Mate* selected for mutation:

1. **Classic mutation:** we consider each gene $G_i$ of $C$ where a melodic exception labeled as "error" occurs. Then, for each of such gene $G_i$ the algorithm generates a new gene $G'_i$ that does not contain melodic exceptions classified as "error" and at the same time with fewer melodic exceptions classified as "warning." The melodic mutation operator produces from $C$ a new chromosome $C'$ where each gene $G_i$ is replaced by the new gene $G'_i$ as stated above.
2. **Rhythmic mutation:** the objective of this operator is to create rhythmic variety in each chromosome generated by inserting *nonharmonic tones*: *passing tones*, *appoggiatura tones* and *suspension tones* (see Piston and DeVoto, 1987 for more details). For the list of notes in each voice, which includes as the last note also the first note of the next beat, the rhythmic mutation operator chooses, uniformly at random, a nonharmonic type of tone, and adds it if possible. More specifically:
   - Passing tone (this is not always possible): Let $n_j, n_{j+1}$ be a pair of consecutive notes and let $key(G_i)$. If $octave(n_j) == octave(n_{j+1})$ and $|position(n_j) - position(n_{j+1})| = 2$ ($n_j$ and $n_{j+1}$ have distance 2 in the scale), then the operator adds the note $n'_j$ in between $n_j$ and $n_{j+1}$, that is, $position(n'_j)$ is such that $|position(n_j) - position(n'_j)| = |position(n'_j) - position(n_{j+1})| = 1$ and $octave(n'_j) = octave(n_j)$. Notice that the insertion of $n'_j$ after $n_j$ involves a redistribution of the original duration of $n_j$, which we split evenly (half to each) between the two notes $n_j$ and $n'_j$.

   - Appoggiatura tone (this is always possible): let $n_j$ be a note in the list; the operator adds a note $n'_j$ before $n_j$ in the list. The new note is such that $pitch(n'_j) = pitch(n_j) + 1$. Appoggiatura tones do not have a duration but are treated in a special way. We use a flag that specifies that the note is an appoggiatura tone; from a practical point of view the note will "take away" a little bit of the duration of $n_j$.

- Suspension tone (this is always possible): let $n_j$ be a note in the list; set $duration(n_j) = duration(n_j) + duration(n_{j+1})/2$ and $duration(n_{j+1}) = duration(n_{j+1})/2$.

## 5 Experimental Analysis

In this section, we report the results of tests that we have carried out to evaluate EvoComposer. We analyzed the *music quality* of the produced compositions from two points of view: (1) *performance*, by computing well-known measures such as hypervolume, $\Delta$ index, coverage of two sets, and (2) *musical creativity*, by interviewing music experts.

In the experiments we compared EvoComposer with the earlier system EvoBassComposer (De Prisco, Zaccagnino et al., 2010), and with other composers based on the following three multiobjective meta-heuristics:

- Tabu Search for Multiobjective Optimization: MOTS (see Hansen, 1997)
- A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA (see Bandyopadhyay et al., 2008)
- Multiobjective Particle Swarm Optimization: MOPSO (see Coello and Lechuga, 2002).

Clearly, and as we explain later, we have implemented our approach in each above frameworks. So in the experiments we considered five composers, EvoComposer, EvoBassComposer, and the above meta-heuristics.

As we will see in Section 5.1, we have run extensive tests by using a dataset of 20 of Bach's chorales (arbitrarily chosen). For each experiment we (1) selected a chorale in such a dataset, (2) selected as input one of the four voices of the chosen chorale, and (3) set the parameters (number of iterations, size of the population, etc.). Then, each of the five composers was run 10 times.

Next we describe the details of each experiment.

### 5.1 Configuration of the Experiments

We denote with $max_{gen}$ the number of iterations and with $p_{size}$ the maximum size of the population. We fix the number $max_{gen}$ of iterations in the set $K = \{10, 50, 100, 500, 750, 1000, 2500, 5000, 7500, 10000\}$, and the max size $p_{size}$ in the set $P = \{10, 50, 100, 500, 750, 1000\}$.

Thus, given a line $line$, for each pair $(max_{gen}, p_{size})$ we performed the following steps:

1. We ran 10 executions of EvoComposer, EvoBassComposer, and each composer implementing one of the meta-heuristics. Each composer takes as input: (1) the bass line $line$, (2) the number of generations $max_{gen}$, (3) the size of the initial population $p_{size}$, (4) the chromosome definition, and (5) the fitness functions.
2. For EvoComposer, we followed the standard experimental setup for NSGA-II (see Deb et al., 2002): The crossover rate is $p_c = 0.9$; the mutation rate is $p_m = 1/n$

where $n$ is the number of beats for each input bass line and binary tournament as selection operator.

3. For each experiment and for each composer, we computed the average harmonic and melodic values.

In total, for each composer we have $|K| \times |P| \times 20$ experiments and in the following section we report the results of the comparison.

For the implementations MOTS, AMOSA, and MOPSO we have used the same settings, fitness functions, chromosome representation, and operators (when possible) defined in Sections 4.2–4.7. In the following, we give details about the implementations. To ease the reader who wishes to check the references about MOTS, AMOSA, and MOPSO, in the following description, we have also reported the actual variable names used in the original papers.

- **MOTS.** Each individual is represented by using the chromosome defined in Section 4.2. We used the harmonic and melodic objectives defined in Section 4.3; the *sample size* was equal to the size of the population fixed for the experiment; for the *tabu list length* we used the values {0, 2, 4, 6, 8, 10} (as suggested in the experiment described in Section 6 of Hansen, 1997). As *stop criterion* we used the number of iterations fixed for the experiment; as the $s*$ function (required by MOTS) we used the weighted Tchebycheff distance described in Section 1.2 of Hansen (1997).

- **AMOSA.** Each individual is represented by using the chromosome defined in Section 4.2; we used the harmonic and melodic objectives defined in Section 4.3. The *perturbation operation* was obtained by randomly selecting one of the mutation operators defined in Section 4.7. The perturbation has been applied with probability $1/n$, where $n$ is the length of the voice line used as input for the experiment; the *minimum, $T_{\min}$, and maximum, $T_{\max}$, values of the temperature*; the *number of iterations at each temperature*, chosen so that total number of fitness evaluations becomes approximately equal to the number of iterations fixed for the experiment; the maximum size $SL$ of the Archive, chosen as the double of the size of the population fixed for the experiment. The minimum size $HL$ of the Archive has been set to the size of the population fixed for the experiment. (The choice of such minimum and maximum size of the Archive is justified by the fact that such sizes are sufficient to handle the size of the population for the experiment and the size of the set of nondominated solutions generated by EvoComposer).

- **MOPSO.** As for the other cases, each individual is represented by using the chromosome defined in Section 4.2 and the harmonic and melodic objectives are those defined in Section 4.3. The size of the *population* array *POP*, of the *velocity* array *VEL*, of the *nondominated* array *REP* and of the *best position* array *PBEST*, were chosen to be equal to the size of the population fixed for the experiment. The value of the *inertia weight W* was set to 0.4 (as suggested in Section 3 of Coello and Lechuga, 2002).

We also analyzed the differences among the five composers using the following studies. Concerning the performance, we studied the difference using nonparametric Friedman tests. Regarding the music creativity, we studied the correlations among the composers, for all the analyzed independent variables, through nonparametric Spearman's rho tests. We remark that, for both of the above nonparametric tests, the $p$-value is used, in the context of null hypothesis testing, in order to quantify the statistical significance: the smaller the $p$-value, the larger the significance. The Shapiro–Wilk goodness-of-fit test (Shapiro and Wilk, 1965) was used to assess the normality of the data. To assess the internal consistency reliability among multi-item scales, we have used Cronbach's alpha (1951).

The output of the experiments has also been the subject of a questionnaire administered to music experts according to the Consensual Assessment Technique, CAT (Amabile, 1982). The details of such a questionnaire will be given later in Section 5.2.2. The responses were analyzed using SPSS[4] (version 20).

## 5.2 Results

In this section, we describe the results of the comparison we carried out to identify the best composer in terms of *performance* and of *music creativity*.

### 5.2.1 Performance Metrics

Over the years, many performance metrics have been introduced for evaluating the quality of the nondominated fronts for multiobjective optimization problems. In this work, we focus on: (1) the minimization of the distance of the resulting nondominated front to the Pareto-optimal front, (2) the search of a desirable distribution of the solutions, and (3) the maximization of the spread of the obtained nondominated front. Thus, in our comparison we evaluated the considered composers by using the following:

- **Hypervolume** (see Zitzler and Thiele, 1999): this metric takes into consideration the size of the dominated volume in the objective space. In the two-dimensional case, this metric is mathematically described as:

  $$H = \left\{ \sum_i S_i | x_i \in P \right\},$$

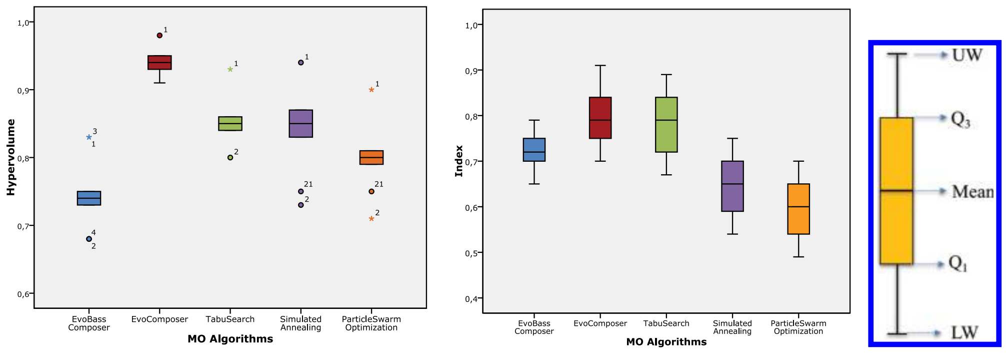
  where $P$ is the nondominated solution set under evaluation and $S_i$ is the area dominated by the solution $x_i$. The areas $S_i$ are computed with respect to a reference point in the objective space which typically is assumed to be composed of the maximum value for each objective. A greater value of $H$ indicates both a better convergence and as well a good coverage of the evaluated front (see Minella et al., 2008).

- **$\Delta$ index** (see Deb et al., 2002): this metric is based on a distance and includes information about both spread and distribution. The $\Delta$ index is computed as:

  $$\Delta = \frac{d_f + d_l + \sum_{i=1}^{|P|-1} |d_i - \overline{d}|}{d_f + d_l + (|P| - 1) \cdot \overline{d}},$$

  where $P$ is the front to be evaluated, $d_f$ and $d_l$ are Euclidean distances between the extreme solutions and the solutions, and the boundary solutions of $P$, $\overline{d}$ is the average of all distances $d_i$, $i = 1, \ldots, |P| - 1$, representing the Euclidean distance between consecutive solutions.

[4]Retrieved from https://www.ibm.com/it-it/analytics/spss-statistics-software



(a) Average hypervolume indicator (b) Average Δ index indicator. (c) Legend

Figure 5: Average metrics indicator for each MO algorithm.

- **Coverage of two sets** (see Zitzler and Thiele, 1999): this binary metric is computed by considering two fronts to be compared one against the other. It is computed as:

$$C(A, B) = \frac{|\{y \in B : \exists x \in A, x \preceq y\}|}{|B|},$$

  where $A$ and $B$ are the two fronts to be compared and $\preceq$ represents the weak dominance relation. Values of the $C$ function range in interval [0,1] ($C(A, B) = 1$ means that all solutions in $B$ are dominated by $A$, and as opposite $C(A, B) = 0$ means that none of the solutions in $B$ is dominated by the front $A$). We say that $A$ outperforms $B$ if $C(A, B) > C(B, A)$.

Figure 5 shows the average hypervolume value, and the average Δ index value respectively, for each algorithm by using a boxplot diagram. Each boxplot shows the average median value, the average first ($Q_1$) and the average third ($Q_3$) quartile, and the average upper (UW) and average lower (LW) whiskers. Values outside the whiskers represent experiments whose result deviated significantly from the median; the circle indicates outliers, that is, values outside the whiskers; the asterisk indicates extreme outliers, that is, values very far from the whiskers (specifically points lower than $Q_1 - 3(Q_3 - Q_1)$ or greater than $Q_3 + 3(Q_3 - Q_1)$). For example, for EvoBassComposer, experiments 2 and 4 gave a result that deviated moderately from the median, while experiments 1 and 3 gave a result that deviated significantly from the median. As we can see, EvoComposer provides the best performance among all compared multiobjective algorithms. Indeed the median of the average hypervolume values and of the average Δ index values is greater than the corresponding medians of the others. We also found out significant statistical differences between the algorithms for both the average hypervolume and the Δ-index indicator ($p$-value $< 0.0001$).

Furthermore, in Table 10 we report the average coverage metric over the 800 experiments. As we can see, also in this case the value for EvoComposer is greater than the values for each of the other composers.

As an example in the online folder (https://goo.gl/swM6V9) we provide the Pareto front for one of the experiments where we have used $max_{gen} = 10000$ and $p_{size} = 1000$. The set has size 30; the average size of the Pareto fronts observed during the experiments is about 43.

Table 10: Average coverage metric.

| | EvoBassComposer | EvoComposer | TabuSearch | SimulatedAnnealing | ParticleSwarm |
|---|---|---|---|---|---|
| EvoBassComposer | 1 | 0.32 | 0.54 | 0.46 | 0.61 |
| EvoComposer | **0.96** | **1** | **0.87** | **0.91** | **0.92** |
| TabuSearch | 0.71 | 0.77 | 1 | 0.84 | 0.79 |
| SimulatedAnnealing | 0.66 | 0.83 | 0.79 | 1 | 0.81 |
| ParticleSwarm | 0.73 | 0.84 | 0.87 | 0.90 | 1 |

#### 5.2.2 Music Creativity

Measuring creativity, and generally any inherently subjective perception of a work, is a difficult task. In our specific domain, the assessment of musical creativity poses many methodological and technical challenges. A common way to deal with these issues consists of proposing both a definition of creativity and a method for its assessment. Among these, the *product-based assessment* is a common way to evaluate musical creativity. The music pieces produced by the algorithm are evaluated by experts, through a questionnaire. The opinions of the experts are then often scored using the Consensual Assessment Technique, CAT for short (Amabile, 1982). CAT, also known as the "Gold Standard" of creativity assessment, is one of the most effective tools for measuring creative work. In this term, creativity is the ability to produce something that is both *new/original* and *valuable/appropriate* to the task or the domain (Amabile, 1996).

More specifically, this technique measures creativity using judges, who assess creativity works individually and in isolation. Judges are individuals who are experts in the reference field, so that the analysis of their opinions produces robust results.

Numerous studies have been proposed about the evaluation of musical creativity and many of these have demonstrated the ability of the CAT to obtain reliable subjective assessments (e.g., Amabile, 1996; Hennessey and Amabile, 2010; Baer et al., 2004; Boden, 1994, 1998, 2009; Hickey, 2001; Lubart et al., 2010; Pearce and Wiggins, 2001; Runco, 2004; Phon-Amnuaisuk et al., 2006; Wiggins, 2006; Langheinrich, 2001). Agres et al. (2016) argue that CAT is a valid assessment tool for the evaluation of music compositions.

Thus we use CAT to evaluate the quality, in terms of creativity, technical quality, and expressiveness, of the compositions produced in the experiments that we presented in Section 5.

**Method** An important factor when using CAT to analyze creativity is selection of credible judges. Our judges were 20 domain experts, either conservatory teachers or professional musicians. Ten music teachers, whose teaching experience ranged from 15 to 25 years, were recruited from two conservatories, that is, the "Carlo Gesualdo da Venosa" Conservatory in Potenza and the "Giuseppe Martucci" Conservatory in Salerno, Italy. The remaining participants were professional musicians, recruited through word-of-mouth advertising. They all had experience in classical music.

Our experts supplied their responses individually (independently and without influencing one another's judgment in any way) and received instructions verbally and in writing. Eleven judges declared high familiarity with the chorale genre, while the remaining nine were moderately familiar. Judges were not compensated for their participation and worked approximately for one hour. To build our dataset we selected the 5 best compositions produced by each system (as described in Section 5), for a total of 25 compositions. Each composition was created from a different input voice line and is long about 16 measures. The generated melodies were recorded as MIDI files and

Table 11: Consensual Assessment Technique questions.

| | Criteria |
|---|---|
| Creativity | |
| C1 | Does the composition show *original* and *imaginative* musical ideas? |
| C2 | Does the composition show *coherent* and *organized* musical ideas? |
| Technical Quality | |
| T1 | How successful is the composition as a *chorale form*? |
| T2 | Do the *harmonic* elements show technical correctness respect to classical rules? |
| T3 | Do the *melodic* elements show technical correctness respect to classical rules? |
| Expressiveness | |
| E1 | Is the composition musically *expressive* and reflects *aesthetic sensitivity* to music? |
| E2 | How do you evaluate the *listenability* of the composition? |

were presented (1) over stereo headphones from a laptop computer running a software media player, and (2) as printed score. The compositions were assessed using a seven-point Likert scale in three independent series for three criteria based on Amabile (1996); Hickey (1999); and Auh (1997): (1) Creativity (degree of both originality and coherence of the composition); (2) Technical quality (degree to which the harmonic and melodic elements in a composition show technical correctness respect to classical music rules); and (3) Expressiveness (degree to which the composition is musically expressive and reflects aesthetic sensitivity to music). The judges' agreement and therefore the inter-rater reliability was measured using Cronbach's (1951) alpha coefficient.

**Procedure** We asked judges to listen to and rate the 25 chorales as well as to analyze the corresponding music scores. They had to listen to each entire chorale before answering the seven questions included in the consensual assessment form. The questions, organized in the three different criteria before described, are shown in Table 11.

The compositions were presented in random order—subject to the constraints that no two compositions generated by the same system, or based on the same chorale, were presented sequentially. A reverse counterbalanced design was used, with half of the judges listening to the melodies in one such order and the other half listening to them in the reverse order.

Finally, judges filled out a questionnaire in which we asked them to provide demographic information, that is, name, age, and gender and their experience in the music field, and specifically the years of experience in music and the familiarity with chorale genre (low, medium, high).

The questionnaires submitted to music experts are available online.[5]

**Results** Results of the creativity analysis are shown in Table 12. As one can see, EvoComposer outperforms the other approaches with respect to all criteria, and thus also with respect to the mean of all criteria, with a 5.58 mean evaluation over a 7-point Likert scale. We also notice that, for all composers, the best rate was achieved in question T3: *"Do the melodic elements show technical correctness respect to classical rules?"* highlighting

[5]Retrieved from https://goo.gl/swM6V9

Table 12: The mean ratings for each question aggregated by composers. The questions labeled with C correspond to the questions about the Creativity criteria, those labeled with T to the Technical Quality, while those labeled with E to the Expressiveness criteria. Ratings on a 7-point Likert scale.

| Composer | C1 | C2 | T1 | T2 | T3 | E1 | E2 | Mean |
|---|---|---|---|---|---|---|---|---|
| *EvoBassComposer* | 3.21 | 3.37 | 3.77 | 3.73 | 4.17 | 3.79 | 3.17 | 3.60 |
| *EvoComposer* | 5.19 | 5.41 | 5.74 | 5.72 | 6.12 | 5.72 | 5.13 | 5.58 |
| *TabuSearch* | 4.20 | 4.36 | 4.78 | 4.74 | 5.16 | 4.79 | 4.21 | 4.60 |
| *SimulatedAnnealing* | 3.80 | 3.99 | 4.36 | 4.33 | 4.77 | 4.38 | 3.77 | 4.20 |
| *ParticleSwarm* | 3.41 | 3.56 | 3.97 | 3.93 | 4.35 | 3.99 | 3.42 | 3.80 |

Table 13: Analysis of correlations among the analyzed questions. Correlation significant at level 0.01 ($^{**}$) and at level 0.05 ($^{*}$).

| | | C1 | C2 | T1 | T2 | T3 | E1 | E2 |
|---|---|---|---|---|---|---|---|---|
| C1 | $\alpha$ | 1.000 | .623$^{**}$ | .727$^{**}$ | .699$^{**}$ | .701$^{**}$ | .626$^{**}$ | .712$^{**}$ |
| | *p-value* | | .003 | .000 | .001 | .001 | .003 | .000 |
| C2 | $\alpha$ | .623$^{**}$ | 1.000 | .848$^{**}$ | .678$^{**}$ | .509$^{*}$ | .594$^{**}$ | .623$^{**}$ |
| | *p-value* | .003 | | .000 | .001 | .022 | .006 | .003 |
| T1 | $\alpha$ | .727$^{**}$ | .848$^{**}$ | 1.000 | .907$^{**}$ | .499$^{*}$ | .858$^{**}$ | .932$^{**}$ |
| | *p-value* | .000 | .000 | | .000 | .025 | .000 | .000 |
| T2 | $\alpha$ | .699$^{**}$ | .678$^{**}$ | .907$^{**}$ | 1.000 | .624$^{**}$ | .907$^{**}$ | .926$^{**}$ |
| | *p-value* | .001 | .001 | .000 | | .003 | .000 | .000 |
| T3 | $\alpha$ | .701$^{**}$ | .509$^{*}$ | .499$^{*}$ | .624$^{**}$ | 1.000 | .649$^{**}$ | .495$^{*}$ |
| | *p-value* | .001 | .022 | .025 | .003 | | .002 | .027 |
| E1 | $\alpha$ | .626$^{**}$ | .594$^{**}$ | .858$^{**}$ | .907$^{**}$ | .649$^{**}$ | 1.000 | .875$^{**}$ |
| | *p-value* | .003 | .006 | .000 | .000 | .002 | | .000 |
| E2 | $\alpha$ | .712$^{**}$ | .623$^{**}$ | .932$^{**}$ | .926$^{**}$ | .495$^{*}$ | .875$^{**}$ | 1.000 |
| | $p$-value | .000 | .003 | .000 | .000 | .027 | .000 | |

how domain experts were positively impressed by the melodic aspect of the generated compositions. The highest value was given for EvoComposer ($M = 6.12$, $SD = 0.32$). It is worth to note that, for EvoComposer, the originality of the composition (Question C1: *"Does the composition present original and imaginative musical ideas?"*) was rated highly positively ($M = 5.19$, $SD = 0.56$).

As mentioned before, responses submitted by judges were then analyzed for inter-judge reliability using the Cronbach's alpha coefficient. For each of the criteria analyzed, an alpha of good reliability is achieved in all cases (0.80 or higher), with a best value of 0.92 for EvoComposer.

Finally, the correlations among the different items are presented in Table 13. As we can see, all independent variables were significantly correlated.

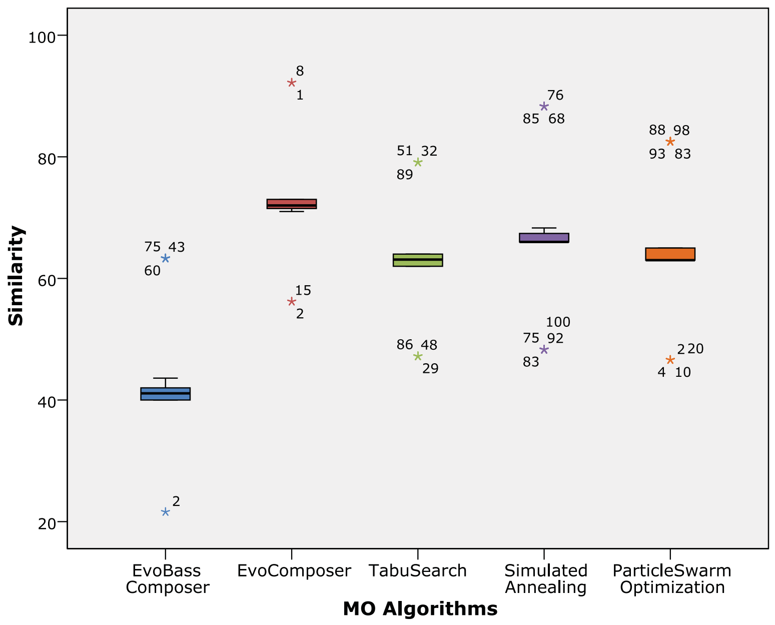


Figure 6: Test results: harmonic similarity with Bach's compositions.

### 5.3 Stylistic Considerations

As explained before, we have also evaluated EvoComposer by looking at how it performed compared, with respect to the harmonic aspect, to the original compositions used to extract statistical information (the chorales by Bach). As we will see, the results highlight EvoComposer's ability to produce very similar music with respect to the other approaches. We remark although the goal of EvoComposer is not that of mimicking Bach's style, the analysis shows that the algorithm can be also fine tailored for such a goal. More specifically, we have selected a large body of Bach's chorales (more than 200) and we have run each composer for each of the chorales giving as input the bass line. Then we compared the output composition to the original chorale written by Bach. The comparison has been made in terms of harmonization, that is, we have looked at the *chords* used for each beat and we have measured how much the automated composition differs from the one written by Bach. The overall measure is the percentage of chords where the automated composition matches the original Bach's composition: a composition that uses exactly the same chords used by Bach would get a score of 100%. Figure 6 shows the scores for the (over) 200 tests that we have performed.

For each composer we report the maximum, the minimum, and the mean scores. As we can see, EvoComposer outperforms the others' composers in terms of similarity, with a minimum of 56.2%, a maximum of 92.2%, and a mean score of 76%. This means that on average EvoComposer made the same harmonic choices made by Bach in about 3 out of 4 cases. We also remark that such a percentage of harmonic similarity with Bach's work is only an indication and does not necessarily mean that a composition with a higher percentage of similarity is better than another with a lower percentage of similarity. For example, we like more the solution provided by EvoComposer, for Chorale BWV 11.6, with a 72.4% harmonic similarity with Bach's harmonization, than the solution provided for Chorale BWV 6.6, with a 92.2% harmonic similarity with

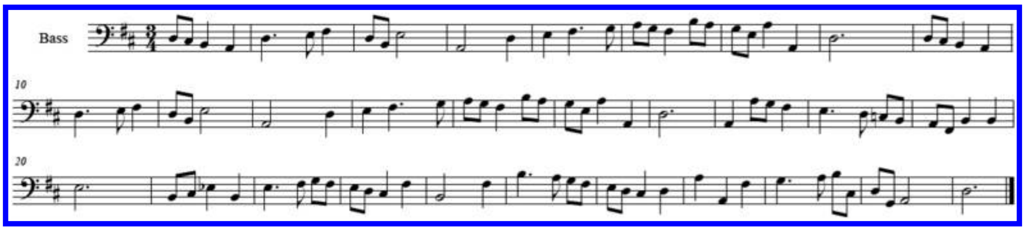


Figure 7: Bass line from Chorale BWV 11.6.

Bach's original harmonization. Such compositions are available online (https://goo.gl/swM6V9).

### 5.3.1 Chorale BWV 6.6 Example

As an example, we provide the output obtained for a specific chorale, namely BWV 6.6. Figures are in the Appendix. Figure 9 shows the original Bach's chorale, while Figures 10–13 provide the output obtained for four different executions of the EvoComposer, each one using a different voice as input: Figure 10 shows the output obtained using the bass line as input, Figure 11 shows the output obtained using the tenor line as input, Figure 12 shows the output obtained using the alto line as input, and Figure 13 shows the output obtained using the bass soprano as input.

### 5.3.2 Chorale BWV 11.6 Example

We also provide the full score of one specific composition obtained by using EvoComposer with the (input) bass line from Chorale BWV 11.6. The bass line is shown in Figure 7. The composition provided by EvoComposer is shown in Figure 8. We asked a string quartet to play the Chorale BWV 11.6 as composed by EvoComposer (the one shown in Figure 8). The execution has been filmed and is available online (https://goo.gl/swM6V9).

## 6 Conclusions

In this article, we have designed and implemented a totally automatic music composer using an evolutionary algorithm. The algorithm can handle music in 4 parts and can take as input any one of the 4 voices and produce as output the remaining 3 voices. The algorithm uses a novel representation for the chromosomes, and customized operators, which exploit specific musical information to improve the quality of the produced individuals and exploits weights obtained from a statistical analysis of a large corpus of Bach's chorales. The algorithm uses a multiobjective fitness function in order to consider simultaneously both the harmonic and the melodic aspects.

We have provided an in-depth description of the algorithm so that it is possible to implement it.

A performance study shows that our composer outperforms other meta-heuristics by producing better solutions in terms of both well-known measures such as hypervolume, $\Delta$ index, and coverage of two sets. Moreover, a qualitative evaluation was performed to measure the creativity of the chorales produced by all the analyzed composers. By applying standard mechanisms to evaluate creativity we found out that the experts involved in the study rated the compositions produced by our algorithm very original and expressive, as well as technically correct.

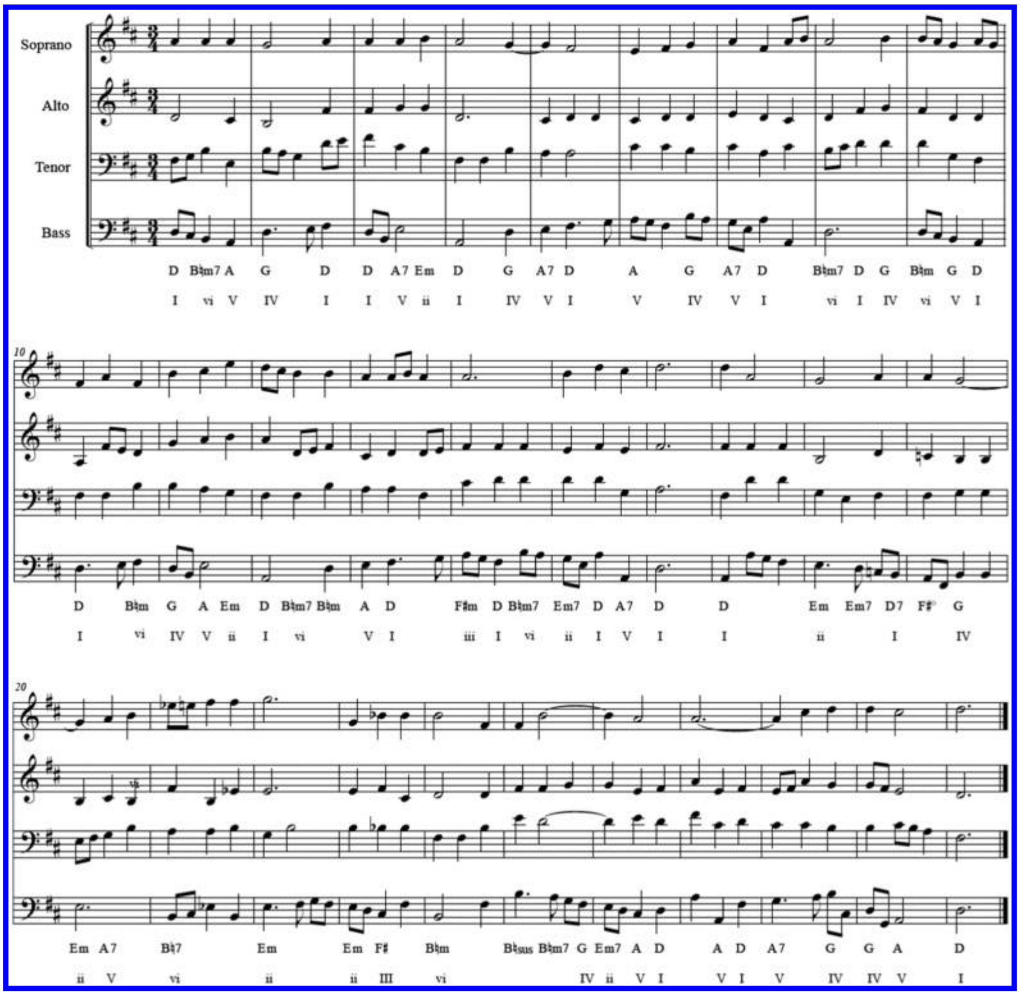

Soprano
Alto
Tenor
Bass
D B♮m7 A G D D A7 Em D G A7 D A G A7 D B♮m7 D G B♮m G D
I vi V IV I I V ii I IV V I V IV V I vi I IV vi V I
10
D B♮m G A Em D B♮m7 B♮m A D F♯m D B♮m7 Em7 D A7 D D Em Em7 D7 F♯ G
I vi IV V ii I vi V I iii I vi ii I V I I ii I IV
20
Em A7 B♮7 Em Em F♯ B♮m B♮sus B♮m7 G Em7 A D A D A7 G G A D
ii V vi ii ii III vi IV ii V I V I V IV IV V I

Figure 8: EvoComposer's harmonization for the bass line of Chorale BWV 11.6.

We have successfully applied the evolutionary computation approach to the music problem of 4-part harmonization, more specifically, the composition of 4-part chorales in the style of Bach, starting from one voice as input. Beside showing that the evolutionary approach can be effective for this specific music problem, this article suggests also that the use of musical information in the chromosome representation of customized operators and of statistical information in the fitness function could be useful for other music applications. Although this is unusual for evolutionary algorithms, it proved to be effective for this particular problem. We believe that this is due to the specific nature of the problem, for which the use of musical information helps the evolution of the solutions.

Future work might investigate the use of a similar approach to other musical problems, for example, targeting other classic composers and musical genres, such as jazz or pop music and tailor the algorithm on those composers/genres. Perhaps such an approach might be useful also for other EC problems.

Concerning the algorithm itself, future work can be targeted at improving several aspects. EvoComposer uses as metrics for the measurement of the quality of a composition only harmonic and melodic considerations. It would be interesting to consider also other aspects, such as rhythmic considerations and an evaluation of the overall structure of the composition.

## Acknowledgments

We would like to thank Matteo Parisi for having contributed with his thesis work to the development of the software tool that allowed us to analyze a large corpus of Bach's chorales to obtain the weights used in the algorithm. We also thank the anonymous referees who helped us to considerably improve the article with their comments.

## References

Agres, K., Forth, J., and Wiggins, G. A. (2016). Evaluation of musical creativity and musical metacreation systems. *Computers in Entertainment*, 14(3):3:1–3:33.

Amabile, T. M. (1982). The social psychology of creativity: A consensual assessment technique. *Journal of Personality and Social Psychology*, 43(5):997–1013.

Amabile, T. M. (1996). *Creativity in context*. Nashville, TN: Westview Press.

Anders, T., and Miranda, E. R. (2008). Constraint-based composition in realtime. In *Proceedings of the 2008 International Computer Music Conference*.

Anders, T., and Miranda, E. R. (2011). Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys*, 43(4):30:1–30:38.

Assayag, G., Rueda, C., Laurson, M., Agon, C., and Delerue, O. (1999). Computer-assisted composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal*, 23(3):59–72.

Auh, M.-S. (1997). Prediction of musical creativity in composition among selected variables for upper elementary students. *Bulletin of the Council for Research in Music Education* (133):1–8.

Baer, J., Kaufman, J. C., and Gentile, C. A. (2004). Extension of the consensual assessment technique to nonparallel creative products. *Creativity Research Journal*, 16(1):113–117.

Bandyopadhyay, S., Saha, S., Maulik, U., and Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm: Amosa. *IEEE Transactions on Evolutionary Computation*, 12(3):269–283.

Bentley, P. J. (2000). Exploring component-based representations—The secret of creativity by evolution? In I. Parmee (Ed.), *Evolutionary design and manufacture*, pp. 161–172. London: Springer.

Boden, M. A. (1994). Creativity: A framework for research. *Behavioral and Brain Sciences*, 17(3):558–570.

Boden, M. A. (1998). Creativity and artificial intelligence. *Artificial Intelligence*, 103(1):347–356.

Boden, M. A. (2009). Mind as machine (extract) and the creative mind: Myths and mechanisms (extract). In M. Boden, M. D'Inverno, and J. McCormack (Eds.), *Computational creativity: An interdisciplinary approach*, number 09291 in Dagstuhl Seminar Proceedings. Dagstuhl, Germany: Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, Germany.

Buck, P. C. (1922). *Unfigured harmony: A short treatise on modulation, harmonization of melodies, unfigured basses, inner melodies, canons and ground basses*. Oxford: Clarendon Press.

Chuan, C.-H., and Chew, E. (2007). A hybrid system for automatic generation of style-specific accompaniment. In *Proceedings of the 4th International Joint Workshop on Computational Creativity*, pp. 57–64.

Coello, C. A. C., and Lechuga, M. S. (2002). MOPSO: A proposal for multiple objective particle swarm optimization. In *Proceedings of the Conference on Evolutionary Computation*, Vol. 2, pp. 1051–1056.

Cope, D. (1996). *Experiments in musical intelligence*. Middleton, WI: A-R Editions.

Cope, D. (2000). *The algorithmic composer*. Middleton, WI: A-R Editions.

Cope, D., and Hofstadter, D. R. (2004). *Virtual music: Computer synthesis of musical style*. Cambridge, MA: MIT Press.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334.

De Prisco, R., Eletto, A., Torre, A., and Zaccagnino, R. (2010). A neural network for bass functional harmonization. In *Proceedings of Applications of Evolutionary Computation, Part II*, pp. 351–360.

De Prisco, R., Zaccagnino, G., and Zaccagnino, R. (2010). EvoBassComposer: A multi-objective genetic algorithm for 4-voice compositions. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pp. 817–818.

De Prisco, R., Zaccagnino, G., and Zaccagnino, R. (2011a). A genetic algorithm for dodecaphonic compositions. In *Proceedings of Applications of Evolutionary Computation*, pp. 244–253.

De Prisco, R., Zaccagnino, G., and Zaccagnino, R. (2011b). A multi-objective differential evolution algorithm for 4-voice compositions. In *2011 IEEE Symposium on Differential Evolution*, pp. 65–72.

De Prisco, R., and Zaccagnino, R. (2009). An evolutionary music composer algorithm for bass harmonization. In *EvoWorkshops*, pp. 567–572. Lecture Notes in Computer Science, Vol. 5484.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *Transactions in Evolutionary Computation*, 6(2):182–197.

Donnelly, P., and Sheppard, J. (2011). Evolving four-part harmony using genetic algorithms. In *EvoApplications 2011, Part II*, pp. 273–282, Berlin, Heidelberg: Springer.

Ebcioglu, K. (1986). An expert system for harmonizing four-part chorales. In *Proceedings of the 1986 International Computer Music Conference*.

Evans, B. L., Fukayama, S., Goto, M., Munekata, N., and Ono, T. (2014). Autochoruscreator: Four-part chorus generator with musical feature control, using search spaces constructed from rules of music theory. In *Proceedings of the International Computer Music Conference*.

Fonseca, C. M., and Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 416–423.

Fonseca, C. M., and Fleming, P. J. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16.

Fux, J. J., and Mann, A. (1965). *The study of counterpoint from Johann Joseph Fuxs Gradus Ad Parnassum—1725*. New York: Norton.

Gang, D., Lehmann, D., and Wagner, N. (1997). Harmonizing melodies in real-time: The connectionist approach. In *Proceedings of the International Computer Music Association*, pp. 27–31.

Gartland-Jones, A., and Copley, P. (2003). The suitability of genetic algorithms for musical composition. *Contemporary Music Review*, 22(3):43–55.

Geis, M., and Middendorf, M. (2007). An ant colony optimizer for melody creation with baroque harmony. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 461–468.

Gimenes, M., Miranda, E. R., and Johnson, C. (2005). Towards an intelligent rhythmic generator based on given examples: A memetic approach. In *Digital Music Research Network Summer Conference*, pp. 41–46.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*, 1st ed. Boston: Addison-Wesley.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Norwell, MA: Kluwer.

Hansen, M. P. (1997). Tabu search for multiobjective optimization: MOTS. In *Proceedings of the Thirteenth International Conference on Multiple Criteria Decision Making*, pp. 6–10.

Hastings, E. J., Guha, R. K., and Stanley, K. O. (2009). Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263.

Hennessey, B. A., and Amabile, T. M. (2010). Creativity. *Annual Review of Psychology*, 61(1):569–598.

Herremans, D., and Sörensen, K. (2013). FuX, an android app that generates counterpoint. In *IEEE Symposium on Computational Intelligence for Creativity and Affective Computing*, pp. 48–55.

Hickey, M. (1999). Assessment rubrics for music composition. *Music Educators Journal*, 85(4):26–33.

Hickey, M. (2001). An application of AMABILE's consensual assessment technique for rating the creativity of children's musical compositions. *Journal of Research in Music Education*, 49(3):234–244.

Hiller, L. A., and Isaacson, L. M. (1958). Musical composition with a high-speed digital computer. *Journal of the Audio Engineering Society*, 6(3):154–160.

Hofmann, D. M. (2015). A genetic programming approach to generating musical compositions. In *4th International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pp. 89–100. Lecture Notes in Computer Science, Vol. 9027.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: The University of Michigan Press.

Horner, A., and Ayers, L. (1995). Harmonization of musical progressions with genetic algorithms. In *International Computer Music Conference*.

Jacob, B. L. (1994). Composing with genetic algorithms. Technical Report. University of Michigan.

Jensen, J. H. (2011). Evolutionary music composition: A quantitative approach. Masters thesis. Institutt for datateknikk og informasjons vitenskap.

Jeong, J., Kim, Y., and Ahn, C. W. (2017). A multi-objective evolutionary approach to automatic melody generation. *Expert Systems Applications*, 90(C):50–61.

Kimbrough, S. O., Koehler, G. J., Lu, M., and Wood, D. H. (2008). On a feasible–infeasible two-population (FI-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research*, 190(2):310–327.

Langheinrich, M. (2001). Privacy by design? Principles of privacy-aware ubiquitous systems. In *International Conference on Ubiquitous Computing*, pp. 273–291.

Lehman, J., Clune, J., Misevic, D., Adami, C., Altenberg, L., . . . et al. (2018). The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *CoRR*. Retrieved from http://arxiv.org/abs/1803.03453

Liu, C. H., and Ting, C. K. (2015). Music pattern mining for chromosome representation in evolutionary composition. In *2015 IEEE Congress on Evolutionary Computation*, pp. 2145–2152.

Lopes, H. B., Martins, F. V. C., Cardoso, R. T. N., and dos Santos, V. F. (2017). Combining rules and proportions: A multiobjective approach to algorithmic composition. In *2017 IEEE Congress on Evolutionary Computation*, pp. 2282–2289.

Lubart, T., Pacteau, C., Jacquet, A.-Y., and Caroff, X. (2010). Children's creative potential: An empirical study of measurement issues. *Learning and Individual Differences*, 20(4):388–392.

Melián, B., Moreno, J., and Moreno, J. (2003). Metaheurísticas: Una visión global. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 7(19):7–28.

Minella, G., Ruiz, R., and Ciavotta, M. (2008). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, 20(3):451–471.

Miranda, E. R. (2000). On the music of emergent behaviour: What can evolutionary computation bring to the musician? In *Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 140–142.

Miranda, E. R. (2001). *Composing music with computers (music technology)*. Waltham, MA: Focal Press.

Miranda, E. R. (2003). On the evolution of music in a society of self-taught digital creatures. *Digital Creativity*, 14(1):29–42.

Miranda, E. R., and Biles, J. A. (2007). *Evolutionary computer music*. Berlin, Heidelberg: Springer.

Moore, A. F. (2001). Categorical conventions in music discourse: Style and genre. In *Music & Letters*, Vol. 82, pp. 432–442.

Moray, A., and Christopher, K. I. W. (2004). Harmonising chorales by probabilistic inference. In L. K. Saul, Y. Weiss, and L. Bottou (Eds.), *Advances in neural information processing systems*, pp. 25–32. Cambridge, MA: MIT Press.

Munoz, E., Cadenas, J. M., Ong, Y. S., and Acampora, G. (2016). Memetic music composition. *IEEE Transactions on Evolutionary Computation*, 20(1):1–15.

Nguyen, A., Yosinski, J., and Clune, J. (2015). Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 Conference on Genetic and Evolutionary Computation (GECCO)*.

Papadopoulos, G., and Wiggins, G. (1998). A genetic algorithm for the generation of jazz melodies. In *Proceedings of STEP 98*, pp. 7–9.

Parisi, M. (2013). Un analizzatore armonico automatico per composizioni musicali a 4 parti. Thesis (in Italian). Retrieved from http://music.di.unisa.it/Musimatica/Download_files/tesi-parisi.pdf

Pearce, M., and Wiggins, G. (2001). Towards a framework for the evaluation of machine compositions. In *Proceedings of the AISB Symposium on AI and Creativity in Arts and Science*, pp. 22–32.

Phon-Amnuaisuk, S., Smaill, A., and Wiggins, G. (2006). Chorale harmonization: A view from a search control perspective. *Journal of New Music Research*, 35(4):279–305.

Phon-Amnuaisuk, S., and Wiggins, G. A. (1999). The four-part harmonization problem: A comparison between genetic algorithms and a rule-based system. In *AISB99 Symposium on Musical Creativity*.

Piston, W., and DeVoto, M. (1987). *Harmony*. New York: Norton.

Risi, S., Lehman, J., D'Ambrosio, D., Hall, R., and Stanley, K. (2015). Petalz: Search-based procedural content generation for the casual gamer. *Transactions on Computational Intelligence in Games*, 8:244–255.

Runco, M. A. (2004). Creativity. *Annual Review of Psychology*, 55(1):657–687.

Schottstaedt, B. (1984). Automatic species counterpoint. Technical Report. CCRMA, Stanford, CA.

Scirea, M., Togelius, J., Eklund, P. W., and Risi, S. (2016). Metacompose: A compositional evolutionary music composer. In *Proceedings of the 5th International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pp. 202–217.

Shapiro, S. S., and Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611.

Towsey, M. W., Brown, A. R., Wright, S. K., and Diederich, J. (2001). Towards melodic extension using genetic algorithms. *Educational Technology & Society*, 4(2):54–65.

Wiggins, G., Papadopoulos, G., Phon-amnuaisuk, S., and Tuson, A. (1998). Evolutionary methods for musical composition. In *International Journal of Computing Anticipatory Systems*.

Wiggins, G. A. (2006). Searching for computational creativity. *New Generation Computing*, 24(3):209–222.

Zaccagnino, R. (2012). Music composition algorithms and musical gestures recognition. PhD thesis. Retrieved from http://music.di.unisa.it/Musimatica/Download_files/Rocco ZaccagninoTesi.pdf

Zipf, G. K. (1949). *Human behaviour and the principle of least effort*. Reading, MA: Addison-Wesley.

Zitzler, E., and Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *Transactions in Evolutionary Computation*, 3(4):257–271.

## Appendix: Examples of Output

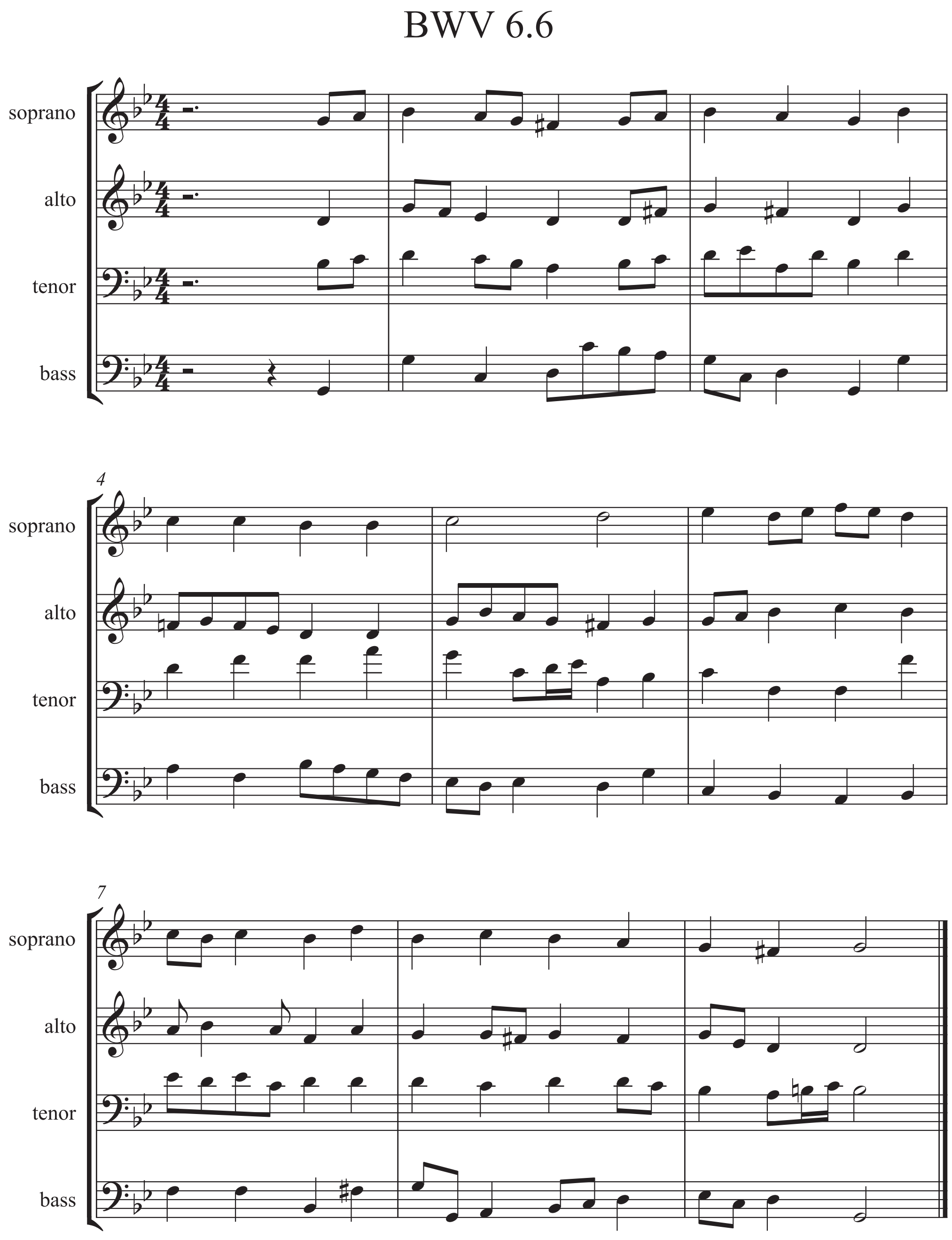


Figure 9: Chorale BWV 6.6.

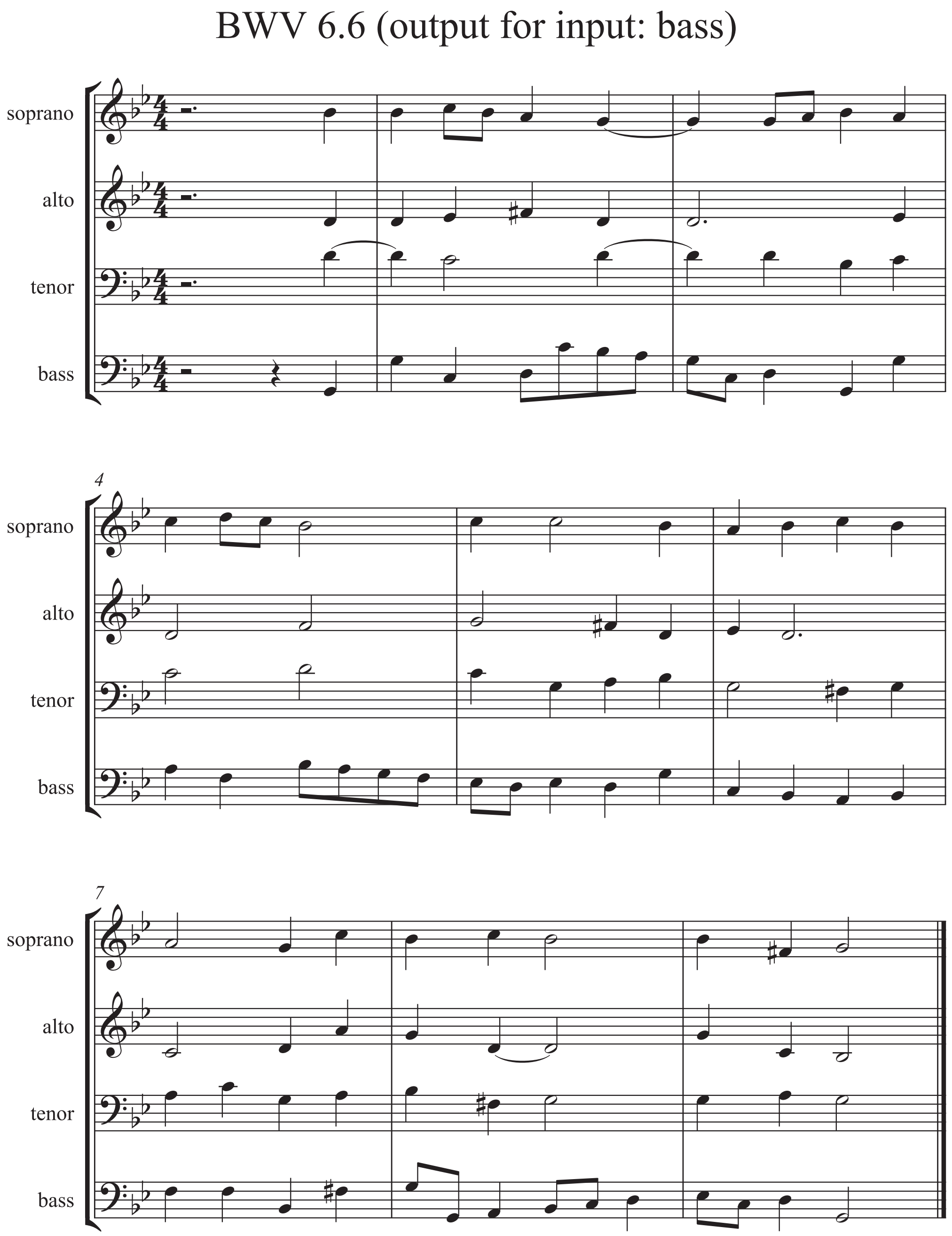


Figure 10: Output of EvoComposer on input the bass line of Chorale BWV 6.6.

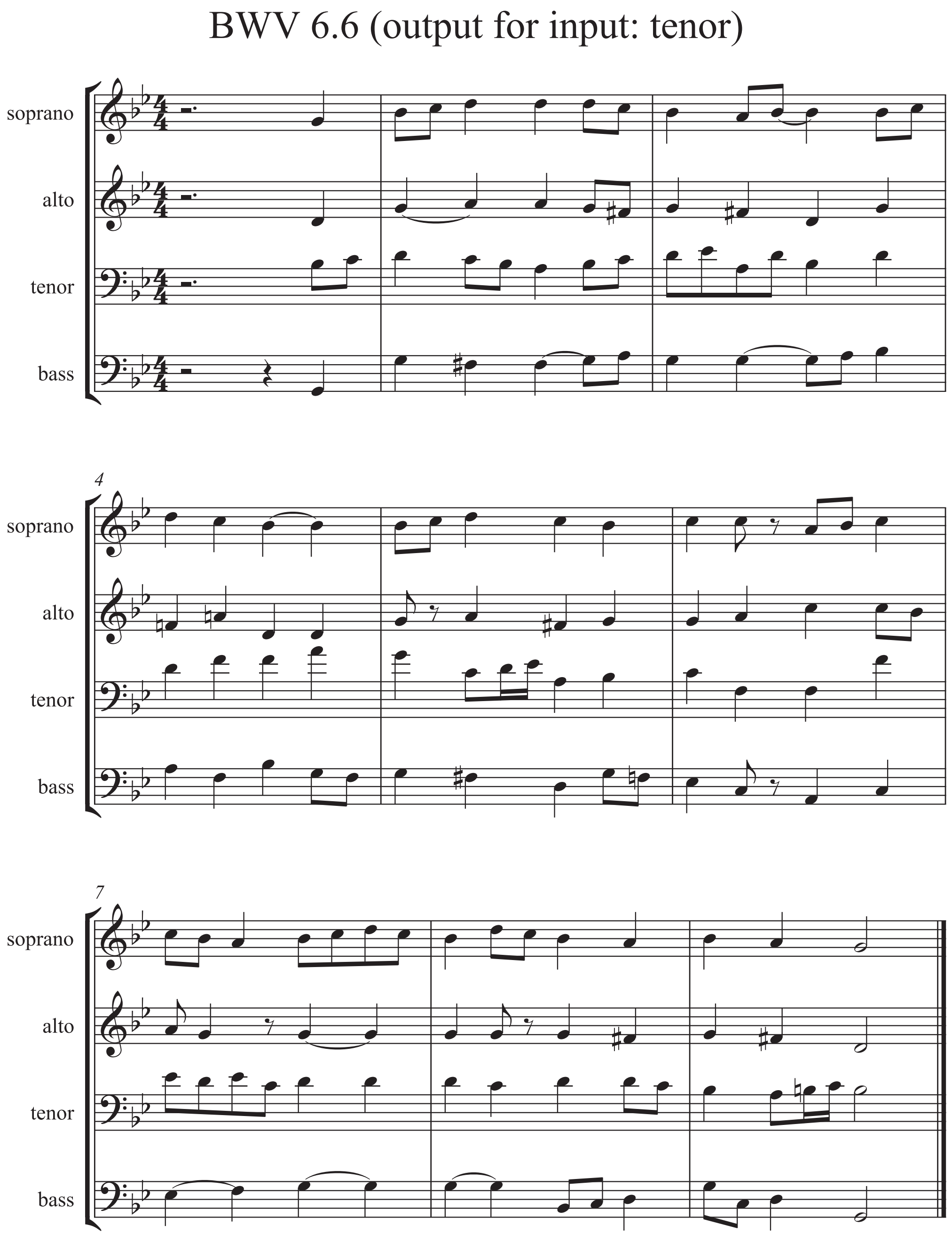


Figure 11: Output of EvoComposer on input the tenor line of Chorale BWV 6.6.

BWV 6.6 (output for input: alto)

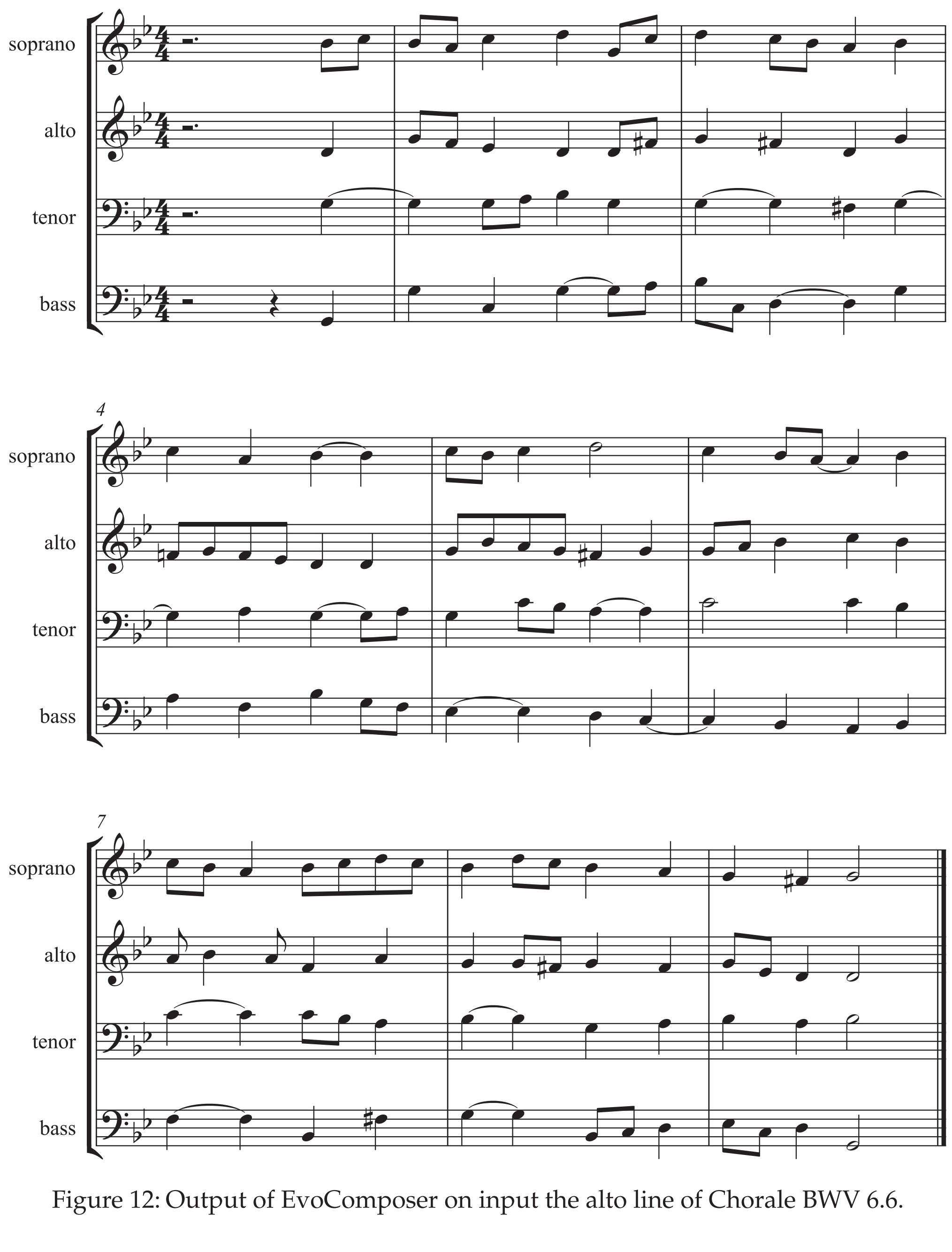


Figure 12: Output of EvoComposer on input the alto line of Chorale BWV 6.6.

BWV 6.6 (output for input: soprano)

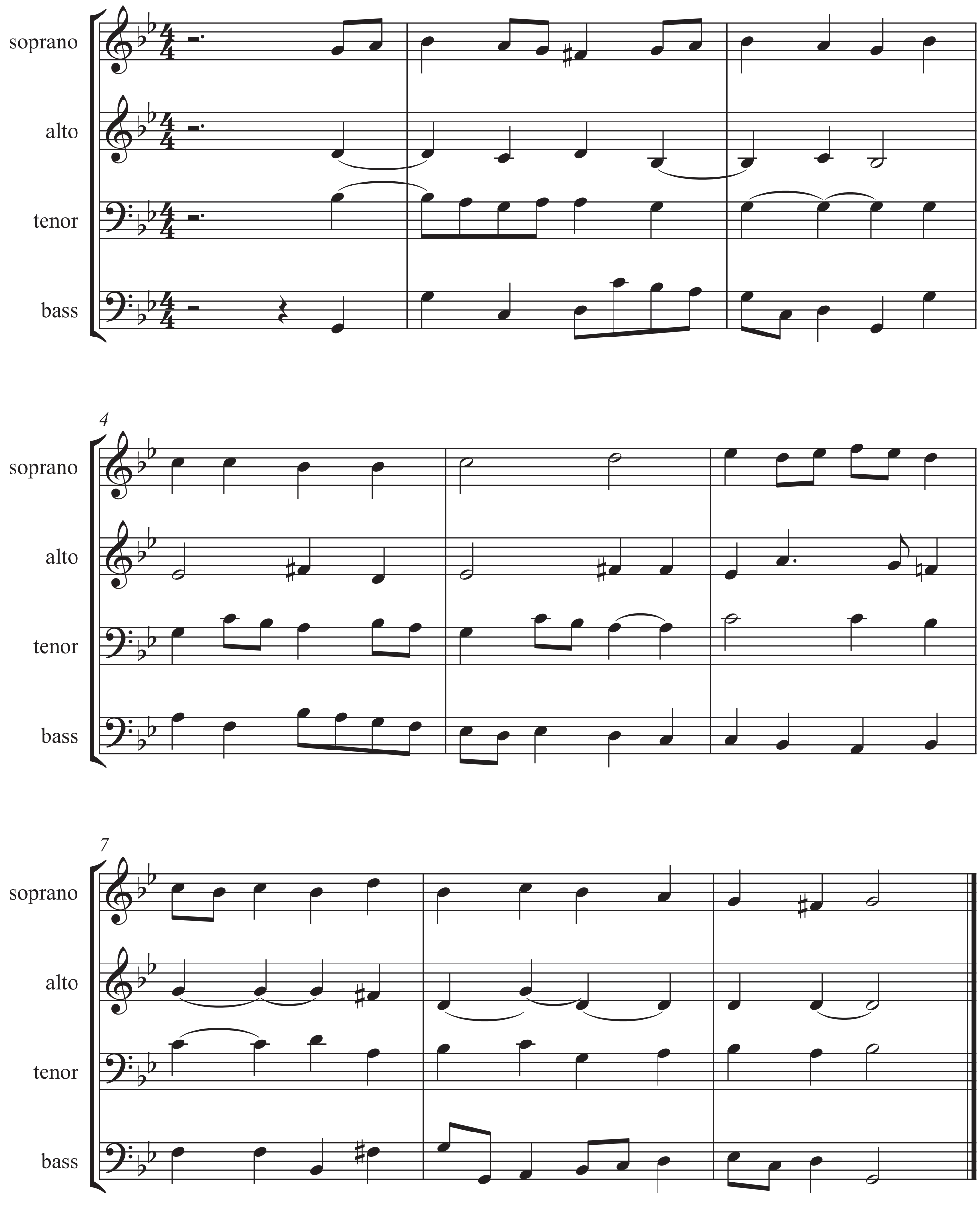


Figure 13: Output of EvoComposer on input the soprano line of Chorale BWV 6.6.