Combining local search and directed mutation in evolutionary approaches to 4-part harmony

Elia Pacioni[0000−0002−1557−4870] and Francisco Fernández De Vega[0000−0002−1086−1483]

Universidad de Extremadura, Av. Santa Teresa de Jornet, 38. 06800 Mérida, Spain
HES-SO Valais/Wallis, Sierre, Switzerland
eliapacioni@unex.es, elia.pacioni@hevs.ch, fcofdez@unex.es
https://www.unex.es

Abstract. Artificial Intelligence assisted music composition has gained popularity during the last decade, but still faces problems and difficulties. This paper approaches 4-part harmonization problem in the context of evolutionary computation, a topic that has been discussed for more than forty years but still represents an open challenge. This research proposes local search to improve directed mutation and guide the algorithm towards qualitatively better solutions. Human learning and Evolutionary Machine Teaching are also used: data from music conservatory students are exploited to guide the algorithm. The results show that local search significantly improves the quality of the mutation performed. Moreover, a series of longer runs are able to find free error scores.

Keywords: Directed Mutation · Local Search · Evolutionary Machine Teaching · Human Teaching · 4-part harmonization.

1 Introduction

Art and music have become integral parts of computational creativity, and their interest in the scientific community is growing. Specifically, a comprehensive search conducted through the Scopus database concerning the intersection of AI or EA with Music or Art, encompassing articles published in peer-reviewed journals and conferences from the years 2018 to 2023, reveals a discernible upward trend, as illustrated in Figure 1. Importantly, research pertaining to AI in relation to Music or Art comprises a total of 14,884 articles, whereas this figure is about 4,024 when considering EA in conjunction with Music or Art.

An analysis of artificial intelligence methods used in music composition has identified the most suitable techniques for each specific task. For example, neural networks are particularly effective in imitative systems, while Markov models excel in predicting musical notes based on previous ones. In addition, genetic algorithms (GAs) prove very suitable for generating chord progressions [1, 2], although completely satisfactory results in the field of evolutionary algorithms (EAs) for music composition have not yet been obtained, despite the numerous studies carried out over the years. The simulations performed, which often do

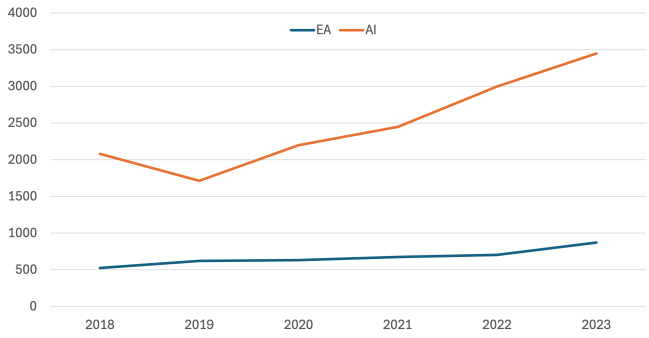

Fig. 1. The interest of the scientific community in evolutionary algorithms (EA, blue), artificial intelligence, (AI, orange) both in the area of art and music (Data collected from Scopus).

not reflect real scenarios, have almost always used limited and simplified problem sets.

4-part harmonization, also known as SATB (soprano, alto, tenor, and bass), is a widely used choral music technique. Bach is considered the starting point of this technique in the Baroque era, and its rules evolved and were polished during the classicism. Nowadays, it is a technique that every music student must learn. The most important aspects studied in this topic are chord progression, prohibition of certain dissonant combinations, melodic movements, and sound balance between different voices. Previous research involved music experts, but their constraints strongly limit the number of valid solutions in a large search space. In contrast, algorithms developed without experts risk oversimplification, producing more but lower-quality solutions. Thus, balancing expert input with coding complexity is essential. Sometimes, despite having experts, translating their knowledge into algorithms is challenging, leading to human integration at various stages. One example is interactive EAs for artistic evaluation. Yet, challenges remain, particularly for the problem we face: four-part harmonization, as simulations often use simplified problem sets. In the next section, we describe the problem, the more relevant approaches, and the difficulties in applying evolutionary techniques for exploring the search space, mainly due to the large number of chord dispositions that are available for harmonizing every single note within a score.

In this paper, we aim to improve the quality of results when evolutionary approaches are applied to the problem at hand. Specifically, a kind of directed mutation operator has been improved by adding a local search mechanism that helps to better explore the search space. As described below, this mechanism saves computing time while improving the quality of solutions found. To the best of our knowledge, this is the first time that results without any error have been found for a complex melody in a reasonable running time.

The paper is organized as follows: Section 2 presents a literature review on the 4-part harmonization problem. Section 3 outlines the methodological approaches employed and details the experimental configurations. Section 4 discusses the results, and Section 5 concludes the paper.

2 4-part harmonization and Evolutionary Algorithms

Computer science and the arts have long been interconnected, giving rise to numerous areas of interest over time, particularly computer-assisted music composition. Over the years, various AI-based approaches to music composition have been introduced, employing different techniques.

2.1 SATB harmonization problem

We focus on 4-part harmony, an introductory composition exercise for music students. Typically, a teacher provides a melody—a single voice, usually the soprano—and the student must construct the remaining three voices: alto, tenor, and bass. This task requires following rules that dictate permissibility, which have evolved over centuries to help create beautiful choral scores. For those interested, the Sharpmony project[1] lists up to 50 commonly used rules and exceptions.

A number of evolutionary approaches have been proposed during the last decades to solve this problem. However, many of these solutions simplify the real challenge, such as the one proposed by Horner and Goldberg in 1991 [3]. In 1994, McIntyre introduced a method that starts with a given melody and key to harmonize it. Unfortunately, harmonizing just nine notes required the efforts of thousands of individuals over hundreds of generations [4].

In 2014, Kaliakatsos et al. attempted to generate a chord progression by assigning scale degrees to each note, then selecting chords and individual voice notes based on these scale degrees [5]. They identified issues associated with the many rules needed to ensure the quality of the produced scores.

More recently, in 2017, Fernández presented a new genetic algorithm-based approach that successfully evolved an entire musical score. By applying only 11 rules during the evolutionary process, the resulting score had 10 errors, although it took 24 hours to achieve this outcome [6].

De Prisco et al. presented EvoComposer [7], an evolutionary algorithm for automatic 4-part harmonization. The system operates based on a multi-objective approach, optimizing both harmonization through appropriate chord selection and the melodic quality of vocal lines. To comply with harmonic and melodic rules, EvoComposer uses a hybrid evaluation function, which combines theoretical rules from classical music with weights derived from a statistical analysis of Bach chorales, ensuring stylistic and creative consistency. However, this approach does not have sufficient granularity to stabilize errors in the final score

[1] https://sharpmony.unex.es/index.php?r=tutorials%2Fharmonic-manual

while assuring the quality of the solution. Because it is based on a statistical approach related to Bach's chorales, it does not consider the evolution of the rules from the Baroque era to the present.

In 2024, Pacioni and Fernández presented a new version with over 50 rules and exceptions [8]. While the increased number of rules brings the system closer to practical reality, it also increases the computational time required for the genetic algorithm to find feasible solutions. To address this issue, the directed mutation was added to guide the system toward better solutions, and anticipatory partial fitness evaluation has been applied to enhance the algorithm's performance. In this context, directed mutation and the creation of synthetic models were also introduced [8], demonstrating how directed mutation can help improve the algorithm's convergence.

Also, in 2024, an approach related to Human Teaching/Learning and Evolutionary Machine Teaching was presented based on data collected from conservatory students using the Sharpmony [2] application [9]. This approach analyzes exercises produced by conservatory students: 13,000 exercises were analyzed, from which chord pairs were extracted and errors were calculated. The results were saved in a database indicating key, tonality, degree, and number and kind of errors in the pair. Additional information to distinguish between chords extracted from students' exercises and those new ones that may automatically be generated and explored by the algorithm was also added. This data is paramount since it allows us to extract all the knowledge teachers convey to students to guide the algorithm as one guides a student through the teaching process. Additionally, these data are crucial for reducing the search space and assisting the algorithm in converging to better quality solutions. The above described approach was able to find in a 24-hour run over student data obtained a musical score with 5 errors at the end of the evolutionary process [9].

This paper builds on these previous versions of the genetic algorithm described in [8] and [9], where the number of rules provided by experts have significantly increased, more than 50 rules and exceptions applied, thus drastically reducing the number of solutions available.

2.2 Directed Mutation

To fully understand the importance of using local search in these experiments, it is necessary to describe first the directed mutation operator. In the problem being addressed, the mutation occurs at the chord level. Specifically, during the mutation process, the algorithm randomly selects a chord and replaces it with a new chord, thus changing the notes and, probably, if the new chord degree is different from the initial one, the chord progression.

To improve this process and guide the algorithm toward better solutions, directed mutation was proposed. This operator uses information about the errors between pairs of chords and the chord positions with errors to determine which chord to mutate. Specifically, an array of the chord positions containing errors

[2] https://sharpmony.unex.es

within the pair is created during fitness evaluation before the mutation occurs. A random chord is extracted from this array and used for mutation. This new operator has been shown to improve the algorithm's convergence [8].

Building on this foundation, we experimented with an approach focused on evolutionary machine teaching. In this case, the search space during mutation was initially restricted to that used by Sharpmony students. When the algorithm performs a directed mutation, it will search for a new chord among those available in the database that has been previously used by students. This approach has further improved the evolutionary process [9].

Although directed mutation with human teaching/learning and evolutionary machine teaching improved previous approaches, it does not guarantee that each mutation will reduce the errors in the pairs undergoing mutation. Therefore, there is still room for improvement, particularly through the use of local search.

2.3 Local search

In the context of genetic algorithms, hybridization through local search techniques has given rise to memetic algorithms (MA) [10]. Moscato, the creator of the term memetic algorithms [10], defines them as an inspiration to Dawkins' concept [13] of memes, in which individuals adapt through imitation and neighborhood exploration, hence local search.

Local search can be an important component in genetic algorithms and, more generally, complex combinatorial problems [11, 12]. Its importance derives from its ability to explore a specific area of the solution space, improving the quality of the proposed solution. This approach emphasizes the use of problem-specific knowledge to make research more effective than classical evolutionary approaches.

Moscato and Cotta also theorized hybrid algorithms with periodic local search phases and global search phases to prevent premature convergence, emphasizing the importance of population heterogeneity [14].

Local search strategies used in MA can vary depending on the type of problem and the characteristics of the solution to be optimized. The local improvement operators typically adopted include mutations with changes introduced with problem-specific knowledge and iterative improvement methods that progressively optimize a solution until an acceptable level of quality is reached or a local maximum is met. In addition, many implementations of MA include local search adaptation mechanisms, in which operators dynamically change based on real-time results, allowing the algorithm to respond to changes in the solution landscape in an agile and flexible manner [13].

MA finds applications in many fields, from engineering to medical, and via multi-objective problems [13].

2.4 Beyond the state of the art: challenges and limitations

The state-of-the-art developments we have explored demonstrate a transition from small-scale experiments that lack real-world relevance to research that in-

creasingly aligns with the 4-part harmonization techniques employed by both students and teachers. However, the approaches presented thus far encounter two significant limitations: (i) the algorithm's convergence is slow and does not guarantee a final music sheet with zero errors, and (ii) the algorithm's runtime is too lengthy to render the system practical for production applications.

By employing a human teaching and learning approach to reduce the solution space, we have achieved a notable decrease in execution time; nonetheless, it remains a challenge.

Therefore, it is crucial to implement techniques that facilitate algorithm convergence without adding excessive overhead. Achieving zero-error scores in fewer generations would significantly decrease the computational time required, thus enhancing the algorithm's applicability.

3 Methodology: applying local search to reduce errors between chord pairs

The methodology proposed in this paper exploits local search within the directed mutation operator to improve the algorithm's convergence. Local search is performed on the entire database of chord pairs produced by the evolutionary machine teaching process on student data. Thus, mutation is used to fix errors between chord pairs. But let us first describe the problem, the main issues we face, and how local search can be added to the directed mutation operator.

3.1 The problem

To fully grasp the problem at hand, it is crucial to understand the genetic algorithm used for evolving SATB musical scores. The starting point is a given melody (the one we want to harmonize) of a series of bars —eight in the example illustrated in Figure 2—which is the starting point for the evolutionary procedure.

First of all, two evolutionary stages are employed, the first one for deciding scale degrees to be applied to every note provided in the melody (for instance degrees II, V, or VII for a D note in C Key), and the second one to decide the distribution of chord notes in the remaining voices, once the scale degree to be applied is known (if II degree is to be applied, and D is in the melody, we must assure that F and A are present in the other voices, if a triad is applied).

Thus, in the first stage, an individual is represented by a sequence of chords (I, II, ..., VII), with one degree per note in the melody, and this initial genetic algorithm is employed to determine an appropriate sequence of chord degrees to emulate the composition style students practice. A given melody can be considered to belong to a single key, C Major, for instance, or instead, can include modulations, which allow to go from one key to another one, such as going from C Major to A minor or from C Major to G Major. Yet, in what follows, we consider that modulations are not allowed, and we will restrict the search for solutions to a single key: C Major.

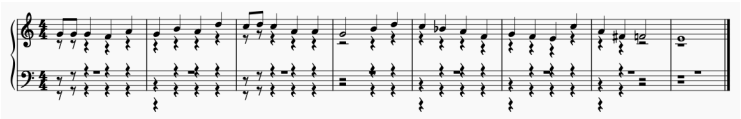

Fig. 2. Melody provided to the algorithm to perform harmonization.

Although many kinds of chords may be used, according to harmony rules, we are only considering here diatonic triads and seventh chords, augmented sixth (italian, french, german and neapolitan chords) as well as secondary dominants.

A second genetic algorithm is initiated afterward to evolve the complete musical score, where the chromosome includes all the notes assigned to all the voices along the score. This involves assigning notes to each voice that aligns with the previously selected chord degrees—for example, assigning C, E, and G to form a I degree triad in C major. Figure 3 shows the chromosome structure of the two genetic algorithms and the correlation between them. Finally, when chord notes are distributed along the remaining voices according to degrees previously evolved, the whole score has to be checked using the fitness function that embodies all the harmony rules usually applied, which includes 50 different errors to be checked, such as:

- Parallel 5th.
- Parallel 8th.
- Overlapping voices.
- ...
- (up to 50 rules, as described above)

We must bear in mind that no specific rules for the degrees movements are included within this fitness function in the second stage, so we are not checking cadences or phrasing endings here. Yet, the first stage of the evolutionary process is in charge of finding an appropriate chord progression, which is then employed by the second one.

Some of the available rules establish if a given set of chords are available or not, such as secondary dominants, seventh chords, neapolitan chords, sixth chords, etc. Thus, when new kinds of chords are added, the search space significantly grows, and rules to be checked also grow, making the search for solutions harder. Moreover, the higher the number of rules to be checked, the longer the time required to check a pair of chords' errors: 22.5 seconds are required for analyzing a single pair of chords when all the rules are applied, according to Sharpmony implementation of the fitness function, implemented in Common Lisp and run over SBCL[3], using the hardware described below. This means that analyzing a single chromosome -fitness function applied to an exercise- may take around 10 minutes.

To mitigate this challenge, we attempted to anticipate the computing of partial fitness values corresponding to potential errors in successive chords and

[3] https://www.sbcl.org

Fig. 3. Above is represented an individual's chromosome from the first genetic algorithm, considering C Major Key. Shown at the bottom is the chromosome of the individual from the second genetic algorithm.

store them in a database. This is possible by analyzing and storing a pair of chords that students apply within their solutions. Once this is available, checking the number of errors of a given pair of chord errors will take the time required to access the database: 0.001 milliseconds instead of 22.5 seconds to compute all the rules.

This strategy would allow us to assess which pairs of chords can be used with a given pair of consecutive melody notes and select those that minimize errors. Yet, there is a difficulty: all possible chord configurations must be precomputed, and then for every possible pair of consecutive chords, errors detected must be stored in the database. As described in [9] the number of possible chord pairs are beyond the available computing capabilities, thus we decided to only precompute those pairs ever employed within the 13,000 students exercises that we have already analyzed. This allowed us to initially restrict our search space to 67,240 pairs of chords instead of the 8,151,025 available in a single key, as described in [8, 9].

3.2 Adding local search to directed mutation

In Pacioni and Fernández 2024 [8], the directed mutation is applied, which restricts mutation to only consecutive chords where errors have been found. The idea is to try to fix errors, while maintaining untouched parts of the chromosome that are correct. In any case, given that crossover is also applied, any chromosome position may change, thus allowing exploration of other areas of the search space. Yet, we try here to improve the mutation operator by adding a local search process: instead of randomly choosing a new chord for a given position, we try to analyze the neighborhood of a chord -chords whose notes are in close positions in the score and try to see which one is better when considering errors that may appear when introducing that new chord.

In practice, each time the mutation is applied, the algorithm creates an array of positions with errors, and from those positions, a chord is chosen to mutate. Two modes of local searches are proposed: (i) the key and degree of the original chord are retained, and a different chord notes disposition is searched in the database; (ii) the original key is retained, and the algorithm can choose the most appropriate chord and degree from those belonging to the same key. In both cases, the main goal is to minimize the number of errors. Figure 4 illustrates the process carried out by local search to select the best chord during mutation

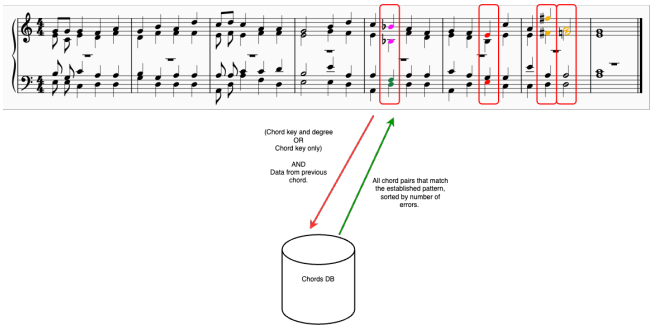

Fig. 4. Local search process: exploring nearby chords in the precomputed chord pairs errors

3.3 Tests Performed

To apply the proposed methodology and compare the current results with those previously obtained, 35 runs were conducted with a population of 8 individuals. For the first stage, when the progression of chords is evolved, the evolutionary algorithm was run for 50 generations. For the second stage, when chord notes are distributed among voices, 10 generations were computed again with 8 individuals in the population. A roulette selection method and directed mutation were used in the second stage. A 50% crossover probability is applied in both stages. Initially, the database includes only the chord pairs derived from the study of student solution space. A total of 67,240 chord pairs are available, which can be extended by the algorithm along the search process: when no suitable chord pair is found, the system can explore new not seen chord pairs, evaluate the quality, and store the information for future search.

All experiments are conducted on a Dell M1000e cluster consisting of 15 M600/M610 blades. The processors in use are Intel Xeon models, specifically E5506, E5507, E5640, E5520, and X5670. The cluster offers a combined total of 136 cores and 376 GB of RAM. Each test run is performed using a virtual machine configured with 8 virtual CPUs and 8 GB of RAM running the Debian 12.

4 Experiments and results: the impact of local search

Experiments are conducted using a database derived from the analysis of student exercises as described above. Together with the student pairs already stored, the algorithm can generate new pairs when no suitable one is found among students' pairs. Yet students ones are prioritized. The search for a new chord, when a mutation operator acts, works as follows: We need a new chord in a given position, but the new chord must feature the melody note already available, that cannot be changed. Moreover, this new chord, when inserted, will be analyzed in its context, and we want it to feature the smallest number of errors possible. Thus, we also consider for the search the previous chord, and try to search for pairs in the database that include as the first chord that previous one, and is followed by another chord with the given note in the soprano (melody). Among those available, we will take the pair with a smaller number of errors. We must remind that this info is available for every pair of precomputed chords. If no pair is found, the system will try to generate and compute pair of chords -much slower process- to decide which chord to use in the mutation process, and this computed information of new pairs will be saved for future search.

Figure 5 shows the process for generating and storing a new chord pair. Also, when the crossover is applied, new chord pairs could be generated since the point where the crossover occurs could combine two chords that are not in the student database as a pair; again, the new information enriches the database. This section examines the evolution of the database, the resulting solution space, and the algorithm convergence based on the configurations used.

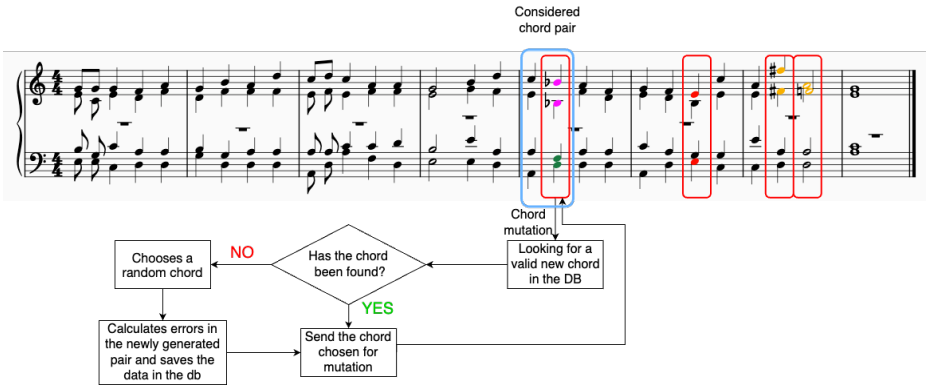

Fig. 5. Process to create and save a new chord pair.

4.1 Updated solution space

During the evolutionary process, the genetic algorithm concatenates the chords by creating musical scores containing pairs in the database and, in some cases,

combines the chords, generating new pairs. In this case, the new pairs are added to the database together with the number of errors that the fitness function has computed, and a specific label that allows us to distinguish them from those coming from students exercises. These labels allow us to analyze the database resulting from the experiments. At the end of the experiments, the database consists of 107,774 chord pairs, of which 40,534 are new ones produced by the EA, and 67,240 were initially collected from the students' exercises, representing 37.61% and 62.39%, respectively. This database will be employed in future experiments, so we foresee it will continue growing and thus allowing to speedup experiments in the future, given that a larger number of pairs are already precomputed.

Figure 6 shows the distribution of chords with the relative number of errors. We can see that students produced about 43% (about 29,000) of chord pairs without errors, while the EA only 24%. Specifically, the EA added 9,813 new items to the database without errors, while the remainder contained between 1 and 13 errors. Notably, most introduced pairs contain less than 4 errors, corresponding to 91.96% (about 37,000). In any case, we see that students are capable of better selecting chord pairs, and this is one of the reasons behind the human teaching inspired approach we follow.

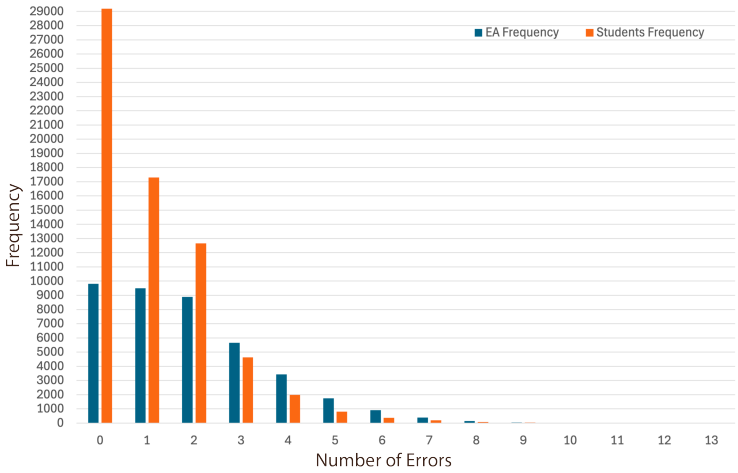

Fig. 6. Errors distribution:

Figure 7 compares the error types committed by the EA and the students. Among students, we can see that the most common errors are (i) *wrong note duplications*, (ii) *missing third in a chord*, and (iii) *wrong chord or chord not in the key*. While EA mainly makes errors in (i) *direct fifths or octaves*, (ii) *overlapping voices*, (iii) *incorrect resolution 7th/9th*. In general, EA has a more uniform distribution of errors than students, except for *direct fifths or octaves*, constitut-

ing 20.8% of EA errors. Interestingly, students fail more frequently missing the third in a chord or duplicating an incorrect note.

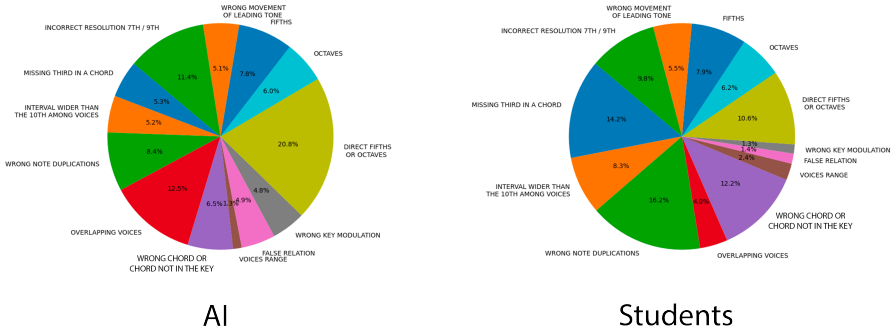

Fig. 7. Error types comparison between students and AI

4.2 Runs

Figure 8 shows the convergence of the algorithm in three different configurations: (i) without local search; (ii) with local search, maintaining chord key and degree; (iii) with local search and only chord key.

At generation 0, the values are quite similar among the different versions of the algorithm. Versions (i) and (ii) show almost equal convergence, with version (ii) slightly better. These negligible differences may be understood if we remember that we do not allow the algorithm to change the chord degree, which means that the only possibility is to change how notes are distributed among voices, so local search is not free enough for a proper search.

Nevertheless, version (iii) shows big differences in the convergence process: when we allow to change chord degree, freedom provided allow the system to find much better solutions. Consequently, local search with the right degree of freedom significantly helps the algorithm to converge to better solutions. In this case, local search is not an overhead because the chord pairs can be sorted by the number of errors directly in the database query, with minimal computational cost. We must also bear in mind that the system allows to search for previously unseen pairs of new chords.

Finally, we decided to launch longer run experiments - 10 executions - in order to foresee the potential of the algorithm and compare it to previous results obtained along 24 hours of evolution. The experiment was thus performed with the same configurations as described above, but over 50 generations (estimated number of generations requiring 24 hours).

Surprisingly, analyzing the execution time for the 10 runs, we get that the average time is 13 hours and 30 minutes, with a standard deviation of 5 hours.

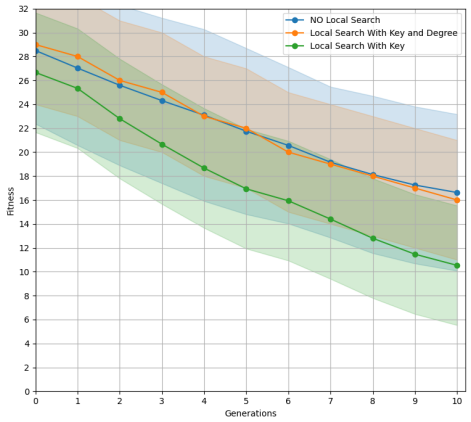

Fig. 8. The plot compares convergence of best fitness average values among the 3 versions of the algorithm over 35 runs

The best time is about 6 hours with one interesting individual who stopped execution at generation number #14, when an error-free solution was found (see Figure 9). Interestingly, this solution applies secondary dominants from C Major, such as chord #19, that includes I* (C-E-G-Bb) to go to IV, which is quite a nice solution to harmonize Bb in the soprano without going out of C major. Similarly, F# in the soprano note is harmonized using a II* triad distributed in voices as F#-D-A-F that performs a chromatic movement towards V7, F-D-B-G, and then to I, E-C-G-C, in a proper final V7-I perfect cadence. Moreover, the lead tone in the tenor voice indirectly resolve to the alto, which is in interesting resource to solve the mandatory resolution of lead tone.

Figure 10 shows the average best fitness over the 10 runs. The graph shows the convergence curve and the standard deviation trend. We can see how, even though we start with more than 30 errors, we arrive at an average fitness of 1.8 with a standard deviation of 1.77. Specifically, 4 runs produce individuals with 0 errors, the rest produce individuals with errors between 0 and 5, respectively: 5,2,3,1,4,3. To the best of our knowledge, this is the first time that an error-free 4-part harmony has been obtained by evolutionary approaches for a complex melody.

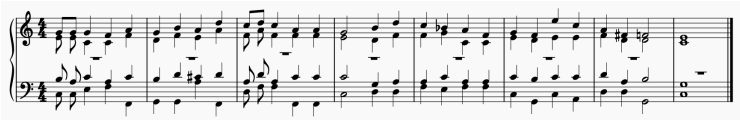

Fig. 9. Error-free harmony produced by the EA.

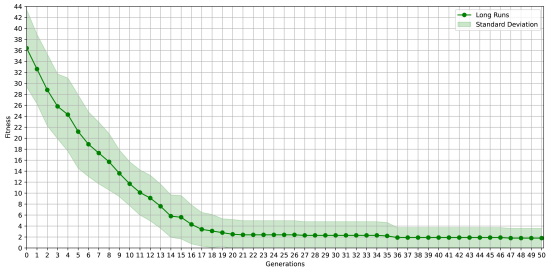

Fig. 10. Fitness convergence for long execution.

5 Conclusions

This paper describes an improvement to the 4-part harmonization problem by adding local search to directed mutation. The approach used is based on the evolutionary machine teaching principles that exploit the idea of human teaching/learning. By using Sharpmony students' data, we narrow the search space to the area exploited by students and thus drive the algorithm toward better solutions. Specifically, the solution space becomes hybrid, with student data combined with the new chord pairs created by the EA. Specifically, the starting database contained 67,240 students' chord pairs, while it now has 107,774. Thus, the EA created 40,534, of which 9,813 were error-free. Local search is included during the mutation step: the experiments show that local search within the original chord key produces better solutions than the version without local search or with too stringent criteria. We also launched longer runs, where we obtained an average fitness of 1.8 and 4 runs found error-free solutions. To the best of our knowledge, this is the first time that a proper solution is found for a complex melody when 50 harmony rules and exceptions are applied.

In future work, we plan to add local search during the initialization step, when individuals are created in the initial population using the progression provided by the first evolutionary stage described above.

In any case, although execution times are still about 13 hours -much better when compared to 24 hours required for previous approach to find a reasonable solution, the solution presented in this study is a first step toward generating error-free music sheets in reasonable time runs.

6 Acknowledgements

We acknowledge support from the Spanish Ministry of Economy and Competitiveness under projects PID2020-115570GB-C21 and PID2023-147409NB-C22 funded by MCIN/AEI/10.13039/501100011033. Junta de Extremadura under project GR15068.

References

1. Omar Lopez-Rincon, Oleg Starostenko and Gerardo Ayala-San Martı'n: Algoritmic Music Composition Based on Artificial Intelligence: A Survey. In: International Conference on Electronics, Communications and Computers (CONIELECOMP), 2018, pp. 187-193 https://doi.org/10.1109/CONIELECOMP.2018.8327197
2. Hung Liu, Chien and Kang Ting and Chuan: Computational Intelligence in Music Composition: A Survey. IEEE Transactions on Emerging Topics in Computational Intelligence **1**(2), 2-15 (2017)
3. Horner, A., and Goldberg, D. E. Genetic algorithms and computer-assisted music composition. In: International Conference on Mathematics and Computing, 1991, pp. 437-441.
4. McIntyre, R. A. Bach in a box: The evolution of four part baroque harmony using the genetic algorithm. In Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence, Proceedings of the First IEEE Conference, 1994, pp. 852-857 https://doi.org/10.1109/ICEC.1994.349943
5. Kaliakatsos-Papakostas, M., and Cambouropoulos, E.: Probabilistic harmonization with fixed intermediate chord constraints. In ICMC, 2014, https://doi.org/10.13140/2.1.3079.5526
6. Francisco Fernández de Vega: Revisiting the 4-part harmonization problem with GAs: A critical review and proposals for improving. In: IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 2017, pp. 1271-1278, https://doi.org/10.1109/CEC.2017.7969451.
7. R. De Prisco, G. Zaccagnino, R. Zaccagnino; EvoComposer: An Evolutionary Algorithm for 4-Voice Music Compositions. Evol Comput 2020; 28 (3): 489–530. doi: https://doi.org/10.1162/evco_a_00265
8. Elia Pacioni, Francisco Fernández de Vega. On the impact of directed mutation applied to Evolutionary 4-part harmony models. In: Artificial Intelligence in Music, Sound, Art and Design. EvoMUSART (EvoStar2024), https://doi.org/10.1007/978-3-031-56992-0_20.
9. Elia Pacioni, Francisco Fernández de Vega. Paving the way towards evolutionary machine teaching: an application to 4-part harmony. In: International Conference, Évolution Artificielle, EA2024.
10. Moscato, Pablo. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech concurrent computation program, C3P Report 826.1989 (1989): 37.
11. Moscato, P., Cotta, C. (2003). A Gentle Introduction to Memetic Algorithms. In: Glover, F., Kochenberger, G.A. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 57. Springer, Boston, MA. https://doi.org/10.1007/0-306-48056-5_5
12. Moscato, P., Cotta, C. (2010). A Modern Introduction to Memetic Algorithms. In: Gendreau, M., Potvin, JY. (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 146. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1665-5_6
13. Neri, F., Cotta, C., Memetic algorithms and memetic computing optimization: A literature review, Swarm and Evolutionary Computation, Volume 2, 2012, Pages 1-14, https://doi.org/10.1016/j.swevo.2011.11.003
14. Cotta, C., Gallardo, J.n Mathieson, L., Moscato, P. (2016). Memetic Algorithms: A Contemporary Introduction. In book: Wiley Encyclopedia of Electrical and Electronics Engineering (pp.1-15)Publisher: John Wiley & Sons https://doi.org/10.1002/047134608X.W8330.