Evolutionary Approach to Melodic Line Harmonization

Jan Mycka[1 [0000−0003−2250−021X]], Adam Żychowski[1 [0000−0003−0026−5183]], and Jacek Mańdziuk[1,2 (✉) [0000−0003−0947−028X]]

[1] Warsaw University of Technology, Warsaw, Poland
[2] AGH University of Science and Technology, Krakow, Poland
jkj.mycka@gmail.com, a.zychowski@mini.pw.edu.pl, mandziuk@mini.pw.edu.pl

Abstract. The paper presents a novel evolutionary algorithm (EA) for melodic line harmonization (MLH) - one of the fundamental tasks in music composition. The proposed method solves MLH by means of a carefully constructed fitness function (FF) that reflects theoretical music laws, and dedicated evolutionary operators. A modular design of the FF makes the method flexible and easily extensible. The paper provides a detailed analysis of technical EA implementation, its parameterization, and experimental evaluation. A comprehensive study proves the algorithm's efficacy and shows that constructed harmonizations are not only technically correct (in line with music theory) but also *nice to listen to*, i.e. they fulfill aesthetic requirements, as well. The latter aspect is verified and rated by a music expert - a harmony teacher.

Keywords: evolutionary algorithm · harmonization · music composition

1 Introduction

The majority of real-life optimization problems are associated with engineering, however, certain aspects of creative activities, such as painting, music composition, poetry, or film making, can be modeled as optimization problems [4, 19], as well. In this paper, one such task – the melodic line harmonization is considered.

The melodic line harmonization is a part of the process of composing music and is about determining the musically appropriate chord accompaniment for a given melody. It is a creative process that requires intuition and experience of the musician, although, the music theory defines certain strict constraints and rules which the composed music should follow in order to sound well [21]. In this perspective, melodic line harmonization can be treated as an optimization problem with maximizing the number of fulfilled constraints.

Evolutionary Algorithms (EAs), thanks to their effectiveness, are widely applied to various practical problems [5, 11, 17, 27]. This paper shows that EAs can also be successfully adapted to the field of art and create formally correct, well-structured, and musically aesthetic melody harmonizations.

2 Related work

Algorithmic music composition is a well-studied area of research with various computational intelligence methods proposed in the literature [13, 25]. There are several research paths in this domain and researchers focus on various aspects of music generation, for instance, style transfer [9], imitating a particular composer (e.g. F. Chopin [15, 16]), real-time music accompaniment [12], timbre, pitch, rhythm, chord [14]. In this paper, we consider the problem of melodic line harmonization which is an essential part of the music composition process. The definition and details of the examined problem are presented in Section 2.1.

The most common approach to solving this task is learning harmonizations based on existing melody lines using neural networks [6, 8, 10], which requires a set of training data and is usually limited to a particular genre or music style, e.g. Bach chorales [8]. Music composition can also be approached with Markov chains [3, 18, 26] or evolutionary algorithms [7, 20, 22]. Evolutionary approaches propose various representations of melodic line and fitness function definitions to assess evolving solutions. Moreover, in [7] a multiobjective genetic algorithm is constructed which, for a given melody, generates a set of harmonic functions without adding new melodic lines.

Due to slightly different problem definitions and the lack of well-established benchmarks, making a direct comparison between methods is usually difficult. Thus, the evaluation process is often performed by human experts who rate the obtained results (music pieces). This approach is also taken in this paper.

2.1 Melodic Line Harmonization

Harmonization of a melodic line is one of the fundamental tasks in music. The input data in a harmonization problem is one melodic line, and the product of harmonization is usually four melodic lines (voices): soprano (the highest), alto, tenor, bass (the lowest). A given (input) melodic line could be also accompanied by harmonic functions which are added to every or almost every note in that line. These functions determine which notes can be included in the chord formed across all four lines (vertically).

Harmonization of a melodic line depends not only on the composer's creativity but also on various theoretical rules derived from music theory. These rules regulate (1) the form of individual melodic lines, (2) chord's construction, and (3) how successive chords should be connected to each other.

The problem considered in this paper is a harmonization of a soprano line, with harmonic functions added to each note. The solution is created based on a selected set of theoretical rules for melodic line harmonization.

2.2 Contribution

The main contribution of the paper includes: (1) a novel evolutionary algorithm capable of designing correct melodic line harmonizations; (2) a specially designed fitness function that reflects theoretical music rules and can be easily tuned

toward certain aspects of the output harmonization; (3) an extensive evaluation of the proposed method which shows its quality and robustness; (4) a detailed analysis of the algorithm's performance and parameterization.

3 Evolutionary Harmonization

3.1 The search space and the initial population

Not every note can be used in a created chord. Harmonic functions define which notes fit into a chord and which do not. Harmonizations containing notes in chords that do not correspond to the required functions are incorrect.

After receiving the input (soprano line with harmonic functions), for each unique function, a set of all possible chords that fulfill that function is created. Created harmonizations are, therefore, not generated from individual notes but from the whole chords. The above rules significantly narrow down the search space, however, due to still many possible arrangements of notes in each chord, the number of potential solutions is still too large (between 3^l - 7^l, where l is a harmonization length) to evaluate all of them. Examples of created chords for one of the functions are shown in Fig. 1.

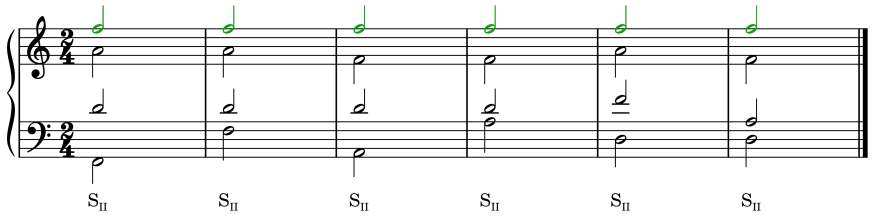


Fig. 1: Various chords for function $\mathbf{S}_{\mathrm{II}}$ with a fixed (green) note in soprano.

Individuals are represented as 4 sequences of notes, one per each harmonized voice. The chord is formed by the notes located across all four voices (vertically). Individuals in the initial population are created randomly. Soprano notes are completed to a randomly selected chord satisfying the following two conditions:

(*) the chord corresponds to the function assigned to the completed note,
(**) the note given in the input voice is located in the chord in the same voice.

3.2 Next generation population

After the generation of the initial population, the EA is run for a predefined number of n generations. In each generation, first s_e *elite* (i.e. currently best) individuals are promoted from the previous generation without any adjustments, so as to ensure that the best individuals found in the entire run of the algorithm

will not be lost. The rest of the population is generated by means of selection procedure and genetic operators (mutation and crossover), following Algorithm 1.

```
1  GenerateNewPopulation (P)
2      CalculateFitnessValues(P)  // calculates fitness of each individual
3      P_new ← GetElite(P, s_e)    // population of new individuals
4      while |P_new| < |P| do
5          c_1 ← Selection(P)
6          if rand([0,1]) < p_c then // crossover
7              c_2 ← Selection(P)
8              c_new = Crossover(c_1, c_2)
9          else
10             c_new ← c_1
11         end
12         c_new ← Mutation(c_new)
13         P_new = P_new ∪ {c_new}
14     end
15     return P_new
```

Algorithm 1: Next generation population procedure.

3.3 Selection method

Selection of individuals from the population is performed in a t_s-tournament with a roulette, i.e. first t_s individuals are uniformly sampled with replacement to participate in the tournament. The drawn individuals are sorted from best to worst according to their score. Let's denote by $c_i,\ i = 1, \ldots, t_s$ the i-th ranked individual. The chance of winning the tournament by c_i is calculated as follows:

$$p(c_i) = \begin{cases} p_s & \text{if } i = 1 \\ (1 - \sum_{j=1}^{i-1} p(c_j)) \cdot p_s & \text{if } 1 \ < \ i \ < t_s \\ (1 - \sum_{j=1}^{i-1} p(c_j)) & \text{if } i = t_s \end{cases} \tag{1}$$

where $p_s \geq 0.5$ is the so-called *selection pressure.*

3.4 Mutation

Generated harmonizations are built using the whole chords, rather than individual notes. For this reason, mutations are also performed on the entire chords and each chord in the harmonization is mutated with the same probability equal to $\frac{p_m}{l}$, where l is the length of the harmonization (the number of notes in the input melodic line) and p_m is mutation coefficient. Mutation of a chord consists in replacing it with another randomly selected chord that satisfies conditions (*)-(**).

3.5 Crossover

Crossover is performed with probability p_c. Two crossover methods are proposed and tested: the classic operator and the one-point operator. Analogously to mutation, the crossover is performed using whole chords rather than individual notes. Both crossover operators are presented in Algorithm 2, where $c[i], i = 1, \ldots, l$ is the chord located at position i in a harmonization of length l.

```
1  Crossover_1 (c_1, c_2)
2      for i ∈ [1, ..., l] do
3          if rand([0,1]) < 0.5 then
4              c[i] = c_1[i]
5          else
6              c[i] = c_2[i]
7          end
8      end
9      return c
```

```
1  Crossover_2 (c_1, c_2)
2      k ← rand(1, ..., l)
3      for i ∈ [1, ..., l] do
4          if i < k then
5              c[i] = c_1[i]
6          else
7              c[i] = c_2[i]
8          end
9      end
10     return c
```

Algorithm 2: Crossover: left - classic method, right - one-point method.

3.6 Fitness function

The fitness function is based on music theory and is composed of 22 rules of harmonization, taken from a harmony textbook [24]. Similar rules can be found in [2, 23]. Each rule is assigned a weight (positive or negative) that affects the final score of the generated harmonization. Examples of violations of three of these rules are shown in Fig. 2. A detailed description and implementation of all rules can be found in a project repository [1].

The fitness function can be divided into 3 main modules:

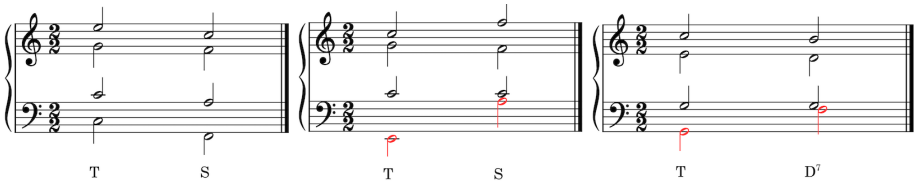
1. Strong constraints C_s (strong penalty terms) - stemming from the rules that must be strictly met in the created harmonization to be considered correct.
2. Weak constraints C_w (weak penalty terms) - derived from rules that do not have to be strictly satisfied in the created harmonization, but their non-fulfillment lowers the harmonization assessment.
3. Aesthetic value V_a (reward terms) - the rules specifying chord arrangements or connections between chords that improve the harmonization sound.

The fitness function f_t for individual c has the following form:

$$\begin{gathered} f_t(c) = V_a + C_w + (p \cdot t)C_s, \\ C_s = \sum_{i=1}^{m_s} \phi_i(c), \quad C_w = \sum_{j=1}^{m_w} \chi_j(c), \quad V_a = \sum_{k=1}^{m_a} \psi_k(c), \end{gathered} \tag{2}$$

where $\phi_i(c) \leq 0$ is the penalty for not fulfilling strong constraint $i, i = 1, \ldots, m_s$, $\chi_j(c) \leq 0$ is the penalty for not fulfilling weak constraint $j, j = 1, \ldots, m_w$,

$\psi_k(c) \geq 0$ is the reward associated with the rule $k, k = 1, \ldots, m_a$, ($m_s = 9$, $m_w = 9$, $m_a = 4$), $t \leq n$ is the generation number, and p is a constant parameter. Please note that during the evolution, the fitness function value is calculated for each individual regardless of the fulfillment of the strong constraints. These constraints, however, define the correctness of each individual.



(a) Strong constraint: At least one voice has to move in different direction than other voices.

(b) Weak constraint: A maximal interval the bass can take in two consecutive moves is tenth.

(c) Weak constraint: There should not be septim interval between two consecutive notes in one voice.

Fig. 2: Examples of rules violations.

4 Experimental results

Since there are no standard benchmarks for the considered problem we decided to use a set of exercises from the harmony textbook [24] as a test set (similar exercises can be found in other harmony textbooks, e.g. [2, 23]). The selected problems were divided into 3 groups based on their complexity and length:

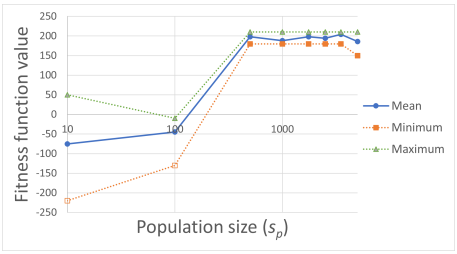
1. long examples (about 20 chords), using only basic functions,
2. short examples (about 10 chords), with more complicated functions,
3. long examples (about 20 chords), with more complicated functions.

4.1 Algorithm parametrization

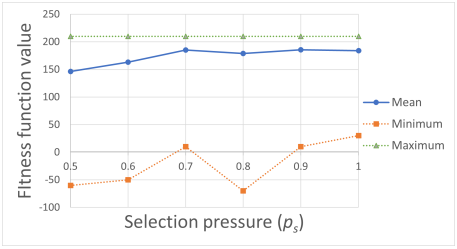
The choice of the evolutionary parameters is crucial for the algorithm performance. The values of the following parameters were selected based on preliminary tests: population size (s_p), tournament size (t_s), elite size (s_e), selection pressure (p_s), mutation coefficient (p_m), crossover method and crossover probability (p_c), number of generations (n).

The following baseline values were selected: $s_p = 1000$, $t_s = 4$, $s_e = 3$, $p_s = 0.7$, $p_m = 1$, classic crossover with $p_c = 0.8$, $n = 5000$. Individual parameters were then optimized (with the remaining parameters frozen) to select the best values for each of them. The tests were run on 3 different examples, one from each group. These examples were different from the ones used as the test set. Each test was repeated 5 times with different seed values for the random number generator.

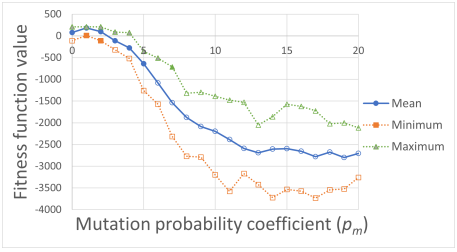
Population size (s_p). The following population sizes were tested: 10, 100, 500, 1000, 1750, 2500, 3500, 5000. As expected, for smaller population sizes, the algorithm performed noticeably worse because the solution space was not searched sufficiently. For larger values (above 1000), the results were not substantially different from each other. Results for an example from the third group are presented in Fig. 3a. The resulting size of the population was chosen as 1000.



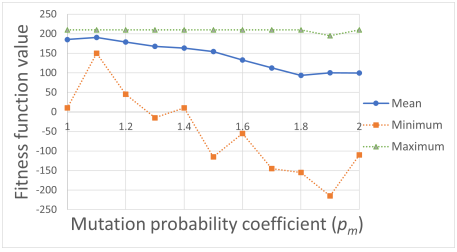
(a) Population sizes (logarithmic scale).



(b) Selection pressures.



(c) Mutation coefficients - crude estimation.



(d) Mutation coefficients - fine-tuning.

Fig. 3: Parameter tuning averaged over 5 runs for an example from the third group. The minimum and maximum are the worst and best fitness function values, resp., for the individuals returned in 5 runs. Empty shape (e.g. ∘) denotes that the algorithm did not return any correct solution over 5 runs and filled shape that at least one solution was correct.

Tournament size (t_s). Four values of tournament size, equal to 2, 4, 8, and 10 were tested (see Table 1). The results for $t_s = 4$ and $t_s = 8$ were similar to each other. At the same time, $t_s = 4$ led to higher standard deviation of the population (larger diversity of individuals) and was therefore selected for the final experiments.

Elite size (s_e). Four values of elite size, equal to 0, 3, 5 and 10 were tested. The results are presented in Table 1. The algorithm with the elite mechanism is more stable and achieves better results. The value of $s_e = 3$ was finally selected.

Selection pressure (p_s). This parameter describes the probability of the best individual winning the tournament. Values between 0.5 and 1 with a step of 0.1

Table 1: Fitness function with respect to the tournament size (top part) and the elite size (bottom part).

Example	$t_s = 2$			$t_s = 4$			$t_s = 8$			$t_s = 10$		
	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
1	11	-150	165	355	355	355	343	305	355	335	305	355
2	188	110	210	210	210	210	210	210	210	210	210	210
3	-171	-370	155	188	180	210	197	175	210	174	95	210
Example	$s_e = 0$			$s_e = 3$			$s_e = 5$			$s_e = 10$		
	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
1	152	75	280	355	355	355	324	250	355	325	305	355
2	110	30	210	210	210	210	210	210	210	210	210	210
3	43	-30	125	188	180	210	187	175	210	206	190	210

were tested. Results of the algorithm are presented in Fig. 3b. The higher the value of p_s, the lower the standard deviation in the population. Too low standard deviation can have a negative impact on the results due to the lack of diversity in the population. At the same time, an increase of p_s results in an increase of the percentage of correct individuals in the population, as shown in Table 2. Finally, to balance the value of standard deviation and the percentage of correct individuals, $p_s = 0.8$ was chosen.

Table 2: Percentage of correct individuals in the population, in relation to p_s.

Example	p_s					
	0.5	0.6	0.7	0.8	0.9	1
1	0.03	0.14	0.28	0.37	0.37	0.46
2	0.02	0.8	0.18	0.27	0.32	0.37
3	0.01	0.06	0.16	0.25	0.32	0.38

Mutation coefficient (p_m). Values between 0 and l were tested, where l is the harmonization length (number of chords), with a step equal to 1. The best results were achieved with $p_m = 0, 1, 2$. For higher values, the results were significantly weaker, and for the highest ones, the returned results were incorrect.

As a further refinement of p_m, the values from 1 to 2 with step 0.1 were tested, which led to the final selection of $p_m = 1.1$. The results for an example from the third group are presented in Fig. 3c (initial tests with larger values) and Fig. 3d (fine-tuning tests).

Crossover method and probability (p_c). To select the crossover method and its probability, various probability values, between 0 and 1 with a step of 0.1, for the two crossover versions were tested. For each value, tests were run thirty times and the values for all three tuning examples were normalized using min-max normalization. The average results are shown in Table 5.

The algorithm achieved similar results for values between $0.4 - 0.8$. For this reason, t-Student tests were performed to select the best values for each model

Table 3: Normalized mean values of crossover tuning procedure.

p_c	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
mean, classic	0.86	0.84	0.9	0.88	0.91	0.89	0.91	0.92	0.93	0.88	0.9
mean, one-point	0.86	0.9	0.9	0.89	0.91	0.93	0.93	0.95	0.96	0.95	0.95

with a significance level of 0.05. A value of 0.8 was selected for both models. In the last step, the t-Student test was conducted between two crossover variants (both with the chosen probability of 0.8) with hypothesis H_0: "the results obtained are not significantly different" and the resulting p-value=0.113. Finally, one-point model with $p_c = 0.8$ was selected.

Generation number (n). This parameter was chosen as a compromise between the quality of results and the running time. The value of $n = 5000$ was selected from the set $\{1000, 3000, 5000, 10000\}$.

The final selection of the steering parameters was as follows: $s_p = 1000$, $s_e = 3$, $t_s = 4$, $p_s = 0.8$, $p_m = 1.1$, $p_c = 0.8$ (one-point crossover), $n = 5000$.

4.2 Algorithm efficacy

The efficacy of the algorithm was checked on 9 samples taken from the harmony textbook [24]. For each sample, the algorithm was able to find the correct solution in a relatively short time. The generation numbers in which the first correct solution and the best solution were found, resp. are shown in Table 4. In each case, the first correct solution was found in less than 90 generations.

The number of generations required to find the correct solution varies between groups and depends mainly on the length of an example (cf. groups 1 and 2) and, to a lesser extent, the example's complexity (cf. groups 1 and 3). At the same time, for more complex problems (group 3) the solution is likely to improve even after 3500 iterations, which does not happen for easier samples (groups 1 and 2).

4.3 Evaluation by the human expert

The algorithm evaluates harmonizations based merely on their numerical fitness. Hence, we asked a harmony teacher to assess their aesthetic value, as well. The evaluation was performed according to a school scale from 1 (lowest score) to 5 (highest score). Out of 9 solutions, 4, 4 and 1 were rated 5, 4.5 and 4, resp., with the average grade of 4.67. This means that the solutions are theoretically and sonically correct. An example solution rated 5 is presented in Fig. 4.

4.4 Algorithm running time

The average running times of the algorithm in three groups are presented in Table 5. One can observe a quasi-linear relationship between the example length and the execution time. Harmonizations in the group 2 are obtained in about

Table 4: The number of generations required to find a solution.

Group no.	Example no.	Generation number in which the result was found:					
		first correct harmonization			finally returned harmonization		
		Mean	Min	Max	Mean	Min	Max
1	1	16.6	14	22	208.2	86	343
	2	16.8	13	22	268.2	129	744
	3	14.8	12	18	218.4	109	397
2	4	8.2	6	10	177.6	27	593
	5	3.2	1	5	24	13	36
	6	6.8	5	8	826.2	75	3200
3	7	33.6	19	86	1933.6	96	3576
	8	19.2	17	22	699.8	249	1224
	9	21	18	25	3098.6	2179	3838

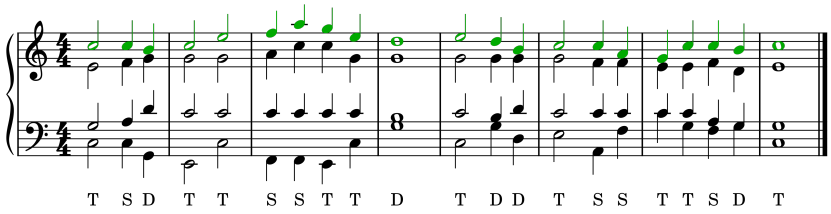


Fig. 4: Harmonization created by the algorithm for an example from the third group. Given line (soprano) is marked in green.

half of the time required for harmonizing samples from groups 1 and 3. On the other hand, it seems that the degree of the example's complexity does not affect the running time - the average times in groups 1 and 3 are similar.

Table 5: The average algorithm's running time (harmonization time) in seconds.

Group 1			Group 2			Group 3		
Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
531.91	485.47	567.02	225.76	180.66	252.05	536.18	508.68	582.39

4.5 Parameters' relevance and robustness

The experiments showed that changing some parameters has a greater effect on the results than changing other parameters. The crossover method, crossover probability p_c, and the selection pressure (p_s) have a relatively small impact on the results. For selection pressure, any value above 0.5 yields satisfactory results.

In contrast, changes of mutation probability value ($\frac{p_m}{l}$) have significant impact. The results achieved for mutation coefficient between 1 and 2 are stable, but increasing p_m above 2 results in a gradual results deterioration. The elite mechanism has been shown to be crucial for the algorithm's performance. Its lack causes significant performance degradation and lower repeatability of results.

5 Conclusions

Creating melodic line harmonization is a non-trivial task. In this paper, we employ EAs to approach this problem. There are two key components of the proposed algorithm: (a) restriction of the search space (*)-(**) to feasible solutions, and (b) specially-designed fitness function, based on theoretical music rules, that defines proper harmonizations. The fitness function consists of three modules: one responsible for the correctness of harmonization and the other two for its quality. The harmonization process is performed for the whole chords and likewise the mutation and crossover operators are applied to the whole chords, not to individual notes.

Harmonizations constructed by the algorithm were evaluated by the harmony teacher so as to additionally assess their aesthetic properties (sound). All but one harmonization were rated at least 4.5 on a scale from 1 to 5, with a good number of them rated 5. This means that in terms of musical quality generated harmonizations meet all expectations. The algorithm finds the solution quickly in terms of both the number of generations and the overall computational time.

The modular design of the fitness function allows it to be easily expanded and modified in the future. Adding more theoretical rules should allow harmonizations to be generated for more advanced and complex harmonic functions. Moreover, the task definition can be extended to the generation of harmonizations for melodic lines without the presence of harmonic functions.

References

1. https://github.com/MelodicLineHarmonization/melodicLineHarmonization.git
2. Benham, H.: A Student's Guide to Harmony and Counterpoint. Rhinegold Publishing Limited (2006)
3. Buys, J., van der Merwe, B.: Chorale harmonization with weighted finite-state transducers. In: Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa. pp. 95–101. PRASA South Africa (2012)
4. Carnovalini, F., Rodà, A.: Computational creativity and music generation systems: An introduction to the state of the art. Frontiers in AI **3**, 14 (2020)
5. Coello, C.A.C., Lamont, G.B.: Applications of multi-objective evolutionary algorithms, vol. 1. World Scientific (2004)
6. De Prisco, R., Eletto, A., Torre, A., Zaccagnino, R.: A neural network for bass functional harmonization. In: European Conference on the Applications of Evolutionary Computation. pp. 351–360. Springer (2010)
7. Freitas, A., Guimaraes, F.: Melody harmonization in evolutionary music using multiobjective genetic algorithms. Proceedings of the Sound and Music Computing Conference. (2011)

8. Gang, D., Lehmann, D., Wagner, N.: Tuning a neural network for harmonizing melodies in real-time. In: ICMC (1998)
9. Grinstein, E., Duong, N.Q., Ozerov, A., Pérez, P.: Audio style transfer. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 586–590. IEEE (2018)
10. Hild, H., Feulner, J., Menzel, W.: Harmonet: A neural net for harmonizing chorales in the style of J.S.Bach. NIPS'91: Proceedings of the 4th International Conference on Neural Information Processing Systems pp. 267—274 (1991)
11. Hu, Y., Liu, K., Zhang, X., Su, L., Ngai, E., Liu, M.: Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review. Applied Soft Computing **36**, 534–551 (2015)
12. Jiang, N., Jin, S., Duan, Z., Zhang, C.: Rl-duet: Online music accompaniment generation using deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 710–718 (2020)
13. Liu, C.H., Ting, C.K.: Computational intelligence in music composition: A survey. IEEE Trans. on Emerging Topics in Computational Intelligence **1**(1), 2–15 (2016)
14. Lopez-Rincon, O., Starostenko, O., Ayala-San Martín, G.: Algoritmic music composition based on artificial intelligence: A survey. In: 2018 International Conference on Electronics, Communications and Computers. pp. 187–193. IEEE (2018)
15. Mańdziuk, J., Goss, M., Woźniczko, A.: Chopin or not? a memetic approach to music composition. In: 2013 IEEE Congress on Evolutionary Computation. pp. 546–553 (2013)
16. Mańdziuk, J., Woźniczko, A., Goss, M.: A neuro-memetic system for music composing. In: Iliadis, L., Maglogiannis, I., Papadopoulos, H. (eds.) Artificial Intelligence Applications and Innovations. pp. 130–139. Springer Berlin Heidelberg (2014)
17. Mańdziuk, J., Żychowski, A.: A memetic approach to vehicle routing problem with dynamic requests. Applied Soft Computing **48**, 522–534 (2016)
18. Moray, A., Williams, C.K.I.: Harmonising chorales by probabilistic inference. Advances in Neural Information Processing Systems **17**, 25–32 (2005)
19. Oliveira, H.G.: A survey on intelligent poetry generation: Languages, features, techniques, reutilisation and evaluation. In: Proceedings of the 10th international conference on natural language generation. pp. 11–20 (2017)
20. Olseng, O., Gambäck, B.: Co-evolving melodies and harmonization in evolutionary music composition. International Conference on Computational Intelligence in Music, Sound, Art and Design. (2018)
21. Pachet, F., Roy, P.: Musical harmonization with constraints: A survey. Constraints **6**(1), 7–19 (2001)
22. Prisco, R.D., Zaccagnino, G., Zaccagnino, R.: Evocomposer: an evolutionary algorithm for 4-voice music compositions. Evolutionary computation **28**(3), 489–530 (2020)
23. Rimsky-Korsakov, N.: Practical Manual of Harmony. C. Fischer (2005)
24. Sikorski, K.: Harmony part 1. PWM (2020)
25. Siphocly, N.N., Salem, A.B.M., El-Horabty, E.S.M.: Applications of computational intelligence in computer music composition. International Journal of Intelligent Computing and Information Sciences **21**(1), 59–67 (2021)
26. Wassermann, G., Glickman, M.: Automated harmonization of bass lines from bach chorales: a hybrid approach. Computer Music Journal **43**(2-3), 142–157 (2020)
27. Żychowski, A., Gupta, A., Mańdziuk, J., Ong, Y.S.: Addressing expensive multi-objective games with postponed preference articulation via memetic co-evolution. Knowledge-Based Systems **154**, 17–31 (2018)