Automated Arrangements of Multi-Part Music for Sets of Monophonic Instruments

Matthew McCloskey[1], Gabrielle Curcio[1], Amulya Badineni[1], Kevin McGrath[1], Georgios Papamichail[2], and Dimitris Papamichail[1] ⋆

[1] The College of New Jersey, Ewing, NJ 08618, USA
[2] New York College, Athens, Greece
papamicd@tcnj.edu

Abstract. Arranging music for a different set of instruments that it was originally written for is traditionally a tedious and time-consuming process, performed by experts with intricate knowledge of the specific instruments and involving significant experimentation. In this paper we study the problem of automating music arrangements for music pieces written for monophonic instruments or voices. We designed and implemented an algorithm that can always produce a music arrangement when feasible by potentially transposing the music piece to a different scale, permuting the assigned parts to instruments/voices, and transposing individual parts by one or more octaves. We also published open source software written in Python that processes MusicXML files and allows musicians to experiment with music arrangements. Our software can serve as a platform for future extensions that will include music reductions and inclusion of polyphonic instruments.

Keywords: music arrangement, music algorithms

1 Introduction

Music arrangements involve the adaptation of a piece of music for different instruments or ensembles. This allows the music to be performed in a variety of settings, enhances the repertory of musicians, and can also help to bring new life to a piece that may have been composed for a specific instrument or ensemble [16]. Additionally, arrangements can help to showcase the unique strengths of different instruments or even create entirely new interpretations of a piece. The process of arranging a piece of music can be a creative endeavor in itself, giving the arranger the opportunity to put their own spin on a familiar work, greatly enhancing the listening experience for audiences [1, 5, 12].

The computational complexity of arranging music written for a set of instruments toward a target single instrument, often employing reasonable reductive constraints, has

⋆ The authors acknowledge use of the ELSA high performance computing cluster at The College of New Jersey for conducting the research reported in this paper. This cluster is funded in part by the National Science Foundation under grant numbers OAC-1826915 and OAC-1828163.

been examined in the work of Moses and Demaine [4]. Complexities of dealing with polyphonic instruments, such as piano and guitar, include the need of considering possible fingerings as well as reductions, the elimination of certain notes for playability of even feasibility. Most research in automating music arrangements has concentrated on the piano, primarily concerning orchestral pieces [2,8,10,11,13,14]. Much of that work involves reductions to enable feasibility. Other work in the field has examined arrangements for the guitar [6,7,15], wind ensembles [9], and other orchestral instruments [3].

Despite its obvious benefits, we are not aware of any published algorithm or widely available software that allows for the automated arrangement of a given music piece to a different set of instruments that it was originally written for in the general case. Working toward filling that need, we designed and implemented an algorithm that arranges music written for monophonic instruments and guarantees a successful outcome when an arrangement is possible without score reduction. Our recursive backtracking algorithm exhaustively examines all feasible assignments of parts to available instruments and all possible transpositions of the piece, including independent octave transpositions of individual parts, to determine a successful arrangement that minimally affects the musicality of the piece. The use of memoization, storing partial results for reuse, further enhances the time efficiency of our software and allows processing of most music pieces in a matter of seconds.

2 Methods

2.1 Definitions

For the purposes of our research, a music piece is written in a chromatic scale and notes are separated by the interval of a semitone. We will assume that all notes fall within a total range of 88 semitones, the notes of a traditional piano, from A0 to C8. We will assign an integer to each note in the range, such that all notes can be represented by an integer from 1 to 88. For our discussion, a monophonic instrument is one that can only play one pitch at a time, such as the flute, the oboe, or a voice. Polyphonic instruments can play multiple notes simultaneously, such as the piano, guitar, or harp. A polyphonic instrument can always play a monophonic part within its range.

For our study an input music piece will consist of n parts, each being assigned to a single monophonic instrument or voice. Such parts are presented in the sheet music representation of the piece in an equal number of staves each. Our algorithm preserves the rhythm, rhythmic values of notes and rests, as well as bar lines of the music piece. Clefs, key signatures and accidentals are adjusted based on the scale of the transposed music and the instruments/voices that parts are assigned to. Our algorithm does not control for instrument timbre that may be expected in any part of the music; similarly, the thickness of the piece is not being necessarily maintained.

We will assume that an input music piece is originally written for n instruments $I_1, I_2, \cdots, I_n$, each assigned to play a part P_i of the piece, with $1 \leq i \leq n$. We seek to arrange the music for n output instruments $O_1, O_2, \cdots, O_n$. The range of each part i is an integer interval $R_i = [\![a_i, b_i]\!]$, where a_i is the integer value corresponding to the lowest frequency note and b_i to the highest frequency note played by instrument I_i in

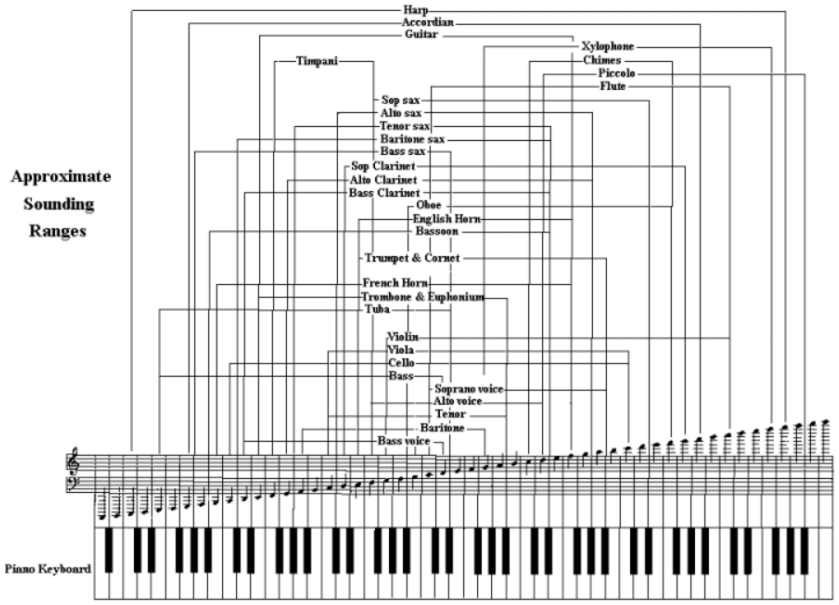


Fig. 1: Approximate sounding ranges of instruments and voices. Figure reproduced with permission from Dr. Brian Blood (dolmetch.com)

part P_i, $1 \leq i \leq n$. Likewise, the playing range of each output instrument O_i will be denoted by OR_i, $1 \leq i \leq n$, indicating the integer interval of values corresponding to the notes the instrument is able to play. Approximate ranges for a set of instruments and voices can be seen in Figure 1.

2.2 Monophonic instrument set arrangement algorithm

Our Monophonic Music Arrangement (MMS) algorithm performs a nearly comprehensive search of possible permutations of parts. The music is transposed to all twelve keys, and the algorithm runs on each key, unless a solution has been found so far that results in fewer sharps/flats over all keys for each part. This is designed to prevent the ”ideal” transposition from having a complex key signature if not necessary. Other than that, the search is fully comprehensive. For each part, the algorithm finds all possible transpositions of each part in the source piece that can be played by at least one available instrument. All permutations of these possible transpositions are then examined. If all parts can be played by at least one instrument, the algorithm then checks if there exists a set of part assignments that is valid. This is performed by a recursive function that is memoized to improve performance. If a transposed key yields valid permutations, the transposition with the least total deviation from the original composition is selected. Once all twelve keys have been checked, all permutations are tried using the selected transposition, unless there is no selected transposition, in which case the algorithm fails. All permutations are checked, and for those that are valid in the given

transposition, the best arrangement is selected based on how closely the average pitch of each part matches the median pitch of the instrument's range.

The MMA algorithm implementation consists of four main function described in pseudocode below.

Algorithm 1 Find Transposed Options

procedure FINDTRANSPOSEDOPTIONS($originalStream, arrangementParts, semitones$)
  $stream \leftarrow originalStream$ transposed by given $semitones$
  $parts \leftarrow$ new list
  for $part$ in $stream$ **do**
    $choices \leftarrow$ new list
    for each $transposition$ **do**
      $set \leftarrow$ the subset of $arrangementParts$ that can play at this transposition
      add $(semitones + transposition, set)$ to $choices$
    end for
    if $choices$ is empty **then**
      return null
    end if
    add $choices$ to $parts$
  end for
  return $parts$
end procedure

Algorithm 2 Run Transposed

procedure RUNTRANSPOSED($stream, parts, semitones$)
  $selections \leftarrow$ new list
  for $option$ in all possible transpositions from FINDTRANSPOSEDOPTIONS($stream, parts, semitones$) **do**
    $partsCovered \leftarrow$ new list
    $selection \leftarrow$ new list
    for $transposition$ in $option$ **do**
      add set of parts covered to $partsCovered$
      add deviation of transposition to $selection$
    end for
    $allPartsCovered \leftarrow$ the union of all sets in $partsCovered$
    if $allPartsCovered$ contains all parts and $ValidateArrangement$($parts$, $partsCovered$, $allPartsCovered$) **then**
      add $selection$ to $selections$
    end if
  end for
  return $selections$
end procedure

Algorithm 3 Find Best Choice

procedure FINDBESTCHOICE($stream, parts$)
  $bestChoice \leftarrow$ null
  $bestSharps \leftarrow \infty$
  for $semitones$ from -6 through 5 **do**
    $sharps \leftarrow$ the total number of sharps/flats that would appear in the key signature for each part
    if $sharps \leq bestSharps$ **then**
      $thisBestChoice \leftarrow$ element from RUNTRANSPOSED($stream, parts, semitones$) with the least deviation
      if $thisBestChoice \neq$ null and either $sharps < bestSharps$ or deviation of $thisBestChoice <$ deviation of $bestChoice$ **then**
        $bestChoice \leftarrow thisBestChoice$
        $bestSharps \leftarrow thisBestSharps$
      end if
    end if
  end for
  return $bestChoice$
end procedure

Algorithm 4 MMA Algorithm

procedure MMA($stream, parts$)
  $bestChoice \leftarrow$ FINDBESTCHOICE($stream, parts$)
  if $bestChoice =$ null **then**
    return null
  end if
  transpose each part by the resulting transposition
  $bestFit \leftarrow \infty$
  for each permutation of $newParts$ **do**
    if all parts are valid in the given permutation **then**
      $fit \leftarrow$ the total absolute difference between the average pitches and the median pitch of each part
      if $fit < bestFit$ **then**
        $bestFit \leftarrow fit$
        $bestPermutation \leftarrow$ this permutation
      end if
    end if
  end for
  return $bestPermutation$
end procedure

2.3 Implementation

The MMA algorithm was implemented in Python utilizing the Music21 library and the MuseScore software. Our program requires two input files and produces a single output file with the music arrangement. The required input files consist of the original piece of music in MusicXML format and a TOML file listing the instrument set to arrange for, where an assigned value of k to an instrument indicates k parts should be arranged for that instrument. An example of a TOML file with an input instrument set consisting of one clarinet, two tenor saxophones, and two alto saxophones is shown in Figure 2a. Metadata about each instrument, consisting of its key in notation and a reasonable note range, is defined in a separate TOML file which is loaded separately by the program and is populated with common music instruments. An example of an entry for the alto saxophone in the instrument metadata file is shown in Figure 2b.

```
clarinet = 1
tenor-sax = 2
alto-sax = 2
```

(a) An example arrangement file

```
[alto-sax]
name = "AltoSaxophone"
minimum = "Db3"
maximum = "Bb5"
key = "Eb"
```

(b) An entry in the instrument metadata file

Fig. 2: Examples of input instrument set and instrument information files

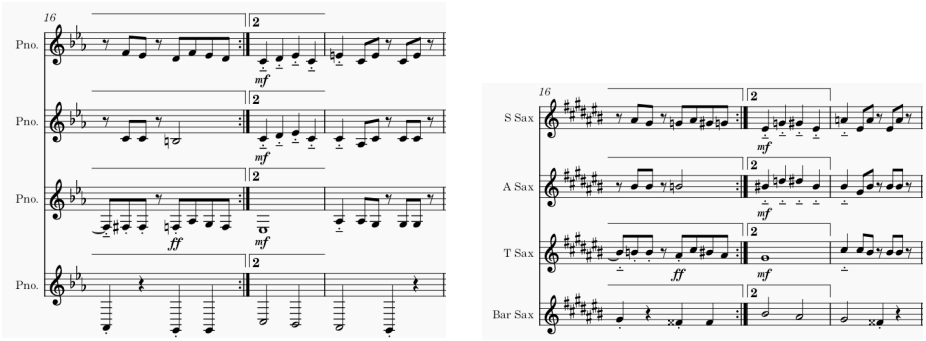
During execution our program checks whether the number of input instruments matches the number of parts in the piece, and then attempts to arrange for the given instruments as previously described. If arrangements are found, the best arrangement based on the criteria described in section 2.2 is output as a MusicXML file. If no feasible arrangement is found, or if the number of instruments does not match, then an error message is displayed and no output file is produced.

3 Results

We tested our software on a variety of music pieces written for monophonic instruments. In Figure 3 we show three measures, starting at measure 16, of the *Puttin' on the Ritz* song by Irving Berlin. Part (a) shows the input score composed of four monophonic parts. Part (b) displays the arranged piece for saxophone quartet, consisting of a soprano, alto, tenor, and baritone saxophones. Similarly, in Figure 4 we display three measures of *Carol of the Bells*, as arranged and performed by the Pentatonix voice group, starting at measure 18 of the piece.

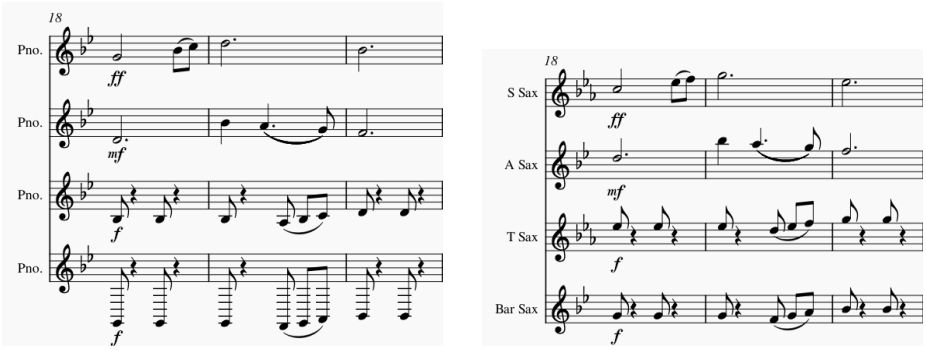
Complete input/output files for three test cases of our software, including the *Puttin' on the Ritz* and *Carol of the Bells* above, can be examined at:
https://owd.tcnj.edu/∼papamicd/music/mma/examples/

The software repository for this project can be found at: https://github.com/spazzylemons/music-arrangement/



(a) Original Score (b) Arranged score

Fig. 3: Three measures from an arrangement of 'Puttin' on the Ritz' from piano to saxophone quartet



(a) Original Score (b) Arranged score

Fig. 4: Three measures from an arrangement of 'Carol of the Bells' from voices to saxophone quartet

4 Conclusions and future work

Our monophonic music arrangement algorithm and its software implementation create a platform for automating music arrangements with minimal user input. Although currently basic in its functionality, it is now being extended in a number of different directions. For accommodating arrangements for a smaller sets of instruments than the number of parts in the music, we are examining score reduction techniques to eliminate certain parts or at least reduce the number of simultaneous notes that are played throughout the piece, while maintaining faithfulness to the original. To allow for the inclusion of polyphonic instruments in the arrangements, we are looking into analyzing and decomposing polyphonic parts into monophonic ones and inversely, while adhering to constraints related to fingerings and other instrument and player restrictions.

References

1. D. Baker. *David Baker's Arranging & Composing: For the Small Ensemble, Jazz, R & B, Jazz-rock*. Alfred Publishing Company, 1988.
2. Shih Chuan Chiu, Man Kwan Shan, and Jiun Long Huang. Automatic system for the arrangement of piano reductions. In *ISM 2009 - 11th IEEE International Symposium on Multimedia*, 2009.
3. Léopold Crestel and Philippe Esling. Live orchestral piano, a system for real-time orchestral music generation. In *Proceedings of the 14th Sound and Music Computing Conference 2017, SMC 2017*, 2019.
4. Erik D. Demaine and William S. Moses. 364Computational Complexity of Arranging Music. In *The Mathematics of Various Entertaining Subjects: Research in Games, Graphs, Counting, and Complexity, Volume 2*. Princeton University Press, 09 2017.
5. J.B. Elder. *The Art of Arranging and Orchestration*. Independently Published, 2018.
6. Gen Hori, Hirokazu Kameoka, and Shigeki Sagayama. Input-output HMM applied to automatic arrangement for guitars. *Journal of Information Processing*, 2013.
7. Gen Hori, Yuma Yoshinaga, Satoru Fukayama, Hirokazu Kameoka, and Shigeki Sagayama. Automatic arrangement for guitars using hidden markov model. *Proceedings of 9th Sound and Music Computing Conference (SMC2012)*, pages 450–455, 7 2012.
8. Jiun-Long Huang, Shih-Chuan Chiu, and Man-Kwan Shan. Towards an automatic music arrangement framework using score reduction. *ACM Trans. Multimedia Comput. Commun. Appl.*, 8(1), feb 2012.
9. Hiroshi Maekawa, Norio Emura, Masanobu Miura, and Masuzo Yanagida. On machine arrangement for smaller wind-orchestras based on scores for standard wind-orchestras. In *International Conference on Music Perception and Cognition, ICMPC 2006*, pages 268–273, 2006.
10. Eita Nakamura and Kazuyoshi Yoshii. Statistical piano reduction controlling performance difficulty. *APSIPA Transactions on Signal and Information Processing*, 2018.
11. Sho Onuma and Masatoshi Hamanaka. Piano arrangement system based on composers' arrangement processes. In *International Computer Music Conference, ICMC 2010*, 2010.
12. T.H. Stefan Kostka, T.H. Dorothy Payne, and B. Almén. *Tonal Harmony*. McGraw-Hill Education, 2017.
13. Hirofumi Takamori, Haruki Sato, Takayuki Nakatsuka, and Shigeo Morishima. Automatic arranging musical score for piano using important musical elements. In *Proceedings of the 14th Sound and Music Computing Conference 2017, SMC 2017*, 2019.
14. Moyu Terao, Yuki Hiramatsu, Ryoto Ishizuka, Yiming Wu, and Kazuyoshi Yoshii. Difficulty-aware neural band-to-piano score arrangement based on note- and statistic-level criteria. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 196–200, 2022.
15. D.R. Tuohy and W.D. Potter. Ga-based music arranging for guitar. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1065–1070, 2006.
16. G.C. White. *Instrumental Arranging*. McGraw-Hill, 1992.