Selecting inversions in a chord progression with the triad by a lexicographic bi-criteria optimization

Yoshiyuki KARUNO* and Ryota YOSHII**

* Faculty of Mechanical Engineering, Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto-shi, Kyoto 606-8585, Japan
E-mail: karuno@kit.ac.jp

** Graduate School of Science and Technology, Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto-shi, Kyoto 606-8585, Japan

Received: 6 October 2019; Revised: 27 January 2020; Accepted: 18 March 2020

Abstract
We are given a sequence of \( m \) machines and an ordered set of \( n \) jobs. The \( n \) jobs are served by the \( m \)-machine system in their indexing order. We are also given a collection of \( p_j \) machine subsets for each job \( j \in N \), where \( N \) denotes the set of jobs. We are primarily asked to choose a machine subset from the \( p_j \) options for each job \( j \in N \) so that the similarity sum over a series of chosen \( n \) machine subsets is maximized, where a degree of similarity of two machine subsets is defined to be the number of common machines between the two machine subsets. The series of chosen \( n \) machine subsets also induces a subsequence of machines for serving the \( n \) jobs. We call the length of an induced machine subsequence its stretch. We are secondarily asked to choose a machine subset from the \( p_j \) options for each job \( j \in N \) so that the stretch of the induced machine subsequence is minimized. The lexicographic machine subset selection problem is motivated by a musical issue of selecting inversions in a chord progression with the triad. In this paper, we propose a polynomial time exact algorithm for the lexicographic machine subset selection problem. We also demonstrate the solutions on three examples of chord progressions with the triad, and on randomly generated instances.

Keywords: Lexicographic optimization, Machine subset selection, Chord progressions, Inversions, Dynamic programming, Iterative guessing, Polynomial time algorithms

1. Introduction

In this paper, we deal with a combinatorial optimization problem which is motivated by a musical issue of selecting inversions in a chord progression (i.e., a succession of chords) with the triad. An ordinary chord with the triad consists of three notes. For example, C major triad consists of C, E and G, where C is the root (e.g., see Koyano, 2011). A chord is said to take the root position when the bottom note in the music score is the root, while it is said to take an inversion when the bottom note is not the root. In the music keyboard play of a chord progression, one may feel something smooth (or steady) in the person’s mind when an inversion is taken for a chord, instead of the root position. We refer to an inversion as a rotated position for convenience, e.g., we refer to the first inversion of a chord as the first rotated position of it. In this paper, we regard a piano key as a machine, each chord in a chord progression as a job, and the choice of a position for each chord as a combinatorial optimization problem. We are going to show a clear example of the correspondence between piano keys and machines later in Table 1. Several computer models of musical creativity have been considered, e.g., see Cope (2005), and those models seem very interesting, of course. On the other hand, we try looking the musical issue of selecting inversions from a mathematical viewpoint of optimization, and we do not treat a model of musical creativity by computers in this paper. We present a further detail of our combinatorial optimization problem below.

We are given a sequence of \( m \) machines and an ordered set of \( n \) jobs. The \( n \) jobs are served by the \( m \)-machine system in their indexing order, which follows an actual situation that the music keyboard play accords with a given chord progression. We are also given a collection of \( p_j \) machine subsets for each job \( j \in N \), where \( N \) denotes the set of jobs,
and each of the $p_j$ options corresponds to a position of a chord. In the music keyboard play, we may often treat $p_j = 3$ for each job $j \in N$, that is, the root, the first rotated and the second rotated positions for a chord. We are primarily asked to choose a machine subset from the $p_j$ options for each job $j \in N$ so that the similarity sum over a series of chosen $n$ machine subsets is maximized, where a degree of similarity of two machine subsets is defined to be the number of common machines between the two machine subsets. Common machines between two machine subsets correspond to common piano keys between positions of two chords. We expect a moderate change on piano keys struck for performing two consecutive chords to be indicated by a large degree of similarity. As mentioned in the above, we are going to illustrate the correspondence between piano keys and machines later in a table.

In addition, the series of chosen $n$ machine subsets also induces a subsequence of machines for serving the $n$ jobs. Of course, we treat the minimal one as the induced machine subsequence, and it contains the chosen machine subset for each of the $n$ jobs. We call the length of an induced machine subsequence its stretch. We are secondarily asked to choose a machine subset from the given $p_j$ options for each job $j \in N$ so that the stretch of the induced machine subsequence is minimized. We hope that the two objective functions, similarity sum and stretch, represent the feeling of something smooth (or steady) in our minds when listening to a music with a given chord progression. In this paper, we propose a polynomial time exact algorithm for the lexicographic machine subset selection problem. The proposed algorithm is based on a dynamic programming procedure for the longest path problem in a directed acyclic graph (DAG for short), e.g., see Skiena (2008), and on an iterative guessing technique. In this paper, we also demonstrate the solutions obtained by the proposed algorithm on three examples of chord progressions with the triad, and on randomly generated instances.

## 2. Problem Description

### 2.1. Notation and Formulation

Let $M = \{1, 2, \ldots, m\}$ denote a sequence of $m$ machines (precisely, $m$ indexes of machines), and let $N = \{1, 2, \ldots, n\}$ denote a set of $n$ jobs (precisely, $n$ indexes of jobs). We can understand the sequence $M$ with length $m$ also as an ordered set of $m$ machines. Let $M[a, b] \subseteq M$ denote a subsequence of machines from machine $a \in M$ to machine $b \in M$ with length $b - a + 1$, where $1 \leq a \leq b \leq m$. The definition implies $M[1, m] = M$. The $n$ jobs are served by the $m$-machine system in their indexing order, that is, the job set $N$ is also an ordered set. For each job $j \in N$ and a given positive integer $p_j$, let $C_j = \{S_j(1), S_j(2), \ldots, S_j(p_j)\}$ denote a collection of non-empty subsets of machines, i.e., for each $k \in \{1, 2, \ldots, p_j\}$, it meets $\emptyset \neq S_j(k) \subseteq M$. Also, for each job $j \in N$ and each $k \in \{1, 2, \ldots, p_j\}$, let $a_j(k) \in M$ and $b_j(k) \in M$ denote the machines with the smallest and the largest indexes in $S_j(k)$, respectively. Obviously, $S_j(k) \subseteq M[a_j(k), b_j(k)]$ holds. Further, let $p = \max_{j \in N}[p_j]$, by which we are going to express the time complexity of the proposed algorithm.

The lexicographic machine subset selection problem to be discussed in this paper requires to choose a machine subset from given collection $C_j$ for each job $j \in N$. In this paper, we consider a situation such that machines in a chosen $S_j(k)$ for each job $j \in N$ serve the job cooperatively at the same time. Recall that some piano keys are simultaneously struck when performing a chord in the music keyboard play. Let $x = (x_1, x_2, \ldots, x_n)$ denote a solution of the lexicographic machine subset selection problem, where $x_j$ is a positive integer for each job $j \in N$. If $x_j \in \{1, 2, \ldots, p_j\}$ is met for every job $j \in N$, then the solution $x = (x_1, x_2, \ldots, x_n)$ is called a feasible one, since we can express by $S_j(x_j)$ the chosen machine subset for each job $j \in N$. For a feasible solution $x = (x_1, x_2, \ldots, x_n)$, let $a(x) = \min\{a_j(x_j) \mid j \in N\}$ and let $b(x) = \max\{b_j(x_j) \mid j \in N\}$. Then, $M[a(x), b(x)] \subseteq M$ denotes the induced machine subsequence by the feasible solution $x$. We see that for each job $j \in N$,

$$S_j(x_j) \subseteq M[a(x), b(x)]$$

clearly holds by definition.

For each $j \in N \setminus \{1\}$, $1 \leq \ell \leq p_{j-1}$ and $1 \leq k \leq p_j$, let

$$d_j(\ell, k) = \left| S_{j-1}(\ell) \cap S_j(k) \right|$$

denote the degree of similarity between two machine subsets $S_{j-1}(\ell)$ and $S_j(k)$, which means the number of common machines contained in $S_{j-1}(\ell)$ and $S_j(k)$. Then, for a feasible solution $x = (x_1, x_2, \ldots, x_n)$, we define the similarity sum over the series $S_1(x_1), S_2(x_2), \ldots, S_n(x_n)$ of chosen machine subsets by

$$f(x) = \sum_{j=2}^{n} d_j(x_{j-1}, x_j) \left( = \sum_{j=2}^{n} \left| S_{j-1}(x_{j-1}) \cap S_j(x_j) \right| \right).$$

[DOI: 10.1299/jamdsm.2020jamdsm0068] © 2020 The Japan Society of Mechanical Engineers
As the primary objective of the lexicographic machine subset selection problem, we consider the similarity sum \( f(x) \) to be maximized. For a given instance of the lexicographic machine subset selection problem, we express by \( f^* \) the maximum of the similarity sum.

We also refer to
\[
g(x) = b(x) - a(x) + 1
\]
as the \textit{stretch} of machine subsequence \( M[a(x), b(x)] \) induced by a feasible solution \( x \). The secondary objective is to minimize the stretch \( g(x) \) of an induced machine subsequence. Eventually, we finish to formulate the lexicographic machine subset selection problem. Let \( x = x' \) denote a feasible solution such that it attains the maximum \( f(x') = f^* \) of similarity sum, and it also meets \( g(x') \leq g(x') \) for any feasible solution \( x' \) with the maximum similarity sum \( f(x') = f^* \). Let \( g^* = g(x') \) denote the \textit{conditionally minimum} of stretch. We call such a feasible solution \( x' \) an \textit{optimal} one. The lexicographic machine subset selection problem asks to find an optimal solution for a given instance.

2.2. An Instance

We here give an instance of the lexicographic machine subset selection problem with \( m = 18 \), \( n = 4 \), \( p_1 = 3 \), \( p_2 = 1 \), \( p_3 = 3 \) and \( p_4 = 1 \):

\[
S_1(1) = \{1, 5, 8\}, \quad S_1(2) = \{5, 8, 13\}, \quad S_1(3) = \{8, 13, 17\},
\]
\[
S_2(1) = \{10, 13, 17\},
\]
\[
S_3(1) = \{3, 6, 10\}, \quad S_3(2) = \{6, 10, 15\}, \quad S_3(3) = \{10, 15, 18\},
\]
\[
S_4(1) = \{8, 12, 15\}.
\]

For a feasible solution \( x = (1, 1, 1, 1) \), we obtain
\[
f(x) = |[1, 5, 8] \cap [10, 13, 17]| + |[10, 13, 17] \cap [3, 6, 10]| + |[3, 6, 10] \cap [8, 12, 15]| = 0 + 1 + 0 = 1
\]
as the similarity sum, and
\[
g(x) = \max(8, 17, 10, 15) - \min(1, 10, 3, 8) + 1 = 17 - 1 + 1 = 17
\]
as the stretch of the induced machine subsequence \( M[1, 17] \). This instance is going to appear again in the numerical section as an example of selecting rotated positions (i.e., inversions) in a chord progression.

3. A Polynomial Time Exact Algorithm

3.1. Recursive for the Maximum Similarity Sum

We first provide a dynamic programming procedure to compute the maximum of the similarity sum over a series of chosen machine subsets for the \( n \) jobs. For each job \( j = 1, 2, \ldots, n \) and each number \( k = 1, 2, \ldots, p_j \), we define the \textit{state variable} \( \omega(j, k) \) to be the maximum of the similarity sum when the first \( j - 1 \) jobs have been served and the \( k \)-th machine subset is chosen for the \( j \)-th job.

We set the boundary condition to be: For \( k = 1, 2, \ldots, p_1 \),
\[
\omega(1, k) := 0.
\]

Then, for \( j = 2, 3, \ldots, n \) and for \( k = 1, 2, \ldots, p_j \), we recursively compute
\[
\omega(j, k) := \max_{1 \leq l \leq p_{j-1}} \left\{ \omega(j - 1, l) + d_j(l, k) \right\}, \quad \omega(j, 0) := 0,
\]
\[
x(j, k) := \arg \max_{1 \leq l \leq p_{j-1}} \left\{ \omega(j - 1, l) + d_j(l, k) \right\},
\]
where \( x(j, k) \) stores the predecessor of the \( k \)-th machine subset for the \( j \)-th job, which is going to be utilized in the backtracking process. We call the dynamic programming procedure of maximizing the similarity sum DP\_SS for short.

Procedure DP\_SS has the same structure of dynamic programming as that for the longest path problem in a DAG (e.g., see Skiena, 2008), and the maximum \( f^* \) of the similarity sum over a series of chosen \( n \) machine subsets can be obtained to be
\[
f^* = \max_{1 \leq k \leq p_n} \omega(n, k).
\]
Also, a feasible solution $x' = (x'_1, x'_2, \ldots, x'_p)$ attaining the maximum similarity sum $f'$ can be obtained by tracing back the predecessors from the machine subset $S_p(k^*)$ with $k^* = \arg \max_{1 \leq k \leq p} \omega(n, k)$ = $x'_p$.

Recall $p = \max_{j \in N}(p_j)$. There exist $O(p^2 n)$ degrees $d_j(\ell, k)$ of similarity, and it takes $O(m)$ time to obtain each $d_j(\ell, k)$ (see Eq. (1)), since it is the cardinality of the intersection between two machine subsets $S_{j-1}(\ell)$ and $S_j(k)$. There are $O(pn)$ state variables $\omega(j, k)$, and it takes $O(p)$ time to find the maximum among $O(p)$ choices in each $\omega(j, k)$ (see Eq. (5)). The backtracking process obviously requires $O(p) + O(m)$ time (see Eq. (7)), since it traces back a machine subset for each of the $n$ jobs by utilizing the predecessors $x(j, k)$. Hence, the time complexity of procedure DP_SS is evaluated to be

$$O(p^2 n) \times O(m) + O(pn) \times O(p) + O(p) + O(n) = O(p^2 mn) + O(p^2 n) + O(p) + O(n) = O(p^2 mn).$$

**Lemma 1.** For an instance of the lexicographic machine subset selection problem, the maximum of the similarity sum can be computed in $O(p^2 mn)$ time.

We remark that given the $O(p^2 n)$ degrees of similarity, procedure DP_SS itself runs in $O(p^2 n)$ time.

### 3.2. Computation of the Minimum Stretch

For an instance of the lexicographic machine subset selection problem, let $g_{\text{min}}$ denote the minimum stretch of an induced machine subsequence. For the conditionally minimum stretch $g'$ of the stretch, it clearly satisfies $g_{\text{min}} \leq g'$ by definition. We observe the following property on the minimum stretch $g_{\text{min}}$:

**Observation 1.** For an instance of the lexicographic machine subset selection problem, let $x = x'$ denote a feasible solution which attains the minimum stretch, i.e., $\omega(x') = g_{\text{min}}$. Then, for the induced machine subsequence $M[a(x'), b(x')]$ with stretch $b(x') - a(x') + 1 = g_{\text{min}}$, each job $j \in N$ has some machine subset $S_j(k)$ in collection $C_j$ such that it meets $S_j(k) \subseteq M[a(x'), b(x')]$.

By the observation, if we find a machine subsequence $M[a, b]$ with stretch $h = b - a + 1$ such that for each job $j \in N$, it contains at least one machine subset $S_j(k)$ in collection $C_j$, then the minimum stretch satisfies $g_{\text{min}} \leq h$. Also, by the minimality of the stretch $g_{\text{min}}$, for any machine subsequence $M[a, b]$ with stretch $h = b - a + 1 < g_{\text{min}}$, there exists some job $j \in N$ such that no machine subsets in collection $C_j$ are contained in $M[a, b]$.

Following the observation, we obtain a polynomial time procedure to compute the minimum stretch $g_{\text{min}}$ of an induced machine subsequence shown in Algorithm 1. The procedure guesses the minimum stretch by the parameter $h$ with $1 \leq h \leq m$, and increases $h$ one by one until a machine subsequence containing at least one machine subset from each collection $C_j$ is found.

**Algorithm 1** (Procedure for the Stretch Minimization)

**Input:** A sequence $M$ of $m$ machines, an ordered set $N$ of $n$ jobs, and a collection $C_j = (S_j(1), S_j(2), \ldots, S_j(p_j))$ of machine subsets for each job $j \in N$.

**Output:** A feasible solution $x'$ such that the stretch of the induced machine subsequence is the minimum $\omega(x') = g_{\text{min}}$.

1. for $h = 1$ to $m$ do /* Guess the minimum stretch by the parameter. */
2. for $i = 1$ to $m - h + 1$ do /* Fix the first machine of a machine subsequence with length $h$. */
3. for $j = 1$ to $n$ do
4. $C'_j := (S_j(k) \in C_j | i \leq a_j(k), b_j(k) \leq i + h - 1)$; /* The last machine in the machine subsequence is $i + h - 1$. */
5. end for
6. if $C'_j \neq \emptyset$ for all jobs $j = 1, 2, \ldots, n$ then
7. $g_{\text{min}} := h$; /* The minimum stretch is found. */
8. for $j = 1$ to $n$ do
9. Choose an integer $k' \in K = \{k \in \{1, 2, \ldots, p_j\} | S_j(k) \in C'_j\}$ arbitrarily and set $x'_j := k'$;
10. end for
11. return the feasible solution $x' = (x'_1, x'_2, \ldots, x'_p)$ with the minimum stretch $\omega(x') = g_{\text{min}}$. /* Terminate the computation. */
12. end if
13. end for /* for $i$ */
14. end for /* for $h$ */

It takes $O(p)$ time to extract collection $C'_j$ from given collection $C_j$ of $p_j$ machine subsets for each job $j \in N$ at Line 4. Obviously, the test at Line 6 requires $O(n)$ time. The feasible solution $x'$ with the minimum stretch $g_{\text{min}}$ is constructed from the extracted collections $C'_1, C'_2, \ldots, C'_n$ in $O(n)$ time at Lines 8–10, and the construction is performed exactly once. Hence, by counting double for-loops at Lines 1–2, the time complexity of Algorithm 1 is evaluated to be

$$O(m^2) \times (O(p) \times O(n) + O(n)) + O(n) = O(m^2) \times O(pn) + O(n) = O(pm^2 n) + O(n) = O(p m^2 n).$$
In addition, the first for-loop for guessing the minimum stretch at Line 1 can be replaced by a binary search (e.g., see Ibaraki, 2014), and we see that a faster implementation of the procedure is possible. Although the \(O(n)\) time construction of a feasible solution at Lines 8–10 may be performed more than one time in the faster implementation, the improved time complexity is evaluated to be

\[
O(\log m) \times O(m) \times (O(p) \times O(n) + O(n) + O(n)) = O(pmn \log m).
\]

**Lemma 2.** For an instance of the lexicographic machine subset selection problem, the minimum of the stretch of an induced machine subsequence can be computed in \(O(pmn \log m)\) time.

### 3.3. The Proposed Algorithm

In this section, we combine procedure DP\_SS provided in Section 3.1 with Algorithm 1 presented in Section 3.2 to propose a polynomial time exact algorithm for the lexicographic machine subset selection problem with the primary objective of similarity sum and the secondary objective of stretch. We describe the proposed algorithm as Algorithm 2.

**Algorithm 2 (The Proposed Algorithm)**

**Input:** A sequence \(M\) of \(m\) machines, an ordered set \(N\) of \(n\) jobs, and a collection \(C_j = (S_j(1), S_j(2), \ldots, S_j(p_j))\) of machine subsets for each \(j \in N\).

**Output:** An optimal solution \(x^*\) such that it attains the maximum similarity sum \(f(x^*) = f^*\) and the conditionally minimum stretch \(g(x^*) = g^*\).

1. Prepare all the \(O(p^2n)\) degrees \(d(f, k)\) of similarity \((j = 2, 3, \ldots, n, f = 1, 2, \ldots, p_j-1, k = 1, 2, \ldots, p_j)\).
2. \(f^* := -1\); /* Initialize the parameter for storing the maximum similarity sum. */
3. for \(h = 1\) to \(m\) do /* Fix the length of a machine subsequence to be examined. */
   4. for \(i = 1\) to \(m - h + 1\) do /* Fix the first machine of a machine subsequence with length \(h\). */
      5. for \(j = 1\) to \(n\) do
         6. \(C_j := (S_j(k) \mid k \leq i \leq j - k \leq i + h - 1\); /* The last machine in the machine subsequence is \(i + h - 1\). */
      7. end for
   8. if \(C^* \neq \emptyset\) for all jobs \(j = 1, 2, \ldots, n\) then
      9. if \(\min_{j=1,2,\ldots,n}\{\min_{k>j}d(f(k), k) \mid S_j(k) \in C_j\} = i\) and \(\max_{j=2,\ldots,n}\{\max_{k}d(f(k), S_j(k) \in C_j)\} = i + h - 1\) then
         10. /* Some redundancy of the computation is removed by the above condition. */
      11. Call dynamic programming procedure DP\_SS for the extracted instance with machine subsequence \(M[i, i + h - 1]\), ordered set \(N\) of \(n\) jobs, and for each job \(j \in N\), non-empty collection \(C_j\) of machine subsets;
      12. Receive a feasible solution \(x^* = (x_1^*, x_2^*, \ldots, x_n^*)\) from procedure DP\_SS;
      13. if \(f(x^*) > f^*\) then /* The condition of \(f(x^*) = f^*\) and \(g(x^*) < g^*\) is not necessary to be checked in the above. */
         14. \(f^* := f(x^*)\); \(g^* := g(x^*)\); \(x^* := x^*\); /* The best incumbent solution is updated. */
      15. end if
      16. end if
     17. end if
   18. end if
   19. end for /* for \(i\) */
20. end for /* for \(h\) */
21. return the best incumbent \(x^*\) as an optimal solution.

As in Algorithm 1 for the minimum stretch \(g_{\text{min}}\), the parameter \(h\) for the length of a machine subsequence is increased one by one. Since Algorithm 2 looks for the maximum similarity sum \(f^*\), it can not stop at \(h = g_{\text{min}}\) in the first for-loop (i.e., at Line 3), and it continues the computation with the lengths \(h > g_{\text{min}}\) of machine subsequences. Without the condition of Line 9, a certain feasible solution may be returned by procedure DP\_SS more than one time at Line 11, since the length \(h\) of a machine subsequence is examined in non-decreasing order. Although the correctness of Algorithm 2 does not depend on the existence of Line 9, we remove such redundancy of the computation from the implementation. (As another implementation of Algorithm 2 with some modifications, we can obtain the maximum similarity sum \(f^*\) in advance by calling procedure DP\_SS at Line 2.)

Whenever a feasible solution with a larger similarity sum is obtained in the computation process, the best incumbent solution is updated at Lines 14–17. Recall that for two distinct feasible solutions \(x\) and \(x^*\), if \(f(x) = f(x^*) = f^*\) and \(g(x) > g(x^*)\) hold, then we understand that the solution \(x^*\) is superior to the other \(x\). However, since the length \(h\) of a machine subsequence is examined in non-decreasing order (see Line 3), a feasible solution obtained by procedure DP\_SS at Line 13 can not have a smaller stretch than that of the best incumbent solution. That is, suppose that we obtain a feasible solution \(x^*\) at Line 13 such that it satisfies \(f(x^*) = f^*\) and \(g(x^*) < g^*\) at Line 14 when the length of a machine subsequence has been set to be \(h = h'\) at Line 3. Note that the best incumbent solution meets \(g^* \leq h'\) at that time. Then, while the length of a machine subsequence was set to be \(h = g(x^*) < g^* \leq h'\) at Line 3, we had already obtained either \(x^*\) itself or another
superior solution to \( x' \), which contradicts the current of the best incumbent solution. Hence, the condition of \( f(x') > f^* \) is sufficient at Line 14 as the updating test.

We evaluate the time complexity of Algorithm 2. The proposed algorithm computes all the degrees \( d_j(\ell, k) \) of similarity at the first line in \( O(p^2 n) \times O(m) = O(p^3 mn) \) time as a preprocessing. It has double for-loops (i.e., Lines 3–4) for fixing a machine subsequence. Within the double for-loops, it first extracts collection \( C'_j \) from given collection \( C_j \) in \( O(p) \) time for each \( j \in N \) at Lines 5–7, and it examines in \( O(n) \) time whether all the collections \( C'_j \) are non-empty. As seen in Section 3.1, the time complexity of procedure DP_SS is \( O(p^2 mn) \), while it runs in \( O(p^2 n) \) times if all the degrees \( d_j(\ell, k) \) of similarity are given. The updating of the best incumbent solution at Line 16 clearly requires \( O(n) \) time. Therefore, the entire complexity of Algorithm 2 is evaluated to be

\[
O(p^2 mn) + O(m^2) \times [O(p) \times O(n) + O(n) + O(p^2 n) + O(n)]
\]

\[
= O(p^2 mn) + O(m^2) \times O(p^2 n) = O(p^2 mn) + O(p^2 m^2 n) = O(p^2 m^2 n).
\]

**Theorem 1.** For an instance of the lexicographic machine subset selection problem with the primary objective of similarity sum and the secondary objective of stretch, an optimal solution can be obtained in \( O(p^2 m^2 n) \) time.

The complementary version of the lexicographic machine subset selection problem, i.e., with the first objective of stretch and the secondary objective of similarity sum, can also be solved in polynomial time. For reference, the polynomial time exact algorithm is provided as Algorithm 3 in an extra section.

### 4. Numerical Results

In this section, we conduct numerical experiments to demonstrate the solutions obtained by the proposed algorithm, i.e., Algorithm 2, for the lexicographic machine subset selection problem. First, we apply the proposed algorithm to three numerical examples associated with the music keyboard play of the triad. A layout of piano keys is illustrated in Fig. 1. For an ordinary chord with the triad, three distinct keys are simultaneously struck at the same time by three fingers, e.g., three keys C3, E3 and G3 for chord C major triad. We also observe the behavior of the proposed algorithm on randomly generated instances, which are independent with the music keyboard play.

![Fig. 1 Layout of piano keys](image)

#### 4.1. Examples with the Triad

##### 4.1.1. Example 1

We again consider the problem instance provided in Section 2.2 as the first example of this section, where \( m = 18 \), \( n = 4 \), \( p_1 = 3 \), \( p_2 = 1 \), \( p_3 = 3 \) and \( p_4 = 1 \). The correspondence between the eighteen machines and actual piano keys is shown in Table 1, and the chord progression is given by (C, Am, Dm, G). The layout of the keys has been illustrated in Fig. 1.

In the feasible solution \( x = (1, 1, 1, 1) \), each of the four chords, C, Am, Dm and G, takes the root position, and please see again that the similarity sum is \( f(x) = 1 \), and the stretch is \( g(x) = 17 \). On the other hand, we see that an optimal solution obtained by Algorithm 2 is \( x' = (3, 1, 3, 1) \), which takes the second rotated positions for the first and the third chords. The maximum similarity sum is

\[
f(x') = ||\{8, 13, 17\} \cap \{10, 13, 17\}|| + ||\{10, 13, 17\} \cap \{10, 15, 18\}|| + ||\{10, 15, 18\} \cap \{8, 12, 15\}|| = 2 + 1 + 1 = 4,
\]

and the stretch of the induced machine subsequence is

\[
g(x') = \max(17, 17, 18, 15) - \min(8, 10, 10, 8) + 1 = 18 - 8 + 1 = 11.
\]
Table 1 Correspondence between the mathematical model and the music keyboard play

| Machines | Piano Keys |
|----------|------------|
| machine 1 | C 3        |
| machine 2 | C# 3       |
| machine 3 | D 3        |
| machine 4 | D# 3       |
| machine 5 | E 3        |
| machine 6 | F 3        |
| machine 7 | F# 3       |
| machine 8 | G 3        |
| machine 9 | G# 3       |
| machine 10| A 3        |
| machine 11| A# 3       |
| machine 12| B 3        |
| machine 13| C 4        |
| machine 14| C# 4       |
| machine 15| D 4        |
| machine 16| D# 4       |
| machine 17| E 4        |
| machine 18| F 4        |

Jobs Chord Progression

\[ N = \{1, 2, 3, 4\} \quad \text{←} \quad (C, Am, Dm, G) \]

For any machine subsequence with length ten, there exists a certain job \( j \in N \) such that no machine subsets in the collection \( C_j \) are contained in the machine subsequence, and hence \( g(x^*) = g_{\text{min}} \) holds in this example.

Scores for the two solutions \( x \) and \( x^* \) are shown in Fig. 2. The optimal solution is consistent with a recommended choice of rotated positions for the given chord progression in Takumi and Sakamoto (2018).

4.1.2. Example 2

We give another problem instance with \( m = 18 \), \( n = 4 \), \( p_1 = 3 \), \( p_2 = 2 \), \( p_3 = 1 \) and \( p_4 = 3 \). The correspondence between the eighteen machines and actual piano keys is the same as that in Table 1. In this example, we are given an ordered set \( N = \{1, 2, 3, 4\} \) of four jobs from a chord progression of (C, F, G, C). The given machine subsets are:

\[
\begin{align*}
S_1(1) &= \{1, 5, 8\}, & S_1(2) &= \{5, 8, 13\}, & S_1(3) &= \{8, 13, 17\}, \\
S_2(1) &= \{6, 10, 13\}, & S_2(2) &= \{10, 13, 18\}, \\
S_3(1) &= \{8, 12, 15\}, & S_3(2) &= \{15, 18\}, & S_3(3) &= \{8, 13, 17\}. \\
S_4(1) &= \{1, 5, 8\}, & S_4(2) &= \{5, 8, 13\}, & S_4(3) &= \{8, 13, 17\}.
\end{align*}
\]

When the choice of a position for each of the four chords is the root one, the corresponding feasible solution is again represented by \( x = (1, 1, 1, 1) \). The similarity sum is

\[
f(x) = ||\{1, 5, 8\} \cap \{6, 10, 13\}|| + ||\{6, 10, 13\} \cap \{8, 12, 15\}|| + ||\{8, 12, 15\} \cap \{1, 5, 8\}|| = 0 + 0 + 1 = 1,
\]

and the stretch of the induced machine subsequence is

\[
g(x) = \max\{8, 13, 15, 8\} - \min\{1, 6, 8, 1\} + 1 = 15 - 1 + 1 = 15.
\]

Fig. 2 Scores for two solutions associated with (C, Am, Dm, G)
By Algorithm 2, we obtain an optimal solution $x^* = (2, 1, 1, 2)$. In addition, we notice that $x^* = (3, 2, 1, 3)$ is also an optimal solution if the condition $f(x') \geq f^*$ is put at Line 14 of Algorithm 2, instead of the current one. The maximum similarity sum is

$$f(x^*) = ||(5, 8, 13) \cap [6, 10, 13]| + ||6, 10, 13] \cap [8, 12, 15]| + ||8, 12, 15] \cap [5, 8, 13]|$$

$$= ||[8, 13, 17] \cap [10, 13, 18]| + ||[10, 13, 18] \cap [8, 12, 15]| + ||8, 12, 15] \cap [8, 13, 17]| = 2,$$

and the stretch of the induced machine subsequence is

$$g(x^*) = \max\{13, 13, 15, 13\} - \min\{5, 6, 8, 5\} + 1$$

$$= \max\{17, 18, 15, 17\} - \min\{8, 10, 8, 8\} + 1 = 11 (= g_{\text{min}}).$$

Each optimal solution is consistent with a recommended choice of rotated positions for the given chord progression in Hirasawa (2011).

### 4.1.3. Example 3

We provide the last problem instance of this section with $m = 18$, $n = 8$, $p_1 = 3$, $p_2 = 1$, $p_3 = 1$, $p_4 = 2$, $p_5 = 2$, $p_6 = 3$, $p_7 = 2$, and $p_8 = 1$. In this example, we are given an ordered set $N = \{1, 2, 3, 4, 5, 6, 7, 8\}$ of eight jobs from a chord progression of (C, G, Am, Em, F, C, F, G). The following machine subsets are given:

$$S_{\gamma}(1) = \{1, 5, 8\}, \quad S_{\gamma}(2) = \{5, 8, 13\}, \quad S_{\gamma}(3) = \{8, 13, 17\}$$
$$S_{\zeta}(1) = \{8, 12, 15\},$$
$$S_{\sigma}(1) = \{10, 13, 17\},$$
$$S_{\varphi}(1) = \{5, 8, 12\}, \quad S_{\varphi}(2) = \{8, 12, 17\},$$
$$S_{\psi}(1) = \{6, 10, 13\}, \quad S_{\psi}(2) = \{10, 13, 18\},$$
$$S_{\rho}(1) = \{1, 5, 8\}, \quad S_{\rho}(2) = \{5, 8, 13\}, \quad S_{\rho}(3) = \{8, 13, 17\},$$
$$S_{\theta}(1) = \{6, 10, 13\}, \quad S_{\theta}(2) = \{10, 13, 18\},$$
$$S_{\sigma}(1) = \{8, 12, 15\}.$$

For a feasible solution $x = (1, 1, 1, 1, 1, 1, 1, 1)$, the similarity sum is

$$f(x) = ||(1, 5, 8) \cap [8, 12, 15]| + ||8, 12, 15] \cap [10, 13, 17]| + ||10, 13, 17] \cap [5, 8, 12]|$$
$$+ ||[5, 8, 12] \cap [6, 10, 13]| + ||6, 10, 13] \cap [1, 5, 8]| + ||[1, 5, 8] \cap [6, 10, 13] + ||6, 10, 13] \cap [8, 12, 15]|$$
$$= 1 + 0 + 0 + 0 + 0 + 0 + 0 = 1,$$

and the stretch of the induced machine subsequence is

$$g(x) = \max\{8, 15, 17, 12, 13, 8, 13, 15\} - \min\{1, 8, 10, 5, 6, 1, 6, 8\} + 1 = 17 - 1 + 1 = 17.$$

On the other hand, Algorithm 2 obtains an optimal solution $x^* = (3, 1, 1, 2, 2, 3, 2, 1)$. The maximum similarity sum is

$$f(x^*) = ||[8, 13, 17] \cap [8, 12, 15]| + ||8, 12, 15] \cap [10, 13, 17]| + ||10, 13, 17] \cap [8, 12, 15]|$$
$$+ ||[10, 13, 18] \cap [8, 13, 17]| + ||8, 13, 17] \cap [10, 13, 18] + ||[10, 13, 18] \cap [8, 12, 15]|$$
$$= 1 + 0 + 1 + 0 + 1 + 1 + 0 + 0 = 4,$$

and the stretch of the induced machine subsequence is

$$g(x^*) = \max\{17, 15, 17, 17, 18, 17, 18, 15\} - \min\{8, 8, 10, 8, 10, 8, 10, 8\} + 1 = 18 - 8 + 1 = 11 (= g_{\text{min}}).$$

Scores for the two solutions $x$ and $x^*$ are illustrated in Fig. 3. The transition of the chords in the music keyboard play following the optimal solution $x^*$ looks moderate.
4.2. Randomly Generated Instances

In the remainder of this numerical section, we leave from the application of music keyboard play, and treat the lexicographic machine subset selection problem purely as a mathematical model. First, each collection of fifty instances of the lexicographic machine subset selection problem to be tested is randomly generated as follows:

- The number of machines: \( m = 30 \).
- The number of jobs: \( n \in \{10, 25, 50, 100\} \).
- The number of machine subsets for every job \( i \in N \): Either \( p_j = 5 \) (\( = p \)), or \( p_j = 10 \) (\( = p \)).
- The cardinality of every machine subset (i.e., \( |S_j(k)| \)): \( c = 3 \).
- An upper bound on the length of the machine subsequence induced by a machine subset (i.e., \( b_j(k) - a_j(k) + 1 \)): \( \delta = 10 \). The machine \( a_j(k) \) with the smallest index in a machine subset \( S_j(k) \) is randomly generated as an integer within \( [1, m - \delta + 1] \), and the remaining \( c - 1 \) machines of the \( S_j(k) \) are generated as integers within \( [a_j(k) + 1, a_j(k) + \delta - 1] \), where \( a_j(k) + \delta - 1 \leq m \) holds.

The program is written in C language, and is run on a laptop computer with Windows 7 Professional (64bit), Intel Core i5 4210M CPU (2.60GHz) and 8GB memory. In Tables 2–4, each of the data indicates the mean value over the fifty test instances. In Table 4, each of the data also indicates the mean value over the fifty test instances, while each collection of the fifty instances is repeated with a number of times to measure the execution time of the proposed algorithm, regarding the precision of the clock in the laptop computer.

4.3. Comparisons between Single Criterion and Lexicographic Bi-criteria Solutions

Table 2 shows the objective function values obtained by four algorithms, i.e., procedure DP_{SS}, Algorithm 1, Algorithm 2 and Algorithm 3 (provided in an extra section), for fifty test instances with \( m = 30 \), \( p = 5 \), \( c = 3 \) and \( \delta = 10 \). Of course, the proposed algorithm (i.e., Algorithm 2) attains the maximum similarity sum as procedure DP_{SS} does. Unfortunately, the improvement of the stretch by the proposed algorithm is very small for the randomly generated instances. In particular, the conditionally minimum of the stretch looks \( g' \approx m = 30 \) when \( n \geq 50 \). That is, for an instance with \( n \geq 50 \), the primary objective of the similarity sum seems to invalidate the secondary objective of the stretch. On the other hand, we observe that, e.g., the complementary algorithm (i.e., Algorithm 3) attains \( 43.8/70.4 \times 100 = 62 \% \) of the maximum similarity sum when \( n = 50 \), and the lexicographic solutions with the minimum stretch may be reasonable for the case of \( g' \approx m \gg g_{\text{min}} \). From this, it would be interesting to reformulate the machine subset selection problem as another multiple criteria optimization model, for example, as an \( \varepsilon \)-constrained model, e.g., see Ehrgott (2005), Nishikawa et al. (1982), instead of the lexicographic model.

Table 3 illustrates the objective function values obtained by the four algorithms for fifty test instances with \( m = 30 \), \( n = 50 \), \( c = 3 \) and \( \delta = 10 \), where we increase the (common) option number \( p \) of machine subsets for each job from \( p = 5 \) to \( p = 10 \). As in Table 2, the proposed algorithm obtains the maximum similarity sum \( f^* \), and the complementary algorithm gets the minimum stretch \( g_{\text{min}} \). It is easy to see that both of the similarity sum and the stretch are improved by the proposed algorithm when the option number \( p \) of machine subsets for each job increases.

Table 4 indicates the execution times of procedure DP_{SS}, Algorithm 1, Algorithm 2 and Algorithm 3 for fifty test instances with \( m = 30 \), \( c = 3 \) and \( \delta = 10 \). Recall that all the algorithms run in polynomial time. Also in practice, there
Table 2  Objective function values on test instances with \( m = 30 \), \( p = 5 \), \( c = 3 \) and \( \delta = 10 \).

| Method | DP_SS | Algorithm 1 | Algorithm 2 | Algorithm 3 |
|--------|-------|-------------|-------------|-------------|
| Objective Functions | Similarity Sum \( f^* \) | (Stretch) \( g_{\min} \) | Similarity Sum \( f^* \) | Stretch \( g^* \) | Similarity Sum \( f^* \) | Stretch \( g^* \) |
| \( n = 10 \) | 13.4 | (27.0) | 14.7 | 13.4 | 25.1 | 5.7 | 14.7 |
| \( n = 25 \) | 34.7 | (29.0) | 17.1 | 34.7 | 28.2 | 19.2 | 17.1 |
| \( n = 50 \) | 70.4 | (29.6) | 18.8 | 70.4 | 29.3 | 43.8 | 18.8 |
| \( n = 100 \) | 143.3 | (29.8) | 19.9 | 143.3 | 29.6 | 94.4 | 19.9 |

Table 3  Objective function values on test instances with \( m = 30 \), \( n = 50 \), \( c = 3 \) and \( \delta = 10 \).

| Method | DP_SS | Algorithm 1 | Algorithm 2 | Algorithm 3 |
|--------|-------|-------------|-------------|-------------|
| Objective Functions | Similarity Sum \( f^* \) | (Stretch) \( g_{\min} \) | Similarity Sum \( f^* \) | Stretch \( g^* \) | Similarity Sum \( f^* \) | Stretch \( g^* \) |
| \( p = 5 \) | 70.4 | (29.6) | 18.8 | 70.4 | 29.3 | 43.8 | 18.8 |
| \( p = 10 \) | 90.3 | (27.7) | 14.8 | 90.3 | 25.7 | 71.1 | 14.8 |

Table 4  Execution times [msec] of the four algorithms on test instances with \( m = 30 \), \( c = 3 \) and \( \delta = 10 \).

| Method | DP_SS | Algorithm 1 | Algorithm 2 | Algorithm 3 |
|--------|-------|-------------|-------------|-------------|
| \( n = 25 \) | \( 1.6 \times 10^{-1} \) | \( 5.2 \times 10^{-1} \) | \( 7.6 \times 10^{-1} \) | \( 1.9 \) | \( 1.2 \) | \( 3.6 \) | \( 1.0 \) | \( 2.0 \) |
| \( n = 50 \) | \( 3.3 \times 10^{-1} \) | \( 1.1 \) | \( 1.8 \) | \( 3.8 \) | \( 2.3 \) | \( 7.2 \) | \( 2.0 \) | \( 4.1 \) |
| \( n = 100 \) | \( 6.6 \times 10^{-1} \) | \( 2.1 \) | \( 3.2 \) | \( 7.9 \) | \( 4.5 \) | \( 14.3 \) | \( 4.0 \) | \( 8.4 \) |

seems to be no problem on each computation of the four algorithms for such sizes of the test instances. For further research, it would be interesting to examine a faster implementation of the proposed algorithm as mentioned in the previous section.

5. Conclusions

In this paper, we formulated a combinatorial optimization problem, which was motivated by a musical issue of selecting inversions in a chord progression with the triad. We were primarily asked to choose a machine subset from given at most \( p \) options for each of \( n \) jobs so that the similarity sum over a series of the chosen \( n \) machine subsets is maximized, and were secondarily asked to choose a machine subset from the given at most \( p \) options for each job so that the stretch of the induced machine subsequence is minimized. In this paper, we proposed an \( O(p^2m^2n) \) time exact algorithm for the lexicographic machine subset selection problem. We also demonstrated the solutions obtained by the proposed algorithm on three examples with the triad, and on randomly generated instances. For each of the first two examples of chord progressions with the triad, the proposed algorithm returned an optimal solution which corresponds to a recommended choice of inversions of the given chords in the literature. Also, the proposed algorithm actually run fast.
for randomly generated instances with up to thirty machines, one hundred jobs and ten options of machine subsets.

For future research, it would be significant to evaluate the feeling of something good when listening to a music by another mathematic criteria in order to understand compositional rules of music more comprehensively, although we wonder the direct helpfulness of the proposed algorithm for musicians. Of course, it would also be interesting to examine a faster implementation of the proposed algorithm, and another multiple criteria model of the machine subset selection problem.

Acknowledgment

We are grateful to two anonymous referees for their constructive suggestions which were truly helpful to improve the readability of this paper. We have presented the earlier version of this paper at International Symposium on Scheduling 2019 (ISS 2019), which was sponsored by the Scheduling Society of Japan (SSJ), and was co-sponsored by the Manufacturing Systems Division of the Japan Society of Mechanical Engineers (JSME). Also, thanks are due to Naoyuki Fujita, who is a co-author of the earlier version (Fujita, et al., 2019). He assisted us in music score drawing, and discussed the lexicographic optimization enjoyably with us even after ISS 2019, though he has moved to another musical topic.

In this paper to the special issue of the journal, we enhance the numerical section to include newly the performance results of the proposed algorithm on randomly generated instances, and present a complementary algorithm to the proposed one. The detail of the complementary algorithm is described in the following extra section.

Appendix: Complementary Version of the Problem

First, we briefly describe the complementary version of the lexicographic machine subset selection problem. The instance is the same as that of the original one defined in Section 2.2. That is, we are given a sequence $M = \{1, 2, \ldots, m\}$ of $m$ machines, an ordered set $N = \{1, 2, \ldots, n\}$ of $n$ jobs, a positive integer $p_j$ and a collection $C_j = \{S_j(k) \mid k = 1, 2, \ldots, p_j\}$ of non-empty machine subsets for each job $j \in N$. We express by $p = \max_{j \in N} \{p_j\}$ the maximum cardinality of a collection of machine subsets. Again, let $x = (x_1, x_2, \ldots, x_n)$ denote a feasible solution, where $x_j \in \{1, 2, \ldots, p_j\}$ is met for each job $j \in N$. Also, let $M[a(x), b(x)] \subseteq M$ denote the machine subsequence induced by a feasible solution $x$ such that it satisfies $a(x) = \min \{a_j(x_j) \mid j \in N\}$ and $b(x) = \max \{b_j(x_j) \mid j \in N\}$.

For a feasible solution $x$, we express by $g(x) = b(x) - a(x) + 1$ the stretch of $M[a(x), b(x)]$. In this section, the first objective is to minimize the stretch of an induced machine subsequence, while the second objective is to maximize the similarity sum (see Eq. (2)). As in Section 3.2, let $g_{\min}$ denote the minimum stretch for a given instance. Let $x = \hat{x}$ denote a feasible solution such that it attains the minimum stretch, i.e., $g(\hat{x}) = g_{\min}$, and it also meets $f(\hat{x}) \geq f(x')$ for any feasible solution $x'$ with the minimum stretch $g(x') = g_{\min}$. In this section, let $\hat{f} = f(\hat{x})$ express the conditionally maximum of similarity sum. We refer to such a feasible solution $\hat{x}$ as an optimal one of the complementary version. For an instance, the complementary version of the lexicographic machine subset selection problem asks to find an optimal solution $\hat{x}$.

In a different manner from that in Section 3.3, we combine dynamic programming procedure DP$_{SS}$ provided in Section 3.1 with Algorithm 1 presented in Section 3.2 to address the complementary version. We show the polynomial time exact algorithm as Algorithm 3. The algorithm first tries to find the minimum $g_{\min}$ of stretch by utilizing Observation 1 as Algorithm 1. That is, the algorithm guesses the minimum stretch by the parameter $h$ with $1 \leq h \leq m$, and increases $h$ one by one until a machine subsequence containing at least one machine subset from each collection $C_j$ is found. After finding the minimum stretch $g_{\min}$, we do not need to continue the first for-loop with $h > g_{\min}$, which is the major difference of Algorithm 3 from Algorithm 2. What we only need to do after that is to exploit all the machine subsequences with the minimum stretch $g_{\min}$. The terminating condition to be tested is represented at Line 16. Notice that procedure DP$_{SS}$ is called at most $m - h + 1$ times (see Line 10) only when $h = g_{\min}$ holds in the first for-loop.

The time complexity evaluation for Algorithm 3 is similar to that of Algorithm 1. It takes $O(p)$ time to extract collection $C'_j$ from given collection $C_j$ of $p_j$ machine subsets for each job $j \in N$ (see Line 6). The test at Line 8 obviously requires $O(n)$ time. A feasible solution $x'$ with the minimum stretch $g_{\min}$ is constructed from the extracted collections $C'_1, C'_2, \ldots, C'_n$ in $O(p^2n)$ time by procedure DP$_{SS}$ at Lines 10–12, and the construction is performed at most $O(m)$ times. All the degrees $d_j(f, k)$ of similarity at Line 1 are computed in $O(p^2n) \times O(m) = O(p^2mn)$ time as a preprocessing. Hence, by counting double for-loops at Lines 3–4, the time complexity of Algorithm 3 is evaluated to be

$$O(p^2mn) + O(m^2) \times (O(p) \times O(n) + O(n)) + O(m) \times \{O(p^2n) + O(n)\} = O(p^2mn) + O(m^2) + O(p^2mn) = O(p^2mn).$$
Algorithm 3 (The Complement to the Proposed Algorithm)

Input: A sequence $M$ of $m$ machines, an ordered set $N$ of $n$ jobs, and a collection $C_j = (S_j(1), S_j(2), \ldots, S_j(p_j))$ of machine subsets for each job $j \in N$.

Output: A feasible solution $x = \hat{x}$ such that it attains the minimum stretch $g(\hat{x}) = g_{\text{min}}$ and the conditionally maximum of similarity sum $f(\hat{x}) = \hat{f}$.

1: Prepare all the $O(p^2n)$ degrees $d_i(\ell, k)$ of similarity $(j = 2, 3, \ldots, n, \ell = 1, 2, \ldots, p_j-1$ and $k = 1, 2, \ldots, p_j)$.
2: $b := 1; \quad \hat{f} := -1; \quad */$ Initialize a flag and the conditionally maximum of the similarity sum */
3: for $h = 1$ to $m$ do /* Guess the minimum stretch by the parameter. */
4: for $i = 1$ to $m - h + 1$ do /* Fix the first machine of a machine subsequence with length $h$. */
5: for $j = 1$ to $n$ do /* The best incumbent solution is updated. */
6: $C_j' := \{S_j(k) \in C_j \mid i \leq a_j(k), b_j(k) \leq i + h - 1\};$ /* The last machine in the machine subsequence is $i + h - 1$. */
7: end for
8: if $C_j' \neq \emptyset$ for all $j = 1, 2, \ldots, n$ then
9: $g_{\text{min}} := h; \quad b := 0;$ /* The minimum stretch has been found, and the flag is switched. */
10: Call dynamic programming procedure DP_SS for the extracted instance with machine subsequence $M[i, i + h - 1]$.
11: ordered set $N$ of $n$ jobs, and non-empty collection $C_j'$ for each job $j \in N$;
12: Receive a feasible solution $x' = (x'_1, x'_2, \ldots, x'_n)$ from procedure DP_SS;
13: if $f(x') > \hat{f}$ then
14: $\hat{f} := f(x'); \quad \hat{x} := x'; \quad */$ The best incumbent solution is updated. */
15: end if
16: if $b = 0$ and $i = m - h + 1$ then /* All machine subsequences with length $h = g_{\text{min}}$ have been exploited. */
17: return the solution $\hat{x}$ as an optimal one.
18: end if
19: end if
20: end for /* for $i$ */
21: end for /* for $h$ */

References

Cope, D., Computer Models of Musical Creativity, (2005), MIT Press, Cambridge, MA.
Ehrgott, M., Multicriteria Optimization, (2005), Springer, Berlin.
Fujita, N., Karuno, Y., and Yoshii, R., Selecting a series of machine subsets with moderate changing, Proceedings of the International Symposium on Scheduling 2019 (ISS 2019, SSJ), (2019), pp. 190–195.
Hirasawa, E. (together with Editorial Office of the Magazine), Introduction to Music Composition by Utilizing Standard Chord Progressions (in Japanese), DTM Magazine, Vol. 18, No. 8, (2011), pp. 21–34.
Ibaraki, T., Algorithms and Data Structures in C (in Japanese), (2014), Ohmusha, Tokyo.
Koyano, K., An Understandable Textbook of Music Grammar (in Japanese), (2011), Yamaha Music Media, Tokyo.
Nishikawa, Y., Sannomiya, N., and Ibaraki, T., Optimization (in Japanese), (1982), Iwanami Shoten, Tokyo.
Skiena, S.S., The Algorithm Design Manual, (2008), Springer, London.
Takumi, H. and Sakamoto, T., Compositional Rules (in Japanese), Vol. 3, (2018), Rittor Music, Tokyo.