Praise Worthy Prize

International Review on Computers and Software (I.RE.CO.S.), Vol. 11, N. 12
ISSN 1828-6003 December 2016

Building Melodic Feature Knowledge of *Gamelan* Music Using Apriori Based on Functions in Sequence (AFiS) Algorithm

Khafiizh Hastuti[1,2], Azhari[1], Aina Musdholifah[1], Rahayu Supanggah[3]

Abstract – *Gamelan is a traditional music ensemble from Java, Indonesia, whose melody has characteristics that make the melodic sound of gamelan music easy to recognize. This research aims at building melodic feature knowledge of gamelan music in terms of note sequences rules. The algorithm called AFiS (Apriori based on Functions in Sequence) was also introduced to produce rules by mining the frequent value of note sequences. The basic idea of the AFiS algorithm is to define functions in a sequence, and then to chain the functions based on its position order to identify the support value for each function. The implementation of AFiS algorithm is aimed to define rules of gamelan music melodic feature in terms of ideal note sequences for composition. The evaluation of the accuracy of the note sequences rules is conducted by developing a recommendation system using rules defined in this research. The program is expected to answer correctly to some notes randomly deleted from the sequences. The result shows that the accuracy of the knowledge, and that the note sequences rules of gamelan music based on the correct answer is up to 86.5%. Another evaluation is to find whether the different answers given by the program are accepted as alternative notes to the original notes. This evaluation involved 4 human experts to describe their acceptance of the alternative notes based on the different answers. The result shows that the different notes in 4 of 5 gendings are accepted by the experts as alternative notes.*

***Keywords**: AFiS Algorithm, Sequential Pattern Mining, Gamelan Music*

Nomenclature

A-B-C-D	A concept of *gatra*
D	Database
F	Series of functions
I	Collection of items
i	items
LN	Frequent sequence
Minsup	Minimum support
P	Data partition
PID	Partition ID
PI	Itemsets in a partition
S	Sequence
SID	Sequence ID
TC	Total number of candidate
TF	Total number of function
TS	Total number of sequence
TSI	Total number of itemsets in a sequence

I. Introduction

The use of artificial intelligence in music composition process is known as algorithmic composition, where a certain algorithm is used to automatically create a music composition [1] [2] [3]. Algorithmic composition has been developed since mid-year 1950, when *Lejaren A. Hiller* implemented a computer-generated composition by designing a computer to generate number sequences randomly representing notes, rhythm, tempo, and others, to create *Illiac Suite* [4]. Xenakis was another pioneer work which used stochastic algorithm and Markov model to generate basic materials for music composition [5]. An expert systems approach has been used in *CHORAL* to harmonize *four-part chorales* in the style of *J.S. Bach* by defining 270 rules for chord skeleton, individual melodic lines, and others [6]. Another example of the use of rule-based for the music composition is represented by *Baroc Harmony* which added a pre-defined melody to control the search space [7]. The grammar approach was used by [8] to develop *Improve Generator*, a system that learns the pattern of percussion in live-streaming, and generates an accompaniment track which is improvised in real-time.

Considering the growth of the use of the algorithmic composition approach in many musical genres, the present paper uses the approach for a traditional music orchestra called *karawitan*. *Karawitan* known also as *gamelan* music is a music genre originally coming from Java, Indonesia. *Karawitan* uses *gamelan* as orchestra music instruments, and *gending* as song. The final goal of this research is to develop a rule-based expert system for the automatic *gamelan* music composition. One of the main parts of the system developed in this research is to build a knowledge base for the system to arrange a note sequence that could fit the melodic feature of *gamelan* music. Therefore, this paper proposes a model to build

 DOI: 10.15866/irecos.v11i12.10841

the melodic feature knowledge of *gamelan* music using the sequential pattern mining technique implemented by identifying the association among notes in a sequence. It also introduces an algorithm, called *AFiS* (*Apriori based on Functions in Sequence)*, as a sequence data mining algorithm used to analyze the sequential patterns of the *gamelan* song notes.

II. Related Researches

One of the researches in algorithmic composition is a phenomenal work conducted by [5], which used an expert systems approach to develop a system called *CHORAL*. The system aimed at harmonizing *four-part chorales* in the style of *Johann Sebastian Bach*. The knowledge models of the *CHORAL* system used 270 rules consisting of: *the chord skeleton* which controls the rules to generate a new key; *the fill-in* which observes the chorale as four interacting automata, and read the output of the chord skeleton for enumerating the possible inessential note patterns; *the time-slice* which controls the constraint about the consecutive octaves and fifths; *the melodic string* which observes the sequence of individual notes of the different voices from a purely melodic point of view; *the merged melodic* string which merges the repeated adjacent pitches into a single note; *The Schenkerian analysis* based on formal theory of hierarchical voice leading to controls the rules of the possible parser actions, a constraint about the key of the chorale, and a heuristic for proper recognition of a *Schenkerian D-C-B-C* ending pattern.

An application based on L-system for music generation was developed by [9] by rendering the contour of music using F which denotes the generation of a note, f, a pause, and + versus -, the increase, versus the decrease of the pitch of the played note. In the conclusion of the experiment, [9] stated that L-system could not be a first choice for the direct composition of traditional music, but could be ideal for the composition of interactive music.

A class of generative grammars called Probabilistic Temporal Graph Grammars (PTGG) implemented in Haskell was proposed by [10]. This class aimed at handling the temporal aspect of the music in a way that could retain a coherent metrical structure. The concept of PTGG is described as: The grammar generates some sequences of duration-parameterized abstract chord. Chords function as both terminals and non-terminals; ignoring the duration in which the context of a chord does not effect the productions that may be applied to it. Rules are functions on the duration of their input symbol. A system called *Kulitta* proposed by [11] used a Haskell-based framework for automated and algorithmic music composition. The modules of *Kulitta* are divided into 3 main categories: abstract/structural generation which contains a collection of schenkerian-inspired models and algorithms for iteratively generating harmonic structure or other abstract; musical interpretation which contains mathematical models and constraint satisfaction algorithms for turning the abstract musical features into more detailed music; *learning* which contains offline learning algorithms to derive the production probabilities for musical grammars. *Kulitta* features a category of grammars called *Probabilistic Temporal Graph Grammars* (PTGGs) which parameterize the harmonic symbols in duration, and the rules are the functions on that duration.

Other examples of works in automatic music generation are focused on more specific tasks with reductionist approach, such as in [12] [13]. The re-orchestration of Beethoven’s *Ode to Joy*, the European anthem, in seven styles of *Bach chorales, bossa nova, jazz, lounge, penny lane, chi mai, prayer in C*, was served using machine learning approach [12]. There were some different approaches in generating different style of music, for instance, in Bach chorales style, the sequence to be generated by a probability distribution is represented using the max entropy model, while the fioriture-based Markov model is used in jazz style. A Natural Language Processing technique is used by [13] for the automatic reduction of melodies using the Probabilistic Context-Free Grammar (PCFG). The PCFG for melodic reduction tasks is trained using supervised learning, and the three analyses provided by the Generative Theory of Tonal Music (GTTM) dataset is used for the learning process. The performance of the melodic reduction includes a melodic identification and similarity, an efficient storage of melodies, the automatic composition, the variation matching, and the automatic harmonic analysis. The natural language text was used to generate music automatically by [14]. A sequence of notes, as well as the rhythm, was generated using text input mapped using vowels in conjunction with the word's part of speech (POS).

The learning technique, probability and statistical analysis were used to generate new music, never heard before, by [15]. Note sequences and other musical parameters such as note length, pitch, accidentals, modification (intensity, speed), and note sequence repetition density were used for the preparation of a probability table that could generate new music. There are three phases to generate new music: Music database creation that stores note identifier of the current note, the note identifier of the previous note, current sequence identifier; Learning and analysis phase based on thematic piece (i.e. rock, classical, instrumental, etc) using statistical analysis to produce a generalized rule-based matrix of note and sequence identifiers for creating new pieces; Music generation phase that generate new music based on the matrix created during statistical analysis.

In *gamelan* music, there are skeleton notes called *balungan*. The *gamelan* composer usually creates the skeleton notes firstly, before completing the composition. The note sequence in a *gending* is arranged based on *gatra*. *Gatra* is the smallest unit that consists of 4 *sabetan balungan* (beats). The research in *gamelan* music conducted by [16] used a grammar approach to define the rules of *gending* (*gamelan* song) composition in a style of *gamelan* music called *srepegan*, by identifying the contour of note sequences. The pattern of *srepegan* is formulated based on *gatra(s)*, which are the smallest unit consisting of 4 beats, and is served in term of a contour which represents the notes scale, where the frequent patterns are identified by higher or lower than the previous note. Another approach was used by [17] by building a frame work of quasi-linguistic approach to describe the structure and content of a style of *gamelan* music called *gending lampah* by defining rules of composition according to *gending lampah*. The grammar consists of the base rules, which are common rules that control the structure of *gending lampah*, the contour assignment rules which control the pitch assignment, the restriction rules which restrict the range of choice of the transformation rules, the transformation rules which control the choice of the notes, and their arrangement, the derivation rules which set up to generate a *gending lampah*, the *gong* assignment rules which controls the assignment of *gong* which is a gamelan instruments. Both these researches used frequent pattern mining to define the rules of a particular style of *gamelan music*. Although the subject of research was different in style or genre, the contour pattern proposed by [16], and quasi-linguistic approach proposed by [17] involved the melodic feature of *gamelan* music as a part of the component controlled by the rules. Instead of using frequent pattern mining, this research preferred to use a sequential pattern mining in building melodic feature knowledge of *gamelan* music. This technique was expected to result in rules which could more generally be applied to *gamelan* music.

Trends of sequential patterns mining for big data have grown rapidly, such as in the works by [18] which developed *Peekquence*; a sequential pattern mining approach to explore event sequences in big data, which used users relevant metrics to sort patterns, to integrate patterns with patient time lines, and to overview and summarize the most common events in patterns. The traditional approaches of sequential patterns mining are considered as a design which does not support a massive amount of data. However traditional approaches of sequential pattern mining still need continuous research.

Kamber et al in [19] stated that sequential patterns are multidimensional association rules; therefore there are different specialized ways to find sequential patterns. Further, it is mentioned that some different approaches in mining sequential patterns are Apriori-based algorithm GSP, SPAM, projection-based FreeSpan, PSPM, vertical data format based algorithm SPADE, and pattern growth based approach in UDDAG.

The sequential patterns mining algorithm is categorized based on its structure by the type of Apriori-based and Pattern Growth-based [20] [21] [22] dataset. Apriori-based algorithm, such as AprioriAll, AprioriSome, DynamicSome, GSP (Generalized Sequential Patterns), SPADE (Sequential Pattern Discovery using Equivalence classes), SPAM (Sequential PAttern Mining), has typically been used to discover intra-transaction associations and generate rules about the discovered associations, the example algorithm; Pattern Growth Algorithms, such as FreeSpan, and PrefixSpan, remove the process of candidate generation and prune steps that occur in the Apriori-type algorithms. The pattern growth algorithms can be faster when given large volumes of data; Temporal sequences, such as WINEPI, MINEPI, PROWL, allows the data used for sequence mining not to be limited to data stored in overtly temporal or longitudinally maintained datasets [21].

P-Prefix Span (Percussive-Prefix Sequential pattern) algorithm is used by [23] to integrate the genetic information and discover complex diseases such as Type-2 Diabetes Mellitus. The P-Prefix Span algorithm works by discovering the length of sequential pattern and counting the support for all gene sequences to generate interesting patterns which satisfy the conditions. The DBP-SPAM algorithm which employs direct bit position manipulation technique is proposed by [24]. The frequent itemsets are discovered by bit position pruning technique.

There are three features in the taxonomy of Apriori-based algorithm defined by [25]: Breadth-first search, where the algorithm construct all *k*-sequences together in each k_{th} iteration of the algorithm as they traverse the search space; Generate-and test, in which the pattern is simply grown one item at a time and tested against the minimum support; Multiple scans of the database that scan the original database to ascertain whether a long list of generated candidate sequences is frequent or not.

In this research a new algorithm for sequential pattern mining is proposed, in which the Apriori-based is used to construct procedures of the algorithm.

III. Proposed Work

A sequential pattern mining technique is used to build melodic feature knowledge of *gamelan* music which is able to represent the *A-B-C-D* concept of *gamelan* music. We use the technique to formulate an appropriate note sequence by finding the frequent notes of a function that implies the existence of the frequent notes of the following function. The use of functions in a sequence is considered for mining a sequential pattern. In this research, the algorithm called *Apriori based on Function in Sequence* (*AFiS*) is proposed.

The AFiS algorithm works based on functions in a sequence. A function contains an item based on its order. The functions are then chained in terms of sequential patterns. Finally, the rules are defined based on frequent sequences. The AFiS algorithm has seven phases in mining sequence data: function definition, data partition, sequential pattern creation, candidate selection, support counting, prune phase, and production rules.

In this explanation, there is database *D*, which contains sequences (S), where each *S* consists of an ordered list of itemset $<s_1, s_2, ..., s_n>$, and the *s* sequence contains items. We denote the collection of items by *I*, where $I = \{i_1, i_2, ..,i_n\}$. We use a sequence database below, as an example in explaining the implementation of AFiS algorithm (Fig. 1).

SID	Transaction
01	c, f, a, b, e, c, b, a, c, d, a
02	f, b, d, a, c, a, c, f
03	b, e, d, e, f, a

Fig. 1. Sequence database

Based on the explanation above, then:

I ={a, b, c, d, e, f}
D = {SID01, SID02, SID03}
SID01 = <c, f, a, b, e, c, b, a , c, d, a>
SID02 = <f, b, d, a, c, a, c, f>
SID03 = <b, e, d, e, f, a>

III.1. Function Definition

The procedure of the AFiS algorithm started by defining the functions in a sequence. The function can be time series, procedures, or just function without any labels. The time series can contain data of days, months, or years, with value of weather, selling of goods, or others. The procedures can be a concept, such as the concept of *A-B-C-D* in *gamelan* music implemented in this research, with notes as the value.

The number of function is determined based on each case. *F* denotes a series of functions, and *TF* denotes the total number of functions, then $F = (F_1, F_2, ..., F_n)$, and the total number of functions is the length of *F* (*TF* = *F.Length*). For an example, define 3 functions in a sequence, then: $F = \{F_1, F_2, F_3\}$, and *TF*= 3.

III.2. Data Partition

The second phase is data partition, where each sequence is partitioned with multiples of *TF*. The data partition for each sequence can be formulated as the pseudo-code of the data partition below:

```
S       : sequence
TSI     : total number of itemsets in a
          sequence
TF      : total number of functions
P       : data partition

n = 0
While ( n < (TSI / TF ) ) {
      P [n] = [ ]
      n++
}

For ( n = 0; n < TSI; n++ ){
    For ( k = 0; k < TF; k++){
        P [n] [k] = S [ (k*TF) + n ]
    }
}
```

The value of *TF* is 3, then the *SID01* containing 11 items will have 3 partitions; *SID02* will have 3 partitions; *SID03* will have 2 partitions, as shown in Fig. 2, with partition ID abbreviated as *PID*.

SID	PID	P
01	01	<c, f, a>
	02	<b, e, c>
	03	<b, a, c>
	04	<d, a>
02	01	<f, b, d>
	02	<a, c, a>
	03	<c, f>
03	01	<b, e, d>
	02	<e, f, a>

Fig. 2. Result of data partition with TF = 3

III.3. Sequential Pattern Creation

The sequential patterns are created by chaining functions. Each function is filled with itemsets from the sequences based on their order in the data partition. This procedure makes the first function is filled with itemsets which are in the first order of the partition, the second function is filled with itemsets which are in the second order of the partition, and so on.

There are 3 functions of *F1*, *F2*, *F3* and 3 sequences which have been partitioned. The result of itemsets defining of each function can be seen in Fig. 3.

SID	PID	P	→	F1	F2	F3
01	01	<c, f, a>		c	f	a
	02	<b, e, c>		b	e	c
	03	<b, a, c>		b	a	c
	04	<d, a>		d	a	-
02	01	<f, b, d>		f	b	d
	02	<a, c, a>		a	c	a
	03	<c, f>		c	f	-
03	01	<b, e, d>		b	e	d
	02	<e, f, a>		e	f	a

Fig. 3. The result of defining the itemsets of each function

The process of defining itemsets of each function based on partition data can be formulated as the pseudocode below:

```
F       : functions
P       : data partition
TP      : total number of partitions

n = 0
while (n < F.length){
    for ( k = 0; k < TP; k++ ){
        F [n] [k] = P [k] [n]
    }
    n++
}
```

The sequential patterns are built by chaining functions. Since chaining functions requires at least 2 functions, then the sequential patterns started with 2-

itemsets up to (*TF-1*)-itemsets. There are 3 functions defined in this example, thus there are 2sequential patterns containing 2-itemsets, and 3-itemsets.

The sequential pattern of 2-itemsets consists of <F1, F2>, <F2, F3>, <F3, F1*>, with asterisk symbols standing for the next partition. The sequential pattern of 3-itemsets consists of <F1, F2, F3>, <F2, F3, F1*>, <F3, F1*, F2*>. Fig. 4 shows the itemsets of the sequential patterns of 2-itemsets, and 3-itemsets.

Itemsets of Sequential Pattern of 2-Itemsets

SID	PID	<F1, F2>	<F2, F3>	<F3, F1*>
01	01	<c, f>	<f, a>	<a, b>
	02	<b, e>	<e, c>	<c, b>
	03	<b, a>	<a, c>	<c, d>
	04	<d, a>	<a >	-
02	01	<f, b>	<b, d>	<d, a>
	02	<a, c>	<c, a>	<a, c>
	03	<c, f>	<f>	-
03	01	<b, e>	<e, d>	<d, e>
	02	<e, f>	<f, a>	<a>

Itemsets of Sequential Pattern of 3-Itemsets

SID	PID	<F1, F2, F3>	<F2, F3, F1*>	<F3, F1*, F2*>
01	01	<c, f, a>	<f, a, b>	<a, b, e>
	02	<b, e, c>	<e, c, b>	<c, b, a>
	03	<b, a, c>	<a, c, d>	<c, d, a>
	04	<d, a>	<a >	-
02	01	<f, b, d>	<b, d, a>	<d, a, c>
	02	<a, c, a>	<c, a, c>	<a, c, f>
	03	<c, f>	<f>	-
03	01	<b, e, d>	<e, d, e>	<d, e, f>
	02	<e, f, a>	<f, a>	<a>

Fig. 4. Sequential patterns created by chaining the functions

The next phase is candidate selection, where the itemsets with a length not equal to the length of the sequential pattern (k-itemsets) is eliminated from the list. For example, itemsets <a> in sequential pattern of 2-itemsets <F2, F3> is eliminated from the list. Fig. 5 shows the results of candidate selection.

Candidates of 2-Itemsets

SID	<F1, F2>	<F2, F3>	<F3, F1*>
01	<c, f>	<f, a>	<a, b>
	<b, e>	<e, c>	<c, b>
	<b, a>	<a, c>	<c, d>
	<d, a>	-	-
02	<f, b>	<b, d>	<d, a>
	<a, c>	<c, a>	<a, c>
	<c, f>	-	-
03	<b, e>	<e, d>	<d, e>
	<e, f>	<f, a>	-

Candidates of 3-Itemsets

SID	<F1, F2, F3>	<F2, F3, F1*>	<F3, F1*, F2*>
01	<c, f, a>	<f, a, b>	<a, b, e>
	<b, e, c>	<e, c, b>	<c, b, a>
	<b, a, c>	<a, c, d>	<c, d, a>
02	<f, b, d>	<b, d, a>	<d, a, c>
	<a, c, a>	<c, a, c>	<a, c, f>
03	<b, e, d>	<e, d, e>	<d, e, f>
	<e, f, a>	-	-

Fig. 5. Candidates of sequential patterns

III.4. Support Counting

The sequence defined as frequent is measured using minimum support value. An itemset must have at least 1 transaction to be defined as frequent, if the given minimum support is 1. The pseudocode below is used to find frequent sequence:

```
LN      : frequent sequences
C       : sequent itemsets
minsup  : minimum support

for ( k = 0; k < TC; k++ ){
    if( C [k] >= minsup ){
        LN.push ( s [k] )
    }
}
```

For an example, if the given minimum support is 1, then all candidates are frequent.

After counting the support, the next step is defining the dominant predicate for the itemset by measuring its weight. The formula below is used to measure the weight of the itemset, where *W* stands for weight, *TI* stands for the total number of itemsets for each pattern, and *TC* stands for the total number of candidates:

$$W = \frac{TI}{TC} \tag{1}$$

Fig. 6 and Fig. 7 show the weight of sequence of 2-itemset and 3-itemsets.

Weight of 2-Itemsets <F1, F2>

SID	I	TI	TC	W
01	<c, f>	1	4	0.25
	<b, e>	1	4	0.25
	<b, a>	1	4	0.25
	<d, a>	1	4	0.25
02	<f, b>	1	3	0.33
	<a, c>	1	3	0.33
	<c, f>	1	3	0.33
03	<b, e>	1	2	0.5
	<e, f>	1	2	0.5

Weight of 2-Itemsets <F2, F3>

SID	I	TI	TC	W
01	<f, a>	1	3	0.33
	<e, c>	1	3	0.33
	<a, c>	1	3	0.33
02	<b, d>	1	2	0.5
	<c, a>	1	2	0.5
03	<e, d>	1	2	0.5
	<f, a>	1	2	0.5

Weight of 2-Itemsets <F3, F1*>

SID	I	TI	TC	W
01	<a, b>	1	3	0.33
	<c, b>	1	3	0.33
	<c, d>	1	3	0.33
02	<d, a>	1	2	0.5
	<a, c>	1	2	0.5
03	<d, e>	1	1	1

Fig. 6. Weight of the sequences of 2-itemsets

Weight of 3-Itemsets <F1, F2, F3>

SID	I	TI	TC	W
01	<c, f, a>	1	3	0.33
	<b, e, c>	1	3	0.33
	<b, a, c>	1	3	0.33
02	<f, b, d>	1	2	0.5
	<a, c, a>	1	2	0.5
03	<b, e, d>	1	2	0.5
	<e, f, a>	1	2	0.5

Weight of 3-Itemsets <F2, F3, F1*>

SID	I	TI	TC	W
01	<f, a, b>	1	3	0.33
	<e, c, b>	1	3	0.33
	<a, c, d>	1	3	0.33
02	<b, d, a>	1	2	0.5
	<c, a, c>	1	2	0.5
03	<e, d, e>	1	1	1

Weight of 3-Itemsets <F3, F1*, F2*>

SID	I	TI	TC	W
01	<a, b, e>	1	3	0.33
	<c, b, a>	1	3	0.33
	<c, d, a>	1	3	0.33
02	<d, a, c>	1	2	0.5
	<a, c, f>	1	2	0.5
03	<d, e, f>	1	1	1

Fig. 7. Weight of the sequences of 3-itemsets

The next step is concatenating the frequent sequences to measure their total weight. For the itemsets containing the same items, their weights are summed up. Then, the weight of each itemset is divided into the total number of sequences. The formula below is used to measure the total weight of the itemset, where *TW* stands for total weight, *W* stands for the weight of itemsets, and *TS* stands for the total number of sequences:

$$TW = \frac{W}{TS} \tag{2}$$

Fig. 8 shows the total weight of the sequences of 2-itemsets and 3-itemsets.

Total weight of 2-Itemsets <F1, F2>

I	W	SW	TW
<a, c>	0.33	0.33	0.110
<b, a>	0.25	0.25	0.083
<b, e>	0.5	0.75	0.250
	0.25		
<c, f>	0.33	0.58	0.193
	0.25		
<d, a>	0.25	0.25	0.083
<e, f>	0.5	0.5	0.167
<f, b>	0.33	0.33	0.110

Total Weight of 2-Itemsets <F2, F3>

I	W	SW	TW
<a, c>	0.33	0.33	0.110
<b, d>	0.5	0.5	0.167
<c, a>	0.5	0.5	0.167
<e, d>	0.5	0.5	0.167
<e, c>	0.33	0.33	0.110
<f, a>	0.5	0.8.3	0.277
	0.33		

Total Weight of 2-Itemsets <F3, F1*>

I	W	SW	TW
<a, b>	0.33	0.33	0.110
<a, c>	0.5	0.5	0.167
<c, b>	0.33	0.33	0.110
<c, d>	0.33	0.33	0.110
<d, a>	0.5	0.5	0.167
<d, e>	1	1	0.333

Total Weight of 3-Itemsets <F1, F2, F3>

I	W	SW	TW
<a, c, a>	0.5	0.5	0.167
<b, a, c>	0.33	0.33	0.110
<b, e, c>	0.33	0.33	0.110
<b, e, d>	0.5	0.5	0.167
<c, f, a>	0.33	0.33	0.110
<e, f, a>	0.5	0.5	0.167
<f, b, d>	0.5	0.5	0.167

Total Weight of 3-Itemsets <F2, F3, F1*>

I	W	SW	TW
<a, c, d>	0.33	0.33	0.110
<b, d, a>	0.5	0.5	0.167
<c, a, c>	0.5	0.5	0.167
<e, c, b>	0.33	0.33	0.110
<e, d, e>	1	1	0.333
<f, a, b>	0.33	0.33	0.110

Total Weight of 3-Itemsets <F3, F1*, F2*>

I	W	SW	TW
<a, b, e>	0.33	0.33	0.110
<a, c, f>	0.5	0.5	0.167
<c, b, a>	0.33	0.33	0.110
<c, d, a>	0.33	0.33	0.110
<d, a, c>	0.5	0.5	0.167
<d, e, f>	1	1	0.333

Fig. 8. Total weight of the 2-itemsets and 3-itemsets

III.5. Prune Phase

Prune phase is used to chain itemsets based on functions. For example, there is an itemset <a, c> in the function <F1, F2>, so the itemset in the function <F2, F3> must start with itemset *c*, such as <c, a>, and the itemset in the function <F3, F1*> must start with item a, such as <a, b>, and <a, c>. The last itemset in the itemset of <F3, F1*> is used as a reference to set the first item in the <F1, F2> itemset. Fig. 9 and Fig. 10 show the results of the prune phase.

Prunes of 2-Itemsets

F1	<F1, F2>		<F2, F3>		<F3, F1*>		F1
	I	TW	I	TW	I	TW	
a	<a, c>	0.110	<c, a>	0.167	<a, b>	0.110	b
					<a, c>	0.167	c
b	<b, a>	0.083	<a, c>	0.110	<c, b>	0.110	b
					<c, d>	0.110	d
	<b, e>	0.250	<e, c>	0.110	<c, b>	0.110	b
					<c, d>	0.110	d
			<e, d>	0.167	<d, a>	0.167	a
					<d, e>	0.333	e
c	<c, f>	0.193	<f, a>	0.277	<a, b>	0.110	b
					<a, c>	0.167	c
d	<d, a>	0.083	<a, c>	0.110	<c, b>	0.110	b
					<c, d>	0.110	d
e	<e, f>	0.167	<f, a>	0.277	<a, b>	0.110	b
					<a, c>	0.167	c
f	<f, b>	0.110	<b, d>	0.167	<d, a>	0.167	a
					<d, e>	0.333	e

Fig. 9. Prunes of 2-itemsets

Prunes of 3-Itemsets

<F1, F2>	<F1, F2, F3>		<F2, F3, F1*>		<F3, F1*, F2*>		<F1, F2>
	I	TW	I	TW	I	TW	
ac	<a, c, a>	0.167	<c, a, c>	0.167	<a, c, f>	0.167	cf
ba	<b, a, c>	0.110	<a, c, d>	0.110	<c, d, a>	0.110	da
be	<b, e, c>	0.110	<e, c, b>	0.110	<c, b, a>	0.110	ba
	<b, e, d>	0.167	<e, d, e>	0.333	<d, e, f>	0.333	ef
cf	<c, f, a>	0.110	<f, a, b>	0.110	<a, b, e>	0.110	be
ef	<e, f, a >	0.167					
fb	<f, b, d>	0.167	<b, d, a>	0.167	<d, a, c>	0.167	ac

Fig. 10. Prunes of 3-itemsets

III.6. Production Rules

The rules are set based on the prunes.For example, the itemset <a, c> in the function <F1, F2> chains to the itemset <c, a> in the function <F2, F3>, and it continues to chain to itemsets<a, b>, and <a, c> in the function <F3, F1*>.The itemset <b> in <a, b> of the function <F3, F1*>is then chained to itemsets <b, a>, <b, e> of the function <F1, F2>. This is also done to the itemset<c> in <a, c> of the function <F3, F1*>, where it can be chained to the itemset <c, f> of the function <F1, F2>. Below are the examples of the 2-itemsets rules set based on the prunes:

Start
IF F1 is *a*
THEN F2 is *c*

IF F1 is *b*
THEN F2 is *a* **OR** *e*

...

IF F1 is *d*
THEN F2 is *a*

IF F1 is *e*
THEN F2 is *f*

IF F1 is *f*
THEN F2 is *b*
...

IF F2 is *e*
THEN F3 is *c***OR** *d*

IF F2 is *f*
THEN F3 is *a*

IF F3 is *a*
THEN F1* is *b***OR** *c*
...

IF F3 is *d*
THEN F1* is *d* **OR** *e*
End

The next examples are the 3-itemsets rules set based on the prunes.

Start
IF F1 is *a*
THEN F2 is *c*
ANDF3 is *a*

IF F1 is *b* **AND** F2 is *a*
THEN F3 is *c*

IF F1 is *b* **AND** F2 is *e*
THEN F3 is *c* **OR***d*
...
IF F1 is *f* **AND** F2 is *b*
THEN F3 is *d*
...
IF F3 is *a* **AND** F1* is *c*
THEN F2* is *f*
...
IF F3 is *d* **AND** F1* is *a*
THEN F2* is *c*

IF F3 is *d* **AND** F1* is *e*
THEN F2* is *f*

End

IV. Simulation Work and Analysis

AFiS algorithm is implemented to build melodic feature knowledge of *gamelan* music, and to set rules of note sequences. The dataset is limited to *gending ladrang laras slendro pathet nem*. 15 *gendings* entitled *Konda, Rangsang, Plupuh, Erang-Erang, Jong Layar, Mangu, Peksi Kuwung, Sobah, Medang Miring, Lung Gadung Pel, Kembang Gadhung Ngayangan, Alas Kobong, Binar, Kandha,Liwung* were collected as a dataset. The discussion below describes the implementation of AFiS algorithm procedures in building melodic feature knowledge, and setting rules of note sequences in *gamelan* music.

Most of the *gendings* collected to be used as dataset have *gatra*s with *balungan nibani* types. According to the characteristics of *balungan nibani*, most *gatra*s use *pins* (dot notes) in their note sequences. For example, a *gending* entitled *Medang Miring* seen in Fig. 11, shows that pins are used in most *gatra*s, and only 2 *gatra*s which have type of *balungan mlaku* (without pins in their note sequences).

Ladrang ‘Medang Miring
Laras Slendro Pathet Nem
. 3 . 2   . 3 . 5
. 3 . 2   . 3 . 5
. 1 . 2   . 1 . 6
2 1 6 1   6 5 3 5

Fig. 11. Pins in note sequences

The analysis is to find the causality correlation among note sequences, and pins are not included as note sequences. The type of *balungan nibani* inserting pins into a *gatra* cannot be used as a dataset. Data cleansing is conducted by converting *balungan nibani* into *balungan mlaku*. Pins are removed from note sequences, and new *gatra*s are constructed through a concatenating order.The process of data cleansing is implemented to all 15 gendings used as dataset. Fig. 12 shows the new*gatra*s used as dataset in gending entitled *‘Medang Miring’*.

Ladrang ‘Medang Miring’
Laras Slendro Pathet Nem

3 2 3 5   3 2 3 5
1 2 1 6   2 1 6 1
6 5 3 5

Fig. 12. New *gatra*s from conversion process

As the first phase of the procedure of AFiS algorithm, after the data cleansing process, the functions in a note sequence are defined. The note sequence in a *gending* is arranged based on *gatra*. *Gatra* is the smallest unit consisting of 4 *sabetan balungan* (beats). Fig. 13 shows an illustration of *gatra*, with *sabetan balungan* symbolized as *A, B, C, D*.

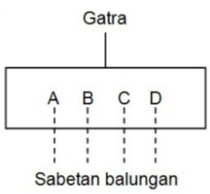

Fig. 13. Illustration of *gatra*

The concept of *gatra* is known as the *A-B-C-D* concept:*A* represents *maju* (forward), *B* represents *mundur* (back), *C* represents *maju* (forward), and *D* represents *seleh* (terminal/the end point of a journey). This concept implies the existence of a hierarchy of functions of each part of the *gatra*, where *D* is the musical point reference or the strongest part, *B* is the second strongest part, *A* is the third strongest part, and *C* is the weakest part.

The concept of *A-B-C-D* in a *gatra* is used as a reference in constructing the functions, where the first note represents function *A*, the second note represents function *B*, the third note represents function *C*, and the fourth note represents function *D*.

Data partition is used to adjust the notes to the functions in sequences. There are 4 functions used in this analysis. Note sequences data are partitioned with length at 4, as the length of the number of functions as a representation of the concept of *A-B-C-D*. After conducing data partition using formula 1, each partition contains 4 notes, where each note fills the value of the function based on its sequence to create sequential patterns.

Data partition and sequential patterns creation are implemented to all 15 *gendings* in the dataset. Table I shows the example of data partition and sequential patterns creation of a *gending* entitled *Konda*.

TABLE I
EXAMPLE OF DATA PARTITION AND SEQUENTIAL PATTERNS CREATION OF A GENDING ENTITLED KONDA

NO	DATA PARTITION	A	B	C	D
1	<3, 5, 3, 2>	3	5	3	2
2	<6, 1, 3, 2>	6	1	3	2
3	<3, 2, 3, 2>	3	2	3	2
4	<6, 5, 3, 2>	6	5	3	2
5	<3, 5, 3, 2>	3	5	3	2
6	<3, 2, 1, 6>	3	2	1	6
7	<3, 2, 1, 6>	3	2	1	6
8	<3, 5, 3, 2>	3	5	3	2
9	<3, 5, 3, 2>	3	5	3	2
10	<5, 6, 3, 5>	5	6	3	5
11	<6, 5, 6, 5>	6	5	6	5
12	<1, 6, 3, 2>	1	6	3	2
13	<3, 5, 3, 2>	3	5	3	2
14	<5, 3, 1, 6>	5	3	1	6
15	<3, 2, 1, 6>	3	2	1	6
16	<3, 5, 3, 2>	3	5	3	2

The notes sequences data which have been partitioned based on the *A-B-C-D* function, and defined as sequential patterns, are then constructed to chain functions.

The sequential patterns started with 2-itemsets up to 4-itemsets. The sequential pattern of 2-itemsets consists of function chaining of <A, B>, <B, C>, <C, D>, <D, A*>, with asterisk symbols that stand for the next partition. The sequential pattern of 3-itemsets consists of function chaining of <A, B, C>, <B, C, D>, <C, D, A*>, <D, A*, B*>.

The sequential pattern of 4-itemsets consists of function chaining of <A, B, C, D>, <B, C, D, A>, <C, D, A*, B*>, <D, A*, B*, C*>.

The process of creating sequential patterns is implemented to all 15 *gendings* in the dataset.

The following tables show examples of sequential patterns of 2-itemsets, 3-itemsets, and 4-itemsets of a *gending* entitled *Konda*.

TABLE II
EXAMPLE OF SEQUENTIAL PATTERN OF 2-ITEMSETS OF A GENDING ENTITLED KONDA

<A, B>	**<B, C>**	**<C, D>**	**<D, A*>**
<3, 5>	<5, 3>	<3, 2>	<2, 6>
<6, 1>	<1, 3>	<3, 2>	<2, 3>
<3, 2>	<2, 3>	<3, 2>	<2, 6>
<6, 5>	<5, 3>	<3, 2>	<2, 3>
<3, 5>	<5, 3>	<3, 2>	<2, 3>
…	…	…	…
<3, 5>	<5, 3>	<3, 2>	<2>

TABLE III
EXAMPLE OF SEQUENTIAL PATTERNS OF 3-ITEMSETS OF A GENDING ENTITLED KONDA

<A, B, C>	**<B, C, D>**	**<C, D, A*>**	**<D, A*, B*>**
<3, 5, 3>	<5, 3, 2>	<3, 2, 6>	<2, 6, 1>
<6, 1, 3>	<1, 3, 2>	<3, 2, 3>	<2, 3, 2>
<3, 2, 3>	<2, 3, 2>	<3, 2, 6>	<2, 6, 5>
<6, 5, 3>	<5, 3, 2>	<3, 2, 3>	<2, 3, 5>
<3, 5, 3>	<5, 3, 2>	<3, 2, 3>	<2, 3, 2>
…	…	…	…
<3, 5, 3>	<5, 3, 2>	<3, 2>	<2>

TABLE IV
EXAMPLE OF SEQUENTIAL PATTERNS OF 4-ITEMSETS OF A GENDING ENTITLED KONDA

<A, B, C, D>	**<B, C, D, A*>**	**<C, D, A*, B*>**	**<D, A*, B*, C*>**
<3, 5, 3, 2>	<5, 3, 2, 6>	<3, 2, 6, 1>	<2, 6, 1, 3>
<6, 1, 3, 2>	<1, 3, 2, 3>	<3, 2, 3, 2>	<2, 3, 2, 3>
<3, 2, 3, 2>	<2, 3, 2, 6>	<3, 2, 6, 5>	<2, 6, 5, 3>
…	…	…	…
<3, 5, 3, 2>	<5, 3, 2>	<3, 2>	<2>

The next phase is candidate selection, where the itemsets with a length not equal to the length of its sequential pattern are eliminated.

For example, in the SID 01, the itemsets <2> in 2-itemsets <D, A*>, <3, 2> in 3-itemsets <C, D, A*>, <2> in the 3-itemsets <D, A*, B*>, <5, 3, 2> in the 4-itemsets <B, C, D, A*>, <3, 2> in the 4-itemsets <C, D, A*, B*>, and <2> in the 4-itemsets <D, A*, B*, C*>, are eliminated.

The candidate selection process is implemented to all 15 *gendings* in the dataset.

The itemsets in all functions of each candidate are then measured using minimum support value. The minimum support is defined at 1.

The results show that all the candidates of all 15 *gendings* are defined as frequent sequences, and then the weight of the frequent itemsets in all functions is measured using formula (1).

The concatenation process is then implemented to the itemsets in all 15 *gendings* based on each function. Then, the concatenation itemsets are measured for their total weight using formula (2).

Tables below show the frequent sequences of 2-itemsets, 3-itemsets, and 4-itemsets resulted from concatenation process of 15 *gendings*.

TABLE V
FREQUENT SEQUENCES OF 2-ITEMSETS RESULTED FROM CONCATENATION PROCESS

<A, B>	<1, 2>, <1, 6>, <2, 1>, <2, 3>, <2, 6>, <3, 1>, <3, 2>, <3, 5>, <3, 6>, <5, 3>, <5, 6>, <6, 1>, <6, 3>, <6, 5>
<B, C>	<1, 2>, <1, 3>, <1, 5>, <1, 6>, <2, 1>, <2, 3>, <2, 5>, <2, 6>, <3, 1>, <3, 5>, <3, 6>, <5, 1>,..., <6, 5>
<C, D>	<1, 6>, <2, 1>, <2, 6>, <3, 2>, <3, 5>, <5, 2>, <5, 3>, <5, 6>, <6, 1>, <6, 5>
<D, A*>	<1, 2>, <1, 6>, <2, 1>, <2, 3>, <2, 5>, <2, 6>, <3, 1>, <3, 2>, <3, 5>, <5, 1>, <5, 2>, <5, 3>,..., <6, 5>

TABLE VI
FREQUENT SEQUENCES OF 3-ITEMSETSRESULTED FROM CONCATENATION PROCESS

<A, B, C>	<1, 2, 1>, <1, 6, 3>, <1, 6, 5>, <2, 1, 2>, <2, 1, 5>, <2, 1, 6>, <2, 3, 1>, <2, 3, 6>, <2, 6, 5>, <3, 1, 3>, <3, 1, 5>, <3, 2, 1>, <3, 2, 3>, ...,<6, 5, 6>
<B, C, D>	<1, 2, 6>, <1, 3, 2>, <1, 5, 3>, <1, 5, 6>, <1, 6, 1>, <2, 1, 6>, <2, 3, 2>, <2, 3, 5>, <2, 5, 3>, <2, 6, 5>, <3, 1, 6>, <3, 5, 2>, <3, 5, 3>, ...,<6, 5, 6>
<C, D, A*>	<1, 6, 1>, <1, 6, 2>, <1, 6, 3>, <1, 6, 5>, <2, 1, 2>, <2, 6, 2>, <2, 6, 3>, <3, 2, 1>, <3, 2, 3>, <3, 2, 5>, <3, 2, 6>, <3, 5, 1>, <3, 5, 3>, ...,<6, 5, 6>
<D, A*, B*>	<1, 2, 6>, <1, 6, 5>, <2, 1, 6>, <2, 3, 1>, <2, 3, 2>, <2, 3, 5>, <2, 5, 3>, <2, 5, 6>, <2, 6, 1>, <2, 6, 5>, <3, 1, 2>, <3, 1, 6>, <3, 2, 1>, ..., <6, 5, 6>

TABLE VII
FREQUENT SEQUENCES OF 4-ITEMSETSRESULTED FROM CONCATENATION PROCESS

<A, B, C, D>	<1, 2, 1, 6>, <1, 6, 3, 2>, <1, 6, 3, 5>, <1, 6, 5, 3>, <1, 6, 5, 6>, <2, 1, 2, 6>, <2, 1, 5, 3>, <2, 1, 6, 1>, <2, 3, 1, 6>, <2, 3, 6, 5>, ...,<6, 5, 6, 5>
<B, C, D, A*>	<1, 2, 6, 2>, <1, 2, 6, 3>, <1, 3, 2, 3>, <1, 3, 2, 5>, <1, 5, 3, 5>, <1, 5, 6, 5>, <1, 6, 1, 6>, <2, 1, 6, 1>, <2, 1, 6, 2>, <2, 1, 6, 3>, ... , <6, 5, 6, 2>
<C, D, A*, B*>	<1, 6, 1, 6>, <1, 6, 2, 1>, <1, 6, 3, 2>, <1, 6, 3, 5>, <1, 6, 5, 3>, <1, 6, 5, 6>, <2, 1, 2, 6>, <2, 6, 2, 1>, <2, 6, 3, 1>, <2, 6, 3, 6>, ...,<6, 5, 6, 5>
<D, A*, B*, C*>	<1, 2, 6, 5>, <1, 6, 5, 3>, <2, 1, 6, 3>, <2, 1, 6, 5>, <2, 3, 1, 3>, <2, 3, 2, 1>, <2, 3, 2, 3>, <2, 3, 2, 5>, <2, 3, 2, 6>, <2, 3, 5, 3>, ...,<6, 5, 6, 5>

Next is the prune phase, where the itemsets resulting from concatenation process of 15 *gendings* are pruned.

The prune phase results in the function chaining of 2-itemsets, 3-itemsets, and 4-itemsets to arrange notes.

Table VIII shows the illustration of notes sequence prunes of 2-itemsets, with note 1 used as a value in function *A*, and also as a starting point of the first note of a *gatra*.

The rules of note sequences are defined based on the prunes. Below is an example of rules of note sequences of *gamelan* music:

2-itemsets Rules
Start
IF A = 1
THEN B = 2**OR** 6
...
IF D = 6
THEN A* =1 **OR** 2 **OR** 3 **OR** 5
End

3-itemsets Rules
Start
IF A = 1 **AND** B = 2
THEN C = 1
...
IF C = 6 **AND**D = 5
THENA* = 3**OR**6
End

4-itemsets Rules
Start
IF A = 1 **AND** B = 2**AND** C = 1
THEND = 6
...
IFD = 6 **AND**A* = 5**AND** B* = 6
THENC* = 2**OR**5
End

TABLE VIII
PRUNES OF 2-ITEMSETS

A	<A, B>	<B, C>	<C, D>	<D, A*>	A
1	<1, 2>	<2, 1>	<1, 6>	<6, 1>	1
				<6, 2>	2
				<6, 3>	3
				<6, 5>	5
		<2, 3>	<3, 2>	<2, 1>	1
				<2, 3>	3
				<2, 5>	5
				<2, 6>	6
			<3, 5>	<5, 1>	1
				<5, 2>	2
				<5, 3>	3
				<5, 6>	6
		<2, 5>	<5, 2>	<2, 1>	1
				<2, 3>	3
				<2, 5>	5
			<5, 3>	<3, 1>	1`
				<3, 2>	2
				<3, 5>	5
			<5, 6>	<6, 1>	1
				<6, 2>	2
				<6, 3>	3
				<6, 5>	5
		...	...	...	...
...	...	...	...	...	...

As the note sequence rules of *gamelan* music have been produced using AFiS algorithm, the experiment continues to melodic feature knowledge evaluation. A recommendation program to arrange note sequences is developed for the evaluation of the accuracy of the implementation of AFiS algorithm in building melodic feature knowledge of *gamelan* music. The knowledge and rules built by AFiS algorithm are implemented to the program. The program is expected to answer correctly some notes of test *gendings* randomly deleted from the sequences.

Five *gendings*, *Liwung*, *Respati*, *Kapilaya*, *Tejamaya*, *Sengsem*, are used as test *gendings*.

The process of data cleansing by removing pins in note sequences is implemented to all test *gending*, and then 30% of total notes of each test gending are changed to 0.

Further, the note sequences test of each *gending* is inputted in the solving program.

The program works by scanning the notes before and after the 0 value using 2-itemsets prunes to find the answer. The scanning process continues to 3-itemsets prunes and 4-itemsets prunes, and stops when there is only one note chosen as an answer. Otherwise, if there is more than 1note chosen as an answer, then the note which has the highest weight value is chosen as the answer. Table IX shows the result of the accuracy evaluation, by showing the original note sequences of each test *gending*, the questions, and the answers notes. The notes highlighted indicate the wrong answers by the system.

TABLE IX
RESULT OF THE ACCURACY EVALUATION

ID.	TYPES	NOTE SEQUENCE
01	Original	1, 6, 1, 6, 2, 1, 5, 3, 5, 6, 5, 3, 1, 2, 1, 6, 2, 1, 5, 3, 1, 6, 5, 3, 5, 6, 5, 3, 1, 2, 1, 6
	Questions	1, 0, 1, 6, 0, 1, 5, 0, 5, 6, 5, 3, 0, 2, 0, 6, 2, 1, 0, 3, 1, 0, 5, 0, 5, 6, 5, 0, 1, 2, 1, 0
	Answers	1, 6, 1, 6, **3**, 1, 5, 3, 5, 6, 5, 3, 1, 2, 1, 6, 2, 1, 5, 3, 1, 6, 5, 3, 5, 6, 5, 3, 1, 2, 1, 6
02	Original	3, 2, 1, 6, 3, 2, 1, 6, 3, 2, 3, 5, 6, 3, 1, 6, 3, 5, 1, 6, 3, 5, 1, 6, 5, 6, 5, 6, 3, 5, 3, 2, 6, 5, 3, 2, 6, 5, 3, 2, 3, 2, 3, 5, 6, 3, 1, 6
	Questions	3, 0, 1, 6, 0, 2, 0, 6, 3, 0, 3, 5, 6, 3, 1, 0, 3, 0, 1, 6, 0, 5, 1, 0, 5, 6, 0, 6, 0, 5, 3, 2, 6, 5, 3, 0, 6, 5, 3, 0, 3, 2, 3, 0, 6, 3, 1, 0
	Answers	3, **5**, 1, 6, 3, 2, 1, 6, 3, **5**, 3, 5, 6, 3, 1, 6, 3, 5, 1, 6, 3, 5, 1, 6, 5, 6, 5, 6, 3, 5, 3, 2, 6, 5, 3, 2, 6, 5, 3, 2, 3, 2, 3, **2**, 6, 3, 1, 6
03	Original	5, 6, 3, 2, 5, 6, 3, 2, 5, 6, 1, 6, 3, 5, 3, 2, 5, 3, 1, 6, 5, 3, 1, 6, 3, 5, 6, 5, 6, 5, 3, 2
	Questions	5, 0, 3, 2, 0, 6, 3, 0, 5, 0, 1, 0, 3, 5, 3, 0, 5, 3, 0, 6, 5, 0, 1, 6, 3, 0, 6, 5, 6, 5, 0, 2
	Answers	5, 6, 3, 2, **3**, 6, 3, 2, 5, **3**, 1, 6, 3, 5, 3, 2, 5, 3, **5**, 6, 5, 3, 1, 6, 3, 5, 6, 5, 6, 5, 3, 2
04	Original	6, 3, 6, 5, 6, 3, 6, 5, 3, 2, 5, 3, 6, 5, 3, 2, 5, 6, 5, 6, 2, 1, 5, 3, 2, 1, 2, 3, 2, 1, 6, 5, 6, 3, 6, 5, 6, 3, 6, 5, 3, 2, 5, 3, 6, 5, 3, 2, 6, 5, 3, 2, 6, 5, 3, 2, 3, 2, 3, 5, 6, 3, 6, 5
	Questions	6, 3, 0, 5, 6, 3, 6, 0, 3, 0, 5, 3, 6, 0, 3, 0, 5, 6, 0, 6, 2, 1, 5, 3, 0, 1, 2, 0, 2, 0, 6, 0, 6, 3, 6, 0, 6, 3, 0, 5, 3, 0, 5, 3, 6, 5, 0, 2, 6, 0, 3, 0, 6, 5, 3, 0, 3, 2, 0, 5, 6, 3, 0, 5
	Answers	6, 3, 6, 5, 6, 3, 6, 5, 3, 2, 5, 3, 6, 5, 3, 2, 5, 6, 5, 6, 2, 1, 5, 3, 2, 1, 2, **6**, 2, 1, 6, 5, 6, 3, 6, 5, 6, 3, 6, 5, 3, 2, 5, 3, 6, 5, 3, 2, 6, 5, 3, 2, 6, 5, 3, 2, 3, 2, 3, 5, 6, 3, 6, 5
05	Original	2, 3, 6, 5, 2, 3, 6, 5, 3, 2, 1, 6, 3, 5, 3, 2, 3, 2, 1, 6, 5, 3, 5, 6, 1, 6, 5, 3, 2, 1, 6, 5
	Questions	2, 3, 6, 0, 2, 3, 6, 0, 3, 2, 1, 0, 3, 0, 3, 0, 3, 2, 1, 0, 5, 3, 0, 6, 1, 0, 5, 0, 2, 1, 0, 5
	Answers	2, 3, 6, 5, 2, 3, 6, 5, 3, 2, 1, 6, 3, 5, 3, 2, 3, 2, 1, 6, 5, 3, 5, 6, 1, 6, 5, 3, 2, 1, **3**, 5

Accuracy is measured by the *x 100%* formula of (*total number of right answers/total number of questions).*

The results show that the accuracy of the implementation of AFiS algorithm in building melodic feature knowledge of *gamelan* music is up to 86.5% achieved by (54 / 62) × 100%.

Table X shows the calculation of the accuracy evaluation.

TABLE X
THE RESULT OF ACCURACY EVALUATION

GENDING	TOTAL NUMBER OF NOTES (TNN)	TOTAL NUMBER OF QUESTIONS (TNN x 30%)	TOTAL NUMBER OF RIGHT ANSWERS
Liwung	32	10	9
Respati	48	14	11
Kapilaya	32	10	7
Tejamaya	64	19	18
Sengsem	32	10	9
		62	**54**

The second evaluation is to find whether the different answers of the program are accepted as alternative notes to the original notes. This evaluation is conducted since some sources have a difference of one or two notes in some *gendings*, but the difference in both sources is still accepted by *gamelan* artists and experts.

Two experts with an academic background and 2 experts with a practitioner background are involved to evaluate whether the different notes answered by the program can be accepted as alternative notes. Experts are asked to give a value of 0 or 1 to describe their acceptance for the different notes answered by the program, where 0 represents not accepted and 1 represents accepted. The values given by experts for each *gendings* are then summed up. A minimum value of 3 is used as a standard to determine that a *gending* can be accepted as alternative notes.

The results show that 4 out of 5 *gendings*, *Liwung*, *Respati*, *Kapilaya*, and *Sengsem*, are accepted by the experts.*Gending Tejamaya* have a value of 2, so the different notes given by the program cannot be accepted as alternative notes. Table XI shows the evaluation of the acceptance of different note answers as alternative notes.

TABLE XI
THE RESULT OF THE EVALUATION OF ALTERNATIVE NOTES

GENDING	EXPERTS				SUM	ACCEPTANCE
	I	II	III	IV		
Liwung	1	1	1	1	4	Y
Respati	1	1	1	1	4	Y
Kapilaya	1	1	1	1	4	N
Tejamaya	0	0	1	1	2	N
Sengsem	1	0	1	1	3	Y

Based on the evaluation, the different notes given by the program in 4 *gendings*, *Liwung*, *Respati*, and *Kapilaya*, and *Sengsem* are accepted as alternative notes, while for *gending Tejamaya* the different notes answered by the program are not accepted as alternative notes.

V. Conclusion

The AFiS algorithm is a type of Apriori-based algorithm, which is suitable for small data volumes, and which was used for mining notes sequences of gamelan music. The implementation of the AFiS algorithm to build melodic feature of *gamelan* music results in an accuracy which is up to 86.5%, while most of 13.5% that is not reached yet is accepted by the experts as alternative notes to the original notes.

Gamelan is a complex music which needs more approaches to accomplish the rules of other components of *gending*, such as *gatra*, *pathet*, *balungan*, *garap*.An improvement in the accuracy of the melodic feature knowledge is still needed as well as rules formulation for other components of gending. For the next research, the melodic feature knowledge resulted in this research can be used as a part of a system to develop an automatic *gending* composition, including the rules formulation, for a more detailed knowledge of *gamelan* music.

References

[1] G. Nierhaus, *Algorithmic Composition: Paradigms of Automated Music Generation*(Springer Wien, New York, 2009)

[2] J.D. Fernandes, F. Vico, AI Methods in Algoritmic Composition: A Comprehensive Survey, *Journal of Artificial Intelligence Research 48*, 2013

[3] E.R. Miranda, *Composing Music with Computers*,(Focal Press, Burlington, 2004)

[4] L.A. Hiller, Jr., L.M. Isaacson, *Experimental Music: Composition with an Electronic Computer*, (McGraw-Hill Book Company, Inc, New York, 1959)

[5] I. Xenakis, *Formalized Music: Thought and Mathematics in Composition*, (Pendragon Press, New York, 1992)

[6] K. Ebcioglu, An Expert System for Chorale Harmonization, *AAAI-86 Procedings*, 1986

[7] R.A. McIntyre, Bach In A Box: The Evolution Of Four Part Baroque Harmony Using The Genetic Algorithm, *IEEE World Congress on Computational Intelligence*, 1994

[8] K.M. Kitani, H. Koike, ImprovGenerator: Online grammatical induction for on-the-fly improvisation accompaniment, *Proceedings of the International Conferenceon New Interfaces for Musical Expression*, 2010

[9] M. Fridenfalk, Algorithmic Music Composition for Computer Games Based on L-system,*Games Entertainment Media Conference (GEM), 2015 IEEE*

[10] D. Quick, P. Hudak,*Grammar-Based Automated Music Composition in Haskell*,ACM Computing Surveys (CSUR) Surveys Homepage archiveVolume 43 Issue 1, November 2010Article No. 3

[11] D. Quick, Composing with Kulitta, *ICMC 2015 – Sept. 25 - Oct. 1, 2015 – CEMI, University of North Texas*, 2015

[12] F. Pachet, A joyful ode to automatic orchestration,*ACM Transactions on Intelligent Systems and Technology, Vol. 8, No. 2, Article 18*, October 2016

[13] Ryan groves, Automatic melodic reduction using a supervisedProbabilistic context-free grammar, *17th International Society for Music Information Retrieval Conference*, 2016.

[14] R. Rangarajan, Generating Music from Natural Language Text, *The Tenth International Conference on Digital Information Management (ICDIM 2015), 2015*

[15] A. Suprem, M. Ruprem, A New Composition Algorithm for Automatic Generation of Thematic Music from the Existing Music Pieces, *Proceedings of the World Congress on Engineering and Computer Science 2013 Vol II WCECS 2013, 23-25 October, 2013, San Francisco, USA*

[16] J. Becker, A. Becker, *A Grammar of the Musical Genre Srepegan*, (University of Texas Press, Texas, 1982)

[17] D. W. Hughes, *Deep Structure and Surface Structure in Javanese Music: A Grammar of Gendhing Lampah*(University of Illinoi Press, Illinois, 1988)

[18] B.C. Kwon, J. Verma, and A. Perer, Peekquence: Visual analytics forevent sequence data, *ACM SIGKDD 2016 Workshop on Interactive Data Exploration and Analytics*, 2016.

[19] M. Kamber, J. Han, JY. Chiang, Metarule-guided mining of multi-dimensional association rules using data cubes, in S. Itkar, U. Kulkarni, Distributed Sequential Pattern Mining: A Survey and Future Scope, *International Journal of Computer Applications (0975 – 8887) Volume 94 – No. 18*, May 2014

[20] R. Boghey,S. Singh,Sequential Pattern Mining: A Survey on Approaches, *2013 International Conference on Communication Systems and Network Technologies*, 2013

[21] C.H. Mooney, J.F. Roddick, Sequential Pattern Mining – Approaches and Algorithms, *ACM Computing Surveys, Vol. 45, No. 2, Article 19*, February 2013.

[22] S. Muthuselvan, Dr. K. Soma Sundaram, *A Survey of Sequence Patterns in Data Mining Techniques, International Journal of Applied Engineering Research*, ISSN 0973-4562 Volume 10, Number 1 (2015) pp. 1807-1815, 2015

[23] P.Nithya,B.U. Maheswari, R.Deepa, Efficient Sequential Pattern Mining Algorithm To Detect Type-2 Diabetes, *International Journal of Advanced Research in Science, Engineering and Technology* Vol. 3, Issue 3, March 2016

[24] K. Subramanian, M. Phil, E. Elakkiya, Modified Sequential Pattern Mining Using Direct Bit Position Method, *International Journal of Science and Research (IJSR) Volume 5 Issue 2, February 2016*

[25] N. Mabroukeh,C.I. Ezeife, *A Taxonomy of Sequential Pattern Mining Algorithms*,ACM Computing Surveys, Vol. 43, No. 1, Article 3, Publication date: November 2010.

Authors' information

[1]Faculty of Mathematics and Natural Science, Universitas Gadjah Mada, Yogyakarta, Indonesia

[2]Faculty of Computer Science, Universitas Dian Nuswantoro, Semarang, Indonesia.

[3]Faculty of Performing Arts, Institut Seni Indonesa, Surakarta, Indonesia.

Khafiizh Hastuti received her M.Kom degree in 2012 from informatics Engineering Department of Universitas Dian Nuswantoro. Her research interests are artificial intelligent, data mining, and software engineering.
E-mail: afis@dsn.dinus.ac.id

Azhari SN, Department of Computer Science and Electronics, Universitas Gadjah Mada. Undergraduate Statistics Universitas Gadjah Mada, Master of Software Engineering Institut Teknologi Bandung, Doctor of Computer Science, Universitas Gadjah Mada. His research interests are Intelligent agent, software engineering, project management.
E-mail: arisn@ugm.ac.id

Aina Musdholifah, Department of Computer Science and Electronics Universitas Gadjah Mada.Undergraduate and master of computer science, Universitas Gadjah Mada, Ph.D Computer Science Universiti Teknologi Malaysia (UTM). Her research interests are genetic algorithm, fuzzy system.
E-mail: aina_m@ugm.ac.id

Rahayu Supanggah, Professor of Ethnomusicology and Music Composition, at ISI Surakarta. Third cycle Ph.D in Ethnomusicology, graduated in 1985. Karawitan composer.
E-mail: supanggah@yahoo.com