Lemmatization Technique in Bahasa: Indonesian Language

Derwin Suhartono
Computer Science Department, Bina Nusantara University, Jakarta, Indonesia
Email: dsuhartono@binus.edu

David Christiandy and Rolando
Computer Science Department, Bina Nusantara University, Jakarta, Indonesia
Email: david.christiandy@gmail.com, rolando_kai@hotmail.com

***Abstract*—many researches and inventions have been made in the field of linguistics and technology. Even so, the integration between linguistics and technology is not always reliable to all language. Every language is unique in its linguistic nature and rules. In this paper, a lemmatization technique in Bahasa (Indonesian language) is presented. It has achieved good precision by using The Indonesian Dictionary and a set of rules to remove affixes. The lemmatization technique is developed based on the previous algorithm, Indonesian stemmer. Both Indonesian stemming and lemmatization method have the same characteristics but a little bit different in its implementation. The way to reach its own goal/purpose is defined as a core difference and therefore possible to modify. The result shows that the algorithm achieved roughly 98% precision on a collection consisting 57,261 valid words with 7,839 unique valid words gathered from Kompas.com, an Indonesian online news article.**

***Index Terms*—stemmer, algorithm, lemmatization, language, Bahasa, Indonesian**

I. INTRODUCTION

Issues about information retrieval occur in many different fields. Its implementation can be found in term of image, text, video, etc. Majority of a document consists of text rather than image, video and so on. That is why the way how to manage text or document is becoming more important.

A document consists of words in which most of them do not use the base word/dictionary entry. It is caused by the adaptation according to the sentence intention.

Lemmatization is the process of finding the base word/dictionary entry (*lemma*) from a word form [6]. In [4], the same definition is also stated. The process is aimed at normalizing the input according to the partner associations of the form based on its own lemma [7]. So far, no attempt has been made to develop a lemmatization method for Bahasa. Instead, stemming, which is considered similar to lemmatization has gained more attention in its development for Indonesian.

From all the published journals related to this topic, only stemming methods have been developed for Bahasa. Stemming aims to reduce the numbers of variation from a language to a standard, canonical representation (known as stem) [5]. Indonesian stemming methods use root word as its stem; which means that mostly they are dictionary dependant. This characteristic is also that of lemmatization; because every headword in the dictionary is a lemma.

The lemmatization process may be different, according to the nature of the language. Indonesian is a morphologically complex language [9] where almost every word can be inflected with affixes.

Instead of finding the root word, we believe that finding the lemma from a given word form will give better precision in semantic, i.e. the meaning of the sentence [8], and more fitting for NLP applications such as morphological analysis and language translation. There are lemmas that consist of more than one word; i.e. a phrase. This plays an important part when analyzing sentences; for example given an Indonesian sentence "*saya harus mempertanggungjawabkannya*" (literally means: I have to be responsible for it), the result is expected to be three lemmas {*saya*, *harus*, *tanggung jawab*}. The lemma *tanggung jawab* is considered as one; they share the same part of speech.

Based on that background, the objective of this paper is to present a lemmatization method for Bahasa, based on Bahasa stemmer which uses Indonesian dictionary and a set of rules to remove inflections.

II. RELATED WORKS

The latest development of Bahasa stemmer was Enhanced Confix-Stripping Stemmer [3]. This stemmer was an improvement of the work initiated by Nazief and Adriani and improved further by Asian and Nazief. Below is the algorithm of Confix-Stripping Stemmer [2] in a detailed explanation :

1. The input is first checked against the dictionary. If the input exists in dictionary, then the input will be returned as result lemma.
2. Inflectional particle suffixes (*-kah*, *-lah*, *-tah*, *-pun*) will be removed from the current input, and the remains will be kept inside a string variable, checked against the dictionary. If exists, then process terminates.

3. Inflectional possessive pronoun (*-ku*, *-mu*, *-nya*) will be removed from the string variable, and checked against the dictionary. If exists, process terminates.
4. Derivational suffixes (*-i*, *-kan*, *-an*) will be removed from the string variable, and checked against the dictionary. If exists, process terminates.
5. This step focuses on removing the derivational prefixes (*beN-*, *di-*, *ke-*, *meN-*, *peN-*, *se-*, *teN-*) from the string variable. Notice the uppercase *N* is a wildcard; it can be any alphabets (a-z). Step 5 is recursive, because in Indonesian morphology derivational prefix can be stacked. Some prefixes (*di-, ke-, se-*) are considered simple, because in practice they do not change the lemma. On the other hand, the other prefixes (*beN-, meN-, peN-, teN-*) do change the lemma, and differs according to the first letter of the lemma. These transformations and variants are listed on the rule tables below.

TABLE 1.
PREFIX STRIPPING RULE SET FOR *BE-*

Rule	Construct	Return
1	berV...	ber-V... | be-rV...
2	berCAP...	ber-CAP... Where C!='r' and P!='er'
3	berCAerV...	ber-CAerV... Where C!='r'
4	belajar...	bel-ajar...
5	beC_1erC_2...	be-C_1erC_2... Where C_1!={'r' | 'l'}

TABLE 2.
PREFIX STRIPPING RULE FOR *TE-*

Rule	Construct	Return
6	terV...	ter-V... | te-rV...
7	terCerV...	ter-cerV... Where C!='r'
8	terCP...	ter-CP... Where C!='r' and P!='er'
9	teC_1erC_2...	te-C_1erC_2... Where C_1!='r'

TABLE 3.
PREFIX STRIPPING RULE FOR *PE-*

Rule	Construct	Return
20	pe{w|y}V...	pe-{w|y}V...
21	perV...	per-V... | pe-rV...
22	perCAP...	per-CAP... where C!=’r’ and P!=’er’
23	perCAerV...	per-CaerV... where C!=’r’
24	pem{b|f|v}...	pem-{b|f|v}...
25	pem{rV|V}...	pe-m{rV|V}... | pe-p{rV|V}...
26	pen{c|d|j|z}...	pen-{c|d|jz}...
27	penV...	pe-nV... | pe-tV...
28	peng{g|h|q}...	peng-{g|h|q}...
29	pengV...	peng-V... | peng-kV...
30	penyV...	peny-sV...
31	pelV...	pe-lV... Except: “pelajar” return “ajar”
32	peCP	pe-CP... where C!={r|w|y|l|m|n} and P!=’er’
33	peCerV	per-CerV. . .where C!={r|w|y|l|m|n}

TABLE 4.
PREFIX STRIPPING RULE FOR *ME-*

Rule	Construct	Return
10	me{l|r|w|y}V...	me-{l|r|w|y}V...
11	mem{b|f|v}...	mem-{b|f|v}...
12	mempe{r|l}	mem-pe...
13	mem{rV|V}...	me-m{rV|V}... | me-p{rV|V}...
14	men{c|d|j|z}...	men-{c|d|j|z}...
15	menV...	me-nV... | me-tV...
16	meng{g|h|q|k}...	meng-{g|h|q|k}...
17	mengV...	meng-V... | meng-kV...
18	menyV...	meny-sV...
19	mempV...	mem-pV... where V!=’e’

V stands for a vowel (a, i, u, e, o), C stands for consonant, A represents any alphabet character (a-z), and P represents a short fragment of words, such as ‘*er*’.
There are several termination conditions for this step:

a. The prefix and the removed suffix are listed in the invalid affix pair table below (Table 5).
b. The removed prefix is literally equivalent to previously removed prefix.
c. The recursive limit for this step is three.

TABLE 5.
DISALLOWED PREFIX AND SUFFIX PAIRS;
EXCEPT THE *KE*- AND -*I* AFFIX PAIR FOR THE ROOT WORD *TAHU*

Prefix	Disallowed Suffixes
ber-	-i
di-	-an
ke-	-i and –kan
me-	-an
ter-	-an
per-	-an

The removed prefix will be recorded, and the string variable will be checked against the dictionary. If the string variable does not exist in the dictionary and no termination condition is satisfied, then step 5 will be repeated, with the string variable as input.

6. If the string variable is still not found after Step 5, then the rule tables will be examined whether recoding (p. 63) is possible. In the rule set, there are several rules that hold more than one output. Take rule 17 for example; *mengV* has two outputs: *meng-V* or *meng-kV*. In step 5, the first output (i.e. left one) will always be picked first, and this can cause error. Recoding is done to undo this kind of error by going back to step 5 where this output selection happens; and instead selects the other output.
7. If the string variable still remains unknown to the dictionary, then the original input word will be returned.

In order to solve the major error causes as stated above (i.e. non-root word in the lookup dictionary, incomplete dictionary, and hyphenated words), three approaches are suggested:

1. Improve dictionary quality by using different dictionary sources, and compare its accuracy with the previous dictionary.
2. Add extra rules to handle hyphenated words. The main idea used to construct this rule is, if a hyphenated word contains an exact same pair word (e.g. *bulir-bulir*) then it will be stemmed to *bulir*. This also applies for hyphenated word with affixes (e.g. *seindah-indahnya*); the affix will be removed first and then checked whether the pair word is *stemmable*.
3. Modification of rules, prefixes and suffixes:
   a. Rules alteration for prefixes (*ter*-, *pe*-, *mem*-, and *meng*-) which has already been applied to the rule tables above. In detail, rule number 9 and 33 were added, rule number 12 and 16 were a modified version from the previous rule.
   b. Prefix removal will be performed before suffix removal if a given word has an affix pair from the list below:
      - *be*- and -*lah*
      - *be*- and -*an*
      - *me*- and -*i*
      - *di*- and -*i*
      - *pe*- and -*i*
      - *ter*- and -*i*

Compared against Nazief with the same dataset, the modified Nazief achieves around 2-3% higher accuracy which approximately is 95% [3] extended the Confix-Stripping Stemmer by solving unhandled cases with specific prefix type (p. 151), listed below:

1. "mem-p" as in *mempromosikan*,
2. "men-s" as in *mensyukuri*,
3. "menge-" as in *mengerem*,
4. "penge-" as in *pengeboman*,
5. "peng-k" as in *pengkajian*,
6. Incorrect affix removal order, resulting in unstemmed input. For example the word *pelaku* is overstemmed because of the "-*ku*" on the last part of the word is considered as an inflectional possessive pronoun suffix. Other example is the word *pelanggan*, which is overstemmed because the "-*an*" part on the last is considered as a derivational suffix.

In order to solve the cases above, suggested two improvements [3] :

1. Rules modification and addition to the rules table, in order to fit the specific unhandled cases above.
2. Extra process of stemming, which is called *loopPengembalianAkhiran* (p. 151), henceforth referenced as LPA. This extra step is appended after the last step of CS Stemmer's stemming process, specifically after the recoding attempt has failed (i.e. Step 8, in the CS Stemmer). After each step, a dictionary lookup is performed to check if the processed input listed in the dictionary. The detailed flow of LPA is as follows:
   a. Return CURRENT_WORD to the state before recoding, and return all prefixes that have been removed in the prefix removal process, and perform a dictionary lookup.
   b. Redo the prefix removal process.
   c. Return the previously removed suffixes in order: derivational, personal pronoun, and particle suffixes. On each order of suffix restoration, step d to step e is performed. An important exception is made for the derivational suffix "-kan". Firstly, only the '-k' is restored and step d and e is performed. However, if fails, then the rest (i.e. '-an') will be restored, and step d and e is performed.
   d. Redo the prefix removal process, and perform recoding if possible.
   e. If dictionary lookup fails, execute step a and restore the next order of suffix according to step c.

III. Proposed Algorithm

The lemmatization algorithm based on the state of the art, Enhanced Confix Stripping Stemmer (henceforth referred as ECS) [3]. This research does not aim to improve ECS, because it is different in goal/purpose. The lemmatization algorithm aims to modify ECS instead, in order to fit the lemmatization concept. However, there are similarities in some of the processes, for example, removal of affix, in order to reach its lemma form. These kinds of process can be re-implemented with minimal changes. There are some cases that does not stemmed successfully by ECS; that will hopefully be solved by lemmatization algorithm. These cases are:

1. Ineffective Rule, especially rules that handle *meny-* and *peny-*. For example, *penyanyi* and *menyatakan*.
2. Compound words constructed from lemma phrases, such as *diberitahukan*.
3. Over-stemming, such as *penyidikan* to *sidi*.
4. Under-stemming, such as *mengalami* produces *alami*.

Depicted in Figure 1, the lemmatization algorithm includes several processes:

A. **Dictionary Lookup**. This process checks whether the word is listed as a lemma in the dictionary. When the lookup succeeds, then the algorithm will stop, and the lemma will be returned as a result. This process is executed at the end of every executed process, to ensure that every applied transformation are always checked and will be immediately returned as result when the lemma is found. There are phrases that are considered as a lemma, for example, 'tanggung jawab'. These lemma phrases, when given a confix, will be joined together and become one word (compound word). e.g. the lemma phrase '*tanggung jawab'*, when given a '*per- -an*' confix, will result in *pertanggungjawaban*. This case is not handled by the previous stemming algorithm, because it consists of more than one word.

B. **Rule Precedence Check**, This process is executed to determine the other processes' execution order. There are some prefix-suffix combinations that produce faster, and more accurate result, if prefix removal is executed prior to suffix removal. These combinations are:

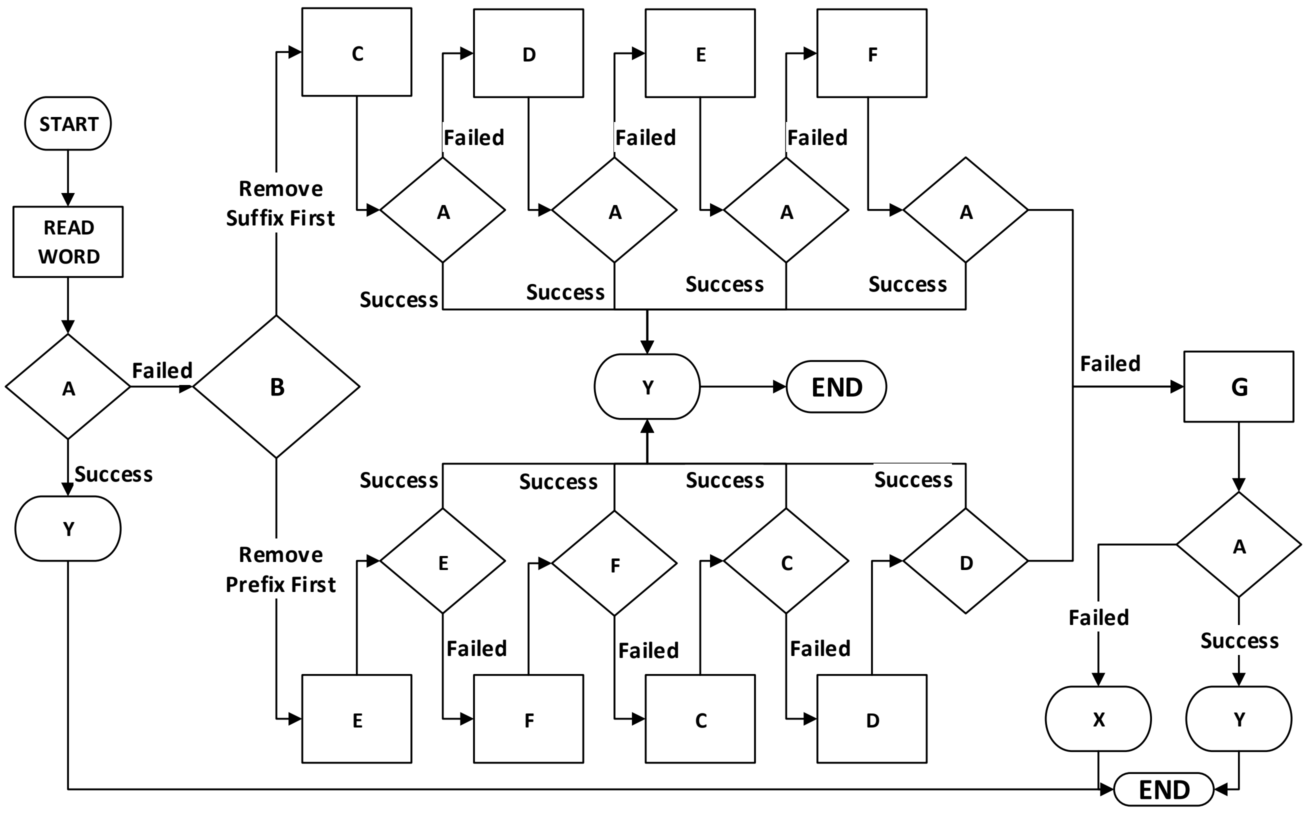


Figure 1. Lemmatization Algorithm Flowchart

be- and *-lah*,

1. *be-* and *-an*,
2. *me-* and *-i*,
3. *di-* and *-i*,
4. *pe-* and *-i*,
5. *te-* and *-i*.

When an input word has a prefix-suffix pair that satisfies the combinations listed, then the execution order will be *derivational prefix removal*, *recoding*, *inflectional suffix removal*, *and derivational suffix removal*. On the contrary, if the affix pair of input word does not match any of the affix combinations listed, then *inflectional suffix removal* and *derivational suffix removal* will be performed/executed first.

C. **Inflectional Suffix Removal.** Inflectional suffix has two types of suffix, *particle* {'-lah', '-kah', '-tah', '-pun'} and *possessive pronoun* {'-ku', '-mu', '-nya'}. Indonesian language structure dictates that *particle* suffix will always be the last suffix added on a word. So, this process will try to remove *particle* suffix before removing *possessive pronoun* suffix. For example, the word '*bajukupun*' contains the particle '-pun', and the possessive pronoun '-ku'. The particle will be removed first, resulting in '*bajuku*', and a dictionary lookup is performed. Since the word is not listed in dictionary, the possessive pronoun is removed, producing the word '*baju*' as result.

D. **Derivational Suffix Removal.** This process will try to remove derivational suffix {*-i, -kan, -an*} from a given word. Derivational suffix is always added to a word before adding inflectional suffix. It means that this process will always be executed after inflectional suffix removal (except when the word has no inflectional suffixes). For example, the word *nyalakan* contains the derivational suffix *-kan*, therefore it will be removed and produces the word *nyala* as result.

E. **Derivational Prefix Removal**. Derivational prefix has two kind of groups, *plain* {'di-', 'ke-', 'se-'} and *complex* {'me-', 'be-', 'pe-', 'te-'}. Plain prefixes, as the name suggests, do not require any rule, nor transform the word when added; which means, the removal process is done by directly removing the detected plain prefixes (e.g. 'dibawa', 'sejalan', 'ketutup'). On the other hand, complex prefixes transform the word when added. Indonesian language permits derivational prefix combination on a word (e.g. 'berkelanjutan' which originates from 'lanjut'), however there are constraints that limit the combination possibility. The possible combinations are:

1. 'di-', followed by 'pe-' or 'be-' prefix type (e.g. 'diperlakukan' and 'diberlakukan')
2. 'ke-', followed by 'be-' or 'te-' prefix type (e.g. 'kebersamaan' and 'keterlambatan')
3. 'be-', followed by 'pe-' prefix type (e.g. 'berpengalaman')
4. 'me-', followed by 'pe-', 'te-, or 'be-' prefix type (e.g. 'mempersulit', 'menertawakan', and 'membelajarkan')
5. 'pe-', followed by 'be-' prefix type (e.g. 'pemberhentian'), with special case for 'tertawa' ('penertawaan').

The lemmatization algorithm will remove up to three prefixes and three suffixes; whereas the three suffixes consists of derivational suffix, possessive pronoun, and and particle suffix types and the prefixes follow the combination rule above. Therefore, this process is repetitive, up to three iterations. At the end of every iteration, the current state of word is checked against the dictionary in order to prevent overstemming. Termination also occurs when the currently identified prefix has been removed in the previous iteration, or the word contains a disallowed affix pair (prefix-suffix), listed below:

TABLE 6.
DISALLOWED AFFIX PAIRS

Prefix	Suffix
be-	-i
di-	-an
ke-	-i, -kan
me-	-an
se-	-i, -kan
te-	-an

A valid word can contain up to two prefixes, and three suffixes. However this is not true for Indonesian scheme; take for example '*sepengetahuan*' which contains the prefix '*se-*', '*pe-*', and '*ke-*'. In this step, the '*se-*' prefix will be removed; which produces '*pengetahuan*'. On the second iteration, '*pe-*' will be removed; which produces '*ketahuan*'. The last iteration will remove '*ke-*', which produces '*tahuan*'. So, the lemmatization algorithm will iterate up to three times.

F. **Recoding**. When affix removal process still fails the dictionary lookup, there are a possibility that the removal process did not transform the word accordingly. For example, the word 'menanya' is transformed into 'me-nanya' which in result fails the lookup; this happens because the original word, 'tanya', is transformed into 'nanya' when combined with the prefix 'me-'. However, there are also cases where the lemma's first letter is 'n', for example 'nama' in the word 'menamai'. The purpose of recoding is to go through all kinds of transformation possibilities. This is achieved by recording alternative path of transformation. Take rule 1 for example, there are two possible output. On affix removal, the chosen output will always be the left one; However when this process is executed, the algorithm checks whether there are any alternative path recorded when removing affixes; and then replaces the current transformation with the alternative. For example, the word '*berima*' (in rhythm), contains the prefix '*be*-', and affix removal rule 1 will be applied (Table 3.1) because it follows the pattern '*berV…*'. However, the default output of this rule is to remove 'ber-' from the word, resulting in '*ima'* and this causes the dictionary lookup to fail. This process checks for the recoding path, i.e. 'berV… to 'be – rV…', reattached the removed prefix (from '*ima*' to '*berima'*), and applied the recoding rule (from '*berima*' to '*rima'*) and produces '*rima'* as result.

G. **Suffix Backtracking**. This process is attempted after affix removal and recoding fails dictionary lookup. On each step, prefix removal and recoding will performed. First, the prefixes that have been removed will be reattached to the word; then prefix removal and recoding is performed. If the result fails the dictionary, the prefixes are reattached and the removed derivational suffix will also be reattached. If the result still fails, reattach prefixes, derivational suffix, and possessive pronoun. If the result still fails, the last step is the reattach the particle. There is a special case, when the removed derivational suffix is '-kan', then '-k' will be attached first. If the result fails, then '-an' will be attached. Considering the word '*pemberhentiannyapun*', and assuming that the dictionary lookup will always returns failure, the reattachment will be:
    1. Reattach prefixes: *pemberhenti*, and perform derivational prefix removal:
        a. '*pe*-' prefix removed, resulting in '*berhenti'*
        b. '*be*-' prefix removed, resulting in '*henti'*
    2. Reattach derivational suffix: *pemberhentian*, and perform derivational prefix removal:
        a. '*pe*-' prefix removed, resulting in '*berhentian'*
        b. '*be*-' prefix removed, resulting in '*hentian'*
    3. Reattach possessive pronoun: *pemberhentiannya*, and perform derivational prefix removal:
        a. '*pe*-' prefix removed, resulting in '*berhentiannya'*
        b. '*be*-' prefix removed, resulting in '*hentiannya'*
    4. Reattach particle: *pemberhentiannyapun* and perform derivational prefix removal:
        a. '*pe*-' prefix removed, resulting in '*berhentiannyapun'*
        b. '*be*-' prefix removed, resulting in '*hentiannyapun'*

H. **Return Original Word** (represented by X). This means that the lemmatization process fails to find the lemma.

I. **Return Lemma** (represented by Y). This means that the lemmatization has successfully found a lemma from the given word.

The test data/sample was collected manually from Kompas.com, one of the biggest news companies in Indonesia. The 25 articles collected were from Kompas.com's online article taken between 1st November 2012 and 15th January 2013, and distributed evenly between 10 news categories. Before lemmatized, the articles were parsed so it fits these conditions:

1. Minimum length of tested word is 4.
2. Numbers and special characters are truncated, leaving alphabets and stripes ('-').
3. The data is supplied in a form of one word per lemmatization process.

The parsed/formatted data contains 57,261 valid words with an average of 6.68 characters per word, and 7,829 unique valid words. The data is stored in a MySQL table to ease testing process. In analyzing the test data, there are several constraints/limitations to which lemmatization process are considered a success, which are considered an error/fault, and specific cases that are out of the current algorithm's scope. A lemmatization is considered successful, if a lemma is correctly produced from the input word. There are some cases that a lemma is produced incorrectly, which will fall to the error category. Out-of-scope cases are considered invalid or unqualified; therefore neither counted as a failure nor success. These out-of-scope cases are:

1. **Proper Nouns and Abbreviations**, which include people, area, or company names (Microsoft, Bandung, PT.KAI, etc.). The main reason this is included as out of scope, because they do not exist in the dictionary.
2. **Foreign Word**, which means other words not in Indonesian language. Same as point 1, they are not listed in Indonesian dictionary.
3. **Infix**, an affix that is inserted inside a word. For example, the infix '-er-' for 'gigi' which produces 'gerigi'. Words that contain infix are already listed as lemma; therefore infix removal procedure is not supported by this algorithm.
4. **Non-standard Words and Affixes**, which mean words that is not defined in Indonesian Dictionary, or *slang* words and affixes. A few examples of these words would be '*nggak*', '*gue*', '*bukain*' with its '*–in'* suffix.

Lemmatization errors can be classified into a few categories:

1. Overlemmatized: This term is similar to overstemming; Affix removal is performed too much/extensively, such that the produced lemma is not as expected. For example, in ECS's case

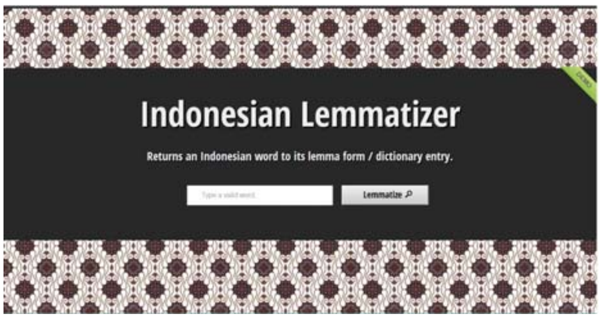
of overstemming, ‘penyidikan’ to ‘sidi’, where the correct one should be ‘sidik’.

2. Underlemmatized: This term is similar to understemming; Affix removal is performed too few, such that the produced lemma is not as expected. In ECS’s case, ‘mengalami’ to ‘alami’, where the correct one should be ‘alam’.
3. Incorrect Rule: In this case, the affix was incorrectly removed because of ineffective or incorrect rule. For example, ‘mengatakan’ may become ‘katak’, by removing ‘-an’ suffix, and ‘meng-‘ prefix.

The test algorithm will fetch all parsed data, and process them one by one. The results are saved in a separate table. When the input word is immediately returned as lemma, the algorithm will return “input_is_lemma” exception message; this does not affect the result in any way. When the algorithm fails to obtain the lemma from input word, it will return the original word, however with an exception message “lemma_not_found”. However, this does not mean that all results produced with an exception message is classified as error; the message can also indicate proper nouns and foreign words. After successfully storing the process results to the database, a manual inspection is done to analyze lemmatization errors. The algorithm itself does not know when it is overlemmatizing/underlemmatizing the input word; when it finds a lemma, then it will be returned as success. These cases need to be classified manually.

IV. Result And Evaluation

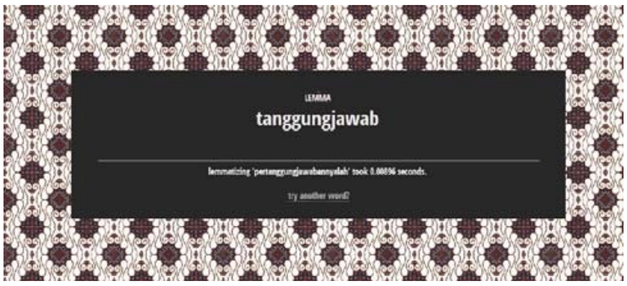
The algorithm was implemented to a simple web application, built in PHP and using MySQL database. The user is asked to directly input the desired word without preparing the batch files. The screenshot is shown in Figure 2. Basically, an input is supplied to the application, and an output will be shown as result.



Returns an Indonesian word to its lemma form / dictionary entry.

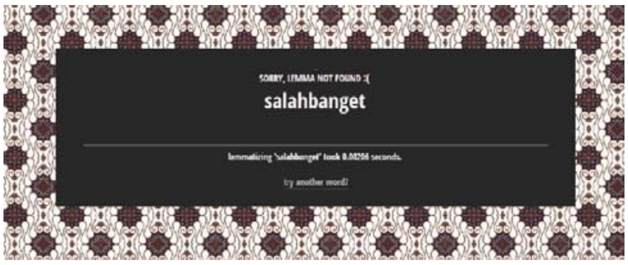
Figure 2. Main Display of Indonesian Lemmatizer

When a lemmatization process is successful, then Figure 3 will be shown; however when the lemmatization process returns error, such as lemma not found, then Figure 4 will be shown instead.



lemmatizing ‘pertanggungjawabannyalah’ took 0.00896 seconds.

Figure 3. Display of Successful Lemmatization



SORRY, LEMMA NOT FOUND :(

lemmatizing ‘salahbanget’ took 0.00266 seconds.

Figure 4. Display of Failed Lemmatization

Based on 25 articles in 10 categories captured from Kompas.com, the results are summarized as seen in table 7 and 8.

From table 7, it can be seen that many words are adopted from each category. From the categories, the total words are put in the T column, while the valid test data count is put in V column. After lemmatizing process, the data count is put in S column. Despite of the successful process, error is also occurred. The total error/failure of the data is put in E column. At the end, precision is calculated by dividing S by V.

TABLE 7.
TEST RESULT FOR NON-UNIQUE COLLECTION

Category	FULL				
	T	V	S	E	P
Business	6344	5627	5550	77	0.98632
Regional	6470	4802	5846	81	0.98313
Education	4165	5927	3598	32	0.99460
Science	6246	5504	5398	73	0.98674
Sports	6231	3242	5522	42	0.98705
International	10953	3630	9917	75	0.97934
Megapolitan	3998	5471	3214	28	0.99488
National	5499	5564	4764	38	0.99317
Oasis	6087	9992	5462	42	0.99580
Travel	8379	7502	7457	45	0.99400
All	**64372**	**57261**	**56728**	**533**	**0.99069**

Table 7 shows that only non-unique, words that appears in many occurrences, are included, while table 8 shows only unique words.

TABLE 8.
TEST RESULT FOR UNIQUE COLLECTION

Category	UNIQUE				
	T	V	S	E	P
Business	1868	1580	1559	21	0.98671
Regional	1213	1011	995	16	0.98417
Education	868	637	623	14	0.97802
Science	874	643	630	13	0.97978
Sports	838	608	604	4	0.99342
International	2037	1593	1575	18	0.98870
Megapolitan	610	302	297	5	0.98344
National	559	326	324	2	0.99387
Oasis	820	528	524	4	0.99242
Travel	892	611	607	4	0.99345
All	**10579**	**7839**	**7738**	**101**	**0.98712**

Where:
T = Total data count V = Valid test data count S = Successful lemmatization
E = Error / Failures P = Precision

As seen on the table, it can be concluded that the lemmatization algorithm works well in Bahasa, either used in non-unique collection or unique collection.

V. CONCLUSION AND SUGGESTION

Based on the test result, we have shown that our lemmatization method achieved a fairly high precision of 0.98. Even though there are still inaccuracies, it is considerably viable to use for implementation, such as morphological analysis, grammar analyzer, and other linguistic applications in the context of Indonesian.
And any suggestions for the next research are:

a. Enhance this algorithm with some words exception. Not all words in Bahasa can follow the rules. Sometimes, some exceptions have to be made because of word context and language transition.
b. Improve this algorithm to be able to receive sentences as input. For now, it could only receive a word as input.
c. Use this lemmatization method as a basic to make a morphological analysis algorithm, since it is a key to actualize many useful applications.
d. Enhance this algorithm to handle repetitive words, words with infixes, proper nouns, abbreviations, and foreign words.

REFERENCES

[1] Adriani, M., Asian, J., Nazief, B., Tahaghoghi, S., & Williams, H. (2007). Stemming Indonesian: A Confix-Stripping Approach. *ACM Transactions on Asian Language Information Processing*, 6(4), pp. 1 – 33.
[2] Asian, J., Williams, H., & Tagaghoghi, S. (2005). Stemming Indonesian. *ACSC '05 Proceedings of the Twenty-eighth Australasian on Computer Science*, 38, pp. 307-314.
[3] Arifin, A., Ciptaningtyas, H., & Mahendra, I. (2009). Enhanced Confix Stripping Stemmer And Ants Algorithm For Classifying News Document In Indonesian Language. *The International Conference on Information & Communication Technology and Systems*, 5, pp. 149-158.
[4] Ingason, K., Helgadóttir, S., Loftsson, H., Rögnvaldsson, E. (2008). A Mixed Method Lemmatization Algorithm Using a Hierarchy of Linguistic Identities (HOLI). Aarne Ranta (Eds,). *Advances in Natural Language Processing*.
[5] Kowalski, M. (2011). *Information Retrieval Architecture and Algorithms*. New York: Springer.
[6] Manning, C. D., Raghavan, P., Schütze, H. (2009). *An Introduction to Information Retrieval*. Cambridge: Cambridge University Press.
[7] Nirenburg, S. (2009). *Language Engineering for Lesser-Studied Languages*. Amsterdam: IOS Press.
[8] Poole, D. L., Mackworth A. K. (2010). *Artificial Intelligence Foundations of Computational Agents*. Cambridge: Cambridge University Press.
[9] Tucker, T. G. (2010). *Introduction to the Natural History of Language (1908).* USA: Kessinger Publishing..

Derwin Suhartono was born on January 24th, 1988. He has completed his bachelor and master degree majoring Computer Science in BINUS University, Jakarta, Indonesia since 2011.

He is a lecturer of Intelligent System field in BINUS University, Jakarta, Indonesia. He is also a researcher in Intelligent System. His previous work was as Java Developer. He was developing web site for banking and telecommunication in probably 1 year. He has published several international paper and also some national paper in the same year. His interest is in Natural Language Processing, Expert System, and Intelligent System Application.

David Christiandy was born on September 17th, 1991. He has completed his bachelor degree majoring Computer Science in BINUS University, Jakarta, Indonesia since 2013.

He is a student of software engineering field in BINUS University, Jakarta, Indonesia. He did a research in artificial intelligence topic about lemmatization for Indonesian language. He is currently working as a web developer, specifically in front-end web development and back-end web development. His interests are in Algorithms, Web Development, and Software Engineering.

Rolando was born on May 14th, 1991. He has completed his bachelor degree majoring Computer Science in BINUS University, Jakarta, Indonesia since 2013.

He is a student of software engineering field in BINUS University, Jakarta, Indonesia. While he was studying in university, he also worked as a web programmer and a front end developer, specializing in PHP, HTML5, CSS3, and JAVASCRIPT.

In the end of his study period in BINUS University, he did some research in artificial intelligence field about Indonesian language lemmatization.