Querying texts with the AntConc corpus software

Gede Primahadi Wijaya Rajeg[1, 2, 3], **Luh Putu Laksminy**[1], **I Made Rajeg**[1, 3]

[1] Faculty of Humanities, Udayana University, Indonesia
[2] Centre for Interdisciplinary Research on the Humanities and Social Sciences (CIRHSS @cirhss_unud)
[3] Computer-assisted Lexicology and Lexicography (CompLexico@complexico.unud) research group

Article Info

Keywords:

AntConc;
Corpus Linguistics;
Indonesian;
Reduplication;
Regular Expressions

ABSTRACT

AntConc is an open-source corpus linguistic tool featuring basic corpus linguistic analytical techniques. A powerful element of AntConc is its various pattern-searching methods. This paper describes the flavours of AntConc's searching methods and illustrates them for the query of phonological, morphological, and syntactic phenomena. The more advanced searching method, namely Regular Expressions (RegEx), is also discussed. The paper illustrates the use of RegEx for querying a rather complex morphological phenomenon in Indonesian, namely reduplication. RegEx search output is compared with the output from the basic Wildcard search feature to accentuate the sophistication and flexibility of RegEx for more targeted and specific results. The paper, overall, underscores the importance of mastering the different methods of querying texts with different complexities in AntConc for investigating a wider range of data-intensive, linguistic inquiries.

Corresponding Author:
Gede Primahadi W. Rajeg
CIRHSS and CompLexico;
primahadi_wijaya@unud.ac.id

This is an open access article under the CC BY-NC-SA license.

1. INTRODUCTION

Corpus Linguistics is a scientific approach to the study of language that has been defined in terms of relative distribution of a linguistic phenomenon found in a linguistic corpus (corpora, for plural) (i.e., in a large digital collection of texts from naturalistic language use) (cf. Stefanowitsch, 2020, p. 56). Given the sheer size of today's corpora, the use of corpus linguistic softwares is central for corpus linguistic investigation because they accelerate the access and query to large collection of digital texts (instead of accessing them manually by hand). There are two general ways to access a language corpus: via installation-based software or via the web-based interface (for a recent overview, see Anthony, 2022). One of the well-known, free installation-based corpus linguistic softwares is *AntConc* (Anthony, 2024). AntConc allows searching for forms/strings in large collection of text faster than manually scanning the forms through reading each text one by one. It also provides a different range of tools central for corpus linguistic key analytical techniques, namely Key Word in Context (KWIC) or Concordance display, collocate analyses, n-grams, (word-)cluster, and keywords analyses. Results produced in AntConc can be exported and saved for further processing and analyses.

After defining one's research questions or hypotheses, corpus studies typically proceed with computationally querying the corpus for strings, that is, sequences of orthographic characters (be they word forms, phrases, or labels/annotations for certain grammatical constructions) (cf. Stefanowitsch, 2020, p. 105). The searches are performed to retrieve potential dataset to answer the research questions or test the research hypotheses (Paquot & Gries, 2020, p. v). Amongst the potential and typical searches that we might perform include (a set of) exact, individual word form(s) (e.g., finding all occurrences of the Indonesian verb *memberi* 'to give'). Also, we might often wish to search for possible variation of word forms (e.g., morphological variation) that are based on a given root (e.g., different morphological variations of word forms based on the Indonesian noun root *hati* 'liver'). It is indeed possible for us to perform separate, individual searches for each theoretically possible variation of word forms from a given root. However, over time, this method practically becomes exhausting, time-consuming, and is prone to missing forms that we have not been aware of. Corpus software such as AntConc (and many other installation-based or web-based corpus tools) provides a range of flexible and efficient ways of capturing possible variation in forms/strings from the text corpus that can be further verified manually after the search.

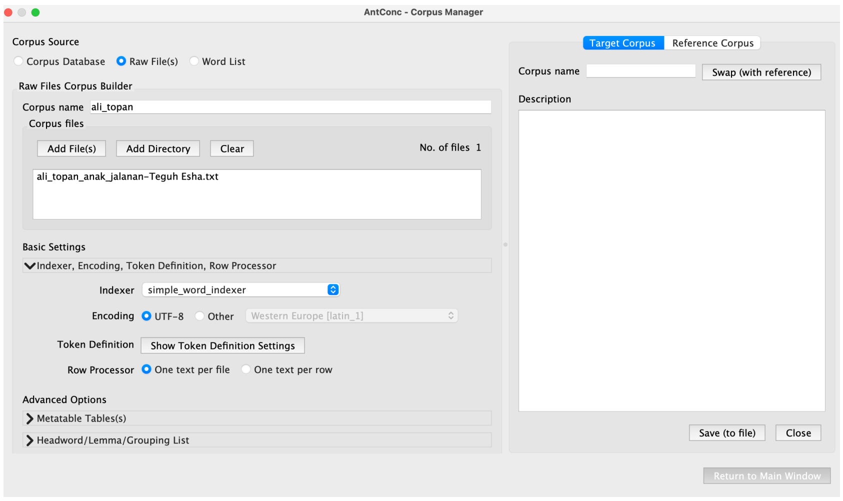
This paper describes various strings-/texts-searching mechanisms available in AntConc version 4.3.1 (cf. Rajeg, 2020). As part of the discussion, it will highlight certain linguistic phenomena relevant for particular use of the searching methods. The example corpus to be used is Indonesian (with Roman script and without any annotations on the corpus text). The searching mechanism will be applicable and extensible for corpus of different languages. The paper, therefore, invites readers' critical imagination on other possible phenomena they could uncover via AntConc's different searching methods for different languages.

2. METHODS

In this section, we describe the corpus files used in the paper (as well as their access) and a brief setup upon loading these corpus files into AntConc.

This paper uses two corpus files for illustrations. The first corpus is from the digitised novel *Ali Topan Anak Jalanan* (Esha, 1977). The second corpus file is from the Indonesian Leipzig Corpora Collection (ILCC) (Goldhahn et al., 2012; Quasthoff & Goldhahn, 2013), especially the re-classified ILCC (Nomoto et al., 2018) with the file name "ind_newscrawl-tufs6_2012_50K-sentences.txt". In order to reproduce the output from the search patterns to be discussed in this paper, readers can request access to these corpus files via these links: https://drive.google.com/file/d/1cZF9PaDNJYf5HzG3nxT0mfUkHyeSdfjx/view?usp=sharing and https://drive.google.com/file/d/1WdVkWt60oOKri1rdrivSiZ7krazW11Mp/view?usp=sharing. Download these corpus files into readers' own computer to be loaded into AntConc. When loading them into AntConc, we need to specify the definition of the tokens (or characters to be recognised by AntConc) via AntConc Corpus Manager. The following paragraphs illustrate these procedures using the first corpus file. The second corpus file is processed in the same manner.

Figure 1 shows that upon opening AntConc's Corpus Manager setup page, users check the Raw File(s) option (rather than Corpus Database and Word List). Next, under the Corpus name field, type in the name of the corpus (in this example, it is ali_topan, or whatever name the users wish). Afterwards, click on Add File(s) and select the first corpus file downloaded from the Google Drive of the author. This corpus file name will then show up in the big white field. Below the big white field, there are three Basic Settings options. One of these, namely "Indexer, Encoding, Token Definition, Row Processor", contains settings to determine the token definition (i.e., characters definition to be recognised later when searching the corpus via AntConc). Click on the right arrowhead > to reveal a drop-down menu of "Indexer, Encoding, Token Definition, Row Processor". After that, click on "Show Token Definition Settings" and a new window will open as shown in Figure 2.



Corpus Source: Corpus Database [✓ Raw File(s)] Word List

Corpus name: ali_topan

No. of files: 1

ali_topan_anak_jalanan-Teguh Esha.txt

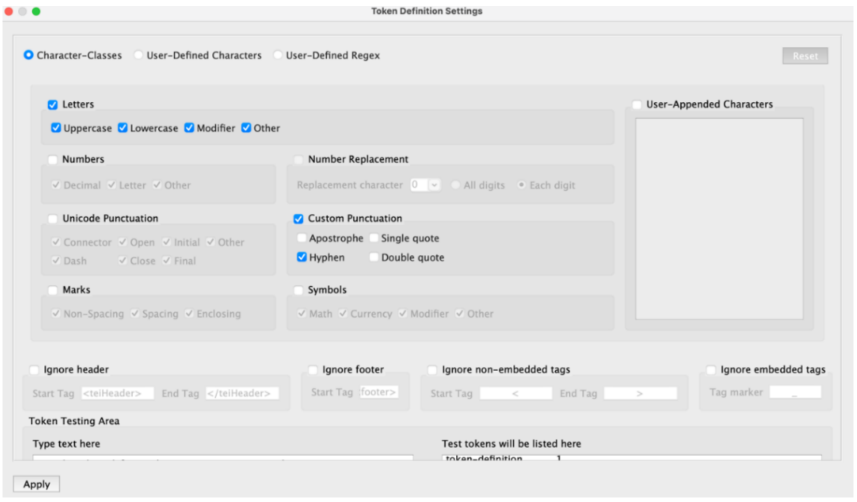
Indexer: simple_word_indexer

Encoding: [✓ UTF-8] Other (Western Europe [latin_1])

Row Processor: [✓ One text per file] One text per row

Figure 1. Corpus Manager interface

In the Character-Classes option in Figure 2, readers can activate/check the relevant tokens/characters classes like the ones shown in Figure 2. That is, check the Hypen of the Custom Punctuation (to allow searching for reduplication form in Indonesian context). After that, click the "Apply" button.


Token Definition Settings

[✓ Character-Classes] User-Defined Characters User-Defined Regex

[✓ Letters]: [✓ Uppercase] [✓ Lowercase] [✓ Modifier] [✓ Other]

Numbers: Decimal, Letter, Other

Number Replacement: Replacement character 0, All digits, Each digit

Unicode Punctuation: Connector, Open, Initial, Other, Dash, Close, Final

[✓ Custom Punctuation]: Apostrophe, Single quote, [✓ Hyphen], Double quote

Marks: Non-Spacing, Spacing, Enclosing

Symbols: Math, Currency, Modifier, Other

User-Appended Characters

Ignore header: Start Tag <teiHeader> End Tag </teiHeader>

Ignore footer: Start Tag footer>

Ignore non-embedded tags: Start Tag < End Tag >

Ignore embedded tags: Tag marker _

Token Testing Area

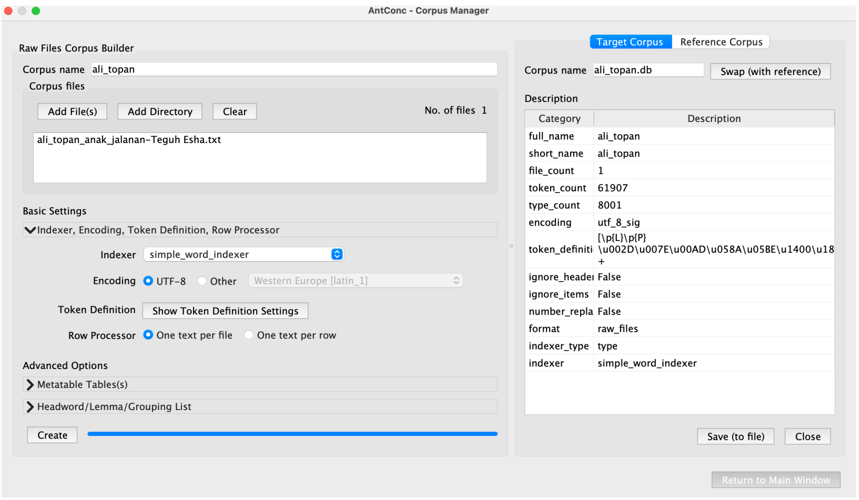
Type text here

Test tokens will be listed here

token-definition 1

Figure 2. Token Definition Settings interface

Users/readers will be taken back to the Corpus Manager interface. Next, hit the “Create” button to load/create the corpus for use in AntConc. The final interface upon creating the corpus will look like Figure 3 below. After that, click Return to Main Window to the main AntConc interface.


AntConc - Corpus Manager

Raw Files Corpus Builder

Corpus name: ali_topan

Corpus files — No. of files 1

ali_topan_anak_jalanan-Teguh Esha.txt

Basic Settings

Indexer, Encoding, Token Definition, Row Processor

Indexer: simple_word_indexer

Encoding: [✓ UTF-8] Other — Western Europe [latin_1]

Token Definition: Show Token Definition Settings

Row Processor: [✓ One text per file] One text per row

Advanced Options

Metatable Tables(s)

Headword/Lemma/Grouping List

Target Corpus | Reference Corpus

Corpus name: ali_topan.db

Description

Category	Description
full_name	ali_topan
short_name	ali_topan
file_count	1
token_count	61907
type_count	8001
encoding	utf_8_sig
token_definiti	[\p{L}\p{P}\u002D\u007E\u00AD\u058A\u05BE\u1400\u18 +
ignore_header	False
ignore_items	False
number_repla	False
format	raw_files
indexer_type	type
indexer	simple_word_indexer

Figure 3. Corpus Manager interface after defining token and creating the corpus

Repeat the procedures for the ILCC file. Afterwards, readers/users can follow along the tutorial of the searching mechanisms available in AntConc.

3. RESULTS AND DISCUSSION

This section discusses results/outputs of querying text-patterns in the corpus via the two main search-methods in AntConc, namely Wildcards and Regular Expressions (RegEx). The specific case study of the Indonesian reduplication will be discussed in the RegEx subsection. For the reduplication topic, comparison of the output between the RegEx and the Wildcard methods will be described.

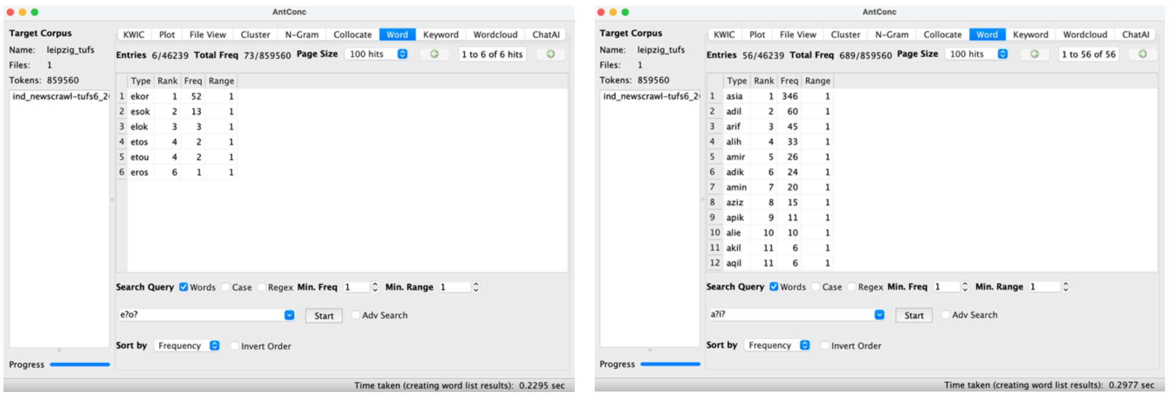
Wildcards search

A Wildcard is a symbol that represents any characters (Bothma, 2017, p. 206). It is also known as the “the don’t care character” (Lewenstein et al., 2014, p. 121). In the context of AntConc, the recognised characters (to be captured by the Wildcards) are those that have been defined in the Token Definition Settings upon loading the corpus files (Figure 2); by default, the characters/tokens set in AntConc are only (lower case and upper case) alphabets/letters. The Wildcards in AntConc (available under the Global Settings > Searches) are mainly concerned with two aspects: (i) quantification of the symbolised characters and (ii) alternative search. Each aspect is represented by different symbols, which will be discussed next, together with elementary query examples.

Searching for any one character with the "?" symbol

The first Wildcard in AntConc is the question mark (`?`) symbol. It represents and will capture any single character (i.e., token). It can be used (i) at any position (i.e., in the beginning, middle, and end) of the search pattern and (ii) for any number of occurrences depending on the need of the researchers.

The use of this `?` character can be driven by a linguistic research question. For example, we could study the productivity of two-syllable word types in Indonesian containing certain vowel combination pairs (e.g., *a-u*, *a-o*, *i-o, i-e*, among others) with such a syllabic structure as [vowel$_1$+any-one-character+vowel$_2$+any-one-character]. To answer this question, we query AntConc with search patterns composed of the target vowel combination and the `?` (e.g., `e?o?`, `a?i?`, etc.). Running the pattern `e?o?` in the Word menu of AntConc produces only six entries/word types in the ILCC corpus file (the second corpus file) (see the left panel of Figure 4). In contrast, the pattern `a?i?` conforms to fifty-six entries, nearly ten times as many word types as the first pattern (the right panel of Figure 4). The two results suggest that there are more word types in Indonesian with the second pattern of vowel combination than with the first one.



Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

Entries 6/46239 Total Freq 73/859560 Page Size 100 hits 1 to 6 of 6 hits

	Type	Rank	Freq	Range
1	ekor	1	52	1
2	esok	2	13	1
3	elok	3	3	1
4	etos	4	2	1
5	etou	4	2	1
6	eros	6	1	1

Search Query [✓ Words] Case Regex Min. Freq 1 Min. Range 1
e?o?
Sort by Frequency Invert Order
Time taken (creating word list results): 0.2295 sec

Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

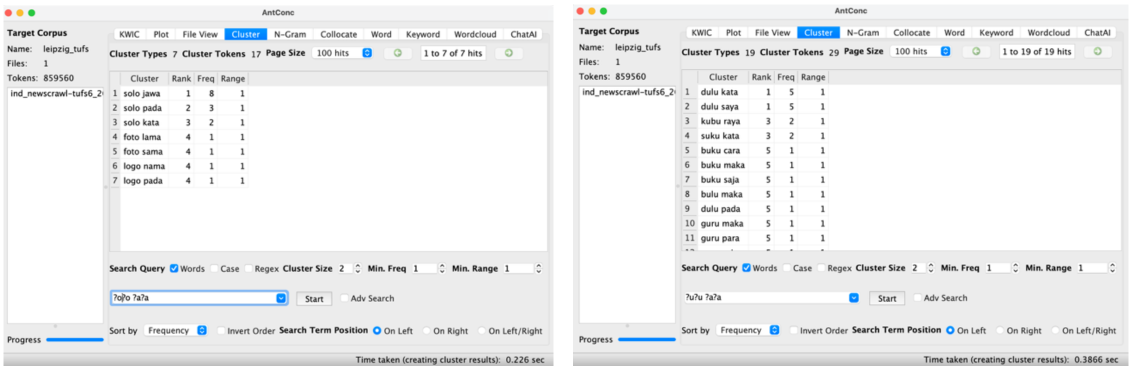
Entries 56/46239 Total Freq 689/859560 Page Size 100 hits 1 to 56 of 56

	Type	Rank	Freq	Range
1	asia	1	346	1
2	adil	2	60	1
3	arif	3	45	1
4	alih	4	33	1
5	amir	5	26	1
6	adik	6	24	1
7	amin	7	20	1
8	aziz	8	15	1
9	apik	9	11	1
10	alie	10	10	1
11	akil	11	6	1
12	aqil	11	6	1

Search Query [✓ Words] Case Regex Min. Freq 1 Min. Range 1
a?i?
Sort by Frequency Invert Order
Time taken (creating word list results): 0.2977 sec

Figure 4. Wordlist output from wildcard query with the ? symbol

Note that AntConc's Wildcards feature can not only be useful for querying word forms. It can also be embedded in search patterns to query phrasal expressions. To stay with our example of four-letter, two-syllable word types, we could compare the number of two-word phrases in the ILCC that contain certain vowel combination types. To illustrate this, we queried two patterns: `?o?o ?a?a` and `?u?u ?a?a`. The question is which of these two queries realises higher number of phrases. We now use the Cluster menu of AntConc to retrieve word-cluster/phrase conforming to the search patterns. The output shown in Figure 5 indicates that there are more phrases with `?u?u ?a?a` string (right panel) than with the `?o?o ?a?a` one (left panel).



Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

Cluster Types 7 Cluster Tokens 17 Page Size 100 hits 1 to 7 of 7 hits

	Cluster	Rank	Freq	Range
1	solo jawa	1	8	1
2	solo pada	2	3	1
3	solo kata	3	2	1
4	foto lama	4	1	1
5	foto sama	4	1	1
6	logo nama	4	1	1
7	logo pada	4	1	1

Search Query [✓ Words] Case Regex Cluster Size 2 Min. Freq 1 Min. Range 1
?o?o ?a?a
Sort by Frequency Invert Order Search Term Position [✓ On Left] On Right On Left/Right
Time taken (creating cluster results): 0.226 sec

Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

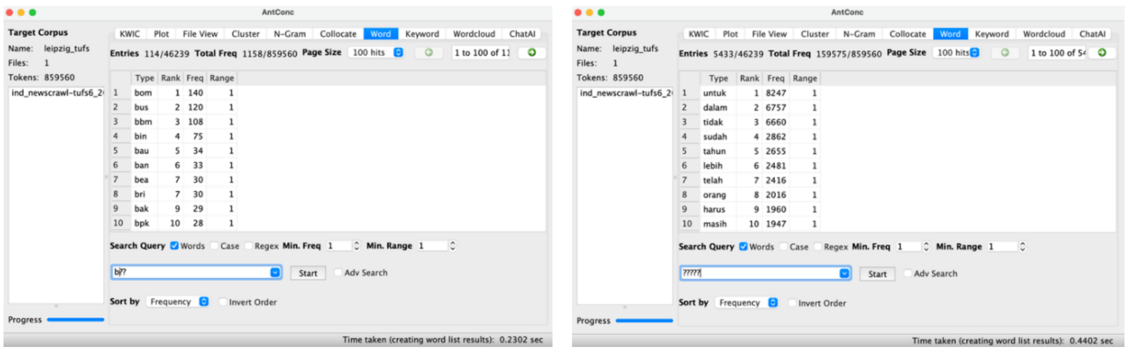
Cluster Types 19 Cluster Tokens 29 Page Size 100 hits 1 to 19 of 19 hits

	Cluster	Rank	Freq	Range
1	dulu kata	1	5	1
2	dulu saya	1	5	1
3	kubu raya	3	2	1
4	suku kata	3	2	1
5	buku cara	5	1	1
6	buku maka	5	1	1
7	buku saja	5	1	1
8	bulu maka	5	1	1
9	dulu pada	5	1	1
10	guru maka	5	1	1
11	guru para	5	1	1

Search Query [✓ Words] Case Regex Cluster Size 2 Min. Freq 1 Min. Range 1
?u?u ?a?a
Sort by Frequency Invert Order Search Term Position [✓ On Left] On Right On Left/Right
Time taken (creating cluster results): 0.3866 sec

Figure 5. Output of phrasal search with the ? symbol

Up to now, the use of `?` is demonstrated as if it is only needed once (i.e., symbolising a single item). However, it can also be multiplied in the search pattern to capture the exact number of (any) character to be retrieved. For example, the search pattern `b??` will capture word forms started with *b* followed by exactly any two characters (the left panel of Figure 6), while `?????` retrieves exactly five-letter words (the right panel).



Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

Entries 114/46239 Total Freq 1158/859560 Page Size 100 hits 1 to 100 of 11

	Type	Rank	Freq	Range
1	bom	1	140	1
2	bus	2	120	1
3	bbm	3	108	1
4	bin	4	75	1
5	bau	5	34	1
6	ban	6	33	1
7	bea	7	30	1
8	bri	7	30	1
9	bak	9	29	1
10	bpk	10	28	1

Search Query ☑ Words ☐ Case ☐ Regex Min. Freq 1 Min. Range 1
b??
Sort by Frequency ☐ Invert Order
Time taken (creating word list results): 0.2302 sec

Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

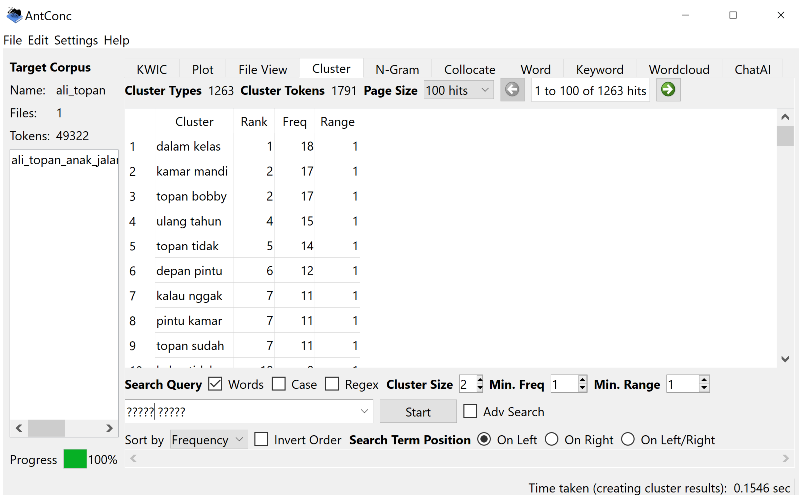
Entries 5433/46239 Total Freq 159575/859560 Page Size 100 hits 1 to 100 of 54

	Type	Rank	Freq	Range
1	untuk	1	8247	1
2	dalam	2	6757	1
3	tidak	3	6660	1
4	sudah	4	2862	1
5	tahun	5	2655	1
6	lebih	6	2481	1
7	telah	7	2416	1
8	orang	8	2016	1
9	harus	9	1960	1
10	masih	10	1947	1

Search Query ☑ Words ☐ Case ☐ Regex Min. Freq 1 Min. Range 1
?????
Sort by Frequency ☐ Invert Order
Time taken (creating word list results): 0.4402 sec

Figure 6. Quantifying character search with the ? symbol

The same technique can be applied when we wish to search for phrases with the exact number of characters (Figure 7 below). The corpus used in this example is from the first corpus file (*Ali Topan anak jalanan*). In Figure 7, the Wildcard query `????? ?????` means finding two-word combination whereby the first and second elements are words composing of five letters.



Target Corpus
Name: ali_topan
Files: 1
Tokens: 49322
ali_topan_anak_jala

Cluster Types 1263 Cluster Tokens 1791 Page Size 100 hits 1 to 100 of 1263 hits

	Cluster	Rank	Freq	Range
1	dalam kelas	1	18	1
2	kamar mandi	2	17	1
3	topan bobby	2	17	1
4	ulang tahun	4	15	1
5	topan tidak	5	14	1
6	depan pintu	6	12	1
7	kalau nggak	7	11	1
8	pintu kamar	7	11	1
9	topan sudah	7	11	1

Search Query ☑ Words ☐ Case ☐ Regex Cluster Size 2 Min. Freq 1 Min. Range 1
????? ?????
Sort by Frequency ☐ Invert Order Search Term Position ◉ On Left ○ On Right ○ On Left/Right
Progress 100%
Time taken (creating cluster results): 0.1546 sec

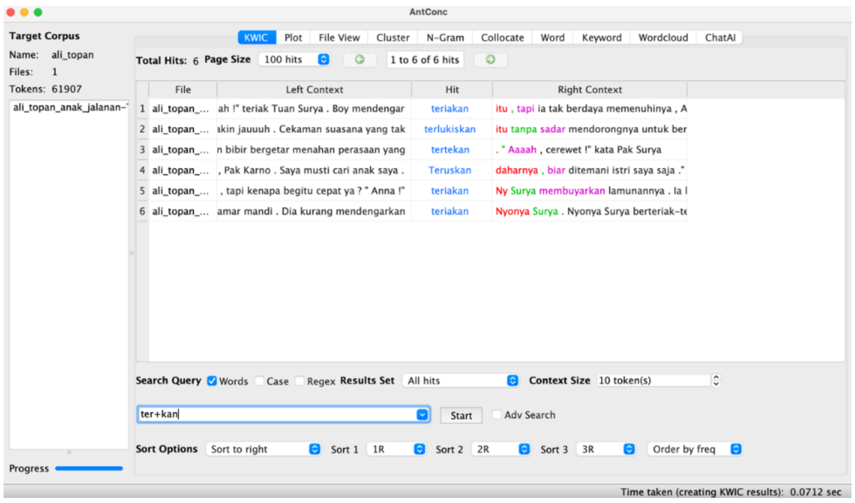
Figure 7. Phrasal search with exact number of characters

The examples discussed here specifies the exact number of times the `?` needs to be repeated to capture forms (e.g., words or phrases) with exact number of characters. In the next section, we will look at another kind of Wildcard that is more flexible than `?` in terms of the way character numbers are quantified and retrieved.

Searching for one or more character with the "+" symbol

The + symbol is more flexible than the ? one because the former is greedier than the latter. This means that a single appearance of + in a search pattern will not only capture any single character like ?, but also more characters. Therefore, the + is efficient if we are not definitive regarding the exact number of characters we want for the search pattern to capture. As with the `?` symbol, the + character can be put at the beginning, in the middle, or at the end of a search pattern.

One useful application of the + in a search pattern is to query potential complex words conforming to a given morphological construction. For example, we might be interested in the potential roots appearing in Indonesian stative passive morphological schema [*ter-*/*-kan*] (cf. Arka, 2010). To answer this, we designed a searching pattern under the following assumption. First, we have known that the construction has specific (presumed) morphological elements, namely the prefix *ter-* and the suffix *-kan*; these elements can be specified explicitly in the search pattern. Second, the variable element in the target construction is the form of the root (which can be more than one-character/letter long); it is in this situation that the + can help capture such variability/non-specificity of the root-candidate. Given these assumptions, the search pattern `ter+kan` is proposed. It will generate word forms (i) started with the string *ter* (ii) followed by any one or more characters (+) and then (iii) ended with the string *kan*. Figure 8 shows the output of the search pattern in the first corpus.



Target Corpus
Name: ali_topan
Files: 1
Tokens: 61907
ali_topan_anak_jalanan-

Total Hits: 6 Page Size 100 hits 1 to 6 of 6 hits

	File	Left Context	Hit	Right Context
1	ali_topan_...	ah !" teriak Tuan Surya . Boy mendengar	teriakan	itu , tapi ia tak berdaya memenuhinya , A
2	ali_topan_...	ıkin jauuuh . Cekaman suasana yang tak	terlukiskan	itu tanpa sadar mendorongnya untuk ber
3	ali_topan_...	n bibir bergetar menahan perasaan yang	tertekan	. " Aaaah , cerewet !" kata Pak Surya
4	ali_topan_...	, Pak Karno . Saya musti cari anak saya .	Teruskan	daharnya , biar ditemani istri saya saja ."
5	ali_topan_...	, tapi kenapa begitu cepat ya ? " Anna !"	teriakan	Ny Surya membuyarkan lamunannya . Ia l
6	ali_topan_...	amar mandi . Dia kurang mendengarkan	teriakan	Nyonya Surya . Nyonya Surya berteriak-te

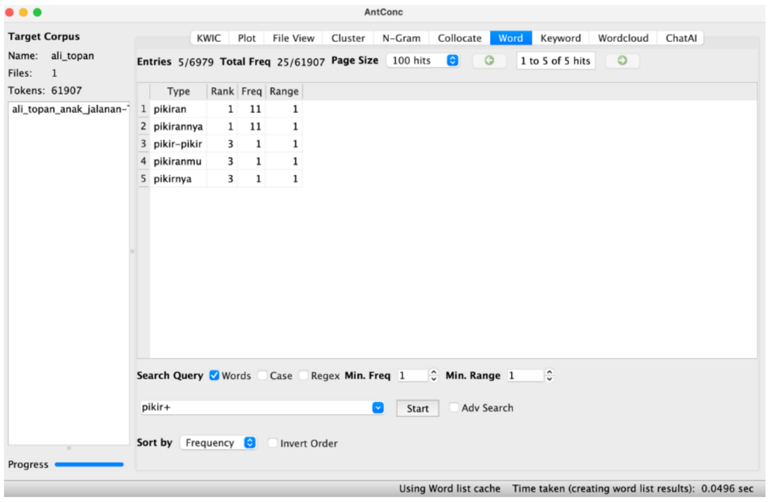
Search Query ✓ Words ☐ Case ☐ Regex Results Set All hits Context Size 10 token(s)
ter+kan ☐ Adv Search
Sort Options Sort to right Sort 1 1R Sort 2 2R Sort 3 3R Order by freq
Time taken (creating KWIC results): 0.0712 sec

Figure 8. Querying word forms under the [ter-/-kan] *schema with the + symbol*

We can observe that only one of the outputs reflects what we might assume to be the morphological schema of the *ter*- prefix followed by a root and ended with the -*kan* suffix. This singleton is the form *ter**lukis**kan* 'be depicted', based on the root *lukis* 'to paint' (row number 3). The remaining outputs do reflect what the search pattern `ter+kan` will theoretically capture, namely forms with (i) the strings *ter* (regardless of its morphological status as a prefix or part of a word), (ii) any one or more characters (the + character), (iii) and the strings *kan* (also, regardless of its morphological status as a suffix or part of a word). The form *ter**us**kan* contains (i) the strings *ter* (part of the word ***terus*** 'keep.going'), (ii) one or more characters (i.e., the strings *us*, part of *ter**us***), and (iii) the strings *kan*, which happens to be the suffix -*kan* in its causative function in the form ***terus**kan* 'cause to keep going; to continue'. On the contrary, for the form *teri**ak**an*, the letter *k* in the strings *kan* is part of the word ***teriak*** 'to shout' (not of the suffix -*kan*) while the *ter* is also not the *ter*- prefix (but part of the word ***teriak***). Only the *an* part of the strings *kan* (in *teriakan* '[the] shouting') reflects the nominalising suffix -*an*; *teriakan* is a combination of the root *teriak* 'to shout' and the suffix -*an*, meaning 'the shouting', but captured via the `ter+kan` search pattern.

In short, if we thought of searching for word forms reflecting the Indonesian morphological construction [*ter*-/-*kan*], the search pattern `ter+kan` could indeed be used. However, as we have seen, we still need to perform manual verification to check if the output contains false positives (i.e., forms captured by the search pattern [positive] but not representing our conceptually/theoretically intended output [false]).

Another use case of + for morphology is to use it at the end of the search pattern. This will search for any ending strings (that could be a suffix) for a given word form. For example, if we are interested in potential suffixation of the root *pikir* 'to think', the search pattern `pikir+` could be used (see Figure 9 for the output).



AntConc

Target Corpus
Name: ali_topan
Files: 1
Tokens: 61907
ali_topan_anak_jalanan-

Entries 5/6979 Total Freq 25/61907 Page Size 100 hits 1 to 5 of 5 hits

	Type	Rank	Freq	Range
1	pikiran	1	11	1
2	pikirannya	1	11	1
3	pikir-pikir	3	1	1
4	pikiranmu	3	1	1
5	pikirnya	3	1	1

Search Query ✓ Words ☐ Case ☐ Regex Min. Freq 1 Min. Range 1

pikir+ Start ☐ Adv Search

Sort by Frequency ☐ Invert Order

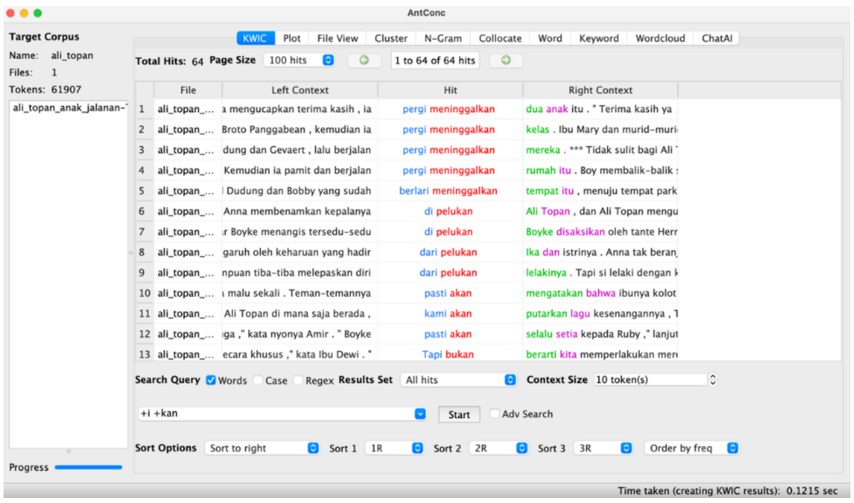
Progress

Using Word list cache Time taken (creating word list results): 0.0496 sec

Figure 9. Output of querying potential suffixation with the + symbol

An interesting output in Figure 9 includes a full reduplication *pikir-pikir* 'to think sth. over/through'. Based on this, the + in `pikir+` apparently captures the hyphen (-) and any existing letters that follow *pikir*. The inclusion of the hyphen is the consequence of our token/character definition setup (see Figure 2). In the setup, we included/selected the hyphen (in addition the alphabets) as part of the token definition to be recognised by AntConc search mechanism. Since both the alphabets and hyphen are set as tokens/characters, the + can capture these two sets of tokens/characters.

Like the question mark (`?`) character before, the + character can also be incorporated into querying multiword expressions. For example, the search pattern `+i +kan` in Figure 10 captures two forms: one form ends with the letter *i* followed by a whitespace and then another form ended with the strings *kan*. This search pattern illustrates the use of + at the start of the searching pattern.



AntConc

Target Corpus
Name: ali_topan
Files: 1
Tokens: 61907
ali_topan_anak_jalanan-

Total Hits: 64 Page Size 100 hits 1 to 64 of 64 hits

	File	Left Context	Hit	Right Context
1	ali_topan_...	a mengucapkan terima kasih , ia	pergi meninggalkan	dua anak itu . " Terima kasih ya
2	ali_topan_...	Broto Panggabean , kemudian ia	pergi meninggalkan	kelas . Ibu Mary dan murid-muri
3	ali_topan_...	dung dan Gevaert , lalu berjalan	pergi meninggalkan	mereka . *** Tidak sulit bagi Ali
4	ali_topan_...	Kemudian ia pamit dan berjalan	pergi meninggalkan	rumah itu . Boy membalik-balik
5	ali_topan_...	Dudung dan Bobby yang sudah	berlari meninggalkan	tempat itu , menuju tempat park
6	ali_topan_...	Anna membenamkan kepalanya	di pelukan	Ali Topan , dan Ali Topan mengu
7	ali_topan_...	r Boyke menangis tersedu-sedu	di pelukan	Boyke disaksikan oleh tante Herr
8	ali_topan_...	garuh oleh keharuan yang hadir	dari pelukan	Ika dan istrinya . Anna tak beran
9	ali_topan_...	npuan tiba-tiba melepaskan diri	dari pelukan	lelakinya . Tapi si lelaki dengan k
10	ali_topan_...	malu sekali . Teman-temannya	pasti akan	mengatakan bahwa ibunya kolot
11	ali_topan_...	Ali Topan di mana saja berada ,	kami akan	putarkan lagu kesenangannya , T
12	ali_topan_...	ga ," kata nyonya Amir . " Boyke	pasti akan	selalu setia kepada Ruby ," lanjut
13	ali_topan_...	ecara khusus ," kata Ibu Dewi . "	Tapi bukan	berarti kita memperlakukan mer

Search Query ✓ Words ☐ Case ☐ Regex Results Set All hits Context Size 10 token(s)

+i +kan Start ☐ Adv Search

Sort Options Sort to right Sort 1 1R Sort 2 2R Sort 3 3R Order by freq

Progress

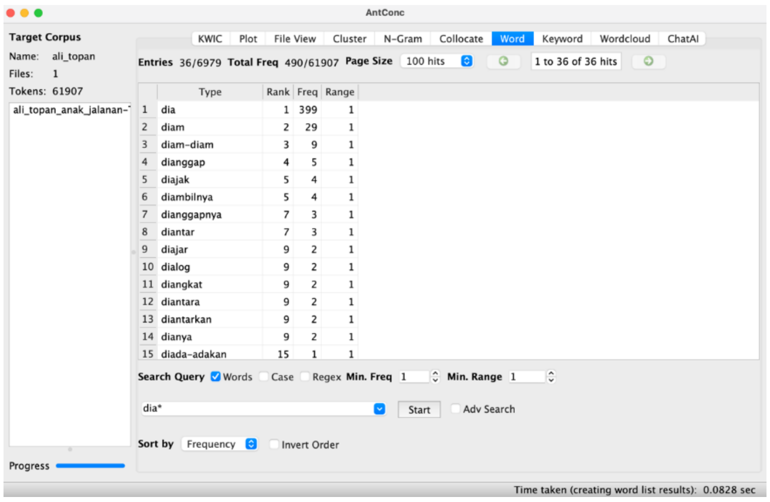
Time taken (creating KWIC results): 0.1215 sec

Figure 10. Output of phrasal search with the + symbol

The output does not necessarily reflect word-combination whereby the first element is a complex, derived word with the suffix *-i* and the second element is a complex, derived word with the suffix *-kan*.

Searching for zero, one or more character with the "*" symbol

The so-called "Kleene star" (Daintith & Wright, 2008) character (`*`) can represent zero, one, or more character (Friedl, 2006, p. 141). This symbol is the most flexible amongst the other Wildcard symbols in AntConc. The search pattern `dia*`, for instance, would generate a word (i) as minimum as *dia* (because * can represent nonce/zero character following the string `dia`) or (ii) as long as *di**am*** 'silent', *Di**ana***, *di**apit*** 'be sandwiched', *di**am-diam*** 'silently', *di**ada-adakan*** 'be intentionally caused to exist', or *di**alamatkannya*** 'be addressed' (see Figure 11 for the output in the Word menu using the search pattern `dia*`).



Target Corpus
Name: ali_topan
Files: 1
Tokens: 61907
ali_topan_anak_jalanan-

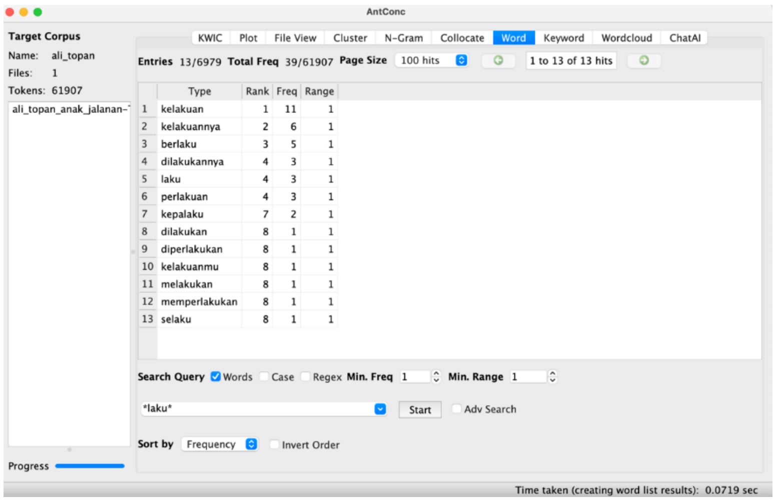
Entries 36/6979 Total Freq 490/61907 Page Size 100 hits 1 to 36 of 36 hits

	Type	Rank	Freq	Range
1	dia	1	399	1
2	diam	2	29	1
3	diam-diam	3	9	1
4	dianggap	4	5	1
5	diajak	5	4	1
6	diambilnya	5	4	1
7	dianggapnya	7	3	1
8	diantar	7	3	1
9	diajar	9	2	1
10	dialog	9	2	1
11	diangkat	9	2	1
12	diantara	9	2	1
13	diantarkan	9	2	1
14	dianya	9	2	1
15	diada-adakan	15	1	1

Search Query [✓ Words] Case Regex Min. Freq 1 Min. Range 1
dia*
Adv Search
Sort by Frequency Invert Order
Time taken (creating word list results): 0.0828 sec

Figure 11. Output of a query with the Kleene star symbol

In the context of Indonesian, the Kleene star (as in the previous Wildcard characters) could be used to find various morphological forms of words containing a certain string, which could represent a root (note that the output needs to be manually verified to determine the relevant related forms if the intention of the string was to represent a given root). For example, we could generate a list of word forms containing the string *laku* (with the search pattern `*laku*`), which we might assume to represent the root *laku* meaning 'act'.



Target Corpus
Name: ali_topan
Files: 1
Tokens: 61907
ali_topan_anak_jalanan-

Entries 13/6979 Total Freq 39/61907 Page Size 100 hits 1 to 13 of 13 hits

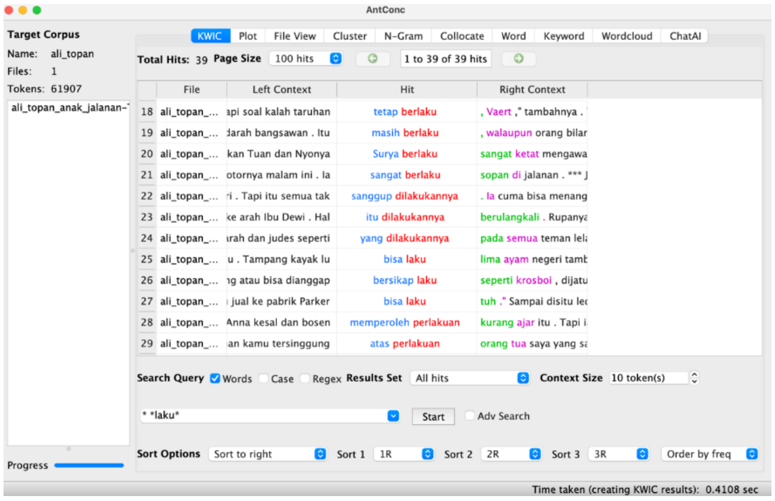
	Type	Rank	Freq	Range
1	kelakuan	1	11	1
2	kelakuannya	2	6	1
3	berlaku	3	5	1
4	dilakukannya	4	3	1
5	laku	4	3	1
6	perlakuan	4	3	1
7	kepalaku	7	2	1
8	dilakukan	8	1	1
9	diperlakukan	8	1	1
10	kelakuanmu	8	1	1
11	melakukan	8	1	1
12	memperlakukan	8	1	1
13	selaku	8	1	1

Search Query [✓ Words] Case Regex Min. Freq 1 Min. Range 1
laku
Adv Search
Sort by Frequency Invert Order
Time taken (creating word list results): 0.0719 sec

*Figure 12. Output of potential morphological search with the * symbol*

Looking at the output in Figure 12, we could observe that there are forms that are not completely related with the root-string *laku*. One such example is *kepa**laku*** 'my head' (see row number 7) that does contain the string *laku*; this form is based on the root *kepala* 'head' affixed with the first-person singular possessive suffix -*ku*. Another unrelated output is the string *laku* that, according to KBBI, represents the second sense of *laku*, namely 'selling well; sold out (of goods)'. The forms *kepalaku* and *laku* in its second sense represent false positives in the search pattern assumed to contain the root-string *laku* 'act'.

As in the previous sections, the Kleene star can be incorporated in a phrase search. For example, if we want to capture potential sequences of any words followed by words containing the string *laku* 'act' (in all its possible morphological forms), we could search for `*  *laku*`. This will produce, among other things, ***bisa** laku* 'can be sold out', ***atas perlakuan*** 'because of the treatment', ***tetap** berlaku* 'still valid', ***masih** berlaku* 'still valid', ***Saya** selaku* 'I am as the ...', ***tahu** kelakuannya* 'know h(is/er) act', etc. See Figure 13.



AntConc

Target Corpus
Name: ali_topan
Files: 1
Tokens: 61907
ali_topan_anak_jalanan-

Total Hits: 39 Page Size 100 hits 1 to 39 of 39 hits

	File	Left Context	Hit	Right Context
18	ali_topan_...	ɪpi soal kalah taruhan	tetap berlaku	, Vaert ," tambahnya .
19	ali_topan_...	darah bangsawan . Itu	masih berlaku	, walaupun orang bilar
20	ali_topan_...	kan Tuan dan Nyonya	Surya berlaku	sangat ketat mengawa
21	ali_topan_...	otornya malam ini . Ia	sangat berlaku	sopan di jalanan . *** J
22	ali_topan_...	ri . Tapi itu semua tak	sanggup dilakukannya	. Ia cuma bisa menang
23	ali_topan_...	ke arah Ibu Dewi . Hal	itu dilakukannya	berulangkali . Rupanya
24	ali_topan_...	ɪrah dan judes seperti	yang dilakukannya	pada semua teman lel
25	ali_topan_...	u . Tampang kayak lu	bisa laku	lima ayam negeri tamb
26	ali_topan_...	ɪg atau bisa dianggap	bersikap laku	seperti krosboi , dijatu
27	ali_topan_...	jual ke pabrik Parker	bisa laku	tuh ." Sampai disitu le
28	ali_topan_...	Anna kesal dan bosen	memperoleh perlakuan	kurang ajar itu . Tapi i
29	ali_topan_...	an kamu tersinggung	atas perlakuan	orang tua saya yang sa

Search Query ☑ Words ☐ Case ☐ Regex Results Set All hits Context Size 10 token(s)

* *laku* Start ☐ Adv Search

Sort Options Sort to right Sort 1 1R Sort 2 2R Sort 3 3R Order by freq

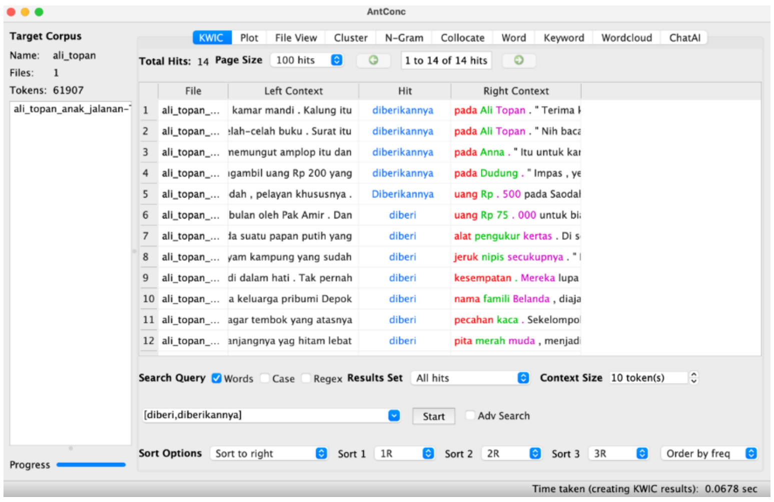
Progress

Time taken (creating KWIC results): 0.4108 sec

Figure 13. Output of phrasal search with the Kleene star

Alternative search

The alternative search Wildcard allows users to search for several alternative forms at once. For example, we can query if either *diberi* 'to be given' or *diberikannya* 'to be given by h(im/er)' exists in the corpus. AntConc provides two options to search for alternative expressions/patterns. The first is with the pattern `[yourpattern1,yourpattern2]`. The second is with `yourpattern1 || yourpattern2`. These two options will find either "`yourpattern1`" OR "`yourpattern2`". The screenshot in Figure 14 shows the KWIC output for the search pattern `[diberi,diberikannya]` (using the first alternative search option; it is important to note that no whitespace is allowed before and after the comma in this first option).



AntConc

Target Corpus
Name: ali_topan
Files: 1
Tokens: 61907
ali_topan_anak_jalanan-

Total Hits: 14 Page Size 100 hits 1 to 14 of 14 hits

	File	Left Context	Hit	Right Context
1	ali_topan_...	kamar mandi . Kalung itu	diberikannya	pada Ali Topan . " Terima k
2	ali_topan_...	elah-celah buku . Surat itu	diberikannya	pada Ali Topan . " Nih baca
3	ali_topan_...	nemungut amplop itu dan	diberikannya	pada Anna . " Itu untuk kar
4	ali_topan_...	ɪgambil uang Rp 200 yang	diberikannya	pada Dudung . " Impas , ye
5	ali_topan_...	dah , pelayan khususnya .	Diberikannya	uang Rp . 500 pada Saodal
6	ali_topan_...	bulan oleh Pak Amir . Dan	diberi	uang Rp 75 . 000 untuk bi
7	ali_topan_...	da suatu papan putih yang	diberi	alat pengukur kertas . Di s
8	ali_topan_...	yam kampung yang sudah	diberi	jeruk nipis secukupnya . "
9	ali_topan_...	di dalam hati . Tak pernah	diberi	kesempatan . Mereka lupa
10	ali_topan_...	a keluarga pribumi Depok	diberi	nama famili Belanda , diaja
11	ali_topan_...	agar tembok yang atasnya	diberi	pecahan kaca . Sekelompo
12	ali_topan_...	ɪnjangnya yag hitam lebat	diberi	pita merah muda , menjadi

Search Query ☑ Words ☐ Case ☐ Regex Results Set All hits Context Size 10 token(s)

[diberi,diberikannya] Start ☐ Adv Search

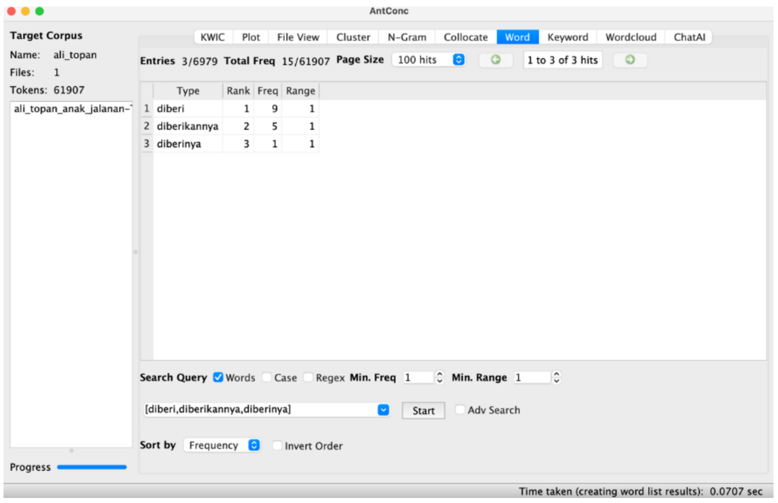
Sort Options Sort to right Sort 1 1R Sort 2 2R Sort 3 3R Order by freq

Progress

Time taken (creating KWIC results): 0.0678 sec

Figure 14. Output of running alternative query

Note that these two options for alternative searches are not restricted to two alternative expressions. We can search for more than two alternative expressions. Figure 15 shows this for querying *diberi* 'to be given', *diberikannya* 'to be given by h(im/er)', OR *diberinya* 'to be given by h(im/er)' under the Word menu.



Target Corpus
Name: ali_topan
Files: 1
Tokens: 61907
ali_topan_anak_jalanan-

Entries 3/6979 Total Freq 15/61907 Page Size 100 hits 1 to 3 of 3 hits

	Type	Rank	Freq	Range
1	diberi	1	9	1
2	diberikannya	2	5	1
3	diberinya	3	1	1

Search Query ✓ Words Case Regex Min. Freq 1 Min. Range 1
[diberi,diberikannya,diberinya] Start Adv Search
Sort by Frequency Invert Order
Progress
Time taken (creating word list results): 0.0707 sec

Figure 15. Output of the alternative query with more than two alternatives

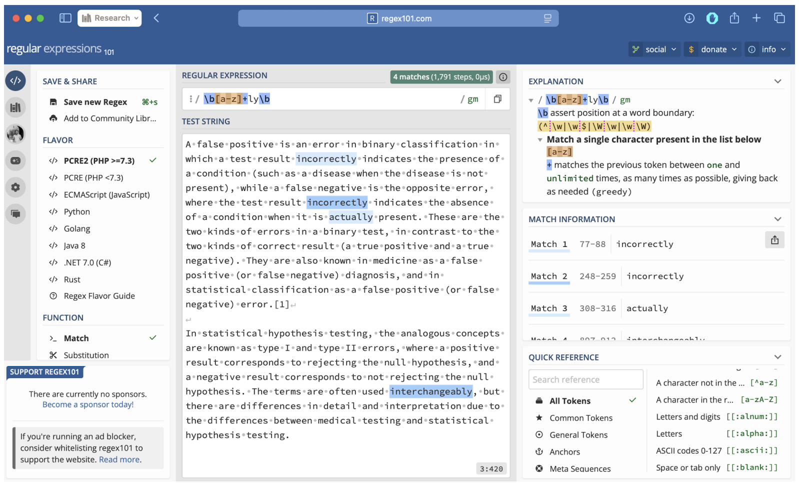
In the next section, we will look at a more powerful and flexible technique for querying text than the Wildcards discussed, namely Regular Expressions (RegEx).

Regular Expressions search

Regular Expressions (RegEx) is a kind of mini programming language specifically designed for flexible, powerful, and efficient text processing procedures. It is a "*generalized pattern language*" that can be used for various purposes (Friedl, 2006, p. 4; italics in original). One key use of RegEx in a corpus linguistic software such as AntConc is for designing (generalized) flexible and more precise search pattern that can fit and match strings of text from the corpus (Weisser, 2016; Stefanowitsch, 2020, p. 107). This searched text could be of various linguistic units, from phonology to syntax.

The way RegEx works resembles Wildcard whereby there are symbols that stand for certain character sets. For example, the symbol `\w` in *Perl-Compatible Regular Expression* (PCRE) stands for any word characters (namely alphabetic letters `[a-z]`, digits `[0-9]`, and an underscore `[_]`), while `\W` captures any non-word characters (e.g., whitespace, punctuations). These symbols, which mean or represent characters that are not their true identity/form, are called *special character* or *metacharacter* in RegEx (Friedl, 2006, p. 5). Other metacharacters include, amongst others, the dot `.`, symbolising any one character (i.e., punctuations, letters, numbers, etc.), or quantifiers such as `+`, which means/captures one or more characters that precedes the plus sign. For example, the RegEx pattern `yes+` will search for one or more occurrences of the letter *s*, that is *yes*, *yess*, *yesss*, *yesss,* and so on.

Compared to AntConc's Wildcard feature, RegEx offers expanded and more flexible searching mechanisms, as will be demonstrated shortly. It is important to note that RegEx has different "flavours" or dialects, depending on the programming language; PCRE is the most common one. In this section, we will illustrate some examples of RegEx search in AntConc. Readers are encouraged to explore RegEx further on their own. A good place to learn and test our regex before implementing/using it in AntConc as search pattern is the open webpage called regular expressions 101 (https://regex101.com) (see Figure 16).



REGULAR EXPRESSION 4 matches (1,791 steps, 0μs)

```
/ \b[a-z]+ly\b / gm
```

TEST STRING

```
A·false·positive·is·an·error·in·binary·classification·in·
which·a·test·result·incorrectly·indicates·the·presence·of·
a·condition·(such·as·a·disease·when·the·disease·is·not·
present),·while·a·false·negative·is·the·opposite·error,·
where·the·test·result·incorrectly·indicates·the·absence·
of·a·condition·when·it·is·actually·present.·These·are·the·
two·kinds·of·errors·in·a·binary·test,·in·contrast·to·the·
two·kinds·of·correct·result·(a·true·positive·and·a·true·
negative).·They·are·also·known·in·medicine·as·a·false·
positive·(or·false·negative)·diagnosis,·and·in·
statistical·classification·as·a·false·positive·(or·false·
negative)·error.[1]↵
↵
In·statistical·hypothesis·testing,·the·analogous·concepts·
are·known·as·type·I·and·type·II·errors,·where·a·positive·
result·corresponds·to·rejecting·the·null·hypothesis,·and·
a·negative·result·corresponds·to·not·rejecting·the·null·
hypothesis.·The·terms·are·often·used·interchangeably,·but·
there·are·differences·in·detail·and·interpretation·due·to·
the·differences·between·medical·testing·and·statistical·
hypothesis·testing.
                                                    3:420
```

EXPLANATION

/ \b[a-z]+ly\b / gm
\b assert position at a word boundary: (^\w|\w$|\W\w|\w\W)
Match a single character present in the list below [a-z]
+ matches the previous token between one and unlimited times, as many times as possible, giving back as needed (greedy)

MATCH INFORMATION

```
Match 1   77-88     incorrectly
Match 2   248-259   incorrectly
Match 3   308-316   actually
Match 4   897-912   interchangeably
```

QUICK REFERENCE

A character not in the ...	[^a-z]
A character in the r...	[a-zA-Z]
Letters and digits	[[:alnum:]]
Letters	[[:alpha:]]
ASCII codes 0-127	[[:ascii:]]
Space or tab only	[[:blank:]]

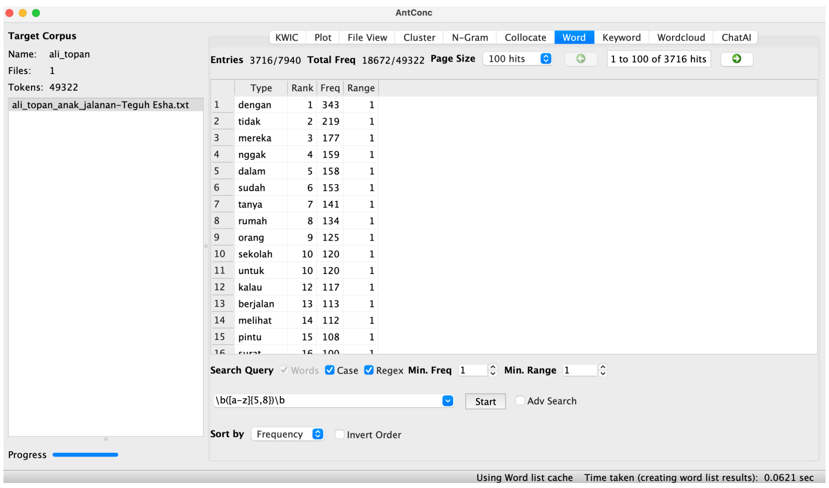
Figure 16. Interface of the regex101 webpage for learning regular expressions

Figure 16 shows the interface of regex101 webpage that has been given a sample text (by the authors) in the area called TEST STRING. The useful learning reference for the different literal characters and (the meaning of the) metacharacters in RegEx is available at the bottom right corner panel called QUICK REFERENCE. This *regex101* webpage is very useful because the matches are highlighted for the search pattern of the RegEx we type in. This allows us to inspect if the strings we intend to match with the RegEx largely match. The right-hand side panels called EXPLANATION provides more detailed description of what our search pattern matches in the text. Beyond this resource, most importantly, mastering regular expressions requires intensive practice (cf. Mertz, 2023, Ch. 1).

Introductory text-querying process with the RegEx feature in AntConc

To perform a RegEx search, check/activate the Regex feature in AntConc (which will automatically deactivate AntConc's Word-based search feature). Users also need to be aware of what characters are activated in the token definition settings when they first load the corpus into AntConc. In the default setting of AntConc token definition, only the (upper- and lower-case) alphabetic letters are considered as tokens/characters. RegEx search for other characters will not produce any output.

One RegEx pattern that the Wildcard method does not allow is controlling the exact minimum and maximum number of characters to be searched for (i.e., the specified range quantifier (Friedl, 2006, p. 32, Table 1-3)). The RegEx pattern `\b[a-z]{5,8}\b` means that we search for a word (indicated by the word boundary `\b`) containing any lowercase letters from *a* to *z* (the `[a-z]` part) of at least five characters long up to a maximum of eight characters long (the quantifying expression `{5,8}`). Compare the output of this RegEx in Figure 17 with the output of the RegEx that changes the minimum and maximum character in Figure 18 (the search pattern is `\b[a-z]{2,3}\b`, which captures words of at least two- up to three-character long).



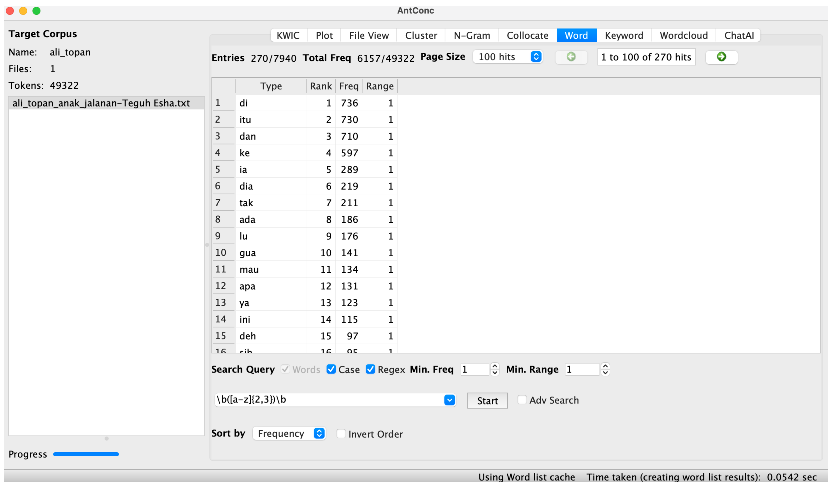
Target Corpus
Name: ali_topan
Files: 1
Tokens: 49322
ali_topan_anak_jalanan–Teguh Esha.txt

Entries 3716/7940 Total Freq 18672/49322 Page Size 100 hits 1 to 100 of 3716 hits

	Type	Rank	Freq	Range
1	dengan	1	343	1
2	tidak	2	219	1
3	mereka	3	177	1
4	nggak	4	159	1
5	dalam	5	158	1
6	sudah	6	153	1
7	tanya	7	141	1
8	rumah	8	134	1
9	orang	9	125	1
10	sekolah	10	120	1
11	untuk	10	120	1
12	kalau	12	117	1
13	berjalan	13	113	1
14	melihat	14	112	1
15	pintu	15	108	1
16	surat	16	100	1

Search Query ✓ Words ✓ Case ✓ Regex Min. Freq 1 Min. Range 1
\b([a-z]{5,8})\b
☐ Adv Search
Sort by Frequency ☐ Invert Order
Using Word list cache Time taken (creating word list results): 0.0621 sec

Figure 17. Output of the RegEx pattern `\b[a-z]{5,8}\b`


AntConc

Target Corpus
Name: ali_topan
Files: 1
Tokens: 49322
ali_topan_anak_jalanan-Teguh Esha.txt

Progress

Entries 270/7940 Total Freq 6157/49322 Page Size 100 hits 1 to 100 of 270 hits

	Type	Rank	Freq	Range
1	di	1	736	1
2	itu	2	730	1
3	dan	3	710	1
4	ke	4	597	1
5	ia	5	289	1
6	dia	6	219	1
7	tak	7	211	1
8	ada	8	186	1
9	lu	9	176	1
10	gua	10	141	1
11	mau	11	134	1
12	apa	12	131	1
13	ya	13	123	1
14	ini	14	115	1
15	deh	15	97	1
16	sih	16	95	1

Search Query ✓ Words ✓ Case ✓ Regex Min. Freq 1 Min. Range 1

```
\b([a-z]{2,3})\b
```

Sort by Frequency Invert Order

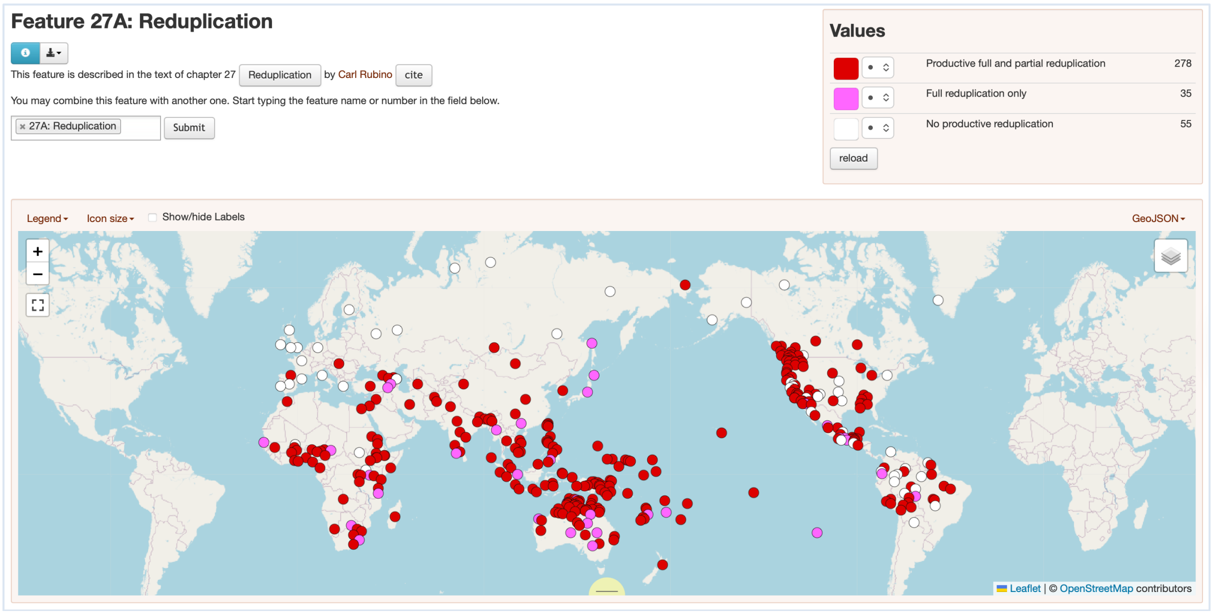
Using Word list cache Time taken (creating word list results): 0.0542 sec

Figure 18. Output of the RegEx pattern `\b[a-z]{2,3}\b`

In the reminder of this part of the paper, we will illustrate how the power and flexibility of RegEx can be leveraged to capture a relatively complex morphological phenomenon in Indonesian, namely reduplication. We will also compare how the RegEx and AntConc's Wildcard method converge and diverge in the output.

Querying Indonesian reduplication with RegEx

Indonesian is one of many languages (Rubino, 2013) that have full reduplication (Wiltshire & Marantz, 2000, p. 558; Inkelas, 2014). Amongst the many functions of the reduplication include (i) plurality of an object (e.g., *guru-guru* '[many] teachers' from the noun *guru* '(a singular) teacher') as well as (ii) derivational meaning (e.g., reduplicated adjectival adverb *cepat-cepat* 'quickly' based on the adjective *cepat* 'quick; fast') (cf. Kiyomi, 1995; Rubino, 2013; Inkelas, 2014; Moeljadi, 2023, inter alia). Figure 19 is a screenshot of the global distribution of languages having (and not having productive) reduplication processes (Rubino, 2013) in the *World Atlas of Language Structures* (WALS) Online (Dryer & Haspelmath, 2013).


Feature 27A: Reduplication

This feature is described in the text of chapter 27 Reduplication by Carl Rubino cite

You may combine this feature with another one. Start typing the feature name or number in the field below.

× 27A: Reduplication Submit

Values

🟥	Productive full and partial reduplication	278
🟪	Full reduplication only	35
⬜	No productive reduplication	55

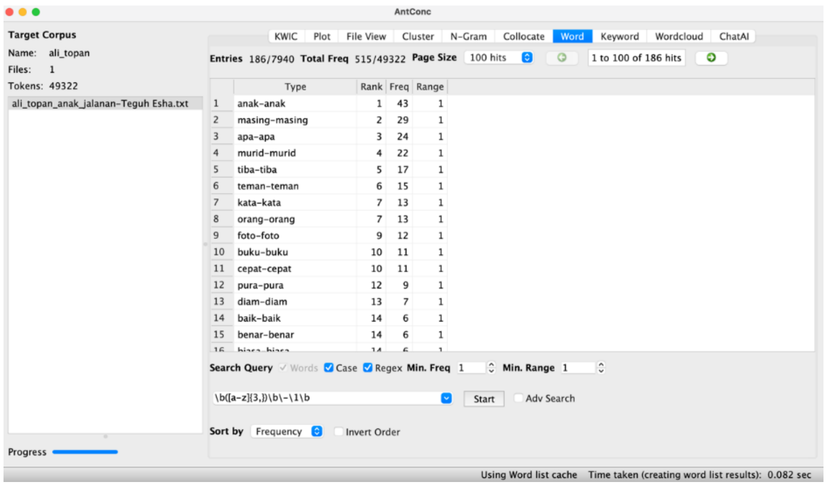
Figure 19. Global distribution of reduplication in WALS

For empirical research, it would be difficult to chart exhaustively all possible reduplications available in Indonesian, relying purely on introspective data. With the availability of large corpora of Indonesian, this reduplication phenomenon could be studied in an empirical manner (see Moeljadi, 2023, for a recent corpus-based study). RegEx search pattern practically provides a tool for retrieving reduplication more efficiently.

One crucial feature of RegEx for querying reduplication is the so-called “backreferencing” (Friedl, 2006, pp. 20–22). Backreferencing allows repeating a grouped pattern (which appears initially) in the subsequent pattern. In the context of full reduplication (e.g., *cepat-cepat*), the first element (i.e., *cepat*) that precedes the hyphen is repeated exactly after the hyphen. In practice, to repeat the first part of the full reduplication (i.e., the base) into the second slot after the hyphen (i.e., into the “reduplicant” (Rubino, 2013) or the “copy” (Wivell et al., 2024, p. 508)), we need to refer to (i.e., backreference) whatever element appears as the first part. The RegEx `\b([a-z]{3,})\-\1\b` can capture this full reduplication. This RegEx can be decoded as follows:

a. the hyphen (`-`) in a full reduplication is captured by the pattern `\-` (the backslash “\” is called an ‘escape character’ to capture a metacharacter in its literal form, that is, the backslash reverts the metacharacter into its original, literal identity)
b. the first element that we aim to refer back/repeat in the second, reduplicant element is captured by `[a-z]{3,}` (meaning, any letters from *a* to *z*, which are at least three characters long up to infinity). This pattern is then put inside parentheses (...) (hence, `([a-z]{3,})`). Parentheses in RegEx indicates a grouping (i.e., a unit of [sub]expression/[sub]pattern). Only a grouped pattern in the parentheses that can be referred and repeated in the subsequent pattern using the backreference method.
c. the backreference capturing exactly the grouped pattern of the first element appears after the hyphen, namely the pattern `\1` (this pattern means ‘repeat the first [hence, the number 1] grouped element [i.e., `([a-z]{3,})`] appearing in the preceding RegEx pattern’).
d. the `\b` part has been introduced before as a marker of word boundary

The output of this RegEx (in the Word menu of AntConc) is shown in Figure 20 below.


AntConc

Target Corpus
Name: ali_topan
Files: 1
Tokens: 49322
ali_topan_anak_jalanan-Teguh Esha.txt

Entries 186/7940 Total Freq 515/49322 Page Size 100 hits 1 to 100 of 186 hits

	Type	Rank	Freq	Range
1	anak-anak	1	43	1
2	masing-masing	2	29	1
3	apa-apa	3	24	1
4	murid-murid	4	22	1
5	tiba-tiba	5	17	1
6	teman-teman	6	15	1
7	kata-kata	7	13	1
8	orang-orang	7	13	1
9	foto-foto	9	12	1
10	buku-buku	10	11	1
11	cepat-cepat	10	11	1
12	pura-pura	12	9	1
13	diam-diam	13	7	1
14	baik-baik	14	6	1
15	benar-benar	14	6	1

Search Query ✓ Words ✓ Case ✓ Regex Min. Freq 1 Min. Range 1

```
\b([a-z]{3,})\b\-\1\b
```

Start Adv Search

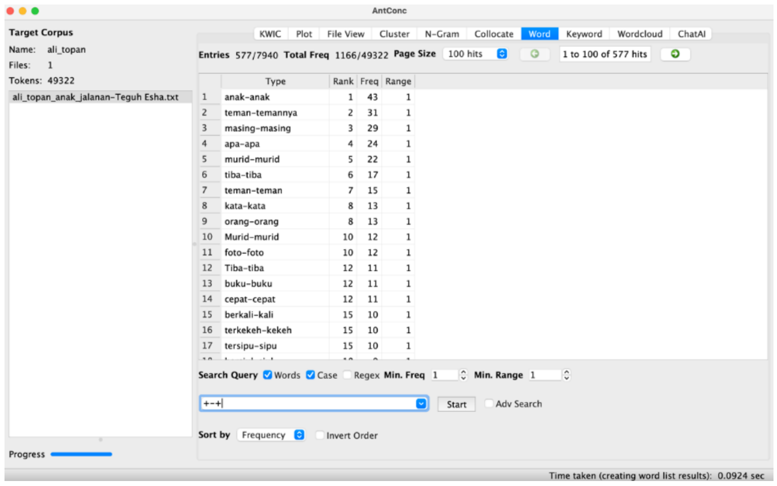
Sort by Frequency Invert Order

Progress

Using Word list cache Time taken (creating word list results): 0.082 sec

Figure 20. Output of the backreferencing technique for capturing full reduplication

It should be mentioned that we could query reduplication using AntConc’s default Wildcard method, especially combining the + symbol with the hyphen (-) (see the output in Figure 21).



AntConc

Target Corpus
Name: ali_topan
Files: 1
Tokens: 49322
ali_topan_anak_jalanan-Teguh Esha.txt

Entries 577/7940 Total Freq 1166/49322 Page Size 100 hits 1 to 100 of 577 hits

	Type	Rank	Freq	Range
1	anak-anak	1	43	1
2	teman-temannya	2	31	1
3	masing-masing	3	29	1
4	apa-apa	4	24	1
5	murid-murid	5	22	1
6	tiba-tiba	6	17	1
7	teman-teman	7	15	1
8	kata-kata	8	13	1
9	orang-orang	8	13	1
10	Murid-murid	10	12	1
11	foto-foto	10	12	1
12	Tiba-tiba	12	11	1
13	buku-buku	12	11	1
14	cepat-cepat	12	11	1
15	berkali-kali	15	10	1
16	terkekeh-kekeh	15	10	1
17	tersipu-sipu	15	10	1

Search Query ✓ Words ✓ Case Regex Min. Freq 1 Min. Range 1

```
+-+
```

Sort by Frequency Invert Order

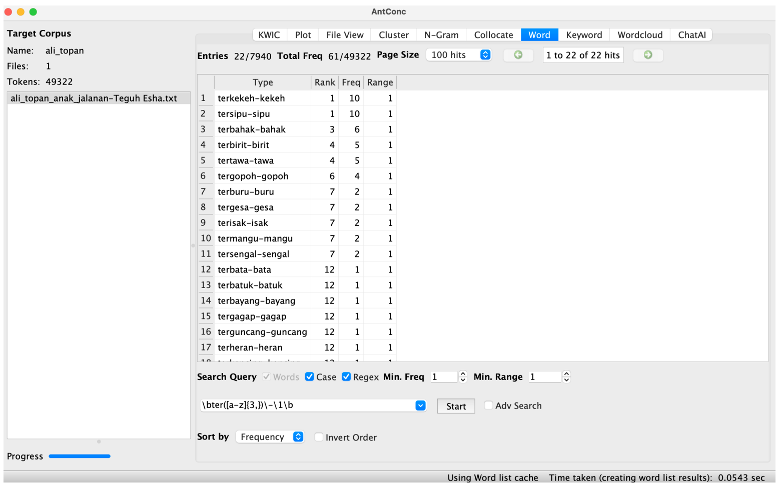
Time taken (creating word list results): 0.0924 sec

Figure 21. Querying full reduplication using AntConc's Wildcard feature

In contrast to the RegEx method, the output of the Wildcard search is slightly less specific. For example, the Wildcard search produces the inflected full reduplication *teman-temannya* 'h(is/er) friends' (see row number 2 in Figure 21) while the RegEx method does not. The reason for this is that the + character captures any one or more character in either side of the hyphen, regardless whether the strings in the reduplicant (to the right of the hyphen) is the exact copy of the first element or not, and vice versa. Hence, *temannya* (the reduplicant) in *teman-temannya* is not an exact repetition of *teman* (the first element); the reverse mismatch exists in the prefixed verbal reduplication *terkekeh-kekeh* 'chuckling', whereby the first element is not an exact match with the reduplicant. Indeed, the exact repetition of the first element in the reduplicant is also captured by the `+-+` pattern (see, for examples, rows number seven [7] and eight [8] in Figure 21). On the contrary, the RegEx pattern specifically defines that the reduplicant of the full reduplication must be an exact copy of the first element.

Based on the above comparison, the choice of the two search methods boils down to the specific purpose of the researcher. The Wildcard method casts wider nets for wider range of outputs/data, which can be further manually selected based on the researcher's need. The RegEx method allows more specific, targeted pattern (that could exclude unwanted output, even though further verification is still needed).

In addition to the full reduplication with identical first and second elements, Indonesian also has prefixed full reduplication whereby the first element is in its prefixed form. The form *terkekeh-kekeh* 'chuckling' is one example. Figure 22 below illustrates how we can define more targeted RegEx pattern (i.e., `\bter([a-z]{3,})\-\1\b`) to capture just reduplication whose first element is prefixed with *ter-* followed by a root element of at least three-character long (and not the full nominal reduplication such as *teman-teman* or the inflected reduplication *teman-temannya*).



AntConc

Target Corpus
Name: ali_topan
Files: 1
Tokens: 49322
ali_topan_anak_jalanan-Teguh Esha.txt

Entries 22/7940 Total Freq 61/49322 Page Size 100 hits 1 to 22 of 22 hits

	Type	Rank	Freq	Range
1	terkekeh-kekeh	1	10	1
2	tersipu-sipu	1	10	1
3	terbahak-bahak	3	6	1
4	terbirit-birit	4	5	1
5	tertawa-tawa	4	5	1
6	tergopoh-gopoh	6	4	1
7	terburu-buru	7	2	1
8	tergesa-gesa	7	2	1
9	terisak-isak	7	2	1
10	termangu-mangu	7	2	1
11	tersengal-sengal	7	2	1
12	terbata-bata	12	1	1
13	terbatuk-batuk	12	1	1
14	terbayang-bayang	12	1	1
15	tergagap-gagap	12	1	1
16	terguncang-guncang	12	1	1
17	terheran-heran	12	1	1

Search Query ✓ Words ✓ Case ✓ Regex Min. Freq 1 Min. Range 1

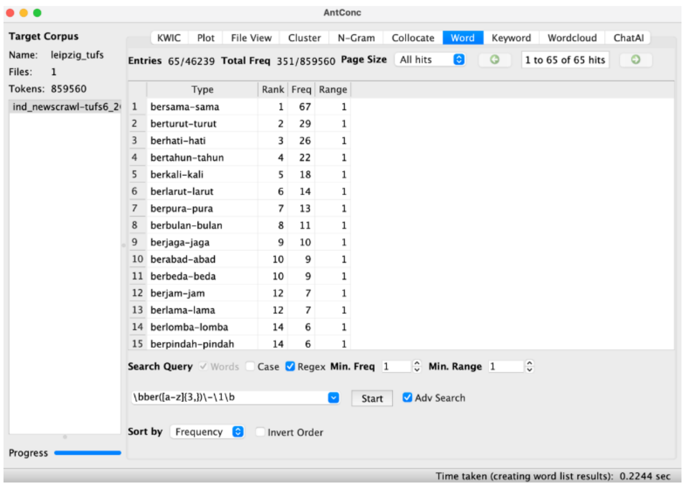
```
\bter([a-z]{3,})\-\1\b
```

Sort by Frequency Invert Order

Using Word list cache Time taken (creating word list results): 0.0543 sec

Figure 22. Output of querying prefixed first-element of full reduplication

Similar approach can also be adopted to capture prefixed reduplication for another Indonesian verbal prefix by modifying the prefix string in the RegEx. For example, to retrieve the *ber*- prefixed full reduplication (e.g., *bersama-sama* 'together', based on *sama* 'equal'), we can query the corpus using this RegEx: `\bber([a-z]{3,})\-\1\b`. The results are shown in Figure 23.


AntConc

Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

Entries 65/46239 Total Freq 351/859560 Page Size All hits 1 to 65 of 65 hits

	Type	Rank	Freq	Range
1	bersama-sama	1	67	1
2	berturut-turut	2	29	1
3	berhati-hati	3	26	1
4	bertahun-tahun	4	22	1
5	berkali-kali	5	18	1
6	berlarut-larut	6	14	1
7	berpura-pura	7	13	1
8	berbulan-bulan	8	11	1
9	berjaga-jaga	9	10	1
10	berabad-abad	10	9	1
11	berbeda-beda	10	9	1
12	berjam-jam	12	7	1
13	berlama-lama	12	7	1
14	berlomba-lomba	14	6	1
15	berpindah-pindah	14	6	1

Search Query ✓ Words ☐ Case ☑ Regex Min. Freq 1 Min. Range 1
\bber([a-z]{3,})\-\1\b Start ☑ Adv Search
Sort by Frequency ☐ Invert Order
Time taken (creating word list results): 0.2244 sec

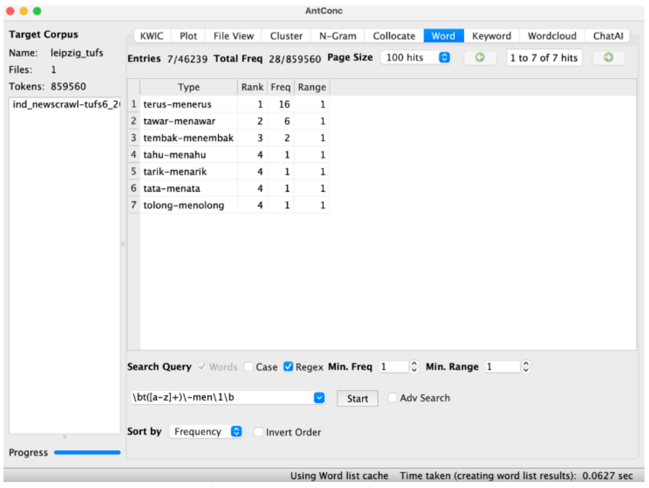
Figure 23. Output of the ber- *prefixed first-element of full reduplication*

Another subtype of reduplication in Indonesian expresses reciprocal action, such as *pukul-memukul* 'hitting each other', based on the root *pukul* 'to punch' (Kiyomi, 1995, pp. 1156–1157; Moeliono et al., 2017, p. 174; Mistica et al., 2009). Both RegEx and Wildcard searches could be used to query such a reduplication type. As will be shown shortly, RegEx allows more targeted output than the Wildcard method, which could generate noises (false positives). RegEx backreferencing is crucial for this purpose.

Take an example when we wish to find reduplication in which the reduplicant is the *meN*- prefixed form of the first-element root. To this end, we need to specify (i) the exact first-letter consonant of the first (i.e., root) element and (ii) the exact prefix form with the assimilated homorganic nasal sound (the *N* in *meN*-) according to the first letter of the root. In the case of root forms beginning with the letter *t*, the prefix *meN*- becomes *men*- (e.g., from *tulis* 'write' into *me**n**ulis* 'to write'). To retrieve this reciprocal reduplication based on the root starting with *t*, this RegEx could be used: `\bt([a-z]+)\-men\1\b`. This pattern means:

a. find a reduplicated word (as indicated by the hyphen `\-`) with the first element (indicated by `t([a-z]+)`) being a potential root form beginning with the consonant *t*
b. group the string (i.e., put it inside parentheses) after the first letter of the root so that this grouped string can be referred and repeated in the reduplicant after the prefix *men*- attached to the root beginning with *t*

The output of this query in the ILCC (not the *Ali Topan* corpus) is shown in Figure 24 (the same RegEx pattern can be used in the KWIC menu to produce the reduplicated forms in their usage contexts).


AntConc

Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

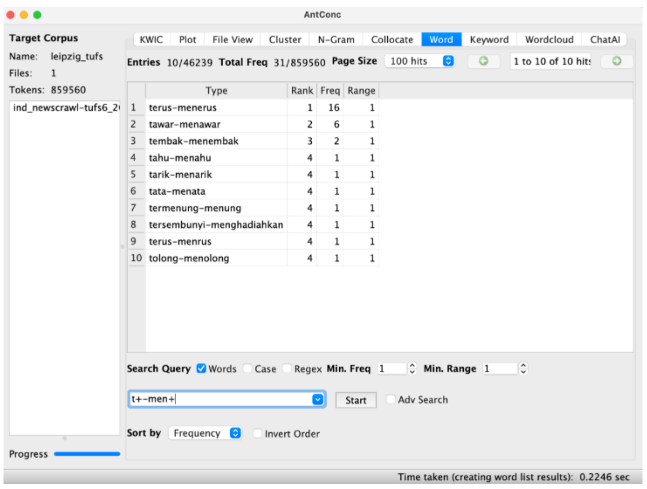
Entries 7/46239 Total Freq 28/859560 Page Size 100 hits 1 to 7 of 7 hits

	Type	Rank	Freq	Range
1	terus-menerus	1	16	1
2	tawar-menawar	2	6	1
3	tembak-menembak	3	2	1
4	tahu-menahu	4	1	1
5	tarik-menarik	4	1	1
6	tata-menata	4	1	1
7	tolong-menolong	4	1	1

Search Query ✓ Words ☐ Case ☑ Regex Min. Freq 1 Min. Range 1
\bt([a-z]+)\-men\1\b Start ☐ Adv Search
Sort by Frequency ☐ Invert Order
Using Word list cache Time taken (creating word list results): 0.0627 sec

Figure 24. Output of RegEx query for reciprocal reduplication

Now compare the output of the Wildcard search for retrieving the same phenomenon in Figure 25 (still via the Word menu of AntConc). The Wildcard search pattern could be `t+-men+`. Remember to deactivate the RegEx option and activate the Word option to run the Wildcard search.



AntConc

Target Corpus
Name: leipzig_tufs
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2

KWIC | Plot | File View | Cluster | N-Gram | Collocate | Word | Keyword | Wordcloud | ChatAI

Entries 10/46239 Total Freq 31/859560 Page Size 100 hits 1 to 10 of 10 hits

	Type	Rank	Freq	Range
1	terus-menerus	1	16	1
2	tawar-menawar	2	6	1
3	tembak-menembak	3	2	1
4	tahu-menahu	4	1	1
5	tarik-menarik	4	1	1
6	tata-menata	4	1	1
7	termenung-menung	4	1	1
8	tersembunyi-menghadiahkan	4	1	1
9	terus-menrus	4	1	1
10	tolong-menolong	4	1	1

Search Query [✓ Words] Case Regex Min. Freq 1 Min. Range 1

t+-men+ Start Adv Search

Sort by Frequency Invert Order

Progress

Time taken (creating word list results): 0.2246 sec

Figure 25. Wildcard query for reciprocal reduplication

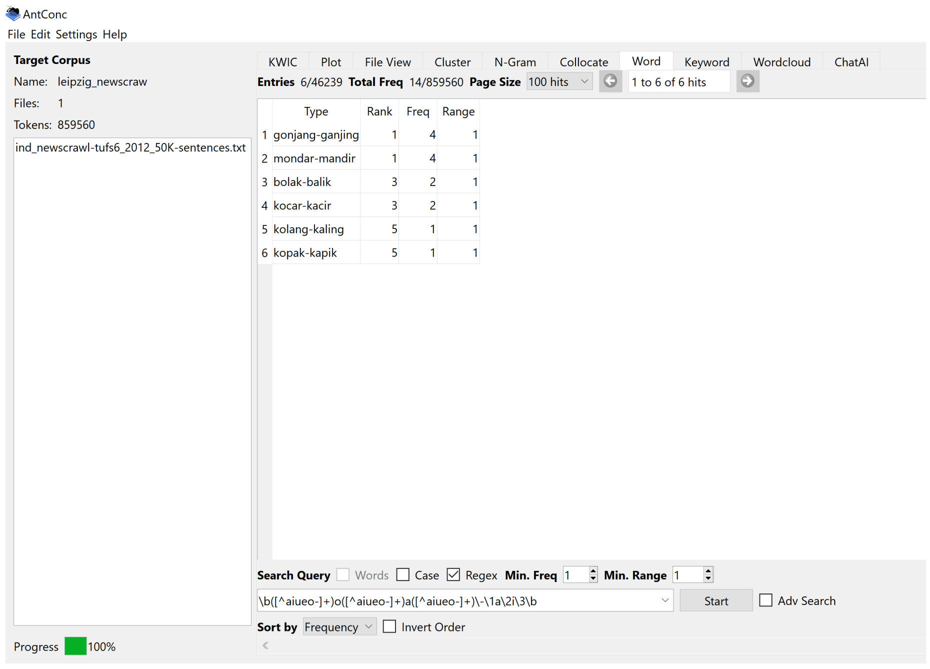
One clear false positive from Figure 25 is the form ***tersembunyi***-*men*_**ghadiahkan**_, whereby the + character captures the boldfaced strings of the form, which is not the targeted reduplication. The other false positive is the form ***termenung***-*me*_**nung**_ 'staring blankly; lost in thought'; it is a verbal reduplication (based on the *ter-* prefixed stative verb *termenung* 'pensive') but is not of the reciprocal type as in ***tolong***-*me*_**nolong**_ 'helping each other' (based on the verb root *tolong* 'to help') or in ***tarik***-*me*_**narik**_ 'pulling each other' (based on *tarik* 'to pull'). With just ten entries in the output of the Wildcard search, it is not too difficult to manually discard irrelevant types (such as rows number eight [8] and seven [7]). The RegEx method provides more precise results, though qualitative inspection of the output is also required to further verify if the pattern reflects reciprocal reduplication (e.g., *terus-menerus* 'continuously', while representing a correct morphological pattern for the reduplication, is semantically an adverb of frequency rather than indicating reciprocity).

The final example to be discussed is the "ablaut reduplication" (Cabrera, 2017; Wivell et al., 2024), a subtype of reduplication involving vowel alternation such as the English *tick-tock* or *zig-zag*. Indonesian has this kind of reduplication in such forms as (i) ***bolak***-*b*_**alik**_ 'back and forth' (based on *balik* 'return; revert') or (ii) ***asal***-_**usul**_ 'origin' (based on *asal* 'origin'). In this paper, the RegEx query to retrieve forms of the first (i) example (with the C[onsonant]*o*C*a*C-C*a*C*i*C pattern) will be demonstrated.

The pattern of this first example type maintains the exact same consonants between the first and second elements of the reduplication, with changes only in the vowels (i.e., ***bolak***-*b*_**alik**_, or ***gonjang***-*g*_**anjing**_ 'rumour', based on *gonjang* 'to shake heavily'). Since the consonants are repeated, backreference can be useful. We then specify the vowels in the RegEx, because this vowel alternation (*o a -a i*) is amongst the common type of ablaut reduplication in Indonesian (Moeljadi, 2023; cf. Sariah, 2018). Here is the proposed RegEx: `\b([^aiueo-]+)o([^aiueo-]+)a([^aiueo-]+)\-\1a\2i\3\b`. The explanation is as follows:

a. the caret "^" in `[^aiueo-]` means to negate the characters following it. In that case, `[^aiueo-]` means any character that is not hypen and not vowel (aiueo) (hence, consonant). This subexpression for consonant is (i) quantified to occur at least one or more time via the + quantifier and (ii) grouped with parentheses (…) to be accessible for backreference
b. the specific vowels in the first element (i.e., *o* and *a*) are spelled out explicitly, similar to the specific vowels in the second element (i.e., *a* and *i*)
c. the `\1`, `\2`, and `\3` respectively are backreference indexes for the grouped/bracketed consonants in the first element. So the `\3` subexpression refers to the third grouped/bracketed consonant pattern in the first element (i.e., the consonant pattern before the hyphen \-)

The output of this RegEx is shown in Figure 26 based on the ILCC corpus file.



AntConc
File Edit Settings Help

Target Corpus
Name: leipzig_newscraw
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2012_50K-sentences.txt

Entries 6/46239 Total Freq 14/859560 Page Size 100 hits 1 to 6 of 6 hits

	Type	Rank	Freq	Range
1	gonjang-ganjing	1	4	1
2	mondar-mandir	1	4	1
3	bolak-balik	3	2	1
4	kocar-kacir	3	2	1
5	kolang-kaling	5	1	1
6	kopak-kapik	5	1	1

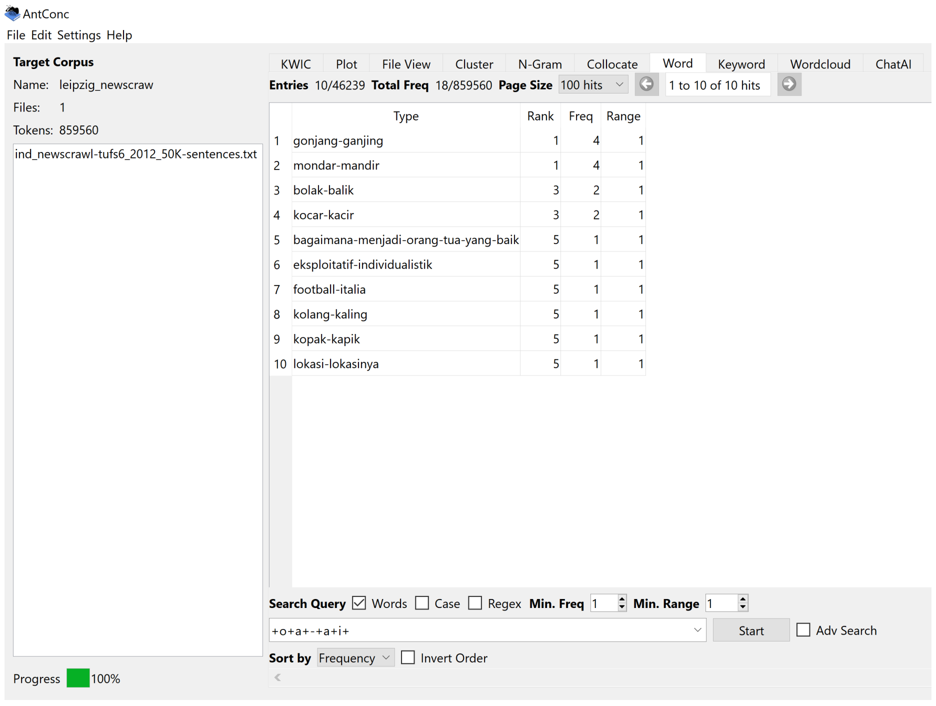
Search Query ☐ Words ☐ Case ☑ Regex Min. Freq 1 Min. Range 1

```
\b([^aiueo-]+)o([^aiueo-]+)a([^aiueo-]+)\-\1a\2i\3\b
```

Sort by Frequency ☐ Invert Order
Progress 100%

Figure 26. Output of the RegEx query for the ablaut reduplication

The Wildcard method could also be used to query ablaut reduplication in Indonesian. However, this method can introduce noises (false positives). See Figure 27 for the output of the Wildcard `+o+a+-+a+i+`.



AntConc
File Edit Settings Help

Target Corpus
Name: leipzig_newscraw
Files: 1
Tokens: 859560
ind_newscrawl-tufs6_2012_50K-sentences.txt

Entries 10/46239 Total Freq 18/859560 Page Size 100 hits 1 to 10 of 10 hits

	Type	Rank	Freq	Range
1	gonjang-ganjing	1	4	1
2	mondar-mandir	1	4	1
3	bolak-balik	3	2	1
4	kocar-kacir	3	2	1
5	bagaimana-menjadi-orang-tua-yang-baik	5	1	1
6	eksploitatif-individualistik	5	1	1
7	football-italia	5	1	1
8	kolang-kaling	5	1	1
9	kopak-kapik	5	1	1
10	lokasi-lokasinya	5	1	1

Search Query ☑ Words ☐ Case ☐ Regex Min. Freq 1 Min. Range 1

```
+o+a+-+a+i+
```

Sort by Frequency ☐ Invert Order
Progress 100%

Figure 27. Output of the Wildcard query for the supposedly ablaut reduplication

One limitation of the Wildcard method is its lack of character class negation. RegEx allows us to negate a certain character type. Meanwhile, Wildcard does not possess mechanism to capture a given character class (e.g., only vowels) or negate certain characters (e.g., negating vowels and hyphen to specifically capture the consonants). The + captures any one or more character (both vowels and consonants and the hyphen) defined in the token definition when we loaded the corpus, while the targeted ablaut reduplication requires exact copy of the consonants in the two elements. That is why we have the results such as *lokasi-lokasinya* 'the locations' or unrelated hyphenated elements such as *football-italia* or even a hyphenated clause *bagaimana-menjadi-orang-tua-yang-baik*. It is evident that RegEx is more precise than the Wildcard method.

4. CONCLUSION

Corpus Linguistics rely on computationally querying text-patterns in a large collection of digital texts as a way to collect dataset in answering one's linguistic research questions. AntConc is one of the established, open-source, installation-based tools for conducting corpus linguistic research. This paper has described two main flavours of querying texts in AntConc, namely the Wildcards and Regular Expressions (RegEx). These two are available in AntConc's various corpus linguistic tools, such as Key Word in Context, Word List, Collocate, and Word Cluster. We hope to have demonstrated that, while the Wildcard method is useful for relatively flexible searches, RegEx, in particular, is more powerful and sophisticated for more precise yet flexible queries of complex pattern. The power of RegEx has been illustrated with querying different types of reduplication, a complex morphological phenomenon in Indonesian. In sum, mastering different methods of querying text-pattern equips us with wider range of searching-options with different levels of complexities.

REFERENCES

Anthony, L. (2022). What can corpus software do? In A. O'Keeffe & M. J. McCarthy (Eds.), *The Routledge Handbook of Corpus Linguistics* (2nd ed., pp. 103–125). Routledge. https://doi.org/10.4324/9780367076399-9

Anthony, L. (2024). *AntConc* (Version 4.3.1) [Computer software]. Waseda University. https://www.laurenceanthony.net/software/AntConc

Arka, I. W. (2010, August 2). *Dynamic and stative passives in Indonesian & their computational implementation*. MALINDO Workshop, Jakarta.

Bothma, T. J. D. (2017). Lexicography and information science. In P. A. Fuertes-Olivera (Ed.), *The Routledge Handbook of Lexicography* (1st ed., pp. 197–216). Routledge. https://doi.org/10.4324/9781315104942-14

Cabrera, J. C. M. (2017). Continuity and change: On the iconicity of Ablaut Reduplication (AR). In A. Zirker, M. Bauer, O. Fischer, & C. Ljungberg (Eds.), *Dimensions of Iconicity* (pp. 63–84). John Benjamins Publishing Company. https://doi.org/10.1075/ill.15.04mor

Daintith, J., & Wright, E. (2008). Kleene star. In *A Dictionary of Computing*. Oxford University Press. https://www.oxfordreference.com/display/10.1093/acref/9780199234004.001.0001/acref-9780199234004-e-2804

Dryer, M. S., & Haspelmath, M. (Eds.). (2013). *WALS Online (v2020.4)*. Zenodo. https://doi.org/10.5281/zenodo.13950591

Esha, T. (1977). *Ali Topan Anak Jalanan*. Cypress.

Friedl, J. E. F. (2006). *Mastering regular expressions* (3rd ed). O'Reilly.

Goldhahn, D., Eckart, T., & Quasthoff, U. (2012). Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages. In N. Calzolari, K. Choukri, T. Declerck, M. U. Doğan, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)* (pp. 759–765). European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2012/pdf/327_Paper.pdf

Inkelas, S. (2014). Non-Concatenative Derivation: Reduplication. In R. Lieber & P. Štekauer (Eds.), *The Oxford Handbook of Derivational Morphology*. Oxford University Press. https://doi.org/10.1093/oxfordhb/9780199641642.013.0011

Kiyomi, S. (1995). A new approach to reduplication: A semantic study of noun and verb reduplication in the Malayo-Polynesian languages. *Linguistics*, *33*, 1145–1167. https://doi.org/10.1515/ling.1995.33.6.1145

Lewenstein, M., Ian Munro, J., Raman, V., & Thankachan, S. V. (2014). Less space: Indexing for queries with wildcards. *Theoretical Computer Science*, *557*, 120–127. https://doi.org/10.1016/j.tcs.2014.09.003

Mertz, D. Q. (2023). *Regular expression puzzles and AI coding assistants: 24 puzzles solved by the author, with and without assistance from Copilot, ChatGPT and more*. Manning.

Mistica, M., Arka, I. W., Baldwin, T., & Andrews, A. (2009). Double Double, Morphology and Trouble: Looking into Reduplication in Indonesian. *Proceedings of the Australasian Language Technology Association Workshop 2009*, 44–52. https://www.aclweb.org/anthology/U09-1007

Moeliono, A. M., Lapoliwa, H., Alwi, H., Tjatur, S. S., Sasangka, W., & Sugiyono, S. (2017). *Tata bahasa baku bahasa Indonesia* (Edisi Keempat). Badan Pengembangan dan Pembinaan Bahasa, Kementrian Pendidikan dan Kebudayaan. http://repositori.kemdikbud.go.id/16351/

Moeljadi, D. (2023). *Reduplikasi sebagian, salin suara, dan trilingga dalam bahasa Indonesia* [Book Chapter Manuscript].

Nomoto, H., Akasegawa, S., & Shiohara, A. (2018). Reclassification of the Leipzig Corpora Collection for Malay and Indonesian. *NUSA*, *65*, 47–66.

Paquot, M., & Gries, S. Th. (Eds.). (2020). *A Practical Handbook of Corpus Linguistics*. Springer International Publishing. https://doi.org/10.1007/978-3-030-46216-1

Quasthoff, U., & Goldhahn, D. (2013). *Indonesian corpora* (No. 7; Technical Report Series on Corpus Building). Abteilung Automatische Sprachverarbeitung, Institut für Informatik, Universität Leipzig.

Rajeg, G. P. W. (Director). (2020, May 24). *Tutorial AntConc* [Video recording]. Universitas Udayana. https://doi.org/10.6084/m9.figshare.12950687

Rubino, C. (2013). Reduplication (v2020.4). In M. S. Dryer & M. Haspelmath (Eds.), *The World Atlas of Language Structures Online*. Zenodo. https://doi.org/10.5281/zenodo.13950591

Sariah. (2018). Dwilingga salin suara dalam bahasa Indonesia. *Widyaparwa*, *46*(2), 99–262.

Stefanowitsch, A. (2020). *Corpus linguistics: A guide to the methodology*. Language Science Press.

Weisser, M. (2016). Regular expressions. In *Practical corpus linguistics: An introduction to corpus-based language analysis* (First edition, pp. 82–100). Wiley-Blackwell.

Wiltshire, C., & Marantz, A. (2000). Reduplication. In G. Booij, C. Lehmann, J. Mugdan, W. Kesselheim, & S. Skopeteas (Eds.), *Morphologie* (pp. 557–567). Walter de Gruyter. https://doi.org/10.1515/9783110111286.1.8.557

Wivell, G., Miatto, V., Karakaş, A., Kostyszyn, K., & Repetti, L. (2024). All about ablaut: A typology of ablaut reduplicative structures. *Linguistic Typology*, *28*(3), 505–536. https://doi.org/10.1515/lingty-2023-0018