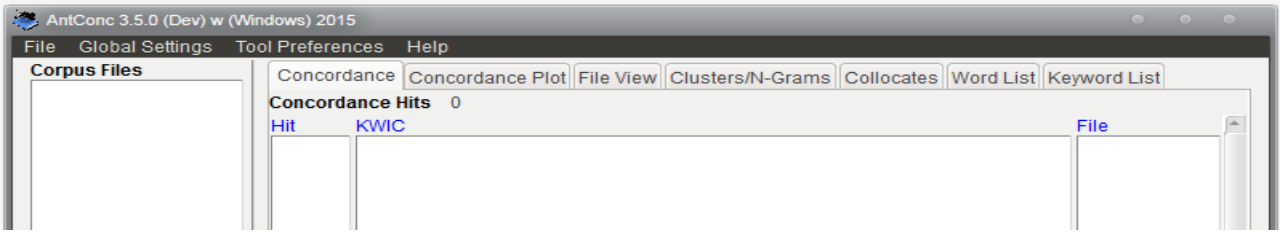
Introducing AntConc

Sandra Kies
Associate Professor
Benedictine University

Daniel Kies
Professor Emeritus
College of DuPage

AntConc has seven tools spread across the tabs on the top of the main screen. See Figure 1, below:

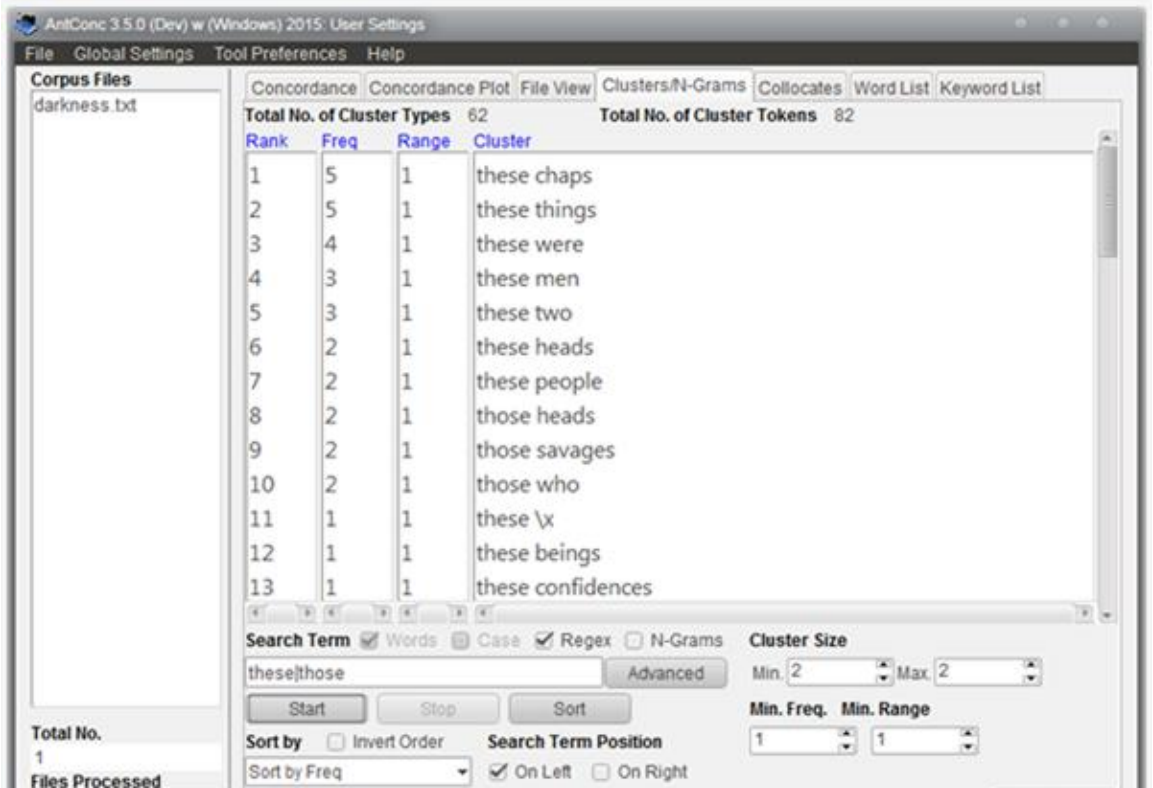
Figure 1: the AntConc toolbar



1. **Concordance:** displays search text in context
2. **Concordance Plot:** displays the distribution of the search text through the corpus as an image
3. **File View:** shows the search term within the corpus as a whole
4. **Clusters/N-Grams:** displays which words will cluster with the search term within *n* words of the search term

   For example, below we see the words that cluster with the definite pronouns *these* or *those* within one word to the right (Figure 2).

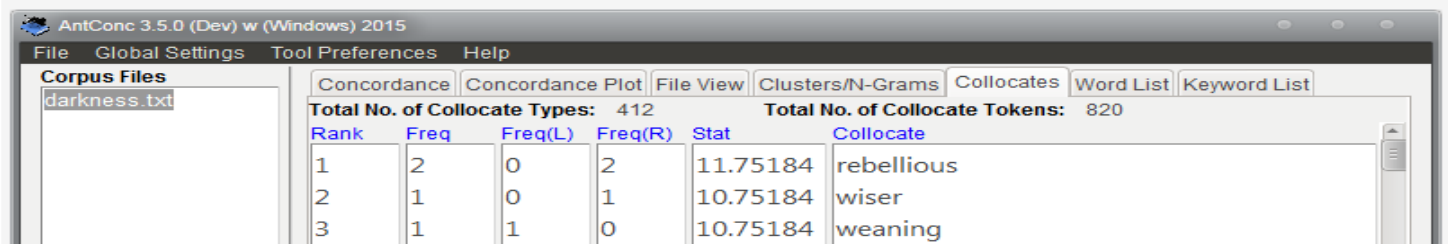
Figure 2: the Clusters/N-gram toolbar



Frequencies

AntConc can display frequencies and a few statistics as well: consider this screen capture showing the first result of collocates for the words *these* and *those* in *Heart of Darkness.* See Figure 3, below.

Figure 3: Frequencies and statistics in AntConc

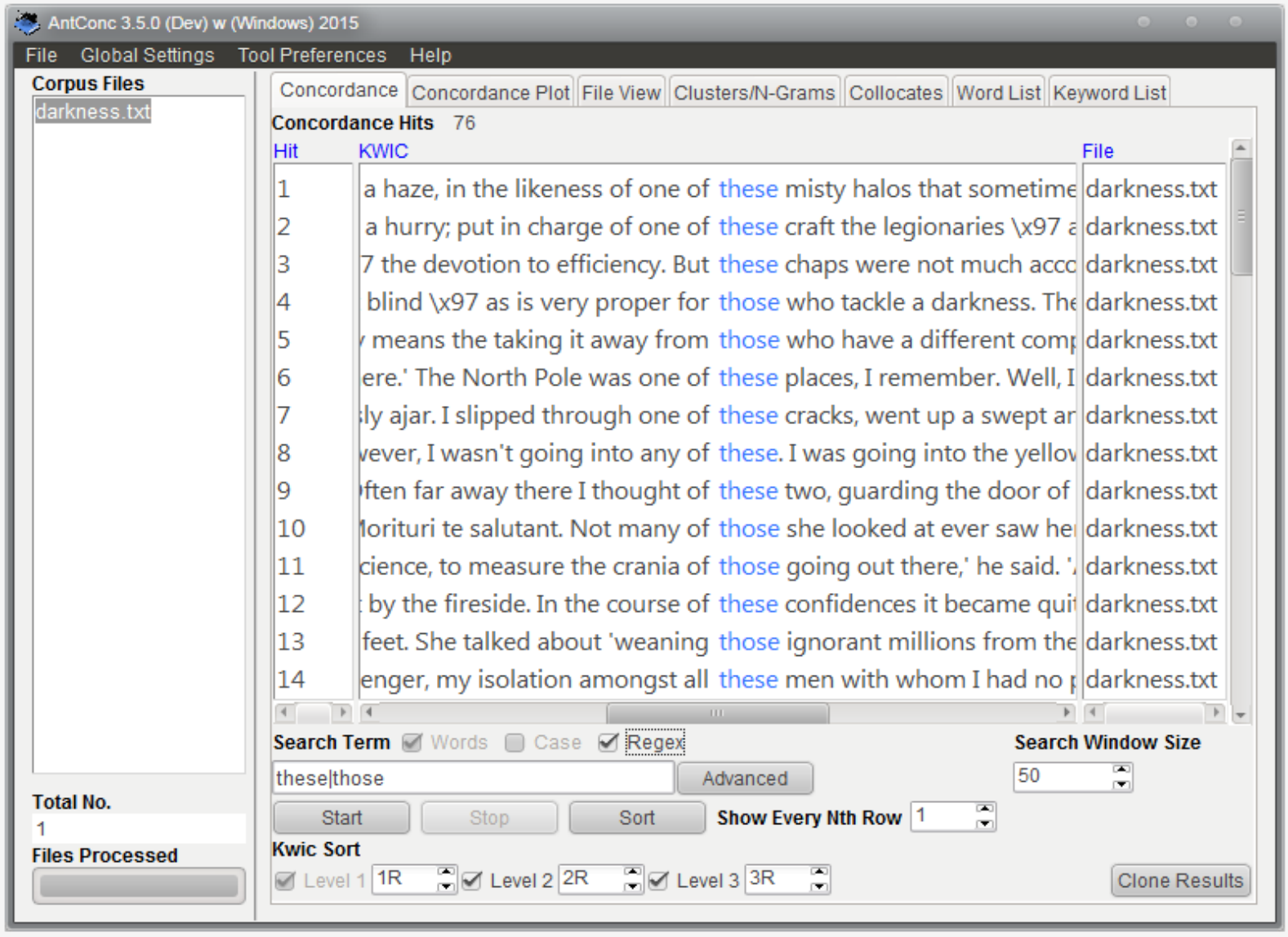


Notice above that there are 410 different types of collocations (but some repeat, which is why we see 820 **tokens** of those 412 word **types** altogether). Next, note that *rebellious* collocates twice with the search terms and that the statistical significance of that is 11.75+. That is an impressively strong collocation, suggesting that *these* or *those rebellious NOUNs* marks a significantly meaningful concept for Conrad.

KWIC searches

KWIC stands for **k**ey**w**ord-**i**n-**c**ontext. This feature is the major tool of a concordancer. The output of KWIC searches highlights the search term with four or five words to the left and the right. All these parameters are variable: for example, the search term might be a phrase or a collection of words, as in the example screen capture below (Figure 4), where we see a search through *Heart of Darkness* for the definite pronouns *these* or *those*.

Figure 4: AntConc's KWIC results window



Visit **http://www.laurenceanthony.net/software/antconc/** and review the video and text guides to using AntConc.

Exploration of AntConc

Go to **https://rhetory.com/antconc** for links to files we will use in our demonstration of AntConc.

Useful terms and concepts

A few important terms to know about corpus linguistics and its tools:

1. **types**

   a word count of all the distinct words in a corpus, not including any repetitions of those words

   For example, in the sentence *The quick fox jumped over the sleeping brown dog* we would say there are eight word **types** (remember that the count of word **types** ignores repetitions).

2. **tokens**

   a word count, including repetitions

   For example, in the example sentence above, we would say that there are nine **tokens**.

   Why **type/token** counts matter: if a researcher wants to study the vocabulary of a corpus or a writer, then the **type/token** ratio reveals the size of the vocabulary relative to the size of the corpus (all the words, including repetitions). Studies of vocabulary development (language development), readability, and syntactic maturity also lean on the researcher's findings of **type/token** ratios.

   A **type/token** ratio measures the lexical diversity in a corpus. A ration of 1:1 would mean that there is no repetition of vocabulary in the corpus at all. Thus, someone with a large active vocabulary will have a smaller **type/token** ratio than someone with a smaller active vocabulary. Shakespeare's **type/token** ratio will be lower than a child's for a similarly sized corpus.

3. **lemma**

   a list of all the words families, sorted by the root word

   For example, the **lemma** of the verb *see* would include *see, saw, sees, seeing,* and *seen*. The lemma for *rebel* would include *rebel* (noun, verb, and adjective), *rebels* (plural noun), *rebellion*, *rebellions*, *rebellious*, *rebelliously*, *rebels* (3rd person singular verb), *rebelled*, *rebeldom*, *rebelism*, *rebellant*, and *rebelliousness*.

   Why **lemmas** matter: as analytic tools, **lemmas** allow researchers to study vocabulary development, syntactic maturity again, and origins of vocabulary in the corpus. If we did a **lemma** study of the spontaneous speech of Donald Trump, we would discover that he uses Anglo-Saxon vocabulary far more often than Latinate (Romance) vocabulary. Thus, he is more likely to say *big* or *huge* than he will say *substantial*, or he will more likely say *good* than *commendable*.

4. **stop list**

   a list of words to be ignored in a simple word list or **keyword** search

   Why **stop lists** matter: when we want to see the most significant words in a corpus, either in the overall word list or in the **keyword** list (discussed below), we don't need to see the thousand or so most commonly used words. Those words comprise articles, prepositions, pronouns, conjunctions, etc., and they distract us from the content words that truly "tell the story" of that corpus.

   For example, the first five words in the **stop list** (in alphabetical order) for this AntConc demonstration are *a, about, after, again, against*.

5. **keywords**

   a list of words that occur more often in the study corpus than in a corpus that the researcher uses as a norm.

   Why **keywords** matter: **keywords** capture the "about-ness" of a corpus, the sense of what that corpus is about. Such searches often show surprising ideas subtly embedded in the corpus. Notice that among the top **keywords** in *Heart of Darkness* are *see, know, look, hear, eye*, and *day*. As we noted earlier, since English has a "super metaphor" in which verbs of perception (*see*, *look*, and *hear*) or words related to light (*eye* and *day*) are semantically related to knowledge (*know*), one could argue that *Heart of Darkness* is a story about the quest for knowledge.

6. **regex**

shortened from "regular expression" or "regular expression engine"

A regular expression engine is a bit of software based on set-theory that is built into a larger piece of software, such as a concordancer.

Why **regex** matters: regular expression engines allow researchers to create pattern-matching formulas, formulas that (in essence) combine many separate searches (sometimes thousands) into one search.

A **regex** pattern uses a set of metacharacters (called operators) which have specific functions in the program. Some of the more common operators follow.

6a. **asterisk (the * character)**

matches the preceding character zero or more times

The pattern **ab*c** matches *ac, abc, abbc, abbbc,* etc.

The **asterisk** operator allows us to compare for small variations in texts, *doin* or *doin'* (as dialectal spellings of *doing*) for example.

6b. **question mark**

matches the preceding character zero or one time

The pattern **ou?r** will include results like *color, colour, honor,* and *honour,* as well as *your* and *our*.

The **question mark** operator allows us to compare for small variations in texts too, such as *honor* (AmE) or *honour* (BE).

6c. **plus sign (the + character)**

matches the preceding character one or more times

The pattern **ye+ah** will match *yeah, yeeah,* and *yeeeeeeeeeeeeeeeeeeeah*.

The **plus sign** allows for searches of unusual repetitions in a corpus.

6d. **dot**

matches any single character, including an empty space

The pattern **dark.** will match *dark, darkness, dark-green,* etc.

The **dot** operator allows us to combine and search for alternatives simultaneously. For example, if we wish to search for the use of the negative prefixes *il-*, *im-*, *in-*, and *ir-* (as in *illegal*, *impossible*, *intractable*, and *irresponsible*) simultaneously, we could create a pattern such as **SPACEi.[bpdtlmnr].** for the task. See below for the use of **square brackets** and the meaning of **SPACE**.

6e. **pipe (the | character)**

the "or" operator

The pattern **these|those** will match *these desks, those desks*, etc.

The **pipe** operator allows us simultaneously search for several terms. For example, **this|that|these|those** will find all deictic pronouns in a corpus in one search.

6f. **square brackets (the [ ] characters)**

marks a set of alternate characters

The pattern **[lw]ater** matches *later* and *water*. The pattern **SPACE[ptkbdg].** will match all the words starting with those consonants (the stop/plosive sounds in the English sound inventory), such as *tide, kite, break, dark, gleam*. Square brackets are limited to individual characters. Compare **parentheses** below.

(**SPACE** simply means an empty space in the search string, i.e., hitting the space bar on the keyboard once when the cursor is in the search box.)

Since square brackets allow us to search for alternates, we could create a pattern like **SPACE i[lmnr][a-z]+** to find the set of words beginning with negative prefixes starting with the letter *i*, such as *illegal*, *immaterial*, *incomplete*, *irresponsible*, etc. (See the **hyphen** operator below for use of **[a-z]**.)

6g. **parentheses**

matches a set of alternative characters, words, and phrases

The pattern **SPACE(the|this|that|these|those)** matches any use of those definite articles and pronouns all in one search.

Often the **parentheses** operator is assumed, therefore unnecessay, as in 6e above. However, we may need the overt use of parentheses in a pattern to isolate and emphasize a particular alternation we wish to study. For example, **SPACE(a|an) [aeiou][a-z]+** will allow us to explore the use of the spelling variant of the

indefinite article.

6h. **NOT operator (the ^ character)**

matches characters, words, and phrases we do not want in our search

The pattern **SPACE[ptkbdg]**, mentioned in 6f above, includes words like *phone, pneumatic, that,* and *know*, which do not start with a plosive sound that we were searching for. To eliminate those *ph, pn,* and *th* clusters, we can use the **NOT operator**, as in the **SPACE[ptkbdg][^(h|n)]** pattern.

However, we must be careful: if our goal is to capture all the plosive stop sounds at the beginning of a word, then the pattern above will not find the word *ghost* too, which does contain a plosive stop at the beginning of the word. As a cautionary tale, we must remember that every tool has its limits. Our task is to remain aware of the limits. Thus, we might find that rewriting the pattern as **(p[^(h|n)]|t[^h]|k[^n]|b|d|g)** gives us the result we want.

Similarly, the pattern **SPACE(the|this|that|these|those)** will match definite articles and deictic pronouns, but it will also match *there* as well, which is not a definite article or deictic pronoun. To eliminate *there*, we could refine our regex as **SPACE(the[^r]|this|that|these|those)**.

6i. **exclamation**

matches and excludes a particular word from a search

The pattern **(no | not | never | none | barely | hardly | rarely |n't)** will match all negation in the corpus, but we can use **(!no | not | never | none | barely | hardly | rarely |n't)** to show form of negation without *no*.

(In my experience, AntConc has trouble with the **exclamation** operator. It sometimes works, sometimes doesn't. Use **exclamation** carefully. I think it's a memory allocation issue within the program: if I close the program, reopen it, and begin the **exclamation** search again it sometimes will work.)

6j. **curly brackets (the { } characters)**

matches a character or word up to a specified number of times

The pattern **s{1,2}** will find one or two instances of the *s* character, as in *see, darkness*, etc. simultaneously.

Similarly, **(had [a-z]+e[dn]){1,10}** will find many regular and irregular past participle *-ed* and *-en* verb phrases in the corpus. Note how we can change the

scope of the search by changing the numbers in the **curly brackets**.

6k. **ESCAPE operator (the \ character)**

matches regex operators and treats them as ordinary characters

In the pattern **[a-zA-Z0-9]+.com**, we can search for any .com domain name, except for one problem: dot is reserved as a regex operator. To instruct that program to treat the dot character as an ordinary character, we much use the **escape** operator, as in **[a-zA-Z0-9]+\.com**.

6l. **hyphen**

matches a range of characters

The pattern **[0-9]** matches individual numbers in a text between 0 to 9. **[0-9]+** will match any set of repeating numbers. Similarly, **[a-z]** will match all lower case characters in the search, while **[A-Z]** matches all upper case characters. **[a-zA-Z]** matches all characters regardless of case, and **[a-zA-Z]+** will match any string of letters.

Hyphen serves as another shortcut in our **regex** patterns, and we have seen it at work in 6f, 6g, 6j, and 6k before.

Operators work well in combinations, and combinations really reveal the power of **regex**. For example, notice how we have already seen combinations of operators in (h) above, where the pattern **[ptkbdg][^h]** combines the **square brackets** and the **NOT operator**. For another example, the pattern **[a-zA-Z]+\?** will find every question in the novel, as long as we remember to use the **ESCAPE operator** so that the concordancer searches for a literal question mark and does not try to use the **question mark** operator.

The pattern **heart\W+(?:\w+\W+){1,6}?darkness** will match any phrase that has both the words *heart* and *darkness* within 6 words of each other. This pattern simulates in AntConc **the NEAR operator** that exists in some other concordance programs. (Of course, we can change the search words to anything that interests us. Furthermore, we can change the size of the search string by changing **6** to any number we wish.)

If the words we are studying might occur in reverse order too, then we need a pattern to search for that possibility as well:

(?:heart\W+(?:\w+\W+){1,6}?darkness|darkness\W+(?:\w+\W+){1,6}?heart)

Trying both searches in AntConc, we discover that Conrad does use *darkness* before *heart* twice in different phrases.

If we want to find any pair of two words **NEAR** each other from a list of words, we can use the following pattern, where we change *word1*, *word2*, *word3* to any words we wish to study. The pattern is flexible, and we can add more words, as long as we separate the words with the **pipe operator**. Again, the size of the search is flexible too; just change **6** to any number you like.

(word1|word2|word3)(?:\W+\w+){1,6}?\W+(word1|word2|word3)

For example, contrast the use of two semantically negative adjectives with the words *forest* or *jungle* with the word *city*:

(dark|danger|forest|jungle)(?:\W+\w+){1,90}?\W+(dark|danger|forest|jungle)

(dark|danger| city)(?:\W+\w+){1,90}?\W+(dark|danger| city)

Notice how *city* does not occur near *dark* or *danger* at all, while *forest* or *jungle* does frequently.[1]

[1] Mathematicians have continuously developed regular expressions since the 1950s (to describe intersections of different sets), and programmers have borrowed the idea for pattern searches since the 1980s. Over the years, programmers have developed slightly different implementations of regular expression engines with slightly different syntaxes. However, all of the operators mentioned above will work in AntConc.