TACL User's Guide (command-line version)

Michael Radich

LAST UPDATED AUGUST 2022.

QUESTIONS AND SUGGESTIONS FOR IMPROVEMENT TO THIS DOCUMENT WELCOME:

michael.radich@hcts.uni-heidelberg.de

Introduction

TACL[1] is a tool for the large-scale comparative analysis of strings contained in two or more bodies of digitised text.

The full set of TACL tools, which we will introduce here, runs from the command line (see further below). By contrast, a streamlined, simplified, more user-friendly version of TACL has now been made available via the TACL GUI. Beginners who would like guidance in getting up and running with the TACL GUI can download a "starter kit" here. However, the TACL GUI only offers a limited subset of the functions of what we will here call "full-fledged" TACL, or "TACL proper".

The first section of the TACL GUI User's Manual gives a general description of the basic conceptual design and operation of TACL, and an overview of typical workflows when working with the tool for typical research problems in Chinese Buddhism. These considerations apply equally to work with the GUI or in "TACL proper". We will not repeat in this document things laid out in the TACL GUI User's Manual, and so readers are requested to read that document as well—especially the section entitled "Background".

Application of TACL to the study of Chinese Buddhist texts requires not only that we can drive TACL. We also require a clear grasp of the conceptual design for a rigorous analysis using data found by TACL, and an understanding of how the human user works through results to find evidence and construct arguments. This User's Guide only treats operation of TACL itself. It should therefore be used in tandem with the "TACL Method Guide".

"TACL proper" is programmed in Python, and the code is always up-to-date and freely available on GitHub. It was initially created for use with the texts of the Chinese Buddhist canon (and related collections), as digitised in the XML files of the Chinese Buddhist Electronic Text Association (CBETA). In principle, however, TACL can be adapted for use with any Unicode corpus, and in

[1] The name "TACL" is a "backronym" motivated mainly by a love of puns, and it doesn't really matter what it stands for. Interested users can contact us with suggestions for fun interpretations; users who prefer to have answers can think of it as meaning "Text And Corpus Lab".

collaboration with other colleagues, we have also conducted limited experiments (unpublished) in its application to corpora in Tibetan, Pali and Latin.

Installation and use

This brief discussion of installation and use is designed to give readers who are unfamiliar with the command-line environment (as I was myself when I first began working with TACL) a little supplementary information that may help them in following the instructions given in the TACL documentation and the `helps` (see below).

This brief discussion will assume familiarity with some computing terms and methods. Terms so assumed will be presented in `consolas font` to allow easy identification. Users unfamiliar with those terms and methods may need a little additional background learning to make sense of the use of those terms in this description (Googling will usually suffice).

Installation of TACL differs to some degree, depending upon the user's operating system. Instructions for installation using `pip` may be found at GitHub. On Windows (and perhaps MAC), experience has shown that `pip` often does not work so well, in which case users can follow the instructions for their respective OS here.

Once installed, TACL is run from the `command line` or `shell`. Users then can call up instructions for use, i.e. `help`, which specifies and exemplifies the correct way to format commands for the various TACL functions. For example, if the user types at the `command prompt`:

```
tacl -h
```

...this calls up a menu listing subcommands available within TACL (each achieving one of the types of functionality described above): `align, catalogue, counts, diff` etc.

If the user then types the name of one of those functions followed by `-h`, for example:

```
tacl align -h
```

...this calls up a different `help` menu specific to that subcommand. Under `usage`, a typical `help` gives, in a strict sequence, required `parameters` (in lowercase font) and `arguments` (in CAPS), and optional `parameters` and `arguments` (enclosed in [square brackets]). This is followed by text that explains the operation of the subcommand, and some explanation of each of the `parameters` and their `arguments`.

In order to make TACL execute a given subcommand, it is necessary to type in the command, with its `parameters` and their `arguments`, exactly right, and in the right sequence.

From this point on, this document will assume that the user has TACL correctly installed, and can use the `helps` to correctly input commands for the various TACL functions. We will also assume that users know how to the `pipe` results of commands to files on the hard drive.[2] We recommend

[2] Basically, piping is achieved by typing after the correctly formatted TACL command (including `parameters` and `arguments`) a right arrow, and then the `(relative) file path` and file name of the file to which the results should be piped. For example:

that users save results in .csv ("comma separated values") format, which allows for convenient further manipulation of the results in tools such as Excel (see further below).

It will also be useful for users to know (if they do not already) that `verbose output`, as specified by the `-v` argument, which can optionally be passed as part of any TACL command, instructs the code to output (to the command `line/shell window` from which the code is run) a running commentary on progress as the code is implemented.

Preparation of the corpus

As mentioned above, TACL (when used for Chinese texts) is adapted for application to the digitised Taishō, as made available by CBETA in XML format. These texts are then pre-processed in two TACL steps, called `tacl prepare` and `tacl strip`, in order for further TACL operations to be possible. This section describes those steps.

In conjunction with the TACL GUI, however, we have now also made available for download at Zenodo two corpora in which these two pre-processing steps have already been achieved: (1) A corpus comprising the Taishō and Xuzangjing/Zokuzōkyō collections, with the content structured exactly as the Taishō/CBETA editors prepared them (DOI: 10.5281/zenodo.6798320). (2) A modified "Radich Taishō corpus" (DOI: 10.5281/zenodo.6798305), in which some texts or collections have been split into smaller constituent parts by Radich, in order to reflect more accurately some aspects of the structure of the texts/collections or their history (described in this document). The instructions given in this section, then, will primarily be useful to users who wish to custom-create a corpus of their own, which differs from those two corpora.

If creating their own corpus, then, users should create, in a directory handy for use from the main directory containing TACL, a corpus containing the raw CBETA XML files of the texts they wish to work with. We then run `tacl prepare` and `tacl strip` as follows. Once more, details on how to perform these steps can be discovered by calling up the `help` pertaining to each subcommand.

First, one must run `tacl prepare` on the folder of CBETA XML files. Details of what is achieved at this stage need not concern users, unless they are interested in the workings of the code itself—in which case they can probably find out what they want to know by directly examining the code.

Next, one must run `tacl strip` on the folder of files resulting from the previous `tacl prepare` operation. This step removes all TEI/XML `markup` (`tags`) from the CBETA files, transforming them into plain `text files`. This is also the step at which TACL handles variant readings in other witnesses, by reconstructing texts corresponding to such witnesses as 宋, 元, 明, 聖, 宮 etc.

`> "tests\Zhi Qian\results 1.csv"`

which, after a correctly formatted TACL command, would look something like this:

`tacl diff "full db.db" "corpora/xml stripped" "tests\Zhi Qian\catalogue.txt" > "tests\Zhi Qian\results 1.csv"`

(Red is used here only to draw attention to part of the command, and is not part of the actual formatting.)

Generation of a database

Next, one must run `tacl ngrams` on the *corpus* produced by `tacl strip`, in order to build a *database*. The nature and contents of a TACL database are described in the "Background" section of the "TACL GUI User's Manual".

Users should be aware that if one builds a database for a large corpus (e.g. the entire CBETA corpus, or the entirety of the Taishō or the *Zokuzōkyō*), the database can be very large, and the `tacl ngrams` operation that builds the database is likely to be the single TACL operation that takes the longest time and the greatest amount of processing memory (RAM). For example, a database for the entire CBETA corpus, for 2-8-grams, is about 220 GB in size. On a souped-up computer, custom-built for use with TACL, which has 64 GB of RAM, and a 2TB SSD (which allows faster read-write times), `tacl ngrams` takes a little over 24 hours to build that database, running at around 28-30 GB of RAM.

Because database build times can be so long on some computers, we have now also made available for download, as an alternative, a full database for the "Radich" Taishō corpus mentioned above: see here. Users should note, however, that the download, even for the database in zipped form, is about 47 GB, which could mean long download times. Thus, users need to weigh up the pros and cons of download of a database, versus generating their own database—both will take some time.

When building a database, it can be especially reassuring to run a command with the abovementioned optional `-v` argument for `verbose output`; this allows the user to keep tabs on what the computer is doing, whereas otherwise, if the computer sits silent for a day or more, it can be easy to get worried than nothing is happening, or something has gone wrong.

Because this process is so memory-hungry, users should also note that `tacl ngrams` has an optional `parameter -r`, which allows the user to specify a maximum quantity of RAM to be used by the process. If one does not thus specify how much RAM the process should use, it usually results in an `out of memory error` (the process crashes, and you have to start again).

However, it is also possible to build a database for any corpus of any size. Thus, users who are certain that they want to use TACL for comparisons only within a smaller set of texts can custom-build a smaller database just for the relevant corpus. If the corpus and resulting database are small enough, this will avoid the challenges described above.

Users should also note that the `tacl ngrams` command requires users to pass `arguments` specifying the shortest and longest n-grams to be indexed. I myself generally find that a range of 2- to 8-grams strikes a good balance between ensuring that the database has most of the information we will need in it (which reduces the processing burden on post-processing steps through `tacl results`, as described below), and keeping the database itself at a more or less manageable size.

Once you have a database, you are ready to apply TACL to research problems.

Catalogues

As we describe in more detail in the "TACL GUI User's Manual", the core TACL functions are comparisons (Intersect, or Difference). For the purposes of comparison, we can select groups of texts from within our corpus. The device used to inform TACL about our selection is the *catalogue.* For TACL purposes, a *catalogue* is a `text file` (saved with the suffix .txt) which lists texts by their *identifiers,* and assigns each text (work) to a group using a *label*.

Identifiers are set properties of the *corpus.* That is to say, catalogue files must identify texts by the exact filenames used for the subdirectory of the corpus that contains the text files for each witness of the work in question. For example, the identifier of Kumārajīva's *Lotus Sūtra* 妙法蓮華經 is T0262, and the corpus contains a folder called T0262; inside that folder are .txt files for each of the witnesses 大, 宋, 元, 明, etc. Identifiers in catalogues, and in the corpus as a whole, must correspond exactly to the corpus that was originally used to build the database.

Labels are arbitrarily defined by users at the point at which they create a catalogue file. For example, if we want to compare the corpus of Zhi Qian to the corpus of Dharmarakṣa, we might add the label "ZQ" to all the identifiers of Zhi Qian works in the catalogue, and "Dhr" to all the identifiers of Dharmarakṣa works.

Each single line of the catalogue file presents one work identifier, and one label. A catalogue file may contain identifiers without labels, but in the resulting TACL test, those lines of the catalogue, and therefore those unlabeled works, will be disregarded.

For example, a catalogue file for comparison between the **Madhyamāgama* T26 and a small corpus of texts by Zhu Fonian (including the **Ekottarikāgama* T125; cf. Radich and Anālayo 2017, Radich 2017a), might look like this:

```
T0026 MA

T0001 ZFn
T0125 ZFn
T0212 ZFn
T0309 ZFn
T0656 ZFn
T1428 ZFn
T1464 ZFn
```

The user must then save this file in plain text format (.txt), and wherever the syntax of a TACL command requires a catalogue, input the name of the catalogue file as an `argument` to the `parameter` `-c` (for *catalogue*).

This aspect of TACL usage raises the problem of how to create catalogue files for very large corpora. For example, one might wish to compare a single text against the rest of the Taishō, but nobody wants to sit and type up a list of identifiers and labels for the entire Taishō. For this purpose, TACL includes a subcommand `tacl catalogue`, which automatically generates a blank catalogue file listing all texts in a given directory (folder) (without labels). Running this command over a corpus provides a useful base catalogue, and it can be convenient to construct catalogues by further editing such a base (e.g. by deleting unwanted files) using a `text editor` like Notepad++.

Core tests: Intersect and Difference

The core TACL tests compare two or more corpora, and are described in more detail, with examples, in the TACL GUI User's Manual. The two main types of comparison are Intersect and Difference. They are achieved, respectively, using the commands `tacl intersect` and `tacl diff`. Once more, instructions on how to run those commands can be retrieved at the command line using `help`: `tacl intersect -h, tacl diff -h.`

Users should also note that it is possible, when running a Difference test, to run an "asymmetric Difference", which outputs results for n-grams on only one side of the comparison (rather than both sides). This option is described in more detail in the TACL GUI User's Manual.

"Concatenated" tests: Supplied Intersect

As described in more detail in the TACL GUI User's Manual and the TACL Methods Guide, it is possible to "concatenate" the raw results of TACL tests, that is to say, link tests together in a concerted series. This allows us to find n-grams distributed in such patterns as: *in A and B, but not in C*.

The TACL operation that allows us to concatenate texts is "Supplied Intersect", or `tacl sintersect`. Once more, instructions on how to run the commands can be retrieved at the command line using `help`: `tacl sintersect -h.`

Note that TACL Supplied Intersect requires as input *raw* TACL results files. That is to say, it is usually highly inadvisable to input results that have already been put through the post-processing and filtering steps achieved by `tacl results`, as described immediately below. Doing so may produce false or misleading results.

Post-processing: TACL results

The raw results of TACL Intersect, Difference and Supplied Intersect tests are output in a format which produces some potential problems for human users. The first such problem is massive redundancy, in terms of the information the human user is typically looking for:

- As mentioned above, when the text-only corpus is created with `tacl strip`, the software reconstitutes text-only versions of all the witness texts documented in the Taishō apparatus. The database built with `tacl ngrams` then actually contains a separate row giving information about n-grams and counts for every one of those witnesses. As a result, any `query` to the database discovers separate information about the presence and count of a given n-gram in each witness, and raw TACL results then have a separate row of data for each witness. The vast majority of the time, however (wherever there is not a variant reading), this information will be completely redundant between witnesses, creating, for example, four copies of “the same” information (for, say, the Korean, Song, Yuan and Ming witnesses to the same text). It multiplies the burden on a human user to wade through these redundancies.
- Where a long string is shared by two (or more) texts in different labels, `tacl intersect` will output results including that string, but it will also include among the results every shorter string included within that string, down to the lower limit of the n-

gram length included in the database, because those shorter strings are also (necessarily, logically) shared by the same texts. For example, as has been known at least since work by Lévi and Chavannes a century ago, T453 is largely verbatim identical to **Ekottarikāgama* 48.3.[3] And indeed, the 12-gram 聽我所說彌勒出現國土豐樂 is shared by these two texts, and only them, in the entire CBETA corpus, and occurs exactly once in both texts. But both texts also contain the two 11-grams 我所說彌勒出現國土豐樂 and 聽我所說彌勒出現國土豐, both of which are included within that 12-gram; and also of the three 10-grams 聽我所說彌勒出現國土, 我所說彌勒出現國土豐, and 所說彌勒出現國土豐樂, all included in the same 12-gram; and so on for 9-grams, 8-grams etc., down to all 11 2-grams occurring in the 12-gram. This means that what, for the human, is a single item of relevant information—that 聽我所說彌勒出現國土豐樂 appears in T453 and EA 48.3—will be represented in raw results by 66 rows of information, for all the "sub-strings" occurring within the 12-gram; and this then needs to be multiplied by all the witnesses in the test—for T453, seven witnesses—to produce a total of 462 rows which, for the human, add up to the same single finding.

In order to reduce redundancies like these in the presentation of raw results, TACL includes a range of subcommands within `tacl results`, which allow filtering and manipulation of raw results files in a number of ways.

— `extend` (`-e` within `tacl results`): This operation takes n-grams occurring in a raw results file, and checks whether the larger n-grams containing those n-grams also occur with the same distribution. Users can envisage this by imagining that the programme begins, for example, with the substring -彌勒出現國土豐- (a 7-gram) from the T453/EĀ 48.3 example above (where it is also a unique match between these two texts), and "looks either side" of the match, to see that the 8-gram 彌勒出現國土豐樂 is also a match, as is the 9-gram 說彌勒出現國土豐樂, and the 10-gram 所說彌勒出現國土豐樂, and the 11-gram 我所說彌勒出現國土豐樂, and so on; eventually arriving at the 35-gram 聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就, which is also a match. This enables the user to see the 35-gram match as a single item of information, rather than have it spread over, say, 26 10-grams (the largest unit contained by a raw database covering 2-10-grams).

— `reduce` (within `tacl results`): Even if we have thus applied `extend` to find the largest contiguous matching string in our two texts, however, our extended results file will still also contain all its shorter constituent sub-strings. To eliminate this redundancy, the `reduce` operation takes a long string (n-gram), and checks all the shorter strings it contains (n-1, n-2, n-3...). Where the counts for the shorter string match the count of the longer string, it discards the shorter string from the results. In application to the 35-gram just "discovered" by `extend`, which has a count of 1 in each text in our results file, this function would eliminate from the results all the constituent sub-strings in each work that also have a count of 1.

[3] Lévi and Chavannes (1916): 191, 263, discussed in Anālayo (2010): p. 7 n. 45.

Applied in combination to `tacl intersect` results, `extend` and `reduce` thus achieve the following transformation. Raw results initially look like this (note that counts—in the second-to-rightmost column—are all identical):

聽我所說彌	5	T0125	base	1	EA
聽我所說彌勒	6	T0125	base	1	EA
聽我所說彌勒出	7	T0125	base	1	EA
聽我所說彌勒出現	8	T0125	base	1	EA
聽我所說彌勒出現國	9	T0125	base	1	EA
聽我所說彌勒出現國土	10	T0125	base	1	EA
我所說彌	4	T0125	base	1	EA
我所說彌勒	5	T0125	base	1	EA
我所說彌勒出	6	T0125	base	1	EA
我所說彌勒出現	7	T0125	base	1	EA
我所說彌勒出現國	8	T0125	base	1	EA
我所說彌勒出現國土	9	T0125	base	1	EA
我所說彌勒出現國土豐	10	T0125	base	1	EA
所說彌	3	T0125	base	1	EA
所說彌勒	4	T0125	base	1	EA
所說彌勒出	5	T0125	base	1	EA
所說彌勒出現	6	T0125	base	1	EA
所說彌勒出現國	7	T0125	base	1	EA
所說彌勒出現國土	8	T0125	base	1	EA
所說彌勒出現國土豐	9	T0125	base	1	EA
所說彌勒出現國土豐樂	10	T0125	base	1	EA

(...etc., for all 26 10-grams comprised within 聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就, and their constituent substrings.)

Extend and reduce yield a single data item like this:

聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	base	1	EA

Thus, the original massive redundancy in the results, from the perspective of what the human is looking for, has been eliminated, and we need no longer look at hundreds of rows of data to find "the same" item of information or evidence.

In addition, `tacl results` also includes the following operations, which also transform results into formats more readily interpreted by the human researcher.

— `zero fill` (`-z` within `tacl results`). For obvious reasons, n-grams with zero count in a given witness will not be otherwise included in raw results—any test only finds n-grams that *are* present in a text. Where an n-gram found in other witnesses for a text does not appear at all in a given witness, `zero fill` adds a row of data to results showing a

zero count for that witness. This allows comparison between witnesses in which a reading does appear, and those in which it does not (readers might like to think of this step as changing from implicit to explicit indication that the reading in question is missing).

— `collapse-witnesses`: For any cases in which the counts are identical for a given n-gram across multiple witnesses, this operation collapses the results into a single row of data, and lists in one column of the results the sigla for all witnesses with that count for that n-gram. This achieves a transformation like the following. Results initially contain a separate row for the above 35-gram for every separate witness to T125 (sigla for witnesses appear in the third-to-rightmost column), but there is in fact no variation between all witnesses (the 35-gram in question appears exactly once in each text):

聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	base	1	EA
聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	元	1	EA
聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	大	1	EA
聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	宋	1	EA
聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	明	1	EA
聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	明異	1	EA
聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	磧砂	1	EA
聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	聖	1	EA
聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	麗	1	EA

`collapse-witnesses` presents all this same information in the following more concise form:

聽我所說彌勒出現國土豐樂弟子多少善思念之執在心懷是時阿難從佛受教即還就	35	T0125	base 元 大 宋 明 明異 磧砂 聖 麗	1	EA

Here, all the information for this 35-gram is collapsed into a single row, and the sigla of the witnesses in which that n-gram appears with that count are all concentrated into a single set of information (again, third-to-rightmost cell).

Thus, for this 35-gram, we now have all the information the human user would want, concentrated into a single row—the 35-gram (like all its subordinate parts!) appears once only in this text (and in T453, which would be represented in a separate row), and the situation is the same in all eight or nine witnesses documented in the Taishō apparatus.

A next set of problems arises from the fact that, even with the results thus cleaned up to eliminate redundancy, it is quite possible, if not likely, that without further sorting and filtering, the raw results file will be so copious in its contents, and include so much extrinsic "noise" (from the perspective of the researcher's ultimate goal), that it will be impossible for a human researcher to go through all the results. To continue with the example of EĀ 48.3 and T453, our results include 3,644 instances of 比丘 (*bhikṣu*) in EĀ, but obviously, the fact that this string occurs in both EĀ and T453 does not indicate any special relationship between EĀ and T453, but rather, merely shows that this word is extremely common in these texts (as many others).

We can handle this problem by using subcommands within `tacl results` to filter the results by such criteria as the length of the n-gram, the count for the n-gram within each witness of each text, the number of works in which an n-gram occurs, and so on (a full list of all available filter criteria can be found by passing `tacl results -h`). In this instance, we might presume, for example, that shared strings of less than four characters in length are more probably recurring items of vocabulary, instead of matches in specific wording and content; and we might also estimate that the most telling evidence of such textual debts will occur when, as in our example above, a match is found precisely once in each text (and never anywhere else). Thus, we might use `tacl results` with the **`parameters`** and **`arguments`** `--max-count 2` and `--min-size 4`, in order to produce a subset of results that occur only once in each text, and are always at least four characters long.

As can be seen from the "help", `tacl results` also includes still other functions. Important among these are the following (note that most of these functions are not available through the TACL GUI):

- `group-by-ngram` and `group-by-work`: useful in tidying up the results of `tacl search` tests (see below) so that the information the human analyst wants is grouped together.
- `ngrams`: the user supplies a text file with a list of n-grams, and results for those n-grams are removed from the results file. This function can be useful when we conduct analyses in more than one phase (for example, if we realise partway through a given piece of research that we want to run the initial tests again with a different catalogue or corpus)—we can then cull from the results of later tests results that we already know about, in order to focus more clearly on discovering new information.
- `remove`: the user supplies a label (which must occur in the results file input to the `tacl results` process), and TACL removes all results with that label from the

results file. Can be useful, for example, if we ran a symmetric Difference test, and then later realise that we want to concentrate on only one side of the comparison—running a "remove" command can be quicker than redoing the initial Difference as an asymmetric Difference.

Human philological analysis: Handling results in Excel (or equivalent)

We suggested earlier that users `pipe` the results of TACL operations, and save them in `csv` format. To view such results in a clearer format, it is useful to import results into Microsoft Excel or an equivalent tool (such as LibreOffice or Calc from Apache OpenOffice; and there are certainly still more options for viewing and manipulating `csv` files.). Excel (or equivalents) also allow further sorting of the results, and this can make it possible to zoom in much more quickly on items of evidential interest. The TACL GUI User's Manual describes in more detail the steps in importing results into Excel and sorting them, and typical sort protocols for typical types of results.

Analysing results in context

As discussed in more detail in the TACL Methods Guide, for most typical research questions, using the methods developed to date, it is vital that results of TACL operations be analysed back in context by a philologically astute human researcher. Readers are urged to consult those other guidelines and take them into consideration in any work they do using TACL.

Other TACL functions

As mentioned earlier, "full-fledged" TACL (or "TACL proper"), i.e. TACL as run at the command line, allows users more exact control over TACL processes than the TACL GUI, and also offers a wider range of functions. A full list of TACL functions is available in the main TACL help menu: `tacl -h`. The most important of the additional functions not already described above—only one of them available through the GUI—are these:

- `tacl align`: the user inputs raw results of a previous TACL Intersect test. TACL uses these results to produce an output file that approximately aligns two user-specified witnesses of two user-specified works, to try and show where the overlap between those two texts is greatest, in terms of the occurrence of the Intersect results.
- `tacl catalogue`: allows the user to generate a full "blank" catalogue (without "labels") of all the works in a corpus.
- `tacl highlight`: the user inputs a list of n-grams, and specifies a single work within a corpus. TACL outputs a directory of HTML files, one for each witness of the specified work, in which all of the n-grams in the input list are highlighted. This function helps users to see quickly where in the text a given set of results occur (and possibly, thereby, to spot patterns in the distribution of results).
- `tacl search`: the user inputs a list of n-grams. TACL generates a list of works in a user-specified corpus, and for each witness of each work, lists in a single row of results all n-grams in the list that occur in that work. This function helps users to see where in a corpus a given set of results occurs most frequently. Such a test can be useful in finding "hotspots" where we find particular concentrations of a given set of n-grams (which might represent, for example, stylistic traits of a certain person or group). `tacl`

`search` is often useful in conjunction with the `tacl results` functions `group-by-ngram` and `group-by-work`.

Contact us

We welcome feedback on this Guide, and on the operation and design of TACL. Please feel free to email Radich.