Research Memorandum

Item Distiller: Text Retrieval for Computer-Assisted Test Item Creation

Derrick Higgins

Research & Development

October 2007
RM-07-05

Item Distiller: Text Retrieval for Computer-Assisted Test Item Creation

Derrick Higgins
ETS, Princeton, NJ

October 2007

As part of its educational and social mission and in fulfilling the organization's nonprofit charter and bylaws, ETS has and continues to learn from and also to lead research that furthers educational and measurement research to advance quality and equity in education and assessment for all users of the organization's products and services.

ETS Research Memorandums provide preliminary and limited dissemination of ETS research prior to publication. To obtain a PDF or a print copy of a report, please visit:

http://www.ets.org/research/contact.html

Abstract

This paper describes Item Distiller, a tool developed at ETS to aid in the creation of sentence-based test items. Item Distiller enables users to search for sentences that contain specific words, phrases, or grammatical constructions, so that these sentences can be edited to produce finished items. This approach to test authoring is both more efficient than writing items from scratch and more principled, due to its links to *item modeling*.

Key words: Grammar, information retrieval, item creation, automated item generation, item modeling

1 Overview

Many different types of educational assessments include a grammar component. Some tests, such as the SAT® (a college-entrance exam), the GRE® test for graduate-school applicants, the PRAXIS™ tests of teacher proficiency, and the GMAT (for business-school applicants) assess grammatical knowledge, among other things, by eliciting a sample of the examinee's writing directly (a *constructed-response* task). These and other tests, such as the TOEIC® (Test of English for International Communication™), may also assess grammar (or verbal skills more generally) by means other than eliciting essays. Such instruments may be either multiple-choice items or free-response items based on a single sentence (where the examinee might, for example, be asked to fill in an appropriate word). The focus of this paper is the creation of sentence-based multiple-choice test items for language assessment.

ETS has developed a targeted information-retrieval (IR) application (Item Distiller) to search a corpus of written English and return sentences that may form the basis of good test items for a specific testing program. This tool allows test items to be developed more quickly and easily than manual item authoring, which is important not only for cost-related reasons, but also because of test security considerations. Most importantly, however, this IR approach to item creation helps to enforce an *item modeling* approach to assessment, in which test items are not constructed and chosen in isolation, but rather are designed as part of a family of items that provide evidence of an examinee's mastery of predefined skills.

Item Distiller builds upon other projects that have been developed to allow users to search for grammatical patterns in corpora, but it is original in its focus on test item development. It is also unique in the simplicity of its architecture, which allows for fast searches and simple integration of new corpora into the system.

2 Test Item Authoring

Examples of the sort of sentence-based test items under consideration here are presented in Figures 1 and 2. Each of these items contains a single sentence in the *stem* (the main body of the item) and four options. The examinee is asked to select the option that best completes the sentence (when the option text is substituted for the lacuna in the stem). This is known as the *incomplete sentence* item type.[1]

Figure 1 shows an item from the 1993 TOEIC exam, which assesses the examinee's ability to properly use the pronoun *another* to refer to a nominal expression. The *distractors* (incorrect responses) consist of other words and phrases that are morphologically related to *another*.

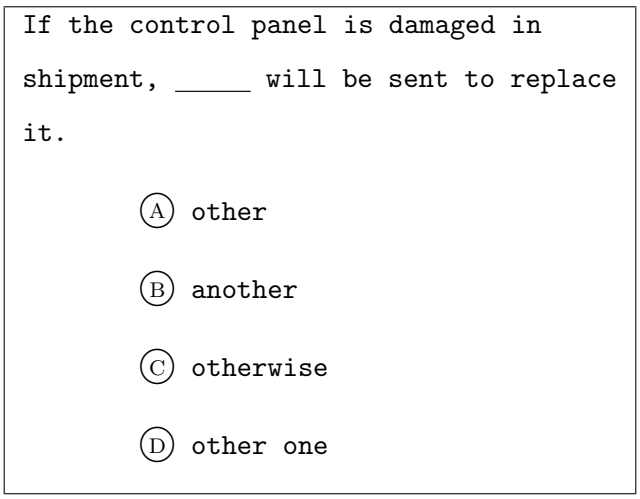

***Figure* 1. Sample item from the TOEIC test.**

Figure 2 is a sample question from the Test of English as a Foreign Language™ (TOEFL®), which tests the use of *only* to contrastively focus a nominal. In this case, the distractors are various incorrect permutations of the correct word order.

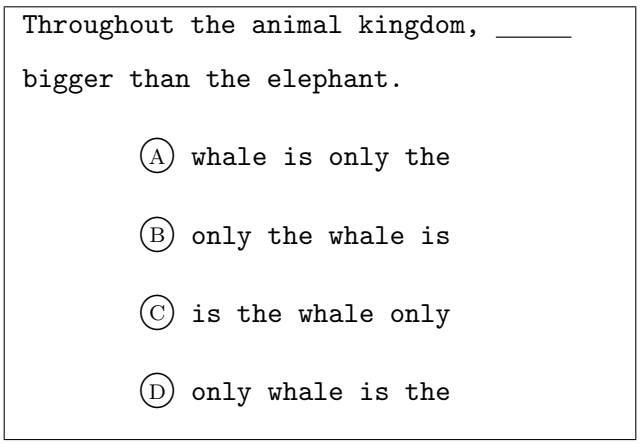

***Figure* 2. Sample item from the TOEFL test.**

Typically, items such as these are written from scratch or constructed from sentences encountered in a text. Test developers determine that they need a certain number of items based on using the comparative construction, on pronoun usage, on redundant word usage, and so on. Later, these newly created items may be pretested in order to determine their difficulty level empirically.

Unfortunately, item writing can be quite labor-intensive. For each item to be written, test developers need either to locate a good sentence in a book or to think up a scenario in which to embed the relevant grammatical material. Then they need to adjust the phrasing of the sentence to suit their needs. (For example, in the case of items built on use of the comparative, they might have to decide whether the item will test comparative adjectives or nouns (*more forgetful* vs. *more cheese*) and whether to use the morphological or periphrastic form of the comparative (*faster* vs. *more quickly*).) Finally, test developers need to determine a good set of distractors for the item.

To allow test developers to produce items more quickly, ETS has developed Item Distiller, an application to search corpora for existing sentences that might be suitable for these items. Such an approach has the added benefit of producing items that are much more natural sounding, because they are based on sentences that were actually employed for some communicative purpose, rather than a stilted set of imagined scenarios.

3 Item Models

Another reason for pursuing this semi-automated approach to sentence-based item authoring is the connection of this method to *item modeling* (Bejar et al., 2002; LaDuca, Staples, Templeton, & Holzman, 1986). The idea behind item modeling is that test items can be produced as members of general classes that share certain characteristics. The exact set of characteristics that item instances must share in order to be considered members of a single model will differ from one implementation to another.

Typically, items from the same model will cover the same or very similar content, but may vary in wording, the order in which distractors and the key occur in a multiple-choice item, and other superficial aspects. Consider, for example, mathematical items dealing with the Pythagorean theorem. It might be useful to generate these items from a general

item model, in which the basic content ($a^2 + b^2 = c^2$) remains the same, but the size of the triangle differs from item to item. In the case of the sort of items shown in Figures 1–2, which are designed to assess grammatical knowledge, an item model might encapsulate a certain narrow content area, such as the use of *only* with a subject noun phrase in its focus. The choice of a sentence instantiating this pattern could vary between item instances.

Another criterion that may be used as a basis for designing item models is that the variation among instances from the model ought to be systematically related to the difficulty level of the item. Referring again to the hypothetical Pythagorean theorem item model, instances might vary in the length of the triangle's sides, whether the lengths are integers or real numbers, and whether the examinee is asked to solve for the hypotenuse or one of the other sides. The model will be more useful, though, if it allows test developers to predict the effect each variation will have on the item's difficulty. In verbal test items, the variables affecting difficulty are more likely to be such things as the particular choice of distractor phrases and the vocabulary level of the text (cf., for example, Embretson & Wetzel, 1987).

The sort of patterns that test developers write in order to search for candidate items with Item Distiller are at approximately the right level of specificity to serve as the basis for new item models. A pattern might encapsulate the content area of using the comparative construction, or plurals versus possessives. Once the model has been built, the number of instances that can be generated from it is limited only by the corpus resources at hand.

4 Construction-Based Sentence Search

Unlike most IR applications, the construction of test items to assess language skills requires more than a bag-of-words approach. Because these items are typically based on grammatical constructions, rather than single vocabulary items, Item Distiller requires the full power of regular expressions to identify good candidate sentences.

Figure 1 shows that sentences suitable for such items should be fairly simple to locate in a corpus. Since the item has to do with the correct use of *another*, all sentences that contain that word can simply be returned. In fact, of course, this approach overgenerates far too much to be practical. What is needed is not just sentences with *another*, but

sentences with *another* functioning as a complete noun phrase and referring to a nominal expression before it in the same sentence.

Similarly, a sentence similar to that in the TOEFL item in Figure 2 requires the word *only*, followed by a subject noun phrase, followed by a verb. It is not enough to find a sentence with the trigger word *only*, since *only* can take phrases of other categories as its focus as well or noun phrases other than the subject of a sentence. (As discussed in the previous section, the issue of exactly how to can divide the *essential* characteristics of a sentence-based item, which are necessary to include in producing similar items, from the *accidental* ones, which may vary between items within a model, is somewhat arbitrary.[2])

In addition, some item types may not demand the appearance of specific words at all (like *another* and *only*). For example, the only distinguishing characteristic of items testing proper verbal agreement is that they have a main verb and multiple noun phrases earlier in the sentence with which the verb could potentially agree. In this case as well, it is clear that a simple exact-match search for specific words will not work.

What is needed, then, is a way of searching for sentences matching a sentence *pattern*, which corresponds to something like a grammatical construction. Test developers sometimes need to write patterns that reference specific words and sometimes also need to use more general classes such as *a noun* or *a noun phrase*.

4.1 Previous Work

This very issue has been addressed by other researchers, although never with the intention of aiding test development. (Typically, these other projects aim to provide data mining tools for linguists interested in finding corpus examples of specific constructions or phrase types.)

One such project is the Linguist's Search Engine (LSE; Resnik & Elkiss, 2004) on the Web at `http://lse.umiacs.umd.edu/`. LSE has a graphical interface for searching an indexed database of syntactically-parsed documents from the Web. The user can construct parse tree fragments and then search for text that matches those tree fragments.

Many similar programs have been developed to search corpora on the user's local machine, rather than a preindexed set of documents in a central repository. These include

the ICE Corpus Utility Program (ICECUP; Wallis, Aarts, & Nelson, 1999; Wallis & Nelson, 2001), the IMS Corpus Query Processor (Christ, 1994), TIGERSearch (Lezius & König, 2000), the NITE XML Toolkit (Carletta et al., 2004), Xaira (Burnard & Dodd, 2003), and Xlex (Lemnitzer, 1997). They differ from one another and from LSE in the interface for specifying search patterns, the format in which corpora are stored, the ease in which new corpora can be indexed for searching, and whether they are freely available for use.

Of course, some of the most widely used tools for pattern-based searching of text are the UNIX regular-expression (Friedl, 1998) tools such as `grep`, `sed`, and `awk`. These tools are not particularly well-tailored to the task of searching for grammatical patterns, but they are simple and flexible enough that they could be used for this purpose. The main challenge is that these tools require patterns to be written in terms of characters and character classes, so that to write a pattern to express the concept *a* sequence of adjectives, for example, would be a very difficult task. Another tool that should be mentioned in this context is `tgrep`, a tool for matching tree fragments in parsed text. `tgrep` is distributed with the Penn Treebank (Marcus, Santorini, & Marcinkiewicz, 1993) and has a similar interface to `grep`.

Finally, there is one additional information retrieval project that, while it does not allow the specification of complex grammatical patterns, does contribute to the goal of test item creation. This is the SourceFinder project (Katz & Bauer, 2001; Passonneau, Hemat, Plante, & Sheehan, 2002) at ETS, which allows test developers to search for articles and paragraphs by topic area and key words. SourceFinder is useful in locating good source texts for reading comprehension test items, which are based on a short reading passage presented to the examinee, but it is clearly not suitable for locating sentences according to grammatical patterns.

4.2 Design of Item Distiller

Item Distiller can be considered as a kind of hybrid combining the best aspects of existing pattern search software for the application of test item generation. Systems like LSE and ICECUP have the advantage that they are specifically designed for linguistic

applications and are therefore relatively simple to use. However, matching arbitrary parse tree fragments can be a slow process. On the other hand, simple applications like `grep` are not as user-friendly, but they are lightweight and fast to execute.

Item Distiller combines the efficiency and power of regular expressions with a simplified user interface in the model of LSE and ICECUP. Figure 3 shows a screenshot of Item Distiller.

ID interface. Item Distiller is written as a Java applet, which can run in a Web browser, as shown in Figure 3. This allows the corpora accessed through the interface to be housed and maintained centrally and obviates the need for the software to be installed on the user's local machine.

Users choose a corpus to search (referred to as a *database* in the screenshot), enter a sentence pattern into the appropriate text entry box and then click on the Find Sentences button. Item Distiller will then return sentences matching the pattern as they are found. The part of the sentence that matched the pattern is highlighted in red as an aid to the user. If the results are too broad, too narrowly focused, or simply wrong, the user can modify the search pattern to rectify the problem.

A number of additional search parameters can be specified in the user interface to modify Item Distiller's behavior. The user can specify a maximum number of sentences to return, constraints on the length of sentences to be considered, and the sentence number at which the search should begin. (The user may have already seen the first set of matches.)

Once a user has identified a useful search pattern for a particular item type, the pattern can be saved as a new item model, which can be loaded at a later time to generate new instances. A number of these predefined patterns are distributed with the Item Distiller software to illustrate how sentence patterns are written and can be loaded directly from the Samples menu.

One final option that can be specified from the user interface is the *language* of the search. Currently, the user can choose to search either English or Spanish texts. The choice of a language in Item Distiller makes a different drop-down menu of corpora available to the user, and it also changes the system's expectations about the grammatical annotation scheme to expect in the corpora to be searched. This paper, unless indicated otherwise,

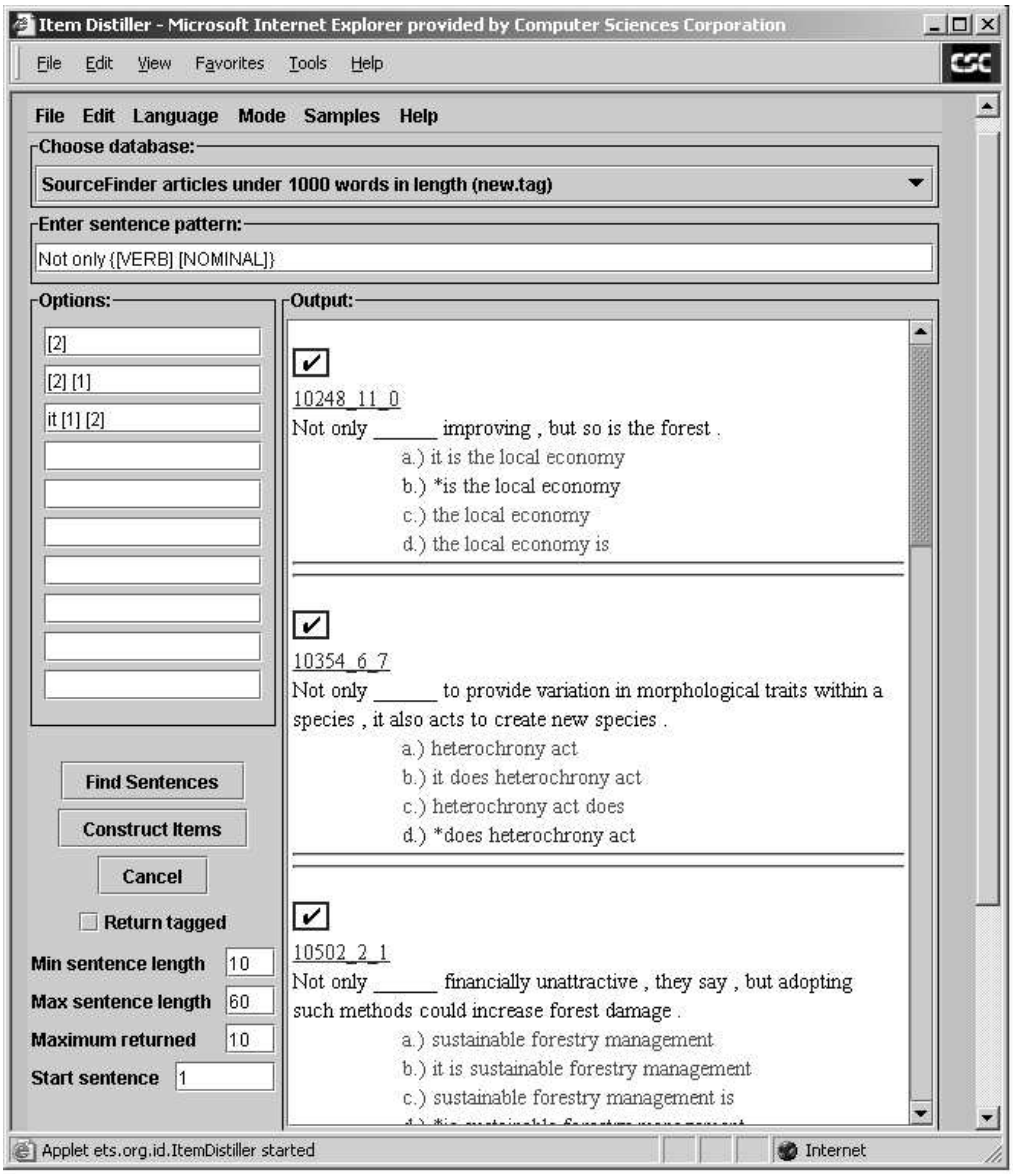

***Figure* 3. Screenshot of Item Distiller applet.**

will cover the English search mode of Item Distiller. Spanish support has been added fairly recently and is under evaluation by a number of test developers.

Corpus preparation. The format of the corpora used with Item Distiller is very simple. Basically, it operates on part-of-speech (POS) tagged text, formatted as in Figure 4. The first token on each line is a unique identifying symbol for the sentence (in Figure 4, this is simply the line number), while the rest of the line consists of the sentence text, tokenized and with the POS tag appended to each word. For English, ETS uses the Penn Treebank tagset (Santorini, 1990); for Spanish, the CRATER tagset (Sánchez-Léon, 1994).

```
1 The_DT wisest_JJS of_IN the_DT wise_JJ may_MD err_VB ._.
2 He_PRP who_WP learns_VBZ must_MD suffer_VB ._.
3 The_DT man_NN whose_WP$ authority_NN is_VBZ recent_JJ is_VBZ always_RB stern_JJ ._.
4 The_DT reward_NN of_IN suffering_NN is_VBZ experience_NN ._.
5 There_EX are_VBP times_NNS when_WRB fear_NN is_VBZ good_JJ ._.
```

***Figure* 4. Illustration of Item Distiller corpus format, using quotes from Aeschylus.**

This representation does not contain as much information as the parsed text used by other systems, but its simplicity is also a virtue. Because taggers are considerably faster than full sentence parsers, it is quicker and easier to incorporate new data for use with Item Distiller. Taggers are also less prone to catastrophic failure when presented with malformed text, so that Item Distiller is more robust than it might be if it used complete sentence parses. Finally, because POS tags do not encode hierarchical information, this representation can be searched with regular expressions, for which there exist very efficient algorithms.

The sentence data may be stored either in a flat file or in an SQL database.

Item Distiller sentence patterns. Sentence patterns in Item Distiller are written in a high level syntax that allows users to refer directly to useful grammatical categories such as nouns, verbs, noun phrases, prepositional phrases, and clauses, even though these categories do not correspond directly to the part-of-speech tags found in the raw data. These high-level patterns are compiled into regular expressions which can be straightforwardly

matched against each candidate sentence by the application.

Single word categories such as `NOUN` or `VERB` are simple to compile into regular expressions. First, designate a certain set of tags as *nominal* (such as `NN`, `NNS`, `NNP`, and `NNPS` in the Penn Treebank tagset). Given there is a regular expression corresponding to *word with tag T*, for each tag T in this set, one can then derive the regular expression for the category NOUN simply by ORing together all of these tag-based regular expressions.

Phrasal categories such as `NOUNP` (noun phrase) are also not too difficult to compile into reasonably accurate regular expressions. This simply involves writing a finite-state approximation to the phrasal rules that define a noun phrase (see Roche & Schabes, 1997, for sample finite-state grammatical rules). Writing the compiler to turn high level linguistic patterns into regular expressions is not difficult, but the resulting regular expressions are generally too complicated for human inspection of them to prove revealing. Figure 5 illustrates this with the regular expression corresponding to the pattern `NOMINAL` for a single noun phrase.

The syntax of the sentence patterns that can be written using these primitives (such as `NOUN, VERB, NOUNP, ADJ,` etc.) is more detailed than can be described here. However, Table 1 provides some examples of the sorts of patterns that can be written. Literal words can be included in patterns along with symbols referring to grammatical classes. The default is to require strict adjacency between elements (for example, the pattern `NOUN VERB` will return only sentences in which a noun is found directly before a verb), but intervening material may be allowed if an ellipsis is added (`NOUN ... VERB`). Quantifiers such as the Kleene star are permitted. (`ADJ+` will match the longest possible sequence of adjectives.) Backreferences are also supported, as discussed below.

The pattern used in Figure 3, *Not only* `[VERB] [NOMINAL]`, identifies sentences with inverted subject-verb word order because of the initial adverbial element *not only*. (A `NOMINAL` is a sort of base noun phrase in this system—a `NOUNP` stripped of any trailing prepositional phrases.) The brackets in this pattern are used in the incomplete sentence item generation mode, discussed below. Finally, it should be noted that even though regular expression matching is the basic technology underlying this pattern search application, the pattern is not simply applied to each sentence in turn to see if it matches. ETS has

```
(?:(?:(?<!\S)[\w\-]+\_DT\b|(?<!\S)[\w\-]+\_DTI\b|(?<!\S)[\w\-]+\_DTX\b|(?<!\S)[\w\-]+\_DTS\b|(?<!\S)[\w\-]+
\_AT\b|(?<!\S)[\w\-]+\_ATI\b|(?<!\S)[\w\-]+\_ABX\b|(?<!\S)[\w\-]+\_PDT\b|(?<!\S)[\w\-]+\_CD\b|(?<!\S)[\w\-\']+
\_WP\$(?!\S)|(?<!\S)[\w\-\']+\_PRP\$(?!\S))\s+)*(?:(?:(?<!\S)[\w\-]+\_NN\b|(?<!\S)[\w\-]+\_NNP\b|(?<!\S)[\w\-]+
\_NNS\b|(?<!\S)[\w\-]+\_NNPS\b|(?<!\S)[\w\-]+\_PN\b|(?<!\S)[\w\-]+\_NR\b)(?:\s+(?:(?<!\S)[\w\-]+\_NN\b|(?<!\S)
[\w\-]+\_NNP\b|(?<!\S)[\w\-]+\_NNS\b|(?<!\S)[\w\-]+\_NNPS\b|(?<!\S)[\w\-]+\_PN\b|(?<!\S)[\w\-]+\_NR\b))*\s+
(?<!\S)[\w\-\']+\_POS\b\s+)?(?:(?:(?:(?<!\S)[\w\-]+\_RB\b|(?<!\S)[\w\-]+\_RBR\b|(?<!\S)[\w\-]+\_RBS\b|(?<!\S)
[\w\-]+\_ABL\b)\s+)*(?:(?<!\S)[\w\-]+\_JJ\b|(?<!\S)[\w\-]+\_JJR\b|(?<!\S)[\w\-]+\_JJS\b|(?<!\S)[\w\-]+\_AP\b)
(?:\s+(?:(?:(?<!\S)[\w\-]+\_RB\b|(?<!\S)[\w\-]+\_RBR\b|(?<!\S)[\w\-]+\_RBS\b|(?<!\S)[\w\-]+\_ABL\b)\s+)*(?:
(?<!\S)[\w\-]+\_JJ\b|(?<!\S)[\w\-]+\_JJR\b|(?<!\S)[\w\-]+\_JJS\b|(?<!\S)[\w\-]+\_AP\b))*\s+)?(?:(?<!\S)[\w\-]+
\_NN\b|(?<!\S)[\w\-]+\_NNP\b|(?<!\S)[\w\-]+\_NNS\b|(?<!\S)[\w\-]+\_NNPS\b|(?<!\S)[\w\-]+\_PN\b|(?<!\S)[\w\-]+
\_NR\b)(?:\s+(?:(?<!\S)[\w\-]+\_NN\b|(?<!\S)[\w\-]+\_NNP\b|(?<!\S)[\w\-]+\_NNS\b|(?<!\S)[\w\-]+\_NNPS\b|(?<!\S)
[\w\-]+\_PN\b|(?<!\S)[\w\-]+\_NR\b))*(?:\s+(?<!\S)[\w\-]+\_CC\b\s+(?:(?:(?<!\S)[\w\-]+\_DT\b|(?<!\S)[\w\-]+
\_DTI\b|(?<!\S)[\w\-]+\_DTX\b|(?<!\S)[\w\-]+\_DTS\b|(?<!\S)[\w\-]+\_AT\b|(?<!\S)[\w\-]+\_ATI\b|(?<!\S)[\w\-]+
\_ABX\b|(?<!\S)[\w\-]+\_PDT\b|(?<!\S)[\w\-]+\_CD\b|(?<!\S)[\w\-\']+\_WP\$(?!\S)|(?<!\S)[\w\-\']+\_PRP\$(?!\S))
\s+)*(?:(?:(?<!\S)[\w\-]+\_NN\b|(?<!\S)[\w\-]+\_NNP\b|(?<!\S)[\w\-]+\_NNS\b|(?<!\S)[\w\-]+\_NNPS\b|(?<!\S)
[\w\-]+\_PN\b|(?<!\S)[\w\-]+\_NR\b)(?:\s+(?:(?<!\S)[\w\-]+\_NN\b|(?<!\S)[\w\-]+\_NNP\b|(?<!\S)[\w\-]+\_NNS\b
|(?<!\S)[\w\-]+\_NNPS\b|(?<!\S)[\w\-]+\_PN\b|(?<!\S)[\w\-]+\_NR\b))*\s+(?<!\S)[\w\-\']+\_POS\b\s+)?(?:(?:
(?:(?<!\S)[\w\-]+\_RB\b|(?<!\S)[\w\-]+\_RBR\b|(?<!\S)[\w\-]+\_RBS\b|(?<!\S)[\w\-]+\_ABL\b)\s+)*(?:(?<!\S)
[\w\-]+\_JJ\b|(?<!\S)[\w\-]+\_JJR\b|(?<!\S)[\w\-]+\_JJS\b|(?<!\S)[\w\-]+\_AP\b)(?:\s+(?:(?:(?<!\S)[\w\-]+
\_RB\b|(?<!\S)[\w\-]+\_RBR\b|(?<!\S)[\w\-]+\_RBS\b|(?<!\S)[\w\-]+\_ABL\b)\s+)*(?:(?<!\S)[\w\-]+\_JJ\b|(?<!\S)
[\w\-]+\_JJR\b|(?<!\S)[\w\-]+\_JJS\b|(?<!\S)[\w\-]+\_AP\b))*\s+)?(?:(?<!\S)[\w\-]+\_NN\b|(?<!\S)[\w\-]+\_NNP\b
|(?<!\S)[\w\-]+\_NNS\b|(?<!\S)[\w\-]+\_NNPS\b|(?<!\S)[\w\-]+\_PN\b|(?<!\S)[\w\-]+\_NR\b)(?:\s+(?:(?<!\S)[\w\-]+
\_NN\b|(?<!\S)[\w\-]+\_NNP\b|(?<!\S)[\w\-]+\_NNS\b|(?<!\S)[\w\-]+\_NNPS\b|(?<!\S)[\w\-]+\_PN\b|(?<!\S)[\w\-]+
\_NR\b))*)?
```

***Figure* 5. Regular expression for the pattern `NOMINAL`.**

implemented some shortcuts where possible to increase the speed of the program, including using knowledge about the exact words that a sentence pattern requires, if any, to prefilter the candidate sentences.

Automatic item creation. In addition to locating sentences for use in test items, Item Distiller provides the option of postprocessing the sentences returned, in order to automatically produce complete items. The test developer can then review these candidate items to ensure that they meet the specifications for the item type desired.

Figure 3 illustrates this facility, which is invoked when the user presses the Construct Items button. The curly braces in the sentence pattern around `[VERB] [NOMINAL]` indicate the position in the sentence where the blank will be found in an incomplete sentence item. Correspondingly, the part of the sentence matching this part of the pattern is provided as

Table 1.

Sample Sentence Patterns Appropriate for Item Distiller

Pattern	Description
`VERB NOUN`	A verb immediately followed by a noun
`NOUN:-ity`	A noun ending in *ity*
`NOUN:8-10`	A noun of 8-10 letters
`VBD out of NOUNP`	A past tense verb followed by *out of*, followed by a noun phrase
`more ADJ ... than`	the word *more* followed by an adjective, with the word *than* appearing later in the sentence (A rough cut at isolating the comparative with adjectives)

the item key in the automatically generated items.

The distractor options are specified in the text entry fields along the left edge of the application window. These distractors may simply be alternate words or phrases, but they may also include material from the matched pattern, as in Figure 3. The elements `[1]` and `[2]` in that figure are *backreferences*, which refer to elements from the pattern that were captured with square brackets. `[1]` refers back to the first element captured, namely `[VERB]`, and `[2]` refers back to the second element captured, `[NOMINAL]`. This capability is important for item creation because the distractors chosen for an item often depend on the exact words of the key.

The syntax for automatic generation of error identification items and Sentence Correction items differs somewhat from that for incomplete sentence items, but the basic function is the same.

5 Future Work

Future work will extend Item Distiller's capabilities in several directions.

First and foremost, ETS will incrementally improve the interface of the program to

address the feedback received from users. The current interface is essentially a prototype that embodies assumptions about how test developers ought to work with the program. Now that users are actually using the tool in their day-to-day work at ETS, the tool is being developed collaboratively. Information is gathered about actual usage patterns, so that users' needs can be accommodated in the next version of Item Distiller. These needs include cosmetic and ergonomic issues and substantive decisions such as the number of item types supported by the interface. At present, Item Distiller can produce candidate items in three different multiple-choice formats, but support for certain free-response item types may be added as well.

Second, search modes are planned for other languages for Item Distiller so that users can construct items based on constructions in French, German, Japanese, or any other language of interest. Because of the lightweight design of the application, it is a fairly straightforward task to add another language. All that is required are a set of tagged corpora from the language and a mapping from low level tagger symbols to the higher level word and phrase classes used in sentence patterns (such as `NOUN` and `NOUNP`). Currently, Item Distiller supports only English and Spanish.

Third, there are plans to integrate Item Distiller with other item creation tools under development at ETS. Ultimately, the goal is to present Item Distiller together with SourceFinder and other tools as a complete item development environment. This environment would allow users not only to search for good sources for their items, but also manage instances of item models and to predict the difficulty level of an item as it is authored.

Fourth, other aspects of linguistic structure will be incorporated into Item Distiller. While the simplicity of the tool—relying only on part-of-speech tags rather than a full-sentence parse—is a virtue, these tags may not provide all the information necessary to extract sentences for some items. In particular, the addition of named entity information to Item Distiller's data files may be useful and is unlikely to result in much complication of the pattern syntax. A phonetic search mode is also being developed for Item Distiller, which will allow users to search for sentences with specific sound patterns. (This may be useful in selecting sentences for a read-aloud task on assessments for English as a foreign

language, for example.)

Finally, while Item Distiller's pattern syntax is fairly simple, it remains something of a hurdle for test developers, who are generally not accustomed to text processing and regular expressions. Ideally, the pattern appropriate to a construction, such as the comparative, could be abstracted by some automated process. A set of existing items based on the comparative construction would be fed in, and the application would produce a suggested pattern as a starting point or perhaps hide the pattern from the user entirely. While this scenario probably cannot be perfectly realized, some progress could be made in identifying likely patterns directly from existing items, to reduce the burden on users.

6 Conclusion

Item Distiller is an example of the judicious application of information retrieval tools in a very specialized domain. It is a somewhat uncharacteristic IR application because it uses the full power of regular expressions, rather than a simple bag-of-words approach. This approach is clearly necessary, however, in order to identify the constructions that form the basis of language assessments.

While other construction-based sentence search algorithms exist, Item Distiller has the advantages of simplicity, which makes it suitable for a broad user base and a targeted focus on item creation, which maximizes its utility for test developers in particular.

References

Bejar, I., Lawless, R., Morley, M., Wagner, M., Bennett, R., & Revuelta, J. (2002). *A feasibility study of on-the-fly item generation in adaptive testing* (ETS Research Rep. No. RR-02-23). Princeton, NJ: ETS.

Burnard, L., & Dodd, T. (2003). Xaira: An XML aware tool for corpus searching [Uerel Technical Rep. No. MS-CIS-90-47]. In D. Archer, P. Rayson, A. Wilson, & T. McEnery (Eds.), *Proceedings of the corpus linguistics 2003 conference* (pp. 142–144). Lancaster, UK: University Centre for Computer Research on Language, Lancaster University.

Carletta, J., McKelvie, D., Isard, A., Mengel, A., Klein, M., & Møller, M. B. (2004). A generic approach to software support for linguistic annotation using XML. In G. Sampson & D. McCarthy (Eds.), *Corpus linguistics: Readings in a widening discipline* (pp. 449–459). New York: Continuum International.

Christ, O. (1994). A modular and flexible architecture for an integrated corpus query system. In *Proceedings of COMPLEX '94: 3rd conference on computational lexicography and text research.* Budapest, Hungary.

Embretson, S., & Wetzel, C. D. (1987). Component latent trait models for paragraph comprehension tests. *Applied Psychological Measurement*, *11*, 175–193.

Friedl, J. E. F. (1998). *Mastering regular expressions*. Cambridge, MA: O'Reilly.

Irvine, S. H., & Kyllonen, P. C. (Eds.). (2000). *Item generation for test development*. Mahwah, NJ: Lawrence Erlbaum Associates.

Katz, I. R., & Bauer, M. I. (2001). SourceFinder: Course preparation via linguistically targeted web search. *Educational Technology & Society*, *4*(3), 45–49.

LaDuca, A., Staples, W. I., Templeton, B., & Holzman, G. B. (1986). Item modeling procedure for producing content-equivalent multiple-choice questions. *Medical Education*, *20*(1), 53–56.

Lemnitzer, L. (1997). *Extraktion komplexer Lexeme aus Textkorpora* (Vol. 180). Tübingen, Germany: Niemeyer.

Lezius, W., & König, E. (2000). Towards a search engine for syntactically annotated corpora. In W. Zühlke & E. G. Schukat-Talamazzini (Eds.), *Konvens 2000 Sprachkommunikation* (pp. 113–116). Ilmenau, Germany: VDE-Verlag.

Marcus, M., Santorini, S., & Marcinkiewicz, M. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330.

Passonneau, R., Hemat, L., Plante, J., & Sheehan, K. (2002). *Electronic sources as input to GRE® reading comprehension item development: SourceFinder prototype evaluation* (ETS Research Rep. No. RR-02-12). Princeton, NJ: ETS.

Resnik, P., & Elkiss, A. (2004). *The Linguist's Search Engine: Getting started guide* (Tech. Rep. No. LAMP-TR-108/CS-TR-4541/UMIACS-TR-2003-109). College Park: University of Maryland.

Sánchez-Léon, F. (1994). *Spanish tagset for the CRATER project.* Retrieved June 30, 2006, from `http://xxx.lanl.gov/format/cmp-lg/9406023`.

Santorini, B. (1990). *Part-of-speech tagging guidelines for the Penn Treebank project* (Tech. Rep. No. MS-CIS-90-47). Philadelphia: Department of Computer and Information Science, University of Pennsylvania.

Wallis, S., Aarts, B., & Nelson, G. (1999). Parsing in reverse – Exploring ICE-GB with fuzzy tree fragments and ICECUP. In J. M. Kirk (Ed.), *Corpora galore: Papers from the 19th international conference on English language research on computerised corpora, ICAMEs-98* (pp. 335–344). Amsterdam: Rodopi.

Wallis, S., & Nelson, G. (2001). Knowledge discovery in grammatically analysed corpora. *Data Mining and Knowledge Discovery*, *5*(4), 305–335.

Notes

[1] There are other sentence-based item types as well, such as the *error identification* type, in which the examinee must identify the location of a grammatical error in a sentence, but for the sake of simplicity this paper will concentrate on the incomplete sentence type.

[2] See Sidney Irvine's introduction to Irvine and Kyllonen (2000) for a related distinction between *radicals*, variables that may affect the cognitive properties of an item, and *incidentals*, variable that do not.