A Python Toolkit for Universal Transliteration

Ting Qian[1], Kristy Hollingshead[2], Su-youn Yoon[3], Kyoung-young Kim[4], Richard Sproat[5]

University of Rochester[1], OHSU[2], ETS[3], UIUC[4], OHSU[5]
ting.qian@rochester.edu[1], hollingk@cslu.ogi.edu[2], syoon9@gmail.com[3], kkim36@illinois.edu[4], rws@xoba.com[5]

Abstract

We describe `ScriptTranscriber`, an open source toolkit for extracting transliterations in comparable corpora from languages written in different scripts. The system includes various methods for extracting potential terms of interest from raw text, for providing guesses on the pronunciations of terms, and for comparing two strings as possible transliterations using both phonetic and temporal measures. The system works with any script in the Unicode Basic Multilingual Plane and is easily extended to include new modules. Given comparable corpora, such as newswire text, in a pair of languages that use different scripts, `ScriptTranscriber` provides an easy way to mine transliterations from the comparable texts. This is particularly useful for underresourced languages, where training data for transliteration may be lacking, and where it is thus hard to train good transliterators. `ScriptTranscriber` provides an open source package that allows for ready incorporation of more sophisticated modules — e.g. a trained transliteration model for a particular language pair. `ScriptTranscriber` is available as part of the nltk_contrib source tree at `http://code.google.com/p/nltk/`.

1. Introduction

This paper reports on a toolkit for extracting transliteration pairs between scripts called `ScriptTranscriber`. `ScriptTranscriber` includes modules for producing guesses at pronunciations for any word in *any* script in the Unicode Basic Multilingual Plane; for computing edit distances between strings using a variety of measures including phonetic distance; for computing time correlations between terms in comparable corpora; and providing a set of prepackaged recipes for mining possible transliteration pairs from comparable corpora. `ScriptTranscriber` is useful in two major ways:

1. Given comparable corpora, such as newswire text, in a pair of languages that use different scripts, `ScriptTranscriber` provides an easy way to mine transliterations from the comparable texts. This is particularly useful for underresourced languages, where training data for transliteration may be lacking, and where it is thus hard to train good transliterators.

2. `ScriptTranscriber` provides an open source package that allows for ready incorporation of more sophisticated modules — e.g. a trained transliteration model for a particular language pair.

`ScriptTranscriber` consists of approximately 7,500 lines of object-oriented Python. Some of the modules require PySNoW, the Python interface to the SNoW machine-learning package (Carlson et al., 1999) available from the Cognitive Computation Group at the University of Illinois at Urbana-Champaign.[1] `ScriptTranscriber` is available as part of the nltk_contrib source tree at `http://code.google.com/p/nltk/` (Loper and Bird, 2002).

[1]PySNoW must be downloaded separately from `http://l2r.cs.uiuc.edu/~cogcomp/`.

2. Modules and classes

The modules and classes of `ScriptTranscriber` are as follows.

First there is the **XML document structure** module, an example of which is shown in Figure 1. The top-level XML representation consists of a set of tupled documents, ordered according to some reasonable criterion such as time. Each *doc* element consists of one or more *lang* elements, which represent the original document(s) in the named language. Within each *lang* are a set of tokens, in no particular order, which represent terms—typically names—that have been extracted during the term extraction phase described below, along with a set of possible pronunciations and their counts. Within each *doc*, the *lang* elements are intended to consist of terms derived from comparable or parallel texts. For example, in Figure 1 the English document is assumed to be comparable to the Chinese document.

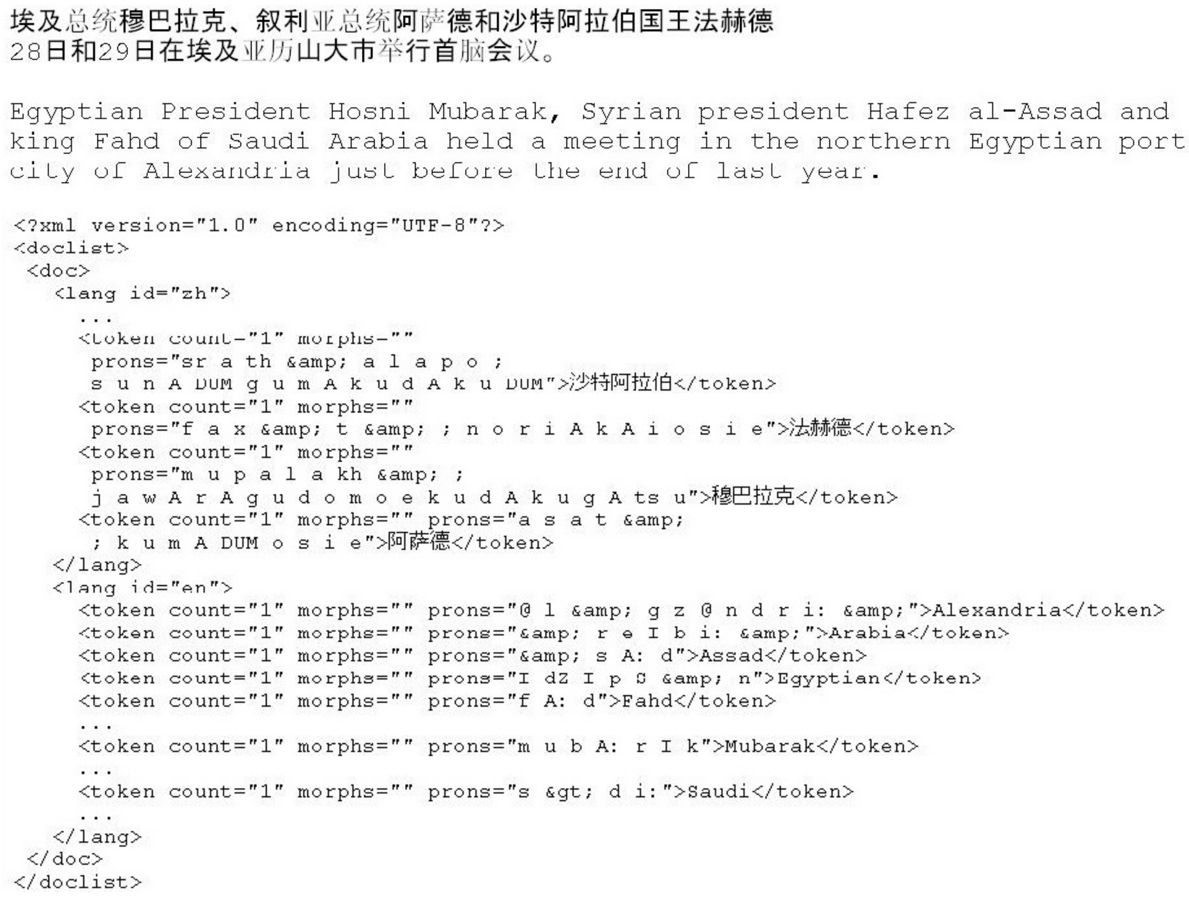
The **term extractor** class extracts interesting terms from raw text, i.e. terms that are likely to be transliterated across scripts. We provide five specializations of this:

- A simple capitalization-based extractor that looks for sentence medial capitalized terms if the script supports capitalization; otherwise just returns all terms.

- A Chinese foreign name extractor. This extractor uses a list of characters that are commonly used to transliterate foreign words in Chinese, and extracts sequences of at least three such characters.

- A Chinese personal name extractor. This uses a list of family names to find possible Chinese personal names.

- A *katakana* extractor, that extracts regions of *katakana* from Japanese text; *katakana* is com-



埃及总统穆巴拉克、叙利亚总统阿萨德和沙特阿拉伯国王法赫德
28日和29日在埃及亚历山大市举行首脑会议。

```
Egyptian President Hosni Mubarak, Syrian president Hafez al-Assad and
king Fahd of Saudi Arabia held a meeting in the northern Egyptian port
city of Alexandria just before the end of last year.
```

```
<?xml version="1.0" encoding="UTF-8"?>
<doclist>
 <doc>
   <lang id="zh">
     ...
     <token count="1" morphs=""
      prons="sr a th &amp; a l a p o ;
      s u n A DUM g u m A k u d A k u DUM">沙特阿拉伯</token>
     <token count="1" morphs=""
      prons="f a x &amp; t &amp; ; n o r i A k A i o s i e">法赫德</token>
     <token count="1" morphs=""
      prons="m u p a l a kh &amp; ;
      j a w A r A g u d o m o e k u d A k u g A ts u">穆巴拉克</token>
     <token count="1" morphs="" prons="a s a t &amp;
      ; k u m A DUM o s i e">阿萨德</token>
   </lang>
   <lang id="en">
     <token count="1" morphs="" prons="@ l &amp; g z @ n d r i: &amp;">Alexandria</token>
     <token count="1" morphs="" prons="&amp; r e I b i: &amp;">Arabia</token>
     <token count="1" morphs="" prons="&amp; s A: d">Assad</token>
     <token count="1" morphs="" prons="I dZ I p S &amp; n">Egyptian</token>
     <token count="1" morphs="" prons="f A: d">Fahd</token>
     ...
     <token count="1" morphs="" prons="m u b A: r I k">Mubarak</token>
     ...
     <token count="1" morphs="" prons="s &gt; d i:">Saudi</token>
     ...
   </lang>
 </doc>
</doclist>
```

Figure 1: Sample comparable texts and extracted XML document structure (including just the extracted names) for `ScriptTranscriber`.

monly used to transliterate foreign terms in Japanese.

- A Thai extractor. This uses a discriminative model (built using SNoW) to predict word boundaries in unsegmented Thai text, and then returns all found terms.

Users can easily define their own extractors so that, for example, if they have a good named entity extractor for a language, they can simply define an interface to that as a derived class of `Extractor`.

We also provide a **morphological analyzer** class, a placeholder for a range of possible morphological analyzers. The one provided looks for words that share common substrings and groups them into tentative morphological equivalence classes, along the lines of (Klementiev and Roth, 2006).

The **pronouncer** module provides a number of classes to convert Unicode strings into phonetic strings; the current version of the software uses WorldBet (Hieronymus, 1993), an ASCII implementation of the International Phonetic Alphabet (IPA). There are three specializations of the **pronouncer** module provided:

- *Unitran* (Yoon et al., 2007), which provides guesses on pronunciations for most grapheme code points in the Unicode Basic Multilingual Plane that are also used as scripts for languages. (For example, the IPA code points are *not* covered, since IPA is not used as the standard orthography for any language.) So, for example, Korean *hangul* 마 is given pronunciation *ma*, Cyrillic Ж is given pronunciation *Z*, and Japanese *katakana* マ is given pronunciation `mA`.
- English pronouncer: provides Festival-derived pronunciations (Taylor et al., 1998) for about 2.9 million words.
- Hanzi (Chinese character) pronouncer. Provides Chinese (Mandarin) and Native Japanese (*kunyomi*) pronunciations for characters. In some cases, there may be more than one Mandarin or *kunyomi* pronunciation for a given character. In such cases, the current implementation picks one pronunciation (i.e. one Chinese pronunciation and one *kunyomi* pronunciation, if there is a *kunyomi* pronunciation). For Chinese, in most cases the variant pronunciations are minor variants so that the choice of one pronunciation will not affect the phonetic comparison, and comparing one string is more efficient than comparing a lattice of possible transcriptions. For Japanese, the situation is certainly more complex, since there are multiple pronunciations for most characters, including both Sino-Japanese and native Japanese (*kunyomi*) pronunciations. `ScriptTranscriber`

provides one native and one Sino-Japanese pronunciation. The *kunyomi* module also computes *rendaku* so that for example 梅干 is pronounced as *umebosu* rather than *umehosu*.

While most of the pronunciation modules provided produce single pronunciations for a given string, the comparator module (below) will consider all possible pronunciations assigned to a string. Thus it is straightforward to incorporate multiple pronunciations, and it would also be straightforward to incorporate weighted pronunciations; one would merely need to define a comparator that makes use of pronunciation weights in its scoring.

The **comparator** module provides the cost for the mapping between strings. Three specializations are provided:

- Hand-built phonetic comparator, which uses the phonetic distance method of (Tao et al., 2006; Yoon et al., 2007).

- Perceptron-based comparator. This uses a perceptron string-to-string transliteration model trained on a dictionary of transliteration pairs, following (Klementiev and Roth, 2006). The particular model provided with `ScriptTranscriber` is based on a 71,548 entry English/Chinese name lexicon from the Linguistic Data Consortium (`http://www.ldc.upenn.edu`), but the implementation (which uses PySNoW (Carlson et al., 1999)) is of course language-pair independent. It would be straightforward to incorporate other learners, such as Winnow, which are provided with the SNoW toolkit.

- Time correlation comparator. For each *doc*, and for each *lang* in the *doc*, we pair each extracted term with the extracted terms in all the other *lang*s in the *doc*. Those pairs for which the phonetic match score is below some threshold can be removed at this stage. We compute similar pairs for each of the *doc*s in the corpus. Then for each pair, we compute the term-relative frequencies across the entire corpus and, following (Sproat et al., 2006), we compute the Pearson correlation co-efficient of these relative frequency values.

`ScriptTranscriber` thus provides general methods to get a baseline system up and running quickly for any pair of languages. Clearly, for any given pair of languages, more specialized methods — e.g. segmenters for languages such as Thai or Japanese, a trained morphological analyzer, more finely tuned pronunciation models —will produce better results. `ScriptTranscriber` makes it easy to incorporate such methods. Furthermore, it is hoped that since `ScriptTranscriber` is in the public domain, people will be motivated to add specialized methods to the toolkit.

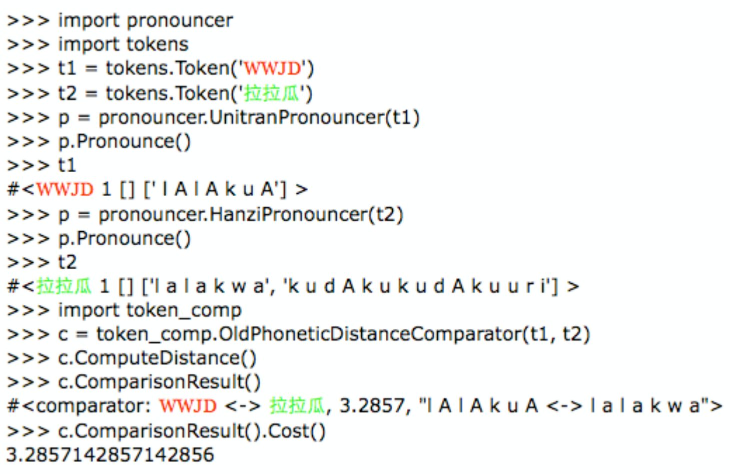
```
#!/bin/env python
# -*- coding: utf-8 -*-

"""Sample transliteration extractor based on the LCTL Thai parallel
data. Also tests Thai prons and alignment.
"""

__author__ = """
xxx@yyyy.zzz (Xxxxx Yyyyyyy)
"""

import sys
import os
import documents
import tokens
import token_comp
import extractor
import thai_extractor
import pronouncer
from __init__ import BASE_

## A sample of 10,000 from each:

ENGLISH_       = '%s/testdata/thai_test_eng.txt' % BASE_
THAI_          = '%s/testdata/thai_test_thai.txt' % BASE_
XML_FILE_      = '%s/testdata/thai_test.xml' % BASE_
MATCH_FILE_    = '%s/testdata/thai_test.matches' % BASE_
BAD_COST_      = 6.0

def LoadData():
  t_extr = thai_extractor.ThaiExtractor()
  e_extr = extractor.NameExtractor()
  doclist = documents.Doclist()
  doc = documents.Doc()
  doclist.AddDoc(doc)
  #### Thai
  lang = tokens.Lang()
  lang.SetId('th')
  doc.AddLang(lang)
  t_extr.FileExtract(THAI_)
  lang.SetTokens(t_extr.Tokens())
  lang.CompactTokens()
  for t in lang.Tokens():
    pronouncer_ = pronouncer.UnitranPronouncer(t)
    pronouncer_.Pronounce()
  #### English
  lang = tokens.Lang()
  lang.SetId('en')
  doc.AddLang(lang)
  e_extr.FileExtract(ENGLISH_)
  lang.SetTokens(e_extr.Tokens())
  lang.CompactTokens()
  for t in lang.Tokens():
    pronouncer_ = pronouncer.EnglishPronouncer(t)
    pronouncer_.Pronounce()
  return doclist


def ComputePhoneMatches(doclist):
  matches = {}
  for doc in doclist.Docs():
    lang1 = doc.Langs()[0]
    lang2 = doc.Langs()[1]
    for t1 in lang1.Tokens():
      hash1 = t1.EncodeForHash()
      for t2 in lang2.Tokens():
        hash2 = t2.EncodeForHash()
        try: result = matches[(hash1, hash2)] ## don't re-calc
        except KeyError:
          comparator = token_comp.OldPhoneticDistanceComparator(t1,  t2)
          comparator.ComputeDistance()
          result = comparator.ComparisonResult()
          matches[(hash1, hash2)] = result
  values = matches.values()
  values.sort(lambda x, y: cmp(x.Cost(), y.Cost()))
  p = open(MATCH_FILE_, 'w') ## zero out the file
  p.close()
  for v in values:
    if v.Cost() > BAD_COST_: break
    v.Print(MATCH_FILE_, 'a')


if __name__ == '__main__':
  doclist = LoadData()
  doclist.XmlDump(XML_FILE_, utf8 = True)
  ComputePhoneMatches(doclist)
```

Figure 2: Sample use of `ScriptTranscriber`. This program computes matches between English and Thai given a sample comparable English-Thai corpus.

3. Sample Use

A sample use of the program is given in Figure 2. This program loads some Thai and English data from the distributed `testdata` directory, extracts terms from each, builds and dumps an XML document representation, and computes phonetic distances for each pair of terms in each document, dumping a best-first sorted list of matches to a file.

Figure 3 shows a sample interactive use of the tools. Here we compute the phonetic distance between the same (nonsense) word *lalagua* transcribed in Chinese



```
>>> import pronouncer
>>> import tokens
>>> t1 = tokens.Token('WWJD')
>>> t2 = tokens.Token('拉拉瓜')
>>> p = pronouncer.UnitranPronouncer(t1)
>>> p.Pronounce()
>>> t1
#<WWJD 1 [] [' l A l A k u A'] >
>>> p = pronouncer.HanziPronouncer(t2)
>>> p.Pronounce()
>>> t2
#<拉拉瓜 1 [] ['l a l a k w a', 'k u d A k u k u d A k u u r i'] >
>>> import token_comp
>>> c = token_comp.OldPhoneticDistanceComparator(t1, t2)
>>> c.ComputeDistance()
>>> c.ComparisonResult()
#<comparator: WWJD <-> 拉拉瓜, 3.2857, "l A l A k u A <-> l a l a k w a">
>>> c.ComparisonResult().Cost()
3.2857142857142856
```

Figure 3: Interactive use of the `ScriptTranscriber` tools. (Note that '>>>' is the standard Python prompt. System responses are indented to the left margin. The two script examples are Cherokee and Hanzi.) The Hanzi pronouncer produces one Chinese and one Native Japanese pronunciation guess for the string. It is the Chinese one — *lalakwa* — that will match with the Cherokee example.

and in Cherokee.

4. Performance

`ScriptTranscriber`, since it is written in Python is not blindingly fast. To give a sense of the speed we computed comparisons between 10,000 Chinese and English parallel sentences from the ISI Chinese-English Automatically Extracted Parallel Text Corpus (`http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2007T09`). On an Intel Pentium 1.80GHz Dual CPU with 2G of memory, it takes about 18 seconds to load the sentences, parse the sentences into documents, and run the Chinese and English extractors. It takes an additional 22 seconds to compute 16,700 matches (760 matches per second) using the phonetic distance comparator of (Tao et al., 2006; Yoon et al., 2007) between English and Chinese potential transliterations, and extract a total of 320 matches that were above threshold. The top 30 strongest matches from this corpus are given in Table 1.

5. Summary

This short paper described `ScriptTranscriber`, an open source Python toolkit for extracting transliteration pairs from comparable corpora in languages that use different scripts. It works with any script in the Unicode Basic Multilingual Plane. The object-oriented design of `ScriptTranscriber` means that it is easy to extend to incorporate other more sophisticated models. `ScriptTranscriber` is available as part of the nltk_contrib source tree at `http://code.google.com/p/nltk/`

6. Acknowledgments

Work reported here was partially funded by NBCHC040176 from the US Department of the Interior, a Google Research Award, and the National Science Foundation under grant #0705708 to the Center for Language and Speech Processing at tne Johns Hopkins University.

7. References

Andrew Carlson, Chad Cumby, Je L. Rosen, and Dan Roth. 1999. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC CS Dept.

Jim Hieronymus. 1993. Ascii phonetic symbols for the world's languages: Worldbet.

Alexandre Klementiev and Dan Roth. 2006. Weakly supervised named entity transliteration and discovery from multilingual comparable corpora. In *Proceedings of COLING-ACL 2006*, Sydney, Australia, July.

Edward Loper and Steven Bird. 2002. Nltk: the natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics*, pages 63–70.

Richard Sproat, Tao Tao, and ChengXiang Zhai. 2006. Named entity transliteration with comparable corpora. In *Proceedings of COLING-ACL 2006*, Sydney, July.

Tao Tao, Su-Youn Yoon, Andrew Fister, Richard Sproat, and ChengXiang Zhai. 2006. Unsupervised named entity transliteration using temporal and phonetic correlation. In *EMNLP 2006*, Sydney, July.

Paul Taylor, Alan Black, and Richard Caley. 1998. The architecture of the Festival speech synthesis system. In *Proceedings of the Third ESCA Workshop on Speech Synthesis*, pages 147–151, Jenolan Caves, Australia.

Su-youn Yoon, Kyoung-young Kim, and Richard Sproat. 2007. Multilingual transliteration using feature based phonetic method. In *ACL*.

Chinese	English	Match score	Chinese Pron	English Pron
卡杜米	Kaddoumi	2.33	kh a t u m i	k @ d u m i:
马尼拉	Manila	2.42	m a n i l a	m & n I l &
卢萨卡	Lusaka	2.42	l u s a kh a	l u s A: k &
夸祖鲁	Kwazulu	2.43	kh w a ts u l u	k w A: z u l u
黎巴嫩	Lebanon	2.50	l i p a n & n	l E b & n & n
希拉里	Hillary	2.58	C i l a l i	h I l & r i:
迈克尔	Michael	2.58	m a i kh & &r	m a I k & l
卡杜纳	Kaduna	2.67	kh a t u n a	k A: d u n &
巴拿马	Panama	2.75	p a n a m a	p @ n & m A:
曼德拉	Mandela	2.79	m a n t & l a	m @ n d E l &
德拉卡马	Dhlakama	2.81	t & l a kh a m a	d & l & k A: m &
突尼斯	Tunisia	2.92	th u n i s i	t u n i: Z &
科威特	Kuwaitis	2.93	kh & w e i th &	k u w e I t i: z
巴拉圭	Paraguay	3.00	p a l a k w e i	p E r & g w e I
卡苏莱德	Cassoulides	3.00	kh a s u l a i t &	k @ s u l a I d z
塔利班	Taliban	3.07	th a l i p a n	t @ l & b & n
科威特	Kuwaiti	3.14	kh & w e i th &	k u w e I t i:
霍梅尼	Khomeini	3.19	x w o m e i n i	k o U m e I n i:
阿莱萨纳	Alesana	3.19	a l a i s a n a	A: l e I s @ n &
萨利纳斯	Salinas	3.19	s a l i n a s i	s & l i: n & s
巴基斯坦	Pakistan	3.22	p a cC i s i th a n	p @ k I s t @ n
布特莱齐	Buthelezi	3.28	p u th & l a i cCh i	b u t & l e I z i:
马普托	Maputo	3.29	m a ph u th w o	m & p u t o U
伊拉克	Iraqis	3.30	i l a kh &	I r @ k i: z
曼德勒	Mandalay	3.38	m a n t & l &	m @ n d & l e I
纳米比亚	Namibia	3.38	n a m i p i j a	n & m I b i: &
普林西比	Principe	3.39	ph u l i n C i p i	p r i: n tS i: p i:
布达拉	Potala	3.42	p u t a l a	p A: t A: l &
卡拉奇	Karachi	3.42	kh a l a cCh i	k A: r A: tS i:
英格兰	England	3.43	i N k & l a n	I N g l & n d
坦塔维	Tantawi	3.44	th a n th a w e i	t @ n t A: w i:

Table 1: Top 30 matches from sample of 10,000 Chinese/English parallel sentences from the ISI Chinese-English Automatically Extracted Parallel Text Corpus. Pronunciations are in WorldBet. Only *Kuwaitis*, *Kuwaiti* and *Iraqis* are technically wrong: the Chinese equivalents are for *Kuwait* and *Iraq*.