Automatically Merging Lexicons that have Incompatible Part-of-Speech Categories

Daniel Ka-Leung CHAN and **Dekai WU**
HKUST
Human Language Technology Center
University of Science and Technology
Clear Water Bay, Hong Kong
{dklchan, dekai}@cs.ust.hk

Abstract

We present a new method to automatically merge lexicons that employ different incompatible POS categories. Such incompatibilities have hindered efforts to combine lexicons to maximize coverage with reasonable human effort. Given an "original lexicon", our method is able to merge lexemes from an "additional lexicon" into the original lexicon, converting lexemes from the additional lexicon with about 89% precision. This level of precision is achieved with the aid of a device we introduce called an *anti-lexicon*, which neatly summarizes all the essential information we need about the co-occurrence of tags and lemmas. Our model is intuitive, fast, easy to implement, and does not require heavy computational resources nor training corpus.

1 Motivation

We present a new, accurate method to automatically merge lexicons that contain incompatible POS categories. In this paper, we look specifically at the problem that different lexicons employ their own part-of-speech (POS) tagsets that are incompatible with each other, owing to their different linguistic backgrounds, application domains, and/or lexical acquisition methods.

Consider the way that lemmas are typically marked with POS information in machine-readable lexicons. For example, here are a few entries from the lexicon in Brill's tagger (Brill, 1994) and the Moby lexicon (Ward, 1996), showing simple pairs of lemmas and POS tags:

lemma	**tag**
apple	NN
boy	NN
calculate	VB

Example entries in Brill lexicon

lemma	**tag**
boy	N
hold	V

Example entries in Moby lexicon

Perhaps the most natural first approach to merging the lexicons is to construct a set of POS mapping rules.

For example, we might wish to acquire the following mapping rules:

$$\langle \text{ "NN", "N" } \rangle, \langle \text{ "VB", "V" } \rangle \ldots$$

Here, the first rule says that the "NN" POS in the Brill lexicon should be mapped to the "N" POS in the Moby lexicon. Of course, not all POS tags can be accurately translated this way, but the strategy is a reasonable first approximation.

In order to incorporate entries from other lexicons into the current knowledge base, the mapping rules between different POS tagsets are usually formulated by hand in *ad hoc* ways. In view of this heterogeneity and human subjectiveness, some people had begun to investigate and develop methods of learning the mapping rules between different POS categories in different lexicons. Teufel described a tool to support manual mapping between different tagsets using a rule-based approach (Teufel, 1995). This approach requires heavy human intervention, and therefore does not scale up easily. Another approach was proposed by Hughes *et al.*, to automatically extract mapping rules from corpora tagged with more than one annotation scheme (Hughes et al., 1995). However, the dependence on multiply-annotated corpora requires heavy annotation and/or computation resources, whereas we are investigating methods with only the information found in existing lexicons.

In this paper, we will begin by presenting a basic method that generates a set of mapping rules. Experimental results on a variety of lexicons will be presented. We will then introduce a mechanism called an "anti-lexicon" that significantly improves precision on the learned rules and merged lexicons, though at a cost to recall.

2 Basics

Our general strategy is to inspect the co-occurrence of tags on those lemmas that are found in both lexicons, and to use that information as a basis for generalizing, thus yielding POS mapping rules. To do this requires several steps, as described in the following subsections. As a preliminary step, we will introduce a way to represent POS tags using feature vectors. We then use these vectors to generate mapping rules. To obtain better accuracy, we can restrict the training examples to entries that occur in both lexicons. The generation algorithm also requires us to define a similarity metric between POS feature vectors.

2.1 Part-of-speech feature vector

A necessary preliminary step of our method is to introduce *POS feature vectors*. A feature vector is a useful representation of a POS tag, because it neatly summarizes all the information we need about which lemmas can and cannot have that POS tag, illustrated as follows.
Given:

- a lemma set $\mathcal{M}$ ={ "apple", "boy", "calculate" }
- a set of POS tags $\mathcal{P}$ ={"NN","VB"}

A tiny example lexicon consisting of lemma and POS tag pairs might be as follows, where each cell with • indicates the existence of that lemma-POS pair in the lexicon:

	apple	boy	calculate
NN	•	•	
VB			•

which, when represented as POS feature vectors, will be:

$$P^1: < \quad 1, \quad 1, \quad 0, \quad >$$
$$P^2: < \quad 0, \quad 0 \quad 1, \quad >$$

where P^1 here is the "NN" POS represented by the set of words that can be nouns in a given lexicon, in this example { "apple", "boy" } and P^2 similarly is the "VB" POS.

The feature value for feature f in $\vec{p}$ can be either:

- 0 to indicate that we are not sure whether p is a tag of f;
- 1 to indicate that p is a tag for f;
- 2 to indicate that p can never be a tag for lemma f.

Obtaining information about the last of these (the value 2) is a non-trivial problem, which we will return to later in this paper. With ordinary lexicons, we only directly obtain feature vectors containing 0 and 1 values.

2.2 Mapping rule learning algorithm

Given a feature vector for every POS tag in both lexicons—say, Brill's lexicon and the Moby lexicon—we use the following algorithm to learn mapping rules from POS tags in Brill's tagset to POS tags in the Moby tagset. The idea is to assume that a mapping rule between two POS tags holds if the similarity between their feature vectors exceeds a preset threshold, called a *sim-threshold* τ. The similarity metric (SimScore) will be described later, but let's first look at the learning algorithm, as described in algorithm 1.

This algorithm does not exclude m-to-n mappings; that is, any Brill POS tag could in principle get mapped to any number of Moby POS tags.

mapper$(\mathcal{P},\mathcal{Q})$

input : Two sets of feature vectors of POS tags
$\mathcal{P} = \{\vec{p^1}, \vec{p^2}, \ldots, \vec{p^n}\}$
$\mathcal{Q} = \{\vec{q^1}, \vec{q^2}, \ldots, \vec{q^m}\}$

output : A mapping table represented as a set of pairs of feature vectors
$\mathcal{B} = \{< \vec{p}, \vec{q} >, \ldots\}$

algorithm:

```
foreach p^i in P do
    foreach q^j ∈ Q do
        if SimScore(p^i, p^j) > sim_threshold τ then
            B ← B ∪ {< p^i, q^j >};
        end
    end
end
```

Algorithm 1: Mapping rule learning algorithm

2.3 Improving the training set by intersecting the lexicons

We can obtain better results by considering only those lemmas that occur in both lexicons. This has the effect of eliminating unreliable features in the POS feature vectors, since lemmas that do not occur in both lexicons cannot be relied upon when judging similarity. This results in pruned versions of both lexicons.

For example, pretend that the following are the *only* entries in the Brill and Moby lexicons:

lemma	tag
apple	NN
boy	NN
calculate	VB

Brill lexicon

lemma	tag
boy	N
hold	V

Moby lexicon

In this case, intersecting the lexicons would result in the following pruned lexicons:

lemma	tag
boy	NN

Brill' lexicon

lemma	tag
boy	N

Moby' lexicon

After pruning, the only remaining lemma is "boy", and the new POS feature vectors for "NN" and "N" have just one dimension corresponding to "boy":

$$\text{NN:} < 1 >$$
$$\text{N:} < 1 >$$

Of course, in reality the lexicons are much bigger and the effect is not so drastic.

In all experiments in this paper, we used lexicon intersection to prune the lexicons.

2.4 Similarity metric

The similarity function we use calculates a similarity score between two feature vectors by counting the number of features with the same feature value 1 or 2, indicating that a lemma either can or cannot belong to that POS category. (Recall that the value

0 means “don’t know”, so we simply ignore any features with value 0.) The score is normalized by the length of the feature vector. We also require that there be at least one positive match in the sense that some lemma is shared by both of the POS categories; otherwise, if there are only negative matches (i.e., lemmas that cannot belong to either POS category), we consider the evidence to be too weak and the similarity score is then defined to be zero. The whole algorithm is described in algorithm 2.

2.5 The “complete lexicon” assumption

As mentioned earlier, ordinary lexicons do not explicitly contain information about which parts of speech a lemma can *not* be used as. We have two choices. In the examples up till now, we used a value of 0 for any lemma-tag pair not explicitly listed in the lexicon, signifying that we don’t know whether the POS category can include that lemma. However, having many “don’t know” values significantly weakens our similarity scoring method. Alternatively, we can choose to assume that our lexicons are complete—a kind of closed world assumption. In this case, we assume that any lemma-tag pair not found in the lexicon is not merely an omission, but really can never occur. This means we use the value 2 instead of the value 0.

The “complete lexicon” assumption only makes sense when we are dealing with large, broad coverage lexicons (as is the case in this paper). It is not reasonable when dealing with small or specialized sublexicons.

3 Anti-lexicon

Based on the above intuition on utilizing negative information, we propose an improved model using something we call an *anti-lexicon*, that indicates the POS tags that a lemma *cannot* have, which we will call its *anti-tags*. A POS tag p is called an anti-tag a of a lemma m if p can never be a tag of m.

The anti-lexicon consists of a set of pieces of this negative information, each called an *anti-lexeme* $\neg l$:

$$\neg l \stackrel{\text{def}}{=} (m, \neg p)$$

where $\neg p$ is the anti-tag of lemma m, and p is a POS used in the lexicon.

Some examples of anti-lexemes are:

$$\langle \text{ happy}, \neg\text{IN } \rangle$$
$$\langle \text{ run}, \neg\text{JJ } \rangle$$
$$\langle \text{ in}, \neg\text{VB } \rangle$$

where “IN”,”JJ” and “VB” are the preposition, adjective and verb tags in Brill lexicon respectively.

Similar to a traditional lexicon which contains lexemes in the form of pairs of lemmas and their corresponding possible POS tag(s), an anti-lexicon contains *anti-lexemes* which are simple pairs that associate a lemma with an anti-tag.

The anti-lexicon can be automatically generated quickly and easily. To illustrate the idea, consider an example lexicon where we add the lemma “Central” and the POS “NP” to the example lexicon we have been working with.

	apple	boy	calculate	Central
NN	•	•		
VB			•	
NP	•			•

Suppose we want to know whether “Central” can be a “NN”, and whether “calculate” can be a “NN”. The fact that “apple” can be tagged by both “NN” and “NP”, but not “VB”, gives “Central” (a lemma that is already known to be able to serve as an “NP”) a higher likelihood of possibly serving as an “NN” than “calculate” (a lemma that is not known to be able to serve as an “NP”).

Based on this assumption that lexemes with similar semantics will have similar POS tags, we conceptualize this kind of pattern in terms of “cohesion” between lemmas and POS tags in a lexicon. The “cohesion” of a lemma l and a POS tag p measures the likelihood of a POS tag p being a possible tag of a lemma l, and is defined as:

SimScore($\vec{p}$, $\vec{q}$)

input : Two POS feature vectors of integers with values $\{0, 1, 2\}$
(representing POS tags that may come from different tagsets)
$\vec{p} = \{p_1, p_2, \ldots, p_n\}$
$\vec{q} = \{q_1, q_2, \ldots, q_n\}$

output : A score in the range $(0, 1)$ representing the similarity between $\vec{p}$ and $\vec{q}$

algorithm:

```
num_agree ← 0
num_known ← 0
all_negative_agree ← true

foreach i from 1 to n do
    if p_i ≠ 0 and q_i ≠ 0 then
        num_known ← num_known + 1
        if p_i = q_i then
            num_agree ← num_agree + 1
            if p_i = 1 then
                all_negative_agree ← false
            end
        end
    end
end
if all_negative_agree then
    return 0
end
else
    return num_agree ÷ num_known
end
```

Algorithm 2: Similarity scoring algorithm

$$cohesion(l, p) = \begin{cases} 1, & \text{if } \langle l, p \rangle \text{ in lexicon;} \\ \Pr(p|p_1, p_2, \ldots., p_n), & \text{if } F(p_1, p_2, \ldots, p_n) > 0; \\ 0, & \text{otherwise} \end{cases}$$

where

$$Pr(p|p_1, p_2, \ldots., p_n) = \frac{F(p, p_1, p_2, \ldots, p_n)}{F(p_1, p_2, \ldots, p_n)}$$

which $F(p_1, p_2, \ldots, p_n)$ denotes the total number of lemmas in the lexicon for which $p_1, p_2, \ldots, p_n$ are all legal POS tags of l, and $Pr(p|p_1, p_2, \ldots, p_n)$ is just a simple relative frequency of the lemmas which have the set of POS $\{p, p_1, p_2, \ldots\ldots, p_n\}$ to the the lemmas with $\{p_1, p_2, \ldots, p_n\}$.

In the last example,

$$\begin{aligned} &cohesion(\text{“boy”}, \text{“NP”}) &&= 0.5 \\ &cohesion(\text{“calculate”}, \text{“NP”}) &&= 0 \end{aligned}$$

Therefore “NP” is more likely to be associated to “boy” than “calculate”, which implies “NP” will be less likely to be the valid POS to “boy” than to “calculate”.

Under this intuition, we create an anti-lexicon by considering the cohesion of all possible combinations of lemmas and POS tags. Entries with low cohesion will be considered as anti-lexemes and inserted into the anti-lexicon.

An anti-lexicon $\mathcal{A}$ is created by:

$$\mathcal{A} = \mathcal{A} \ \cup \ \langle l, a \rangle \ \text{ iff } cohesion(l, a) \leq \lambda$$

where λ is a threshold called *anti-threshold*, usually a very small real number between 0 and 1.

In our example, if we set anti-threshold to 0.6, "NP" will become an anti-tag for "calculate" but not for "boy". In actual lexicons, since a lemma usually have many possible POS tags, the value of anti-threshold is usually much lower.

4 Lexicon merging algorithm

Given a POS mapping table $\mathcal{B}$ between the POS tagset $\mathcal{P}$ used by the original lexicon $\mathcal{L}^q$ and the POS tagset $\mathcal{Q}$ used by the additional lexicon $\mathcal{L}^p$, we merge the entries from the additional lexicon into the original lexicon by an algorithm as shown in algorithm 3.

This algorithm does not exclude m-to-n POS mappings; that is, a lexeme in the additional lexicon can generate more than one lexeme and we can merge all of them into the original lexicon.

5 Experiment

5.1 Setup

We tested the above method in a set of experiments using four commonly-used machine-readable dictionaries. They are Brill's lexicon, the Moby lexicon, the Collins lexicon, the Oxford machine-readable dictionary, with characteristics as summarized in table 1. The lexicons use distinct POS tagsets of different tag granularities, as summarized in table 2.

lexicon	POS tagset tagset	number of lexemes
Brill	Penn TreeBank	105199
Collins	Collins tagset	100566
Moby	Moby tagset	250441
Oxford	Oxford tagset	84588

Table 1: Summary of English monolingual lexicons

With these four training lexicons we can test twelve pairwise lexicon merging tasks, as shown in table 3. For each pairs of lexicon combination, we intersect them by the strategy mentioned before and produced a new set of training lexicons in each task. Note that the trimmed down Brill lexicon in the "Brill-to-Collins" task is not the same as the trimmed down Brill lexicon in "Brill-to-Moby".

In order to evaluate the accuracy of our methods, we asked a linguist to manually create twelve "gold standard" sets of POS mapping rules, $\mathcal{R}$, one for each of the twelve pairwise lexicons on the semantics between the POS tag only. We then ran the experiments to automatically generate two sets of POS mapping tables, with one under the complete world assumption and another using an anti-lexicon in each merging task. We evaluated precision and recall on POS mapping rules as follows:

$$\text{precision on POS mapping rules} = \frac{\mid \mathcal{E}' \mid}{\mid \mathcal{E} \mid}$$

where

- $\mathcal{E}$ is the resulting tagset mapping table containing all mapping rules obtained from experiment;
- $\mathcal{E}'$ is the subset of $\mathcal{E}$ which contains all correct mapping rules in $\mathcal{R}$. ($\mathcal{E} \in \mathcal{R}$)

$$\text{recall on POS mapping rules} = \frac{\mid \mathcal{E}' \mid}{\mid \mathcal{R} \mid}$$

Using an anti-threshold $\lambda = 0.00001$, we created twelve anti-lexicons which can then be used in our algorithm. We obtained the POS mapping results as shown in table 4.

In the baseline model, the precision is very low, mainly due to data sparseness caused by the fact that machine readable lexicons usually do not contain full lexeme coverage. This means our "complete lexicon assumption" which says that we can interpret entries not being in the lexicon as "negative examples" is not correct.

tagset	granularity	size	example tags on noun, proper noun, adjective, verb
Penn TreeBank	fine	43	NN, NP, JJ, VV
Collins tagset	fine	32	n, n, adj, vb
Moby tagset	coarse	15	n, n, a, v
Oxford tagset	coarse	20	K, G, M, N

Table 2: Summary of original POS tagsets in lexicons

task	additional lexicon	size after lexicon intersection	original lexicon	size after lexicon intersection
bm	Brill	48097	Moby	47486
bc	Brill	29861	Collins	35933
bo	Brill	48154	Oxford	46952
cm	Collins	96149	Moby	90255
om	Oxford	50562	Moby	52508
co	Collins	42146	Oxford	33056
oc	Oxford	33056	Collins	42146
mo	Moby	52508	Oxford	50562
mc	Moby	90255	Collins	96149
ob	Oxford	46952	Brill	48154
cb	Collins	35933	Brill	29861
mb	Moby	47486	Brill	48097

Table 3: Size of trimmed lexicons after lexicon intersection

task	precision w/o anti-lexicon	precision w/anti-lexicon	recall w/o anti-lexicon	recall w/anti-lexicon
bm	0.1606	1.0000	0.4070	0.0349
bc	0.1399	1.0000	0.4409	0.0215
bo	0.1944	0.2727	0.6222	0.0667
cm	0.1419	0.7143	0.3929	0.1786
om	0.1811	1.0000	0.5897	0.0513
co	0.1358	0.5714	0.4314	0.0784
oc	0.1420	0.7500	0.5227	0.0682
mo	0.1811	0.6667	0.6216	0.1081
mc	0.1290	1.0000	0.4762	0.1904
ob	0.1979	0.3333	0.6333	0.0444
cb	0.1434	0.2500	0.4118	0.0392
mb	0.1651	0.3750	0.4932	0.0822
average	0.1594	0.6611	0.5036	0.0803

Table 4: Results for POS mapping rule learning

lexicon_insertor($\mathcal{L}^p$,$\mathcal{L}^q$)

input : 1. Two sets of lexemes, each lexeme in the form of a pair of lemma and POS.
. $\mathcal{L}^p = \{\langle m_i, p_j\rangle, \ldots\}$
. $\mathcal{L}^q = \{\langle m_k, q_l\rangle, \ldots\}$
2. A POS mapping table $\mathcal{B}$ from the POS tagset $\mathcal{P}$ to POS tagset $\mathcal{Q}$
. $\mathcal{B} = \{\langle p_x, q_y\rangle \ldots\}$

output : An enlarged set of lexemes in $\mathcal{L}^q$, which contains newly inserted lexemes converted from $\mathcal{L}^p$.
$\mathcal{L}^{q\prime} = \{\langle m_k, q_l\rangle, \ldots, \langle m_r, q_s\rangle, \ldots\}$
where $\langle p_j, q_s\rangle \in \mathcal{B}$, and $\langle m_r, q_s\rangle \notin \mathcal{L}^q$ for all k, l, p, q, r, s

algorithm:

foreach $\langle m_i, p_j\rangle$ in $\mathcal{L}^p$ **do**
  foreach $\langle p_j, q_s\rangle$ in $\mathcal{B}$ **do**
    if $\langle m_i, q_s\rangle$ not in $\mathcal{L}^q$ **then**
      $\mathcal{L}^q \leftarrow \mathcal{L}^q \cup \langle m_i, q_s\rangle\}$
    end
  end
end

Algorithm 3: Lexicon merging algorithm

In the anti-lexicon model, the precision greatly improves, with some experiments even achieving 100% precision. Unfortunately, the recall suffers sharply.

After automatically constructing the POS mapping tables from training, we proceeded to merge lexicons in each testing task using the lexicon merging algorithm described above, and evaluated the accuracy of the merged lexicons as follow.

In each merging task, we randomly selected 100 lexemes from the additional lexicon. Given these 100 lexemes, a linguist first manually constructs a set of correctly converted lexemes, which will be used as the "gold standard" set of lexemes, $\mathcal{R}^L$. Similar to the evaluation criteria outlined for POS mappings, we define the precision and recall on lexicon merging as the following:

$$\text{precision on lexicon merging} = \frac{\mid \mathcal{E}^{L\prime} \mid}{\mid \mathcal{E}^{L} \mid}$$

where

- $\mathcal{E}^L$ is the set of lexemes generated by the lexicon insertor.
- $\mathcal{E}^{L\prime}$ is the subset of $\mathcal{E}^{\mathcal{L}}$ that contains all lexemes in $\mathcal{R}^L$.

$$\text{recall on lexicon merging} = \frac{\mid \mathcal{E}^{L\prime} \mid}{\mid \mathcal{R}^{L} \mid}$$

5.2 Results

We obtain the results on lexicon merging as shown at table 5.

The anti-lexicon model significantly improves the precision in both the generated POS mapping rules and merged lexicons. Most of the 12 lexicon merging tasks achieve nearly more than 92% precision, which cannot be obtained by using even the gold standard mapping rules, as shown in table 6.

The recall degradation using anti-lexicon is lower in lexicon merging than in POS mapping rule learning, owing to the fact that not all POS tags appear in lexicons with

task	precision w/o anti-lexicon	precision w/anti-lexicon	recall w/o anti-lexicon	recall w/anti-lexicon
bm	0.2263	0.9841	0.3410	0.2857
bc	0.1111	0.9429	0.3789	0.1737
bo	0.1147	0.8800	0.6356	0.1864
cm	0.2758	0.8276	0.7333	0.1778
om	0.1413	1.0000	0.4514	0.3194
co	0.1355	0.8667	0.3592	0.1262
oc	0.1625	1.0000	0.6193	0.2081
mo	0.1233	0.6000	0.4907	0.2222
mc	0.1813	0.9899	0.6946	0.5868
ob	0.0592	0.7692	0.5315	0.0699
cb	0.1114	0.9286	0.3546	0.0922
mb	0.1029	0.9615	0.3165	0.1582
average	0.1454	0.8959	0.4922	0.2172

Table 5: Results for lexicon merging

same frequency. For example, nouns and verbs occur far more frequently than prepositions and adverbs. High recall in POS mapping rules will not necessarily yield more accurate converted lexemes, if all the mapping rules obtained are only those rarely-occurring POS tags. Conversely, the successful generation of a single correct mapping rule for a frequently-occuring POS tag greatly improves recall. The mapping rules generated by our anti-lexicon model confirm this assumption: recall for POS mapping rules is 8%, but for lexicon merging it improves to about 22%.

Recall suffers sharply, but precision is more important than recall in lexicon merging. This is because the cost of post-lexicon clean up on lexemes with incorrect POS tag in a lexicon after merging is very expensive. A set of high precision POS mapping rules guarantees a much cleaner resulting lexicon after merging. Thus during lexicon merging, a conservative algorithm, which generates fewer but more exact lexemes is preferable.

To show how anti-lexicons affect the precision and recall on lexicon merging, we also ran experiments using different combinations of sim-thresholds and anti-thresholds. In most cases, the precision of lexicon merging obtained from anti-lexicon are much higher than those without. The results are summarized in table 7 and table 8. The best precision for lexicon merging is obtained from $\tau = 0.8$ and $\lambda = 0.00001$ in a grid-search.

5.3 Discussion

As mentioned earlier, the mapping rule learning algorithm we used permits m-to-n mappings so long as the mapping rules created for every tag in a lexicon reach the sim-threshold, that is, the confidence level specified by the lexicographer. An alternative approach that we are experimenting with is to allow only m-to-1 mappings, by simply choosing the mapping rule with highest similarity score. In theory, this would seem to limit the possible accuracy of the algorithm, but empirically we have found that this approach often yields higher precision and recall. Further investigation is needed.

Different similarity scoring functions can also be used. If data sparseness is a serious problem, we can use a similarity score which counts only the lemmas which are tagged, but not the lemmas which are not tagged. One effect of ignoring unlikely tags in this way is that the need for an anti-lexicon is

τ	baseline	$\lambda = 0.1$	0.001	0.0001	0.00001
0.5	0.1159	0.0803	0.2294	0.7068	0.7189
0.6	0.1178	0.1316	0.5392	0.7981	0.8094
0.7	0.1208	0.1166	0.5747	0.8711	0.8832
0.8	0.1454	0.0918	0.6444	0.8945	0.8959
0.9	0.1636	0.0980	0.5475	0.5454	0.5457
0.99	0.2450	0.2171	0.1488	0.2449	0.1408

Table 7: Average precision on lexicon merging using different sim-thresholds τ and anti-thresholds λ

τ	baseline	$\lambda = 0.1$	0.001	0.0001	0.00001
0.5	0.8082	0.5591	0.4152	0.4063	0.4181
0.6	0.7512	0.4493	0.3475	0.3383	0.3501
0.7	0.6318	0.0918	0.6444	0.2745	0.2864
0.8	0.4922	0.2152	0.1847	0.2092	0.2172
0.9	0.2404	0.1090	0.0884	0.0973	0.0973
0.99	0.0458	0.0341	0.0135	0.0458	0.0366

Table 8: Average recall on lexicon merging using different sim-thresholds τ and anti-thresholds λ

eliminated. We are also currently investigating the mapping power of such variant methods.

In general, we have observed different behaviors depending on factors such as the granularity of the tagsets, the linguistic theories behind the tagsets, and the coverage of the lexicons.

Finally, in addition to lexicon merging, POS mapping table is also useful in other applications. Wu and Wong apply them in their SITG channel model to give better performance in their translation application (Wu and Wong, 1998).

There is a serious problem of low recall on our anti-lexicon model. This is because our model prunes out many possible POS mapping rules which results in very conservative lexeme selection during the lexicon merging process. Moreover, our model cannot discover which POS tags in original lexicon have no corresponding tag in the additional lexicon.

Our model took POS mapping rules as a natural starting point since this representation has been used in earlier related work. However, our experiments showing low precision on lexicon merging even using the human-generated gold standard mapping rules indicates it might not be a good approach to use POS mapping rules at all to tackle the lexicon merging problems. Our next step will be to investigate models that are not constrained by the POS mapping rule representation.

6 Conclusion

We present a new method to automatically merge lexicons that employ different incompatible POS categories, which merges lexemes from an additional lexicon into an original lexicon with 89% in average precision. We showed how precision in the final merged lexicon can be improved by introducing a model called *anti-lexicon*, which neatly summarizes all the essential information we need about the co-occurrence of tags and lemmas. Our model is intuitive, fast, easy to implement, and does not require heavy computational resources nor training corpus.

7 Acknowledgments

Many thanks to Josephine Kwan and Harriet Lee for their help on hand-crafting the gold standard POS mapping rules and lexeme sets in the evaluation phrase. We also thank the SILC members, Aboy Wong, Hong-sing Wong, Vincent Chow, James Pang for their rigorous constructive criticisms on our model and experiments.

References

Eric Brill. 1994. Some Advances in Transformation-Based Part of Speech Tagging. In *Twelfth National Conference on Artificial Intelligence (AAAI-94)*.

Edward Fox, R. Wohlwend, P. Sheldon, Q. Chen, , and R. France. 1986. Coder Lexicon: The Collins English Dictionary and its Adverb Definitions. Technical Report TR-86-23, Department of Computer and Information Science, Virginia Tech, Oct.

A. S. Hornby. 1995. Oxford Machine Readable Dictionary http://info.ox.ac.uk/. ˜ archive/ota.html.

John Hughes, Clive Souter, and E. Atwell. 1995. Automatic Extraction of Tagset Mappings from Parallel-Annotated Corpora. *Computation and Language*.

task	precision	recall
bm	0.6953	0.8203
bc	0.5081	0.8263
bo	0.3478	0.8136
cm	0.3697	0.9037
om	0.4006	0.9236
co	0.3103	0.9612
oc	0.4804	0.9340
mo	0.3160	0.9537
mc	0.3996	0.9162
ob	0.1590	0.8671
cb	0.2272	0.9007
mb	0.2157	0.8861
average	0.3664	0.8922

Table 6: Lexicon merging results using gold standard POS mapping rules

Beatrice Santorini. 1990. Part-of-speech Tagging Guidelines for the Penn Treebank Project. Technical Report MS-CIS-90-47, Department of Computer and Information Science, University of Pennsylvania.

Simone Teufel. 1995. A Support Tool for Tagset Mapping. In *EACL-Sigdat 95*.

Grady Ward. 1996. Moby Lexicon. http://www.dcs.shef.ac.uk/research/ilash /Moby/.

Dekai Wu and H. S. Wong. 1998. Machine Translation with a Stochastic Grammatical Channel. In *Coling-ACL98*, pages 1408–1415.