PYCOT: An Optimality Theory-based Pronoun Resolution Toolkit

Whitney Gegg-Harrison*, Donna K. Byron†

*Department of Linguistics
The Ohio State University
1712 Neil Avenue
Columbus, OH 43210-1298
whitney@ling.ohio-state.edu

† Department of Computer Science and Engineering
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210-1277
dbyron@cse.ohio-state.edu

Abstract

In this paper, we present PYCOT, a pronoun resolution toolkit. This toolkit is written in the Python programming language and is intended to be an addition to the open-source NLTK collection of natural language processing tools. We discuss the design of the module as well as studies of its performance on pronoun resolution in English and in Korean.

1. Introduction

Pronoun resolution is an important subtask within many language processing applications, including semantic interpretation, translation, question answering, etc. Although there are a number of widely-known techniques for performing pronoun resolution (Brennan et al., 1987; Tetreault, 2001; Kennedy and Boguraev, 1996; Hobbs, 1986), there has been a lack of shared pronoun resolution software available for those researchers who do not wish to implement a pronoun resolution component themselves. This paper presents a pronoun resolution toolkit, called PYCOT, which can be used either as an off-the-shelf pronoun resolution module, or can be easily extended and modified to implement different pronoun resolution algorithms, or even a wider range of anaphora resolution tasks. The implementation is written in the Python program language, which allows it to be embedded within the processing pipeline of the NLTK open-source collection of natural language tools[1].

After examining other available anaphora resolution components (GuiTAR[2], Lingpipe[3]), RAP (Qiu et al., 2004), we felt there was a need for a well documented, modular pronoun resolution toolkit which allowed the user to easily modify the behavior of the resolution algorithm, and which could also be adapted for use with different languages or data formats without requiring the developer to alter significant amounts of code. The modular nature of PYCOT allows these customizations, when necessary, to be isolated to only certain functions within the overall program.

A novelty in this toolkit, compared to other anaphora resolution components, is that it is constructed using an Optimality Theory design, in which a collection of separate constraints are evaluated to judge pronoun resolution preferences. The basic module that we provide implements the Centering pronoun resolution algorithm from (Brennan et al., 1987), as re-formulated into Optimality Theory by (Beaver, 2004). As we report in (Byron and Gegg-Harrison, 2004), the modular, declarative style of algorithm implementation has several benefits, and seems particularly suited towards a flexible pronoun resolution module that can be easily tailored to work across languages and input formats. The module has been used by our team to perform pronoun resolution on both English and Korean bracketed trees (Byron and Gegg-Harrison, 2004). Although these two languages are quite different, we found the OT-inspired code very simple to port from English to Korean.

2. Overall Goal

Our goal in creating this software is to distribute a working pronoun resolution module that could be used as-is by developers who need access to an off-the-shelf pronoun resolution component, and that could also be easily modified be those researchers who want to change the pronoun resolution algorithm implemented in the module, for purposes of experimentation or system development. To achieve our goal of creating a flexible, user-friendly system, we chose to implement our module using Python, a popular, easy-to-learn programming language, so that the resulting code would be easy to understand and modify. In addition, the modular, constraint-based resolution algorithm, inspired by the OT reformulation of Centering Theory presented in (Beaver, 2004), creates a particularly flexible and easy to modify resolution engine.

2.1. Python Programming Language

We chose to implement PYCOT using the Python programming language for a number of reasons. Python is a free, powerful, portable, easy to use and easy to learn programming language (Lutz and Ascher, 2004). In our own expe-

[1] http://nltk.sourceforge.net

[2] GuiTAR, reported at LREC2004 (Poesio and Kabadjov, 2004), is not yet available to download

[3] www.alias-i.com

Table 1: Cospecification hypotheses for sentence (2).The preferred hypothesis is indicated with ≫

(1) Jane$_i$ likes Mary$_j$. (2) She$_k$ often brings her$_l$ flowers.	Agree	Disjoint	Pro-Top	Fam-Def	Cohere	Align	Value
≫ $k = i, l = j$					*		10
$k = l = i$		*			*		10010
$k = j, l = i$					*	*	11
$k = l = j$		*			*		10010
$k = i, l \notin \{i, j\}$				*	*		110
$k = j, l \notin \{i, j\}$				*	*		110
$k \notin \{i, j\}, l = i$				*	*	*	111
$k \notin \{i, j\}, l = j$				*	*	*	111
$k, l \notin \{i, j\}, k \neq l$			*	**	*	*	1211
$k = l \notin \{i, j\}$		*	*	**	*	*	11211

rience, we have found that code written in Python is easy to understand and modify and that we are able to develop code very quickly due to Python's clear syntax; in fact, it is said that Python is "optimized for speed of development" (Lutz and Ascher, 2004). Python is popular among computational linguists, as evidenced by the existence of the open-source NLTK collection of natural language processing tools, and its use in introductory computational linguistics courses at a number of universities including our own (Bird, 2006).

2.2. Optimality Theory Modular Design

PYCOT uses as its starting point the restatement of pronoun resolution as a set of Optimality Theory constraints (Prince and Smolensky, 1993) described in (Beaver, 2004). Optimality Theory (henceforth OT) requires that the preferences for pronoun resolution be stated as a set of independently-functioning constraints, which are ordered in terms of their importance. Constraint functions can calculate either binary distinctions or continuous variables modeling stochastic processes (Boersma, 2005). An advantage of OT is that constraints are violable. OT uses a shared memory structure called a *tableau* to store the results of constraint evaluation. An example tableau (from Beaver, 2004) is shown in Figure 1. This tableau compares 10 different hypotheses for co-indexing of entities k and l in sentence (2) with referents i and j introduced in sentence (1). Each possible interpretation is represented as an alternate hypothesis. Each constraint is evaluated against each hypothesis, and the result is recorded in the tableau, where the constraint's ranking within the tableau controls the magnitude of the value contributed to the overall evaluation value (in the right-most column). The hypothesis with the smallest number in the evaluation column is selected as the preferred hypothesis.

We find the constraint-based design produces modularity and transparency in the resolution process; it is easy to see exactly what caused a particular pronoun to be matched with a particular antecedent by examining the tableau generated for that hypothesis. The set of constraints included in the program together formulate the pronoun resolution algorithm that PYCOT implements. The distribution version of PYCOT includes the constraints required to implement Beaver's COT algorithm (Beaver, 2004). To change the behavior of the algorithm, additional constraints may be added, or the constraints may be re-ordered to reflect a different relative preference ranking on the constraints.

3. Architecture and Implementation

According to (Byron and Tetreault, 1999), the processing for anaphora resolution is broken into two stages: an initial preprocessing phase which converts the original source text into a set of discourse entity tokens, and a second phase which contains core pronoun resolution processing. This architecture is depicted in Table 1. We discuss the details of our implementation in the remainder of this section.

3.1. Preprocessing Phase

The preprocessor takes the source text as input and reduces it to a list of items to be resolved and candidate antecedents. This design places all dependencies on the input text format in the preprocessor, thus isolating the resolution engine from variations due to tagging schemes, bracketing guidelines used for different corpora, morphological analysis for different input languages, etc. Because this module must work with the formatting style of the source text, modifications may be required if a developer wishes to customize the system to run against a new input source. The preprocessor is responsible for turning the data with pronouns to be resolved into a sequence of lists of Referring Expression Tokens, or RETokens, described below. We use the term Referring Expression for these items rather than Noun Phrase, because an RE might be either an overt noun phrase or an unexpressed argument that needs to be resolved. The distribution version of PYCOT includes preprocessors to convert data from the English-language Wall Street Journal articles from Treebank-3 (Marcus et al., 1993) and data from the Penn Korean Treebank (Palmer et al., 2002) into RETokens. Despite the fact that both of these datasets are formatted as treebanks, there were enough differences in the formatting to require that we write two different, albeit very similar, preprocessors.

The basic preprocessing algorithm for parsed input involves searching the syntactic tree for base noun phrases or phonologically null elements and creating a new REToken for each item encountered in the source text. An REToken is an object class that captures all of the fields of interest from the referring expression, such as its lexical content, determiner, semantic information such as agreement features, etc. In Treebank data, this information is gleaned from both the

Figure 1: Dataflow architecture of PYCOT

lexical string comprising the RE and the POS tags. In addition, during the search, information regarding each noun phrases' syntactic context, such as unique identifiers for its parent constituents, its linear position in the sentence, and depth in the syntactic tree is placed in the relevant fields in its REToken representation, as described below. These fields are used to determine anti-coindexing based on coargument relations. Gold-standard information from coreference annotation can also be included in the RETokens, if the code is being used for experimental evaluation purposes.

3.2. Referring Expression Tokens

The REToken class described above is a subclass of NLTK's Token class. It is essentially a Python dictionary, wherein each field represents a piece of data (ie., gender, number, syntactic parents, etc) about the referring expression the token represents. Our implementation currently contains two subclasses of REToken, one with fields and functions specific to English, and one with fields and functions specific to Korean, as well as the general REToken, whose fields should be seen as a suggestion about the types of information that would be useful to pronoun resolution in general. Each REToken represents a referring expression, either a pronoun or null argument to be resolved or a non-pronominal base noun phrase that might serve as an antecedent. An instance of the English-specific subclass of REToken is shown in Figure 2.

New languages can be supported either by extending the general REToken, or by creating a new subclass of it with fields specific to the new language, which requires only that a new set of fields to be filled by the preprocesser (and any functions which use those fields, if desired) be defined. When creating our Korean subclass of REToken, it was possible to reuse many of the relevant fields (ie, the fields for syntactic parents, depth in the tree, etc) from our general RETokens. In order to keep track of information specific to anaphora in Korean, we added fields for morphological endings and functions for determining whether certain markers (particularly the topic marker *nun*/*un*) were present in the morphology, as well as a function for determining whether the given referring expression was a null element. The REToken format makes porting PYCOT to new languages very easy, in that adding new fields to capture language-specific information is as simple as adding a field to a Python dictionary. Figure 3 shows a sample Korean REToken. Unlike English, where the 'LEX' field simply contains the string comprising the referring expression, in Korean, the corresponding entry consists of a list of morphemes with their tags, since morphological information such as topic-marking is relevant to pronoun resolution in Korean.

After preprocessing converts the input text into a sequence of lists of RETokens, the Resolution Engine performs pronoun resolution on the sequence. The current implementation includes a Centering-based algorithm, which only resolves pronouns with antecedents in the prior sentence, using the constraints, described below, to rank and select the optimal resolution. While Centering-based pronoun resolution algorithms are inherently limited by the fact that Centering theory does not make any predictions about pronouns whose antecedents are not in the previous sentence, the constraints that embody Centering's predictions can be combined with other constraints, for example preferring an antecedent from the current sentence over an antecedent from the previous sentence.

The constraint-based implementation makes it possible to test other factors that are claimed to play a role in local coherence and to test the relative importance of these factors against a corpus, as in (Byron et al., in press). For our own experimentation purposes, we have implemented both

```
<AGR=     {'PERSON':    3,
           'NUM':       'S',
           'GENDER':    'M'}
DEPTH=    1,
DET=      None,
DIS=      1,
ID=       'N1',
LEX=      'MR.-VINKEN',
MENTION=  2,
PARENTS=  ['S1'],
POSITION= 1,
QUOTED=   False,
REF=      2,
TYPE=     'NP-SBJ',
UTT=      2>
```

Figure 2: Sample English REToken

```
<DEPTH=   1,
DIS=      '08',
LEX=      [['(character1)', 'NNC'],
           ['(character2)', 'NNC'],
           ['(character3', 'NNC'],
           ['(character4)', 'PAD'],
           ['(character5)', 'PAU']],
MENTION=  '1',
PARENTS=  ['S1'],
POSITION= 1,
REF=      '100',
ROLE=     'COMP',
TYPE=     '#n',
UTT=      '1'>
```

Figure 3: Sample Korean REToken

the LRC-algorithm (Tetreault, 2001) and Hobbs' semantically naïve pronoun resolution algorithm (Hobbs, 1986) by adding new constraints into PYCOT. The results are reported in (Byron and Gegg-Harrison, 2004). Other types of anaphora resolution could also be added to the Resolution Engine.

3.3. Resolution Engine

The resolution engine contains functions to drive the resolution process as well as the pronoun resolution evaluation code which can be used for experimentation and evaluation purposes. The control flow encoded in the distribution version of PYCOT is a batch process which expects to receive the entire batch of sentences to be processed as one large list, in which each sublist of RETokens represents the discourse entities in a sentence. This processing flow can be customized by other developers wishing to create a more incremental process. The batch driver code works through the list developing pronoun resolution hypotheses for one sentence at a time, and evaluating those hypotheses using the OT constraint-checking functions. As the constraints are evaluated on each possible interpretation, an OT-style tableau is constructed containing the results. The preferred interpretation is the one whose constraint-violations, if any, are in the lowest-ranking constraints.

The driver code has access to one global variable, CONSTRAINTS, which is a list containing the strings of names of the constraint-checker functions to be run, in ranked order. This list controls which constraints the algorithm uses and the placement of their output into columns in the tableau. The behavior of the algorithm can be modified simply by changing this list of constraint-checking function names, either by adding/deleting entries or re-ordering the function names within the list.

3.4. Constraint implementation

```
def Disjoint(hypothesis, thisS, prevS):
    total = 0
    for pair in hypothesis:
            if pair[0].contind(pair[1]):
                        total += 1
    return total
```

Figure 4: Sample Constraint Function

The Resolution Engine program includes a set of optimality-theory constraint evaluation functions. Each constraint is implemented as an individual function within the program. Crucial to the design is that all of the constraint-checking functions that will be used to evaluate the hypothesis expose an identical interface: they each receive the same input parameter list and they each return a numeric value representing how many violations of the constraint are present in each hypothesis they evaluate. An example constraint function is shown in Figure 4. In our implementation, the constraints each take three arguments: the set of possible pairings between pronouns and antecedents, the list of tokens in the current sentence, and the list of tokens in the previous sentence.

In the Disjoint function shown above, the hypothesis is a complete pairing of each pronoun in the current sentence with a candidate antecedent from the prior sentence. The constraint code can rely on additional methods implemented in the resolution engine, such as determining the topic of the current sentence (the backward-looking center, in Centering terms), performing unification on agreement features, calculate co-argument relations, etc.

Because of the modularity of the constraint functions, porting the algorithm to new languages is quite simple. The Resolution Engine remains the same, but new constraints relevant to the resolution of pronouns in the new language can be defined and added to the constraint list used by the Engine.

3.4.1. Centering and Optimality Theory

The basis of the algorithm that is coded in the distribution version of PYCOT is the OT formulation of Centering Theory presented in (Beaver, 2004). Centering Theory (Grosz et al., 1995) is a theory of discourse coherence which has been used in the computational literature (starting with (Brennan et al., 1987)) to aid in determining the referents of anaphoric elements in discourse. Since many theoretical claims about anaphora are couched in terms of Centering Theory, this Centering-based module is also a useful tool for testing such claims against a corpus, as in (Byron et al., in press). Centering assumes that topical continuity is necessary for producing coherent discourse, and prescribes a set of rules that can be used to evaluate proposed pronoun resolutions in terms of topical continuity vs. topical transitions.

In the OT-based restatement of centering, the transition preferences are encoded in terms of a set of ranked constraints (Beaver, 2004). The constraints used in the implementation of Beaver's basic reformulation of Centering are discussed in (Byron and Gegg-Harrison, 2004).

- AGREE: Anaphoric expressions agree with their antecedents in terms of number and gender
- DISJOINT: Co-arguments of a predicate are disjoint
- PRO-TOP: The topic is pronominalized
- FAM-DEF: Each definite NP is familiar
- COHERE: The topic of the current sentence is the topic of the previous one
- ALIGN: The topic is in subject position

Although Beaver proposes that the list of constraints which replicates centering-based pronoun resolution is [Agree, Disjoint, ProTop, Local, Cohere, Align], our experiments with the Wall Street Journal portion of the Penn Treebank indicate that the best performing ranking is [Agree, Disjoint, ProTop, Local, Cohere].

3.4.2. Additional Korean Constraints

To port PYCOT to process Korean anaphors annotated in the Penn Korean Treebank, we created a new preprocessor which was modified to handle this input format, and also added the following constraint-checking functions:

- GTOPIC: is violated if a discourse-topic has appeared in the current discourse (marked with *nun* or *un*) and the highest ranking zero pronoun in the current sentence does not refer to it. GTopic is vacuously satisfied if no entity in the current discourse was marked as a topic.
- ZEROALIGN: is violated if the previous sentence contains a zero anaphor and the highest ranking zero pronoun in the current sentence does not refer to it. ZeroAlign is vacuously satisfied if the current sentence does not contain a zero anaphor.
- DELETION: is violated if two pronouns in the sentence have the same proposed antecedent and one of them is not a zero pronoun (from (Lee, 2003)).

```
def GTopic(hypothesis, thisS, prevS):
    zeros = findzeros(thisSent)
    if globnun is None and zeros:
      for pair in hypothesis:
        if pair[0] in zeros and
            pair[1].refersto(globnun):
                return 1
    return 0
```

Figure 5: Sample Korean Constraint Function

Figure 5 shows the code for the GTopic constraint, which compares proposed antecedents to the global topic, stored in a variable named `globnun`. Our experiments included determining the optimal ordering of the constraints we developed for Korean. Using a small annotated subset of the Penn Korean Treebank for evaluation, the optimal ordering we determined was: [Agree, ProTop, GTopic, Local, Cohere, Align].

4. Testing PYCOT

We have used PYCOT to perform pronoun resolution in English (Byron and Gegg-Harrison, 2004) and Korean (Byron et al., in press). These studies demonstrate PYCOT's usefulness and the simplicity of porting between languages.

5. Distribution

The code and documentation for PYCOT is available from our lab's webpage, at http://slate.cse.ohio-state.edu/pycot. The user agreement allows the code to be freely used for non-commercial research purposes only. Licensing arrangements are possible if any user wishes to incorporate PYCOT into a commercial product.

6. Conclusions

Our goal with PYCOT is to provide a useful tool for pronoun resolution that can be easily tailored to work for a variety of natural languages without requiring the user to rewrite significant amounts of code, or that can be simply used out-of-the-box for users who wish to add a centering-style pronoun resolution module into an existing language processing pipeline. We hope we have achieved this goal by creating PYCOT with the following properties:

1. The use of Python as the implementation language makes the code simple to understand and modify.
2. The overall program is separated into a preprocessing stage, which can process a variety of different input formats, tagging schemes, etc. and a pronoun resolution engine which implements the pronoun resolution algorithm. This design encapsulates the pronoun resolution code from variations in input formats, and, we hope, makes the pronoun resolution process more easily customized for different pronoun resolution techniques.
3. The modular, constraint-based OT approach makes altering the behavior of the algorithm a simple task and provides a means for testing theoretical claims about constraints and preferences for the resolution of anaphora against a corpus.

Acknowledgements

The authors would like to thank the Korean Research Foundation Grant (KRF-2004-037-A00098) and the GE Foundation Faculty for the Future grant for their support of this project.

7. References

David I. Beaver. 2004. The optimization of discourse anaphora. *Linguistics and Philosophy*, 27(1):3–56.

Steven Bird. 2006. Teaching with NLTK.

Paul Boersma. 2005. A stochastic ot account of paralinguistic tasks such as grammatiality and prototypicality judgments. manuscript.

Susan Brennan, Marilyn Friedman, and Carl Pollard. 1987. A centering approach to pronouns. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics (ACL '87)*, pages 155–162.

Donna K. Byron and Whitney Gegg-Harrison. 2004. Evaluating optimality theory for pronoun resolution algorithm specification. In *Proceedings of the Discourse Anaphora and Anaphor Resolution Colloquium (DAARC 2004)*, pages 27–32.

Donna K. Byron and Joel R. Tetreault. 1999. A flexible architecture for reference resolution. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL '99)*.

Donna K. Byron, Whitney Gegg-Harrison, and Sun-Hee Lee. in press. Resolving zero anaphors and pronouns in korean. *Traitement Automatique des Langues (TAL), special issue on anaphora resolution*.

Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. 1995. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–226.

Jerry Hobbs. 1986. Resolving pronoun reference. In *Readings in Natural Language Processing*, pages 339–352. Morgan Kaufmann.

Christopher Kennedy and Branimir Boguraev. 1996. Anaphora in a wider context: Tracking discourse referents. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI96)*.

Dong-Young Lee. 2003. An extended centering theoretic approach to the recovery of omitted constituents in korean dialogues. *Korean Journal of Linguistics*, 35:153–175.

Mark Lutz and David Ascher. 2004. *Learning Python*. O'Reilly.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Martha Palmer, Chung-Hye Han, Na-Rae Han, Eon-Suk Ko, Hee-Jong Yi, Alan Lee, Chris Walker, John Duda, and Nianwen Xue. 2002. The penn korean treebank. Technical Report LDC2002T26, UPenn.

Massimo Poesio and Mijail A. Kabadjov. 2004. A general-purpose off-the-shelf anaphora resolution module: Implementation and preliminary evaluation. In *Proceedings of LREC 2004*.

A. Prince and P. Smolensky. 1993. Optimality theory: Constraint interaction in generative grammar. Technical Report 2, Rutgers Center for Cognitive Science.

Long Qiu, Min-Yen Kan, and Tat-Seng Chua. 2004. A public reference implementation of the rap anaphora resolution algorithm. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, pages 291–294.

Joel Tetreault. 2001. A corpus-based evaluation of centering and pronoun resolution. *Computational Linguistics*, 27(4):507–520.