Teaching computational linguistics to a large, diverse student body: courses, tools, and interdepartmental interaction

Jason Baldridge and Katrin Erk
Department of Linguistics
The University of Texas at Austin
{jbaldrid,erk}@mail.utexas.edu

Abstract

We describe course adaptation and development for teaching computational linguistics for the diverse body of undergraduate and graduate students the Department of Linguistics at the University of Texas at Austin. We also discuss classroom tools and teaching aids we have used and created, and we mention our efforts to develop a campus-wide computational linguistics program.

1 Introduction

We teach computational linguistics courses in the linguistics department of the University of Texas at Austin, one of the largest American universities. This presents many challenges and opportunities; in this paper, we discuss issues and strategies for designing courses in our context and building a campus-wide program.[1] The main theme of our experience is that courses should be targeted toward specific groups of students whenever possible. This means identifying specific needs and designing the course around them rather than trying to satisfy a diverse set of students in each course. To this end, we have split general computational linguistics courses into more specific ones, e.g., working with corpora, a non-technical overview of language technology applications, and natural language processing. In section 2, we outline how we have stratified our courses at both the graduate and undergraduate levels.

[1]Links to the courses, tools, and resources described in this paper can be found on our main website: http://comp.ling.utexas.edu

As part of this strategy, it is crucial to ensure that the appropriate student populations are reached and that the courses fulfill degree requirements. For example, our *Language and Computers* course fulfills a Liberal Arts science requirement and our *Natural Language Processing* is cross-listed with computer science. This is an excellent way to get students in the door and ensure that courses meet or exceed minimum enrollments. We find that many get hooked and go on to specialize in computational linguistics.

Even for targeted CL courses, there is still usually significant diversity of backgrounds among the students taking them. Thus, it is still important to carefully consider the teaching tools that are used; in section 3, we discuss our experience with several standard tools and two of our own. Finally, we describe our efforts to build a campus-wide CL program that provides visibility for CL across the university and provides coherence in our course offerings.

2 Courses

Our courses are based on those initiated by Jonas Kuhn between 2002 and 2005. Since 2005, we have created several spin-off courses for students with different backgrounds. Our broad goals for these courses are to communicate both the practical utility of computational linguistics and its promise for improving our understanding of human languages.

2.1 Graduate

We started with two primary graduate courses, *Computational Linguistics I and II*. The first introduces central algorithms and data structures in computational linguistics, while the second focuses on learn-

Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics (TeachCL-08), pages 1–9,
Columbus, Ohio, USA, June 2008.

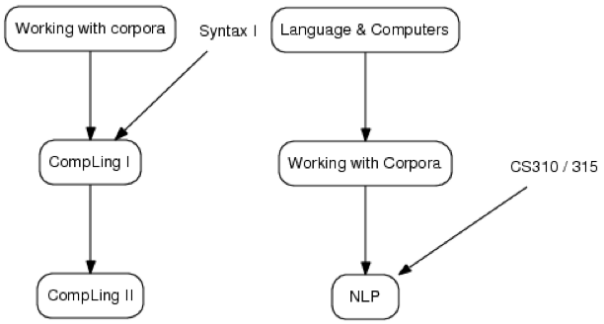

Figure 1: Flow for non-seminar courses. Left: graduate courses, right: undergraduate courses.

ing and disambiguation. This served a computationally savvy segment of the student population quite well. However, we view one of our key teaching contributions as computational linguists in a linguistics department to be providing non-computational students with technical and formal skills useful for their research. We discovered quickly that our first computational linguistics course did not fill these needs, and the second is not even accessible to most students. The graduate linguistics students did put in the effort to learn Python for *Computational Linguistics I*, but many would have preferred a (much) gentler introduction and also more coverage of issues connected to their linguistic concerns. This led us to create a new course, *Working with Corpora*.

Still, there is a need for the primary courses, which receive interest from many students in computer science and also graduate students from other departments such as German and English. One of the great surprises for us in our graduate courses has been the interest from excellent linguistics and computer science undergraduates.

We have sought to encourage our students to be active in the academic community outside of UT Austin. One way we do this is to have a final project for each course (and most seminars) that has four distinct stages: (i) a project proposal halfway through the semester, (ii) a progress report three-quarters of the way through, (iii) a 10-minute presentation during the last week of class, and (iv) a final report at the end. We have found that having course projects done in this staged manner ensures that students think very thoroughly about what their topic is early on, receive significant feedback from us, and then still have enough time to do significant implementation for their project, rather than rushing everything in last minute. Also, by having students do presentations on their work before they hand in the final report, they can incorporate feedback from other students. A useful strategy we have found for scoring these projects is to use standard conference reviews in *Computational Linguistics II*. The final projects have led to several workshops and conference publications for the students so far, as well as honors theses. The topics have been quite varied (in line with our varied student body), including lexicon induction using genetic algorithms (Ponvert, 2007), alignment-and-transfer for bootstrapping taggers (Moon and Baldridge, 2007), lemmatization using parallel corpora (Moon and Erk, 2008), graphical visualization of articles using syntactic dependencies (Jeff Rego, CS honors thesis), and feature extraction for semantic role labeling (Trevor Fountain, CS honors thesis).

Working with corpora. Computational linguistics skills and techniques are tremendously valuable for linguists using corpora. Ideally, a linguist should be able to extract the relevant data, count occurrences of phenomena, and do statistical analyses. The intersection of these skills and needs is the core of this course, which covers corpus formats (XML, bracket formats for syntax, "word/POS" formats for part-of-speech information), query languages and tools (regular expressions, cqp, tregex), and some statistical analysis techniques. It also teaches Python gently for liberal arts students who have *never programmed* and have only *limited or no* knowledge of text processing. Other main topics are the compilation of corpora and corpus annotation, with issues like representativity and what meta-data to include. At the end of this course, students are prepared for our primary computational courses.

We observed the tremendous teaching potential of effective visualization in this course with the R statistics package. It was used for statistical analyses: students loved it because they could produce meaningful results immediately and visualize them. The course includes only a very short two-session introduction to working with R. We were worried that this would overtax students because R is its own

programming language. But interestingly they had no problems with learning this second programming language (after Python). This is particularly striking as most of the students had no programming experience prior to the class.

We have not yet used the Natural Language Toolkit (Loper and Bird, 2002) (see Section 3.1) in this course. But as it, too, offers visualization and rapid access to meaningful results, we intend to use it in the future. In particular, the NLTK allows very easy access to Toolbox data (Robinson et al., 2007), which we feel will greatly improve the utility and appeal of the course for the significant number of documentary linguistics students in the department.

Seminars. We also offer several seminars in our areas of interest. These include *Categorial Grammar*, *Computational Syntax*, and *Lexical Acquisition*. These courses have attracted "non-computational" linguistics students with related interests, and have served as the launching point for several qualifying papers and masters theses. It is important to offer these courses so that these students gain a view into computational linguistics from the standpoint of a topic with which they already have some mastery; it also ensures healthier enrollments from students in our own department.

We are currently co-teaching a seminar called *Spinning Straw into Gold: Automated Syntax-Semantics Analysis*, that is designed to overlap with the CoNLL-2008 shared task on joint dependency parsing and semantic role labeling. The entire class is participating in the actual competition, and we have been particularly pleased with how this external facet of the course motivates students to consider the topics we cover very carefully – the papers truly matter for the system we are building. It provides an excellent framework with which to judge the contributions of recent research in both areas and computational linguistics more generally.

2.2 Undergraduate

Our first undergraduate course was *Introduction to Computational Linguistics* in Fall 2006. Our experience with this course, which had to deal with the classic divide in computational linguistics courses between students with liberal arts versus computer science backgrounds, led us to split it into two courses. We briefly outline some of the missteps with this first course (and what worked well) and how we are addressing them with new courses.

Introduction to Computational Linguistics. This course is a boiled-down version of the graduate *Computational Linguistics I* taught in Fall 2006. Topics included Python programming, regular expressions, finite-state transducers, part-of-speech tagging, context-free grammar, categorial grammar, meaning representations, and machine translation.

Overall, the course went well, but enrollment dropped after the mid-term. As many have found teaching such courses, some students truly struggled with the course material while others were ready for it to go much faster. Several students had interpreted "introduction" to mean that it was going to be *about* computational linguistics, but that they would not actually have to *do* computational linguistics. Many stayed with it, but there were still others who could have gone much further if it had not been necessary to slow down to cover basic material like *for* loops. Note that several linguistics majors were among the compationally savvy students.

In fairness to the students who struggled, it was certainly ill-advised to ask students with no previous background to learn Python and XFST in a single semester. One of the key points of confusion was regular expression syntax. The syntax used in the textbook (Jurafsky and Martin, 2000) transfers easily to regular expressions in Python, but is radically different from that of XFST. For students who had never coded anything in their life, this proved extremely frustrating. On the other hand, for computationally savvy students, XFST was great fun, and it was an interesting new challenge after having to sit through very basic Python lectures.

On the other hand, the use of NLTK to drive learning about Python and NLP tasks (like building POS-taggers) significantly eased the burden for new programmers. Many of them were highly satisfied that they could build interesting programs and experiment with their behavior so easily.

Language and Computers. We had fortunately already planned the first replacement course: *Language and Computers*, based on the course designed at the Department of Linguistics at the Ohio State University (Brew et al., 2005). This course intro-

duces computational linguistics to a general audience and is ideal for students who want exposure to computational methods without having to learn to program. We designed and taught it jointly, and added several new aspects to the course. Whereas OSU's course fulfills a Mathematical and Logical Analysis requirement, our course fulfills a Science requirement for liberal arts majors. These requirements were met by course content that requires understanding and thinking about formal methods.

The topics we added to our course were question answering, cryptography,[2] and world knowledge. The course provides ample opportunity to discuss high-level issues in language technology with low-level aspects such as understanding particular algorithms (e.g., computing edit distance with dynamic programming) and fundamental concepts (such as regular languages and frequency distributions).

In addition to its target audience, the course nonetheless attracts students who are already well-versed in many of the low-level concepts. The high-level material plays an important role for such students: while they find the low-level problems quite easy, many find a new challenge in thinking about and communicating clearly the wider role that such technologies play. The high-level material is even more crucial for holding the interest of less formally minded students. It gives them the motivation to work through and understand calculations and computations that might otherwise bore them. Finally, it provides an excellent way to encourage class discussion. For example, this year's class became very animated on the question of "Can a machine think?" that we discussed with respect to dialogue systems.

Though the course does not require students to do any programming, we do show them short programs that accomplish (simplified versions of) some of the tasks discussed in the course; for example, short programs for document retrieval and creating a list of email address from US census data. The goal is to give students a glimpse into such applications, demonstrate that they are not hugely complicated magical systems, and hopefully entice some of them to learn how to do it for themselves.

The 2007 course was quite successful: it filled up (40 students) and received very positive feedback from the students. It filled up again for this year's Spring 2008 offering. The major challenge is the lack of a textbook, which means that students must rely heavily on lecture slides and notes.

Words in a Haystack: Methods and Tools for Working with Corpora. This advanced undergraduate version of *Working with corpora* was offered because we felt that graduate and undergraduate linguistics students were actually on an equal footing in their prior knowledge, and could profit equally from a gentle introduction to programming. Although the undergraduate students were active and engaged in the class, they did not benefit as much from it as the graduate students. This is likely because graduate students had already experienced the need for extracting information from corpora for their research and the consequent frustration when they did not have the skills to do so.

Natural Language Processing. This is an demanding course that will be taught in Fall 2008. It is cross-listed with computer science and assumes knowledge of programming and formal methods in computer science, mathematics, or linguistics. It is designed for the significant number of students who wish to carry on further from the courses described previously. It is also an appropriate course for undergraduates who have ended up taking our graduate courses for lack of such an option.

Much of the material from *Introduction to Computational Linguistics* will be covered in this course, but it will be done at a faster pace and in greater detail since programming and appropriate thinking skills are assumed. A significant portion of the graduate course *Computational Linguistics II* also forms part of the syllabus, including machine learning methods for classification tasks, language modeling, hidden Markov models, and probabilistic parsing.

We see cross-listing the course with computer science as key to its success. Though there are many computationally savvy students in our liberal arts college, we expect cross-listing to encourage significantly more computer science students to try out a course that they would otherwise overlook or be unable to use for fulfilling degree requirements.

[2]The idea to cover cryptography came from a discussion with Chris Brew; he now teaches an entire course on it at OSU.

3 Teaching Tools and Tutorials

We have used a range of external tools and have adapted tools from our own research for various aspects of our courses. In this section, we describe our experience using these as part of our courses.

We have used Python as the common language in our courses. We are pleased with it: it is straightforward for beginning programmers to learn, its interactive prompt facilitates in-class instruction, it is text-processing friendly, and it is useful for gluing together other (e.g., Java and C++) applications.

3.1 External tools and resources

NLTK. We use the Natural Language Toolkit (NLTK) (Loper and Bird, 2002; Bird et al., 2008) in both undergraduate and graduate courses for in-class demos, tutorials, and homework assignments. We use the toolkit and tutorials for several course components, including introductory Python programming, text processing, rule-based part-of-speech tagging and chunking, and grammars and parsing. NLTK is ideal for both novice and advanced programmers. The tutorials and extensive documentation provide novices with plenty of support outside of the classroom, and the toolkit is powerful enough to give plenty of room for advanced students to play. The demos are also very useful in classes and serve to make many of the concepts, e.g. parsing algorithms, much more concrete and apparent. Some students also use NLTK for course projects. In all, NLTK has made course development and execution significantly easier and more effective.

XFST. A core part of several courses is finite-state transducers. FSTs have unique qualities for courses about computational linguistics that are taught in linguistics department. They are an elegant extension of finite-state automata and are simple enough that their core aspects and capabilities can be expressed in just a few lectures. Computer science students immediately get excited about being able to relate string languages rather than just recognizing them. More importantly, they can be used to elegantly solve problems in phonology and morphology that linguistics students can readily appreciate.

We use the Xerox Finite State Toolkit (XFST) (Beesley and Karttunen, 2003) for in-class demonstrations and homeworks for FSTs. A great aspect of using XFST is that it can be used to show that different representations (e.g., two-level rules versus cascaded rules) can be used to define the same regular relation. This exercise injects some healthy skepticism into linguistics students who may have to deal with formalism wars in their own linguistic subfield. Also, XFST allows one to use lenient composition to encode Optimality Theory constraints and in so doing show interesting and direct contrasts and comparisons between paper-and-pencil linguistics and rigorous computational implementations.

As with other implementation-oriented activities in our classes, we created a wiki page for XFST tutorials.[3] These were adapted and expanded from Xerox PARC materials and Mark Gawron's examples.

Eisner's HMM Materials. Simply put: the spreadsheet designed by Jason Eisner (Eisner, 2002) for teaching hidden Markov models is fantastic. We used that plus Eisner's HMM homework assignment for *Computational Linguistics II* in Fall 2007. The spreadsheet is great for interactive classroom exploration of HMMs—students were very engaged. The homework allows students to implement an HMM from scratch, giving enough detail to alleviate much of the needless frustration that could occur with this task while ensuring that students need to put in significant effort and understand the concepts in order to make it work. It also helps that the new edition of Jurafsky and Martin's textbook discusses Eisner's ice cream scenario as part of its much improved explanation of HMMs. Students had very positive feedback on the use of all these materials.

Unix command line. We feel it is important to make sure students are well aware of the mighty Unix command line and the tools that are available for it. We usually have at least one homework assignment per course that involves doing the same task with a Python script versus a pipeline using command line tools like `tr`, `sort`, `grep` and `awk`. This gives students students an appreciation for the power of these tools and for the fact that they are at times preferable to writing scripts that handle everything, and they can see how scripts they write can form part of such pipelines. As part of this module,

[3] `http://comp.ling.utexas.edu/wiki/doku.php/xfst`

we have students work through the exercises in the draft chapter on command line tools in Chris Brew and Marc Moens' Data-Intensive Linguistics course notes or Ken Church's *Unix for Poets* tutorial.[4]

3.2 Internal tools

Grammar engineering with OpenCCG. The grammar engineering component of *Computational Syntax* in Spring 2006 used OpenCCG,[5] a categorial grammar parsing system that Baldridge created with Gann Bierner and Michael White. The problem with using OpenCCG is that its native grammar specification format is XML designed for machines, not people. Students in the course persevered and managed to complete the assignments; nonetheless, it became glaringly apparent that the non-intuitive XML specification language was a major stumbling block that held students back from more interesting aspects of grammar engineering.

One student, Ben Wing, was unhappy enough using the XML format that he devised a new specification language, DotCCG, and a converter to generate the XML from it. DotCCG is not only simpler—it also uses several interesting devices, including support for regular expressions and string expansions. This expressivity makes it possible to encode a significant amount of morphology in the specification language and reduce redundancy in the grammar.

The DotCCG specification language and converter became the core of a project funded by UT Austin's Liberal Arts Instructional Technology Services to create a web and graphical user interface, VisCCG, and develop instructional materials for grammar engineering. The goal was to provide suitable interfaces and a graduated series of activities and assignments that would allow students to learn very basic grammar engineering and then grow into the full capabilities of an established parsing system.

A web interface provided an initial stage that allowed students in the undergraduate *Introduction to Computational Linguistics* course (Fall 2006) to test their grammars in a grammar writing assignment. This simple interface allows students to first write out a grammar on paper and then implement it and test it on a set of sentences. Students grasped the concepts and seemed to enjoy seeing the grammar's coverage improve as they added more lexical entries or added features to constrain them appropriately. A major advantage of this interface, of course, is that it was not necessary for students to come to the lab or install any software on their own computers.

The second major development was VisCCG, a graphical user interface for writing full-fledged OpenCCG grammars. It has special support for DotCCG, including error checking, and it displays grammatical information at various levels of granularity while still allowing direct *source text* editing of the grammar.

The third component was several online tutorials—written on as publicly available wiki pages—for writing grammars with VisCCG and DotCCG. A pleasant discovery was the tremendous utility of the wiki-based tutorials. It was very easy to quickly create tutorial drafts and improve them with the graduate assistant employed for creating instructional materials for the project, regardless of where we were. More importantly, it was possible to fix bugs or add clarifications while students were following the tutorials in the lab. Furthermore, students could add their own tips for other students and share their grammars on the wiki.

These tools and tutorials were used for two graduate courses in Spring 2007, *Categorial Grammar* and *Computational Linguistics I*. Students caught on quickly to using VisCCG and DotCCG, which was a huge contrast over the previous year. Students were able to create and test grammars of reasonable complexity very quickly and with much greater ease. We are continuing to develop and improve these materials for current courses.

The resources we created have been not only effective for classroom instruction: they are also being used by researchers that use OpenCCG for parsing and realization. The work we did produced several innovations for grammar engineering that we reported at the workshop on Grammar Engineering Across the Frameworks (Baldridge et al., 2007).

3.3 A lexical semantics workbench: Shalmaneser

In the lexical semantics sections of our classes, word sense and predicate-argument structure are core topics. Until now, we had only discussed word sense

[4] http://research.microsoft.com/users/church/wwwfiles/tutorials/unix_for_poets.ps

[5] http://openccg.sf.net

disambiguation and semantic role labeling theoretically. However, it would be preferable to give the students hands-on experience with the tasks, as well as a sense of what does and does not work, and why the tasks are difficult. So, we are now extending Shalmaneser (Erk and Pado, 2006), a SHALlow seMANtic parSER that does word sense and semantic role assignment using FrameNet frames and roles, to be a teaching tool. Shalmaneser already offers a graphical representation of the assigned predicate-argument structure. Supported by an instructional technology grant from UT Austin, we are extending the system with two graphical interfaces that will allow students to experiment with a variety of features, settings and machine learning paradigms. Courses that only do a short segment on lexical semantic analysis will be able to use the web interface, which does not offer the full functionality of Shalmaneser (in particular, no training of new classifiers), but does not require any setup. In addition, there will be a stand-alone graphical user interface for a more in-depth treatment of lexical semantic analysis. We plan to have the new platform ready for use for Fall 2008.

Besides a GUI and tutorial documents, there is one more component to the new Shalmaneser system, an adaptation of the idea of grammar engineering workbenches to predicate-argument structure. Grammar engineering workbenches allow students to specify grammars declaratively. For semantic role labeling, the only possibility that has been available so far for experimenting with new features is to program. But, since semantic role labeling features typically refer to parts of the syntactic structure, it should be possible to describe them declaratively using a tree description language. We are now developing such a language and workbench as part of Shalmaneser. We aim for a system that will be usable not only in the classroom but also by researchers who develop semantic role labeling systems or who need an automatic predicate-argument structure analysis system.

4 University-wide program

The University of Texas at Austin has a long tradition in the field of computational linguistics that goes back to 1961, when a major machine translation project was undertaken at the university's Linguistics Research Center under the direction of Winfred Lehman. Lauri Karttunen, Stan Peters, and Bob Wall were all on the faculty of the linguistics department in the late 1960's, and Bob Simmons was in the computer science department during this time. Overall activity was quite strong throughout the 1970's and 1980's. After Bob Wall retired in the mid-1990's, there was virtually no computational work in the linguistics department, but Ray Mooney and his students in computer science remained very active during this period.[6]

The linguistics department decided in 2000 to revive computational linguistics in the department, and consequently hired Jonas Kuhn in 2002. His efforts, along with those of Hans Boas in the German department, succeeded in producing a computational linguistics curriculum, funding research, (re)establishing links with computer science, and attracting an enthusiastic group of linguistics students.

Nonetheless, there is still no formal interdepartmental program in computational linguistics at UT Austin. Altogether, we have a sizable group of faculty and students working on topics related to computational linguistics, including many other linguists, computer scientists, psychologists and others who have interests directly related to issues in computational linguistics, including our strong artificial intelligence group. Despite this, it was easy to overlook if one was considering only an individual department. We thus set up a site[7] to improve the visibility of our CL-related faculty and research across the university. There are plans to create an actual program spanning the various departments and drawing on the strengths of UT Austin's language departments. For now, the web site is a low-cost and low-effort but effective starting point.

As part of these efforts, we are working to integrate our course offerings, including the cross-listing of the undergraduate *NLP* course. Our students regularly take Machine Learning and other courses from the computer science department. Ray Mooney will teach a graduate *NLP* course in Fall 2008 that will offer students a different perspective and we hope that it will drum up further interest

[6] For a detailed account, see: `http://comp.ling.utexas.edu/wiki/doku.php/austin_compling_history`

[7] `http://comp.ling.utexas.edu`

in CL in the computer science department and thus lead to further interest in our other courses.

As part of the web page, we also created a wiki.[8] We have already mentioned its use in teaching and tutorials. Other uses include lab information, a repository of programming tips and tricks, list of important NLP papers, collaboration areas for projects, and general information about computational linguistics. We see the wiki as an important repository of knowledge that will accumulate over time and continue to benefit us and our students as it grows. It simplifies our job since we answer many student questions on the wiki: when questions get asked again, we just point to the relevant page.

5 Conclusion

Our experience as computational linguists teaching and doing research in a linguistics department at a large university has given us ample opportunity to learn a number of general lessons for teaching computational linguistics to a diverse audience.

The main lesson is to stratify courses according to the backgrounds different populations of students have with respect to programming and formal thinking. A key component of this is to make expectations about the level of technical difficulty of a course clear before the start of classes and restate this information on the first day of class. This is important not only to ensure students do not take too challenging a course: other reasons include (a) reassuring programming-wary students that a course will introduce them to programming gently, (b) ensuring that programming-savvy students know when there will be little programming involved or formal problem solving they are likely to have already acquired, and (c) providing awareness of other courses students may be more interested in right away or after they have completed the current course.

Another key lesson we have learned is that the formal categorization of a course within a university course schedule and departmental degree program are massive factors in enrollment, both at the undergraduate and graduate level. Computational linguistics is rarely a required course, but when taught in a liberal arts college it can easily satisify undergraduate math and/or science requirements (as *Language and Computers* does at OSU and UT Austin, respectively). However, for highly technical courses taught in a liberal arts college (e.g., *Natural Language Processing*) it is useful to cross-list them with computer science or related areas in order to ensure that the appropriate student population is reached. At the graduate level, it is also important to provide structure and context for each course. We are now coordinating with Ray Mooney to define a core set of computational linguistics courses that we offer regularly and can suggest to incoming graduate students. This will not be part of a formal degree program per se, but will provide necessary structure for students to progress through either the linguistics or computer science program in a timely fashion while taking courses relevant to their research interests.

One of the big questions that hovers over nearly all discussions of teaching computational linguistics is: how do we teach the computer science to the linguistics students and teach the linguistics to the computer science students? Or, rather, the question is how to teach both groups *computational linguistics*. This involves getting students to understand the importance of a strong formal basis, ranging from understanding what a tight syntax-semantics interface really means to how machine learning models relate to questions of actual language acquisition to how corpus data can or should inform linguistic analyses. It also involves revealing the creativity and complexity of language to students who think it should be easy to deal with. And it involves showing linguistics students how familiar concepts from linguistics translate to technical questions (for example, addressing agreement using feature logics), and showing computer science students how familiar friends like finite-state automata and dynamic programming are crucial for analyzing natural language phenomena and managing complexity and ambiguity. The key is to target the courses so that the background needs of each type of student can be met appropriately without needing to skimp on linguistic or computational complexity for those who are ready to learn about it.

Acknowledgments. We would like to thank Hans Boas, Bob Harms, Ray Mooney, Elias Ponvert, Tony Woodbury, and the anonymous reviewers for their help and feedback.

[8] http://comp.ling.utexas.edu/wiki/doku.php

References

Jason Baldridge, Sudipta Chatterjee, Alexis Palmer, and Ben Wing. 2007. DotCCG and VisCCG: Wiki and programming paradigms for improved grammar engineering with OpenCCG. In *Proceeings of the GEAF 2007 Workshop*.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Publications.

Steven Bird, Ewan Klein, Edward Loper, and Jason Baldridge. 2008. Multidisciplinary instruction with the Natural Language Toolkit. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*. Association for Computational Linguistics.

C. Brew, M. Dickinson, and W. D. Meurers. 2005. Language and computers: Creating an introduction for a general undergraduate audience. In *Proceedings of the Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing And Computational Linguistics*, Ann Arbor, Michigan.

Jason Eisner. 2002. An interactive spreadsheet for teaching the forward-backward algorithm. In Dragomir Radev and Chris Brew, editors, *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, pages 10–18.

Katrin Erk and Sebastian Pado. 2006. Shalmaneser – a flexible toolbox for semantic role assignment. In *Proceedings of LREC-2006*, Genoa, Italy.

D. Jurafsky and J. H. Martin. 2000. *Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, Upper Saddle River, NJ.

Edward Loper and Steven Bird. 2002. NLTK: The natural language toolkit. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 62–69. Somerset, NJ: Association for Computational Linguistics.

Taesun Moon and Jason Baldridge. 2007. Part-of-speech tagging for middle English through alignment and projection of parallel diachronic texts. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 390–399.

Taesun Moon and Katrin Erk. 2008. Minimally supervised lemmatization scheme induction through bilingual parallel corpora. In *Proceedings of the International Conference on Global Interoperability for Language Resources*.

Elias Ponvert. 2007. Inducing Combinatory Categorial Grammars with genetic algorithms. In *Proceedings of the ACL 2007 Student Research Workshop*, pages 7–12, Prague, Czech Republic, June. Association for Computational Linguistics.

Stuart Robinson, Greg Aumann, and Steven Bird. 2007. Managing fieldwork data with Toolbox and the Natural Language Toolkit. *Language Documentation and Conservation*, 1:44–57.