Declarative Syntactic Processing of Natural Language Using Concurrent Constraint Programming and Probabilistic Dependency Modeling

Irene Langkilde-Geary
Natural Language Technology Group
University of Brighton
Brighton BN2 4GJ, UK
ilg10@brighton.ac.uk

Abstract

This paper describes a declarative approach to parsing and realization of natural language using a probabilistic dependency model of syntax within a constrained optimization framework. Such an approach is particularly well-suited for applications like machine translation. The paper describes a test-of-concept implementation applied to the classic sentence "Time flies like an arrow." and discusses the further research necessary for scaling up to general broad-coverage processing of language.

1 Introduction

Interest has grown recently in using syntactically-informed statistical models to improve the quality of machine translation (MT) output (eg., (Charniak et al., 2003; Och et al., 2004; Galley et al., 2006; Hassan et al., 2007; Chang and Toutanova, 2006)), in part because syntactic generalizations can help overcome sparse data problems that are inherent in natural language processing. (See (Koehn and Hoang, 2007) for a sample of some more detailed motivations.).

However, leveraging syntactic information poses several challenges. For one, a system needs syntactic information about the subphrases involved in a translation before a statistical model of syntax can be applied to help generate fluent output. Such information can be expensive and/or time-consuming to obtain (especially if manual annotation were involved), or available only in approximate forms (such as via an automatic parser). Another challenge can be the need for consistent or compatible syntactic representations across different components or stages of processing—in particular, across the analysis versus realization components for the target language.

Yet another challenge is the increased model complexity inherent in richer representations that include syntax. The size of the model space may increase both from additional attributes associated with individual words (or phrases) as well as from additional statistical dependencies such as a parent or head, besides adjacent neighboring words. The combinatorial explosion that occurs with higher dimensional model spaces can easily become intractable for any algorithm that operates on tuples of three or more word-and-attribute structures.

Finally, the high degree of interdependence between subtasks such as lexical choice, choice of syntactic structure, and word order, etc., poses a challenge for organizing processing steps. Although a simple pipeline architecture has most commonly been used for generation systems, one study of 19 systems (Cahill et al., 1999) found that no two systems implemented exactly the same pipeline. As a system scales up to broad coverage of vocabulary and syntax (as it must in translation of newspaper texts), a pipeline-style architecture becomes less and less tenable because of the degradations in output quality that result from separating inter-related decisions and ignoring the full range of statistical dependencies.

A long history of work within the generation community on system architectures (including (Cahill et

al., 2000; Calder et al., 1999; Beale et al., 1998; Beale, 1997; Elhadad et al., 1997; Smedt et al., 1996; Robin, 1994; Meteer, 1990)) as well as our own previous work on realization for MT suggests to us that fine-grained declarativeness is necessary for any general solution to the problem of broad-coverage, high-quality language realization. Declarativeness means that the language representation is stateless and the computation mechanisms impose no artificial restrictions on the sequential order of processing steps. Instead, the flow of processing is implicitly directed by the propagation of inferences given an input and a set of relations/constraints. Fine-grained means that constraints on attributes and dependencies are as unbundled from each other as possible, so that they can be handled independently. Finer granularity inherently enhances declarativeness.

A declarative approach not only addresses the challenge of subtask interdependence, it can simultaneously solve the problem of obtaining syntactic analyses for training syntactically-informed MT systems, by the very nature of being declarative. The syntactic analyses thus obtained are also naturally compatible with the task of realization. Declarativeness offers a further advantage of genericity, making the approach potentially applicable to a wide range of applications and domains beyond MT.

Declarative representations and computation mechanisms have been the subject of much research in the fields of both Linguistics and Computer Science over the last 30-40 years, at times motivating each other but also sometimes developing independently. Currently some of the most popular linguistic theories of research include HPSG (hpsg.stanford.edu), LFG (www.essex.ac.uk/linguistics/LFG/), TAG (www.cis.upenn.edu/ xtag/home.html), and CCG (groups.inf.ed.ac.uk/ccg/). Older work on declarative linguistic processing mechanisms was purely symbolic, and had a reputation for being brittle and slow. More recent work incorporates statistical modeling—improving robustness—but still tends to be aimed at parsing more strongly than realization, resulting in a lack of complete declarativeness. For example, TAG and CCG explicitly model derivation order, while the phrase structure grammars traditionally employed by HPSG and LFG encode more implicit restrictions on derivation order.

Furthermore, in practice some existing broad-coverage realizers based on declarative theories programmatically break processing into stages (usually to improve efficiency), thereby imposing a greater sequential order on processing. For example, (White, 2006; Carroll and Oepen, 2005) delay the realization of free-er order elements like modifiers and conjuncts to avoid wasted evaluation of all possible permutations of constituent order within incomplete phrases. However, this makes certain other decisions harder, such as choosing the best lexemes to express a predicate and its dependent when the dependent might be best realized as an argument with some lexemes but as an adjunct with others.

In the last 15 years, the field of Computer Science has made some interesting and significant advances in the context of developing modeling languages for efficiently solving large combinatorial problems, giving rise to a sub-area called Constraint Programming (Dechter, 2003; Apt, 2003; Marriott and Stuckey, 1998; Rossi, 2006). However, these advances do not yet seem to have become widely known within the Natural Language Processing community, where statistical approaches have been blossoming. Constraint Programming (CP) is a general-purpose methodology with strong theoretical foundations whose declarativeness is enhanced by concurrency (Roy and Haridi, 2004; Sangiorgi and Walker, 2001). It is designed for integrating heterogeneous constraint solving mechanisms, with roots in Artifical Intelligence constraint satisfaction and Operations Research optimization techniques. In contrast, theoretical linguistic formalisms traditionally used only a single kind of constraint solver—unification.

Another important point of difference is in the nature of how constraints are processed. While unification-based approaches tend to passively test-then-generate (or even weaker, generate-then-test) value assignments for variables during a search procedure, hard constraint solvers in a constraint programming system actively propagate inferences to a stronger level of consistency between each step of search. Active propagation tends to dramatically reduce the size of the search space, leading to more efficient and powerful processing. See (Blache, 2000) for further elaboration on the difference between ac-

tive versus passive use of constraints in linguistic processing.

One actively emerging area of current research in both linguistics and computer science is combining hard constraints (ie. logic-oriented reasoning) with soft preferences, including probabilistic reasoning. In a previous exploratory paper (Langkilde-Geary, 2005), we proposed a novel optimization approach to declarative syntactic language processing that applied probabilistic modeling within the framework of Concurrent Constraint Programming (CCP). In this paper we extend and refine that work, giving stronger evidence of the potential of this solution. Our results are still preliminary, however, as the computationally hard nature of this problem and the ambitiousness of the proposed solution necessitate solving multiple challenging subproblems before a full-scale evaluation can be performed.

The paper is organized as follows: we first describe our refined formulation of parsing/generation as a constraint program that optimizes sentence probability in Section 2. Then in Section 3 we discuss how the program performs on the classic input sentence, illustrating the synergy of combining logical and probabilistic reasoning and the flexibility of the approach. Finally, Section 4 concludes.

2 Constraint Program Formulation

A problem in the framework of CCP is represented as constraints on sets of variables. A constraint program consists of a declarative specification of these variables and contraints, together with a search strategy. One basic kind of constraint specifies a set of possible domain values for a variable. Non-basic constraints are associated with specialized solvers that incrementally infer additional domain restrictions.

The general methodology employed by CCP first allows the solvers to execute until no further inferences can be drawn about the variable domains. When propagation has stabilized, the problem space is split into two or more complimentary sub-cases. The reduced domains within each subproblem may make new inferences possible, triggering additional propagation. Search again waits until stability before performing a new split. Thus, propagation and search are iteratively applied. The repeated splitting of the search space defines a search tree.

A search strategy specifies exactly how a problem space is split into subcases at each new step of search and in which order the subcases are to be explored. Search stops when one or all solutions of interest have been found. A solution is defined as a complete assignment of values to variables that is consistent with all the constraints.

Constraint programming consists of a language for expressing the specifications of variables, constraints, and strategies. The high-level of abstraction employed makes it easy describe and experiment with different models. Our work uses the Mozart/Oz programming language environment, which provides built-in constructs for concurrent constraint programming as well as support for new user-defined constraints. Mozart/Oz is a general-purpose programming language that has pioneered the development of light-weight data-flow concurrency. Data-flow concurrency synchronizes on the determination of logic (non-mutable) variables so that the processing of constraints is implicitly input driven and declarative. CCP in Mozart/Oz can be viewed as an implementation of a whiteboard architecture.

Our CCP formulation of natural language syntactic processing makes use of the dependency-style corpus described in (Langkilde-Geary and Betteridge, 2006). This corpus is derived from the Penn Treebank (Marcus et al., 1993) and designed especially for realization. It represents a sentence as a sequence of word tokens where each word is associated with a set of more than 50 linguistically-motivated features, one of which is a head relation. It is flatter and more regular than the original annotation and other similar corpora, in part because of its choice of head words for modeling dependencies. The quality of the corpus has in a certain sense been validated through experiments in which almost fully-specified inputs constructed from the corpus were regenerated using the HALogen system, producing exact matches with the original sentence a high percentage of the time. These characteristics are its main advantages for our purposes.

2.1 Variables

In our test-of-concept implementation, we restrict our attention to just a handful of features per word

FEATURE	VALUE
ID	a word id
HeadID	id of head word
Token	inflected word form
Role	syntactic function
Group type (GT)	clause, np, other
Direction (DIR)	+/- from head
Relative position (RP)	tree distance from head
Absolute position (AP)	distance from start of sent

Table 1: Word features

plus some needed auxiliary variables. Table 1 summarizes the main features associated with each word that we use in this paper. They are node ID, token, functional role, group type, direction, relative position, and absolute position. In addition, each node is associated with a set of variables representing its head, consisting of the same set of features.

The **ID** is an arbitrary number associated with a word, and is used together with the **HeadID** feature to represent the dependency structure of a sentence. The value of the ID has no linguistic significance, and we assume that it is internally assigned during initialization. Each word has exactly one head. The top word in the sentence is a special case, defined to have a null node as its head.

The **role** feature represents the functional relationship of a dependent with respect to its head. It is derived from both functional roles and part-of-speech tags in the Penn Treebank. The **group-type** (gt) feature is a generalization of constituent-style non-terminal labels associated with the head word of the constituent. It distinguishes between just three coarse categories: clause, noun phrase (np), and other.

The **direction** (dir) feature indicates whether a dependent comes to the left or right of its head. It is partially redundant with the relative position feature, but useful as a generalization of it. The **relative position** (rp) indicates that a word is the nth dependent attached to one side of a head with the sign indicating which side. The value used for *n* is actually offset by 1000 to keep the domain positive as required by Mozart. Finally, **absolute position** (ap) designates the linear order of words in a sentence, with the first word of the sentence assigned position 1.

2.2 Constraints

The basic symbolic constraints that enumerate the domains of each variable are defined according to Table 1. Our implementation limits the domains of the relative position and absolute position features according to the number of nodes given in the input. To be precise, each relative position variable is constrained to the range 1000-NumNodes..1000+NumNodes, and the absolute position is constrained to be between 1 and NumNodes. The token feature in our implementation has 39067 possible values (drawn from the first 22 sections of the PennTreebank), including a special token for unseen words. For implementation in Mozart, symbolic domain values are arbitrarily mapped to integers.

Besides domain constraints for each variable, there are constraints that define the relationships between the features of words, the tree structure of a sentence and the probabilistic score of the sentence as a whole. They are as follows, starting with the simplest to explain.

2.2.1 Definitional Constraints

The definitional constraints define the representation itself. For example, the direction feature is constrained to have the value '-' if it occurs to the left of its head in the linear order of words in the sentence. In other words, it is '-' if its absolute position is less than the absolute position of its head. Also, by definition, the relative position features must have a value strictly less than 1000 iff the direction is '-'. Note that the direction and relative position features are partially redundant logically. This is on purpose, to facilitate generalization in the face of sparse data for the probabilistic model.

Other definitional constraints are that the top node in the sentence has the role 'top', and a relative position of exactly 1000, and that the relative positions of nodes with the same head must all be distinct and sequential, except skipping the value 1000. The absolute positions of all nodes must also be distinct. The relative positions are constrained to correlate with the absolute positions, such that among nodes with the same head, smaller relative positions are associated with smaller absolute positions, and vice versa.

2.2.2 Tree Structure Constraints

Another group of constraints define the projective tree structure between the words in the sentence and their heads. We implement this using Mozart's set constraints. We define several set variables over node IDs to do this: ancestors, directdeps, leftdirectdeps, rightdirectdeps, and descendents. The set of ancestors and set of descendents are disjoint (ie. no head cycles), with the ancestors defined to be a node's head unioned with the ancestors of the head's head; while the descendents are defined as a node's directdeps unioned with each direct dependent's descendents, with the subsets actually forming a partition among a node's descendents. A node's directdeps consist of unioning the disjoint leftdirectdeps and rightdirectdeps, where the left- and rightdirectdeps are defined by the direction and head id features described earlier. Finally, the set of absolute positions associated with the descendents of a node together with the absolute position of a node itself must form a convex set, or in other words, be a continuous sequence of integers.

Also, as alluded to earlier, each node is associated with a set of variables representing a head node. A selection constraint enforces that the head node variables must eventually unify with a word node in the sentence or the null-head node. A sentence is constrained to have one node with a null head. The descendents of the null head node include all the word nodes in the sentence.

2.2.3 Probabilistic Dependency Constraints

With each of the features *grouptype*, *position direction*, *relative position*, *role* and *token* of every word node we associate a positive conditional log probability score. This score is equal to -logprob(Feature | history). The *history* in this context refers to features associated with the structural head of a node and other features of the same node upon which the *Feature* depends statistically. These feature scores are summed to compute a likelihood score for a whole sentence. The score for the whole sentence is interpreted as a cost to be minimized in searching for an optimal solution. Since Mozart currently only allows constraint variable domains that are integers, the log probability scores are multiplied by 10000, rounded, and then truncated to integers.

The conditioning (history) features of each feature f in a node are assumed to be as shown in Table 2. Our implementation waits for the head features to be determined before computing a feature's score, but does not wait for the interdependent "self" features. Once f and the head features it depends on are determined, only the "self" features that have been previously determined are used for conditioning, iff f depends on it according to Table 2. The "self" dependencies listed in the table are therefore symmetric, so that "self" features determined later are conditioned on ones determined earlier, according to the chain rule for probabilities.

Feature	*Interdependencies*
group type	head: gt; self: role, dir, rp, token
role	head: role, gt; self: gt, dir, rp, token
position direction	self: rp, role, gt
relative position	self: dir, role, gt
token	head: gt; self: gt, role

Table 2: Statistical Feature Dependencies

For example, suppose that for a particular node its token and role features are given in the input (as it is in two of the experiments described later), and that the three other statistical features are determined in this order: position direction, relative position, and group type. Then the score for that node is computed as

```
NodeScore
  = FeatScore(role) + FeatScore(pd)
    + FeatScore(rp) + FeatScore(gt)

  = logprob(role) + logprob(pd|role)
    + logprob(rp | role, pd, h_gt)
    + logprob (gt | h_gt, role, pd, rp)
```

The probabilities are computed using raw (unsmoothed) empirical frequencies extracted from the (Langkilde-Geary and Betteridge, 2006) corpus. By lucky coincidence, the probabilities needed to process the sample sentence illustrated in this paper given the dependencies assumed above were all nonzero. However, sparse data problems prevent an immediate larger-scale evaluation on more sentences.

2.2.4 Probabilistic Modeling Issues

Dynamic conditioning is important for enabling truly declarative processing. (In future work, we intend to investigate how to avoid waiting for the relevant head features, to further improve the declarativeness of the approach.) To our knowledge,

all existing work in NLP uses static conditioning. Although dynamic conditioning can sometimes be simulated with a statically trained model either by marginalizing, or in conjunction with dynamic programming techniques, or by training multiple sets of models—one for each possible dynamic decomposition, these approaches can all require exponential amount of work, both in theory and in practice. We do not believe that any of them would ultimately be very satisfactory for the problem in this paper.

One not-uncommon alternative to an exact simulation is to approximate by substituting the desired probability with a statically-smoothed probability using a somewhat different conditioning context. There are various options for how to choose a substitute context. However, dynamic conditioning is used so extensively in our approach that an approximation is also unlikely to be satisfactory.

The ideal approach with respect to accuracy is to perform smoothing dynamically. However, this poses a different set of challenges: it has been studied relatively little to date and is very much an unsolved problem; it is also inherently slower than a simple look-up. We believe these challenges can be overcome, and so we are currently in progress of developing approach along these lines.

2.3 Search Algorithm and Case-Splitting Strategy

In Constraint Programming the shape of the search space can be specified separately from the order in which subparts of the space are searched. Orthogonally, one can also specify whether to search for one, all, or a best solution. Our approach in this paper is to perform a best-solution search using Mozart's built-in branch-and-bound algorithm. This algorithm injects a new constraint into the problem space after each intermediate solution is found that requires any further solutions to have an equal or better score than the current solution. (An intermediate solution is any complete assignment of values to variables arrived at during search that satisfies all the hard constraints.) The last solution found is then the overall best solution returned by the algorithm.

For case-splitting, we define a two-stage strategy. Before the first intermediate solution has been found, we split on the variable whose two most likely values have the greatest difference in likelihood. After that, the search space is distributed according to the variable with the greatest number of suspensions waiting on it. Ties are broken with the likelihood difference, as in the first stage. The motivation for this hybrid strategy is to facilitate the calculation of features scores, which may be waiting on one or more variables before they can be calculated. Earlier calculation of scores enables greater pruning of the search space. Note that since the absolute position and head id features can be determined from the others, they are not considered for distribution.

Token	Role	GT	AP	RP	Dir	ID	Head
time	sbj	np	1	-1	-	5	2
flies	top	c	2	0	+	2	1
like	rma	o	3	-2	-	7	4
an	det	o	4	-1	-	6	4
arrow	ajt	np	5	1	+	4	2
.	rpunc	o	6	2	+	3	2

Table 3: Solution Sentence

3 Experimental Runs

We experimented with our implementation using the classic sentence "Time flies like an arrow." The complete solution for this sentence with respect to the features we are using is shown in Table 3. Note that there is no element of our program that is tied/hardwired to this particular input.

Table 4 describes three runs of the program demonstrating the declarativeness of our approach and the computational benefits of constraint propagation. The first run performs realization, and the last two do parsing. The first parsing experiment (which is the second run) provides each word's syntactic role in the input. The second parsing experiment (the third run) is more realistic, requiring the program to solve for each word's syntactic role. The first two experiments (realization and parsing-given-roles) are comparable in difficulty, while the third is clearly harder.

The second column reports the size of the search tree in terms of search nodes. For the sake of comparison, the third column shows the size of search achieved in our previous work, while the raw size of the total search space for the main variables of interest is estimated in the fourth column. The latter is calculated simply by multiplying the initial domain sizes of the unknown variables listed in the

Given VS. Unknown	Search Tree Size	Prev. Search Tree Size	Total Search Tree Size	Depth	Num Sols	Best Sol	Num Vars	% Correct
token, role, head VS. dir, rp, ap, gt	70	194	2376	8	1	8	24	100.0
token, role, ap VS. dir, rp, head, gt	71	259	2772	9	2	12	24	100.0
token, ap VS. role, dir, rp, head, gt	5485	na	63756	16	2	17	30	96.7

Table 4: Experimental Runs

first column. These columns show that the refined set of contraints we applied in this paper reduced the search by 64-73% compared to the constraints used in our previous work, and by 91-97% when compared to the size of the entire search space. (Our previous work did not use tokens at all for conditioning any probability scores, and thus could not have performed well at all on the second parsing experiment.) We expect that constraint propagation can be considerably strengthed for the third experiment, and plan to work on this further in the future.

The Depth column is the depth of the search tree. The next column shows the number of intermediate solutions found during the branch-and-bound search. The last "solution" is always the best one. The "Best Sol" column reports how many search tree nodes were explored before the best solution was uncovered. It is especially noteworthy that in all three runs, the best solution was uncovered remarkably early. (The minimum is six, since there are six tokens in the sentence.)

The last column indicates whether the "best" solution matches our desired solution. The percentage shown is calculated based on the number of variables whose values were initially unknown. This was six words times four attributes in the first two experiments, for example. That column shows that our program arrived at the correct solution for the first two experiments when given the role as input, and very nearly obtained the correct solution even without the role and given only the limited set of attributes we used in this test.

The sole error was in labeling "time" as an adjunct rather than as the subject of the sentence. However, the program correctly identified "time" as a noun phrase, and "flies" as its head. Upon further analysis of the corpus, the error can seem reasonable given the available conditioning features. In the Penn Treebank the word "time" heads an adjunct much more often than it heads a subject, even when occuring directly adjacent and left of its head. For example, the Treebank contains the following similar sentences: "Mr. Petrie or his company have been accumulating Deb Shops stock for several years, each time issuing a similar regulatory statement." Also, "The last time the S&P 500 yield dropped was ...."; and "CalTrans began working on a second round ..., this time wrapping freeway columns....". We expect that a model augmented with additional features should be able to resolve this error and correctly solve the parse of the sentence.

Table 5 shows the variables chosen for distribution along the search path to the best solution for each run of the program. Variables not shown in this table had their value determined via constraint propagation, rather than search. The Rel column indicates whether the selected variable was associated with the word node or its head. The "F" column indicates which feature the variable represented.

The "Vals" column shows in order of decreasing likelihood the set of possible values for the chosen variable at that point in time. Values from the original domain not listed were eliminated by that point via constraint-based inference. Note that the ordering of values reflects their context-dependent probabilities, which takes into account dependencies on previously determined variables (whether determined via inference or via selection during case-splitting). The"V" column indicates which path in the search tree the solution was on by listing the value assigned to the chosen variable in the current sub-case.

Finally, the Diff column contains the likelihood difference between the first and second value possibilities. The large differences between the first and second values for most search steps is striking. There is often a clear preference for the top value over the others. Note, however, that the top value is not always correct. For example, in step 3 of the second run, the probability model first postulates that

the head of arrow has a grouptype of “other”, although it ultimately settles on the correct grouptype of “clause”.

	Token	Rel	F	Vals	V	Diff
	Run 1					
1	time	head	gt	c, np, o	c	0.99
2	an	head	gt	np, o	np	0.99
3	time	self	dir	-, +	-	0.93
4	time	self	gt	np, o	np	0.62
5	arrow	self	dir	+, -	+	0.60
6	arrow	self	rp	1, 2	1	0.58
7	an	self	rp	1, 2	1	0.22
	Run 2					
1	time	head	gt	c, np, o	c	0.99
2	time	self	gt	np, o	np	0.62
3	arrow	head	gt	o, c	c	0.55
4	an	head	gt	np, o	np	0.99
5	arrow	self	rp	1, 2	1	0.58
6	.	head	gt	o, np, c	not o	0.43
7	.	head	gt	c, np	c	0.93
8	like	head	gt	o, np	np	0.43
	Run 3					
1	an	self	role	det, obj, ajt, sbj, clr, rma, lp	det	0.99
2	flies	self	dir	+, -	+	0.82
3	time	self	role	ajt, sbj, rma, tpc	ajt	0.75
4	like	self	role	rma, ajt, top, tpc, prd, clr, sbj	rma	0.72
5	arrow	head	gt	o, c	c	0.55
6	an	head	gt	np, o	np	0.99
7	arrow	self	rp	1, 2	1	0.64
8	an	head	role	ajt, sbj	ajt	0.49
9	.	head	gt	o, np, c	not o	0.43
10	.	head	gt	c, np	c	0.93
11	like	head	gt	o, np	np	0.43

Table 5: Search path to best solution for each run

4 Conclusion

This paper argues that a declarative approach is probably the most appropriate way to address the challenges of high-quality realization for MT, and provides additional evidence for the merits of an approach based on CCP combined with probabilistic modeling. The evidence is still preliminary, because such an ambitious project poses several challenging subproblems that will take time to address before a full-scale evaluation can be performed. However, this work illustrates the flexibility of the approach and has demonstrated some significant computational benefits resulting from the synergy.

A statistical training algorithm designed to learn weights/smooth probabilities and avoid overfitting cannot learn from data alone to distinguish between infrequent versus impossible events (when the population space of events is not finite). At best it can approximate the distinction via regularization, which limits the amount of weight/probability that can be retained by seen events or redistributed to unseen events according to an estimate of the variance in a sample of data. The reliability of any such approximation is directly proportional to the size of the sample/corpus used for obtaining the estimate.

In contrast, hard constraints can easily eliminate impossible values and vastly reduce the space of possibilities that must be considered. The reduction is more dramatic the more that a model is factored into attributes that are at least somewhat correlated. Factoring enhances the ability of a model to generalize in the presence of sparse data, but loses information about invalid or impossible combinations of attributes, thereby spuriously increasing model size and complexity. Hard constraints can be applied to reduce or eliminate this spurious increase in complexity and thereby improve the tractability of probabilistic modeling within rich feature spaces.

We are not aware of very many people in NLP taking advantage of the heterogeneity and stronger propogation offered by CP beyond the following: Denys Duchier et al., Claire Gardent et al., Tomasz Marciniak et al., and Dan Roth et al. The first also uses Mozart/Oz like we do, but focuses on solely hard constraints. The second is purely symbolic as well. The latter two actually apply integer linear programming (ILP), which can be viewed as a special case of CP involving only variables with integer domains. Though theoretically just as powerful as CP, ILP can be less natural to use since all domains must be mapped to integers. ILP also lacks the high-level modeling language interface that CP offers. The ability to combine complimentary techniques (especially hard and soft constraints) and tackle larger problems that are intractable for simpler methods without resorting to adhoc pipelines that sacrifice optimality is the main reason CP holds a lot of promise for NLP, and we hope it becomes more widely adopted.

References

K. Apt. 2003. *Constraint Programming*. Cambridge University Press.

S. Beale, S. Nirenburg, E. Viegasy, and L. Wanner. 1998. De-constraining text generation. In *Proc. INLG*.

S. Beale. 1997. *Hunter-Gatherer: applying constraint satisfaction, branch-and-bound and solution synthesis to computational semantics*. Ph.D. thesis, Carnegie Mellon University.

P. Blache. 2000. Constraints, linguistic theories and natural language processing. *Natural Language Processing*, 1835.

L. Cahill, C. Doran, R. Evans, C. Mellish, D. Paiva, M. Reape, D. Scott, and N. Tipper. 1999. In search of a reference architecture for NLG systems. In *Proc. EWNLG*.

L. Cahill, C. Doran, R. Evans, R. Kibble, C. Mellish, D. Paiva, M. Reape, D. Scott, and N. Tipper. 2000. Enabling resource sharing in language generation: an abstract reference architecture. In *Proc. LREC*.

J. Calder, R. Evans, C. Mellish, and M. Reape. 1999. "free choice" and templates: how to get both at the same time. In *Proc. KI Workshop*.

J. Carroll and S. Oepen. 2005. High efficiency realization for a wide-coverage unification grammar. In R. Dale and K-F. Wong, editors, *Proc. IJCNLP*, volume 3651. Springer LNAI.

P. Chang and K. Toutanova. 2006. A discriminative syntactic word order model for machine translation. In *Proc. ACL*.

E. Charniak, K. Knight, and K. Yamada. 2003. Syntax-based language models for machine translation. In *Proc. MT Summit IX*.

R. Dechter. 2003. *Constraint Processing*. Morgan Kaufmann.

P. Dienes, A. Koller, and M. Kuhlmann. 2003. Statistical a-star dependency parsing. In *Prospects and Advances in the Syntax/Semantics Interface*.

D. Duchier. 2003. Configuration of labeled trees under lexicalized constraints and principles. *Journal of Research on Language and Computation*.

M. Elhadad, J. Robin, and K. McKeown. 1997. Floating constraints in lexical choice. *Computational Linguistics*, 23(2).

M. Galley, J. Graehl, K. Knight, D. Marcu, S. DeNeefe, W. Wang, and I. Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proc. ACL*.

H. Hassan, K. Sima'an, and A. Way. 2007. Supertagged phrase-based statistical machine translation. In *Proc. ACL*.

P. Koehn and H. Hoang. 2007. Factored translation models. In *Proc. EMNLP*.

I. Langkilde-Geary and J. Betteridge. 2006. A factored functional dependency transformation of the english penn treebank for probabilistic surface generation. In *Proc. LREC*.

I. Langkilde-Geary. 2005. An exploratory application of constraint optimization in mozart to probabilistic natural language processing. In H. Christiansen, P. Skadhauge, and J. Villadsen, editors, *Proceedings of the International Workshop on Constraint Solving and Language Processing (CSLP)*, volume 3438. Springer-Verlag LNAI.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of english: the Penn treebank. *Computational Linguistics*, 19(2).

K. Marriott and P. Stuckey. 1998. *Programming with Constraints*. MIT Press.

M. Meteer. 1990. *The Generation Gap - the problem of expressibility in text planning*. Ph.D. thesis, U. of Massachusetts.

F. Och, D. Gildea, S. Khudanpur, A. Sarkar, K. Yamada, A. Fraser, S. Kumar, L. Shen, D. Smith, K. Eng, V. Jain, Z. Jin, and D. Radev. 2004. Final report of johns hopkins 2003 summer workshop on syntax for statistical machine translation. Technical report, John Hopkins University.

J. Robin. 1994. *Revision-based generation of natural language summaries providing historical background: corpus-based analysis, design, implementation and evaluation.* Ph.D. thesis, Columbia University.

F. Rossi. 2006. *Handbook of Constraint Programming*. Elsevier.

P. Van Roy and S. Haridi. 2004. *Concepts, Techniques, and Models of Computer Programming*. MIT Press.

D. Sangiorgi and D. Walker. 2001. *The Pi-calculus: A Theory of Mobile Processes*. Cambridge University Press.

I. Schroder. 2002. *Natural Language Parsing with Graded Constraints*. Ph.D. thesis, University of Hamburg.

K. De Smedt, H. Horacek, and M. Zock. 1996. Architectures for natural language generation. In *Trends in Natural Language Generation*. Springer, Berlin.

M. White. 2006. Ccg chart realization from disjunctive inputs. In *Proc. INLG*.