PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 104 OCTOBER 2015 51–62

Grasp: Randomised Semiring Parsing

Wilker Aziz

Universiteit van Amsterdam

Abstract

We present a suite of algorithms for inference tasks over (finite and infinite) context-free sets. For generality and clarity, we have chosen the framework of *semiring parsing* with support to the most common semirings (e.g. Forest, Viterbi, k-best and Inside). We see parsing from the more general viewpoint of weighted deduction allowing for arbitrary weighted finite-state input and provide implementations of both bottom-up (CKY-inspired) and top-down (Earley-inspired) algorithms. We focus on approximate inference by Monte Carlo methods and provide implementations of *ancestral sampling* and *slice sampling*. In principle, sampling methods can deal with models whose independence assumptions are weaker than what is feasible by standard dynamic programming. We envision applications such as monolingual constituency parsing, synchronous parsing, context-free models of reordering for machine translation, and machine translation decoding.

1. Introduction

Many inference tasks in Natural Language Processing (NLP) involve operations over weighted context-free sets of solutions. Typical examples are constituency parsing and hierarchical Statistical Machine Translation (SMT). These weighted sets represent functions over large spaces of tree-structured solutions. We focus on cases where these functions have a probabilistic interpretation and can be represented by a weighted Context-Free Grammar (CFG). Common inference tasks involve finding the solution which maximises the underlying function (*optimisation*), or the one which minimises an expected loss (*minimum Bayes risk*, MBR), or the one that is marginally optimum. Depending on the complexity of the underlying distribution, these *decision rules* can be hard to compute. Particularly, in its most general form, the marginalisation problem is NP-complete (Sima'an, 1996). The MBR objective can also become

 Corresponding author: w.aziz@uva.nl
Cite as: Wilker Aziz. Grasp: Randomised Semiring Parsing. The Prague Bulletin of Mathematical Linguistics No. 104, 2015, pp. 51–62. doi: 10.1515/pralin-2015-0013.

unwieldy depending on the complexity of the loss function. To all such tasks, Monte Carlo (MC) methods provide sound approximations (often with some guarantees) based on random simulation techniques. We investigate and propose a framework towards a general and flexible MC approach to inference with weighted CFGs.

2. Semiring parsing

A first step in generalising parsing is understanding it as the intersection between the language of a grammar and that of a Finite-State Automaton (FSA) (Bar-Hillel et al., 1961; Billot and Lang, 1989). This insight motivates a generalisation of parsing to arbitrary weighted finite-state input (Nederhof and Satta, 2003; Dyer and Resnik, 2010). In this context, not only the set intersection is computed, but strings in the resulting set are weighted by the product of their input weights. Parsing (or *intersecting*) an automaton finds use in applications where it is natural to represent uncertainty over finite-state input, for instance due to automatic pre-processing.

Another general view of parsing, orthogonal to the previous point, arises from the relationship between parsing and logic deductive systems (Pereira and Warren, 1983). Specifying a parser by means of a deductive system abstracts away from implementation details redirecting attention back to the parsing strategy itself. Shieber et al. (1995) lay down the principles of parsing as deduction and offer an extensive discussion regarding implementation. Arguably their most exciting result is to uncover the relationship between parsers for different grammar formalisms (context-free and beyond) which turn out to share the same (or very closely related) set of deduction rules under perhaps different control mechanisms.

In deductive parsing, a *deduction rule* is a template $\frac{A_1 \dots A_k}{B} C_1 \dots C_j$, where $A_1 \dots A_k$ (called *antecedents*), $C_1 \dots C_j$ (called *side condition*) and B (called *consequent*) are *items*. A rule states that, if the side condition holds and the antecedents have already been inferred, we can infer the consequent. Each possible way to deduce an item from the grammar rules by instantiation of deduction rules is called an *item derivation* and denoted by D. Any given instance x of an item implicitly defines a set D_x of possible ways to infer x. To test membership, the parser needs to prove at least one item derivation (a sequence of instantiation of deduction rules) that leads to a *goal* item form, or to show that no such sequence exists. A parse *forest* can be thought of as the exhaustive instantiation of all item derivations compatible with the grammar and the input.[1]

On top of this item-based description, parsing can be further generalised to comply with an algebraic view of formal languages (Goodman, 1999). In this view, a parser does not simply test for membership, instead, it performs abstract computa-

[1]The word **exhaustive** here does not imply inefficiency. In fact, item derivations are defined recursively which leads to efficient representation. Suppose $\frac{a_1 \dots a_k}{b} c_1 \dots c_j$ an instantiation of a deduction rule and $D_{a_1} \dots D_{a_k}$ sets of item derivations deducing $a_1 \dots a_k$, then $\langle b : D_{a_1}, \dots, D_{a_k} \rangle$ is itself an item derivation. This is very similar to the use of *back-pointers* in chart parsing (Billot and Lang, 1989).

tions under a given *semiring*.[2] Goodman (1999) combines the representational power of deductive systems with the generality of semirings to describe, under a uniform and simple representation, parsers that perform a multitude of tasks from recognition (Boolean semiring) to best-first enumeration (k-best semiring).

In this work, we provide implementation of semiring parsing for CFGs and the following semirings: Boolean and Counting for recognition and counting; Inside used to compute the probability of a string; Viterbi used to compute the probability of the best derivation; 1-best and k-best used to enumerate derivations in best-first order; and Forest used to represent the set of all derivations in some compact manner.

We deploy semiring parsing as deductive programs which intersect an epsilon-free weighted CFG with a weighted deterministic FSA. Because CFGs are closed under intersection with automata (Hopcroft and Ullman, 1979), the result is another CFG. We observe from the state of the deductive program's execution a parse forest which we represent using a hypergraph (Gallo et al., 1993).[3] This is basically our implementation of the Forest semiring. Computing values in the other semirings follows by direct application of the value recursion (Goodman, 1999, Equations 5 and 7) using the corresponding semiring operators (Goodman, 1999, Figure 5). In solving the value recursion, we consult the forest as a data structure which compactly organises items and their derivations.

We present Grasp, a toolkit which incorporates these ideas while facilitating research on sampling methods for structured prediction problems, particularly, parsing and machine translation. In the remainder of this paper we present its design as well as Sample and Slice, two novel semirings which play a central role in delivering sampling algorithms for complex distributions over context-free sets of solutions.

3. Randomised semiring parsing

Our goal is to sample probabilistically from a distribution proportional to $f(\mathbf{d})$ defined over a support $\mathcal{D}_G(\mathbf{x})$, where $\mathbf{x}$ is an input object and $\mathcal{D}_G(\mathbf{x})$ is the derivation forest resulting of intersecting G (a CFG) and an FSA based on $\mathbf{x}$. Moreover, we assume f to factorise as shown in Equation 1, where $\psi(\mathbf{d})$ is some non-negative function of $\mathbf{d}$ which is not assumed to factorise further (we call ψ a *nonlocal* parameterisation), and $\theta(\mathbf{d})$ is some non-negative function of $\mathbf{d}$ compliant with the independence assumptions of G which underlie $\mathcal{D}_G(\mathbf{x})$ (we call θ a *local* parameterisation). Equation

[2] A semiring is an algebraic structure with operations that generalise the arithmetic addition and multiplication. The generalisation extends beyond numerical objects and may be defined over arbitrary sets (e.g. the tree-language of a context-free grammar).

[3] A hypergraph, more specifically a *backward-hypergraph*, is a generalisation of a graph where an edge connects a sequence of tail nodes under a head node and carries a weight. The nodes can be thought of as grouping item derivations, an edge can be thought of as an instantiated deduction rule. Hypergraphs are popular data structures in parsing (Klein and Manning, 2005) and MT (Huang, 2008).

2 illustrates how $\theta(\mathbf{d})$ factorises in terms of the rules used in the derivation $\mathbf{d}$, where r_s is a rule headed by s and θ_{r_s} is the rule's value.

$$f(\mathbf{d}) = \psi(\mathbf{d}) \times \theta(\mathbf{d}) \tag{1}$$

$$= \psi(\mathbf{d}) \times \prod_{r_s \in \mathbf{d}} \theta_{r_s} \tag{2}$$

The nonlocal dependencies introduced by $\psi(\mathbf{d})$ make inference over $f(\mathbf{d})$ a very difficult task. In order to make inference feasible, we resort to the principle of sampling by data augmentation (Tanner and Wong, 1987). In Data Agumentation (DA), we extend the target distribution with auxiliary variables $\mathbf{u}$ that make sampling from the joint $f(\mathbf{d}, \mathbf{u})$ easier than sampling from the original distribution. Simulation then follows by Gibbs sampling, a Markov Chain Monte Carlo (MCMC) technique, whereby we sample each component in turn holding the remaining components fixed. That is, if we draw $\mathbf{u} \sim f(\mathbf{u}|\mathbf{d})$ followed by a draw $\mathbf{d} \sim f(\mathbf{d}|\mathbf{u})$, then the pair $(\mathbf{u}, \mathbf{d})$ is a draw from the joint distribution, and $\mathbf{d}$ is a draw from the target.

The benefit of DA comes from the fact that we can choose the shape of the joint relatively flexibly, and we do so aiming at conditionals $f(\mathbf{u}|\mathbf{d})$ and $f(\mathbf{d}|\mathbf{u})$ that are easy to sample from. A common choice of joint for functions of the form shown in Equation 2 is illustrated in Equation 3,[4] where $\delta_A(x)$ is the Dirac's measure which equals 1 iff $x \in A$. This principle underlies, and can be seen as a multivariate generalisation of, a univariate sampling technique called *slice sampling* (Neal, 2003). With such a choice of joint, we typically call the auxiliary variables *slice variables*. They define a "slice" of the distribution, i.e. a subset of $\mathcal{D}_G(\mathbf{x})$ for which $f(\mathbf{d}, \mathbf{u}) > 0$.

$$f(\mathbf{d}, \mathbf{u}) = \psi(\mathbf{d}) \times \prod_{s : r_s \in \mathbf{d}} \delta_{(0, \theta_{r_s})}(u_s) \times \prod_{s : r_s \notin \mathbf{d}} \phi(u_s; \boldsymbol{\alpha}) \tag{3}$$

First, we make the harmless independence assumption that $f(\mathbf{u}|\mathbf{d}) = \prod_{u_s} f(u_s|\mathbf{d})$, where the auxiliary random vector $\mathbf{u}$ is indexed by items in the derivation forest (nodes in the hypergraph), and f factorises as a product of independent distributions (one for each item). Then, we specify the conditional $f(u_s|\mathbf{d})$ as shown in Equation 4, where ϕ is a probability distribution with parameters $\boldsymbol{\alpha}$. Basically, the conditional is a uniform distribution when the item associated with u_s participates in the conditioning derivation through rule r_s, and it falls back to a given less informed distribution ϕ otherwise.

$$f(u_s|\mathbf{d}) = \begin{cases} \frac{\delta_{(0, \theta_{r_s})}(u_s)}{\theta_{r_s}} & \text{if } r_s \in \mathbf{d} \\ \phi(u_s; \boldsymbol{\alpha}) & \text{otherwise} \end{cases} \tag{4}$$

[4] Errata: after publication, we noticed that we had misrepresented $f(\mathbf{d}, \mathbf{u})$ in Equation 3, a mistake we remedy in this electronic version.

Blunsom and Cohn (2010) devised a special case of this slice sampling technique for binary Inversion Transduction Grammars (ITGs) in the context of Bayesian grammar induction for translation. In their work, s represents a pair of aligned spans, $0 \leq \theta_{r_s} \leq 1$ with $\sum_{r_s} \theta_{r_s} = 1$ for any given s are the parameters of a probabilistic ITG, and there is no $\psi(\mathbf{d})$ (otherwise thought of as $\psi(\mathbf{d}) = 1$). Thus they choose $\phi(u_s; \boldsymbol{\alpha}) = \text{Beta}(u_s; a, b)$, a Beta distribution with shape parameters a and b. We generalise their work to arbitrary weighted CFGs including those parameterised by log-linear models as common in SMT. To do so, we extend s to correspond to an arbitrary item in the derivation forest program, and we choose ϕ in different ways.[5]

The final step in the design of our sampler is to derive what $f(\mathbf{d}|\mathbf{u})$ turns out to be under the conditions above. Equations 5 to 7 are crucial, particularly, the latter implies that every derivation for which some $\theta_{r_s} \leq u_s$ will have zero probability regardless of the assignments of other slice variables. This basically means that $f(\mathbf{d}|\mathbf{u})$ can be represented by a "truncated" derivation forest such that $u_s < \theta_{r_s}$ for every r_s. We can interpret the slice variables as random thresholds on the values associated with item derivations of a given item.

$$f(\mathbf{d}|\mathbf{u}) \propto f(\mathbf{d}) \times f(\mathbf{u}|\mathbf{d}) \tag{5}$$

$$= \psi(\mathbf{d}) \times \prod_{r_s} \theta_{r_s} \times \prod_{u_s : r_s \in \mathbf{d}} \frac{\delta_{(0,\theta_{r_s})}(u_s)}{\theta_{r_s}} \times \prod_{u_s : r_s \notin \mathbf{d}} \phi(u_s; \boldsymbol{\alpha}) \tag{6}$$

$$\propto \psi(\mathbf{d}) \times \prod_{r_s} \frac{\delta_{(0,\theta_{r_s})}(u_s)}{\phi(u_s; \boldsymbol{\alpha})} \tag{7}$$

Obviously, because $\psi(\mathbf{d})$ breaks context-free independence assumptions, sampling from $f(\mathbf{d}|\mathbf{u})$ (Equation 7) might remain challenging. On the one hand, if $\psi(\mathbf{d})$ makes a low-order Markov assumption, exactly rescoring the slice might be possible, particularly, if slices are sufficiently thin. On the other hand, thin slices reduce the mobility of the sampler increasing autocorrelation. Moreover, if slices are deterministically too thin, we may risk trapping the sampler to a subset of the state space, which would compromise ergodicity. A more general solution is not to assume that we can sample from $\psi(\mathbf{d})$ exactly, and rely on MCMC techniques instead. One possible technique is to uniformly sample a subset (of some predetermined size) of the slice, and evaluate Equation 7 only for the derivations in that subset.

Finally, we note a connection to semiring parsing. We can define a Sample semiring which is to the Inside semiring as 1-best is to Viterbi. Informally, by modifying the 1-best addition operation so that it samples from the distribution associated with the Inside values of item derivations (instead of maximising over their Viterbi values), we

[5]For example, for log-linear models $\phi(u_s; \boldsymbol{\alpha}) = \text{Exp}(u_s, \lambda)$, an exponential distribution with rate parameter λ. Note that the greater λ, the thicker the slice (the mean of the exponential distribution is λ^{-1}).

```
# Create and source a dedicated virtual environment for Grasp
~$ virtualenv -p python3 ~/envs/grasp; source ~/envs/grasp/bin/activate
# Clone and install kenlm (if you intend to use Grasp's decoder)
(grasp)~$ git clone https://github.com/kpu/kenlm.git; cd kenlm
(grasp)~$ python setup.py install; cd ..
# Install general dependencies
(grasp)~/grasp$ pip install numpy scipy cython
# Clone and install Grasp
(grasp)~$ git clone https://github.com/wilkeraziz/grasp.git; cd grasp
(grasp)~/grasp$ python setup.py install
# A better option for contributors is: python setup.py develop
```

Figure 1. Installing Grasp*.*

get an independent random sample. Similarly, we can define a Slice semiring which is analogous to the Forest semiring, however, it produces a sliced forest. Slice conditions on random assignments of an auxiliary vector based on a previous sample.[6]

4. Grasp

Our toolkit is available on github `https://github.com/wilkeraziz/grasp` under Apache 2.0 license. It is written in *python3* and *cython*, it builds trivially with *setuptools*, and it has very few (rather standard) dependencies. Grasp is primarily developed for Unix-based systems (e.g. Linux and OS X). Figure 1 illustrates how to install Grasp using a dedicated *virtual environment* running *python3*.

Grasp comes with a CFG parser and a hierarchical SMT decoder. Figure 2 illustrates how to run the parser in two different modes. The first command illustrates exact inference (which requires complete parse forests) and outputs the Viterbi derivation, the 100-best derivations (`--kbest 100`), and 1000 independent samples (`--samples 1000`). The second command illustrates slice sampling, where $\phi(u_s; \boldsymbol{\alpha})$ defaults to $\text{Beta}(u_s; 0.1, 1)$. In both cases, `python -m grasph.cfg.parser` invokes the parser; `wsj00` is an example grammar shipped with Grasp, and `output` specifies a directory where output files are stored. The last command illustrates the output directory structure: it contains the independent ancestral samples, the k-best derivations, the Viterbi derivation, the slice samples and a configuration file which documents the experiment (`args.ini`).[7]

[6] An initial derivation can be obtained by sampling from the local model alone. If the grammar is too large, an initial sample can be obtained from a smaller grammar sharing a subset of the nonterminals of the original one (or where a coarse-to-fine mapping exists).

[7] A complete list of features and options can be found at `https://github.com/wilkeraziz/grasp/blob/master/grasp/cfg/README.md`.

```
# Framework: exact inference
(grasp)~/grasp/examples/ptb$ echo -e 'I was given a million dollars .' | \
 python -m grasp.cfg.parser wsj00 output --grammarfmt discodop --start TOP \
 --unkmodel stfd6 --log --viterbi --kbest 100 --samples 1000 --experiment mtm -v
# Framework: slice sampling
(grasp)~/grasp/examples/ptb$ echo -e 'I was given a million dollars .' | \
 python -m grasp.cfg.parser wsj00 output --grammarfmt discodop --start TOP \
 --unkmodel stfd6 --log --samples 1000 --framework slice --experiment mtm -v
# Output directory
(grasp)~/grasp/examples/ptb$ tree --charset=ascii output/mtm/
output/mtm/
|-- ancestral            |-- args.ini       |-- slice
|   |-- derivations      |-- kbest          |   |-- derivations
|   |   |-- 0.gz         |   |-- 0.gz       |   |   |-- 0.gz
|   |-- trees            |-- viterbi        |   |-- trees
|       |-- 0.gz             |-- 0.gz       |       |-- 0.gz
```

Figure 2. Example run of Grasp*'s CFG parser.*

Figure 3 illustrates how to run the hierarchical decoder. The first command illustrates exact inference. Note that with exhaustive forest rescoring, we cannot go beyond a bigram language model and very short sentences. The second command illustrates sliced rescoring: $\phi(u_s; \boldsymbol{\alpha})$ is set to $\text{Exp}(u_s; 1)$ (`--prior const 1`), each slice is uniformly subsampled (`--within uniform`) making small fixed-size batches (`--batch 100`) for rescoring (Equation 7). In both cases, `python -m grasph.cfg.decoder` invokes the decoder; synchronous grammars are stored in `grammars`; `--weights` specify model weights for model components such as rule table (`--rt`), word penalty (`--wp`), arity penalty (`--ap`), and language model (`--lm`) . The output directory structure is very similar to the one produced by the parser.[8] Output files are sorted lists of hypotheses (best-first), as the last command illustrates.

In this section we also provide some preliminary experiments with our proposed sampler in the context of decoding for hierarchical phrase-based (hiero) models (Chiang, 2005). We report on experiments conducted using the BTEC Chinese-English corpus (Takezawa et al., 2002). Grammar extraction follows the approach of Lopez (2007), we use the implementation of Baltescu and Blunsom (2014). We use cdec (Dyer et al., 2010) to train a linear model with MIRA (Cherry and Foster, 2012).[9] We trained mod-

[8] A complete list of features and options can be found at `https://github.com/wilkeraziz/grasp/blob/master/grasp/mt/README.md`.

[9] Such models are not inherently probabilistic, thus we need to artificially scale the parameters returned by the optimiser. In the experiments in this paper, we multiplied the weights by 10 (`--temperature 0.1`).

```
# Framework: exact inference
(grasp)~/grasp/examples/mt$ head -n1 input | python -m grasp.mt.decoder output \
 --grammars grammars --glue-grammar glue --pass-through \
 --rt --wp WordPenalty -0.43429466 --ap Arity -0.43429466 \
 --lm LanguageModel 2 btec.klm2 --weights mira/lm2-p \
 --temperature 0.1 --viterbi --kbest 100 --samples 1000 --experiment mtm -v
# Framework: slice sampling
(grasp)~/grasp/examples/mt$ head -n1 input | python -m grasp.mt.decoder output \
 --grammars grammars --glue-grammar glue --pass-through \
 --rt --wp WordPenalty  -0.43429466 --ap Arity -0.43429466 \
 --lm LanguageModel 3 btec.klm3 --weights mira/lm3-p \
 --temperature 0.1 --temperature0 10 --samples 1000 --framework slice \
 --batch 100 --within uniform  --prior const 1 --experiment mtm -v
(grasp)~/grasp/examples/mt$ zcat < output/mtm/slice/yields/0.gz
# MCMC samples=1000
# estimate          count  derivations  yield
0.601               601    5            a throbbing pain .
0.271               271    5            throbbing pain .
0.065               65     4            is a throbbing pain .
0.032               32     4            is throbbing pain .
0.027               27     2            throbbing pain in my temples .
0.002               2      1            throbbing pain in pain .
0.001               1      1            throbbing pain the pain .
0.001               1      1            throbbing pain hurts my .
```

Figure 3. Example run of Grasp*'s hierarchical SMT decoder.*

els using standard features as well as a bigram/trigram Language Model (LM) component. Language modelling and LM queries are done with lmplz/kenlm (Heafield, 2011; Heafield et al., 2013). Additionally, we trained two locally parameterised models whose LMs were made *stateless*, i.e. n-grams are scored using as much context as available within translation rules, but context information (LM *state*) is discarded at the boundaries of nonterminals. Finally, as a decision rule, Grasp uses an approximation to MBR known as *consensus decoding* (DeNero et al., 2009) based on an empirical distribution estimated from 2,000 samples.

We report BLEU scores (Papineni et al., 2002) from *multibleu*, an implementation distributed with Moses (Koehn et al., 2007). Results are averaged over 3 runs for a random subset comprising of 20% of BTEC's development set.[10] Table 1 compares Grasp's performance to cdec's. The stateless models can be decoded exactly by both cdec (which uses a Viterbi decision rule) and Grasp (which applies consensus decod-

[10]The exact subset is available at the tool's repository on github.

	System	LM	BLEU
1	cdec	2-gram (stateless)	33.6
2	Grasp	2-gram (stateless)	33.85
3	Grasp (batch=100)	2-gram	36.59 ± 0.61
4	Grasp (batch=200)	2-gram	37.47 ± 0.67
5	cdec (pop=200)	2-gram	41.46
6	cdec	3-gram (stateless)	33.92
7	Grasp	3-gram (stateless)	33.9
8	Grasp (batch=100)	3-gram	37.78 ± 0.64
9	Grasp (batch=200)	3-gram	38.89 ± 0.67
10	cdec (pop=200)	3-gram	46.33

Table 1. Slice sampler using $\phi(u_s; \boldsymbol{\alpha}) = \mathrm{Exp}(u_s; \lambda = 1.0)$.

ing to 2,000 ancestral samples), thus both systems perform equally well (rows 1-2 and 6-7). For nonlocal models, cdec employs *cube pruning* (pop limit 200), and Grasp employs the proposed *slice sampling* procedure uniformly subsampling slices to produce batches of 100 or 200 samples. We can see that Grasp succeeds to incorporate the language model to a certain extent, it outperforms stateless models by about 5 BLEU points (rows 3-4 vs rows 1-2, and rows 8-9 vs rows 6-7). However, it still lags behind cube pruning (rows 5 and 10). By rescoring larger batches from each slice (row 4 vs row 3, and row 9 vs row 8), we notice significant improvements, which indicates that sampling from $f(\mathbf{d}|\mathbf{u})$ is indeed a bottleneck.

In order to improve Grasp's translation accuracy, we need to sample more efficiently from slices. The general aim is to increase the sampler's mobility and reduce autocorrelation. Indeed, subsampling slices uniformly is a very simple strategy. We are currently investigating alternatives based on more sophisticated sampling techniques. In the experiments reported, Grasp draws on average 7.5 samples per second (with batch=100) and 4.7 samples per second (with batch=200). Note that to obtain a sample, Grasp must first obtain a slice (a random subset of the complete forest), subsample such slice reducing it to a fixed-size batch, rescore the batch producing an empirical distribution for $f(\mathbf{d}|\mathbf{u})$ (Equation 7), and finally, sample a derivation from this empirical distribution. To improve Grasp's time performance we are translating to *cython* some of the core procedures associated with these steps.

Finally, Figure 4 lists the features Grasp currently supports (this set is growing at a fast pace!).

Grammar formalism	epsilon-free CFGs
Weighted deduction	bottom-up (exact and sliced), top-down (exact and sliced)
Forest rescoring	top-down (exact and sliced)
Real-valued semirings	Boolean, Counting, Viterbi, Inside
Value recursion	robust to cycles
Derivation semirings	1-best, k-best, Sample, Forest, Slice
Sampling algorithms	ancestral sampling, slice sampling
LM queries	kenlm
Applications	constituency parsing, decoding for hiero models

Figure 4. Features in Grasp*'s current release.*

5. Conclusion

We have given a quick overview of semiring parsing and discussed a novel sampling technique for inference over complex structured spaces which is compatible with this general framework. We have presented Grasp, a toolkit for new directions in inference for applications such as parsing and machine translation. With this release, we hope to lower the initial implementation burden associated with complex structured state spaces, and we invite contributors to explore new sampling algorithms as well as new applications.

Acknowledgements

This research is funded by The Netherlands Organisation for Scientific Research (NWO), NWO VICI grant nr. 277-89-002.

Bibliography

Baltescu, Paul and Phil Blunsom. A Fast and Simple Online Synchronous Context Free Grammar Extractor. *The Prague Bulletin of Mathematical Linguistics*, 102(1):17–26, October 2014.

Bar-Hillel, Yehoshua, Micha A. Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, (14):143–172, 1961.

Billot, Sylvie and Bernard Lang. The Structure of Shared Forests in Ambiguous Parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada, June 1989. Association for Computational Linguistics.

Blunsom, Phil and Trevor Cohn. Inducing Synchronous Grammars with Slice Sampling. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 238–241, Los Angeles, California, June 2010. Association for Computational Linguistics.

Cherry, Colin and George Foster. Batch Tuning Strategies for Statistical Machine Translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 427–436, Montréal, Canada, June 2012. Association for Computational Linguistics.

Chiang, David. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 263–270, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

DeNero, John, David Chiang, and Kevin Knight. Fast Consensus Decoding over Translation Forests. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 567–575, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

Dyer, Chris and Philip Resnik. Context-free Reordering, Finite-state Translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 858–866, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

Dyer, Chris, Jonathan Weese, Hendra Setiawan, Adam Lopez, Ferhan Ture, Vladimir Eidelman, Juri Ganitkevitch, Phil Blunsom, and Philip Resnik. cdec: a decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, ACLDemos '10, pages 7–12, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

Gallo, Giorgio, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201, Apr. 1993.

Goodman, Joshua. Semiring parsing. *Computational Linguistics*, 25(4):573–605, Dec. 1999.

Heafield, Kenneth. KenLM: faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, WMT '11, pages 187–197, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

Heafield, Kenneth, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 690–696, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Hopcroft, John E. and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1979.

Huang, Liang. Advanced Dynamic Programming in Semiring and Hypergraph Frameworks. In *Coling 2008: Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications - Tutorial notes*, pages 1–18, Manchester, UK, August 2008. Coling 2008 Organizing Committee.

Klein, Dan and ChristopherD. Manning. Parsing and Hypergraphs. In Bunt, Harry, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, volume 23 of *Text, Speech and Language Technology*, pages 351–372. Springer Netherlands, 2005.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

Lopez, Adam. Hierarchical Phrase-Based Translation with Suffix Arrays. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 976–985, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Neal, Radford M. Slice Sampling. *Annals of statistics*, 31:705–741, June 2003.

Nederhof, Mark-Jan and Giorgio Satta. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148, Nacy, France, April 2003.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

Pereira, Fernando C. N. and David H. D. Warren. Parsing As Deduction. In *Proceedings of the 21st Annual Meeting on Association for Computational Linguistics*, ACL '83, pages 137–144, Stroudsburg, PA, USA, 1983. Association for Computational Linguistics.

Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1995.

Sima'an, Khalil. Computational complexity of probabilistic disambiguation by means of tree-grammars. In *Proceedings of the 16th conference on Computational linguistics - Volume 2*, COLING '96, pages 1175–1180, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.

Takezawa, Toshiyuki, Eiichiro Sumita, Fumiaki Sugaya, Hirofumi Yamamoto, and Seiichi Yamamoto. Toward a Broad-coverage Bilingual Corpus for Speech Translation of Travel Conversations in the Real World. In *Third International Conference on Language Resources and Evaluation*, LREC, Las Palmas, Canary Islands - Spain, May 2002. European Language Resources Association.

Tanner, Martin A. and Wing Hung Wong. The Calculation of Posterior Distributions by Data Augmentation. *Journal of the American Statistical Association*, 82(398):528–540, June 1987.

Address for correspondence:
Wilker Aziz
w.aziz@uva.nl
Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107, F2.11
Netherlands