# Transition-Based Dependency Parsing with Heuristic Backtracking

**Jacob Buckman**[♣] **Miguel Ballesteros**[◇] **Chris Dyer**[♠♣]
[♣]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[◇]NLP Group, Pompeu Fabra University, Barcelona, Spain
[♠]Google DeepMind, London, UK
jacobbuckman@cmu.edu, miguel.ballesteros@upf.edu, cdyer@google.com

## Abstract

We introduce a novel approach to the decoding problem in transition-based parsing: heuristic backtracking. This algorithm uses a series of partial parses on the sentence to locate the best candidate parse, using confidence estimates of transition decisions as a heuristic to guide the starting points of the search. This allows us to achieve a parse accuracy comparable to beam search, despite using fewer transitions. When used to augment a Stack-LSTM transition-based parser, the parser shows an unlabeled attachment score of up to 93.30% for English and 87.61% for Chinese.

## 1 Introduction

Transition-based parsing, one of the most prominent dependency parsing techniques, constructs a dependency structure by reading words sequentially from the sentence, and making a series of local decisions (called transitions) which incrementally build the structure. Transition-based parsing has been shown to be both fast and accurate; the number of transitions required to fully parse the sentence is linear relative to the number of words in the sentence.

In recent years, the field has seen dramatic improvements in the ability to correctly predict transitions. Recent models include the greedy Stack-LSTM model of Dyer et al. (2015) and the globally normalized feed-forward networks of Andor et al. (2016). These models output a local decision at each transition point, so searching the space of possible paths to the predicted tree is an important component of high-accuracy parsers.

One common search technique is beam search. (Zhang and Clark, 2008; Zhang and Nivre, 2011; Bohnet and Nivre, 2012; Zhou et al., 2015; Weiss et al., 2015; Yazdani and Henderson, 2015) In beam-search, a fixed number of candidate transition sequences are generated, and the highest-scoring sequence is chosen as the answer. One downside to beam search is that it often results in a significant amount of wasted predictions. A constant number of beams are explored at all points throughout the sentence, leading to some unnecessary exploration towards the beginning of the sentence, and potentially insufficient exploration towards the end.

One way that this problem can be mitigated is by using a dynamically-sized beam (Mejia-Lavalle and Ramos, 2013). When using this technique, at each step, prune all beams whose scores are below some value $s$, where $s$ is calculated based upon the distribution of scores of available beams. Common methods for pruning are removing all beams below some percentile, or any beams which scored below some constant percentage of the highest-scoring beam.

Another approach to solving this issue is given by Choi and McCallum (2013). They introduced selectional branching, which involves performing an initial greedy parse, and then using confidence estimates on each prediction to spawn additional beams. Relative to standard beam-search, this reduces the average number of predictions required to parse a sentence, resulting in a speed-up.

In this paper, we introduce heuristic backtracking, which expands on the ideas of selectional branching by integrating a search strategy based on a heuristic function (Pearl, 1984): a function which estimates the future cost of taking a particular decision. When paired with a good heuristic, heuristic backtracking maintains the property of reducing wasted predictions, but allows us to more fully explore the space of possible transition sequences (as compared to selectional branching). In this paper, we use a heuristic based on the confidence of transition predictions.

We also introduce a new optimization: heuristic backtracking with cutoff. Since heuristic backtracking produces results incrementally, it is possible to stop the search early if we have found an answer that we believe to be the gold parse, saving time proportional to the number of backtracks remaining.

We compare the performance of these various decoding algorithms with the Stack-LSTM parser (Dyer et al., 2015), and achieve slightly higher accuracy than beam search, in significantly less time.

## 2 Transition-Based Parsing With Stack-LSTM

Our starting point is the model described by Dyer et al. (2015).[1] The parser implements the arc-standard algorithm (Nivre, 2004) and it therefore makes use of a stack and a buffer. In (Dyer et al., 2015), the stack and the buffer are encoded with Stack-LSTMs, and a third sequence with the history of actions taken by the parser is encoded with another Stack-LSTM. The three encoded sequences form the parser state $\mathbf{p}_t$ defined as follows,

$$\mathbf{p}_t = \max\left\{\mathbf{0}, \mathbf{W}[\mathbf{s}_t; \mathbf{b}_t; \mathbf{a}_t] + \mathbf{d}\right\}, \tag{1}$$

where $\mathbf{W}$ is a learned parameter matrix, $\mathbf{b}_t$, $\mathbf{s}_t$ and $\mathbf{a}_t$ are the stack LSTM encoding of buffer, stack and the history of actions, and $\mathbf{d}$ is a bias term. The output $\mathbf{p}_t$ (after a component-wise rectified linear unit (ReLU) nonlinearity (Glorot et al., 2011)) is then used to compute the probability of the parser action at time $t$ as:

$$p(z_t \mid \mathbf{p}_t) = \frac{\exp\left(\mathbf{g}_{z_t}^\top \mathbf{p}_t + q_{z_t}\right)}{\sum_{z' \in \mathcal{A}(S,B)} \exp\left(\mathbf{g}_{z'}^\top \mathbf{p}_t + q_{z'}\right)}, \tag{2}$$

where $\mathbf{g}_z$ is a column vector representing the (output) embedding of the parser action $z$, and $q_z$ is a bias term for action $z$. The set $\mathcal{A}(S, B)$ represents the valid transition actions that may be taken in the current state. The objective function is:

$$\mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{w}, \boldsymbol{z}) = \sum_{t=1}^{|\boldsymbol{z}|} \log p(z_t \mid \mathbf{p}_t) \tag{3}$$

where $\boldsymbol{z}$ refers to parse transitions.

## 3 Heuristic Backtracking

Using the Stack-LSTM parsing model of Dyer et al. (2015) to predict each decision greedily yields very high accuracy; however, it can only explore one path, and it therefore can be improved by conducting a larger search over the space of possible parses. To do this, we introduce a new algorithm, heuristic backtracking. We also introduce a novel cutoff approach to further increase speed.

### 3.1 Decoding Strategy

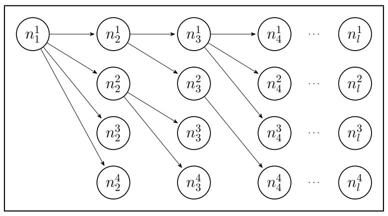
We model the space of possible parses as a tree, where each node represents a certain parse state (with complete values for stack, buffer, and action history). Transitions connect nodes of the tree, and leaves of the tree represent final states.

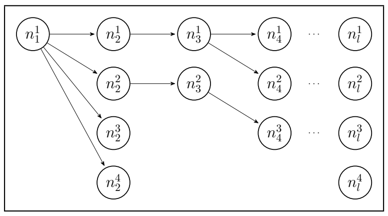
During the first iteration, we start at the root of the tree, and greedily parse until we reach a leaf. That is, for each node, we use the Stack-LSTM model to calculate scores for each transition (as described in Section 2), and then execute the highest-scoring transition, generating a child node upon which we repeat the procedure. Additionally, we save an ordered list of the transition scores, and calculate the confidence of the node (as described in Section 3.2).

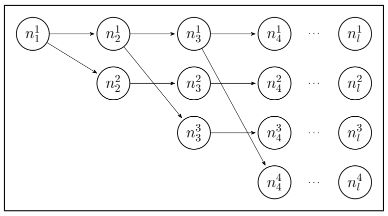
When we reach the leaf node, we backtrack to the location that is most likely to fix a mistake. To find this, we look at all explored nodes that still have at least one unexplored child, and choose the node with the lowest heuristic confidence (see Section 3.2). We rewind our stack, buffer, and action history to that state, and execute the highest-scoring transition from that node that has not yet been explored. At this point, we are again in a fully-unexplored node, and can greedily parse just as before until we reach another leaf.

Once we have generated $b$ leaves, we score them all and return the transition sequence leading up to the highest-scoring leaf as the answer. Just as in previous studies (Collins and Roark, 2004), we use the

[1]We refer to the original work for details.


(a) Beam Search


(b) Dynamic Beam Search


(c) Selectional Branching

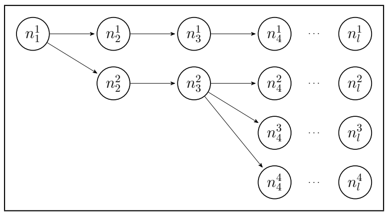

(d) Heuristic Backtracking

Figure 1: Visualization of various decoding algorithms

sum of the log probabilities of all individual transitions as the overall score for the parse.

## 3.2 Calculating Error Likelihood

Let $n$ indicate a node, which consists of a state, a buffer, and an action history. We may refer to a specific node as $n_i^j$, which means it has $i$ actions in its action history and it is part of the history of the $j$th leaf (and possibly subsequent leaves). Let the function $T(n)$ represent a sorted vector containing all possible transitions from $n$, and $S(n)$ represent a sorted vector containing the scores of all of these transitions, in terms of log probabilities of each score. We can index the scores in order of value, so $T_1(n)$ is the highest-scoring transition and $S_1(n)$ is its score, $T_2(n)$ is the second-highest-scoring transition, etc. Here, let $u_n$ indicate the ranking of the transition leading to the first unexplored child of a node $n$. Also, let $V(n)$ represent the total score of all nodes in the history of $n$, i.e. the sum of all the scores of individual transitions that allowed us to get to $n$.

To calculate the confidence of an individual node, Choi and McCallum (2013) simply found the score margin, or difference in probability between the top-scoring transition and the second-highest scoring transition: $C(n) = S_1(n) - S_2(n)$. In selectional branching, the only states for which the confidence was relevant were the states in the first greedy parse, i.e. states $n_i^1$ for all $i$. For heuristic backtracking, we wish to generalize this to any state $n_i^j$ for all $i$ and $j$.

We do this in the following way:

$$H(n_i^j) = (V(n_i^1) - V(n_i^j)) + (S_{(u_{n_i^j})-1}(n_i^j) + S_{(u_{n_i^j})}(n_i^j)) \tag{4}$$

Intuitively, this formula means that the node that will be explored first is the node that will yield a parse that scores as close to the greedy choice as possible. The first term ensures that it has a history of good choices, and the second term ensures that the new child node being explored will be nearly as good as the prior child.

## 3.3 Number of Predictions

As discussed earlier, we use number of predictions made by the model as a proxy for the speed; execution speed may vary based on system and algorithmic implementation, but prediction count gives a good estimate of the overall work done by the algorithm.

Consider a sentence of length $l$, which requires at most $2l$ transitions with the greedy decoder (Nivre, 2004). The number of predictions required for heuristic backtracking for $b$ leaves is guaranteed to be less than or equal to a beam search with $b$ beams.

When doing a beam search, the first transition will require 1 prediction, and then every subsequent transition will require 1 prediction per beam, or $b$ predictions. This results in a total of $b(2l - 1) + 1$ predictions.

When doing heuristic backtracking, the first greedy search will require $2l$ predictions. Every subsequent prediction will require a number of predictions dependent on the target of the backtrack: backtracking to $n_i^j$ will require $2l - (i + 1)$ predictions. Note that $0 < i < 2l$. Thus, each backtrack will require at maximum $2l - 1$ predictions. Therefore, the maximum total amount of predictions is $2l + (b - 1)(2l - 1) = b(2l - 1) + 1$.

However, note that on average, there are significantly fewer. Assuming that all parts of a sentence have approximately equal score distributions, the average backtrack will be where $i = l$, and reduce predictions by 50%.

An intuitive understanding of this difference can be gained by viewing the graphs of various decoding methods in Figure 1. Beam search has many nodes which never yield children that reach an end-state; dynamic beam search has fewer, but still several. Selectional branching has none, but suffers from the restriction that every parse candidate can be no more than one decision away from the greedy parse. With heuristic backtracking, there is no such restriction, but yet every node explored is directly useful for generating a candidate parse.

### 3.4 Early Cutoff

Another inefficiency inherent to beam search is the fact that all $b$ beams are always fully explored. Since the beams are calculated in parallel, this is inevitable. However, with heuristic backtracking, the beams are calculated incrementally; this gives us the opportunity to cut off our search at any point. In order to leverage this into more efficient parsing, we constructed a second Stack-LSTM model, which we call the cutoff model. The cutoff model uses a single Stack-LSTM[2] that takes as input the sequence of parser states (see Eq 1), and outputs a boolean variable predicting whether the entire parse is correct or incorrect.

To train the cutoff model, we used stochastic gradient descent over the training set. For each training example, we first parse it greedily using the Stack-LSTM parser. Then, for as long as the parse has at least one mistake, we pass it to the cutoff model as a negative training example. Once the parse is completely correct, we pass it to the cutoff model as a positive training example. The loss function that we use is:

$$\mathcal{L}_{\boldsymbol{\theta}} = -\log\, p(\mathbf{t} \mid \boldsymbol{s}) \qquad (5)$$

where $\mathbf{s}$ is the LSTM encoded vector and $\mathbf{t}$ is the truth (parse correct/incorrect).

When decoding using early cutoff, we follow the exact same procedure as for normal heuristic backtracking, but after every candidate parse is generated, we use it as input to our cutoff model. When our cutoff model returns our selection as correct, we stop backtracking and return it as the answer. If we make $b$ attempts without finding a correct parse, we follow the same procedure as before.

## 4 Experiments and Results

To test the effectiveness of heuristic backtracking, we compare it with other decoding techniques: greedy, beam search,[3], dynamic beam search (Mejia-Lavalle and Ramos, 2013), and selectional branching (Choi and McCallum, 2013). We then try heuristic backtracking (see Section 3.1), and heuristic backtracking with cutoff (see Section 3.4). Note that beam search was *not* used for early-update training (Collins and Roark, 2004). We use the same greedy training strategy for all models, and we only change the decoding strategy.

We tested the performance of these algorithms on the English SD and Chinese CTB.[4] A single model was trained using the techniques described in Section 2, and used as the transition model for all decoding algorithms. Each decoding technique was tested with varying numbers of beams; as $b$ increased, both the predictions per sentence and accuracy trended upwards. The results are summarized in Table 1.[5] Note that we report results for only the highest-accuracy $b$ (in the development set) for each.

We also report the results of the cutoff model in Table 2. The same greedily-trained model as above was used to generate candidate parses and confidence estimates for each transition, and then the cutoff model was trained to use these confidence esti-

[2] 2 layers and 300 dimensions.

[3] Greedy and beam-search were already explored by Dyer et al. (2015)

[4] Using the exact same settings as Dyer et al. (2015) with pretrained embeddings and part-of-speech tags.

[5] The development sets are used to set the model parameters; results on the development sets are similar to the ones obtained in the test sets.

| English | | | |
|---|---|---|---|
| **Decoding** | **Pred/Sent** | **UAS** | **LAS** |
| Greedy – Dyer et al. | 47.92 | 93.04% | 90.87% |
| Beam Search | 542.09 | 93.32% | 91.19% |
| Dynamic Beam Search | 339.42 | 93.32% | 91.19% |
| Sel. Branching | 59.66 | 93.24% | 91.12% |
| Heur. Backtr. | 198.03 | 93.30% | 91.18% |
| Heur. Backtr. w/ Cutoff | 108.32 | 93.27% | 91.16% |

| Chinese | | | |
|---|---|---|---|
| **Decoding** | **Pred/Sent** | **UAS** | **LAS** |
| Greedy – Dyer et al. | 53.79 | 87.31% | 85.88% |
| Beam Search | 815.65 | 87.62% | 86.17% |
| Dynamic Beam Search | 282.32 | 87.62% | 86.17% |
| Sel. Branching | 91.51 | 87.53% | 86.08% |
| Heur. Backtr. | 352.30 | 87.61% | 86.16% |
| Heur. Backtr. w/ Cutoff | 162.37 | 87.60% | 86.15% |

Table 1: UAS and LAS of various decoding methods. Pred/Sent refers to number of predictions made by the Stack-LSTM per sentence.

mates to discriminate between correctly-parsed and incorrectly-parsed sentences.

## 5 Discussion

In Table 1 we see that in both English and Chinese, the best heuristic backtracking performs approximately as well as the best beam search, while making less than half the predictions. This supports our hypothesis that heuristic backtracking can perform at the same level as beam search, but with increased efficiency.

Dynamic beam search also performed as well as full beam search, despite demonstrating a reduction in predictions on par with that of heuristic backtracking. Since the implementation of dynamic beam search is very straightforward for systems which have already implemented beam search, we believe this will prove to be a useful finding.

Heuristic backtracking with cutoff outperformed greedy decoding, and reduced transitions by an additional 50%. However, it increased accuracy slightly less than full heuristic backtracking. We believe this difference could be mitigated with an improved cutoff model; as can be seen in Table 2, the cutoff model was only able to discriminate between correct and incorrect parses around 75% of the time. Also, note that while predictions per sentence were low, the overall runtime was increased due to running the cutoff LSTM multiple times per sentence.

| **Language** | **Cutoff Accuracy** |
|---|---|
| English | 72.43% |
| Chinese | 75.18% |

Table 2: Test-set accuracy of cutoff model on English and Chinese.

## 6 Related Work

Heuristic backtracking is most similar to the work of Choi and McCallum (2013), but is distinguished from theirs by allowing new beams to be initialized from any point in the parse, rather than only from points in the initial greedy parse. Heuristic backtracking also bears similarity to greedy-best-first-search (Pearl, 1984), but is unique in that it guarantees that $b$ candidate solutions will be found within $b(2l-1)+1$ predictions. Our work also relates to beam-search parsers (Zhang and Clark, 2008, *inter alia*).

## 7 Conclusions

We have introduced a novel decoding algorithm, called heuristic backtracking, and presented evidence that it performs at the same level as beam search for decoding, while being significantly more efficient. We have demonstrated this for both English and Chinese, using a parser with strong results with a greedy decoder. We expect that heuristic backtracking could be applied to any other transition-based parser with similar benefits.

We plan on experimenting with various heuristics and cutoff models, such as adapting the attention-based models of Bahdanau et al. (2014) to act as a guide for both the heuristic search and cutoff.

## Acknowledgments

Miguel Ballesteros was supported by the European Commission under the contract numbers FP7-ICT-610411 (project MULTISENSOR) and H2020-RIA-645012 (project KRISTINA).

## References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *CoRR*, abs/1603.06042.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1455–1465, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based dependency parsing with selectional branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1052–1062, Sofia, Bulgaria, August. Association for Computational Linguistics.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 334–343. The Association for Computer Linguistics.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proc. AISTATS*.

Manuel Mejia-Lavalle and Cesar Geovani Pereyra Ramos. 2013. Beam search with dynamic pruning for artificial intelligence hard problems. In *Proceedings of the 2013 International Conference on Mechatronics, Electronics and Automotive Engineering*, November.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*.

Judea Pearl. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China, July. Association for Computational Linguistics.

Majid Yazdani and James Henderson. 2015. Incremental recurrent neural network dependency parser with search-based discriminative training. In *CoNLL*, pages 142–152. ACL.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 562–571, Stroudsburg, PA, USA. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 188–193, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing, China, July. Association for Computational Linguistics.