Heuristic Cube Pruning in Linear Time

Andrea Gesmundo
Department of
Computer Science
University of Geneva
andrea.gesmundo@unige.ch

Giorgio Satta
Department of
Information Engineering
University of Padua
satta@dei.unipd.it

James Henderson
Department of
Computer Science
University of Geneva
james.henderson@unige.ch

Abstract

We propose a novel heuristic algorithm for Cube Pruning running in linear time in the beam size. Empirically, we show a gain in running time of a standard machine translation system, at a small loss in accuracy.

1 Introduction

Since its first appearance in (Huang and Chiang, 2005), the Cube Pruning (CP) algorithm has quickly gained popularity in statistical natural language processing. Informally, this algorithm applies to scenarios in which we have the k-best solutions for two input sub-problems, and we need to compute the k-best solutions for the new problem representing the combination of the two sub-problems.

CP has applications in tree and phrase based machine translation (Chiang, 2007; Huang and Chiang, 2007; Pust and Knight, 2009), parsing (Huang and Chiang, 2005), sentence alignment (Riesa and Marcu, 2010), and in general in all systems combining inexact beam decoding with dynamic programming under certain monotonic conditions on the definition of the scores in the search space.

Standard implementations of CP run in time $\mathcal{O}(k \log(k))$, with k being the size of the input/output beams (Huang and Chiang, 2005). Gesmundo and Henderson (2010) propose Faster CP (FCP) which optimizes the algorithm but keeps the $\mathcal{O}(k \log(k))$ time complexity. Here, we propose a novel heuristic algorithm for CP running in time $\mathcal{O}(k)$ and evaluate its impact on the efficiency and performance of a real-world machine translation system.

2 Preliminaries

Let $\mathcal{L} = \langle x_0, \ldots, x_{k-1} \rangle$ be a list over $\mathbb{R}$, that is, an ordered sequence of real numbers, possibly with repetitions. We write $|\mathcal{L}| = k$ to denote the length of $\mathcal{L}$. We say that $\mathcal{L}$ is **descending** if $x_i \geq x_j$ for every i, j with $0 \leq i < j < k$. Let $\mathcal{L}_1 = \langle x_0, \ldots, x_{k-1} \rangle$ and $\mathcal{L}_2 = \langle y_0, \ldots, y_{k'-1} \rangle$ be two descending lists over $\mathbb{R}$. We write $\mathcal{L}_1 \oplus \mathcal{L}_2$ to denote the descending list with elements $x_i + y_j$ for every i, j with $0 \leq i < k$ and $0 \leq j < k'$.

In **cube pruning** (CP) we are given as input two descending lists $\mathcal{L}_1$, $\mathcal{L}_2$ over $\mathbb{R}$ with $|\mathcal{L}_1| = |\mathcal{L}_2| = k$, and we are asked to compute the descending list consisting of the first k elements of $\mathcal{L}_1 \oplus \mathcal{L}_2$.

A problem related to CP is the k-**way merge** problem (Horowitz and Sahni, 1983). Given descending lists $\mathcal{L}_i$ for every i with $0 \leq i < k$, we write $\mathsf{merge}_{i=0}^{k-1} \mathcal{L}_i$ to denote the "merge" of all the lists $\mathcal{L}_i$, that is, the descending list with all elements from the lists $\mathcal{L}_i$, including repetitions.

For $\Delta \in \mathbb{R}$ we define $\mathsf{shift}(\mathcal{L}, \Delta) = \mathcal{L} \oplus \langle \Delta \rangle$. In words, $\mathsf{shift}(\mathcal{L}, \Delta)$ is the descending list whose elements are obtained by "shifting" the elements of $\mathcal{L}$ by Δ, preserving the order. Let $\mathcal{L}_1, \mathcal{L}_2$ be descending lists of length k, with $\mathcal{L}_2 = \langle y_0, \ldots, y_{k-1} \rangle$. Then we can express the output of CP on $\mathcal{L}_1, \mathcal{L}_2$ as the list

$$\mathsf{merge}_{i=0}^{k-1} \mathsf{shift}(\mathcal{L}_1, y_i) \tag{1}$$

truncated after the first k elements. This shows that the CP problem is a particular instance of the k-way merge problem, in which all input lists are related by k *independent* shifts.

Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 296–300,
Jeju, Republic of Korea, 8-14 July 2012. ©2012 Association for Computational Linguistics

Computation of the solution of the k-way merge problem takes time $\mathcal{O}(q \log(k))$, where q is the size of the output list. In case each input list has length k this becomes $\mathcal{O}(k^2 \log(k))$, and by restricting the computation to the first k elements, as required by the CP problem, we can further reduce to $\mathcal{O}(k \log(k))$. This is the already known upper bound on the CP problem (Huang and Chiang, 2005; Gesmundo and Henderson, 2010). Unfortunately, there seems to be no way to achieve an asymptotically faster algorithm by exploiting the restriction that the input lists are all related by some shifts. Nonetheless, in the next sections we use the above ideas to develop a heuristic algorithm running in time linear in k.

3 Cube Pruning With Constant Slope

Consider lists $\mathcal{L}_1, \mathcal{L}_2$ defined as in section 2. We say that $\mathcal{L}_2$ has **constant slope** if $y_{i-1} - y_i = \Delta > 0$ for every i with $0 < i < k$. Throughout this section we assume that $\mathcal{L}_2$ has constant slope, and we develop an (exact) linear time algorithm for solving the CP problem under this assumption.

For each $i \geq 0$, let I_i be the left-open interval $(x_0 - (i+1) \cdot \Delta,\ x_0 - i \cdot \Delta]$ of $\mathbb{R}$. Let also $s = \lfloor (x_0 - x_{k-1})/\Delta \rfloor + 1$. We split $\mathcal{L}_1$ into (possibly empty) sublists σ_i, $0 \leq i < s$, called **segments**, such that each σ_i is the descending sublist consisting of all elements from $\mathcal{L}_1$ that belong to I_i. Thus, moving down one segment in $\mathcal{L}_1$ is the closest equivalent to moving down one element in $\mathcal{L}_2$.

Let $t = \min\{k, s\}$; we define descending lists M_i, $0 \leq i < t$, as follows. We set $M_0 = \mathsf{shift}(\sigma_0, y_0)$, and for $1 \leq i < t$ we let

$$M_i = \mathsf{merge}\{\mathsf{shift}(\sigma_i, y_0), \mathsf{shift}(M_{i-1}, -\Delta)\} \quad (2)$$

We claim that the ordered concatenation of M_0, $M_1, \ldots, M_{t-1}$ truncated after the first k elements is exactly the output of CP on input $\mathcal{L}_1, \mathcal{L}_2$.

To prove our claim, it helps to visualize the descending list $\mathcal{L}_1 \oplus \mathcal{L}_2$ (of size k^2) as a $k \times k$ matrix L whose j-th column is $\mathsf{shift}(\mathcal{L}_1, y_j)$, $0 \leq j < k$. For an interval $I = (x, x']$, we define $\mathsf{shift}(I, y) = (x+y, x'+y]$. Similarly to what we have done with $\mathcal{L}_1$, we can split each column of L into s segments. For each i, j with $0 \leq i < s$ and $0 \leq j < k$, we define the i-th segment of the j-th column, written $\sigma_{i,j}$, as the descending sublist consisting of all elements of that column that belong to $\mathsf{shift}(I_i, y_j)$. Then we have $\sigma_{i,j} = \mathsf{shift}(\sigma_i, y_j)$.

For any d with $0 \leq d < t$, consider now all segments $\sigma_{i,j}$ with $i + j = d$, forming a subantidiagonal in L. We observe that these segments contain *all and only* those elements of L that belong to the interval I_d. It is not difficult to show by induction that these elements are exactly the elements that appear in descending order in the list M_i defined in (2).

We can then directly use relation (2) to iteratively compute CP on two lists of length k, under our assumption that one of the two lists has constant slope. Using the fact that the merge of two lists as in (2) can be computed in time linear in the size of the output list, it is not difficult to implement the above algorithm to run in time $\mathcal{O}(k)$.

4 Linear Time Heuristic Solution

In this section we further elaborate on the exact algorithm of section 3 for the constant slope case, and develop a heuristic solution for the general CP problem. Let $\mathcal{L}_1, \mathcal{L}_2, L$ and k be defined as in sections 2 and 3. Despite the fact that $\mathcal{L}_2$ does not have a constant slope, we can still split each column of L into segments, as follows.

Let $\widetilde{I}_i$, $0 \leq i < k-1$, be the left-open interval $(x_0 + y_{i+1},\ x_0 + y_i]$ of $\mathbb{R}$. Note that, unlike the case of section 3, intervals $\widetilde{I}_i$'s are not all of the same size now. Let also $\widetilde{I}_{k-1} = [x_{k-1} + y_{k-1},\ x_0 + y_{k-1}]$. For each i, j with $0 \leq j < k$ and $0 \leq i < k - j$, we define segment $\widetilde{\sigma}_{i,j}$ as the descending sublist consisting of all elements of the j-th column of L that belong to $\widetilde{I}_{i+j}$. In this way, the j-th column of L is split into segments $\widetilde{I}_j, \widetilde{I}_{j+1}, \ldots, \widetilde{I}_{k-1}$, and we have a variable number of segments per column. Note that segments $\widetilde{\sigma}_{i,j}$ with a constant value of $i+j$ contain *all and only* those elements of L that belong to the left-open interval $\widetilde{I}_{i+j}$.

Similarly to section 3, we define descending lists $\widetilde{M}_i$, $0 \leq i < k$, by setting $\widetilde{M}_0 = \widetilde{\sigma}_{0,0}$ and, for $1 \leq i < k$, by letting

$$\widetilde{M}_i = \mathsf{merge}\{\widetilde{\sigma}_{i,0}\ ,\ \mathsf{path}(\widetilde{M}_{i-1}, L)\} \quad (3)$$

Note that the function $\mathsf{path}(\widetilde{M}_{i-1}, L)$ should not return $\mathsf{shift}(\widetilde{M}_{i-1}, -\Delta)$, for some value Δ, as in the

1: **Algorithm 1** $(\mathcal{L}_1, \mathcal{L}_2) : \widetilde{\mathcal{L}}^\star$
2: $\widetilde{\mathcal{L}}^\star$.insert($L[0,0]$);
3: referColumn $\leftarrow 0$;
4: $x_{follow} \leftarrow L[0,1]$;
5: $x_{deviate} \leftarrow L[1,0]$;
6: $\mathcal{C} \leftarrow$ CircularList($[0,1]$);
7: $\mathcal{C}$-iterator $\leftarrow \mathcal{C}$.begin();
8: **while** $|\widetilde{\mathcal{L}}^\star| < k$ **do**
9:   **if** $x_{follow} > x_{deviate}$ **then**
10:     $\widetilde{\mathcal{L}}^\star$.insert($x_{follow}$);
11:     **if** $\mathcal{C}$-iterator.current()=$[0,1]$ **then**
12:       referColumn++;
13:     $[i,j] \leftarrow \mathcal{C}$-iterator.next();
14:     $x_{follow} \leftarrow L[i$,referColumn$+j]$;
15:   **else**
16:     $\widetilde{\mathcal{L}}^\star$.insert($x_{deviate}$);
17:     $i \leftarrow x_{deviate}$.row();
18:     $\mathcal{C}$-iterator.insert($[i, -$referColumn$]$);
19:     $x_{deviate} \leftarrow L[i+1, 0]$;

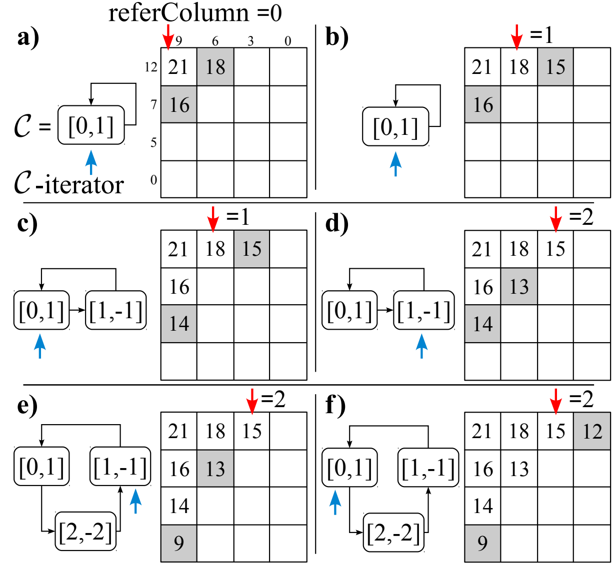

Figure 1: A running example for Algorithm 1.

case of (2). This is because input list $\mathcal{L}_2$ does not have constant slope in general. In an exact algorithm, $\mathsf{path}(\widetilde{M}_{i-1}, L)$ should return the descending list $\mathcal{L}^\star_{i-1} = \mathsf{merge}^i_{j=1}\ \widetilde{\sigma}_{i-j,j}$: Unfortunately, we do not know how to compute such a i-way merge without introducing a logarithmic factor.

Our solution is to define $\mathsf{path}(\widetilde{M}_{i-1}, L)$ in such a way that it computes a list $\widetilde{\mathcal{L}}_{i-1}$ which is a permutation of the correct solution $\mathcal{L}^\star_{i-1}$. To do this, we consider the "relative" path starting at $x_0 + y_{i-1}$ that we need to follow in L in order to collect all the elements of $\widetilde{M}_{i-1}$ in the given order. We then apply such a path starting at $x_0 + y_i$ and return the list of collected elements. Finally, we compute the output list $\widetilde{\mathcal{L}}^\star$ as the concatenation of all lists $\widetilde{M}_i$ up to the first k elements.

It is not difficult to see that when $\mathcal{L}_2$ has constant slope we have $\widetilde{M}_i = M_i$ for all i with $0 \leq i < k$, and list $\widetilde{\mathcal{L}}^\star$ is the exact solution to the CP problem. When $\mathcal{L}_2$ does not have a constant slope, list $\widetilde{\mathcal{L}}^\star$ might depart from the exact solution in two respects: it might not be a descending list, because of local variations in the ordering of the elements; and it might not be a permutation of the exact solution, because of local variations at the end of the list. In the next section we evaluate the impact that our heuristic solution has on the performance of a real-world machine translation system.

Algorithm 1 implements the idea presented in (3). The algorithm takes as input two descending lists $\mathcal{L}_1, \mathcal{L}_2$ of length k and outputs the list $\widetilde{\mathcal{L}}^\star$ which approximates the desired solution. Element $L[i,j]$ denotes the combined value $x_i + y_j$, and is always computed on demand.

We encode a relative path (mentioned above) as a sequence of elements, called **displacements**, each of the form $[i, \delta]$. Here i is the index of the next row, and δ represents the *relative* displacement needed to reach the next column, to be summed to a variable called referColumn denoting the index of the column of the first element of the path. The reason why only the second coordinate is a relative value is that we shift paths only horizontally (row indices are preserved). The relative path is stored in a circular list $\mathcal{C}$, with displacement $[0,1]$ marking the starting point (paths are always shifted one element to the right). When merging the list obtained through the path for $\widetilde{M}_{i-1}$ with segment $\widetilde{\sigma}_{i,0}$, as specified in (3), we update $\mathcal{C}$ accordingly, so that the new relative path can be used at the next round for $\widetilde{M}_i$. The merge operator is implemented by the while cycle at lines 8 to 19 of algorithm 1. The if statement at line 9 tests whether the next step should follow the relative path for $\widetilde{M}_{i-1}$ stored in $\mathcal{C}$ (lines 10 to 14) or

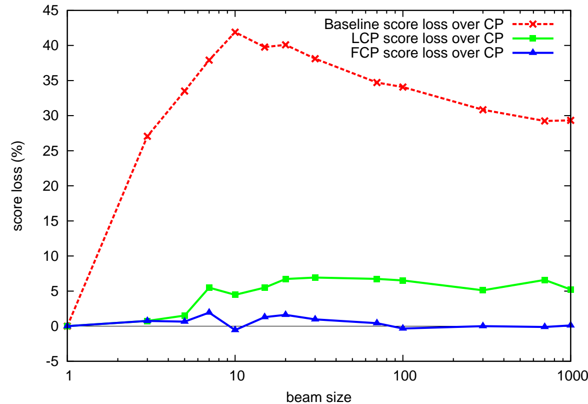

Figure 2: Search-score loss relative to standard CP.

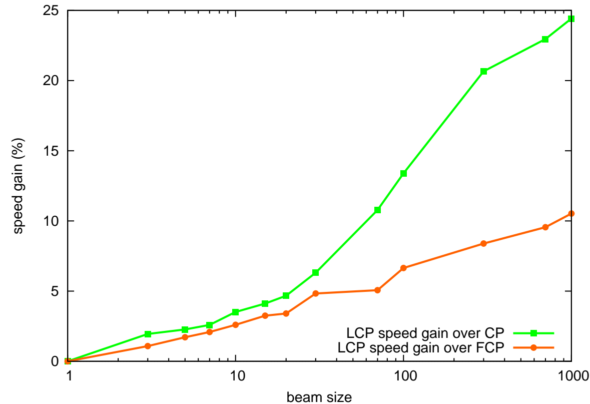

Figure 3: Linear CP relative speed gain.

else depart visiting an element from $\widetilde{\sigma}_{i,0}$ in the first column of L (lines 16 to 19). In the latter case, we update $\mathcal{C}$ with the new displacement (line 18), where the function insert() inserts a new element before the one currently pointed to. The function next() at line 13 moves the iterator to the next element and then returns its value.

A running example of algorithm 1 is reported in Figure 1. The input lists are $\mathcal{L}_1 = \langle 12, 7, 5, 0 \rangle$, $\mathcal{L}_2 = \langle 9, 6, 3, 0 \rangle$. Each of the picture in the sequence represents the state of the algorithm when the test at line 9 is executed. The value in the shaded cell in the first column is $x_{deviate}$, while the value in the other shaded cell is x_{follow}.

5 Experiments

We implement Linear CP (LCP) on top of Cdec (Dyer et al., 2010), a widely-used hierarchical MT system that includes implementations of standard CP and FCP algorithms. The experiments were executed on the NIST 2003 Chinese-English parallel corpus. The training corpus contains 239k sentence pairs. A binary translation grammar was extracted using a suffix array rule extractor (Lopez, 2007). The model was tuned using MERT (Och, 2003). The algorithms are compared on the NIST-03 test set, which contains 919 sentence pairs. The features used are basic lexical features, word penalty and a 3-gram Language Model (Heafield, 2011).

Since we compare decoding algorithms on the same search space, the accuracy comparison is done in terms of search score. For each algorithm we compute the average score of the best translation found for the test sentences. In Figure 2 we plot the score-loss relative to standard CP average score. Note that the FCP loss is always $< 3\%$, and the LCP loss is always $< 7\%$. The dotted line plots the loss of a baseline linear time heuristic algorithm which assumes that both input lists have constant slope, and that scans L along parallel lines whose steep is the ratio of the average slope of each input list. The baseline greatly deteriorates the accuracy: this shows that finding a reasonable linear time heuristic algorithm is not trivial. We can assume a bounded loss in accuracy, because for larger beam size all the algorithms tend to converge to exhaustive search.

We found that these differences in search score resulted in no significant variations in BLEU score (e.g. with $k = 30$, CP reaches 32.2 while LCP 32.3).

The speed comparison is done in terms of algorithm run-time. Figure 3 plots the relative speed gain of LCP over standard CP and over FCP. Given the log-scale used for the beam size k, the linear shape of the speed gain over FCP (and CP) in Figure 3 empirically confirms that LCP has a $\log(k)$ asymptotic advantage over FCP and CP.

In addition to Chinese-English, we ran experiments on translating English to French (from Europarl corpus (Koehn, 2005)), and find that the LCP score-loss relative to CP is $< 9\%$ while the speed relative advantage of LCP over CP increases in average by 11.4% every time the beam size is multiplied by 10 (e.g. with $k = 1000$ the speed advantage is 34.3%). These results confirm the bounded accuracy loss and $\log(k)$ speed advantage of LCP.

References

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

Chris Dyer, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Hendra Setiawan, Ferhan Ture, Vladimir Eidelman, Phil Blunsom, and Philip Resnik. 2010. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *ACL '10: Proceedings of the ACL 2010 System Demonstrations*, Uppsala, Sweden.

Andrea Gesmundo and James Henderson. 2010. Faster Cube Pruning. In *IWSLT '10: Proceedings of the 7th International Workshop on Spoken Language Translation*, Paris, France.

Kenneth Heafield. 2011. KenLM: Faster and smaller language model queries. In *WMT '11: Proceedings of the 6th Workshop on Statistical Machine Translation*, Edinburgh, Scotland, UK.

E. Horowitz and S. Sahni. 1983. *Fundamentals of data structures*. Computer software engineering series. Computer Science Press.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *IWPT '05: Proceedings of the 9th International Workshop on Parsing Technology*, Vancouver, British Columbia, Canada.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *ACL '07: Proceedings of the 45th Conference of the Association for Computational Linguistics*, Prague, Czech Republic.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of the 10th Machine Translation Summit*, Phuket, Thailand.

Adam Lopez. 2007. Hierarchical phrase-based translation with suffix arrays. In *EMNLP-CoNLL '07: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Prague, Czech Republic.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *ACL '03: Proceedings of the 41st Conference of the Association for Computational Linguistics*, Sapporo, Japan.

Michael Pust and Kevin Knight. 2009. Faster MT decoding through pervasive laziness. In *NAACL '09: Proceedings of the 10th Conference of the North American Chapter of the Association for Computational Linguistics*, Boulder, CO, USA.

Jason Riesa and Daniel Marcu. 2010. Hierarchical search for word alignment. In *ACL '10: Proceedings of the 48th Conference of the Association for Computational Linguistics*, Uppsala, Sweden.