Optimizations and Extensions for Weighted CFG Parsers

Kilian Gebhardt

Chair for foundations of programming
Institute for theoretical computer science
TU Dresden

2019-06-04
Outline

Pruning
  - Pruning for CKY parsing
  - Pruning for deductive parsing

$k$-best parsing

A*-parsing
Pruning

- During CKY or deductive parsing many items are explored which are not part of the best derivation.
Pruning

- During CKY or deductive parsing many items are explored which are not part of the best derivation
- Idea: avoid items that are not part of the best derivation to speed up parsing
During CKY or deductive parsing many items are explored which are not part of the best derivation.

Idea: avoid items that are not part of the best derivation to speed up parsing.

Problem: How can we know these items in advance?
Pruning

- During CKY or deductive parsing many items are explored which are not part of the best derivation
- Idea: avoid items that are not part of the best derivation to speed up parsing
- Problem: How can we know these items in advance?
- Practical solution: Use simple methods but take the risk of finding suboptimal derivation.
Pruning for CKY parsing

Require: weighted binary cfg \((N, \Sigma, P, S, \mu)\), word \(t_1 \ldots t_n\) where \(t_1, \ldots, t_n \in \Sigma\)

Ensure: family \((c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)\) such that, for all \(i, j, A\),
\[
c_{i,j,A} = \max\{\mu(d) \mid d \in D^A_G(t_i+1 \ldots t_j)\} \cup \{0\}
\]

1: function CKY\((P, \mu, t_1 \ldots t_n)\)
2: \((c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N)\)
3: for \(1 \leq i \leq n\) do
4: for \(A \rightarrow t_i \in P\) do
5: \(c_{i-1,i,A} := \max\{c_{i-1,i,A}, \mu(A \rightarrow t_i)\}\)
6: for \(2 \leq r \leq n\) do
7: for \(0 \leq i \leq n - r\) do
8: \(j := i + r\)
9: for \(m \in \{i + 1, i + 2, \ldots, j - 1\}\) do
10: for \(B, C \in N\) do
11: for \(A \in N\) such that \(A \rightarrow BC \in R\) do
12: \(c_{i,j,A} := \max\{c_{i,j,A}, \mu(A \rightarrow BC) \cdot c_{i,m,B} \cdot c_{m,j,C}\}\)
13: return c
Pruning for CKY parsing

Require: weighted binary cfg \((N, \Sigma, P, S, \mu)\), word \(t_1 \ldots t_n\) where \(t_1, \ldots, t_n \in \Sigma\)

Ensure: family \((c_{i,j,A} \in \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)\) such that, for all \(i, j, A\),
\[
c_{i,j,A} \leq \max\{\mu(d) \mid d \in D_G^A(t_{i+1} \ldots t_j)\} \cup \{0\}
\]

1: function CKY\((P, \mu, t_1 \ldots t_n)\)
2: \((c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N)\)
3: for \(1 \leq i \leq n\) do
4: for \(A \rightarrow t_i \in P\) do
5: \(c_{i-1,i,A} := \max\{c_{i-1,i,A}, \mu(A \rightarrow t_i)\}\)
6: \((c_{i-1,i,A} \mid A \in N) := \text{prune}((c_{i-1,i,A} \mid A \in N))\)
7: for \(2 \leq r \leq n\) do
8: for \(0 \leq i \leq n - r\) do
9: \(j := i + r\)
10: for \(m \in \{i + 1, i + 2, \ldots, j - 1\}\) do
11: for \(B, C \in N\) do
12: \(\text{if } c_{i,m,B} = 0 \text{ or } c_{m,j,C} = 0 \text{ then } \text{continue}\)
13: for \(A \in N\) such that \(A \rightarrow BC \in R\) do
14: \(c_{i,j,A} := \max\{c_{i,j,A}, \mu(A \rightarrow BC) \cdot c_{i,m,B} \cdot c_{m,j,C}\}\)
15: \((c_{i,j,A} \mid A \in N) := \text{prune}((c_{i,j,A} \mid A \in N))\)
16: return \(c\)
Pruning for CKY parsing

**Threshold beam**

*Require:* family $c = (c_{i,j,A} \in \mathbb{R} \mid A \in N)$, threshold $\theta \in [0,1]$  
*Ensure:* family $(c_{i,j,A} \in \mathbb{R} \mid A \in N)$

1: function `prune` ($c$)  
2: $m = \max_{A \in N} \{c_{i,j,A} \mid A \in N\}$  
3: for $A \in N$ do  
4: if $c_{i,j,A} < m \cdot \theta$ then  
5: $c_{i,j,A} := 0$  
6: return $c$
Pruning for CKY parsing

▶ Threshold beam

**Require:** family \( c = (c_{i,j,A} \in \mathbb{R} \mid A \in N) \), threshold \( \theta \in [0, 1] \)

**Ensure:** family \( (c_{i,j,A} \in \mathbb{R} \mid A \in N) \)

1: \quad \textbf{function} \ PRUNE(c)
2: \quad m = \max_{A \in N}\{c_{i,j,A} \mid A \in N\}
3: \quad \textbf{for} A \in N \textbf{ do}
4: \quad \quad \textbf{if} c_{i,j,A} < m \cdot \theta \textbf{ then}
5: \quad \quad \quad c_{i,j,A} := 0
6: \quad \quad \textbf{return} c

▶ Fixed-sized beam

**Require:** family \( c = (c_{i,j,A} \in \mathbb{R} \mid A \in N) \), size \( 1 \leq n \leq |N| \)

**Ensure:** family \( (c_{i,j,A} \in \mathbb{R} \mid A \in N) \)

1: \quad \textbf{function} \ PRUNE(c)
2: \quad [s_1, \ldots, s_n] = n\text{-best}\{c_{i,j,A} \mid A \in N\}
3: \quad \textbf{for} A \in N \textbf{ do}
4: \quad \quad \textbf{if} c_{i,j,A} < s_n \textbf{ then}
5: \quad \quad \quad c_{i,j,A} := 0
6: \quad \quad \textbf{return} c
Pruning for CKY parsing—implementation considerations

- No changes to data structures required.
- More speed-ups might be obtained by not adding items to chart which for sure would later be pruned:
  - Threshold beam: store weight $m$ of currently best item. If new item has weight $m$ below $\theta \cdot m$, it is safe to prune immediately.
  - Fixed-size beam: store weights of the $n$ best items. If the weight of new is below of worst item, prune immediately.
Pruning for deductive parsing

Require: weighted binary cfg \((N, \Sigma, P, S, \mu)\), word \(t_1 \ldots t_n\) where \(t_1, \ldots, t_n \in \Sigma\)

Ensure: family \((c_{i,j,A} : \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)\) such that

\[
c_{i,j,A} = \max\{\mu(d) \mid d \in D_G^A(t_{i+1} \ldots t_j)\} \cup \{0\}
\]

1: function \(\text{DEDUCE}(P, \mu, t_1 \ldots t_n)\)
2: \(\text{queue} := \{(i - 1, A, i, \mu(A \to t_i)) \mid 1 \leq i \leq n, A \to t_i \in P\}\)
3: \((c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N)\)
4: \(\text{while queue} \neq \emptyset \text{ do}\)
5: \((i, A, j, w) := \arg\max_{(i, A, j, w) \in \text{queue}} w\)
6: \(\text{queue} \setminus= \{(i, A, j, w)\}\)
7: \(\text{if } c_{i,j,A} = 0 \text{ then}\)
8: \(c_{i,j,A} := w\)
9: \(\text{queue} \cup= \{(i, A', j', \mu(A' \to AC) \cdot w \cdot c_{j',i',c}) \mid A' \to AC \in P\}\)
10: \(\text{queue} \cup= \{(i', A', j, \mu(A' \to BA) \cdot c_{i',i,B} \cdot w) \mid A' \to BA \in P\}\)
11: \(\text{queue} \cup= \{(i, A', j, \mu(A' \to A) \cdot w) \mid A' \to A \in P\}\)
12: \(\text{return } c\)
Pruning for deductive parsing

**Require:** weighted binary cfg \((N, \Sigma, P, S, \mu)\), word \(t_1 \ldots t_n \) where \(t_1, \ldots, t_n \in \Sigma\)

**Ensure:** family \((c_{i,j,A}: \mathbb{R} | 0 \leq i < j \leq n, A \in N)\) such that
\[
c_{i,j,A} \leq \max\{\mu(d) | d \in D_G^{A}(t_{i+1} \ldots t_j)\} \cup \{0\}
\]

1: function *deduce* \((P, \mu, t_1 \ldots t_n)\)
2: \(\text{queue} := \{(i - 1, A, i, \mu(A \rightarrow t_i)) | 1 \leq i \leq n, A \rightarrow t_i \in P\}\)
3: \((c_{i,j,A} := 0 | 0 \leq i < j \leq n, A \in N)\)
4: while \(\text{queue} \neq \emptyset\) do
5: \((i, A, j, w) := \text{argmax}_{(i, A, j, w) \in \text{queue}} w\)
6: \(\text{queue} \setminus= \{(i, A, j, w)\}\)
7: if \(c_{i,j,A} = 0\) then
8: \(c_{i,j,A} := w\)
9: \(\text{queue} \cup= \{(i', A', j', \mu(A' \rightarrow AC) \cdot w \cdot c_{j',i',c}) | A' \rightarrow AC \in P\}\)
10: \(\text{queue} \cup= \{(i', A', j, \mu(A' \rightarrow BA) \cdot c_{i',i,B} \cdot w) | A' \rightarrow BA \in P\}\)
11: \(\text{queue} \cup= \{(i, A', j, \mu(A' \rightarrow A) \cdot w) | A' \rightarrow A \in P\}\)
12: \(\text{prune}(\text{queue})\)
13: return \(c\)
Pruning for deductive parsing

► Threshold beam

Require: set \( queue \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \), threshold \( \theta \in [0, 1] \)
Ensure: set \( queue' \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \)

1: function \( \text{PRUNE}(queue) \)
2: \( m = \max_{(i,A,j,w)\in queue} w \)
3: return \( \{(i,A,j,w)\in queue \mid w > \theta \cdot m\} \)
Pruning for deductive parsing

Threshold beam

Require: set \( \text{queue} \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \), threshold \( \theta \in [0, 1] \)
Ensure: set \( \text{queue}' \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \)

1: function \( \text{prune}(\text{queue}) \)
2: \( m = \max_{(i,A,j,w) \in \text{queue}} w \)
3: return \( \{ (i, A, j, w) \in \text{queue} | w > \theta \cdot m \} \)

Fixed-sized beam

Require: set \( \text{queue} \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \), size \( n \in \mathbb{N} \)
Ensure: set \( \text{queue}' \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \)

1: function \( \text{prune}(\text{queue}) \)
2: \([i_1, \ldots, i_n] = n-\text{best}(\text{queue})\) w.r.t
3: return \( \{i_1, \ldots, i_n\} \)
Pruning for deductive parsing– implementation considerations

- Best implement queue as Min-max heap [Atk+86], because access to best and worst elements is required.
Pruning for deductive parsing– implementation considerations

- Best implement queue as Min-max heap [Atk+86], because access to best and worst elements is required.
- Again, don’t add items to queue if they would be pruned immediately.
Pruning for deductive parsing—implementation considerations

- Best implement queue as Min-max heap [Atk+86], because access to best and worst elements is required.
- Again, don’t add items to queue if they would be pruned immediately.
- Alternatively, one can shrink the queue only occasionally and not in each iteration of the main loop.

(Beware: Items for large spans are often more probable than items for small spans. Risk of pruning “good” large items in favour of “bad” small items. Solution: see A*-star parsing below)
Pruning for deductive parsing—implementation considerations

- Best implement queue as Min-max heap [Atk+86], because access to best and worst elements is required.
- Again, don’t add items to queue if they would be pruned immediately.
- Alternatively, one can shrink the queue only occasionally and not in each iteration of the main loop.

- Beware: Items for large spans are often more probable than items for small spans. Risk of pruning “good” large items in favour of “bad” small items.
Pruning for deductive parsing—implementation considerations

- Best implement queue as Min-max heap [Atk+86], because access to best and worst elements is required.
- Again, don’t add items to queue if they would be pruned immediately.
- Alternatively, one can shrink the queue only occasionally and not in each iteration of the main loop.

- Beware: Items for large spans are often more probable than items for small spans. Risk of pruning “good” large items in favour of “bad” small items.
Pruning for deductive parsing– implementation considerations

- Best implement queue as Min-max heap [Atk+86], because access to best and worst elements is required.
- Again, don’t add items to queue if they would be pruned immediately.
- Alternatively, one can shrink the queue only occasionally and not in each iteration of the main loop.

- Beware: Items for large spans are often more probable than items for small spans. Risk of pruning “good” large items in favour of “bad” small items. (Solution: see A*-star parsing below)
Outline

Pruning
  Pruning for CKY parsing
  Pruning for deductive parsing

$k$-best parsing

$A^*$-parsing
$k$-best parsing

- Problem: syntactic ambiguity, e.g., “She saw the astronomer with the telescope.”
Problem: syntactic ambiguity, e.g., “She saw the astronomer with the telescope.”
- “with the telescope” modifies “saw”
\textit{k-best parsing}

- Problem: syntactic ambiguity, e.g., “She saw the astronomer with the telescope.”
  - “with the telescope” modifies “saw”
  - “with the telescope” modifies “the astronomer”
Problem: syntactic ambiguity, e.g., “She saw the astronomer with the telescope.”
  ▶ “with the telescope” modifies “saw”
  ▶ “with the telescope” modifies “the astronomer”
Goal: given a sentence $w$, a PCFG $G$, and a positive integer $k$, find the $k$ most probable derivations of $G$ for $w$
**k-best parsing – naïve**

Require:  $k \in \mathbb{N}$, weighted binary CFG $(N, \Sigma, S, R, \mu)$, word $t_1 \cdots t_n$

Ensure:  $k$ most probable parse trees of PCFG for $t_1 \cdots t_n$

1: function $k\text{-BEST}(k, R, \mu, t_1, \ldots, t_n)$
2: $b[i, j, A] := [ ]$ for each cell $(i, j, A)$
3: for $i \in \{0, \ldots, n - 1\}$ do
4:    $c := \{(A(t_{i+1}), w) \mid A \rightarrow t_{i+1} \text{ in } R, w = \mu(A \rightarrow t_{i+1})\}$
5:    $b[i, j, A] = \text{take}(k, \text{sort}(c))$
6: for $z \in \{2, \ldots, n\}$ do
7:    for $i \in \{0, \ldots, n - z\}$ do
8:        $j := i + z$
9:    for $A \in N$ do
10:       $c := \{(A(d_1, d_2), w) \mid A \rightarrow BC \text{ in } R, m \in \{i + 1, \ldots, j - 1\},$
11:          $(d_1, w_1) \in b[i, m, B], (d_2, w_2) \in b[m, j, C],$
12:          $w = \mu(A \rightarrow BC) \cdot w_1 \cdot w_2\}$
13:       $b[i, j, A] = \text{take}(k, \text{sort}(c))$
14: return $b[0, n, S]$
10: $c := \{(A(d_1, d_2), w) \mid A \rightarrow BC, m \in \{i + 1, \ldots, j - 1\}$,
\hspace{1cm} $(d_1, w_1) \in b[i, m, B], (d_2, w_2) \in b[m, j, C],$
\hspace{1cm} $w = \mu(A \rightarrow BC) \cdot w_1 \cdot w_2\}$

11: $b[i, j, A] = \text{take}(k, \text{sort}(c))$

can be implemented as

10: $b[i, j, A] := [ ]$
11: for $m \in \{i + 1, \ldots, j - 1\}$ do
12: for $A \rightarrow BC$ in $R$ do
13: $c := \{(A(d_1, d_2), w) \mid (d_1, w_1) \in b[i, m, B], (d_2, w_2) \in b[m, j, C],$
\hspace{1cm} $w = \mu(A \rightarrow BC) \cdot w_1 \cdot w_2\}$
14: $b[i, j, A] := \text{mergeAndTakeK}(k, b[i, j, A], c)$
$k$-best parsing – merging more efficient

(a)

(b)

(c)

10: $b[i, j, A] := [ ]$
11: for $m \in \{i+1, \ldots, j-1\}$ do
12:   for $A \rightarrow BC$ in $R$ do
13:     denote $w_{u,v} := \mu(A \rightarrow BC) \cdot w_u^1 \cdot w_v^2$ where
14:     $(d_u^1, w_u^1) := b[i, m, B][u]$ and
15:     $(d_v^2, w_v^2) := b[m, j, C][v]$
16:     $F := \{(1, 1)\}$
17:     while $\max_{(u,v) \in F} w_{u,v} > \min_{(d,w) \in b[i,j,A]} w$ or $|b[i,j,A]| < k$ do
18:       $(u, v) = \arg\max_{(u,v) \in F} w_{u,v}$
19:       $F := (F \setminus \{(u, v)\}) \cup \{(u+1, v), (u, v+1)\}$

[HC05]
Outline

Pruning
- Pruning for CKY parsing
- Pruning for deductive parsing

k-best parsing

A*-parsing
A*-parsing

- weighted deductive parsing computes for each item \((i, j, A)\) in the chart the weight of the most probable derivation from \(A\) to \(t_{i+1} \cdots t_j\).
A*-parsing

- weighted deductive parsing computes for each item \((i,j,A)\) in the chart the weight of the most probable derivation from \(A\) to \(t_{i+1} \cdots t_j\).
- How about future costs, i.e., the weight of \(S \Rightarrow^*_G t_1 \cdots t_i A t_{j+1} \cdots t_n\)?
A*-parsing

- weighted deductive parsing computes for each item \((i, j, A)\) in the chart the weight of the most probable derivation from \(A\) to \(t_{i+1} \cdots t_j\).

- How about future costs, i.e., the weight of \(S \Rightarrow^*_{G} t_1 \cdots t_i A \ t_{j+1} \cdots t_n\)?

- If future costs are taken into account, then maybe less items from the queue need to be processed.
A*-parsing

- weighted deductive parsing computes for each item \((i, j, A)\) in the chart the weight of the most probable derivation from \(A\) to \(t_{i+1} \cdots t_j\).
- How about future costs, i.e., the weight of \(S \Rightarrow^*_G t_1 \cdots t_i \ A \ t_{j+1} \cdots t_n\)?
- If future costs are taken into account, then maybe less items from the queue need to be processed.
  - Why?: Usually items with small spans are more probable than items with large spans.
A*-parsing

- weighted deductive parsing computes for each item \((i, j, A)\) in the chart the weight of the most probable derivation from \(A\) to \(t_{i+1} \cdots t_j\).
- How about future costs, i.e., the weight of \(S \Rightarrow^*_G t_1 \cdots t_i \ A \ t_{j+1} \cdots t_n\)?
- If future costs are taken into account, then maybe less items from the queue need to be processed.
  - Why?: Usually items with small spans are more probable than items with large spans.
  - This is counteracted by future costs which are higher for items with small spans.

Klein and Manning [KM03] propose several admissible heuristics. A heuristic may also be useful when pruning items during CKY parsing.
A*-parsing

- weighted deductive parsing computes for each item \((i, j, A)\) in the chart the weight of the most probable derivation from \(A\) to \(t_{i+1} \cdots t_j\).
- How about future costs, i.e., the weight of \(S \Rightarrow^*_G t_1 \cdots t_i \ A \ t_{j+1} \cdots t_n\)?
- If future costs are taken into account, then maybe less items from the queue need to be processed.
  - Why?: Usually items with small spans are more probable than items with large spans.
  - This is counteracted by future costs which are higher for items with small spans.
- Klein and Manning [KM03] propose several admissible heuristics.
A*-parsing

- weighted deductive parsing computes for each item \((i, j, A)\) in the chart the weight of the most probable derivation from \(A\) to \(t_{i+1} \cdots t_j\).

- How about future costs, i.e., the weight of \(S \Rightarrow^*_G t_1 \cdots t_i A t_{j+1} \cdots t_n\)?

- If future costs are taken into account, then maybe less items from the queue need to be processed.
  - Why?: Usually items with small spans are more probable than items with large spans.
  - This is counteracted by future costs which are higher for items with small spans.

- Klein and Manning [KM03] propose several admissible heuristics.

- A heuristic may also be useful when pruning items during CKY parsing.
A*-parsing – Viterbi outside score

We use the admissible heuristic $\text{out}$:

$$\text{out}(A) = \max_{d \in D_G, u, w \in \Sigma^* : S \xrightarrow{d} G u A w} \text{weight}(d)$$

It can be computed by a variant of the inside/outside algorithm:

1: function $\text{INSIDE}$
2:     for $A \in N$ do
3:         $\text{in}(A) := \max(\{\mu(A \rightarrow \alpha) | A \rightarrow \alpha \in R\} \cup \{0\})$
4:     while not converged do
5:         for $A \in N$ do
6:             $\text{in}(A) = \max(\{\text{in}(A)\} \cup \{\mu(A \rightarrow BC) \cdot \text{in}(B) \cdot \text{in}(C) | A \rightarrow BC \in R\})$

7: function $\text{OUTSIDE}$
8:     set $\text{out}(B) := \begin{cases} 1 & B = S \\ 0 & \text{otherwise} \end{cases}$ for each $B \in N$
9:     while not converged do
10:        for $B \in N$ do
11:            $\text{out}(B) := \max(\{\text{out}(B)\} \cup \{\text{out}(A) \cdot \mu(A \rightarrow BC) \cdot \text{in}(C) | A \rightarrow BC \in R\}$
12:                $\cup \{\text{out}(A) \cdot \mu(A \rightarrow CB) \cdot \text{in}(C) | A \rightarrow CB \in R\})$
A*-parsing – parsing algorithm with heuristic

Require: weighted binary cfg \((N, \Sigma, P, S, \mu)\), word \(t_1 \ldots t_n\) where \(t_1, \ldots, t_n \in \Sigma\)
Ensure: family \((c_{i,j,A} : \mathbb{R} \mid 0 \leq i < j \leq n, A \in N)\) such that
\[
c_{i,j,A} = \max\{\mu(d) \mid d \in D^A_G(t_{i+1} \ldots t_j)\} \cup \{0\}
\]

1: function \textsc{deduce}(\(P, \mu, t_1 \ldots t_n\))
2: \hspace{1cm} \text{queue} := \{(i−1, A, i, \mu(A \rightarrow t_i)) \mid 1 \leq i \leq n, A \rightarrow t_i \in P\}
3: \hspace{1cm} (c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N)
4: \hspace{1cm} \textbf{while} \text{ queue} \neq \emptyset \textbf{ do}
5: \hspace{2cm} (i, A, j, w) := \text{argmax}_{(i, A, j, w) \in \text{queue}} w \cdot \text{out}(A)
6: \hspace{1cm} \text{queue} \setminus= \{(i, A, j, w)\}
7: \hspace{1cm} \textbf{if} \ c_{i,j,A} = 0 \textbf{ then}
8: \hspace{2cm} c_{i,j,A} := w
9: \hspace{1cm} \text{queue} \cup= \{(i', A', j', \mu(A' \rightarrow AC) \cdot w \cdot c_{j,j',A}) \mid A' \rightarrow AC \in P\}
10: \hspace{1cm} \text{queue} \cup= \{(i', A', j, \mu(A' \rightarrow BA) \cdot c_{i,i',B} \cdot w) \mid A' \rightarrow BA \in P\}
11: \hspace{1cm} \text{queue} \cup= \{(i, A', j, \mu(A' \rightarrow A) \cdot w) \mid A' \rightarrow A \in P\}
12: \hspace{1cm} \textbf{return} c
