Optimizations and Extensions for Weighted CFG Parsers

Richard Mörbitz

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

2021-06-21
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
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?
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.
Consider this slightly modified (red) version of the CKY algorithm:

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

An abstract “pruning operation“ on the chart can be included (lines 6, 15).
We speed up the algorithm by skipping the application of cfg rules to chart cells with 0 probability (line 12).

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 \(i, j, A\), \(c_{i,j,A} \leq \max\{\mu(d) \mid d \in D^A_G(t_{i+1} \ldots t_j)\} \cup \{0\}\)

1: function \(\text{CKY}(P, \mu, t_1 \ldots t_n)\)
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: if \(c_{i,m,B} = 0\) or \(c_{m,j,C} = 0\) then 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

How can the pruning operation be implemented?

▶ Threshold beam (set all cell probabilities to 0 if worse than $\theta \cdot$ best cell probability)

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

How can the pruning operation be implemented?

- **Threshold beam** (set all cell probabilities to 0 if worse than $\theta \cdot$ best cell probability)
  
  **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 \textsc{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 $c$
  ```

- **Fixed-sized beam** (set all but $n$ best cell probabilities to 0)
  
  **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: function \textsc{prune}(c)
  2:     $[s_1, \ldots, s_n] = n$-best$\{c_{i,j,A} \mid A \in N\}$
  3:     for $A \in N$ do
  4:         if $c_{i,j,A} < s_n$ then
  5:             $c_{i,j,A} := 0$
  6:     return $c$
  ```

$n$-best$(C)$ returns list of the $n$ highest values in $C$ in descending order
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 save 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

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

Pruning operation on queue is added.

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

1: function \textsc{deduce}(\(P, \mu, t_1 \ldots t_n\))
2: \(\text{queue} := \{(i-1, A, i, \mu(A \to t_i)) \mid 1 \leq i \leq n, A \to t_i \in P\}\)
3: \((c_{i,j,A} := 0 \mid 0 \leq i < j \leq n, A \in N)\)
4: 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: \(\text{queue} \cup= \{(i, A', j', \mu(A' \to AC) \cdot w \cdot c_{j,j',A}) \mid A' \to AC \in P\}\)
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{prune}(\text{queue})
13: return \(c\)
Pruning for deductive parsing

Again two options for pruning:

- **Threshold beam** (remove each queue item if its probability is worse than \( \theta \cdot \text{prob. of best queue item} \))

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

- **Fixed-sized beam** (only keep the \( n \) most probable queue items)

  **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 `prune(queue)`
  2: \[ i_1, \ldots, i_n \] = \( n \) - best \( \text{queue} \) w.r.t. 4th tuple component
  3: return \( \{i_1, \ldots, i_n\} \)
Pruning for deductive parsing

Again two options for pruning:

- **Threshold beam** (remove each queue item if its probability is worse than $\theta \cdot$ prob. of best queue item)

  **Require:** set $\textit{queue} \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}$, threshold $\theta \in [0, 1]$

  **Ensure:** set $\textit{queue}' \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}$

  1: function $\text{PRUNE}(\textit{queue})$

  2: $m = \max_{(i, A, j, w) \in \textit{queue}} w$

  3: return $\{(i, A, j, w) \in \textit{queue} | w > \theta \cdot m\}$

- **Fixed-sized beam** (only keep the $n$ most probable queue items)

  **Require:** set $\textit{queue} \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}$, size $n \in \mathbb{N}$

  **Ensure:** set $\textit{queue}' \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}$

  1: function $\text{PRUNE}(\textit{queue})$

  2: $[i_1, \ldots, i_n] = n\text{-best}(\textit{queue})$ w.r.t. 4th tuple component

  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.

[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.
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 \([\text{Atk}+86]\), 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
Problem: syntactic ambiguity, e.g., “She saw the astronomer with the telescope.”
k-best parsing

- Problem: syntactic ambiguity, e.g., “She saw the astronomer with the telescope.”
  - “with the telescope” modifies “saw”
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“

Each reading has a different parse tree.
Problem: syntactic ambiguity, e.g., “She saw the astronomer with the telescope.”
- “with the telescope” modifies “saw”
- “with the telescope” modifies “the astronomer“

Each reading has a different parse tree.

Solution: return multiple parse trees per sentence.
Problem: syntactic ambiguity, e.g., “She saw the astronomer with the telescope.”

- “with the telescope” modifies “saw”
- “with the telescope” modifies “the astronomer”

Each reading has a different parse tree.

Solution: return multiple parse trees per sentence.

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 algorithm**

Similar to CKY algorithm. Uses functions:
- `sort(c)` – sorts a set `c` of tuples (according to 2nd component)
- `take(k, ℓ)` – returns first `k` elements of list `ℓ`

**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 \( \text{kbest}(k, R, \mu, t_1, \ldots, t_n) \)
2: \( b[i,j,A] := \emptyset \) for each cell \((i,j,A)\)
3: for \( i \in \{0,\ldots, n-1\} \) and \( A \in N \) 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: \( b[i, i+1, A] = \text{take}(k, \text{sort}(c)) \)
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: \( c := \{(A(d_1, d_2), w) \mid A \rightarrow BC \text{ in } R, m \in \{i+1, \ldots, j-1\},\)
\( (d_1, w_1) \in b[i, m, B], (d_2, w_2) \in b[m, j, C],\)
\( w = \mu(A \rightarrow BC) \cdot w_1 \cdot w_2\} \)
11: \( b[i, j, A] = \text{take}(k, \text{sort}(c)) \)
12: return \( b[0, n, S] \)
$k$-best parsing – implementation of merging

10: $c := \{(A(d_1, d_2), w) \mid A \to BC, m \in \{i+1, \ldots, j-1\},$
   $(d_1, w_1) \in b[i, m, B], (d_2, w_2) \in b[m, j, C],$
   $w = \mu(A \to BC) \cdot w_1 \cdot w_2\}$

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

can be implemented as

10: $b[i, j, A] := \[]$
11: \textbf{for} $m \in \{i+1, \ldots, j-1\}$ \textbf{do}
12: \hspace{1em} \textbf{for} $A \to BC$ in $R$ \textbf{do}
13: \hspace{2em} $c := \{(A(d_1, d_2), w) \mid (d_1, w_1) \in b[i, m, B], (d_2, w_2) \in b[m, j, C],$
   \hspace{3em} $w = \mu(A \to BC) \cdot w_1 \cdot w_2\}$
14: \hspace{2em} $b[i, j, A] := \text{mergeAndTakeK}(k, b[i, j, A], c)$

here \text{mergeAndTakeK}$(k, \ell, c)$ returns the list of $k$-best items in the union of list $\ell$ and set $c$
k-best parsing – merging more efficiently [HC05]

the set $c$ can be computed lazily
avoid considering all $k^2$ items obtained by combining each of $b[i, m, B]$ with $b[m, j, C]$

instead, we only combine the best items of $b[i, m, B]$ with the best items of $b[m, j, C]$: if the combination of the $u$-th best of $b[i, m, B]$ and $v$-th best of $b[m, j, C]$ was considered, then consider also the combinations $(u + 1, v)$ and $(u, v + 1)$

<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

(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
\begin{align*}
(d_1^u, w_u^1) &:= b[i, m, B][u] \\
(d_2^v, w_v^2) &:= b[m, j, C][v]
\end{align*}
14: $F := \{(1, 1)\}$
15: 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
16: $(u, v) = \arg\max_{(u,v) \in F} w_{u,v}$
17: $b[i, j, A] := \text{insertAndTakeK}(k, b[i, j, A], \{(A(d_1^u, d_2^v), w_{u,v})\})$
18: $F := (F \setminus \{(u, v)\}) \cup \{(u + 1, v), (u, v + 1)\}$
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 \(t_{i+1} \cdots t_j\).
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.
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 $\text{out}$:

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

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

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

7. **function** `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.   $\text{out}(B) := \max(\{\text{out}(B)\} \cup \{\text{out}(A) \cdot \mu(A \to BC) \cdot \text{in}(C) \mid A \to BC \in R\}$
12.     $\cup \{\text{out}(A) \cdot \mu(A \to CB) \cdot \text{in}(C) \mid A \to CB \in R\}$
13.     $\cup \{\text{out}(A) \cdot \mu(A \to B) \mid A \to B \in R\})$
A*-parsing – parsing algorithm with heuristic

The out-value is now simply included when selecting the best item from the queue:

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 \textsc{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: while \(\text{queue} \neq \emptyset\) do
5: \((i, A, j, w) := \arg\max_{(i,A,j,w) \in \text{queue}} w \cdot \text{out}(A)\)
6: \(\text{queue} \setminus\{ (i, A, j, w) \}\)
7: if \(c_{i,j,A} = 0\) then
8: \(c_{i,j,A} := w\)
9: \(\text{queue} \cup= \{(i, A', j', \mu(A' \to AC) \cdot w \cdot c_{j',j',c}) \mid A' \to AC \in P\}\)
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: return \(c\)
Pruning for deductive parsing – in the context of CKY

Can A*-parsing be applied for CKY parsing?
Yes: in combination with pruning:

- 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: $m = \max_{A \in N} \{c_{i,j,A} \cdot \text{out}(A) \mid A \in N\}$
  3: for $A \in N$ do
  4:     if $c_{i,j,A} \cdot \text{out}(A) < m \cdot \theta$ then
  5:         $c_{i,j,A} := 0$
  6: return $c$
Pruning for deductive parsing – in the context of CKY

Can A*-parsing be applied for CKY parsing?
Yes: in combination with pruning:

▶ 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: \( m = \max_{A \in N} \{c_{i,j,A} \cdot \text{out}(A) \mid A \in N\} \)
3: for \( A \in N \) do
4: \hspace{1em} if \( c_{i,j,A} \cdot \text{out}(A) < m \cdot \theta \) then
5: \hspace{2em} \( c_{i,j,A} := 0 \)
6: \hspace{1em} return c

▶ Fixed-size 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: function PRUNE(c)
2: \( [s_1, \ldots, s_n] = n\text{-best}\{c_{i,j,A} \cdot \text{out}(A) \mid A \in N\} \)
3: for \( A \in N \) do
4: \hspace{1em} if \( c_{i,j,A} \cdot \text{out}(A) < s_n \) then
5: \hspace{2em} \( c_{i,j,A} := 0 \)
6: \hspace{1em} return c
