Context-free Algorithms

Jonathan Graehl
SDL Research *

July 20, 2005

Abstract

Algorithms on grammars/transducers with context-free derivations: hypergraph reachability, shortest path, and inside-outside pruning of 'relatively useless' arcs that are unused by any near-shortest paths.

1 Introduction

We present algorithms on context-free grammars (and also on hypergraphs and regular tree grammars, which share the same context-free derivation rule): hypergraph reachability, shortest path, and inside-outside pruning of 'relatively useless' arcs that are unused by any near-shortest paths. Section 2 is optional for those already familiar with context free grammars and their derivation trees.

2 Notation

2.1 Strings

$\Sigma^\star$ are the *strings over alphabet* Σ. For $s = (s_1, \ldots, s_n)$ the *length* of s is $|s| \equiv n$ and the *ith letter* is $s[i] \equiv s_i$, for all $i \in indices_s \equiv \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$, and the concatenation of a sequence of letters by index is $s[(f_1, \ldots, f_n) \in indices_s^\star] \equiv (s[f[1]], \ldots, s[f[n]])$. *Concatenation* of strings is specified by the $\cdot$ operator, where $a \cdot b \equiv (a[1], \ldots, a[|a|], b[1], \ldots, b[|b|])$.

2.2 Multisets

A *multiset* M *of* S is a partial function $M : S \to \mathbb{N}$, or equivalently, a functional binary relation $M \subset S \times \mathbb{N}$. The class of multisets of S is written $\mathcal{M}(S)$. If $M(s) = m \in \mathbb{N}$, we say $(x, m) \in M$, $x \in M$, and the *multiplicity of* x *in* M is m. Intuitively, the multiplicity is the number of times an element occurs. The *domain of* M is dom $M \equiv \{x \in M\}$. In some cases it is convenient to interpret M as a total function from $S \to (\mathbb{N} \cup \{0\})$ where $M(x \notin \text{dom } M) \equiv 0$. A set S can be interpreted as a multiset where each $x \in S$ has multiplicity $S(x) \equiv 1$. A sequence $V = (v_1, \ldots, v_n) \in S^\star$ can also be seen as a multiset with $V(x) \equiv \sum_{i:v_i=x} 1$ (after all, another notation of a multiset is just a set listed without removal of duplicates, e.g. $\{a, b, a\}$).

2.3 Trees

T_Σ is the set of *(rooted, ordered, labeled, finite) trees over alphabet* Σ.

$T_\Sigma(X)$ are the *trees over alphabet* Σ*, indexed by* X—the subset of $T_{\Sigma \cup X}$ where only leaves may be labeled by X. ($T_\Sigma(\emptyset) = T_\Sigma$.) *Leaves* are nodes with no children.

*Work done at University of Southern California, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292

The *nodes* of a tree t are identified one-to-one with its *paths*: $paths_t \subset paths \equiv \mathbb{N}^\star \equiv \bigcup_{i=0}^{\infty} \mathbb{N}^i$ ($A^0 \equiv \{()\}$). The path to the root is the empty sequence $()$, and p_1 *extended by* p_2 is $p_1 \cdot p_2$, where $\cdot$ is concatenation.

For $p \in paths_t$, $rank_t(p)$ is the number of children, or *rank*, of the node at p in t, and $label_t(p) \in \Sigma \cup X$ is its *label*. The *root of* t is $root(t) = label_t(())$. The *ranked label* of a node is the pair $labelandrank_t(p) \equiv (label_t(p), rank_t(p))$. For $1 \leq i \leq rank_t(p)$, the i^{th} *child* of the node at p is located at *path* $p \cdot (i)$. The *subtree at path* p *of* t is $t \downarrow p$, defined by $paths_{t\downarrow p} \equiv \{q \mid p \cdot q \in paths_t\}$ and $labelandrank_{t\downarrow p}(q) \equiv labelandrank_t(p \cdot q)$. The *children of* t are $children_t \in T^\star_\Sigma$, with $children_t[i] = t \downarrow (i), \forall 1 \leq i \leq rank(t)$.

The *paths to* X *in* t are $paths_t(X) \equiv \{p \in paths_t \mid label_t(p) \in X\}$. A *frontier* is a set of paths f that are *pairwise prefix-independent*:

$$\forall p_1, p_2 \in f, p \in paths : p_1 = p_2 \cdot p \implies p_1 = p_2$$

A *frontier of* t is a frontier $f \subseteq paths_t$.

For $t, s \in T_\Sigma(X), p \in paths_t$, $t[p \leftarrow s]$ is the *substitution of* s *for* p in t, where the subtree at path p is replaced by s. For a frontier f of t, the *mass substitution of* X *for the frontier* f *in* t is written $t[p \leftarrow X, \forall p \in f]$ and is equivalent to substituting the $X(p)$ for the p serially in any order.

The *yield of* X *in* t is $yield_t(X)$, the string formed by reading out the leaves labeled with X in left-to-right order. The usual case (the *yield of* t) is $yield_t \equiv yield_t(\Sigma)$.

We may also consider the *monadic strings* in t, $mstrings_t \subset \Sigma^\star$, obtained by reading off the labels along some path from the root down. The paths that read off a monadic string s in t are $mpaths^{\approx}_t(s) \equiv \{p \in paths_t \mid \forall 1 \leq i \leq |p| + 1 : label_t(p \downarrow (1, i)) \approx s[i]\}$, and the string of labels along a path is $mstring_t(p \in paths_t) \equiv \bullet_{i=1}^{|p|+1}(label_t(p \downarrow (1, i)))$ (so $\forall p \in mpaths^{\approx}_t(s) : mstring_t(p) \approx s$). Then $mstrings_t \equiv \{mstring_t(p \in paths_t)\}$ and $t \downarrow s$ is the sequence of *subtrees of* t *along the monadic string* s (in lexicographic path order):

$$t \downarrow^{\approx} s \in mstrings_t \equiv \bullet_{p \in mpaths^{\approx}_t(s) \text{ in lexicographic order}} (t \downarrow p)$$

Naturally, the path in t to the i^{th} element of $t \downarrow s$ is the i^{th} (in lexicographic order) $mpaths_t(s)$.

2.4 Regular Tree Grammars

A *weighted regular tree grammar* (**wRTG**) G is a quadruple (Σ, N, S, P), where Σ is the alphabet, N is the finite set of *nonterminals*, $S \in N$ is the *start (or initial) nonterminal*, and $P \subseteq N \times T_\Sigma(N) \times \mathbb{R}^+$ is the finite set of *weighted productions* ($\mathbb{R}^+ \equiv \{r \in \mathbb{R} \mid r > 0\}$). We define the binary relation $\Rightarrow_G$ (*single-step derives in* G) on $T_\Sigma(N) \times (paths \times P)^\star$, pairs of trees and *derivation histories*, which are logs of (location, production used):

$$\Rightarrow_G \equiv \Big\{((a, h), (b, h \cdot (p, (l, r, w))) \mid (l, r, w) \in P \wedge p \in paths_a(\{l\}) \wedge b = a[p \leftarrow r]\Big\}$$

where $(a, h) \Rightarrow_G (b, h \cdot (p, (l, r, w)))$ iff tree b may be derived from tree a by using the rule $l \rightarrow^w r$ to replace the nonterminal leaf l at path p with r. For a derivation history $h = ((p_1, (l_1, r_1, w_1)), \ldots, (p_n, (l_1, r_1, w_1)))$, the *weight of* h is $w(h) \equiv \prod_{i=1}^{n} w_i$, and call h *leftmost* if $L(h) \equiv \forall 1 \leq i < n : p_{i+1} \not<_{lex} p_i$.[1]

The reflexive, transitive closure of $\Rightarrow_G$ is written $\Rightarrow^\star_G$ (*derives in* G), and the restriction of $\Rightarrow^\star_G$ to leftmost derivation histories is $\Rightarrow^{L*}_G$ (*leftmost derives in* G).

The *weight of* a *becoming* b *in* G is $w_G(a, b) \equiv \sum_{h:(a,())\Rightarrow^{L*}_G(b,h)} w(h)$, the sum of weights of all unique (leftmost) derivations transforming a to b, and the *weight of* t *in* G is $W_G(t) = w_G(S, t)$. The *weighted regular tree language produced by* G is $L_G \equiv \{(t, w) \in T_\Sigma \times \mathbb{R}^+ \mid W_G(t) = w\}$.

The *derivation tree grammar* for a **wRTG** $G = (\Sigma, N, S, P)$ is $DG(G) = (P, N, S, P')$, where

$$P' \equiv \{(l, p(yield_N(r)), w) \mid p = (l, r, w) \in P\}$$

($p((s_1, \ldots, s_n) \in N^\star)$ is the tree with root label p, rank n, and ith child leaf s_i). The produced trees are called *derivation trees* and correspond one-to-one with tree-producing derivations in G.

[1] $() <_{lex} (a), (a_1) <_{lex} (a_2)$ iff $a_1 < a_2$, $(a_1) \cdot b_1 <_{lex} (a_2) \cdot b_2$ iff $a_1 < a_2 \vee (a_1 = a_2 \wedge b_1 <_{lex} b_2)$

2.5 Hypergraphs

A *(directed) hypergraph* G is a pair $G = (V, E)$ where V is a set of *vertices* (or *nodes*) of G, and E are the *edges* (or *hyperarcs*) of G. An edge $e = (h_e \in V, T_e, c_e : \mathbb{R}^{|T_e|} \to \mathbb{R})$ has *head* h_e, *tails* T_e, and *cost function* c_e. The cost function for an edge maps the costs of reaching its tails to the cost of reaching the head through that edge.

In a hypergraph, $T_e \subseteq V$—the tails are subsets of the vertices.

In an *ordered multi-hypergraph*, $T_e \in V^\star$—the tails are ordered sequences.

Typically hyperarc cost functions are symmetric; if not, then the order of arguments is the same as the order of tails. , or for unordered hypergraphs, fixed by some arbitrary total order $<_G$ on V. The usual cost function is given by $c_e(x_1, \ldots, x_n) \equiv l_e + \sum_{i=1}^{n} x_i$, where l_e is the *length* of the edge. A typical asymmetric cost function would combine tail hyperpath costs with different weights for each tail.

We say there is a *hyperpath from* $X \subseteq V$ *to* $y \in V$ *in* $G = (V, E)$, written $X \rightsquigarrow_G y$, if $y \in X \vee \exists e \in E : h_e = y \wedge \forall t \in T_e : X \rightsquigarrow_G t$. A *hyperpath-tree* $t \in (X \rightsquigarrow_G y)$ is a tree labeled by edges, corresponding to a proof of $X \rightsquigarrow_G y$ (with a separate proof for each multiple occurrence of a tail vertex - note: the usual B-hyperpath allows only a single incoming hyperarc/proof of each vertex - our hyperpath-trees are more like derivations in a context-free grammar). The *cost* of a hyperpath-tree p is written $c(p)$ and is computed bottom-up for each subtree with root label e using c_e.

For any derivation grammar $G' = (P, N, S, P')$ of **wRTG** $G = (\Sigma, N, S, P)$, there is an equivalent ordered multi-hypergraph $H = (N \cup \{\omega\}, E)$ with an edge $e \in E$ for each production $p = (l, r, w) \in P'$ such that $h_e = l$, $T_e = \begin{cases} \{\omega\} & \text{if } yield_N(r) = \emptyset \\ yield_N(r) & \text{otherwise} \end{cases}$, and the usual cost function with $l_e = -\ln w$. The hyperpath-trees $\omega \rightsquigarrow_H S$ are exactly the derivation trees for G, with the cost of the hyperpath-tree equal to the $\ln$ of the weight of the tree (obviously, the labels of the hyperpath-tree are $e \in E$ and the labels of the derivation tree are $p \in P$, but there is an isomorphism between them, due to the construction of E).

A hypergraph (V, E) may be interpreted as a multigraph (V, E') with an edge for every tail of each hyperarc ($E' = \{(h_e, t \in T_e, c_e) \mid (h_e, T_e, c_e) \in E\}$). We can refer to *simple* (or *monadic*) paths corresponding to the usual paths in the graph. In fact, monadic strings s of hyperarcs from a hyperpath-tree for (V, E) correspond to a simple path in $h_{s[|s|]} \rightsquigarrow_{(V,E')} h_{s[1]}$.

3 Pruning Along a Hyperpath-Tree

If we are only interested in hyperpath-trees $X \rightsquigarrow_G y$, we can *prune* G *along* X *to* y by eliminating vertices and hyperarcs that don't appear in any (cheap) hyperpath-tree. This is analogous to the problem of reducing a context free grammar by eliminating useless nonterminals (Hopcroft and Ullman, 1979), except that we wish to also eliminate those useful only for high-cost hyperpath-trees.

Since we care only for the existence of a (cheapest) path for each node, tails of edges may be considered as sets while addressing this problem, so that multiply appearing tails t in a multi-hypergraph always reuse the same hyperpath-tree $X \rightsquigarrow_G t$. We assume the cost function $c_e(c) = l_e + \sum_{(t,m) \in T_e} w_e(t) mc(t)$, where $c(t)$ is the cost due to the hyperpath-tree $X \rightsquigarrow t$ and $w_e(t)$ is a weight given to t-tails of that edge.

Unweighted pruning consists of first eliminating vertices (and hyperarcs they occur in) that cannot be reached from the start, and second, eliminating from the remainder all those that do not lie along any hyperpath-tree to the destination. The first step can be performed in linear time by Algorithm 1.

The weighted version of Algorithm 1 establishes the lowest cost way of reaching each vertex from a start set (or that there is none). Algorithm 2, adapted from (Knuth, 1977) (first published in (Knight and Graehl, 2005)), is an extension of the graph shortest path problem (Dijkstra, 1959) to the hypergraph case. It works the same except that vertices are visited in increasing order of the cost of reaching them from X, and so requires a priority queue. Activated hyperarcs serve to potentially lower the cost of reaching their head, but visiting the head is deferred until it is certain that its minimal cost hyperpath-tree is known. This is in contrast to the simple depth first approach in the unweighted case, where the head is visited immediately with a recursive function call (using the implicit program stack for queuing nodes).

Algorithm 1: Single-source-set hypergraph reachability

Input:
A set of source nodes $X \subseteq V$ in a hypergraph $G = (V, E)$, nodes V, and hyperarcs $E = \{e_1, \ldots, e_m\}$ indexed by $1 \leq i \leq m$. Each hyperarc has tail nodes $T_i \subseteq V \equiv T_{e_i}$ and head $h_i \in V \equiv h_{e_i}$.

Output:
For all $y \in V$, $B[y] =$ **true** if $X \rightsquigarrow_G y$, **false** otherwise. Time complexity is $O(t)$ where t is the total size of the input.

begin
  for $y \in V$ **do** $B[y] \leftarrow$ **false**
  Adj$[y] \leftarrow \{\}$

  for $1 \leq i \leq e$, *index of a hyperarc* $(T_i = \{x_1, \ldots, x_k\}) \rightarrow \{h_i\}$ **do**
    $r[i] \leftarrow k$
    `/* r[i] is the number of tail nodes remaining before edge i fires. */`
    for $1 \leq j \leq k$ **do** *Adj*$[x_j] \leftarrow$ *Adj*$[x_j] \cup \{i\}$

  for $y \in X$ **do REACH**(y)

REACH$(y) \equiv$ **begin**
  if $\neg B[y]$ **then**
    $B[y] \leftarrow$ **true**
    for $i \in$ *Adj*$[y]$ **do**
      if $\neg B[h_i]$ **then**
        $r[i] \leftarrow r[i] - 1$
        if $r[i] = 0$ **then REACH**(h_i)

Algorithm 2: ViterbiInside: single-source-set, multi-destination shortest hyperpath-trees.

Input:
A set of source nodes $X \subseteq V$ with initial costs $\{i_x, \forall x \in X\}$, and a hypergraph with n nodes V, and m hyperarcs $(e_1, \ldots, e_m)$ indexed by $1 \leq i \leq m$. Each hyperarc has tail nodes $T_i \subseteq V \equiv T_{e_i}$, head $h_i \in V \equiv h_{e_i}$, and superior cost function $c_i \equiv c_{e_i}$ (f is *superior* iff $f(x_1, \ldots, x_k) \geq x_i, \forall 1 \leq i \leq k$ (Knuth, 1977)) of variables T_i. The cost functions are implemented by constant time operations **BIND**$(c_i, y \in T_i,$ cost of $y)$ and **INF**(c_i), which returns a lower bound on the cost given the variables bound so far.
For a context-free grammar or regular tree grammar, introduce a fictitious sink nonterminal ω to the rhs of terminal rules. Now let the V be the nonterminals, and let X be ω. For each ith rule, let h_i be the lhs nonterminal, T_i be the set of rhs nonterminals (or ω if there are none). Finally, initialize **INF**(c_i) to $w_i = -\log P(i|h_i)$, the negative log rule probability of rule i, and define **BIND**$(c_i, y \in T_i, c)$ as increasing **INF**(c_i) by $\#_i(y)c$, where $\#_i(t)$ is the number of occurrences of nonterminal t in rule i.
Output:
For all $v \in V$, $\pi[v] = i$ is the index of the cheapest hyperarc with head $h_i = v$, giving the predecessor relation of the cheapest unordered hyperpath-tree from the $X \rightsquigarrow t$), and $\beta[v]$ is minimum cost of reaching v. $\pi[v] = 0$ if there is no cost-improving edge to v. Time complexity is $O(n \lg n + t)$ where (t is the total size of the input) if a Fibonacci heap is used, or $O(m \lg n + t)$ if a binary heap is used.

begin
  for $y \in V$ **do**
    if $y \in X$ **then** $\beta[y] \leftarrow i_y$
    else $\beta[y] \leftarrow \infty$
    $\pi[y] \leftarrow 0$
    Adj$[y] \leftarrow \{\}$
  $Q \leftarrow$ **HEAP-CREATE**()
  for $x \in X$ **do HEAP-INSERT**(Q, x, i_x)
  for $1 \leq i \leq m$, *index of a hyperarc* $(T_i = \{x_1, \ldots, x_k\}) \rightarrow^{c_i} \{h_i\}$ **do**
    $r[i] \leftarrow k$
    `/* r[i] is the number of tail nodes remaining before edge i fires. */`
    for $1 \leq j \leq k$ **do** *Adj*$[x_j] \leftarrow$ *Adj*$[x_j] \cup \{i\}$
  while $Q \neq \emptyset$ **do**
    $y \leftarrow$ **HEAP-EXTRACT-MIN**(Q)
    for $i \in$ *Adj*$[y]$ **do**
      `/* edge i with y as a tail */`
      if ***INF***$(c_i) < \beta[h_i]$ **then**
        BIND$(c_i, y, \beta[y])$
        $r[i] \leftarrow r[i] - 1$
        if $r[i] = 0$ **then**
          $c \leftarrow$ **INF**(c_i)
          if $c < \beta[h_i]$ **then**
            if $\beta[h_i] = \infty$ **then HEAP-INSERT**(Q, h_i, c)
            else HEAP-DECREASE-KEY(Q, h_i, c)
            $\pi[h_i] \leftarrow i$
            $\beta[h_i] \leftarrow c$

Having eliminated parts of the hypergraph that aren't reachable from X, it still remains to further remove any parts that don't contribute to reaching y. In Algorithm 3, we perform a simple depth-first traversal from heads to tails of hyperarcs, starting with the destination y, ultimately saving only vertices that can help reach y.

To see how this works, let the *restriction* of hypergraph $G = (V, E)$ to a subset of its vertices $V' \subseteq V$ be $G\langle V'\rangle \equiv (V', E) : E' = \{e \in E \mid h_e \in V' \wedge T_e \subseteq V'\}$. First, run Algorithm 1 on G to find $V' = \{v \in V' \mid X \rightsquigarrow_G\}$, then second, run Algorithm 3 on the resulting restriction $G' = G\langle V'\rangle$ to find $V'' = \{v \in V' \mid \exists F \supseteq \{v\} : F \rightsquigarrow_{G'} y$. Then the hypergraph $G'' = G'\langle V''\rangle$ has the same hyperpath-trees $X \rightsquigarrow_{G''} y$ as G, and is the minimal such.

The order of these steps is essential - there may be vertices that only help reach y through hyperarcs that are eliminated in Algorithm 1. In the second step, we qualify each node $t \in T_e$ that is connected through e to y as participating in a path to $X \rightsquigarrow_G h_e$ automatically, which is sound only if we can assume some path from $X \rightsquigarrow_G t'$, for all $t' \in T_e$. But the first step guarantees this by removing all nodes that aren't reachable from X.

Algorithm 3: Single-destination hypergraph reachability

Input:
A destination node $y \in V$ in a hypergraph $G = (V, E)$, with n nodes V, and m hyperarcs $E = \{e_1, \ldots, e_m\}$ indexed by $1 \leq i \leq m$. Each hyperarc has tail nodes $T_i \subseteq V \equiv T_{e_i}$ and head $h_i \in V \equiv h_{e_i}$.
Output:
For all $x \in V$, $A[x] =$ **true** if there is a hyperpath-tree $X \rightsquigarrow_G y$ such that $x \in X$, **false** otherwise. Time complexity is $O(t)$ where t is the total size of the input (this is simple depth-first search on the projected regular graph).

```
begin
    for x ∈ V do A[x] ← false

    USE(y)

USE(y) ≡ begin
    A[y] ← true
    for t ∈ T_i do
        if ¬A[t] then
            USE(t)
```

What we are really doing is reversing a hypergraph by interpreting it as a monadic graph consisting of all edges formed by selecting just one tail of each hyperarc, and plugging in a default rule for completing the omitted siblings. We can extend this strategy to the weighted case, using the shortest hyperpath-tree $X \rightsquigarrow v$ ($\pi[v]$) (from from Algorithm 2) for each omitted sibling v. Then we can attribute to each monadic arc the cost of those omitted hyperpath-trees ($\beta[v]$), in addition to the cost of its original hyperarc. Then we can perform the usual single-source shortest graph paths computation(Dijkstra, 1959) on the this reverse monadic graph.

Since any subtree of a shortest hyperpath-tree $t \in (X \rightsquigarrow y)$ is a shortest hyperpath-tree from X to its root-head $h_{label_t(())}$, we can decompose the shortest hyperpath-tree using node v into the shortest *inside* $X \rightsquigarrow v$ plus the *outside* $v \rightsquigarrow y$ formed by reconstituting a path in the monadic graph with the default interpretation of omitted siblings. The outside part is an almost-hyperpath-tree, missing only an inside subtree for $X \rightsquigarrow v$ (an outside tree would be a hyperpath-tree from $X \cup \{v\} \rightsquigarrow y$). This is the insight behind the inside-outside algorithm(Lari and Young, 1990) for training context free string grammars, and also its extension to training tree transducers(Graehl and Knight, 2004).

Note that this decomposition means that the cost functions for hyperarcs must be separable into an independent sum over parts due to the tails and a part due to the arc.

In Algorithm 4, we implicitly perform this reversal and monadification of a hypergraph and obtain for each vertex v the cheapest way to complete the hyperpath-tree $X \rightsquigarrow v$ into $X \rightsquigarrow v \rightsquigarrow y$ (by that we mean adjoining some inside hyperpath-tree $X \rightsquigarrow v$ with , using parent $\psi[v]$ with total outside cost (leaving out the cost of $X \rightsquigarrow v$) $\alpha[v]$.

Then, the *utility* of v, or the cost of the cheapest hyperpath-tree using it, is just $\gamma[v] \equiv \alpha[v] + \beta[v]$ and the utility of hyperarc e is $\gamma[e] \equiv \alpha[h_e] + l_e + \sum_{(t,m)\in T_e} m\beta[t]$. It is then easy to select vertices and edges for removal based

on some criteria on their utility relative to the cost of the cheapest hyperpath-tree $X \rightsquigarrow y$, which is $\beta[y]$.

Algorithm 5 selects the minimal subset of the hyperarcs and vertices necessary to include the best hyperpath-tree $x \rightsquigarrow y$ with cost $\beta[y]$ and all hyperpath-trees with cost no worse than $\beta[y] + \delta$.

References

Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:269–271.

Graehl, Jonathan and Kevin Knight. 2004. Training tree transducers. In *Proceedings of the 2004 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-04)*.

Hopcroft, John and Jeffrey Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley, London.

Knight, K. and J. Graehl. 2005. An overview of probablistic tree transducers for natural language processing. In *Proceedings of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*.

Knuth, D. 1977. A generalization of Dijkstra's algorithm. *Info. Proc. Letters*, 6(1).

Lari, K. and S. J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language, 4*, pages 35–56.

Algorithm 4: ViterbiOutside - single-destination, shortest outside hyperpath-trees

Input:
A destination $y \in V$ and default (inside) costs $\beta[v]$ for reaching each $v \in V$ from X (computed with **ViterbiInside**), for a hypergraph with n nodes V, and m hyperarcs $(e_1, \ldots, e_m)$ indexed by $1 \leq i \leq m$. Each hyperarc has length (i.e. cost to use) $l_i \equiv l_{e_i}$, a multiset of tails $T_i \equiv T_{e_i} \in \mathcal{M}(V)$, and head $h_i \in V \equiv h_{e_i}$. The cost for hyperpath-tree from $X \rightsquigarrow h_e$ using edge e and the best hyperpath-trees from X to each of its tails t with cost $\beta[t]$ is $c_e = l_e + \sum_{(t,m)\in T_i} m\beta[t]$ (where m is the number of occurrences of t in the tails), but other cost functions are possible - what is important is the ability to build up the cost for using an edge assuming the default for its tails, and later subtract out the contribution from the default of a single instance of a tails.

Output:
For all $v \in V$, $\psi[v]$ is the index of the hyperarc used to reach y from v (or 0 if none was taken) with the minimum outside cost $\alpha[v]$=$\beta[y] - \beta[v]$ given by assuming the default cost way to was used to reach its siblings from X. Time complexity is $O(n \lg n + t)$ where (t is the total size of the input) if a Fibonacci heap is used, or $O(m \lg n + t)$ if a binary heap is used.

begin

  for $x \in V$ **do**
    $\psi[x] \leftarrow 0$
    $\alpha[x] \leftarrow \infty$
    $Adj^{-1}[x] \leftarrow \{\}$
  for $1 \leq i \leq m$, *index of a hyperarc* $(T_i = \{x_1, \ldots, x_k\}) \rightarrow^{l_i} \{h_i\}$ **do**
    for $1 \leq j \leq k$ **do** $Adj^{-1}[h_i] \leftarrow Adj^{-1}[h_i] \cup \{x_j\}$
  $\alpha[y] \leftarrow 0$
  $Q \leftarrow$ **HEAP-CREATE**()
  HEAP-INSERT$(Q, y, 0)$
  while $Q \neq \emptyset$ **do**
    $x \leftarrow$ **HEAP-EXTRACT-MIN**(Q)
    for $i \in Adj^{-1}[x]$ **do**
      /* edge i with x as a head */
      $c \leftarrow \alpha[x] + l_i + \sum_{(t,m)\in T_i} m\beta[t]$ /* c=total cost of $X \rightsquigarrow e_i \rightsquigarrow y$ */
      for $t \in T_i$ **do**
        $c' \leftarrow c - \beta[t]$ /* c' is the proposed improved outside cost for t through e_i, removing $X \rightsquigarrow t$ */
        if $c' < \alpha[t]$ **then**
          if $\alpha[h_i] = \infty$ **then HEAP-INSERT**(Q, t, c')
          else HEAP-DECREASE-KEY(Q, t, c')
          $\psi[t] \leftarrow i$
          $\alpha[t] \leftarrow c'$

Algorithm 5: Prune relatively-useless vertices and hyperarcs

Input:
$\beta[v]$ and $\alpha[v]$, the Viterbi inside and outside costs of each vertex V over all hyperpath-trees from $X \rightsquigarrow y$ (computed with **ViterbiInside** and **ViterbiOutside**) in a hypergraph $G = (V, E)$ with m hyperarcs $E = \{e_1, \ldots, e_m\}$ indexed by $1 \leq i \leq m$. Each hyperarc has tail nodes $T_i \subseteq V \equiv T_{e_i}$ and head $h_i \in V \equiv h_{e_i}$. The cost for hyperpath-tree from $X \rightsquigarrow h_e$ using edge e and the best hyperpath-trees from X to each of its tails t with cost $\beta[t]$ is $c_e = l_e + \sum_{t \in T_i} m_t \beta[t]$, where l_e is the weight on hyperarc e and m_t is a weight, e.g. the number of occurrences of t in the rhs of a grammar production.
δ is a beam (cost distance from the best hyperpath-tree).

Output:
For all $x \in V \cup E$, $\gamma[x]$ is the cost of the best hyperpath-tree $t \in (X \rightsquigarrow_G y)$ such that x is used in t, or ∞ if none exists, $\kappa[x] =$ **true** iff that cost is not more worse than δ from the best $\beta[y]$.
Time complexity is $O(t)$ where t is the total size of the input. (total complexity including **ViterbiInside** is $O(n \lg n + t)$).

begin

$l \leftarrow \beta[y] + \delta$
for $v \in V$ **do**
  $\gamma[v] \leftarrow \beta[v] + \alpha[v]$
for $e \in E$ **do**
  $\gamma[e] \leftarrow \alpha[h_e] + l_e + \sum_{t \in T_i} m_t \beta[t]$
for $x \in V \cup E$ **do** $\kappa[x] \leftarrow (\gamma[x] \leq l)$