$k$-best Knuth Algorithm

Liang Huang

August 11, 2005

Knuth (1977) describes an elegant extension of the Dijkstra (1959) algorithm for the case of directed hypergraphs, which is essentially a best-first (also known as uniform-cost) search algorithm for the 1-best hyperpath. The A* parsing of Klein and Manning (2003), for example, is an instance of it on binary-branching hypergraphs with admissible heuristics.

This note further extends the Knuth algorithm to the $k$-best case, in the fashion of the simple $n$-best Dijkstra algorithm (Mohri and Riley, 2002). The $k$-best Knuth algorithm also exploits the lazy frontier idea adapted from $k$-best Viterbi parsing (Jiménez and Marzal, 2000; Huang and Chiang, 2005).

1 Knuth Algorithm

The pseudo-code in Algorithm 1 is adapted from (Knight and Graehl, 2005) which provides an efficient implementation of Knuth’s original algorithm.

2 $k$-best Dijkstra Algorithm

The $k$-best extension of Dijkstra seems to be a folklore from the 1960s. The pseudo-code presented in Algorithm 2 is adapted from (Mohri and Riley, 2002).

3 $k$-best Knuth Algorithm

We combine the ideas in the previous section and the lazy frontier idea in (Jiménez and Marzal, 2000; Huang and Chiang, 2005) to the Knuth algorithm (pseudo-code in Algorithm 3).

Define $1_m$ to be the vector of length $m$ whose components are all 1 and $b^i_m$ the vector of length $m$ whose components are all 0 except the $i^{th}$ is 1.

References


Algorithm 1: 1-best Knuth: The Knuth 1977 Algorithm

Input:
Output:

begin
  \( Q \leftarrow \{s\} \)
  \( d(s) \leftarrow 0 \)
  foreach \( e \in E \) do
    \( r[e] \leftarrow |T(e)| \)
    // \( r[e] \) is the number of tail nodes remaining before edge \( e \)
    fires.
while \( Q \neq \emptyset \) do
  \( u \leftarrow \text{Extract-Best}(Q) \)
  foreach \( e \in FS(u) \) do
    \( e = ((u_1, u_2, \ldots, u_i = u, \ldots, u_m), h(e) = v, f_e) \)
    \( r[e] \leftarrow r[e] - 1 \)
    if \( r[e] = 0 \) then
      \( d(v) \oplus = f_e(d(u_1), d(u_2), \ldots, d(u), \ldots, d(u_m)) \)
      \( \text{Improve-Key}(Q, v) \)
end

Algorithm 2: \( k \)-best Dijkstra: The \( k \)-best Dijkstra Algorithm

Input:
Output:

begin
  foreach \( v \in V \) do \( n[v] \leftarrow 0 \)
  //\( n[v] \) is the number of times vertex \( v \) is popped from the queue
  \( Q \leftarrow \{(s, 0)\} \)
while \( Q \neq \emptyset \) do
  \( (u, c) \leftarrow \text{Extract-Best}(Q) \)
  \( n[u] \leftarrow n[u] + 1 \)
  if \( n[u] \leq k \) then
    foreach \( e \in FS(u) \) do
      \( e = (u, v, f_e) \)
      \( d' \leftarrow f_e(c) \)
      // more general than \( c \otimes w(e) \)
      \( \text{Enqueue}(Q, (v, c')) \)
end
Algorithm 3: $k$-best Knuth: The $k$-best Knuth Algorithm

Input:
Output:

begin
  foreach $v \in V$ do $n[v] \leftarrow 0$
  // $n[v]$ is the number of times vertex $v$ is popped from the queue
  foreach $e \in E$ do
    $r[e] \leftarrow |T(e)|$
    // $r[e]$ is the number of tail nodes remaining before edge $e$
    fires
    for $i \leftarrow 1 \ldots |T(e)|$ do $\text{need}[e][i] = 1$
    $Q \leftarrow \{(s, 0, \text{null}, 0)\}$
  while $Q \neq \emptyset$ do
    $(u, c, e_0, \hat{j}) \leftarrow \text{Extract-Best}(Q)$
    // $c$ is the cost; $e_0$ and $\hat{j}$ are backpointers
    $n[u] \leftarrow n[u] + 1$
    if $n[u] \leq k$ then
      $d_{n[u]}(u) \leftarrow c$
      // the cost of $u$’s $n[u]^{th}$ best derivation is $c$
    foreach $e \in FS(u)$ do
      $e = ((u_0, \ldots, u_m), v, f_e)$
      if $n[u] = 1$ then
        // a new tail node ready
        $r[e] \leftarrow r[e] - \{|i \mid u_i = u\}$
        // # of occurrences of $u$ in the tail
      if $r[e] = 0$ then
        // fire this edge $e$
        for $i \leftarrow 1 \ldots m$ do
          if $u_i = u$ and $\text{need}[e][i] = n[u]$ then
            // needed on $i^{th}$ dimension
            $c'' \leftarrow f_e(d_{i}(u_0), \ldots, d_{i}(u_{i-1}), d_{n[u]}(u), d_1(u_{i+1}), \ldots, d_1(u_m))$
            $\text{Enqueue}(Q, (v, c', e, 1_m + b^i_m \cdot (n[u] - 1)))$
            $\text{need}[e][i] = 0$
      if $n[u] < k$ and $e_0 \neq \text{null}$ then
        // get successor along the edge $e_0$
        $e_0 = ((x_1, \ldots, x_p), u_0, f_0)$
        for $l \leftarrow 1 \ldots p$ do
          $\hat{j}' \leftarrow j + b^l_p$
          if $\hat{j}' \leq \min(n[x_i], k)$ then
            $c''' \leftarrow f_0(d_{\hat{j}'}(x_1), \ldots, d_{\hat{j}'}(x_l), \ldots, d_{\hat{j}'}(x_p))$
            $\text{Enqueue}(Q, (u, c'', e_0, \hat{j}'))$
          else
            $\text{need}[e_0][l] = j'$
    end
end


