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 admissable 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 algorithmn (pseudo-code in Algorithm 3).

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

References

Dijkstra, Edsger W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, (1):267–271.

Huang, Liang and David Chiang. 2005. Better k-best parsing. Technical Report MS-CIS-05-08, University of Pennsylvania.

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

```
Input:
Output:
begin
    Q ← {s}
    d(s) ← 0
    foreach e ∈ E do
        r[e] ← |T(e)|
        // r[e] is the number of tail nodes remaining before edge e
           fires.
    while Q ≠ ∅ do
        u ← Extract-Best(Q)
        foreach e ∈ FS(u) do
            e = ((u_1, u_2, ..., u_i = u, ..., u_m), h(e) = v, f_e)
            r[e] ← r[e] − 1
            if r[e] = 0 then
                d(v)⊕ = f_e(d(u_1), d(u_2), ..., d(u), ..., d(u_m))
                Improve-Key(Q, v)
end
```

Algorithm 2: k**-best Dijkstra**: The k-best Dijkstra Algorithm

```
Input:
Output:
begin
    foreach v ∈ V do n[v] ← 0
    //n[v] is the number of times vertex v is popped from the queue
    Q ← {(s, 0)}
    while Q ≠ ∅ do
        (u, c) ← Extract-Best(Q)
        n[u] ← n[u] + 1
        if n[u] ≤ k then
            foreach e ∈ FS(u) do
                e = (u, v, f_e)
                c' ← f_e(c)
                // more general than c ⊗ w(e)
                Enqueue(Q, (v, c'))
end
```

Algorithm 3: k**-best Knuth**: The k-best Knuth Algorithm

```
Input:
Output:
begin
    foreach v ∈ V do n[v] ← 0
    // n[v] is the number of times vertex v is popped from the queue
    foreach e ∈ E do
        r[e] ← |T(e)|
        // r[e] is the number of tail nodes remaining before edge e
          fires
        for i ← 1 ... |T(e)| do need[e][i] = 1
    Q ← {(s, 0, null, 0)}
    while Q ≠ ∅ do
        (u, c, e_0, j) ← Extract-Best(Q)
        // c is the cost; e_0 and j are backpointers
        n[u] ← n[u] + 1
        if n[u] ≤ k then
            d_{n[u]}(u) ← c
            // the cost of u's n[u]^{th} best derivation is c
            foreach e ∈ FS(u) do
                e = ((u_1, u_2, ..., u_m), v, f_e)
                if n[u] = 1 then
                    // a new tail node ready
                    r[e] ← r[e] − |{i | u_i = u}|
                    // # of occurrences of u in the tail
                if r[e] = 0 then
                    // fire this edge e
                    for i ← 1 ... m do
                        if u_i = u and need[e][i] = n[u] then
                            // needed on i^{th} dimension
                            c' ← f_e(d_1(u_1), ..., d_1(u_{i−1}), d_{n[u]}(u), d_1(u_{i+1}), ..., d_1(u_m))
                            Enqueue(Q, (v, c', e, 1_m + b_m^i · (n[u] − 1)))
                            need[e][i] = 0
        if n[u] < k and e_0 ≠ null then
            // get successor along the edge e_0
            e_0 = ((x_1, x_2, ..., x_p), u, f_0)
            for l ← 1 ... p do
                j' ← j + b_p^l
                if j'_l ≤ min(n[x_l], k) then
                    c'' ← f_0(d_{j'_1}(x_1), ..., d_{j'_l}(x_l), ..., d_{j'_p}(x_p))
                    Enqueue(Q, (u, c'', e_0, j'))
                else
                    need[e_0][l] = j'_l
end
```

Jiménez, Víctor M. and Andrés Marzal. 2000. Computation of the n best parse trees for weighted and stochastic context-free grammars. In *Proc. of the Joint IAPR International Workshops on Advances in Pattern Recognition*.

Klein, Dan and Chris Manning. 2003. A* parsing: Fast exact viterbi parse selection. In *Proc. HLT-NAACL*.

Knight, Kevin and Jonathan Graehl. 2005. An overview of probabilistic tree transducers for natural language processing. In *Proc. of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, LNCS.

Knuth, Donald. 1977. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1).

Mohri, Mehryar and Michael Riley. 2002. An efficient algorithm for the n-best-strings problem. In *Proceedings of the International Conference on Spoken Language Processing 2002 (ICSLP '02)*, Denver, Colorado, September.