Deplump: A streaming lossless compressor

Nicholas Bartlett Frank Wood

Department of Statistics, Columbia University, New York, USA

Abstract

1 Algorithm

Probabilistic compression algorithms work by using a generative model to predict the sequence. The predictive distribution function is then used as the parameter in a range encoder to compress the stream. The details of a range encoder implementation are not included here, we only note that if the predictive distribution function is F and the next symbol in the stream is s then the parameters required by the range encoder are $F(s-1)$ and $F(s)$. In order to decompress the stream the exact same predictive model will need to be built. This requires that the model estimate prior to compressing s_n is a function of fixed parameters and the symbols $[s_0, s_1, \ldots, s_{n-1}]$ because those are the only symbols available to the decompressor for decompressing s_n.

The algorithm operates primarily on a suffix tree. A suffix tree is a data structure for keeping track of unique suffices of a set of strings. The tree structure arranges the suffices hierarchically which makes it easy to search. In the case of single stream the set of strings to consider is the set of contexts $\{[], [s_0], [s_0, s_1], [s_0, s_1, s_2], \ldots\}$. The suffix tree is comprised of nodes, each corresponding to a context of the form $[s_m, \ldots, s_{m+k}]$. Nodes other than the root have exactly one parent, but could have many children.

The suffix tree must be incrementally constructed as elements of the sequence are processed. Construction of the tree is handled by the function GetNode in Algorithm 1. An illustration of the incremental suffix tree construction can be seen in Figure 1 for the toy sequence [PATAT]. Note that in frame 5 the node [A] had to be inserted in order to incorporate the node [PATA]. Using the notation of GetNode([PATA], $\mathcal{T}$) to describe the frame we find $\mathcal{N}$ = [PATA], $\mathcal{M}$ = [PA], and $\mathcal{P}$ = [A].

Algorithm 1 Deplump

```
1: procedure DEPLUMP(s = $[s_0, s_1, s_2, \dots, s_m]$, depth)
2:     Set $nc$ = 0 (node count), $\mathcal{RS}$ = [ ] (reference sequence)
3:     Initialize $\mathcal{T}$ (tree) and update $nc$
4:     Set $\mathcal{D} = \{d_0, d_1, d_2, \dots, d_{\min(10,depth)}, \alpha\}$ (discount parameters)
5:     for i = 0 : m do
6:         $[F(s-1), F(s)]$ = CDFNextSymbol($s_i$,$\mathcal{T}$,$\mathcal{RS}$)
7:         Update discount parameters based on gradients and learning rate
8:         Use range encoder to encode the symbol $s_i$ using values of $F(s-1)$ and $F(s)$
9:         Add $s_i$ to $\mathcal{RS}$
10:    end for
11: end procedure
12: function CDFNEXTSYMBOL($s$,$\mathcal{T}$,$\mathcal{RS}$)
13:    while $\mathcal{RS}$ is longer than allowable do
14:        Shorten reference sequence
15:    end while
16:    while $nc >$ (max allowable nodes - 2) do
17:        Delete leaf node uniformly at random
18:    end while
19:    $\mathcal{N}$= GetNode($\mathcal{RS}$, $\mathcal{T}$)
20:    $[F(s-1), F(s)]$ = CDF($s$, $\mathcal{N}$, cdf = zero array, $d$ = 1)
21:    UpdateCountsAndDiscounts($\mathcal{N}$,$s$, TRUE)
22:    return $[F(s-1), F(s)]$
23: end function
24: function GETNODE(Context, $\mathcal{T}$)
25:    $\mathcal{N}$ is a node/context corresponding to $\mathcal{RS}$
26:    Find the node $\mathcal{M}$ in suffix tree sharing the longest suffix with $\mathcal{N}$
27:    if $\mathcal{M}$ is a suffix of $\mathcal{RS}$ then
28:        if $\mathcal{N}$ is not equal to $\mathcal{M}$ then
29:            Add node $\mathcal{N}$ as a child of $\mathcal{M}$
30:            return $\mathcal{N}$
31:        else
32:            return $\mathcal{M}$
33:        end if
34:    else
35:        $\mathcal{P}$ = GetNode(shared suffix of $\mathcal{M}$ and $\mathcal{N}$, $\mathcal{T}$)
36:        Add node $\mathcal{N}$as a child of $\mathcal{P}$ for the current context
37:        return $\mathcal{N}$
38:    end if
39: end function
```

Algorithm 2 Deplump Continued

```
function UpdateCountsAndDiscounts($\mathcal{N}$, $s$, AddCount)
    if AddCount then
        $c_s^{\mathcal{N}} += 1$
        AddCount = 1 with probability $\frac{t^{\mathcal{N}} d^{\mathcal{N}}}{c_s^{\mathcal{N}} + (t^{\mathcal{N}} - t_s^{\mathcal{N}})d}$ else 0
        $t_s^N += (\text{AddCount} == 1)$
    end if
    Update discount parameter gradients
    UpdateCountsAndDiscounts(parent of $\mathcal{N}$, $s$, AddCount)
end function
function CDF($s$, $\mathcal{N}$, cdf, $m$)
    for i = 1 : size of symbol set do
        cdf[i] += $m(\frac{c_i^{\mathcal{N}} - t_i^{\mathcal{N}} d^{\mathcal{N}}}{c^{\mathcal{N}}})$
    end for
    if $\mathcal{N}$ has parent then
        return CDF($s$, parent of $\mathcal{N}$, cdf, $d^{\mathcal{N}} m$)
    else
        cdf = (1- $d^{\mathcal{N}} m$)cdf + $d^{\mathcal{N}} m$(uniform distribution over symbol set)
        $F(s-1)$ = sum(cdf(1:($s$-1)))
        return [$F(s-1), F(s) = F(s-1)$ + cdf($s$)]
    end if
end function
```

Algorithm 3 Creating the Tree

```
function FractureNode($\mathcal{N}, d, c$)
    Initialize $\mathcal{M}$(a new node)
    for each $s$ observed in node $\mathcal{N}$ do
        partition = GetPartition($c_s^{\mathcal{N}}$, $t_s^{\mathcal{N}}$,-c)
        Set $c_s^{\mathcal{M}} = 0$, $t_s^{\mathcal{M}} = t_s$, and $t_s = 0$
        for $i$ = 1 : length(partition) do
            Set t = DrawCRP(partition[i],d,c)
            Set $t_s^{\mathcal{N}}, c_s^{\mathcal{M}} += t$
        end for
    end for
end function
function GetPartition($c, t, d$)
    Set $M = d\times\ c$ matrix of zeros
    Set $M(d, c) = 1.0$
    for $j = (c - 1) : 1$ do
        for $i = 1 : (t - 1)$ do
            Set $M(i, j) = M(i + 1, j + 1)(id) + M(i + 1, j)(j - id)$
        end for
        Set $M(d, j) = M(t, j + 1)$
    end for
    Set partition = $[p_1, p_2, \ldots, p_t]$ with $p_i = 0$ for $i > 1$ and $p_1 = 1$
    Set $k = 1$
    for j = 2 : c do
        Set $M(k, j) = M(k, j)(j - 1 - kd)$
        Set $M(k + 1, j) = M(k + 1, j)kd$
        Set $r = 1$ with probability $\frac{M(k+1,j)}{M(k+1,j)+M(k,j)}$ else 0
        if r == 1 then
            $k += 1$
            partition[$k$] = 1
        else
            partition$[m] += 1$ with probability $\frac{\text{partition}[m]-d}{j-1-kd}$ for $1 \leq m \leq k$
        end if
    end for
    return partition
end function
function DrawCRP($n, d, c$)
    Set $t = 1$
    for i = 2 : n do
        Set $r = 1$ with probability $\frac{td+c}{i-1+c}$ else 0
        Set $t += (r == 1)$
    end for
    return $t$
end function
```

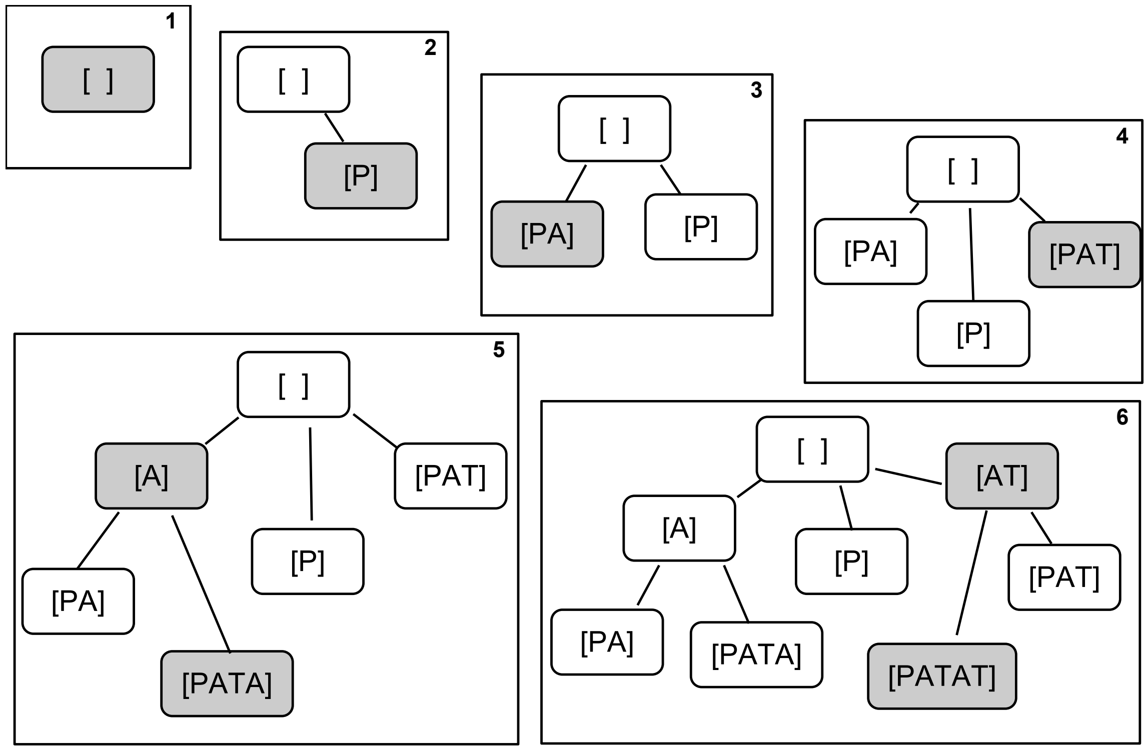

Figure 1: Construction of suffix tree for string "PATAT". In each frame the new nodes are shaded in gray.