

Deplump: A streaming lossless compressor

Nicholas Bartlett

Frank Wood

Department of Statistics, Columbia University, New York, USA

Abstract

1 Introduction

2 Previous Work

A lossless compressor based on a probabilistic model called the Sequence Memoizer (Wood et al. [2009]) was proposed in Gasthaus et al. [2010]. Empirical performance was evaluated on the calgary corpus and an algorithm was outlined. Unfortunately, In Gasthaus et al. [2010] the algorithm proposed has a memory footprint which grows linearly in the length of the input sequence, making it unsuitable for the compression of streams. A constant space algorithm for inference in the Sequence Memoizer model has since been developed by Bartlett et al. [2010]. This implementation of Deplump builds on the inference algorithms proposed in Bartlett et al. [2010] and Gasthaus et al. [2010] to develop a compressor which has finite memory complexity.

3 Algorithm

Given an ordered symbol set Σ , probabilistic compression algorithms work by using a probabilistic model to predict a sequence of symbols. 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 a range encoder takes $F(s - 1)$ and $F(s)$ as arguments and returns a bit stream (possibly null). To decompress, the range decoder takes F and the compressed stream as arguments and returns the next symbol in the uncompressed sequence. In the algorithm the functions `RangeEncode()` and `RangeDecode()` indicate these operations. The use of a cumulative distribution function is well defined since the symbols are ordered. The notation $s - 1$ refers to the symbol prior to s in the symbol ordering. In order to decompress the stream the exact same predictive model will need to be built from the compressed stream. This requires that the model estimate prior to compressing s_n is a function of fixed parameters and the symbols $[s_0, s_1, \dots, s_{n-1}]$ because those are the only symbols available to the decompressor for decompressing s_n .

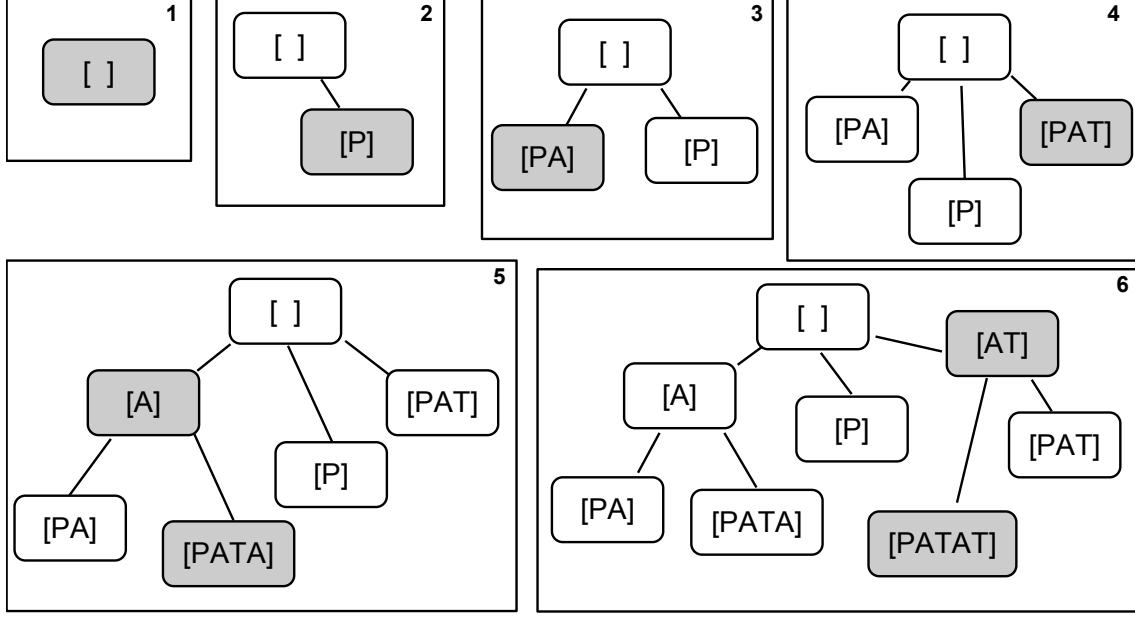


Figure 1: Construction of suffix tree for string “PATAT”. In each frame the new nodes are shaded in gray.

The algorithm operates on a suffix tree. A suffix tree is a data structure for keeping track of the unique suffixes in a set of strings. The tree structure arranges the suffixes hierarchically which makes it easy to search. In the case of a single stream the set of strings to consider is the set $\{[], [s_0], [s_0, s_1], [s_0, s_1, s_2], \dots\}$, which we refer to as the set of contexts. Each node of the suffix tree corresponds to a string of the form $[s_m, \dots, s_{m+k}]$. In general we use \mathcal{N} to refer interchangeably to a node instance and the context to which the node corresponds. The function $\text{CreateNode}(\mathcal{N}, \mathcal{M})$ makes the creation of node \mathcal{N} with parent \mathcal{M} explicit. The parent of node \mathcal{N} is referenced as $\text{PA}(\mathcal{N})$.

Each node instance \mathcal{N} contains two counts for each $s \in \Sigma$, c_s and t_s . We use c and t to refer to the marginal counts $\sum_{s \in \Sigma} c_s$ and $\sum_{s \in \Sigma} t_s$. Each node also has a discount d associated with it. The discount associated with \mathcal{N} is a function of \mathcal{D} (the discount parameters of the model), $|\mathcal{N}|$, and $|\text{PA}(\mathcal{N})|$. The discount for \mathcal{N} is calculated by $\text{GetDiscount}(\mathcal{N})$. Suffix tree data structures use a reference sequence (\mathcal{RS}) to store the unique suffixes in the tree. Therefore, each node instance also contains two indices related to \mathcal{RS} from which the context specific to that node can be reconstructed. If the indices for \mathcal{N} are i and j , then the context associated with \mathcal{N} is $\mathcal{RS}[i : j]$.

The reference sequence grows with the length of the input sequence and must be shortened as the algorithm progresses. The shortening of \mathcal{RS} is made explicit by the σ function in CDFNextSymbol . The function $\sigma(\mathcal{S})$ returns $\mathcal{S}[2 : \text{end}]$. When \mathcal{RS} is shortened, nodes in the suffix tree which reference removed sections are no longer usable and must be removed from the tree to prevent a memory leak. To facilitate the removal process pointers are maintained from the elements of \mathcal{RS} to the suffix tree nodes which reference them. Without the use of pointers, deletion of the unusable nodes requires a search over the tree

which is prohibitive for large trees. The cost of these operations can be amortized by shorting \mathcal{RS} in chunks and keeping pointers from each chunk of \mathcal{RS} instead of each element. To minimize the impact of rendering nodes unusable by shortening the reference sequence, suffix tree nodes are updated as the algorithm progresses to reference recent sections of \mathcal{RS} .

Several parameters required by the model must be assigned at initialization. They are \mathcal{D} , k , L , η , and depth. $\mathcal{D} = [\delta_0, \delta_1, \dots, \delta_{10}, \alpha]$ is a list of discount parameters, each taking a real value in $(0, 1)$. The parameter k is positive integer valued and is an upper bound on the total count c in each node. The parameter L is an upper bound on the number of node instances in the suffix tree. We use $100L$ as an upper bound on the length of \mathcal{RS} , but there is no explicit reason to tie these two upper bounds together. The parameter η is a learning rate for the updating of the discount parameters and is typically set very small. Finally, the depth of the tree can be limited by a positive integer. Depth is not considered in Algorithm 1, but it could be incorporated into the function `GetNode`.

Since the model must be estimated incrementally, the suffix tree must also be incrementally constructed. Construction of the tree is handled by the function `GetNode` and `FragmentNode` in Algorithm 1. An illustration of the incremental construction can be seen in Figure 1 for the toy sequence [PATAT]. In frame 4 the function `GetNode` assigns $[\]$ to \mathcal{M} and then [PAT] to \mathcal{S} with $\mathcal{M} = \text{PA}(\mathcal{S})$. In Frame 5 `GetNode` assigns [PA] to \mathcal{M} , but then must assign $[A] = \text{FragmentNode}(\mathcal{M})$ to \mathcal{P} and $\text{PA}(\mathcal{M})$ to \mathcal{P} . Node \mathcal{S} is then created by `CreateNode([PATA], \mathcal{P})`. In each frame the first step is to find \mathcal{M} , which can be achieved by descending an appropriate path of the suffix tree. All of the nodes on the path to \mathcal{M} and possibly \mathcal{M} itself can have the indices into \mathcal{RS} updated to point to a more recent section of the reference sequence.

For each s in the input sequence the function `PMFNextSymbol` is called to obtain the predictive probability mass function (PMF). The function `PMFNextSymbol` enforces the upper bounds on $|\mathcal{RS}|$ and nc . If $|\mathcal{RS}|$ is hitting the upper bound then \mathcal{RS} is shortened as described, if nc is hitting the upper bound then leaf nodes of the suffix tree are deleted uniformly at random. Note that the deleting of leaf nodes uniformly at random is non-trivial to implement. One approach is to maintain an active list must of all the leaf nodes throughout the algorithm. A second approach is for each node to maintain a count of thenumber of leaf nodes in the subtree beneath it. Given that each node contains the exact number of leaf nodes beneath it a random leaf node can be obtained by taking a weighted random path down the tree. Finally, `PMFNexSymbol` returns the predictive PMF.

After encoding or decoding using the predictive PMF p_i the first step in updating the model estimate is performed by the function `UpdateCountsAndDiscounts`. Starting at node \mathcal{N} and progressing up to the root of the tree, c_s is incremented if t_s was incremented in the node below. If c_s is incremented, a stochastic decision is made to increment t_s . The gradients for the discount parameters \mathcal{D} are updated by the function `UpdateDiscountParameterGradients`. If c is larger than k in any of the nodes, the counts c_s and potentially t_s are reduced by the function `ThinCounts`. Finally, \mathcal{D} is updated based on the calculated gradients and the specified learning rate η and the symbol is appended to \mathcal{RS} .

Algorithm 1 Deplump

```
1: procedure DEPLUMP/PLUMP( $\mathcal{IS}$ )
2:    $\mathcal{RS} \leftarrow []$  ▷ reference sequence
3:   Initialize  $[]$  node of  $\mathcal{T}$  ▷ suffix tree
4:    $nc \leftarrow 1$  ▷ node count
5:    $\mathcal{D} \leftarrow \{\delta_0, \delta_1, \delta_2, \dots, \delta_{10}, \alpha\}$  ▷ discount parameters
6:    $\mathcal{G} \leftarrow \vec{0}$  ▷ discount parameter gradients,  $|\mathcal{G}| = |\mathcal{D}|$ 
7:    $\mathcal{OS} \leftarrow []$  ▷ output sequence
8:   for  $i = 1: |\mathcal{IS}|$  do
9:      $[\pi, \mathcal{N}] \leftarrow \text{PMFNextSymbol}(\mathcal{RS})$ 
10:    if Plump then
11:       $s \leftarrow \text{RangeDecode}(\pi, \mathcal{IS})$ 
12:       $\mathcal{OS} \leftarrow [\mathcal{OS} \ s]$ 
13:    else
14:       $s \leftarrow \mathcal{IS}[i]$ 
15:       $b \leftarrow \text{RangeEncode}(\sum_{i=1}^{s-1} \pi_i, \sum_{i=1}^s \pi_i)$ 
16:       $\mathcal{OS} \leftarrow [\mathcal{OS} \ b]$ 
17:    end if
18:     $\text{UpdateCountsAndDiscountGradients}(\mathcal{N}, s, \pi_s, \text{TRUE})$ 
19:     $\mathcal{D} \leftarrow \mathcal{D} + \mathcal{G}\eta/(\pi_s)$  ▷ update discount parameters
20:     $\mathcal{G} \leftarrow \vec{0}$  ▷ reset gradients to zero
21:     $\mathcal{RS} \leftarrow [\mathcal{RS} \ s]$  ▷ append symbol to reference sequence
22:  end for
23:  return  $\mathcal{OS}$ 
24: end procedure
25: function PMFNEXTSYMBOL( $\mathcal{RS}$ )
26:  while  $|\mathcal{RS}| \geq 100L$  do
27:    Delete nodes referencing  $\mathcal{RS}[1]$  and update  $nc$ 
28:     $\mathcal{RS} \leftarrow \sigma(\mathcal{RS})$ 
29:  end while
30:  while  $nc > (L - 2)$  do
31:    Delete leaf node uniformly at random
32:     $nc \leftarrow nc - 1$ 
33:  end while
34:   $\mathcal{N} \leftarrow \text{GetNode}(\mathcal{RS}, \mathcal{T})$ 
35:   $\pi \leftarrow \text{PMF}(\mathcal{N}, \vec{0}, 1.0)$  ▷  $|\vec{0}| = |\Sigma|$ 
36:  return  $[\pi, \mathcal{N}]$ 
37: end function
```

Algorithm 2 Deplump Continued

```
1: function GETDISCOUNT( $\mathcal{N}$ )
2:    $d = 1.0$ 
3:   if  $\mathcal{N} = [ ]$  then
4:     return  $\delta_0$ 
5:   end if
6:   for  $i = (|\text{PA}(\mathcal{N})| + 1) : |\mathcal{N}|$  do
7:     if  $i \leq 10$  then
8:        $d \leftarrow d\delta_i$   $\triangleright$  multiply by discount parameter  $i$ 
9:     else
10:       $d \leftarrow d\delta_{10}^{\alpha^i}$ 
11:    end if
12:  end for
13:  return  $d$ 
14: end function
15: function GETNODE( $\mathcal{S}, T$ )
16:   Find the node  $\mathcal{M}$  in the suffix tree sharing the longest suffix with  $\mathcal{S}$ .
17:   if  $\mathcal{M}$  is a suffix of  $\mathcal{S}$  then
18:     if  $\mathcal{S} = \mathcal{M}$  then
19:       return  $\mathcal{M}$ 
20:     else
21:        $\mathcal{S} \leftarrow \text{CreateNode}(\mathcal{S}, \mathcal{M})$ 
22:        $nc \leftarrow nc + 1$ 
23:       return  $\mathcal{S}$ 
24:     end if
25:   else
26:      $\mathcal{P} \leftarrow \text{FragmentNode}(\mathcal{M}, \mathcal{S})$ 
27:      $\mathcal{S} \leftarrow \text{CreateNode}(\mathcal{S}, \mathcal{P})$ 
28:      $nc \leftarrow nc + 1$ 
29:     return  $\mathcal{S}$ 
30:   end if
31: end function
```

Algorithm 3 Deplump Continued

```

1: function UPDATECOUNTSANDDISCOUNTS( $\mathcal{N}$ ,  $s$ ,  $p$ , BackOff)
2:    $d \leftarrow \text{GetDiscount}(\mathcal{N})$ 
3:    $pp \leftarrow p$ 
4:   if  $c > 0$  then
5:      $pp \leftarrow (p - \frac{c_s - t_s d}{c})(\frac{c}{td})$ 
6:      $w \leftarrow c_s + d(t * pp - t_s)$ 
7:   end if
8:   if BackOff and  $c > 0$  then
9:      $c_s \leftarrow c_s + 1$ 
10:    BackOff  $\leftarrow 0$ 
11:    BackOff  $\leftarrow 1$  w.p.  $pp(\frac{td}{w})$   $\triangleright$  w.p abbreviates “with probability”
12:    if BackOff then
13:       $t_s \leftarrow t_s + 1$ 
14:    end if
15:  else if BackOff then
16:     $c_s \leftarrow c_s + 1$ 
17:     $t_s \leftarrow t_s + 1$ 
18:  end if
19:  UpdateDiscountParameterGradients( $t_s$ ,  $t$ ,  $pp$ ,  $d$ )
20:  UpdateCountsAndDiscounts(PA( $\mathcal{N}$ ),  $s$ ,  $pp$ , BackOff)
21:  ThinCounts( $\mathcal{N}$ )
22: end function
23: function THINCOUNTS( $\mathcal{N}$ )
24:    $d \leftarrow \text{GetDiscount}(\mathcal{N})$ 
25:   while  $c > k$  do
26:      $s \leftarrow \text{DrawMultinomial}(\pi)$  s.t.  $\pi_l = \frac{c_l}{c}$   $\triangleright \pi$  is a distribution over  $\Sigma$ 
27:      $\phi \leftarrow \text{SamplePartition}(c_s, t_s, d)$ 
28:      $i \leftarrow \text{DrawMultinomial}([\frac{\phi_1}{c_s}, \frac{\phi_2}{c_s}, \dots, \frac{\phi_{t_s}}{c_s}])$ 
29:     if  $\phi_i == 1$  then
30:        $t_s \leftarrow t_s - 1$ 
31:     end if
32:      $c_s \leftarrow c_s - 1$ 
33:   end while
34: end function

```

Algorithm 4 Deplump Continued

```
1: function PMF( $\mathcal{N}, \pi, m$ )
2:    $d \leftarrow \text{GetDiscount}(\mathcal{N})$ 
3:   if  $c > 0$  then
4:     for  $s \in \Sigma$  do
5:        $\pi_s \leftarrow \pi_s + m(\frac{c_s - t_s d}{c})$ 
6:     end for
7:   end if
8:   if  $\text{PA}(\mathcal{N}) \neq \text{null}$  then
9:     return  $\text{PMF}(\text{PA}(\mathcal{N}), \pi, dm)$ 
10:  else
11:     $\pi \leftarrow (1 - dm)\pi + dm\mathcal{U}(\Sigma)$   $\triangleright \mathcal{U}(\Sigma)$  is the uniform distribution over  $\Sigma$ 
12:    return  $\pi$ 
13:  end if
14: end function
15: function FRAGMENTNODE( $\mathcal{M}, \mathcal{S}$ )
16:    $d^{\mathcal{M}} \leftarrow \text{GetDiscount}(\mathcal{M})$ 
17:    $\mathcal{P} \leftarrow$  maximum overlapping suffix of  $\mathcal{M}$  and  $\mathcal{S}$ 
18:    $\mathcal{P} \leftarrow \text{CreateNode}(\mathcal{P}, \text{PA}(\mathcal{M}))$ 
19:    $nc \leftarrow nc + 1$ 
20:    $\text{PA}(\mathcal{M}) \leftarrow \mathcal{P}$ 
21:    $d^{\mathcal{P}} \leftarrow \text{GetDiscount}(\mathcal{P})$ 
22:   for  $s \in \Sigma$  do
23:      $\phi \leftarrow \text{SamplePartition}(c_s^{\mathcal{M}}, t_s^{\mathcal{M}}, d^{\mathcal{M}})$ 
24:      $t_s^{\mathcal{P}} \leftarrow t_s^{\mathcal{M}}$ 
25:      $t_s^{\mathcal{M}} \leftarrow 0$ 
26:     for  $i = 1 : |\phi|$  do
27:        $a \leftarrow \text{DrawCRP}(\phi[i], d^{\mathcal{M}}/d^{\mathcal{P}}, -d^{\mathcal{M}})$ 
28:        $t_s^{\mathcal{M}} \leftarrow t_s^{\mathcal{M}} + a$ 
29:     end for
30:      $c_s^{\mathcal{P}} \leftarrow t_s^{\mathcal{M}}$ 
31:   end for
32:   return  $\mathcal{P}$ 
33: end function
34: function DRAWCRP( $n, d, c$ )  $\triangleright n \geq 1$ 
35:    $t \leftarrow 1$ 
36:   for  $i = 2 : n$  do
37:      $r \leftarrow 0$ 
38:      $r \leftarrow 1$  w.p.  $\frac{td+c}{i-1+c}$ 
39:      $t \leftarrow t + r$ 
40:   end for
41:   return  $t$ 
42: end function
```

Algorithm 5 Deplump Continued

```
1: function SAMPLEPARTITION( $c, t, d$ )
2:    $M \leftarrow t \times c$  matrix of zeros
3:    $M(t, c) \leftarrow 1.0$ 
4:   for  $j = (c - 1) : 1$  do
5:     for  $i = 1 : (t - 1)$  do
6:        $M(i, j) \leftarrow M(i + 1, j + 1) + M(i, j + 1)(j - id)$ 
7:     end for
8:      $M(d, j) \leftarrow M(t, j + 1)$ 
9:   end for
10:   $\phi \leftarrow \vec{0}$   $\triangleright |\vec{0}| = t$ 
11:   $\phi[1] \leftarrow 1$ 
12:   $k \leftarrow 1$ 
13:  for  $j = 2 : c$  do
14:     $M(k, j) \leftarrow M(k, j)(j - 1 - kd)$ 
15:     $r \leftarrow 0$ 
16:     $r \leftarrow 1$  w.p.  $\frac{M(k+1, j)}{M(k+1, j) + M(k, j)}$ 
17:    if  $r = 1$  then
18:       $k \leftarrow k + 1$ 
19:       $\phi[k] \leftarrow 1$ 
20:    else
21:       $i \leftarrow \text{DrawMultinomial}([\frac{\phi[1]-d}{j-1-kd}, \frac{\phi[2]-d}{j-1-kd}, \dots, \frac{\phi[k]-d}{j-1-kd}])$ 
22:       $\phi[i] \leftarrow \phi[i] + 1$ 
23:    end if
24:  end for
25:  return  $\phi$ 
26: end function
27: function UPDATEDISCOUNTPARAMETERGRADIENTS( $\mathcal{N}, t_s, c, t, pp, d, m$ )
28:  if  $c > 0$  then
29:    if  $|\mathcal{N}| = 0$  then
30:       $\psi \leftarrow \frac{1.0}{\delta_0}$ 
31:       $\mathcal{G}_0 \leftarrow \mathcal{G}_0 + (d(t * pp - t_s)\psi/c)m$ 
32:    else
33:       $z \leftarrow |\text{PA}(\mathcal{N})| + 1$ 
34:      while  $z \leq |\mathcal{N}|$  and  $z < 10$  do
35:         $\psi \leftarrow \frac{1.0}{\delta_z}$ 
36:         $\mathcal{G}_z \leftarrow \mathcal{G}_z + (d(t * pp - t_s)\psi/c)m$ 
37:      end while
38:      if  $|\mathcal{N}| \geq 10$  then
39:         $a \leftarrow z - 10$ 
40:         $b \leftarrow |\mathcal{N}| - z + 1$ 
41:         $\psi \leftarrow \alpha^a(1 - \alpha^b)/((1 - \alpha)\delta_{10})$ 
42:         $\mathcal{G}_{10} \leftarrow \mathcal{G}_{10} + (d(t * pp - t_s)\psi/c)m$ 
43:         $\psi \leftarrow \log(\delta_{10})(a\alpha^{a-1} - (a + b)\alpha^{a+b-1})/(1 - \alpha) + (\alpha^a - \alpha^{a+b})/(1 - \alpha)^2$ 
44:         $\mathcal{G}_{11} \leftarrow \mathcal{G}_{11} + (d(t * pp - t_s)\psi/c)m$ 
45:      end if
46:    end if
47:  end if
48: end function
```

4 Experiments

The empirical performance of Deplump was evaluated using the most recent Wikipedia content dump as a test corpus.

to evaluate the effect of various parameter settings on the compression performance of Deplump. The empirical per

5 Conclusion

References

- Bartlett, N., Pfau, D., and Wood, F. (2010). Forgetting counts : Constant memory inference for a dependent hierarchical Pitman-Yor process. In (to appear) ICML.
- Gasthaus, J., Wood, F., and Teh, Y. W. (2010). Lossless compression based on the Sequence Memoizer. In *Data Compression Conference 2010*, pages 337–345.
- Wood, F., Archambeau, C., Gasthaus, J., James, L., and Teh, Y. W. (2009). A stochastic memoizer for sequence data. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1129–1136, Montreal, Canada.