A CNF Transformation for Multidimensional Grammars

David Brown, Ian Kelly, Colin Kern, Alex Lemann, Greg Sandstrom
Earlham College
Department of Computer Science
Richmond, Indiana
{brownda,kellyia,kernco,lemanal,sandsgr}@cs.earlham.edu

September 17, 2004

Abstract

This paper explores a transformation for factoring arbitrary branching multidimensional local trees into sets of strictly 2-branching local trees, resembling the conversion to Chomsky Normal Form for 2-dimensional trees. We outline a converse process to extract the original arbitrary branching local tree from sets of local tree factors. Once we complete the converse process, we expect to simultaneously prove the correctness of both. The paper also explores the rate of growth in multidimensional trees with constant branching factor and increasing dimensionality, and relate it to the growth in the size of the grammar as a result of the factorization.

1 Introduction

Our group is exploring the parsing of grammars represented as higher-dimensional trees[1] using a CKY-style algorithm [HMU01]. These grammars are sets of local multidimensional trees corresponding with the rewriting rules of a Context Free Grammar, which can be though of as 2-dimensional trees. Because the complexity of CKY is dependent on the branching factor of the local trees, we need an algorithm to reduce the branching factor to 2-branching in arbitrary dimensionality, akin to the conversion to Chomsky Normal Form (CNF) in the 2-dimensional case [HMU01]. As is the case with the conversion to CNF in two dimensions, the reduction from arbitrary branching to 2-branching in higher dimensions does not change the generative power of the grammar.

[1]For an introduction to these grammars and multidimensional trees see our paper Representing Multidimensional Trees also submitted to this conference [BKK$^+$04] and [Rog03].

2 d-dimensional Trees

We use a left-child/right-sibling form to represent 2-dimensional trees [BKK+04]. Instead of the traditional idea of a parent with a left child and a right child (in a binary branching tree), we consider the left child to be the 2-dimensional successor of the parent, and the right child is the 1-dimensional successor of the parent's 2-dimensional successor. This allows arbitrary branching trees while keeping the number of successors per node constant. Henceforth, we will use the term successor instead of child to make it distinct from the classic notion of a child. We will use these two terms consistently throughout the paper, with child referring to any node related in the dimension, and successor referring to only the minimum child in that dimension.

We then extend this idea to more than two dimensions, where every node can have up to d successors, one per dimension, where d is the dimensionality of the tree. Figure 1 shows a 3-dimensional tree. The successors of each node are indicated with bold lines, while all other lines indicate children.

3 Size of Trees

Since our transformation will factor b-branching trees into a sets of 2-branching trees in a process similar to the CNF transformation, we can expect the size of the grammar to grow in a similar fashion. Since the set of 2-branching trees must preserve every node in the original b-branching tree, we can expect the optimal growth in the size of our grammars to be related to the growth in the size of b-branching multidimensional trees as a function of the dimension.

3.1 Size of 2-branching Trees

Lemma 1 *Every n-dimensional successor (in the left-child, right-sibling sense) in a d-dimensional, 2-branching tree can have only an $(n-1)$-dimensional successor and a d-dimensional successor.*

Proof: An n-dimensional successor in a d-dimensional, 2-branching tree cannot have any successor in dimensions less than $n-1$, because for any root X of an i-dimensional local yield, X has no j-dimensional successor for $j < i$,[2] and it can have no successor in any dimension greater than or equal to n, except for dimension d, because the node already has a parent in each such dimension, and so the inclusion of such a successor would make the branching factor in the dimension of $n-1$ more than 2. $\dashv$

Corollary 1 *In a d-dimensional 2-branching local tree there are exactly $d+1$ nodes.*

Since no element in the yield of the local tree will have a successor in the d-dimension,

[2]This is shown in Representing Multidimensional Trees. Since X is a root of an i-dimensional local yield, having a successor in any dimension less than i would invalidate its role as the unique minimum of an i-dimensional local yield.

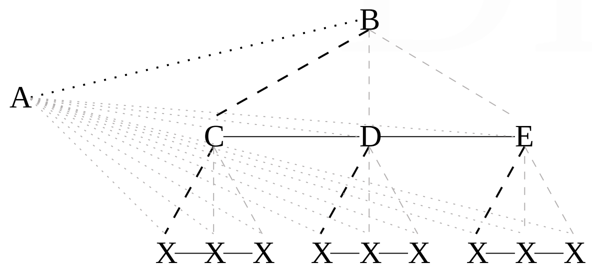

Figure 1: A full 3-dimensional, 3-branching tree.

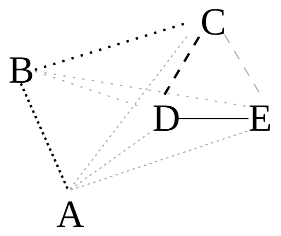

Figure 2: A 2-branching 4-dimensional local structure.

and the root will only have a successor in the d-dimension, each node in the local tree only has one successor.

Looking at a d-dimensional 2-branching structure, Corollary 1 requires there to be $d + 1$ nodes in the structure. Thus the size of the 2-branching local tree is linear in the dimension d.[3]

[3]Another perspective on the growth of 2-branching trees is to look at them as simplexes. When we increase the dimensionality of a local 2-branching tree, we are simply adding a root in an orthogonal dimension to an existing 2-branching local tree.

3.2 Size of b-branching Trees

Now we need to extend this to the b-branching case. Since we are only interested in the lower bound, looking at the 3-branching case will suffice. As seen in Figure 1, the number of nodes in a 3-dimensional, 3-branching tree (14 nodes) is large compared to the 2-dimensional 3-branching tree (4 nodes). Since the number of nodes we have to add to increase the dimensionality is dependent on the number of nodes in the tree, we expect the growth to be faster than linear.

Let $N_b(d)$ be the number of nodes in a d-dimensional tree of branching factor b. Looking again at Figure 1, yield of the 3-dimensional tree (i.e. the tree rooted at B) can be seen as a full 3-branching 2-dimensional local tree with another full 3-branching 2-dimensional local tree attached at every node (C,D, and E), except the root (B). So we are multiplying the number of nodes in a $(d-1)$-dimensional structure, $N_3(d-1)$, by the number of nodes in the original $(d-1)$-dimensional yield minus the root (B), $N_3(d-1)-1$. Then we have to add the root (B) back in, plus a new root (A) to anchor the tree in the d^{th}-dimension. The number of nodes in a full 3-branching d-dimensional local tree is given by the recurrence

$$\begin{aligned} N_3(0) &= 1 \\ N_3(d) &= N_3(d-1)^2 - N_3(d-1) + 2 \end{aligned}$$

We are interested in the lower bound on the size of these trees, so when we solve this recurrence we only need to look at the largest term. The first step in the recursion is

$$\begin{aligned}&N_3(d) = \\ &\quad (N_3(d-2)^2 - N_3(d-2) + 2)^2 - \\ &\quad (N_3(d-2)^2 - N_3(d-2) + 2) + 2\end{aligned}$$

The largest term is $(N_3(d-2)^2)^2$. If we carry the recursion out to the next step, this term grows to $(N_3((d-3)^2)^2)^2$. In general, the entire term is squared for every step in the recursion, of which there are d. So we can say that the number of nodes in a 3-branching n-dimensional tree is $\Omega(k^{(2^{(d-1)})})$ for some k.[4]

Moving from 2-branching to higher-branching, then, means an abrupt shift from size linear in the dimension to hyper-exponential in the dimension.

4 Embed Algorithm

Similar to the CNF transformation, our transformation will factor single b-branching local trees to sets of 2-branching local trees. We call this process "embed" because it induces an embedding of b-branching trees into 2-branching trees.

To factor an arbitrary branching d-dimensional local tree into a set of 2-branching d-dimensional local trees, we can simply traverse the local tree checking for illegal links. Since the factoring depends on the order in which we handle the links, we will standardize the process by always looking for illegal links in the higher dimensions first. When such links are found, we replace the parent node of the link with a new, unique

[4]This can be verified by an easy induction on d.

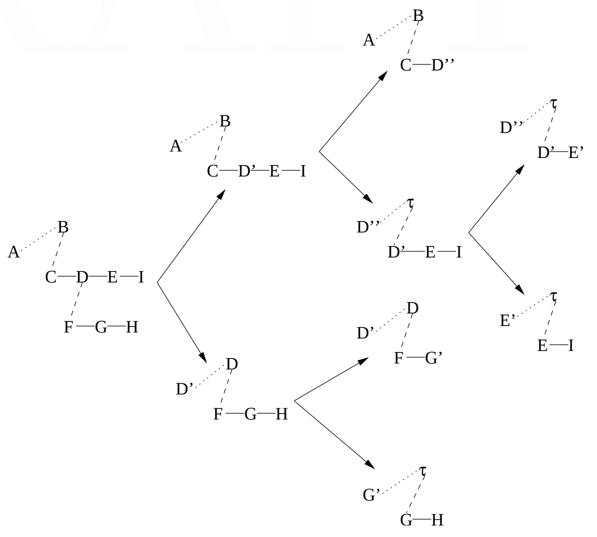

Figure 3: Example factoring of a 3d tree.

label,[5] and we introduce a new rule rooted in that new label. The tree for the new rule consists of the original parent node of the link along with its i-dimensional yield, with i being the dimension of the illegal link. Figure 3 shows the progression from left to right of an arbitrary branching 3-dimensional structure being factored into 2-branching structures. In the first factoring, the node D is found to have an illegal link in the second dimension (it also has one in the first dimension, but this will be handled later). The resulting two structures are shown.

If the dimension of an illegal link is less

[5]As a convention, if the original label was X, we will make the new label X'. This is simply a shorthand. It is important that X' be unique in the grammar as a whole to prevent the introduction of ambiguity. If this node X' already exists, we will simply append another prime marker until a unique label is formed.

than $n-1$, then simply adding the new label as the root will create a lower dimensional structure. To promote the structure to the proper dimension, nodes are added with a special value which we will call τ. This can be seen in the structures rooted at G', E', and D'' in Figure 3. Since these nodes are added by the factoring process, there is no circumstance in which a rule will be rooted at τ. The only rules generated by the factoring process are rules that already existed, or rules rooted at one of the unique labels created. Since τ has this restriction, we can give all such nodes the same label, without having to distinguish one from the other. A τ node is not part of the image of tree but is intended to help us "excavate" or reconstruct the arbitrary-branching structure after parsing is complete.

Because τ is never the root and occurs only in 2-branching rules, we can guarantee that each τ node will always have exactly one successor. Figure 4 shows a 4-dimensional rule being factored into two 2-branching rules. Notice how τ is used to allow E' to be the root of a full rule containing only E and F. Without the use of τ, the rule would only be 2-dimensional. Because of this use of chaining τ nodes, there will never be more than one successor for any such node.

Here is a more detailed explanation of the factoring shown in Figure 3. The first illegal link encountered is the second dimensional successor of D. D is re-labeled as D' and the illegal yield is removed. This result is shown in the top branch in the figure. A new tree is formed with its root labeled D' and the illegal yield is set as the yield of the new tree. This

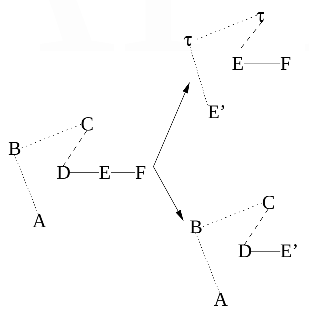

Figure 4: A 4-dimensional local tree of a grammar being factored into 2-branching local trees.

new tree is shown in the lower branch in the figure. Now the two new trees are processed. The upper tree finds another illegal link, the 1-dimensional link on D'. As before, D' is re-labeled D'' and the illegal yield is chopped off (upper branch). A new tree with a root labeled D'' is formed, similar to before. This time, however, we cannot just set the illegal yield as the yield of the new tree, because we only have a 1-dimensional yield, while a 2-dimensional yield is required. This is where τ comes in. To create a 2-dimensional yield, we will set the 1-dimensional yield as the successor of a τ node. Now we have a 2-dimensional yield that can be used (lower branch). This process continues until trees with no illegal nodes are left. In this example, the tree is factored into five 2-branching trees.

4.1 Algorithm

```
;embed takes the label of the root of a
; local tree in a grammar, the local
; yield of that root, and a grammar;
; and adds the rule to the grammar
; in embedded form.
```

```
embed(label,(n-1)-dimensional tree t,
      automaton) {

  new tree startTree = t

  for dim from n-1 to 0 {
    ;Loop Invariant: There are no illegal
    ; links in the nodes above t in
    ; startTree.

    for i from dim+1 to n {
     ;Loop Invariant: t has no illegal
     ; links in all dimensions from dim+1
     ; to i.

      ;At this point, an i-dimensional
      ; link is illegal.
      if (t has link i) {
        new tree u
        relabel t with unique label

        for j from 1 to n-1 {
          ;Loop invariant: u has no
          ; successor in any dimension
          ; less than j, except for an
          ; i-dimensional successor.

          if ( j != i and
              u has no j-dimensional
                link )
            delete j-dimensional link of u

        ;filltree extends u into a d-dimen-
        ; sional tree by addint tau.
        u = filltree(t.link(i), i,
                     automaton)
        ;Now we want to keep looking for
        ; illegal links
        embed(unique label T', u, t')
        ;We can finally remove the
        ; illegal link
        t.unlink(i)
      }
      ;Postcondition: u has no successor
      ; except for an i-dimensional
        ; successors.
      }
      ;Postcondition: t has no illegel links
      ; in any dimension

      ;All illegal links have been fixed, so
      ; move to the next node.
      t = t.link(dim) if dim > 0
    }
    ;Postcondition:  There are no illegal
    ; links anywhere in startTree.

}
```

```
;filltree takes a tree, the dimension of
; the tree, and a grammar and adds
; roots to t so that it is now a complete
; grammar rule.
filltree(n-dim tree t, dim, grammar) {

  for i from dim to n-1 {
    ;Loop invariant:  t is an i-dimensional
    ; local yield.

    if ( i = t->dim() )
       new tree newtree is labeled tprime
    otherwise
       new tree newtree has label of tau
       create empty tree with label
         of tau
       add that tree to the grammar
    newtree->set_link(i,t)
  }
  ;Postcondition: t is an (n-1)-dimensional
  ; local yield.

  return newtree
}
```

5 Growth in Grammar Size

While it does allow us to parse with simpler structures, the factoring process cannot improve the hyper-exponential size of the grammar as a function of the dimensionality. Since every node in the n-branching structure must be represented in the grammar, the growth in the size of the grammar must coincide with the growth in the size of n-branching trees. We can see this by looking at the illegal links present in a 3-branching local tree. Remember that we consider all links that are not in a 2-branching local tree but are in an n-branching local tree illegal. This means that the number of illegal links in a given structure can be defined as $N_k(d) - N_2(d)$, where k is the branching factor of the tree, since all the links in the 2-branching tree must also be in the k-branching tree and no other links in the k-branching tree can be legal.

The *embed* function is designed to fix one illegal link for every new local tree created. There is a side-effect, however, when a higher dimensional illegal link is fixed. In a full tree, every n-dimensional illegal link fixed also fixes an i-dimensional illegal link for all $i < n$. This can be seen in Figure 5. When the 2-dimensional link labeled "A" is fixed, the link labeled "a" will also be fixed as well. When the structure is moved to a new local tree to fix "A", "a" will automatically move to a 2-branching position, so no new local tree needs to be created to fix it. Similarly, when the illegal 3-dimensional link labeled "B" is fixed, the two links labeled "b" will be also be fixed.

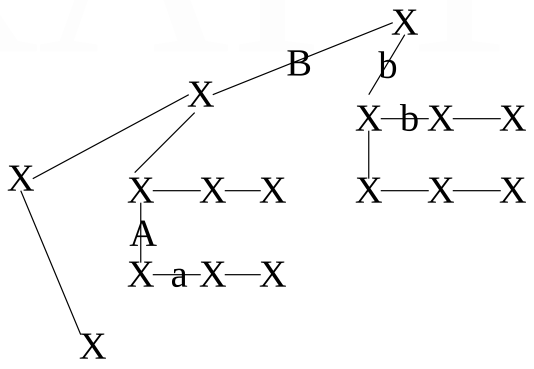

Figure 5: When one of the links labeled with a capital letter is fixed, the links labeled with the same letter in lowercase will also be fixed.

Looking at a full 3-dimensional 3-branching tree, there are 3 illegal 2-dimensional links and 7 illegal 1-dimensional links. Since each time we fix a 2-dimensional link, a 1-dimensional link will also be fixed, we only need to create new rules to fix 4 1-dimensional links ($7 - 3 = 4$). In total, we will create 7 new rules, 3 to fix 2-dimensional links and 4 to fix 1-dimensional links. In the 4-dimensional case, let there be x 3-dimensional illegal links, y 2-dimensional illegal links, and z 1-dimensional illegal links. We will fix x 3-dimensional links, $y-x$ 2-dimensional links, and $z - x - (y - x)$ 1-dimensional links.

$$\begin{aligned} x &= x \\ y &= y - x \\ z &= z - x - (y - x) = z - y \end{aligned}$$

Notice that when we add the terms together to get the total number of links to be fixed, everything cancels except for z, the number of 1-dimensional links. In fact, the number of links to be fixed is always equal to the number of 1-dimensional links.

Theorem 1 *For a full d-dimensional, n-branching local tree, the number of local trees in the factored form required is equal to the number of 1-dimensional illegal links.*

Before we can prove this, we need to establish some lemmas.

Lemma 2 *In a full n-branching tree, when an i-dimensional link is fixed using embed, a j-dimensional link is also fixed for all $0 < j < i$.*

Proof: *Embed* will set the i-dimensional link to the i-dimensional position in a 2-branching tree. Since the i-dimensional link is now a legal link, it is allowed to have an $(i-1)$-dimensional successor by Lemma 1. In a full n-branching tree, it is guaranteed to have such a link. Since that $(i-1)$-dimensional successor is now also a legal link, it is allowed to have an $(i-2)$-dimensional link, which it is guaranteed to have in a full n-branching tree. This continues to the minimal dimension. $\dashv$

Lemma 3 *Let $l(i)$ be the number of i-dimensional links that are not fixed by the side-effect of fixing higher dimensional links as in Lemma 2 and $t(i)$ be the total number of i-dimensional illegal links, then $l(x) = t(x) - t(x+1)$.*

Proof: [by induction on d - x]
Base Case: $l(d-1) = t(d-1)$ (By Lemma 2).
Inductive Hypothesis: For inductive purposes, assume that $l(d-j) = t(d-j) - t(d-j-1)$ for all $1 < j < d-x$.
Induction:

$$\begin{aligned} & l(x) \\ &= t(x) - \sum_{x<i<d}[l(i)] \\ &= t(x) - \sum_{x<i<d-1}[l(i)] + l(d-1) \end{aligned}$$

$$\begin{aligned} &= t(x) - \sum_{x<i<d-2}[l(i)] + \\ &\qquad l(d-2) + t(d-1) \\ &\qquad\qquad \text{(By base case)} \\ &= t(x) - \sum_{x<i<d-2}[l(i)] + \\ &\qquad (t(d-2) - t(d-1)) + t(d-1) \\ &\qquad\qquad \text{(By IH)} \\ &= t(x) - \sum_{x<i<d-2}[l(i)] + t(d-2) \\ &\text{repeating for upper bound } d-3\ldots x+1 \\ &= t(x) - \sum_{x<i<x+2}[l(i)] + t(x+2) \\ &= t(x) - (t(x+1) - t(x+2)) + \\ &\qquad t(x+2) \\ &= t(x) - t(x+1) \end{aligned}$$

$\dashv$

Proof: Proof of Theorem 1
Let the total number of local trees in factored form required be equal to

$$l[1] + l[2] + \ldots + l[d-1]$$

or the sum of all illegal links not fixed by

Lemma 2. Using Lemma 3,

$$\begin{aligned}&l(1) + l(2) + \ldots + l(d-1) = \\ &\quad (t(1) - t(2)) + (t(2) - t(3)) + \\ &\quad (t(3) - t(4)) + \ldots + \\ &\quad (t(d-2) - t(d-1]) + t(d-1) \\ &\quad = t(1).\end{aligned}$$

$\dashv$

Theorem 1 is not true for trees without the maximum number of illegal links. It is the worst case, however, since the side-effect described in Lemma 2 will only fail to fix a lower dimensional link if it does not exist, and therefore does not require a new factored tree anyway.

Even though Theorem 1 proves that we do not need a new grammar rule for every illegal link in the tree, the growth of the grammar will still be hyper-exponential in the dimension of the local trees because the number of 1-dimensional links is a constant fraction of the number of total links. In the worst case (a full tree), every 2-dimensional successor will be the root of a 1-dimensional local tree containing $n-1$ 1-dimensional links, where n is the branching factor. Similarly, every 3-dimensional successor will be the root of a 2-dimensional local tree containing $n-1$ 2-dimensional links. In general, every d-dimensional successor will be the root of a $(d-1)$-dimensional local tree containing $n-1$ $(d-1)$-dimensional links. This is optimal, though, since we cannot get around representing the entire n-branching structure, whether it be in complete or factored form.

6 Future Work

We are working on the converse of the embed operation, which we call "excavate". In a top-down traversal of the tree, every τ node will be replaced with its only successor, and each unique label created during the factoring will be replaced with its n-dimensional successor. If we are careful not to leave a node until it does not contain a τ or a unique label (this may require many operations on one node), we will have the original structure when the traversal has terminated.

While we are confident in the correctness of *embed*, we have not provided a formal proof. When we have finished developing *excavate*, we hope to simultaneously prove the correctness of both algorithms by proving

$$T(G) = \mathit{excavate}(T(\mathit{embed}(G)))$$

References

[BKK+04] David Brown, Ian Kelly, Colin Kern, Alex Lemann, and Greg Sandstrom. Representing multi-dimensional trees. Submitted to this conference, September 2004.

[HMU01] John E. Hopcroft, Jajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2001.

[Rog03] James Rogers. wMSO theories as grammar formalisms. *Theoretical Computer Science*, 293(2):291–320, 2003.