An Application of Level Sequences to Parallel Generation of RootedTrees

Ewa M. Kubicka
University of Louisville
and
Kathleen A. McKeon
Connecticut College

Abstract

An efficient method for generating level sequence representations of rooted trees in a well-defined order was developed by Beyer and Hedetniemi. In this paper, we extend Beyer and Hedetniemi's approach to produce an algorithm for parallel generation of rooted trees. This is accomplished by defining the lexicographic distance between two rooted trees to be the number of rooted trees between them in the ordering of trees produced by the Beyer and Hedetniemi algorithm. Formulas are provided for the lexicographic distance between rooted trees with certain structures. In addition, we present algorithms for ranking and unranking rooted trees based on the ordering of the trees that is induced by the Beyer and Hedetniemi generation algorithm.

Keywords: trees, combinatorial generation, level sequences
AMS Classification: 05C05, 05C85

1 Introduction

The work presented in this paper is motivated by the problem of generating all trees of a specified type in order to evaluate some tree parameter for all trees of that type. For example, when the problem of determining an optimal value among all trees for a particular parameter is very complicated or is NP-complete, it may be necessary to list all trees under consideration, compute the value of the parameter for each tree, and then select the optimal value. Various researchers have addressed the problem of generating all trees of a specified type by representing the trees with different kinds of finite sequences. The general approach is to define a feasible sequence for each tree and then devise an efficient method for producing the feasible sequences for the trees under consideration in lexicographic order. Although the number of trees under consideration is exponential in the number of vertices, the various methods are efficient because, in each method, all trees are produced in *constant amortized time* [9], i.e., the average number of steps required to produce the sequence for the *next* tree is bounded by a constant.

The different sequence representations used by the various authors encode different structural features of the trees. For example, Ruskey and Hu [15] applied this general approach for binary trees with m leaves; they represented such a binary tree with the sequence of the level numbers of its leaves from left to right. Ruskey [13] generalized their approach to k-ary trees. Ruskey [14] also went on to use sequence representations to list all subtrees of an ordered tree. Zaks [22] used sequences which result from performing a pre-order traversal of the tree and recording the number of children of each vertex. Zaks and Richards [24] generalized Zaks' sequences to k-ary trees and other combinatorial objects and Zaks [23] has also extended his approach to the case of a general tree. A number of other authors have used sequences to represent and generate trees (see Klarner [5]; Gupta and Lee [2]; Gupta, Lee and Wong [3]; Li [10]; Ruskey and Proskurowski [16]; Trojanowski [18]; Vajnovszki [19]; Wilf and Yoshimura [20]; Yoshimura [21]).

The method under consideration here uses level sequences to represent and generate unlabeled rooted trees. Level sequences were first defined by Scions [17] and were used by Beyer and Hedetniemi [1] to generate rooted trees in constant time per tree. The *level sequence* for an unlabeled rooted tree on n vertices is defined as follows. The root of the tree is assigned the level 1 and each child of a vertex of level k is assigned the level $k+1$. A sequence of length n is obtained by performing a pre-order traversal of the tree and recording these levels. To obtain a unique level sequence for each rooted tree, an ordering is imposed on the subtrees consisting of a vertex and its descendants. The subtrees are ordered recursively from left to right in nondecreasing order lexicographically by their level sequences. This ordering of the subtrees results in an ordered rooted tree, which is called the *canonical ordering* of the underlying rooted tree. Note that the canonical ordering is the ordering of the rooted tree that produces the lexicographically largest level sequence. This level sequence is the *canonical level sequence* of the underlying rooted tree. In Figure 1.1, T^* is the canonical ordering of the underlying rooted tree T and produces the canonical level sequence. Throughout this paper, the canonical level sequence is referred to simply as the level sequence of the rooted tree.

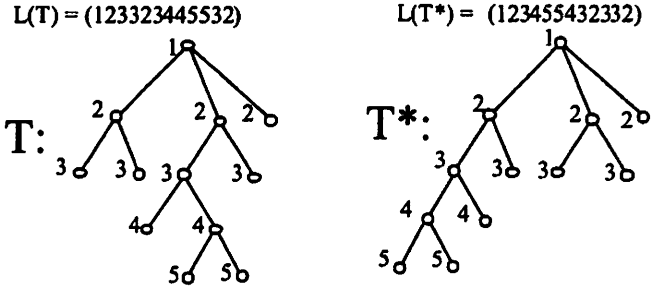

Figure 1.1

Beyer and Hedetniemi developed concise, straightforward rules for generating all rooted trees on n vertices in decreasing lexicographic order, i.e., starting with the largest level sequence $(1, 2, \ldots, n)$, which corresponds to the path P_n rooted at an end-vertex, and ending with the smallest level sequence (1, 2, 2, ..., 2), which corresponds to the star $K_{1,n-1}$ rooted at its center. For the level sequence $L = (a_1 \ldots a_n)$ of a rooted tree, let p denote the largest integer such that $a_p > 2$ and let q be the largest integer such that $q < p$ and $a_q = a_p - 1$. Note that a_p is the level of v_p, the rightmost vertex in the tree that is at a level greater than 2 and a_q is the level of its parent, v_q. In general, we use v_j to denote the vertex whose level is given by the j^{th} entry of the level sequence L. Then the level sequence immediately following L and called its successor, $SF(L) = (s_1 \ldots s_n)$ is given by:

$$s_i = \begin{cases} a_i \text{ for } 1 \leq i < p \\ s_{i-p+q} \text{ for } p \leq i \leq n \end{cases} \qquad (1)$$

Note that the effect of the successor function is to replace the vertices $v_p, v_{p+1}, \ldots, v_n$ by as many copies of the subtree consisting of v_q and its descendants $v_{q+1}, \ldots, v_{p-1}$ as necessary to result in a tree with n vertices. For example, in the level sequence $L = (1, 2, 3, 4, 4, 2, 2), p = 5, q = 3$, and $SF(L) = (1, 2, 3, 4, 3, 4, 3)$. To illustrate further, the level sequences of all twenty rooted trees of order six are listed below in the order in which they are generated by the Beyer and Hedetniemi algorithm.

1. (1,2,3,4,5,6)	6. (1,2,3,4,4,4)	11. (1,2,3,4,3,2)	16. (1,2,3,3,2,3)
2. (1,2,3,4,5,5)	7. (1,2,3,4,4,3)	12. (1,2,3,4,2,3)	17. (1,2,3,3,2,2)
3. (1,2,3,4,5,4)	8. (1,2,3,4,4,2)	13. (1,2,3,4,2,2)	18. (1,2,3,2,3,2)
4. (1,2,3,4,5,3)	9. (1,2,3,4,3,4)	14. (1,2,3,3,3,3)	19. (1,2,3,2,2,2)
5. (1,2,3,4,5,2)	10. (1,2,3,4,3,3)	15. (1,2,3,3,3,2)	20. (1,2,2,2,2,2)

Beyer and Hedetniemi showed that the average number of entries scanned and altered per tree generated is bounded by a small constant independent of the order of the rooted trees being generated. Kubicka ([6], [7]) provided an asymptotic limit for this average and showed that the average number of entries scanned and altered approaches $\frac{1}{1-\rho} = 1.511$, where ρ is the radius of convergence of the generating functions for rooted and unrooted trees.

In addition, level sequences have an advantage over other sequence representations of trees since much information about the structure of the tree is easily obtained from its level sequence. This is because the entries of the level sequence corresponding to the parent and the children of a vertex are easily identified, and the level sequence of each subtree consisting of a vertex and its descendants appears as a contiguous and easily recognizable subsequence of the level sequence of the tree. Kubicka [6, 7] took advantage of these characteristics of level sequences to extend the Beyer and Hedetniemi algorithm to generate and, simultaneously, evaluate tree parameters for all rooted trees in constant amortized time.

In this paper we adapt the Beyer and Hedetniemi algorithm to generate rooted trees of a given order in parallel. The basic approach is to assign the processors approximately equal numbers of trees to generate. In order to identify the starting level sequence (tree) and number of level sequences (trees) to be generated by each processor, we must be able to determine the *lexicographic distance* between two rooted trees T_1 and T_2, i.e., the number of rooted trees (level sequences) generated to reach the tree T_2 when the Beyer and Hedetniemi algorithm is applied starting with T_1. For example, among trees of order six, the lexicographic distance between the trees represented by the level sequences $(1,2,3,4,4,3)$ and $(1,2,3,4,2,2)$ is 6. While the lexicographic distance is very difficult to determine in general, we give formulas for the lexicographic distance between trees with certain structures which are easily identified from the level sequence. This information enables us to determine the lexicographic distance between any two rooted trees of the same order and to determine the starting sequences and counts for each processor in the parallel generation.

We also address the ranking and unranking problems for the Beyer and Hedetniemi rooted tree generation algorithm. The ranking problem for a given tree generation algorithm is to determine the position of a specified tree in the ordering of all trees as induced by that generation algorithm. Similarly, the unranking problem for a particular tree generation algorithm is to determine the i^{th} tree in the ordering of all such trees as induced by that particular generation algorithm. Thus, each generation algorithm results in new ranking and unranking problems for the type of tree under consideration. For rooted trees represented by level sequences, the ranking problem is to determine the rank of the given rooted tree on n vertices in the ordering of all rooted trees with n vertices as induced by the Beyer and Hedetniemi algorithm. Similarly, the unranking problem is to determine the i^{th} rooted tree in the Beyer and Hedetniemi lexicographic ordering of all rooted trees with a given number of vertices. While other authors have addressed the ranking and unranking problems for other generation algorithms, these problems have not previously been considered for the Beyer and Hedetniemi generation algorithm and level sequence representation of trees. Together with known results for counting trees by height, our method for determining lexicographic distances is used to produce ranking and unranking algorithms relative to the Beyer and Hedetniemi generation algorithm. The unranking algorithm is also used in the parallel generation algorithm.

2 Definitions and Formulas

Throughout this paper, all trees are assumed to be unlabeled rooted trees. We denote the order of a tree by n and its (canonical) level sequence by L. Since each tree is identified with its level sequence, reference to a level sequence will also be reference to the tree that the level sequence represents. The lexicographic distance between trees T_1 and T_2, where the level sequence of T_1 is lexicographically larger than that of T_2, is denoted by $N(T_1, T_2)$. Equivalently, the number of level sequences generated to reach the level sequence L_2 from the level sequence L_1 is denoted by $N(L_1, L_2)$. In order to describe the particular types of trees between which we can determine the lexicographic distance, we need the following definitions and notations. We use v_j to denote the vertex whose level is given by the j^{th} entry of the level sequence L.

As noted earlier, the level sequence of a subtree consisting of a vertex and its descendants appears as a contiguous subsequence. If such a subsequence forms the end of the level sequence, then the subtree consisting of the vertex and its descendants is called a *rightmost subtree* of the tree. In the tree pictured in Figure 2.1, the rightmost subtree rooted at the vertex v corresponds to the subsequence $(2, 3, 4, 4, 3, 4)$ and the rightmost subtree rooted at the vertex w corresponds to the subsequence $(3, 4)$.

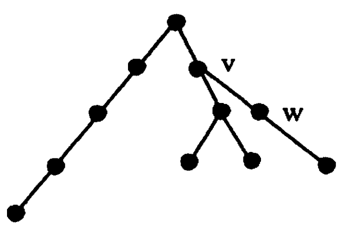

Figure 2.1 $L(T) = (12345234434)$

Definition 1. *An ancestor v_j of a vertex v_i is said to lie on a long enough path (relative to v_i) if there is a path with at least $(n-i+1)$ vertices that begins at v_j and does not include v_i.*

For example, the level sequence $L = (1\ 2\ 3\ 4\ 5\ 6\ 2\ 3\ 4\ 5\ 5\ 3\ 3\ 3\ 3)$, the parent v_7 of the vertex v_{12} lies on a path with 4 vertices which is a long enough path relative to v_{12}.

Our general approach is to start with a tree T_1 in which a rightmost subtree has one of four particular forms and end with a tree T_2 in which that subtree has been replaced by a star with its center at the root of T_2. In the first two cases the rightmost subtree is a star and in the other two cases the rightmost subtree is a path. We denote the leftmost vertex changed by v_i and note that we are changing the last $m = n - i + 1$ entries of the level sequence to 2's to get the level sequence of T_2. In each case the property that a particular ancestor of v_i lies on a long enough path allows us to determine the form of the intermediate trees. This is due to the fact that the Beyer and Hedetniemi algorithm changes the tree by replacing the rightmost vertices of the tree with copies (full or partial) of the subtree rooted at the parent of the leftmost vertex that is being replaced.

Definition 2. *Let T_1 denote the tree with level sequence $L = (l_1, l_2, \ldots, l_{i-1}, k, k, \ldots, k)$, where the parent of v_i lies on a long enough path, $l_{i-1} \geq k$, and $m = n - i + 1$. Let T_2 denote the tree with level sequence $L' = (l_1, l_2, \ldots, l_{i-1}, 2, 2, \ldots, 2)$. Then we define $S(m,k) = N(T_1, T_2) + 1$.*

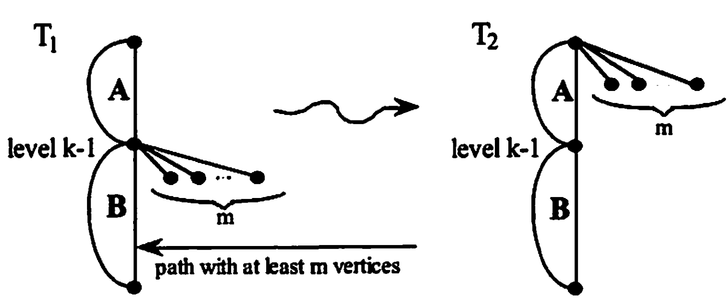

Figure 2.2 $S(m,k) = N(T_1, T_2) + 1$

For example, if T_1 has level sequence
$L = (1,2,3,4,5,5,5,2,3,4,5,5,4,4,4)$ and T_2 has level sequence
$L' = (1,2,3,4,5,5,5,2,3,4,5,5,2,2,2)$, then $S(3,4) = N(T_1, T_2) + 1$.

Definition 3. *Let T_1 denote the tree with level sequence $L = (l_1, l_2, \ldots, l_{i-2}, k-1, k, k, \ldots, k)$, where the grandparent of v_i lies on a long enough path and $m = n - i + 1$. Let T_2 denote the tree with level sequence $L' = (l_1, l_2, \ldots, l_{i-2}, k-1, 2, 2, \ldots, 2)$. Then we define $S'(m,k) = N(T_1, T_2) + 1$.*

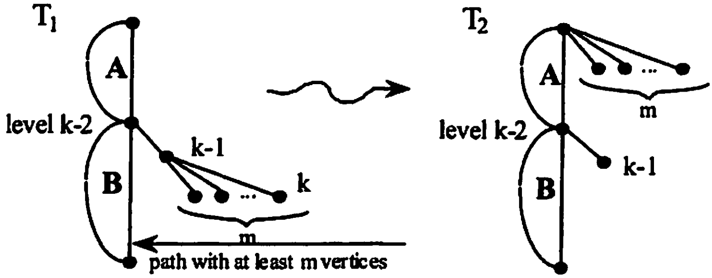

Figure 2.3 $S'(m,k) = N(T_1,T_2) + 1$

For example, if T_1 has level sequence $L = (1,2,3,4,5,6,6,6,2,3,4,5,6,4,5,5,5)$ and T_2 has level sequence $L' = (1,2,3,4,5,6,6,6,2,3,4,5,6,4,2,2,2)$, then $S'(3,5) = N(T_1,T_2) + 1$.

Definition 4. *Let T_1 denote the tree with level sequence $L = (l_1, l_2, \ldots, l_{i-1}, k, k+1, \ldots, k+m-1)$, where the parent of v_i lies on a long enough path, $l_{i-1} > k$, and $m = n-i+1$. Let T_2 denote the tree with level sequence $L' = (l_1, l_2, \ldots, l_{i-1}, 2, 2, \ldots, 2)$. Then we define $C(m,k) = N(T_1,T_2) + 1$.*

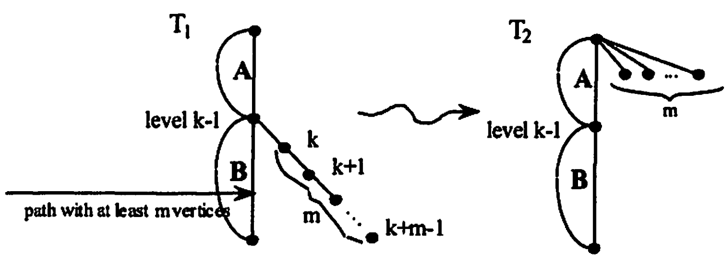

Figure 2.4 $C(m,k) = N(T_1,T_2) + 1$

For example, if T_1 has level sequence $L = (1,2,3,4,5,6,6,5,4,5,6)$ and T_2 has level sequence $L' = (1,2,3,4,5,6,6,5,2,2,2)$, then $C(3,4) = N(T_1,T_2) + 1$.

Definition 5. *Let T_1 denote the tree with level sequence $L = (l_1, l_2, \ldots, l_{i-2}, k-1, k, \ldots, k+m-1)$, where the grandparent of v_i lies on a long enough path and $m = n-i+1$. Let T_2 denote the tree with level sequence $L' = (l_1, l_2, \ldots, l_{i-2}, k-1, 2, 2, \ldots, 2)$. Then we define $C'(m,k) = N(T_1,T_2) + 1$.*

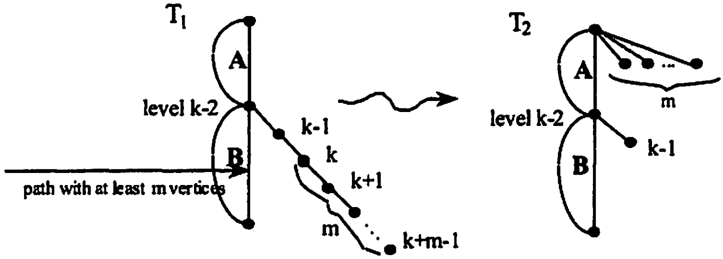

Figure 2.5 $C'(m,k) = N(T_1, T_2) + 1$

For example, if T_1 has level sequence $L = (1\ 2\ 3\ 4\ 5\ 6\ 6\ 5\ 4\ 5\ 6)$ and T_2 has level sequence $L' = (1\ 2\ 3\ 4\ 5\ 6\ 6\ 5\ 4\ 2\ 2)$, then $C'(2,5) = N(T_1,T_2)+1$.

Note that the four parameters S, C, S', and C' do not depend on the order of the trees under consideration.

These parameters can be expressed in terms of the numbers of rooted trees of a given order and height. Recall that the height of a rooted tree is the length of a longest path from the root to an end-vertex. Let $T(n)$ represent the number of rooted trees of order n and $H(n,h)$ represent the number of rooted trees with n vertices and height h. Methods for computing $T(n)$ and $H(n,h)$ can be found in Harary and Palmer[4] and Riordan [12], respectively. Formulas expressing S, S', C, and C' in terms of T and H are given in the following four theorems.

Theorem 1. *i) For $m \geq 2$ and $k \geq 3$,*
$S(m,k) = \sum_{r=0}^{m} \sum T(n_1)T(n_2)\dots T(n_{k-2})$ *where the second sum is taken over all ordered $(k-2)$-tuples $(n_1, n_2, \dots, n_{k-2})$ such that $n_1 + n_2 + \dots + n_{k-2} = k - 2 + m - r$.*

ii) For all $m \geq 1$, $S(m,2) = 1$.

Proof. Let L and L' be level sequences representing the trees T_1 and T_2 as described in Definition 2 and for which $N(L,L') + 1 = S(m,k)$. As the Beyer and Hedetniemi algorithm is applied repeatedly to progress from T_1 to T_2, the last m entries of the level sequence are modified to produce a level sequence that is lexicographically smaller. Therefore, any rooted tree T'' of order n whose level sequence is lexicographically between L and L' has the form pictured below. Note that r is the number of vertices in the rightmost subtree that have not been altered and satisfies $0 \leq r < m$. In addition, the remaining $m - r$ vertices have been distributed to form the subtrees, A_i, where each A_i is a rooted tree of order $1 \leq n_i \leq m + 1$, and $m = r + \sum_{i=1}^{k-2}(n_i - 1)$. Since $r < m$, it is clear that the level sequence corresponding to the tree in Figure 2.6 is lexicographically smaller than L.

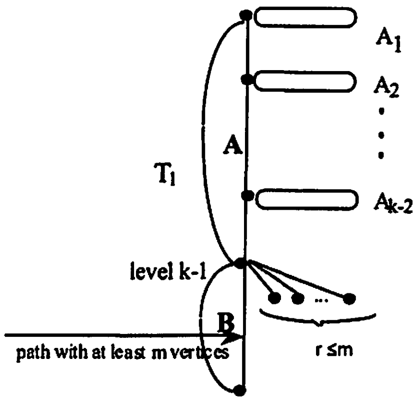

Figure 2.6

We claim that there are no other restrictions on the A_i's, i.e., for any set of A_i's satisfying the conditions on their orders stated above, the level sequence of the tree in Figure 2.6 is the canonical level sequence for the underlying tree. To verify this, remove all the vertices of the A_i's, except for their roots from T''. The resulting tree is the canonical ordering of the underlying rooted tree. Since the parent of v_i lies on a long enough path relative to v_i, the height of B is at least $m - 1$. Now reattach A_{k-2} as in Figure 2.6. Since A_{k-2} has most $m + 1$ vertices, counting its root, it has height at most m. In fact, if it does have height equal to m, A_{k-2} is a path on $m + 1$ vertices. Therefore, since A_{k-2} is attached at level $k - 2$, the canonical ordering is preserved. By similar arguments, we see that the canonical ordering is preserved when we reattach the remaining subtrees, A_i, for $i \leq k - 3$. Therefore, there is a one-to-one correspondence between all trees from T_1 to T_2, inclusive, and the set of all $(k - 1)$-tuples $(K_{1,r}, A_1, A_2, \ldots, A_{k-2})$ where each A_i ranges over all rooted trees of order n_i and $K_{1,r}$ is the star with r end-vertices. The formula in part (i) counts the number of these $(k - 1)$-tuples.

Part (ii) follows directly from the definition of $S(m, k)$. □

Theorem 2. *i) For $m \geq 2$ and $k \geq 4$,*

$$S'(m,k) = \sum_{n_{k-2}=3}^{m+2} [(H(n_{k-2},2)+1)\sum T(n_1)T(n_2)\dots T(n_{k-3})]$$
$$+\sum T(n_1)T(n_2)\dots T(n_{k-3})$$

where the second sum is taken over all ordered $(k-3)$-tuples $(n_1, n_2, \dots, n_{k-3})$ such that $n_1 + n_2 + \dots + n_{k-3} = k - 1 + m - n_{k-2}$ and the third sum is taken over all ordered $(k-3)$-tuples $(n_1, n_2, \dots, n_{k-3})$ such that $n_1 + n_2 + \dots + n_{k-3} = k + m - 3$

ii) For all $m \geq 1$, $S'(m,3) = H(m+2,2)+1$.

iii) For all $m \geq 1$, $S'(m,2) = 1$.

Proof. Let L and L' be level sequences representing the trees T_1 and T_2 as described in Definition 3 and for which $N(L,L')+1 = S'(m,k)$. The formula in (i) follows from the observation that any rooted tree T'' of order n whose level sequence is lexicographically between L and L' has the form pictured below.

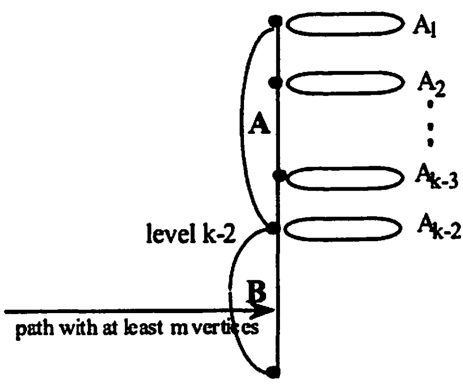

Figure 2.7

Each subtree A_i is a rooted tree of order n_i and $m+1 = \sum_{i=1}^{k-2}(n_i - 1)$. For $1 \leq i \leq k-3$, $1 \leq n_i \leq m$. Note that in T_1, A_{k-2} is the lexicographically largest tree with height 2 and $m+2$ vertices and $A_i = K_1$ for $i = 1, 2, \dots$, $k-3$. Thus, in any intermediate tree, the height of A_{k-2} can be either 1 or

2 and $2 \leq n_{k-2} \leq m+2$. There are no restrictions on the $A_i's$ for $i \leq k-3$ except for the condition on the order stated above. By the same argument as in the proof of Theorem 1, the level sequence of the tree in Figure 2.7 is lexicographically smaller than that of T_1 and is the canonical level sequence of T''. Therefore, there is a one-to-one correspondence between the trees from T_1 to T_2, inclusive, and the set of all $(k-2)$-tuples $(A_1, A_2, ..., A_{k-2})$ where, for $i \leq k-3$, each A_i ranges over all rooted trees of order n_i and A_{k-2} ranges over all rooted trees of order n_{k-2} and height at most 2. The formula in part (i) counts the number of these $(k-2)$-tuples.

Part (ii) follows from Figure 2.7 also. However, with $k = 3$, only the subtree A_{k-2} is present in the intermediate tree and ranges over all rooted trees of order $m+2$ and height at most 2.

Part (iii) follows directly from the definition of $S'(m,k)$. □

Theorem 3. *For $m \geq 1$ and $k \geq 2$, $C(m,k) = \sum T(n_1)T(n_2)\ldots T(n_{k-1})$ where the sum is taken over all ordered $(k-1)$-tuples $(n_1, n_2, \ldots, n_{k-1})$ such that $n_1 + n_2 + \ldots + n_{k-1} = k + m - 1$.*

Proof. Let L and L' be level sequences representing the trees T_1 and T_2 as described in Definition 4 and for which $N(L, L') + 1 = C(m,k)$. The formula for $C(m,k)$ follows from the observation that any rooted tree T'' of order n whose level sequence is lexicographically between L and L' has the form pictured below.

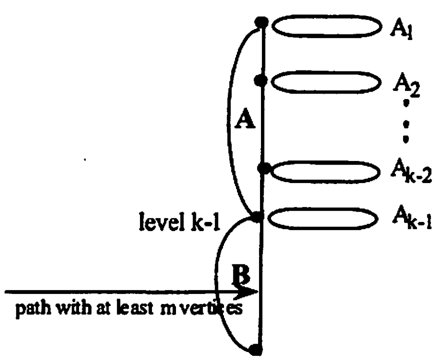

Figure 2.8

Each subtree A_i is a rooted tree of order $1 \leq n_i \leq m+1$ and $m = \sum_{i=1}^{k-1}(n_i - 1)$. Note that in T_1, A_{k-1} is a path on $m+1$ vertices and $A_i = K_1$ for $i = 1, 2, ..., k-2$. Thus, in any intermediate tree, there are no restrictions on any of the $A_i's$ except for the conditions on their orders. By the same argument as in the proofs of the previous theorems, there is a one-to-one correspondence between the trees from T_1 to T_2, inclusive, and the set of all $(k-1)$-tuples $(A_1, A_2, ..., A_{k-1})$ where each A_i ranges over all rooted trees of order n_i. The formula for $C(m,k)$ counts these $(k-1)$-tuples. □

Theorem 4. *i) For $m \geq 2$ and $k \geq 4$,*

$$C'(m,k) = \sum_{n_{k-2}=2}^{m+2} \sum T(n_1)T(n_2)\ldots T(n_{k-3})T(n_{k-2})$$

where the second sum is taken over all ordered $(k-3)$-tuples $(n_1, n_2, \ldots, n_{k-3})$ such that $n_1 + n_2 + \ldots + n_{k-3} = k + m - 1 - n_{k-2}$.

ii) For $m \geq 2$, $C'(m,3) = T(m+2)$.

iii) For $m \geq 2$, $C'(m,2) = T(m+1)$.

iv) For $k \geq 2$, $C'(1,k) = k-1$.

Proof. Let L and L' be level sequences representing the trees T_1 and T_2 as described in Definition 5 and for which $N(L,L')+1 = C'(m,k)$. The formula in (i) follows from the observation that any intermediate rooted tree T'' of order n whose level sequence is lexicographically between L and L' has the form pictured in Figure 2.7. For $1 \leq i \leq k-3$, each subtree A_i is a rooted tree of order $1 \leq n_i \leq m+1$, A_{k-2} is a rooted tree of order $2 \leq n_{k-2} \leq m+2$, and $m+1 = \sum_{i=1}^{k-2}(n_i - 1)$. By the same arguments as in the proofs of the previous theorems, there is a one-to-one correspondence between the trees from T_1 to T_2, inclusive, and the set of all $(k-2)$-tuples $(A_1, A_2, ..., A_{k-2})$ where each A_i ranges over all rooted trees of order n_i. The formula in part (i) counts the number of these $(k-2)$-tuples.

Part (ii) follows from Figure 2.7 also. However, with $k=3$, only the subtree A_1 is present in intermediate tree and ranges over all rooted trees of order $m+2$.

For part (iii) note that $C'(m,2) = C(m,2)$. Part (iv) follows directly from the definition of $C'(m,k)$. □

Observe the following simplifications of the formulas from Theorems 1-4, $S(1,k) = S'(1,k) = C(1,k) = C'(1,k) = k-1$ and $C(m,2) = T(m+1)$.

The formulas for S, S', C and C' can be represented in a more concise manner by introducing a new function $A(m,k) = \sum T(n_1)T(n_2)\ldots T(n_k)$ where the sum is taken over all ordered k-tuples $(n_1, n_2, \ldots, n_k)$ such that

$n_1 + n_2 + \ldots + n_k = m$. Then

$$S(m,k) = \sum_{r=0}^{m} A(k-2+m-r, k-2),$$
$$S'(m,k) = A(k+m-3, k-3) + \sum_{n_{k-2}=3}^{m+2} (H(n_{k-2}, 2)+1)A(k-1+m-n_{k-2}, k-3),$$
$$C(m,k) = A(k+m-1, k-1) \textit{ and}$$
$$C'(m,k) = \sum_{n_{k-2}=2}^{m+2} T(n_{k-2})A(k+m-1-n_{k-2}, k-2).$$

Since the A's satisfy the recurrence relation

$$A(m,k) = \sum_{n_k=0}^{m} T(n_k)A(m-n_k, k-1),$$

the complexity of computing $A(i,j)$ for all $0 \le i \le m$ and $1 \le j \le k$ is $O(km^2)$. Therefore, it is easily seen that the complexity of computing $S(m,k)$, $S'(m,k)$, $C(m,k)$ and $C'(m.k)$ is $O(k(k+m)^2)$. Tables 4-7 containing values for S, S', C and C' appear in the appendix. Computer programs that compute values for S, S', C, C', H and T may be obtained from the authors.

If L and L' are level sequences that have the forms described in the definitions of S,S', C, and C', then we say that we apply the corresponding parameter (S, S', C, C') or the successor function to *jump* from L to L'. We denote the size of this jump, i.e., the number of level sequences between L and L' including L' by J. If S, S', C or C' is used for the jump we set J equal to one less than the value of the parameter $(S, S', C$ or $C')$ that is used for the jump. If the successor function is used, we set J equal to 1. Note that any level sequence L has either a star or a path as a rightmost subtree. If it is a star with its center at the root of the tree, then the successor function must be used. Otherwise, one of these four parameters may be applied to jump to some lexicographically smaller level sequence. We identify the parameter that can be used by scanning the level sequence from right to left. A star is a rightmost subtree if the rightmost entries of the level sequence are equal. A path is a rightmost subtree if the entries of the level sequence decrease by 1 moving from right to left. Thus, we can use the parameters S, S', C and C' and jumps to determine the number of level sequences between a pair of level sequences or to determine the level sequence that is the i^{th} sequence following a given level sequence.

3 Parallel Generation and Processing of Trees

In this section we present an algorithm for parallel generation of trees based on the Beyer and Hedetniemi sequential algorithm. Although the Beyer and Hedetniemi algorithm for sequential generation of trees is as efficient as possible (constant time per tree or constant amortized time), the number of rooted trees of order n is exponential in n. Consequently, any gains achieved through parallel generation of trees would be most significant for generating trees of large order or for generating trees and simultaneously studying certain properties of the trees.

As noted previously, Kubicka [6, 7] took advantage of the structural information that is encoded in a tree's level sequence and adapted the Beyer and Hedetniemi algorithm for sequential generation of trees to evaluate tree properties for all trees in constant amortized time. This approach is particularly useful in cases when it is necessary to conduct an exhaustive search of all trees, for example, when the problem of finding an optimal value among all trees for a particular parameter is very complicated or is NP-complete. Our algorithm for parallel generation of trees also could be adapted to evaluate tree properties for each tree generated as Kubicka [6, 7] did for the sequential algorithm.

For parallel generation of trees, we use the parameters S, S', C, and C' to compute starting sequences and counts for the number of sequences to be generated sequentially by each processor. The main idea is to assign to each processor approximately the same number of trees to generate. Let P be the number of processors, n the order of the trees under consideration, and $T(n)$ the number of trees with n vertices. Then $M = \lceil \frac{T(n)}{P} \rceil$ is the approximate number of trees that each processor will generate. Separate processors will be devoted to assigning starting sequences and counts to each of the other processors. We maintain a set of available, i.e., currently not working, processors and call this set SAP. When the starting sequence and count of number of sequences to be generated by a single processor is determined, we assign an available processor to generate those sequences. When the processor has completed its work, it is returned to SAP.

Since trees with the same height occur consecutively in the lexicographic ordering of the level sequences, generation will be by height. The assignment of heights to processors is done in two phases. In phase 1, a single processor will generate all sequences for trees of several different heights, those heights for which the number of trees does not exceed M. In phase 2, several processors will generate the sequences for all trees of one height for each of the remaining heights. The assignment of starting sequences and counts for these remaining heights is done in parallel; for each such height, one processor is devoted to computing the starting sequences and counts for the processors that actually generate the trees of that particular height.

To make the assignments for a height h, its directing processor starts with the first (lexicographically largest) tree of that height, $L_1 = (1, 2, 3, \ldots, h-1, h, h, \ldots, h)$. Then the maximum feasible jumps in the form of S, S', C, C' or the successor function are applied successively until the sum of the jump sizes reaches or just surpasses M. The sum of the jump sizes and the sequence L_1 are the count and starting sequence that are assigned to the first available processor P_1 from SAP. If the number of remaining trees of height h is greater than M, the successor of the tree reached by the last jump becomes the starting sequences L_2 for the next available processor P_2 from SAP and, as above, the directing processor uses the jumps to determine the count for P_2. This process is repeated until the number of remaining trees of height h is at most M. Then this number and the last starting sequence are assigned to the directing processor to generate the remaining trees of height h.

The following theorem shows that each processor will generate its assigned level sequences as efficiently as possible, i.e., in constant amortized time.

Theorem 5. *(i) Generation of any N successive trees of order n by the Beyer and Hedetniemi algorithm takes $O(N)$ time.*

(ii) The average number of entries of the level sequence that are scanned and altered by the Beyer and Hedetniemi algorithm to generate all trees of a fixed height h and order n is asymptotic to $\frac{1}{1-\sigma_h}$, where is σ_h is the radius of convergence of the generating function for rooted trees of height h.

Proof. Part (i) follows directly from Beyer and Hedetniemi's proof [1] that their algorithm generates all rooted trees of a given order in constant amortized time.

For part (ii), we follow the technique in Kubicka's proof [6, 7] that provided the asymptotic limit on the average number of entries scanned and altered by the Beyer and Hedetniemi algorithm for all rooted trees.

Let $H_h(x)$ be the generating function for rooted trees of height h. Then $x^i H_h(x)$ is the generating function for rooted trees of height h with at least i end-vertices on level 2. Thus, the expression $x^{i-1}H_h(x) - x^i H_h(x)$ represents the generating function for rooted trees of height h with exactly $i-1$ end-vertices on level 2. When the Beyer and Hedetniemi algorithm is applied to such a tree, exactly i entries of the level sequence are scanned and altered to produce the level sequence for the next tree. Now let $G(x) = \sum_{n=0}^{\infty} g_n x^n$ be the generating function describing the complexity of the algorithm, i.e., g_n corresponds to the number of steps the algorithm has to perform in order to generate all rooted trees of order n. Then we have:

$$
\begin{aligned}
G(x) &= (H_h(x) - xH_h(x)) + 2(xH_h(x) - x^2H_h(x)) \\
&\quad + 3(x^2H_h(x) - x^3H_h(x)) + \ldots \\
&= H_h(x)[1 + x + x^2 + x^3 + \ldots] = H_h(x)\frac{1}{1-x}.
\end{aligned}
$$

Therefore, as in [7], we can conclude that $g_n \sim H(n,h)\frac{1}{1-\sigma_h}$, and the average number of steps per tree for the Beyer and Hedetniemi algorithm to generate all trees of a fixed height h and order n is asymptotic to $\frac{1}{1-\sigma_h}$. □

Example 1. Now we shall demonstrate the gains in efficiency achieved by this approach to parallel generation for trees of order 20. We assume that one time unit is required to perform one jump or to produce the next level sequence. Therefore, we measure the efficiency in terms of the number of level sequences actually generated. Suppose we have 128 processors, then M is approximately 105,000. In phase 1, one processor will generate all trees of heights 1, 2 and 3 (44,533 trees), and another will generate all trees of heights 13 - 19 (94, 504 trees). The data for the trees of heights 4 - 12 generated in phase 2 is summarized in Table 1.

From Table 1, 16 processors are allocated to height 5. One processor determines and assigns the starting sequences and counts to the other 15 processors that generate the trees in parallel. This processor produces 174,009 sequences in order to make these assignments. The maximum number of trees generated by one of the other 15 processors is 106,614 trees. Therefore, the total time units required to generate the 1,599,205 trees of height 5 is 280,623. Since this is the maximum time required to generate the trees of one particular height, this is the total time required to generate all the trees with 20 vertices in parallel. Sequential generation takes 12,826,228 units of time to generate the 12,826,228 trees with 20 vertices. Consequently, this method of parallel generation is approximately 45 times faster than sequential generation for trees of order 20.

Our empirical data shows that we get greater gains in efficiency for trees of higher order, i.e., as the order of the trees increases, the ratio of the time for sequential generation to the time for parallel generation increases. A theoretical analysis requires knowledge of the distribution of the jumps made when we begin with the first tree of a particular height. This seems to be extremely difficult to ascertain since it depends on the height and the structure of the trees produced by the jumps.

Ht	# of Trees	# of Processors	Time for Processor Assignment	Max # of Trees Generated per Processor	Total Time for Parallel Generation
4	495417	6	84602	99083	183685
5	1599205	16	174009	106614	280623
6	2564164	25	168343	106840	272183
7	2740448	27	94888	105402	200290
8	2256418	22	37565	107448	145013
9	1530583	15	11563	109327	120890
10	880883	9	3285	110110	113395
11	435168	4	943	108792	109735
12	184903	1	278	184903	185181

Table 1: Data for Parallel Generation of Trees of Order 20

4 Ranking and Unranking Algorithms

As described in the introduction, the ranking problem is to determine the rank of a specified tree in a particular ordering of all trees with a given number of vertices and the unranking problem is to determine the i^{th} tree in a particular ordering of all trees with a given number of vertices. Consequently, each generation algorithm that produces trees in a particular order results in new, unsolved ranking and unranking problems. In this section, we address the ranking and unranking problems for the ordering on rooted trees as induced by the Beyer and Hedetniemi algorithm.

In this context, the ranking problem is: given a level sequence L of length n, determine its rank in the (decreasing) lexicographic ordering of rooted trees with n vertices. Note that trees with the same height occur consecutively in the lexicographic ordering of the level sequences of trees with a fixed number of vertices. Consequently, we determine the height of the given tree and find its rank among the trees of the same height. The algorithm is as follows.

Step 1. Determine the height h of the tree represented by the level sequence L:

- Scan the entries of L from left to right to find the smallest integer i such that $l_{i+1} \neq l_i + 1$.
- Set $h = i - 1$ and $L' = (1, 2, 3, \ldots, h, h + 1, 2, 2, , 2)$, the last tree of height h.
- Initialize $t = 0, L_t = L$, and $M = 0$.

Step 2. Determine the distance between L and L':

While $L_t \neq L'$, do
Begin

- Apply the largest feasible jump (S, S', C, C' or the successor function) to L_t to produce the level sequence L_{t+1} with jump size J such that $L' \leq L_{t+1} < L_t$.
- Set $M = M + J$ and $t = t + 1$.

End {While}

Step 3. Calculate the rank of L: Rank$(L) = \sum_{i=h}^{n-1} H(n,i) - M$.

Note that the ranking algorithm can be modified to determine the lexicographic distance between any two trees with the same number of vertices. Once the ranks of the two trees are known, the lexicographic distance between the trees is the difference in their ranks.

In the context of the ordering induced by the Beyer and Hedetniemi algorithm, the unranking problem is to determine the i^{th} level sequence in the decreasing lexicographic ordering of the level sequences of rooted trees of order n for given integers i and n. The unranking algorithm is, for the most part, a reversal of the ranking algorithm and is given below.

Step 1. Determine the height h of the i^{th} tree and initialize variables:

- Set h equal to the smallest integer such that $\sum_{j=h+1}^{n-1} H(n,j) < i \leq \sum_{j=h}^{n-1} H(n,j))$.
- Set $L_0 = (1, 2, 3, \ldots, h, h+1, h+1, \ldots, h+1)$, the first tree of height h.
- Set $M = \sum_{j=h+1}^{n-1} H(n,j) + 1$, the rank of L_0, and initialize $t = 0$.

Step 2. Determine the level sequence with rank i:
While $M < i$, do
Begin

- Apply the largest feasible jump (S, S', C, C' or the successor function) to L_t to produce the level sequence L_{t+1} such that $M + J \leq i$.

- Set $M = M + J$ and $t = t + 1$.

End {While}

Step 3. L_t is the level sequence of the tree with rank i.

Note that the unranking algorithm is actually applied in the parallel generation algorithm. This method of unranking could also be part of an algorithm to produce a random tree by first randomly producing an integer corresponding to the rank of the tree and then identifying the level sequence of the tree with that rank.

Example 2. We will use the ranking algorithm to determine the rank of the level sequence L = (1 2 3 4 5 3 4 4 4 4) among the 719 rooted trees with 10 vertices. From step 1, we have the height of the tree is 4 and L' = (1 2 3 4 5 2 2 2 2 2). There are 542 rooted trees with 10 vertices and height at least 4 [12]. To determine the position of L among the trees of height 4, we produce the following intermediate level sequences and jumps.

Level Sequence	*Jump*
$L_0 = (1234534444)$	$S'(4,4) = 35, J = 34, M = 34$
$L_1 = (1234532222)$	$SuccessorFunction, J = 1, M = 35$
$L_2 = (1234523452)$	$SuccessorFunction, J = 1, M = 36$
$L_3 = (1234523444)$	$S(2,4) = 8, J = 7, M = 43$
$L_4 = (1234523422)$	$SuccessorFunction, J = 1, M = 44$
$L_5 = (1234523333)$	$S'(4,3) = 7, J = 6, M = 50$
$L_6 = (1234522222) = L'$	

Thus, the rank of L = 542 - 50 = 492.

The complexities of the ranking and unranking algorithms can be measured in terms of the number of jumps required to traverse the trees of the given order or given order and height. This is extremely difficult to determine precisely since the number of jumps required to traverse a particular set of trees depends on the structure of the starting and ending points and the intermediate trees reached by the jumps. Clearly, the method presented in this paper is an improvement over the Beyer and Hedetniemi sequential generation of all trees to reach the tree of the desired rank but it is possible that the complexity of this method is exponential in n, the order of the trees under consideration. However, the data in Tables 2 and 3 shows that the ratio of the number of jumps required to the total number of trees under consideration decreases as n, the order of the trees, increases. Table 2 contains data for traversing all trees of the given order. Table 3 contains data for traversing trees of a given order and the height with the greatest number of trees.

Number of Trees	Average Number of Jumps per Tree
10	.119
11	.010
12	.088
13	.078
14	.071
15	.065
16	.060
17	.055
18	.052
19	.048
20	.046

Table 2: Average Number of Jumps per Tree Generated

Number of Trees	Height	Average Number of Jumps per Tree
10	4	.111
11	4	.119
12	5	.060
13	5	.070
14	5	.077
15	6	.040
16	6	.046
17	6	.051
18	6	.056
19	6	.060
20	7	.035

Table 3: Average Number of Jumps per Tree Generated by Height

References

[1] T. Beyer and S. M. Hedetniemi, Constant time generation of rooted trees,*SIAM J. Comput.* **9** (1980) 706–712.

[2] U. Gupta and D. T. Lee, Ranking and unranking B-trees, *J. Algorithms* **4** (1983) 51–60.

[3] U. Gupta, D. T. Lee and C. K. Wong, Ranking and unranking 2-3 trees, *SIAM J. Computing* **11** (1982) 582–590.

[4] F. Harary and E. M. Palmer, *Graphical Enumeration*, Academic Press, New York, 1973.

[5] D. A. Klarner, Correspondences between plane trees and binary sequences, *J. Comb. Theory Ser. B* **9** (1970) 401–411.

[6] E. Kubicka, The chromatic sum and efficient tree algorithms, Ph. D. Dissertation, Western Michigan University, 1989.

[7] E. Kubicka, An efficient method for examining all trees, *J. Combinatorics, Probability and Computing* **5** (1996) 403–413.

[8] E. Kubicka and G. Kubicki, Constant time algorithm for generating binary rooted trees, *Congressus Numerantium* **90** (1992) 57–64.

[9] G. Li and F. Ruskey, Fast algorithms for generating rooted and free trees, *10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (1999) S939–940.

[10] L. Li, Ranking and unranking AVL-trees, *SIAM J. Computing* **15** (1986) 1025–1035.

[11] K. A. McKeon and A. J. Schwenk, The advantage of the lexicographically maximum level sequence code for caterpillars, *Graph Theory, Combinatorics and Applications: Proceedings of the Seventh Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms, and Applications*, (Y. Alavi et al., eds) Wiley, New York (1995) 777–787.

[12] J. Riordan, The enumeration of trees by height and diameter, *IBM J. Res. Develop.* **4** (1960) 473–478.

[13] F. Ruskey, Generating t-ary trees lexicographically, *SIAM J. Computing* **7** (1978) 424–439.

[14] F. Ruskey, Listing and counting subtrees of a tree, *SIAM J. Computing* **10** (1981) 141–150.

[15] F. Ruskey and T. C. Hu, Generating binary trees lexicogrphically, *SIAM J. Computing* **6** (1977) 745–758.

[16] F. Ruskey and A. Proskurowski, Generating binary trees by transpositions, *J. of Algorithms* **11** (1990) 68–84.

[17] Scions, Placing trees in lexicographic order, *Machine Intelligence* **3** (1960) 43–60.

[18] A.Trojanowski, Ranking and listing algorithms for k-ary trees, *SIAM J. Computing* **7** (1978) 492–509.

[19] V. Vajnovszki, Constant time generation of binary unordered trees, *Bulletin of EATCS* **57** (1995) 221–229.

[20] H. S. Wilf and N. A. Yoshimura, Ranking rooted trees and a graceful application, *Perspectives in Computing, Proc. Japan-U. S. Joint Seminar in Discrete Algorithms and Complexity*, 1987, Academic Press, 341-350.

[21] N. A. Yoshimura, Ranking and unranking algorithms for trees and other combinatorial objects, Ph.D. Dissertation, 1987, University of Pennsylvania.

[22] S. Zaks, Generating k-ary trees lexicographically, University of Illinois Tech. Rept. UICSCS-R77-901: Urbana, IL (1977).

[23] S. Zaks, Lexicographic generation of ordered trees, *Theoret. Comput. Sci.* **10** (1980) 63–82.

[24] S. Zaks and D. Richards, Generating trees and other combinatorial objects lexicographically, *SIAM J. Computing* **8** (1979) 73–81.

A Appendix of Values for S, S', C, C'

k \m	2	3	4	5	6	7	8
3	4	8	17	37	85	200	486
4	8	20	50	124	312	790	2025
5	13	38	107	293	796	2149	5800
6	19	63	196	584	1700	4868	13806
7	26	96	326	1047	3247	9822	29207
8	34	138	507	1743	5732	18254	56789
9	43	190	750	2745	9535	31873	103525
10	53	253	1067	4139	15135	52967	179268
11	64	328	1471	6025	23125	84532	297589
12	76	416	1976	8518	34228	130418	476778
13	89	518	2597	11749	49314	195493	741027

k \m	9	10	11	12	13
3	1205	3047	7813	20299	53272
4	5239	13689	36059	95735	255875
5	15665	42413	115142	313589	856748
6	38906	109240	306064	856720	2397552
7	85782	249734	722379	2079929	5968749
8	173597	523661	1563557	4631702	13635884
9	328904	1027105	3164037	9641431	29122515
10	591092	1909008	6063376	18998764	58869604
11	1016957	3393281	11104229	35759816	113634179
12	1686438	5807833	19566443	64724823	210844713
13	2709719	9622086	33347338	113243704	377999239

Table 4: Values of $S(m, k)$.

k \m	2	3	4	5	6	7	8
3	3	5	7	11	15	22	30
4	7	16	35	78	175	403	949
5	12	33	86	220	558	1417	3616
6	18	57	168	476	1318	3603	9785
7	25	89	290	895	2668	7773	22305
8	33	130	462	1537	4892	15095	45560
9	42	181	695	2474	8358	27190	86020
10	52	243	1001	3791	13532	46244	152919
11	63	317	1393	5587	20993	75136	259077
12	75	404	1885	7976	31449	117583	421884
13	88	505	2492	11088	45754	178303	664465

k \m	9	10	11	12	13
3	42	56	77	101	135
4	2291	5650	14205	36294	94005
5	9298	24107	63030	166126	441115
6	26507	71800	194760	529507	1443566
7	63369	178875	502889	1410527	3951683
8	135329	397319	1156596	3345899	9635290
9	266476	812355	2445981	7294196	21588831
10	492999	1558114	4847381	14891352	45283087
11	867610	2838713	9115318	28825776	90017650
12	1465114	4956067	16409159	53376055	171081121
13	2389326	8345983	28464337	95165860	312892475

Table 5: Values of $S'(m, k)$.

k \m	2	3	4	5	6	7	8
2	2	4	9	20	48	115	286
3	4	9	20	48	115	286	719
4	8	21	54	140	363	949	2495
5	13	39	112	315	875	2416	6651
6	19	64	202	613	1815	5287	15235
7	26	97	333	1084	3407	10447	31475
8	34	139	515	1789	5947	19150	60233
9	43	191	759	2801	9816	33117	108571
10	53	254	1077	4206	15494	54649	186445
11	64	329	1482	6104	23575	86756	307544
12	76	417	1988	8610	34783	133303	490292
13	89	519	2610	11855	49989	199174	759032

k \m	9	10	11	12	13
2	719	1842	4766	12486	32973
3	1842	4766	12486	32973	87811
4	6608	17604	47190	127167	344426
5	18298	50359	138771	383019	1059165
6	43586	124095	352209	997673	2822788
7	93618	275821	806894	2347988	6805135
8	186112	567251	1710595	5115362	15195448
9	348137	1097036	3409249	10476902	31904173
10	619715	2017436	6457994	20389756	63647254
11	1058408	3556580	11720199	38003523	121577131
12	1745071	6047662	20502793	68245527	223679514
13	2790972	9966642	34737986	118635645	398223824

Table 7: Values of $C'(m,k)$.