THE ROTATION GRAPH OF BINARY TREES IS HAMILTONIAN

Joan Lucas

CS-TR-021-86

January, 1986

The Rotation Graph of Binary Trees is Hamiltonian

Joan M. Lucas

Department of Computer Science
Princeton University
Princeton, N.J. 08544

Abstract

In this paper we study the rotation transformation on binary trees and consider the properties of binary trees under this operation. The rotation is the universal primitive used to rebalance binary search trees. It is a simple, local, restructuring technique that alters the depths of some of the nodes in the binary tree. New binary search tree algorithms have recently been introduced in [ST]. It has been conjectured that these algorithms are as efficient as any algorithm that dynamically restructures the tree using rotations. We hope that by studying rotations in binary trees we shall gain a better understanding of the nature of binary search trees, which in turn will lead to a proof of this "dynamic optimality conjecture".

We define a graph, RG(n), whose vertex set consists of all binary trees containing n nodes, and which has an edge between two trees if they differ by only one rotation. We shall introduce a new characterization of the structure of RG(n) and use it to demonstrate the existence of a Hamiltonian cycle in the graph. The proof is constructive and can be used to enumerate all $C_n = \frac{1}{(n+1)}\binom{2n}{n}$ *binary trees with n nodes in* $O(C_n)$ *time, where each tree in the list differs from its predecessor by only one rotation.*

1. Introduction

In this paper we study the rotation transformation on binary trees and consider the properties of binary trees under this operation. The rotation is the universal primitive used to rebalance binary search trees. It is a simple, local, restructuring technique that alters the depths of some of the nodes in the binary tree. New binary search tree algorithms have recently been introduced in [ST]. It has been conjectured that these algorithms are as efficient as *any* algorithm that dynamically restructures the tree using rotations. We hope that by studying rotations in binary trees we shall gain a better understanding of the nature of binary search trees, which in turn will lead to a proof of this "dynamic optimality conjecture".

We define a graph, which we call the Rotation Graph for n node binary trees, or RG(n). The vertex set consists of all binary trees containing n nodes, with an edge between two trees if they differ by only one rotation. We introduce a new characterization of the structure of RG(n) and use it to demonstrate the existence of a Hamiltonian cycle in the graph, i.e a simple cycle that contains each node of RG(n). The proof is constructive and can be used to enumerate all $C_n = \frac{1}{(n+1)}\binom{2n}{n}$ binary trees with n nodes in $O(C_n)$ time, where each tree in the list differs from its predecessor by only one rotation. Thus trees that are close together in the list are also close together structurally. This is analogous to using Gray codes[Kn] to enumerate all n bit binary numbers. In a Gray code, two adjacent numbers differ only at a single bit, whereas in the Hamiltonian cycle, adjacent trees differ by only one rotation. Thus in both enumerations adjacent elements are very similar structurally. In the case of binary trees, however, the graph has a less regular structure and a more complicated induction is required to prove the existence of a Hamiltonian cycle. The Hamiltonian cycle given here is an improvement over previous enumerations of the binary trees in [K,RV,Ze]. In each of these works a binary tree is represented by some n-vector, and the enumeration given is ordered lexicographically with respect to the representation. This lexicographic order does not imply any natural relationship between adjacent trees in the list with respect to the rotation operation. In fact in [RV] and [Ze] as many as $n-1$ rotations may be needed to transform a tree to the next tree on the list.

Section 2 gives some of the background of the problem. In Section 3 we outline the scheme used for coding binary trees, characterize the operations on codes that are equivalent to rotations, and give an intuitive notion of the structure of the Rotation Graph. Section 4 demonstrates the existence of a Hamiltonian path in RG(n), and gives an $O(C_n)$ algorithm for enumerating all binary trees. This section is in some sense redundant, for in Section 5 we show how to construct a Hamiltonian cycle in RG(n), but the algorithm for the Hamiltonian path is extremely simple and elegant, and in practice would certainly be preferred over the cycle algorithm. Section 6 contains conclusions.

2. Binary Trees and Equivalent Representations

The binary search tree is one of the most basic and simple of data structures. It can be used to maintain any sorted set that must be accessed and updated. The items of the set are stored in a rooted binary tree, usually by storing one item at each vertex of the tree. The items are stored in symmetric order; that is, for every vertex x, the item stored at x is greater than any item stored in the left subtree of x, and less than any item stored in the right subtree of x. This arrangement allows one to access any item in the tree by starting at the root and searching down the tree, branching left or right at each node depending on whether the item should lie in the left or right subtree of that node. The time taken to access an item is equal to the depth of the node containing that item, i.e. to the number of nodes on the search path from the root to that node. Insertions and deletions in a binary tree can be done easily; the time needed for either operation is at most the depth of some node in the tree.

To insure fast execution of any series of access, insert and delete operations the binary search tree must remain "balanced". There are many different ways of defining a balanced binary tree [AVL,Kn,M,NR], but the fundamental goal of all such definitions is to guarantee that the depth of every vertex of an n node binary tree is $O(\log(n))$. The shape of a binary search tree changes after an insert or delete operation, therefore any search tree algorithm must guarantee that the new tree is still balanced, or, if this is not the case, restructure the tree so that it becomes balanced. The basic primitive used to restructure the tree in all search tree algorithms is the "rotation". Figure 1 illustrates this operation. The tree T_2 is derived from T_1 by a right rotation of the edge e. Similarly, a left rotation of the edge e in T_2 results in the tree T_1. The rotation is a very useful operation because in constant time it alters the depths of some of the nodes in the tree while maintaining the symmetric order of the items.

In studying the effect of rotations on binary trees it is useful to consider other combinatorial objects that are equivalent to binary trees. The two most well known of these are sets of balanced parentheses and triangulations of a polygon. We shall use the equivalence to triangulations in our study of rotations.

There is a one-to-one correspondence between the set of all binary trees containing n nodes and the set of all possible triangulations of a convex $(n + 2)$-sided polygon. We demonstrate the correspondence by showing how to construct a triangulation given a binary tree and vice-versa (see Figure 2).

Given a binary tree T, we construct the corresponding triangulation as follows. Let e be some distinguished edge on the polygon P, and number the vertices of the polygon from 0 to $n + 1$ counter-clockwise such that the vertices adjacent to e have labels 0 and $n + 1$. The root of T corresponds to the distinguished edge and each non-root node of T corresponds to some diagonal of the triangulation. Let k be the number of nodes in the left subtree of the root of T, and $n - k - 1$ be the number of nodes in the root's right subtree. Add a diagonal from the vertex 0 of the polygon to the vertex $k + 1$ (if $k \neq 0$), and add a diagonal from $n + 1$ to $k + 1$ (if $k \neq n - 1$). These diagonals correspond to the left and right children of the root, respectively. Now recursively construct the triangulations corresponding to each subtree. For the left subtree, construct the triangulation using the polygon formed by the vertices 0 to $k + 1$, and the edges $(i,i + 1)$, $0 \leq i < k + 1$ of P, together with the diagonal from 0 to $k + 1$, which is considered the distinguished edge of the polygon. The right subtree is treated similarly. Its triangulation is built inside the polygon formed by the vertices $k + 1$ to $n + 1$ and the edges $(i,i + 1)$, $k + 1 \leq i < n + 1$, together with the diagonal from $n + 1$ to $k + 1$, where the distinguished edge is taken to be the edge from $n + 1$ to $k + 1$.

We can construct the binary tree corresponding to any given triangulation T' by reversing the process used above. The root of the binary tree is associated with the distinguished edge of the polygon. Consider the triangle of T' that contains the distinguished edge of the polygon. Then the three vertices of P on the triangle are 0, $n + 1$, and $k + 1$, for some k, $0 \leq k \leq n - 1$. The left subtree of the root will contain k nodes and (if $k \neq 0$) can be constructed by building the binary tree corresponding to the triangulation of the polygon formed by all the edges and diagonals of P both of whose endpoints are between 0 and $k + 1$, and whose distinguished edge is the edge from 0 to $k + 1$. We can construct the right subtree of the root similarly using the tri-

angulation formed by the edges and diagonals of P whose endpoints are between $k+1$ and $n+1$. Thus there is a one-to-one correspondence between binary trees with n nodes and triangulations of an $(n+2)$-gon.

The equivalence between binary trees and triangulations of a polygon is of interest in our study of rotations because rotations have a very simple analogue in the triangulation representation. Given any diagonal of a triangulation T_1, we can derive a different triangulation T_2 by "flipping" that diagonal. That is, we remove the diagonal, thus creating a quadrilateral, and then insert a new diagonal that triangulates the quadrilateral in the opposite way (Figure 3). This operation corresponds exactly to a rotation in the binary tree corresponding to T_1, where the edge that is rotated is the one between the node corresponding to the chosen diagonal and its parent. Thus instead of considering rotations in binary trees, we can consider how triangulations are related under the operation of flipping diagonals. This representation is not only conceptually easier to work with, it emphasizes the symmetry inherent in the problem and seems to allow more insight.

We shall use triangulations to define a numerical representation for binary trees that provides a great deal of insight into the structure of RG(n), which we then use to show the existence of a Hamiltonian path and then a Hamiltonian cycle in RG(n). The triangulation construction has already been used in [STT] to show that the diameter of RG(n) is $2n$ - 6 for infinitely many n.

3. Codeword Representation

In [Ze], Zerling presents a numerical representation for binary trees. In his scheme any n node binary tree is represented by an $(n-1)$-tuple, ($x(n-1)$, ..., $x(1)$), called the codeword of the binary tree. He derives these codewords from directly manipulating the binary trees using rotations. We shall derive equivalent codewords using the triangulations of a polygon. This approach is helpful because it simplifies the proof about the analogue of rotations in codewords, and also because we can recognize symmetries that are harder to capture using the binary trees directly.

First we introduce some notation. Let P be a convex polygon with $n+2$ vertices which are

labeled 0 to $n + 1$ in counter-clockwise order. Then $P[i,j]$ denotes the path on P from i to j that travels counter-clockwise around P. The sum $i + j$ indicates the vertex on P reached by traveling j steps in the counter-clockwise direction from the vertex labeled i. The label of the vertex reached is simply $i + j$ *modulo* $(n + 2)$. We now define the codeword corresponding to a triangulation.

Definition: (Codewords)

Let T be some triangulation of a regular polygon P with $n + 2$ sides, then the codeword for T is the following n-tuple:

$\forall\, i, 0 \leq i \leq n-1 : w(i) =$ the number of diagonals from i to points on $P[i+2,n+1]$.

The following lemma characterizes which n-tuples are valid codewords, i.e. actually correspond to a triangulation.

Lemma 1: (Valid Codewords) [Zerling]

A necessary and sufficient condition for an n-tuple w to be a valid codeword is:

$$\forall\, i, 1 \leq i \leq n-1 : w(i) \leq n - i - \sum_{l=i+1}^{n-1} w(l)$$

and

$$w(0) = n - 1 - \sum_{l=1}^{n-1} w(l)$$

Proof:

First we show that the codeword w must satisfy the conditions above. For any i, $1 \leq i \leq n$–1, consider the polygon P' formed by $P[i,n+1]$ plus a diagonal from i to $n + 1$ (see Figure 4a). This polygon has $n + 2 - i$ sides. Let T' be all those diagonals of T that are also diagonals of P'. Clearly the number of diagonals in T' that are not incident on i equals $\sum_{l=i+1}^{n-1} w(l)$. Any diagonal of T incident on i that contributes to the value of $w(i)$ must also be a diagonal of P' or the diagonal from i to $n + 1$. Any m-sided polygon can have at most $m - 3$ diagonals, therefore:

$w(i) \leq$ (maximum possible number of diagonals in P') - $\sum_{l=i+1}^{n-1} w(l) + 1$

$$\therefore\ w(i) \leq n - i - 1 - \sum_{l=i+1}^{n-1} w(l) + 1$$

$$\therefore\ w(i) \leq n - i - \sum_{l=i+1}^{n-1} w(l)$$

Clearly any triangulated $(n+2)$-gon has $n-1$ diagonals. Any diagonal in T not counted in $\sum_{l=1}^{n-1} w(l)$ must be incident to 0, therefore $w(0) = n - 1 - \sum_{l=1}^{n-1} w(l)$.

We now show that given any codeword w that satisfies the conditions of the lemma, we can construct a triangulation T such that w is the codeword corresponding to T. For each entry i, from $n-1$ to 0, add $w(i)$ diagonals from i to the $w(i)$ closest vertices to i on the path $P[i+2,n+1]$, such that none of the new diagonals intersects a diagonal previously placed. By adding the diagonals in this manner, we know that for any diagonal (i,k), $i < k$, the polygon formed by the edges of $P[i,k]$ plus the diagonal (i,k) is fully triangulated. Consider the procedure for vertex i. Let j_1, j_2, ..., j_m be those vertices of P that are greater than i and visible from i, i.e. an edge from i to any j_l would not intersect a diagonal previously placed (see Figure 4b). Number the vertices so that $i < j_1 = i+1 < j_2 < \cdots < j_m = n+1$. Each of the polygons formed by $P[j_l,j_{l+1}]$ plus the diagonal (j_l,j_{l+1}) is fully triangulated, therefore the number of diagonals placed at vertices $n-1$ through $i+1$ equals

$$\sum_{l=i+1}^{n-1} w(l) = \sum_{l=1}^{m-1} (j_{l+1} - j_l + 1 - 3 + 1).$$

The right hand side telescopes to yield

$$\sum_{l=i+1}^{n-1} w(l) = j_m - j_1 - (m-1)$$

$$= (n+1) - (i+1) - (m-1).$$

therefore

$$m - 1 = n - i - \sum_{l=i+1}^{n-1} w(l)$$

Given the upper bound on $w(i)$ we have

$$w(i) \leq m - 1$$

But the number of diagonals that can be placed at i is $m-1$. Therefore it is always possible to place $w(i)$ diagonals at i. Thus entry i satisfies the condition for $1 \leq i \leq n-1$. Note that the $w(i)$ diagonals placed at i *must* go to the closest $w(i)$ visible vertices. If this is not the case, then for the final configuration to be a triangulation there must exist some diagonal (j,k), $i < j < k$, which was not placed by the procedure with the other diagonals of j. But then the number of diagonals from j to vertices on $P[j+2,n+1]$ does not equal $w(j)$, contradicting the fact that the codeword corresponds to the constructed triangulation.

For the final entry, 0, the argument is the same, except that now the number of diagonals that can be placed at 0 is $m-2$, since $(0,n+1)$ is not a diagonal. But this is precisely the value of $w(0)$, therefore the remaining diagonals are placed at 0.

QED

There are $2(n+2)$ different ways to define the codeword corresponding to a given triangulation; each of the $n+2$ vertices of the polygon can serve as the vertex 0, and the polygon can be numbered in either the clockwise or counter-clockwise direction. The set of valid codewords is the same no matter which orientation is chosen. We shall always number the vertices from the left endpoint of the distinguished edge and go counter-clockwise. These are the same codewords as used in [Ze], though he does not include the redundant entry w(0). We include w(0) since it simplies some of the analysis.

We next describe how performing diagonal flips on T affects T's corresponding codeword. Consider two triangulations T_1 and T_2 of an $(n+2)$-gon P such that T_2 differs from T_1 by one diagonal flip. How do the two codewords for T_1 and T_2 differ? Intuitively it seems clear that the two triangulations should have very similar codewords. The following lemma bears this out.

Lemma 2: (Rotations in Codewords)

Let T_1 and T_2 be two triangulations of a convex $(n+2)$-gon, and let their corresponding codewords be w_1 and w_2. Then T_1 and T_2 differ by only one diagonal flip if and only if one of the following conditions holds.

1. For some a and b, $0 \leq a < b \leq n-1$:
   $w_1(i) = w_2(i)$, for all $i \neq a,b$
   $w_2(b) = w_1(b) + 1$ and $w_2(a) = w_1(a) - 1$
   $\sum_{l=k}^{b-1} w_1(l) < b - k$, for all k, $b > k > a$
   $\sum_{l=a}^{b-1} w_1(l) > b - a$ (if $a \neq 0$)
   $\sum_{l=a}^{b-1} w_1(l) \geq b - a$ (if $a = 0$)
2. For some a and b, $0 \leq a < b \leq n-1$:
   $w_1(i) = w_2(i)$, for all $i \neq a,b$
   $w_2(b) = w_1(b) - 1$ and $w_2(a) = w_1(a) + 1$
   $\sum_{l=k}^{b-1} w_1(l) < b - k$, for all k, $b > k > a$
   $\sum_{l=a}^{b-1} w_1(l) \geq b - a$ (if $a \neq 0$)
   $\sum_{l=a}^{b-1} w_1(l) \geq b - a - 1$ (if $a = 0$)

Proof:

Assume that T_1 and T_2 differ by one diagonal flip. Let the flipped diagonal of T_1 be contained in the quadrilateral formed by the vertices a, b, c and d, where $0 \leq a < b < c < d \leq n+1$ (see Figure 3). If T_1 contains the diagonal (a,c), then clearly the codewords for w_1 and w_2 are the same everywhere except at a and b, and $w_2(a)$ and $w_2(b)$ satisfy condition 1 of the lemma. To show that the inequalities must hold, note that $\forall\, k$, $a < k < b$, all the diagonals from a vertex on $P[k,b\text{–}1]$ to some vertex with a greater label must be contained inside the polygon formed by $P[k,b]$ plus an edge from k to b (except possibly a diagonal from k to b). This polygon has $b - k + 1$ sides, therefore:

$$\sum_{l=k}^{b-1} w_1(l) \leq (b - k + 1) - 3 + 1$$

$$\therefore \sum_{l=k}^{b-1} w_1(l) < b - k$$

Since T_1 contains the diagonals (a,b), (a,c) and (a,d), and the polygon formed by $P[a,b]$ plus the diagonal (a,b) must contain $b - a + 1 - 3$ diagonals, we have:

$$\sum_{l=a}^{b-1} w_1(l) \geq (b - a + 1) - 3 + 3$$

$$\therefore \sum_{l=a}^{b-1} w_1(l) > b - a$$

The only exception is the case when $a = 0$, in which case the edge (a,d) may not be a diagonal, but the edge $(0,n+1)$ of the polygon. In either case w_1 and w_2 satisfy condition 1 of the lemma.

If T_1 contains instead the diagonal (b,d), we can use the same arguments to show that w_1 and w_2 satisfy the second set of conditions. The only difference is that the lack of the diagonal (a,c) in T_1 results in the inequality:

$$\sum_{l=a}^{b-1} w_1(l) \geq b - a$$

And again, if $a = 0$, the right hand side of this condition decreases by one.

We now show that if w_1 and w_2 satisfy one of the two conditions of the lemma, then T_1 and T_2 must differ by one rotation. Assume w_1 and w_2 satisfy condition 1. Construct the two triangulations T_1 and T_2 using the method described in the Valid Codeword Lemma. These two triangulations will be identical up to vertex b. T_2 places one more diagonal at b than does T_1. Let $c = \max \{ k \mid k > b \text{ and } (b,k) \text{ is a diagonal in } T_1 \}$, $c = b + 1$ if $w_1(b) = 0$. Define d similarly for vertex b in T_2. At this point the triangulations are as shown in Figure 5. As we continue placing diagonals for vertices k, $a < k < b$, the condition $\sum_{l=k}^{b-1} w_1(l) < b - k$ guarantees that all of the diagonals placed have both endpoints on $P[k,b]$. Up to this point T_1 and T_2 differ only in the fact that T_2 has one more diagonal, (b,d). The condition $\sum_{l=a}^{b-1} w_1(l) > b - a$ guarantees that the diagonals incident to a in T_1 include the diagonals (a,b), (a,c) and (a,d). $w_2(a)$ is one less than $w_1(a)$, but the presence of the diagonal (b,d) causes T_2 to "skip over" c as it places the diagonals for a. Therefore the diagonals incident to a in T_1 and T_2 are the same except that T_1 contains one more diagonal (a,c). The remaining diagonals placed by T_1 and T_2 are identical; therefore T_1 and T_2 differ by one diagonal flip.

Similar arguments can be used to show that if w_1 and w_2 satisfy condition 2, then T_1 and T_2 differ by only one diagonal flip.

Thus the lemma is proved.

QED.

To facilitate the study of the Rotation Graph we introduce some terminology concerning the legal manipulations of codewords. We say that w_1 "pulls" at b using a to reach the codeword w_2 if w_1 and w_2 satisfy condition 1 of the Rotation Lemma. Similarly we say that w_1 "pushes" at b using a to reach the codeword w_2 if w_1 and w_2 satisfy condition 2. We will also speak of a as the "pushing point" or "pulling point" of b, depending on which case applies.

Note that a codeword can always push at i if $w(i) > 0$. This is not true about pulling.

Definition:

Given a valid length n codeword w, we say that j, $0 < j < i$, is a *sticking point* for i in w if

$\sum_{l=k}^{i-1} w(l) < i - k$, for all k, $i > k > j$.

and

$\sum_{l=j}^{i-1} w(l) = i - j$.

If i has a sticking point in w, then w cannot pull at i. Note that i has either a pulling point or a sticking point, but not both, and that in either case the pushing point of i is the same entry. We shall use our terminology loosely, i.e. when we speak of pulling or pushing on a vertex x to reach an adjacent vertex, we actually are referring to the operation being performed on x's corresponding codeword.

If $x = (\ x(n\text{–}1),\ \cdots\ ,\ x(0)\ )$ is a codeword then we define its prefix of length i to be $(\ x(n\text{–}1),\ \cdots\ ,\ x(n\text{–}i)\ )$, and its suffix of length i to be $(\ x(i\text{–}1),\ \cdots\ ,\ x(0)\ )$.

Consider now the structure of the graph RG(n). Each vertex in RG(n) represents a binary tree and hence has a corresponding codeword. We can gain a great deal of insight if we think of the codeword of each node as a point in $n - 1$ dimensions. The last entry in a codeword is redundant, since its value can be deduced from the other $n - 1$ entries. Thus only the first $n - 1$ entries are of interest, and it is these that are taken to be the $(n - 1)$-dimensional vector associated with the codeword. From this point on all codewords specified will not include the redundant $w(0)$ entry.

We now introduce a graph that is a generalized version of RG(n). The graph is called a "stack" for reasons that will become obvious. It is of interest because not only is RG(n) a stack, but many of its subgraphs also fit the definition.

Definition: (Stacks)

A d dimensional *stack* of height h is a graph whose vertex set consists of all those codewords w of length $d+1$ which satisfy the following property:

$$\forall\, i,\ 1 \le i \le d : w(i) \le (d+h-1) - i - \sum_{l=i+1}^{d} w(l)$$

and

$$w(0) = (d+h-2) - \sum_{l=1}^{d} w(l)$$

and which contains an edge between two codewords if they satisfy one of the conditions of the Rotation Lemma.

In other words, an $n-1$ dimensional stack differs from RG(n) only in that the maximum value of an entry now is allowed to depend on h, whereas in RG(n) the value of h is fixed at 2. In the rest of this section we give a more pictorial description of the structure of a stack and describe how stacks arise in RG(n).

Consider any subgraph S of RG(n) induced by the set of nodes in RG(n) whose length $n-1-d$ prefix has some fixed value. This subgraph is a d dimensional stack. We say it is the substack of RG(n) defined by the given prefix. The possible values of $w(d)$ for any node w in S are 0 to $n-d-\sum_{i=d+1}^{n-1} w(i)$. Each possible value of $w(d)$ defines a subgraph of S induced by all those nodes w in S where $w(d)$ is that particular value. These are the $d-1$ dimensional substacks of S, and we say that S is a d dimensional stack of height $n-d-\sum_{i=d+1}^{n-1} w(i)+1$, i.e. the height of S is equal to the number of $d-1$ dimensional substacks in S. Any node w in S in the $d-1$ dimensional substack defined by $w(d)=k$ can only be adjacent to nodes in the same $d-1$ dimensional substack, or to nodes in the two $d-1$ dimensional substacks defined by $w(d)=k+1$ or $w(d)=k-1$. This follows directly from the Rotation Lemma, which states that any two adjacent nodes can differ in the dth entry by no more than 1.

Therefore we can visualize S as a "stack". The bottom of the stack consists of those nodes w such that $w(d)=0$. The next higher element on the stack is the subgraph defined by

$w(d) = 1$. This process of building up S from its subgraphs continues until we reach the top of the stack, which is the subgraph induced by the nodes w such that $w(d)$ is the maximum possible value, $n - d - \sum_{i=d+1}^{n-1} w(i)$. Nodes in the stack can only be adjacent to nodes in the same level, or to nodes in the next higher or next lower levels of the stack. In fact any vertex x, with $x(d) = k$, $k > 0$, is adjacent to exactly one node in the subgraph defined by $w(d) = k - 1$, and this is the node reached by pushing x at d. And any node x, with $x(d) = k$, $k < n - d - \sum_{i=d+1}^{n-1} x(i)$, is adjacent to at most one node in the subgraph defined by $w(d) = k + 1$, and this is the node reached by pulling x at d, if x has a pulling point at d. Note that no two nodes in the same $d - 1$ dimensional substack are adjacent to the same node in the next higher or next lower substacks.

Let X and Y be two different d dimensional substacks of some stack S. We say that X and Y are "adjacent" substacks if there exists an edge between a node in X and a node in Y. If this is the case then the heights of X and Y can differ by at most one.

At times it will be useful to consider the subgraph of a d dimensional stack which consists of a set of consecutive $d - 1$ dimensional substacks. We shall refer to such a graph as a "truncated" stack. A truncated stack is defined by the four parameters (d, h_1, h_2, h_3), and it is the subgraph of the d dimensional stack of height $h_1 + h_2 + h_3$ induced by those nodes w such that $w(d)$ is in the range h_1 to $h_1 + h_2 - 1$. In other words, the truncated stack is formed by discarding the topmost h_3 $d - 1$ dimensional substacks and the bottommost h_1 $d - 1$ dimensional substacks from a normal d dimensional stack of height $h_1 + h_2 + h_3$. The height of a truncated stack is defined in the same way as that of a regular stack, i.e. it is equal to the number of $d - 1$ dimensional substacks, h_2.

Figures 6 to 8 give some examples of stacks; the labels on the nodes give the codeword corresponding to that node, though the redundant $w(0)$ entry is omitted. Figure 6 shows a 1 dimensional stack of height 4. Figure 7 shows a 2 dimensional stack of height 3 and a 2 dimensional stack of height 4. Figure 8 shows a 3 dimensional stack of height 2.

In the figures, we represent stacks by triangles. Substacks of a stack are represented by triangles inside a larger triangle. Two substacks connected by a solid line are adjacent, and two substacks connected by a dotted line represent an unspecified number of substacks of which only the topmost and bottommost are explicitly drawn.

4. Hamiltonian Path

In this section we demonstrate the existence of a Hamiltonian path in RG(n) and give an O(C_n) algorithm to generate all shapes of binary trees. The algorithm described below will generate a Hamiltonian path in any stack. Let S be a d dimensional stack of height h. The algorithm works by considering each of the $d-1$ dimensional substacks of S as a unit. Starting from the topmost substack of S, the algorithm first generates a path that is Hamiltonian in that substack by using induction on d. This path is then extended down to the next lower $d-1$ dimensional substack of S, and all the nodes in that substack are added to the Hamiltonian path of S, again using induction on d. The algorithm proceeds in this manner until all the $d-1$ substacks of S have been traversed, and the Hamiltonian path for S is complete. Alternatively, the algorithm can begin in the bottommost substack of S and work its way up.

If this algorithm is to work we must show how to construct the subpaths in the $d-1$ dimensional substacks so that whenever the algorithm completes the traversal of one substack the last node on the path is adjacent to a node in the next higher substack (if S is being traversed bottom-up) or a node in the next lower substack (if S is being traversed top-down).

Let s be the first node on the Hamiltonian path in S. Then s must lie in either the topmost or bottommost $d-1$ dimensional substack of S. Since the algorithm uses induction in the substacks, s must be in either the topmost or bottommost $d-2$ dimensional substack of the substack defined by the prefix ($s(d)$). Similarly, s must be in either in the topmost or bottommost $d-3$ dimensional substack of the substack defined by the prefix ($s(d)$, $s(d-1)$). Extending this reasoning to all the entries of s it is clear that the nodes of S that are of interest are those identified by the following definition.

Definition: (Extreme Form)

Let w be a codeword of length n. We say that the length i suffix of w ($2 \leq i \leq n$) is in *extreme form* if $\forall\ k,\ 1 \leq k \leq i-1$:

$w(k) = 0$

or

$$w(k) = n - k - \sum_{l=k+1}^{n-1} w(l)$$

and

$$w(0) = n - 1 - \sum_{l=1}^{n-1} w(l)$$

That is, every entry in the suffix is either 0 or its maximum possible value. We say a codeword is in extreme form if its length n suffix is in extreme form.

The key to the Hamiltonian path algorithm lies in the next lemma. The lemma shows that by using extreme form nodes for the starting and ending nodes of the subpath through the $d-1$ dimensional substack, the algorithm will be able to extend this path to either the next higher or the next lower substack.

Lemma 3:

Let w be a valid codeword of length n, such that the length i suffix of w is in extreme form ($2 \leq i < n$). If $w(i) < n - i - \sum_{l=i+1}^{n-1} w(l)$ then w can pull at i, and the length i suffix of the resulting codeword w' is in extreme form. Similarly, if $w(i) > 0$ then w can push at i, and the length i suffix of the resulting codeword w' is in extreme form.

Proof

We prove the case for pulling. Let j be the maximum value such that $j < i$ and $w(j) \neq 0$. This value is guaranteed to exist, since if $w(k) = 0$ for all k, $1 \leq k < i$ then $w(0) > 0$. We claim that j is the pushing point of i in w. Clearly none of the entries k, $i > k > j$ can serve as the pushing point, since $\sum_{l=k}^{i-1} w(l) = 0 < i - k$. The entry $w(j)$ takes on its maximum value, i.e.

$$w(j) = n - j - \sum_{l=j+1}^{n-1} w(l) - \delta,$$

where $\delta = 1$ if $j = 0$, and $\delta = 0$ otherwise. Every entry in w satisfies the inequality of Lemma 1, therefore :

$$w(i) \leq n - i - \sum_{l=i+1}^{n-1} w(l)$$

Putting these together and noting that

$$\sum_{l=j+1}^{i-1} w(l) = 0$$

we get:

$$w(j) \geq n - j - \sum_{l=i+1}^{n-1} w(l) - (n - i - \sum_{l=i+1}^{n-1} w(l)) - \delta$$

$$w(j) \geq i - j - \delta$$

But this is exactly the condition of the Rotation Lemma. Thus j is the pushing point of i in w. To see that w can pull at i note that $w(i) < n - i - \sum_{l=i+1}^{n-1} w(l)$, therefore by the same process used above we get $w(i) > i - j - \delta$, which is precisely the condition of the Rotation Lemma.

It remains to be shown that the length i suffix of w' is in extreme form. Clearly every entry of zero in the suffix remains zero. The maximum value of entry j is one less in w' than in w, but $w'(j) = w(j) - 1$, therefore $w'(j)$ has the maximum possible value. For all non-zero entries k, $k < j$ in w', the maximum value of $w'(k)$ is the same as that of $w(k)$, therefore the length i suffix of w' is in extreme form.

The proof for pushing is similar.

QED

We can now prove the following theorem.

Theorem 1: (Hamiltonian Path)

Given any d dimensional stack S of height h and node $w \in S$ such that the codeword corresponding to w is in extreme form, there exists a Hamiltonian path in S which begins at w and ends at a node in extreme form.

Proof:

The proof is by induction on d. The basis case for $d = 1$ is trivial, since S consists of a simple path and the only extreme form nodes of S are the degree 1 nodes.

Assume the Theorem is true for all dimensions less than d. Let S be a d dimensional stack of height h and let s_0 be the given starting node. Without loss of generality assume that $s_0(d) = 0$. s_0 is also an extreme form node in the bottommost $d-1$ dimensional substack of S. By induction there exists a path P_0 that is Hamiltonian in this substack. This path ends at some node t_0 whose length d suffix is in extreme form. By Lemma 3, t_0 is adjacent to a node s_1 in the next higher substack of S, i.e. the one defined by the prefix (1), and s_1 is an extreme form node in this substack. Again induction on the theorem implies that there exists a path $P_1[s_1,t_1]$ that is Hamiltonian in this substack such that the length d suffix of t_1 is in extreme form. Repeating this procedure for each of the h $d-1$ dimensional substacks of S yields a path that is Hamiltonian in S. The length d suffix of the last node on this path, t_{h-1}, is in extreme form; therefore t_{h-1} is in extreme form, since $t_{h-1}(d)$ takes on its maximum value.

QED

A simple corollary to Theorem 1 is the following.

Corollary:

There exists a Hamiltonian path in RG(n).

In the remainder of this section we demonstrate the existence of an algorithm that generates all "shapes" of n node binary trees in $O(C_n)$ time, such that any two consecutive trees in the list differ by exactly one rotation.

We can generate all valid codewords by implementing the proof of Theorem 1. The only subtleties are the following. When doing a pull or push operation at i we must know the pushing point of i without scanning the codeword entries less than i. And when we are generating the path bottom-up we must know when we have reached the topmost substack, i.e. we must know the maximum possible value for $w(i)$ without scanning the entries greater than i. This is easy to accomplish. At any given point in the algorithm, induction is being implied on an i dimensional substack. The codeword entries from $i+1$ to $n-1$ are fixed, the entry i is moving from either its maximum possible value down to zero or vice-versa, and between each move at i, all possible

values for the length i suffix are generated. For each entry from i to $n-1$ the algorithm maintains its maximum possible value with respect to the current codeword, and for each entry from 1 to i the algorithm keeps the index of its pushing point, i.e. the nearest non-zero entry with smaller index. With this information the algorithm can generate the next codeword in only O(1) time and any necessary updating of the information can also be done in constant time. The algorithm to generate all valid codewords is given at the end of this section.

So far we have described how to generate all valid codewords of length n in $O(C_n)$ time. It is also possible to generate all "shapes" of n node binary trees in $O(C_n)$ time. Given any codeword and a data structure representing the corresponding tree we will show that given any push/pull operation on that codeword, we can locate in O(1) time the corresponding edge of the tree that must be rotated.

We shall assume that the binary tree is represented by n nodes, each node containing a pointer to its parent, left child and right child, and also the labels of the two endpoints of the diagonal corresponding to the node. For each entry i of the codeword, $1 \leq i \leq n-1$, we associate two tree node pointers, $p[i]$ and $q[i]$. If a pull is performed at i, then the tree corresponding to the new codeword can be obtained by rotating the edge between the node pointed to by $p[i]$ and that node's parent. Similarly a push at i corresponds to rotating the edge between the node pointed to by $q[i]$ and its parent in the tree. Note that at any given time, only $n-1$ different rotations are possible, therefore of the $2(n-1)$ pointers only $n-1$ of them are non-null.

It remains to be shown how to update these pointers in constant time after a rotation has been performed. To show how this is done we return to the triangulations of an $(n+2)$-gon. For each entry b we identify the diagonals which correspond to $p[b]$ and $q[b]$. If a push operation is performed at b, then (b,c), where $c = \max \{ l \mid (b,l) \text{ is a diagonal } \}$ is the corresponding diagonal to be flipped. If a pull operation is performed at b, then the corresponding diagonal to be flipped is the diagonal (a,c), where $a = \min \{ l \mid (b,l) \text{ is a diagonal } \}$, $c = \max \{ l \mid (b,l) \text{ is a diagonal } \}$, and the final index d of the quadrilateral that contains the diagonal (a,c) is greater than c. For each b we assign the pointers $p[b]$ and $q[b]$ based on these

observations. Note that $p[b]$ can only point to a node that is a left child and $q[b]$ can only point to a node that is a right child.

Figures 9 and 10 illustrate the only possible pointer reassignments that need be done after a rotation. At most 4 pointers must be changed and all of them can be clearly determined by examining the nodes within one or two steps of the edge being rotated. Thus all possible n node binary trees generated in O(1) time per tree.

The program given below generates all valid codewords and all shapes of binary trees, starting from the tree consisting of only the left path.

```
int         codeword[MAX_N];    /* codeword representing the binary tree */
int         direction[MAX_N];   /*direction[i] indicates whether entry i is running up or down*/
int         push_point[MAX_N]; /*push_point[i] is the index of the pushing point of i*/
int         max_value[MAX_N];  /*max possible value in ith position for current codeword */
tree_node   tree_node[MAX_N];  /*structures representing binary tree nodes */
                               /*each node contains the pointers parent, */
                               /*left_child, right_child, and the endpoints */
                               /*of its diagonal in diagonal[1] and diagonal[2] */
int         n;                   /*number of nodes in binary tree */
int         a,b,c,d,f;
tree_node   x,y,z;

main()
{
 n = number of nodes in binary tree;
 initialize();
 generate_all_trees(n-1);
 }

/*generate all valid codewords whose length ( n - position - 1 ) prefix is fixed */
generate_all_trees(position)
int    position;
{
 int i;

 if ( position = 0 )
   return;

 if ( position = n - 1 )
   max_value[position] = 1;
 else if ( position ≠ 0 )
   max_value[position] = max_value[position+1] + 1 - codeword[position+1];

 if  ( codeword[position] = 0 )
   direction[position] = UP;
 else
   direction[position] = DOWN;
```

```
generate_all_trees(position-1);
for ( i=0; i < max_value[position]; i = i + 1 )
 { if ( direction[position] = UP )
     pull(position,push_point[position]);
   else
     push(position,push_point[position]);
   generate_all_trees(position-1);
   }

if ( position ≠ n-1 )
 { if ( direction[position] = UP )
     push_point[position+1] = position;
   else
     push_point[position+1] = push_point[position];
   }
 }

/*PUSH at i using j */
push(i,j)
int i,j;
{
 codeword[i] = codeword[i] - 1;
 codeword[j] = codeword[j] + 1;
 left_rotate(q[i]);
 }

/*PULL at i using j */
pull(i,j)
int i,j;
{
 codeword[i] = codeword[i] + 1;
 codeword[j] = codeword[j] - 1;
 right_rotate(p[i]);
 }

/*initialize the tree to consist of only the left path */
initialize()
{
 int j;

 for ( j=n-1; j ≥ 1; j=j-1 )
  { codeword[j] = 0;
    push_point[j] = 0;
    }
 codeword[0] = n - 1;

 root = tree_node[1];
 root.diagonal[1,2] = [0,n+1];

 /*initialize the left_child/right_child/parent pointers so that */
 /*the tree to consist of only the left path */

 for ( j=2; j ≤ n; j = j+1 )
  { tree_node[j].diagonal[1,2] = [0,n+2-j];
```

```
    p[j] = pointer to tree_node[n+1-j];
    q[j] = null;
    }
 }

right_rotate(x)
{
 z = parent(x);
 y = right_child(x);

 a = z.diagonal[1];
 c = x.diagonal[2];
 if ( y ≠ null )
  { b = y.diagonal[1];
    if ( right_child(y) ≠ null )
      f = right_child(y).diagonal[1];
    else
      f = c - 1;
    }
 else
   b = c - 1;
 d = z.diagonal[2];

 x.diagonal = [a,d];
 z.diagonal = [b,d];

 /*update the parent/left_child/right_child/root/x/z pointers for the rotation */

 /*reset the p/q pointers */
   if ( p[c] = z )
    { p[c] = null;
      p[b] = x;
      }
    else
     { p[b] = null;
       if ( q[a] = z )
         q[a] = x;
       }
    q[b] = z;
    if ( y ≠ null )
      p[f] = y;
 }

 left_rotate(z)
 {
  x = parent(z);
  y = left_child(z);

  a = x.diagonal[1];
  b = z.diagonal[1];
  if ( y ≠ null )
   { c = y.diagonal[2];
     if ( right_child(y) ≠ null )
       f = right_child(y).diagonal[1];
```

```
  else
    f = c - 1;
  }
else
  c = b + 1;
d = x.diagonal[2];

x.diagonal = [a,c];
z.diagonal = [a,d];

/*update the parent/left_child/right_child/root/x/z pointers for the rotation */

/*reset the p/q pointers */
  if ( y ≠ null )
   { q[b] = y;
     p[f] = null;
     }
  else
    q[b] = null;

  if ( p[b] = x )
    p[c] = z;
  else if ( q[a] = x )
    q[a] = z;
  p[b] = x;
}
```

5. Hamiltonian Cycle

In this section we demonstrate the existence of a Hamiltonian cycle in RG(n). To build a Hamiltonian cycle, we shall find it advantageous to first show the existence of several different types of Hamiltonian paths in any d dimensional stack of height h (possibly truncated). These Hamiltonian paths always will start and end at certain "critical" nodes. The proofs are all by induction, and the induction will always be on the dimension d, the height h, or both.

One dimensional stacks are not interesting. They only consist of a simple path, so finding a Hamiltonian path is trivial and there is no choice available. Once d is at least 2, a great deal of flexibility is introduced into the stack. The basis of our induction will be the 2 and 3 dimensional stacks. First we identify some useful Hamiltonian paths in 2 dimensional stacks. From the examples in Figure 7, it is clear that any 2D stack has the shape of a pentagon. We define the "corner" nodes of a 2D stack of height h to be the nodes with the codewords (0,0), (0,1), (0,h), (h-1,0) and (h-1,1) (see Figure 11). All our Hamiltonian paths in a 2D stack will begin and end at one of these corner nodes. These nodes are of interest because if any two 2D substacks are

adjacent in some larger stack, then each type of corner node in the first substack is adjacent to the same type of corner node in the other substack. This allows us to string together paths that are Hamiltonian in the 2D substacks to create a Hamiltonian path for the entire stack.

Lemma 4: (Hamiltonian Paths in 2D)

Let S be a 2 dimensional stack of height h. Then when the condition on h given below is met, there exists a Hamiltonian path in S between each of the listed pairs of corner nodes. These Hamiltonian paths are summarized in Figures 20 and 21.

h odd : (0,0) to $(h-1,1)$, (0,0) to $(0,h)$, (0,1) to $(h-1,0)$, (0,1) to $(0,h)$, $(0,h)$ to $(h-1,1)$ and $(0,h)$ to $(h-1,0)$.

h odd and $h > 3$: (0,1) to $(h-1,1)$.

h even : (0,0) to $(h-1,0)$, (0,0) to (0,1), (0,1) to $(0,h)$, $(0,h)$ to $(h-1,1)$ and $(h-1,1)$ to $(h-1,0)$.

h even and $h > 2$: (0,0) to $(h-1,1)$ and (0,1) to $(h-1,0)$.

Proof:

The proof is "by picture". For each case we show the existence of the Hamiltonian path by using induction. The figures should be interpreted as follows. The solid lines indicate the Hamiltonian path. For the induction step we use pentagons outlined by a dashed line to indicate a 2 dimensional stack contained inside a large 2 dimensional stack. A plus sign or minus sign inside the dashed pentagon indicates that the height of the pentagon is even or odd, respectively. An arrow drawn inside the dashed pentagon indicates a Hamiltonian path between those two corner nodes of the smaller stack. Figures 12 through 19 demonstrate the Hamiltonian path for each non-symmetric case.

QED

Consider now stacks of higher dimension. First we identify the critical nodes of a d dimensional stack. All the Hamiltonian paths in stacks of dimension 3 or more that we discuss will begin and end at one of these nodes.

Definition: (Critical Nodes)

Let S be the stack defined by the four-tuple (d, h_1, h_2, h_3). The three critical nodes of S are :

BB : $w(d) = h_1$, $w(d\text{-}1) = 0$, and $w(i) = 0 \; \forall \; i$, $d - 2 \geq i \geq 1$

BT : $w(d) = h_1$, $w(d\text{-}1) = h_2 + h_3$, and $w(i) = 0 \; \forall \; i$, $d - 2 \geq i \geq 1$

TB : $w(d) = h_1 + h_2 - 1$, $w(d\text{-}1) = 0$, and $w(i) = 0 \; \forall \; i$, $d - 2 \geq i \geq 1$

The two letters used to name each critical node indicate its position in S. For example, BT lies in the bottommost $d - 1$ dimensional substack of S, and in that substack it lies in the topmost $d - 2$ dimensional substack.

Note that these nodes are simply three extreme form nodes of S. There are many other sets of nodes that could have served our purpose. The only important criteria is that if two d dimensional stacks are adjacent in G then every critical node of the first stack is adjacent to the same type of critical node in the second stack.

The basis of the main induction are the 2 and 3 dimensional stacks, therefore we first show how to find Hamiltonian paths in 3 dimensional stacks.

Theorem 2: (Hamiltonian Paths for 3D)

Given any stack S defined by (3, h_1, h_2, h_3), there exists a Hamiltonian path in S between any pair of critical nodes.

Proof:

The proof is by induction on h_2. Let S_i be the 2D substack of S induced by those nodes w, such that w(3) = i. S(3, h_1 + i, h_2 - i - j, h_3+j) denotes the 3 dimensional substack of S obtained by deleting the bottommost i and the topmost j 2D substacks of S. The desired paths are built by piecing together Hamiltonian paths in the 2D substacks of S, which are guaranteed to exist by Lemma 4. We divide the proof into 6 cases, depending of the starting and ending vertices of the path and the parity of the height of the bottom 2D substack of S. We give a Hamiltonian path for each case in turn. In each case we specify the path using the following notation. S_k[x,y] indicates a path from node x to node y in S_k. This path will be Hamiltonian in S_k unless indicted otherwise by the accompanying figure. S(3, h_1 + i, h_2 - i - j, h_3+j)[x,y] will indicate a Hamiltonian

path through this 3D substack of S between the nodes x and y. We use the words pull up and push down to indicate where the transitions between two 2D stacks occur.

First consider the case when $h_2 = 2$. If the height of the bottom 2D substack of S is odd then the Hamiltonian paths can be constructed as follows.

1. BB to TB :

   if $h_3 = 0$: see Figure 22

   if $h_3 > 0$: S_{h_1}[BB,x_1] pull up S_{h_1+1}[x_2,TB] (see Figure 23)

2. BT to TB :

   S_{h_1}[BT,x_3] pull up S_{h_1+1}[x_4,TB] (see Figure 23)

3. BB to BT :

   S_{h_1}[BB,x_3] pull up S_{h_1+1}[x_4,x_6] push down S_{h_1}[x_5,BT] (see Figures 23 and 25)

If the height of the bottom substack is even then the Hamiltonian paths can be constructed as follows.

1. BB to TB :

   S_{h_1}[BB,x_1] pull up S_{h_1+1}[x_2,TB] (see Figure 23)

2. BT to TB :

   S_{h_1}[BT,x_1] pull up S_{h_1+1}[x_2,TB] (see Figure 23)

3. BB to BT :

   S_{h_1}[BB,x_5] pull up S_{h_1+1}[x_6,x_2] push down (x_1,BT) (see Figures 23 and 26)

Now consider the case when $h_2 > 2$. If the height of the bottom substack is odd then the Hamiltonian paths can be constructed as follows.

1. BB to TB :

   if $h_2 = 3$: S_{h_1}[BB,x_1] pull up S_{h_1+1}[x_6,x_8] pull up S_{h_1+2}[x_{13},x_{15}] (see Figure 24)

   if $h_2 > 3$: S_{h_1}[BB,x_1] pull up S_{h_1+1}[x_6,x_{10}] pull up S(3,h_1 + 2,h_2 - 2,h_3)[x_{15},TB] (see Figure 24)

2. BT to TB :

if $h_2 = 3$: S_{h_1}[BT,x_4] pull up S_{h_1+1}[x_9,x_8] pull up S_{h_1+2}[x_{13},x_{15}] (see Figure 24)

if $h_2 > 3$: S_{h_1}[BT,x_4] pull up S_{h_1+1}[x_9,x_5] pull up S(3,h_1 + 2,h_2 - 2,h_3)[x_{11},TB] (see Figure 24)

3. BB to BT :

if $h_2 = 3$: S_{h_1}[BB,x_{16}] pull up S_{h_1+1}[x_{17},x_9] pull up S_{h_1+2}[x_{14},x_{11}] push down S_{h_1+1}(x_5,x_8) push down S_{h_1}(x_3,BT) (see Figure 24)

if $h_2 > 3$: S_{h_1}[BB,x_2] pull up S_{h_1+1}[x_7,x_{10}] pull up S(3,h_1 + 2,h_2 - 2,h_3)[x_{15},x_{11}] push down S_{h_1+1}(x_5,x_8) push down S_{h_1}(x_3,BT) (see Figure 24)

If the height of the bottom substack is even then the Hamiltonian paths can be constructed as follows.

1. BB to TB :

S_{h_1}[BB,BT] pull up S(3,h_1 + 1,h_2 - 1,h_3)[x_5,TB] (see Figure 24)

2. BT to TB :

S_{h_1}[BT,BB] pull up S(3,h_1 + 1,h_2 - 1,h_3)[x_{10},TB] (see Figure 24)

3. BT to TB :

the construction given for the case when the bottom substack has odd height and $h_2 > 3$ works for this case if $h_2 > 3$ or $h_3 > 0$. Figure 27 illustrates a Hamiltonian path for the case when $h_2 = 3$ and $h_3 = 0$.

QED

We now show the existence of these Hamiltonian paths in a general stack.

Theorem 3: (Hamiltonian Paths)

Let S be a stack defined by the 4-tuple (d, h_1, h_2, h_3), d $\geq$ 4. Then there exists a Hamiltonian path in S between each pair of critical nodes.

Proof:

The proof is by induction on d and h_2. The induction hypothesis is that the 3 types of Hamiltonian paths exist in all $d-1$ dimensional stacks (possibly truncated) and that the BB to

TB Hamiltonian path exists in $d-2$ dimensional stacks of height 2. Clearly this is true for the basis case $d = 4$.

We consider each type of Hamiltonian path in turn. For each case we construct the needed path using paths that are Hamiltonian in substacks of S and whose existence is guaranteed by induction. Note that when we specify a particular codeword the 0^{th} entry is not included and the notation 0, ..., 0 means that all entries in the indicated range are zero.

Hamiltonian Path from BB to TB (see Figures 28 and 29)

Consider first the case when $h_2 = 2$. Let $x_1 = (\ h_1,\ h_2 + h_3,\ 0\ .....\ 0\ )$, and let $x_2 = (\ h_1 + 1,\ h_2 + h_3 - 1,\ 0\ .....\ 0\ )$. Then the Hamiltonian path from BB to TB equals:

$$P_1[\mathrm{BB}, x_1]\ (x_1, x_2)\ P_2[x_2, \mathrm{TB}]$$

where P_1 is a Hamiltonian path in the substack of S defined by by the prefix (h_1) and P_2 is a Hamiltonian path in the substack of S defined by the prefix ($h_1 + 1$). The induction hypothesis implies that both P_1 and P_2 exist since they are both BB to TB paths in $d-1$ dimensional stacks.

Consider now the case when $h_2 > 2$. Let x_1 and x_2 be defined as above. Then the Hamiltonian path from BB to TB equals:

$$P_1[\mathrm{BB}, x_1]\ (x_1, x_2)\ P_2[x_2, \mathrm{TB}]$$

where P_1 is a Hamiltonian path in the substack of S defined by the prefix (h_1) and P_2 is a Hamiltonian path in the substack of S induced by those nodes w such that $h_1 + 1 \leq w(d) \leq h_1 + h_2 - 1$. The induction hypothesis implies that P_1 exists since it is a BB to TB path in a $d-1$ dimensional stack, and that P_2 exists since it is a BT to TB path in a d dimensional stack of height $h_2 - 1$.

Hamiltonian Path from BT to TB (see Figures 30 and 31)

Consider first the case when $h_2 = 2$. Let $x_1 = (\ h_1,\ 0,\ h_2 + h_3 + 1,\ 0\ .....\ 0\ )$, and let $x_2 = (\ h_1 + 1,\ 0,\ h_2 + h_3,\ 0\ .....\ 0\ )$. Then the Hamiltonian path from BT to TB equals:

$$P_1[\mathrm{BT},x_1]\ (x_1,x_2)\ P_2[x_2,\mathrm{TB}]$$

where P_1 is a Hamiltonian path in the substack of S defined by the prefix (h_1), and P_2 is a Hamiltonian path in the substack of S defined by the prefix ($h_1 + 1$). P_1 is a BT to TB path in a $d-1$ dimensional stack and P_2 is a BB to BT path in a $d-1$ dimensional stack.

Consider now the case when $h_2 > 2$. Let $x_1 = (\ h_1, 0, 0 \ldots\ldots 0\ )$, and let $x_2 = (\ h_1 + 1, 0, 0 \ldots\ldots 0\ )$. Then the Hamiltonian path from BT to TB equals:

$$P_1[\mathrm{BT},x_1]\ (x_1,x_2)\ P_2[x_2,\mathrm{TB}]$$

where P_1 is a Hamiltonian path in the substack of S defined by the prefix (h_1), and P_2 is a Hamiltonian path in the substack of S induced by those nodes w such that $h_1 + 1 \leq w(d) \leq h_1 + h_2 - 1$. P_1 is a BB to TB path in a $d-1$ dimensional stack and P_2 is a BB to TB path in a d dimensional stack of height $h_2 - 1$.

Hamiltonian Path from BB to BT (see Figures 32, 33 and 34)

Consider first the case when $h_2 = 2$. Identify the following nodes:

$x_1 = (\ h_1, 0, h_3 + 3, 0 \ldots\ldots 0\ )$

$x_2 = (\ h_1 + 1, 0, h_3 + 2, 0 \ldots\ldots 0\ )$

$x_3 = (\ h_1 + 1, h_3, 0 \ldots\ldots 0\ )$

$x_4 = (\ h_1 + 1, h_3 + 1, 0 \ldots\ldots 0\ )$

$x_5 = (\ h_1 + 1, h_3 + 1, 1, 0 \ldots\ldots 0\ )$

$x_6 = (\ h_1, h_3 + 2, 1, 0 \ldots\ldots 0\ )$

Then the Hamiltonian path from BB to BT equals:

$$P_1[\mathrm{BB},x_1]\ (x_1,x_2)\ P_2[x_2,x_3]\ (x_3,x_4)\ P_3[x_4,x_5]\ (x_5,x_6)\ P_4[x_6,\mathrm{BT}]$$

P_1 is a Hamiltonian path in the substack of S induced by those nodes w with $w(d) = h_1$ and $0 \leq w(d\text{-}1) \leq h_3 + 1$; it is a BB to BT path in a $d-1$ dimensional stack. P_2 is a Hamiltonian path in the substack of S induced by those nodes w with $w(d) = h_1 + 1$ and $0 \leq w(d\text{-}1) \leq h_3$; if

$h_3 > 0$, then it is a BT to TB path in a $d-1$ dimensional stack, otherwise it is a BB to TB path in a $d-2$ dimensional stack of height 3. P_3 is a Hamiltonian path in the substack of S defined by the prefix (h_1+1, h_3+1); it is a BB to TB path in a $d-2$ dimensional stack of height 2. P_4 is a Hamiltonian path in the substack of S defined by the prefix (h_1, h_3+2); it is a BB to TB path in a $d-2$ dimensional stack of height 2. All of these subpaths are guaranteed to exist if $d > 4$ or if $d = 4$ and $h_3 > 0$. Figure 35 illustrates a BB to BT Hamiltonian path if $d = 4$ and $h_3 = 0$.

Consider now the case when $h_2 > 2$. Identify the following nodes:

$x_1 = (h_1, 0, h_2 + h_3 + 1, 0 \ldots\ldots 0)$

$x_2 = (h_1 + 1, 0, h_2 + h_3, 0 \ldots\ldots 0)$

$x_3 = (h_1 + 1, 0, 0 \ldots\ldots 0)$

$x_4 = (h_1 + 2, 0, 0 \ldots\ldots 0)$

$x_5 = (h_1 + 2, h_2 + h_3 - 2, 0 \ldots\ldots 0)$

$x_6 = (h_1 + 1, h_2 + h_3 - 1, 0 \ldots\ldots 0)$

$x_7 = (h_1 + 1, h_2 + h_3 - 1, 1, 0 \ldots\ldots 0)$

$x_8 = (h_1, h_2 + h_3, 1, 0 \ldots\ldots 0)$

Then the Hamiltonian path from BB to BT equals:

$$P_1[\mathrm{BB}, x_1]\ (x_1, x_2)\ P_2[x_2, x_3]\ (x_3, x_4)\ P_3[x_4, x_5]\ (x_5, x_6)\ P_4[x_6, x_7]\ (x_7, x_8)\ P_5[x_8, \mathrm{BT}]$$

P_1 is a Hamiltonian path in the substack of S induced by those nodes w with $w(d) = h_1$ and $0 \leq w(d\text{-}1) \leq h_2 + h_3 - 1$; it is a BB to BT path in a $d-1$ dimensional stack. P_2 is a Hamiltonian path in the substack of S induced by those nodes w with $w(d) = h_1 + 1$ and $0 \leq w(d\text{-}1) \leq h_2 + h_3 - 2$; it is a BB to BT path in a $d-1$ dimensional stack. P_3 is a Hamiltonian path in the substack of S induced by those nodes w with $h_1 + 2 \leq w(d) \leq h_1 + h_2 - 1$; if $h_2 = 3$, then it is a BB to TB path in a $d-1$ dimensional stack, otherwise it is a BB to BT path in a d dimensional stack of height $h_2 - 2$. P_4 is a Hamiltonian path in the substack of S defined by the prefix ($h_1 + 1$, $h_2 + h_3 - 1$); it is a BB to TB path in a $d-2$ dimensional stack of height 2. P_5 is a Hamiltonian path in the substack of S defined by the prefix (h_1, $h_2 + h_3$); it is a BB to

TB path in a $d-2$ dimensional stack of height 2.

All of these subpaths are guaranteed to exist, therefore the Theorem is proved.

QED.

Having established these types of Hamiltonian paths, finding a Hamiltonian cycle in RG(n) is trivial.

Theorem 4: (Hamiltonian Cycle in RG(n))

There exists a Hamiltonian cycle in the Rotation Graph for n node binary trees, $n \geq 3$.

Proof:

RG(n) is an $n-1$ dimensional stack of height 2. By Lemma 4, Theorem 2 or Theorem 3, there exists a BB to TB Hamiltonian path in RG(n) between the codewords $w_1 = (\ 1, 0, 0, \ldots\ldots, 0\ )$ and $w_2 = (\ 0, 0, 0, \ldots\ldots, 0\ )$. Clearly w_1 and w_2 are adjacent in RG(n), therefore there exists a Hamiltonian cycle in RG(n).

QED.

The proof of this theorem can be used to write an $O(C_n)$ time algorithm to generate the trees on this cycle. Each push or pull operation takes only constant time since the pushing point of i, $i > 4$, can only be $i-1$, $i-2$, or 0, and the corresponding edge to be rotated can be determined in O(1) time using the techniques of section 4.

6. Conclusions

In this paper we have examined a new representation for binary trees and showed how it sheds light on the structure of RG(n). This new characterization helps elucidate some of the properties of RG(n). By emphasizing the inherent dimensionality of the problem, we are able to break it up into smaller pieces and apply induction. In particular we have been able to show the existence of a Hamiltonian cycle in RG(n).

We are currently investigating whether the characterization of RG(n) presented here is useful in studying other properties of RG(n). In particular we believe that it will lead to a

polynomial time algorithm for finding a shortest path in RG(n) between any two given binary trees. As yet there is no better algorithm for the shortest path than a simple breadth-first search, which can take as long as $O(C_n)$. We also hope that this approach will lead to a simpler proof of the result of [STT] that the diameter of RG(n) is $2n - 6$ for infinitely many n, $n > 13$. The proof of [STT] is quite deep and calls upon results in hyperbolic geometry. We believe that this better understanding of the nature of rotations in binary trees will be of use in further exploring the properties of binary search tree algorithms.

Acknowledgements

I would like to thank Bob Tarjan, Andrea LaPaugh, and Dan Sleator for many stimulating discussions on this subject.

References

[AVL] Adelson-Velskii, G. M., and Landis, E. M., "An Algorithm for the Organization of Information", Sov. Math. Dokl. 3, 1962, 1259-1262.

[Kn] Knuth, D. E., The Art of Computer Programming. Vol. 4, Addison-Wesley, Reading, Mass., 1973.

[K] Knott, G.D., "A Numbering System for Binary Trees", Comm. ACM 2, Feb. 1977, 113-115.

[M] Mehlhorn, K., "Dynamic Binary Search Trees", SIAM J. Comput. 8, 1979, 175-198.

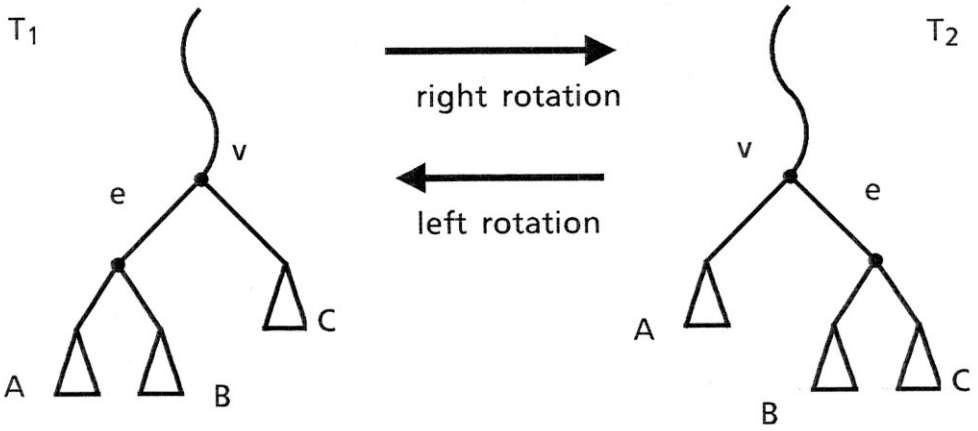
[NR] Nievergelt, J., and Reingold, E. M., "Binary Search Trees of Bounded Balance", SIAM J. Comput. 2, 1973, 33-43.

[RV] Rotem, D., and Varol, Y. L., "Generation of Binary Trees from Ballot Sequences", J. ACM, 25, 3, July, 1978, 396-404.

[ST] Sleator, D. D., and Tarjan, R. E., "Self-Adjusting Binary Search Trees", J. ACM, 32, 3, July, 1985, 652-686.

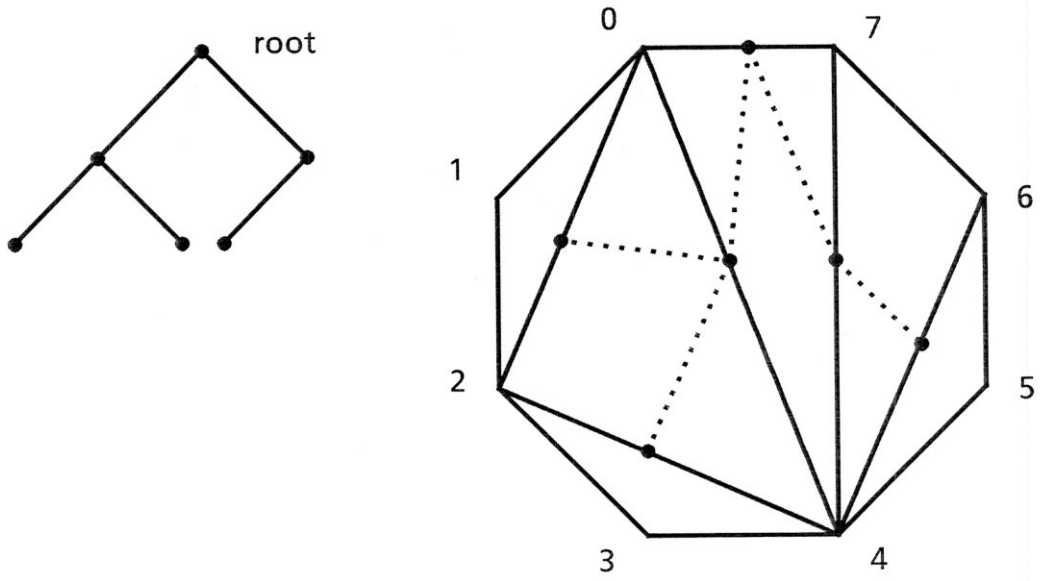
[STT] Sleator, D. D., Tarjan, R. E., and Thurston, W. P., private communication.

[Ze] Zerling, D., "Generating Binary Trees Using Rotations", J. ACM, 32, 3, July, 1985, 694-701.



Rotation of edge e, v is any node in the tree

Figure 1



Example of a binary tree for n = 6 and corresponding triangulation

Figure 2

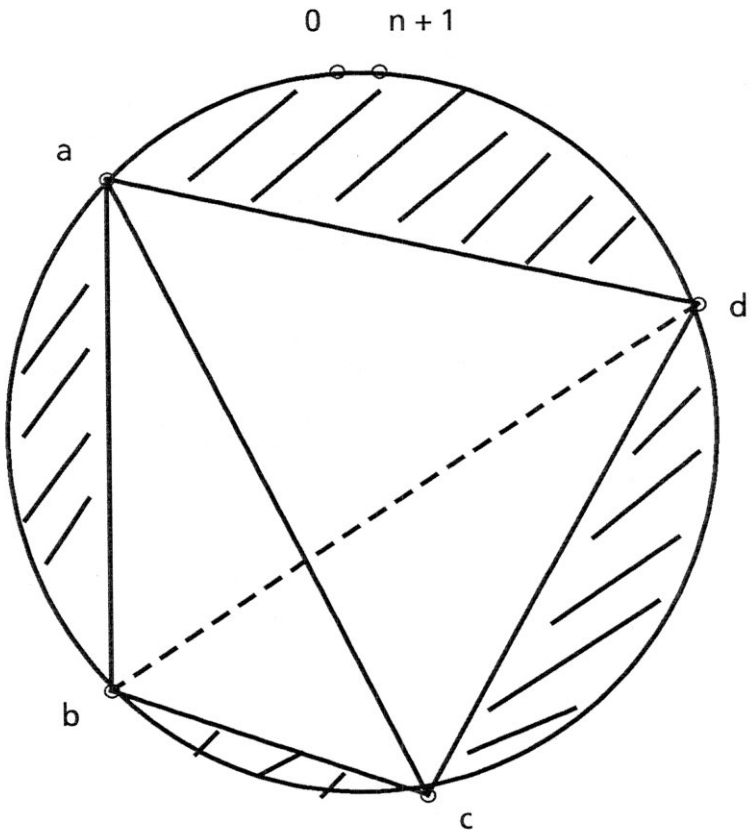


Figure 3

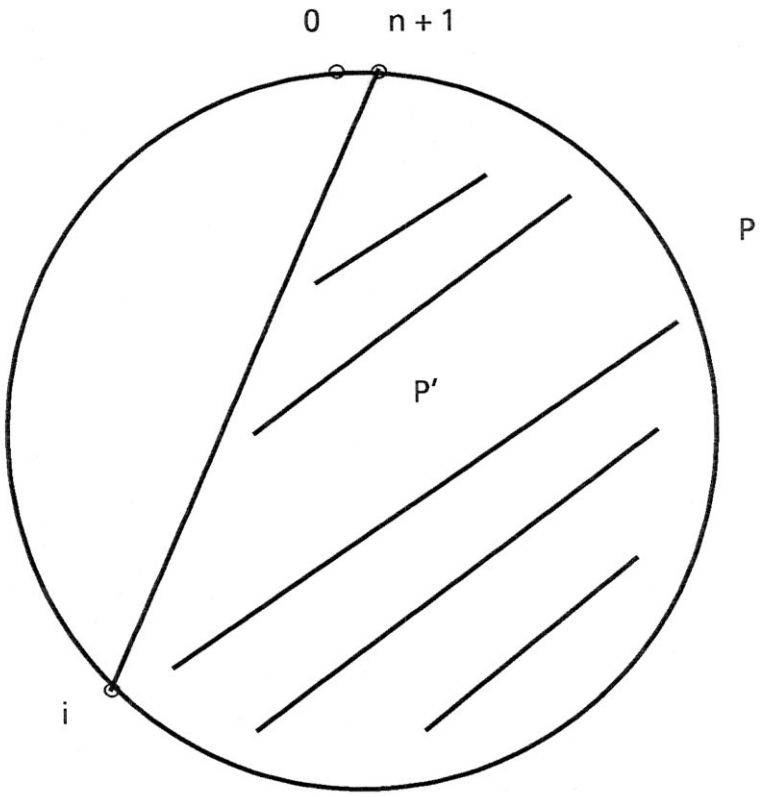


Figure 4a

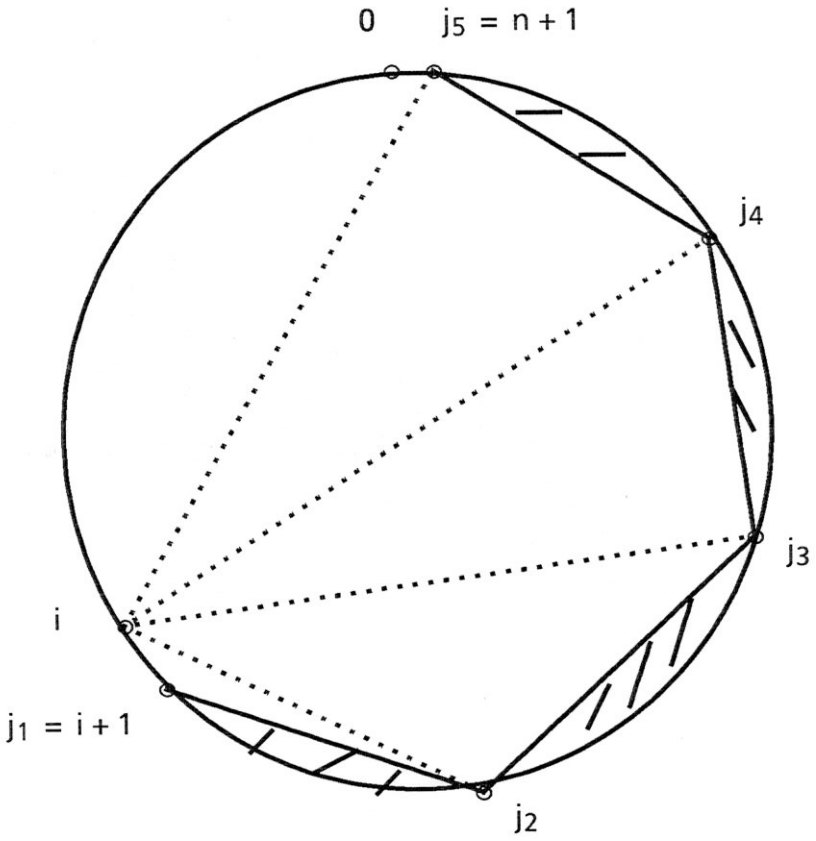


Figure 4b

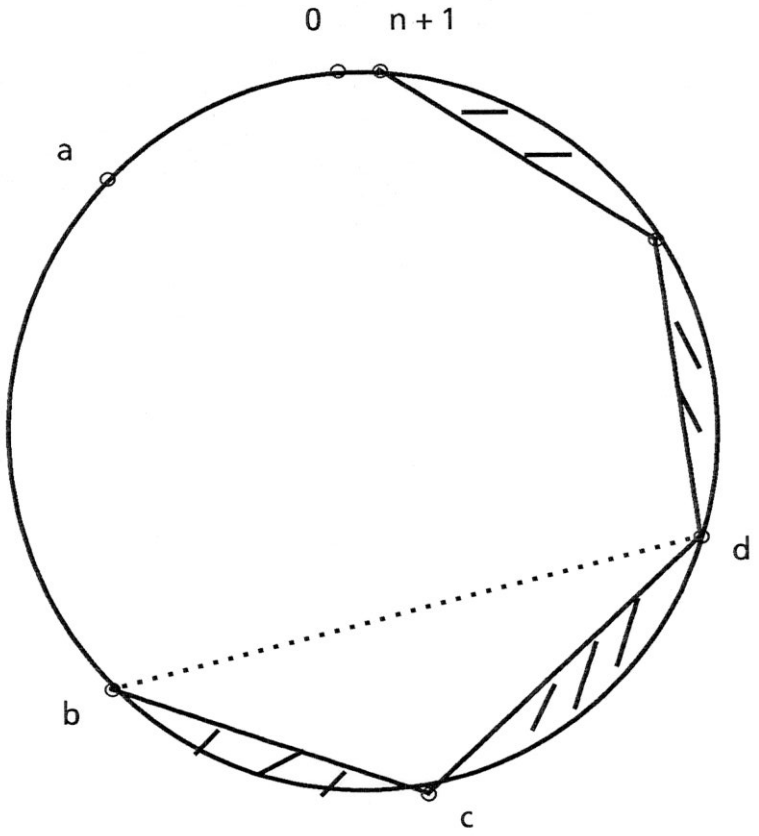
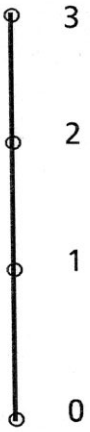
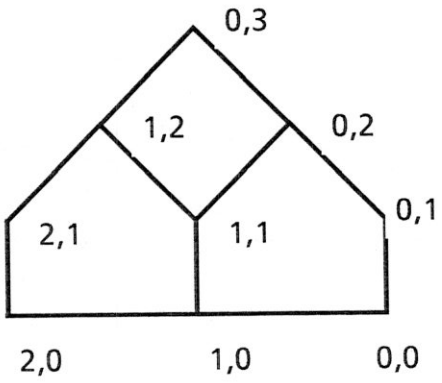


Figure 5

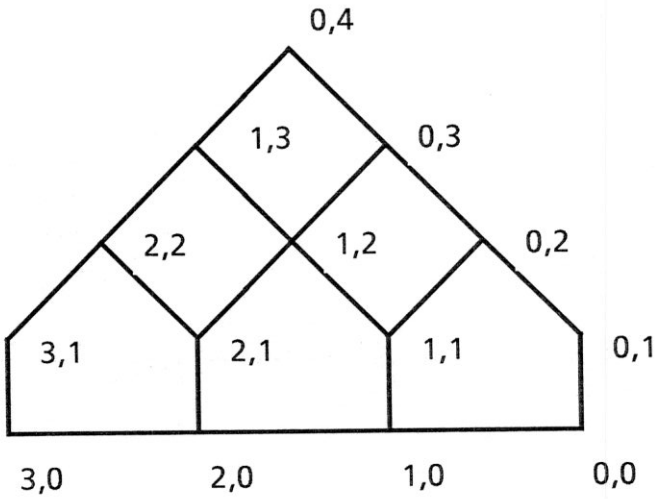


1 dimensional stack of height 4

Figure 6

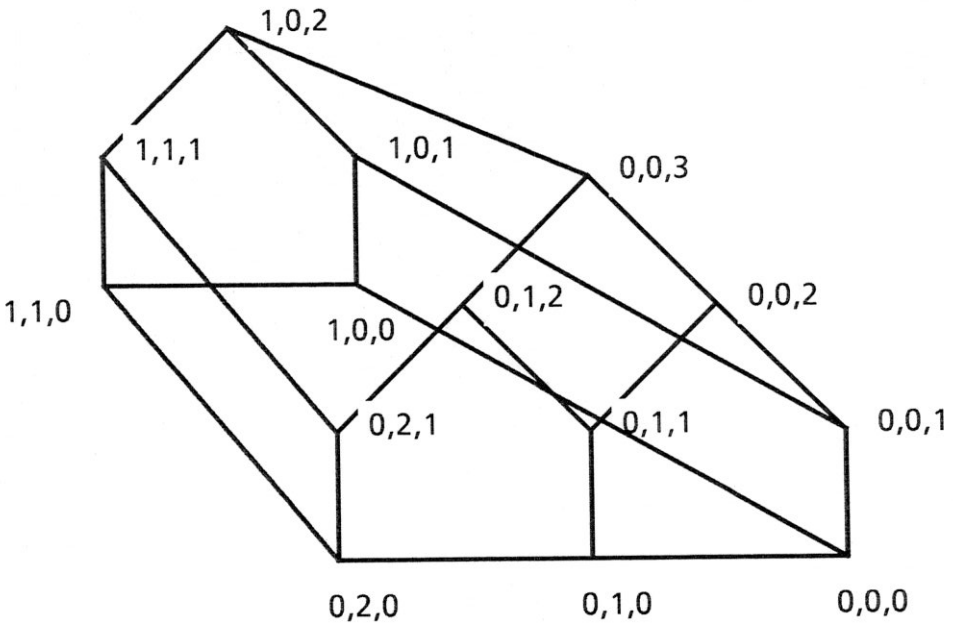


2 dimensional stack of height 3



2 dimensional stack of height 4

Figure 7



3 dimensional stack of height 2

Figure 8

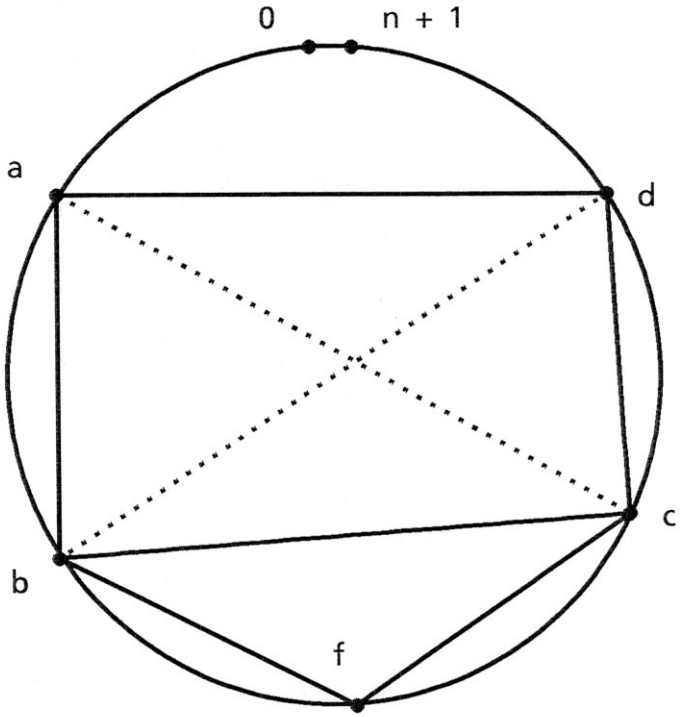


Figure 9

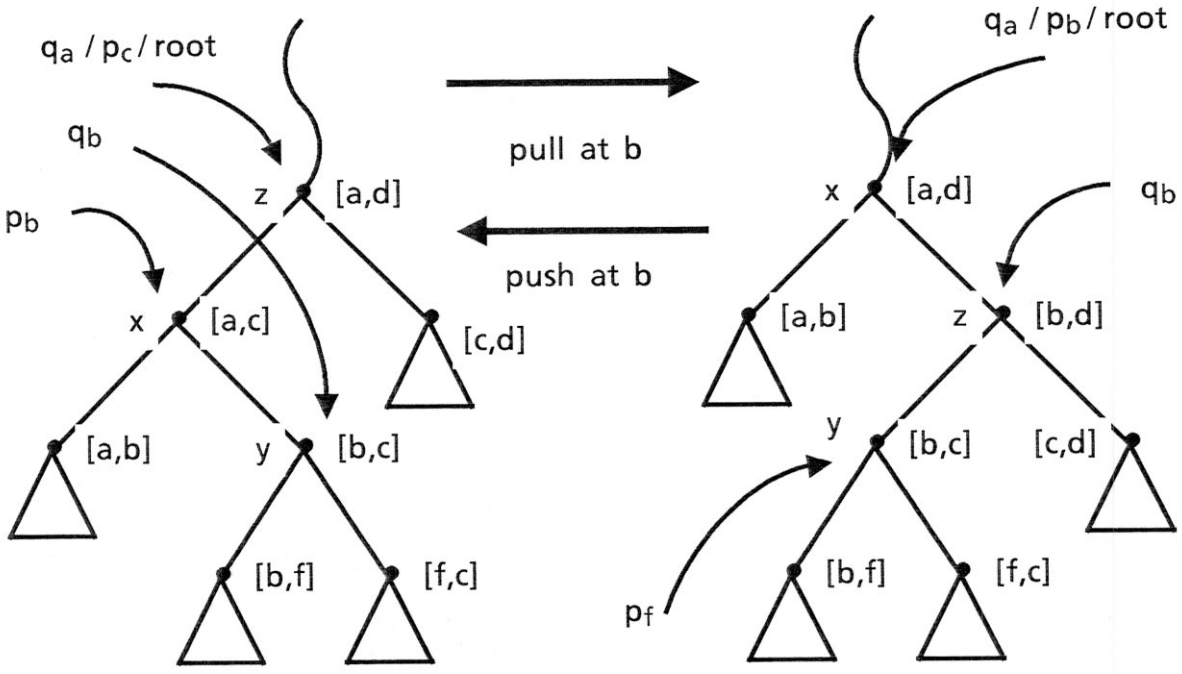
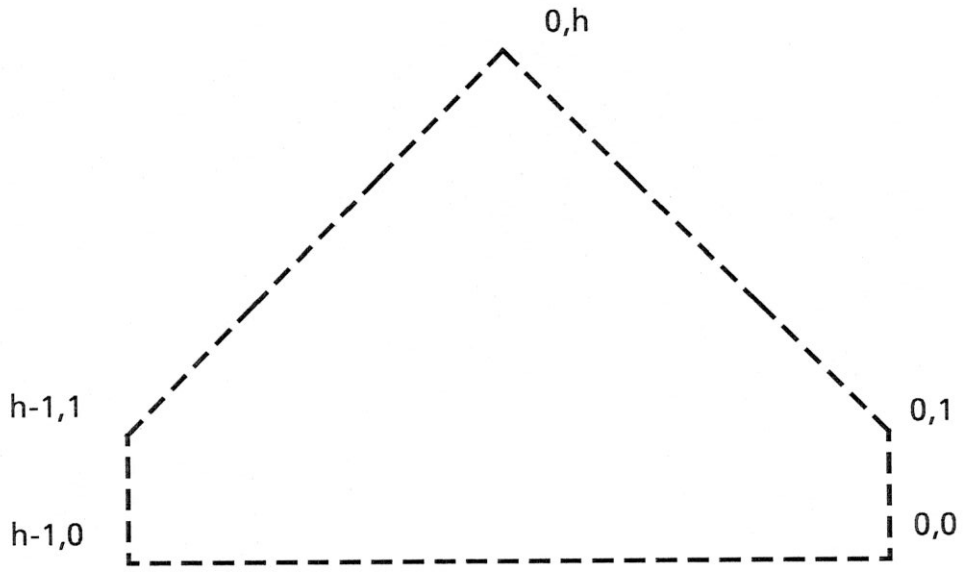


Figure 10



Corner nodes of a 2 dimensional stack of height h

Figure 11

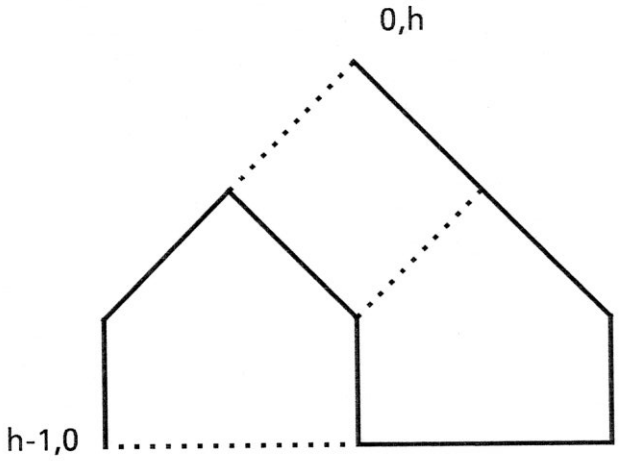


Figure 12a : (h-1,0) to (0,h) for h = 3

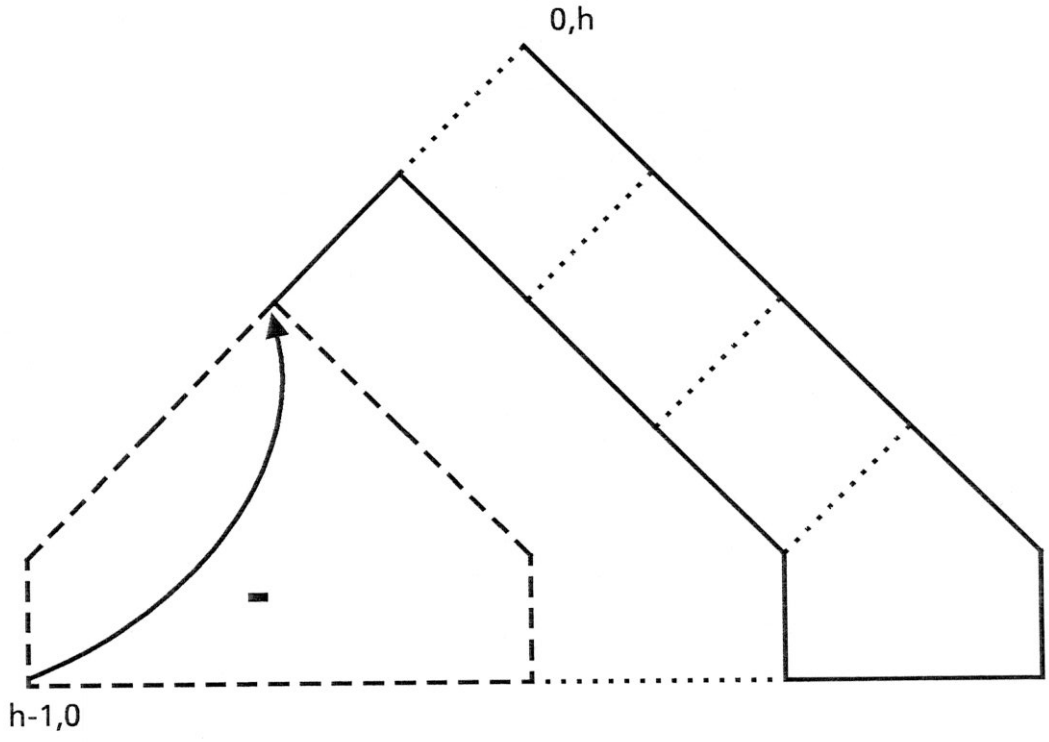


Figure 12b : (h-1,0) to (0,h) for h > 3

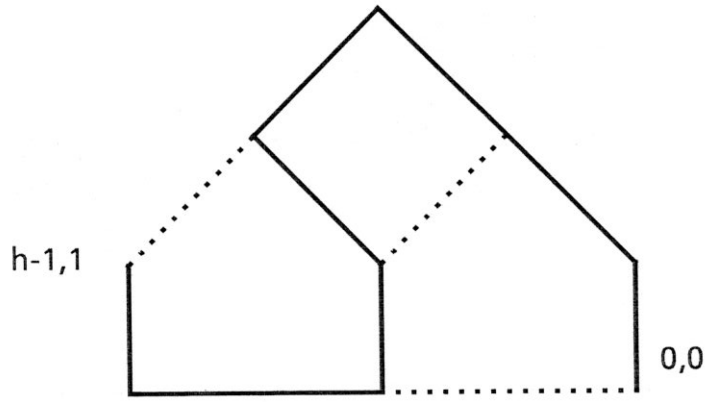


Figure 13a : (h-1,1) to (0,0) for h = 3

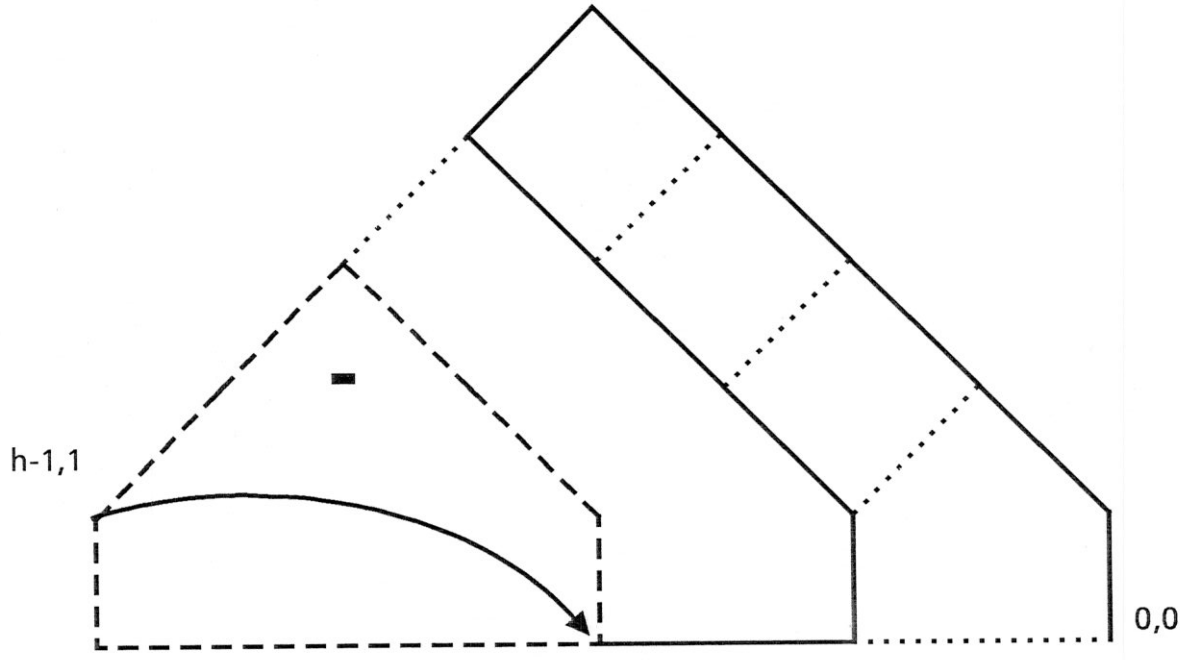


Figure 13b: (h-1,1) to (0,0) for h > 3

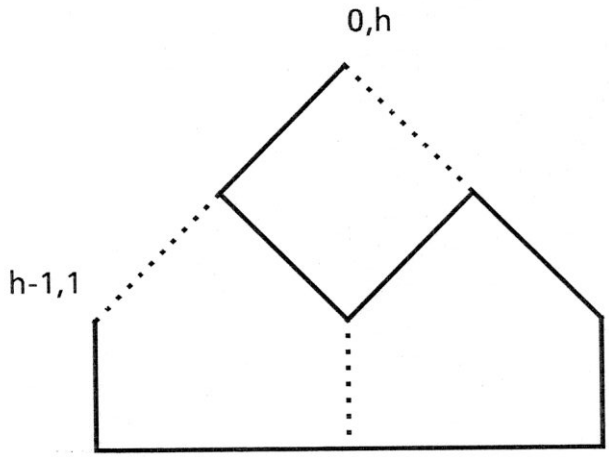


Figure 14a : (h-1,1) to (0,h) for h = 3

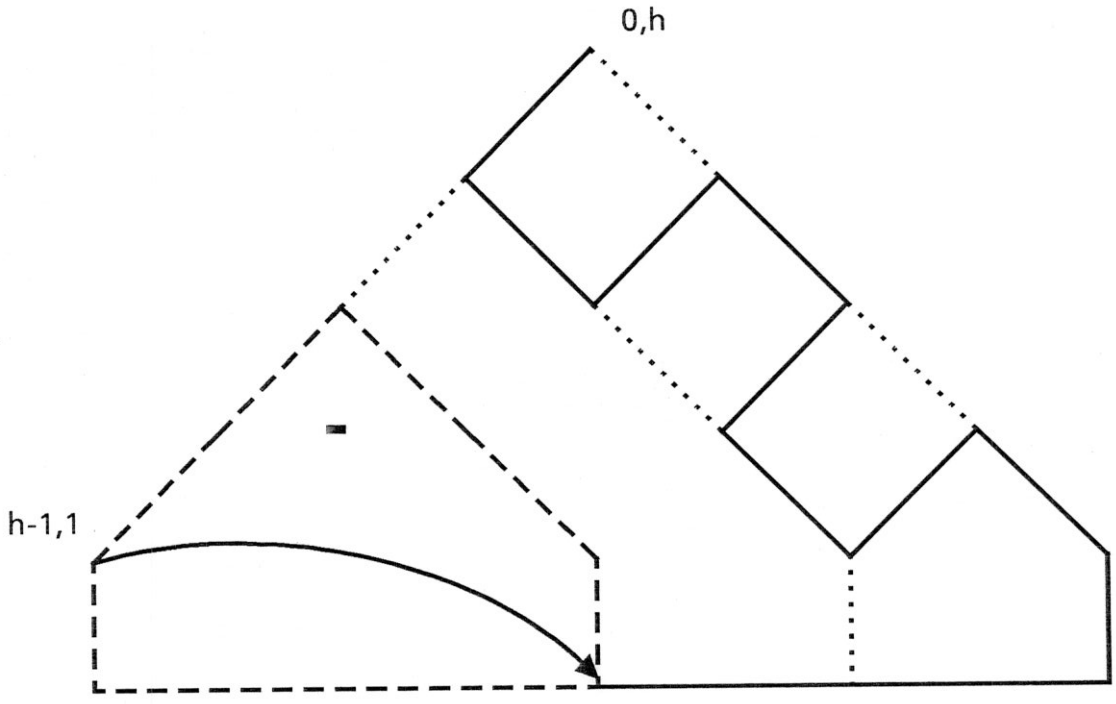


Figure 14b : (h-1,1) to (0,h) for h > 3

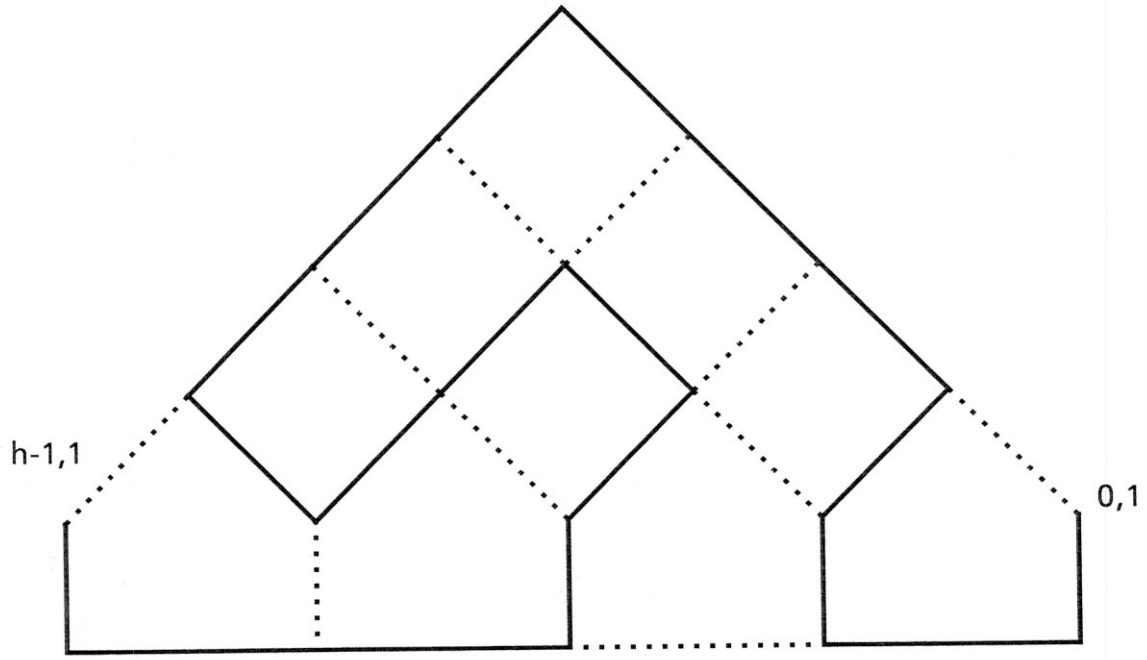


Figure 15a : (h - 1, 1) to (0,1) for h = 5

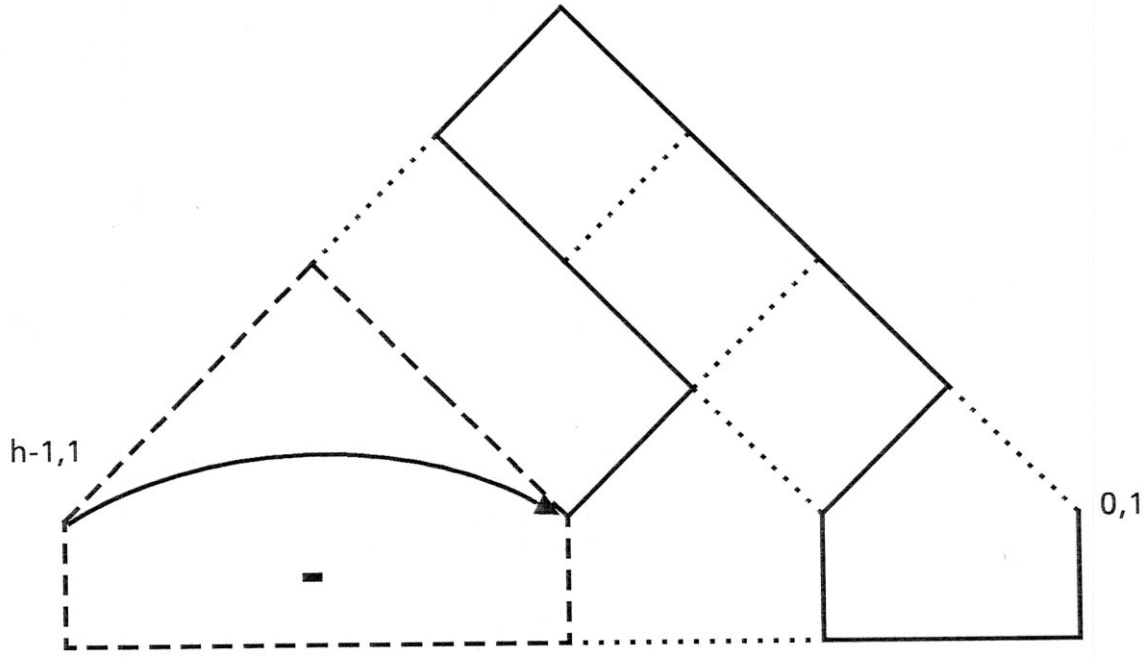


Figure 15b : (h - 1,1) to (0,1) for h > 5

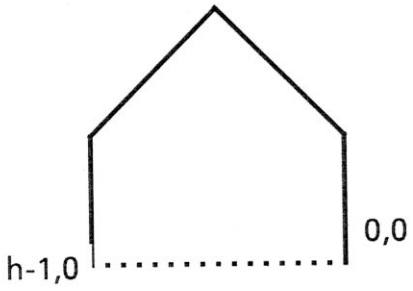


Figure 16a: (h-1,0) to (0,0)
for h = 2

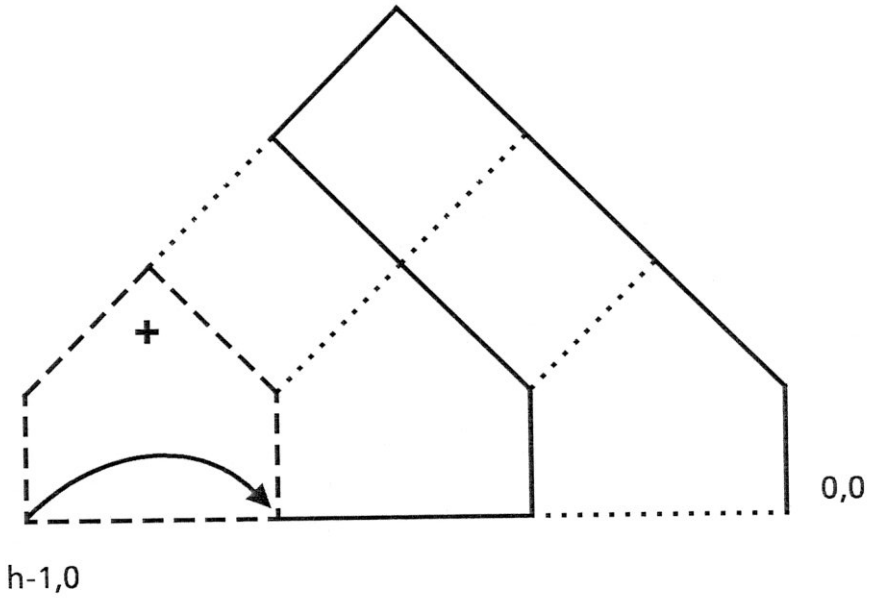


Figure 16b : (h-1,0) to (0,0) for $h > 2$

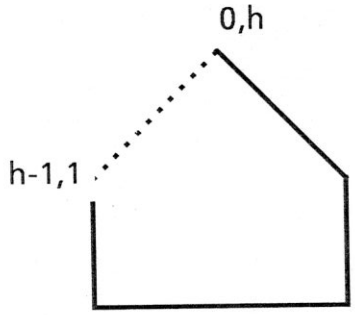


Figure 17a : (h-1,1) to (0,h) for
h = 2

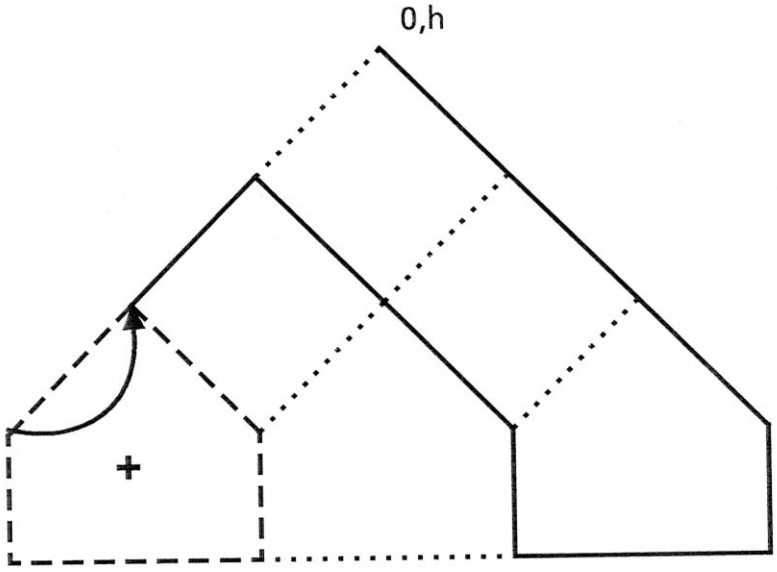


Figure 17b, (h-1,1) to (0,h) for h > 2

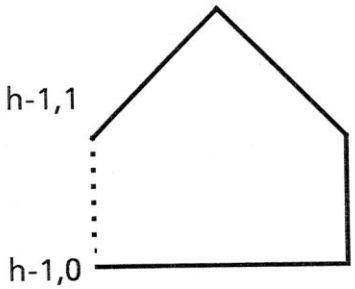


Figure 18a : (h-1,1) to (h-1,0)
for h = 2

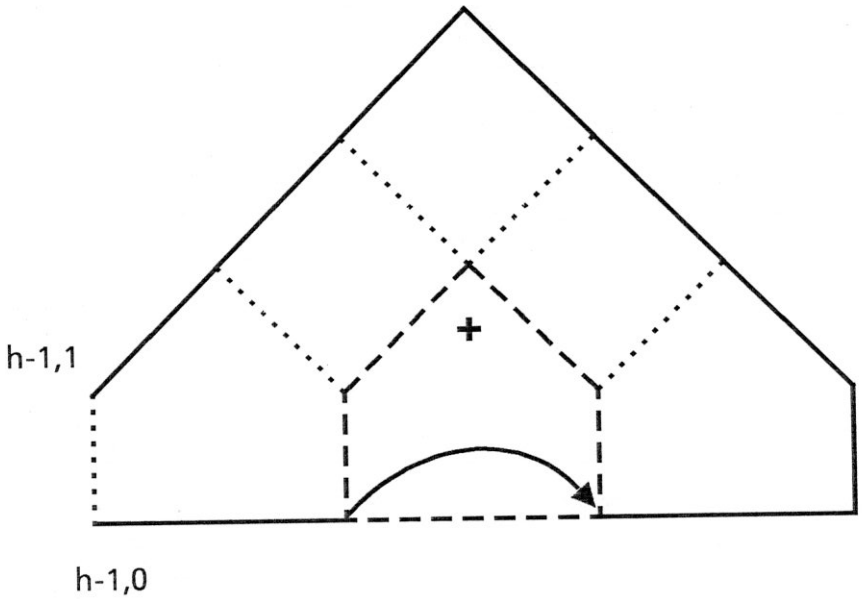


Figure 18b : (h-1,1) to (h-1,0) for h > 2

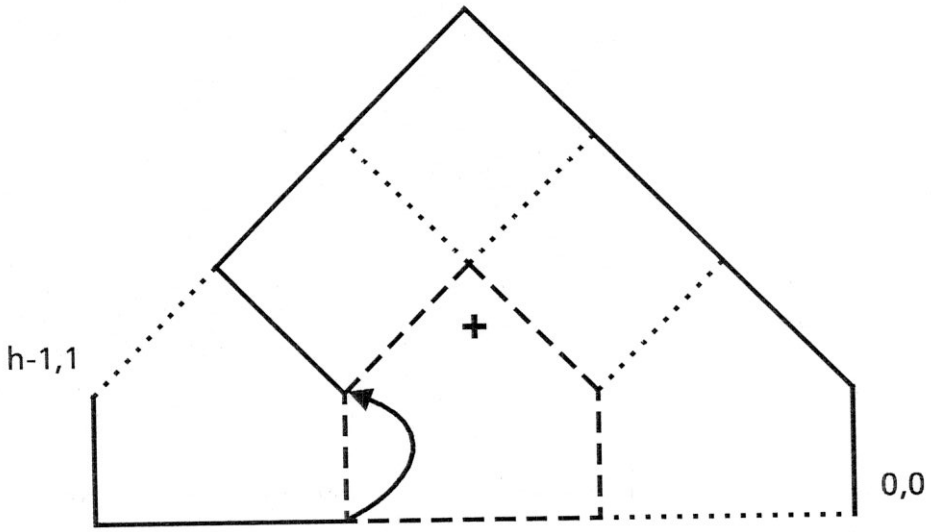


Figure 19 : (h-1,1) to (0,0) for $h > 2$

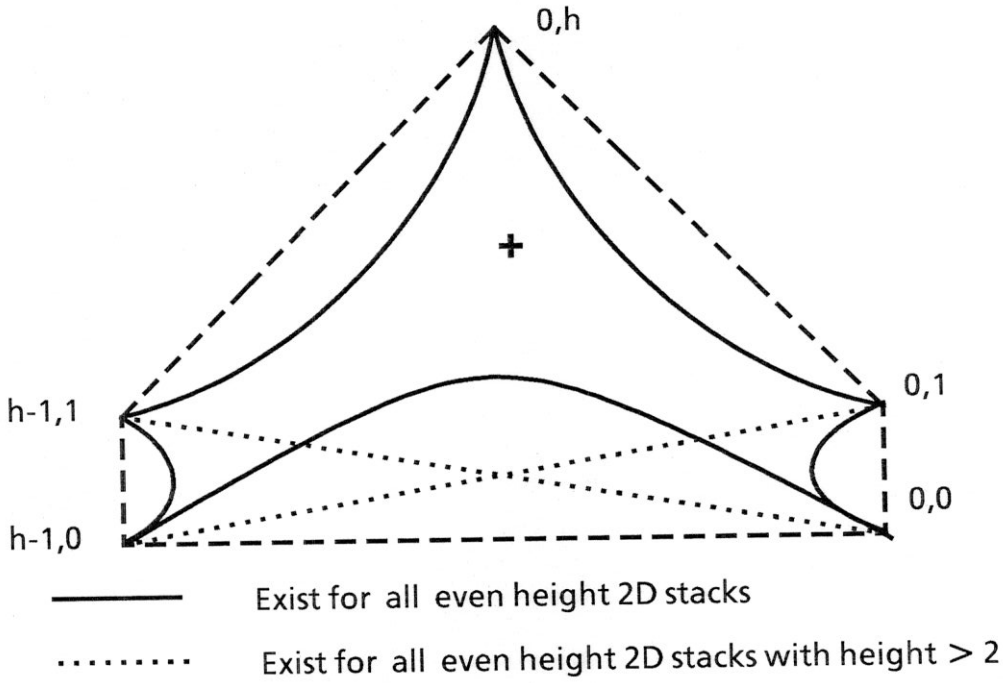


Figure 20
Summary of Corner Node to Corner Node
Hamiltonian Paths in 2D Stacks of Even Height

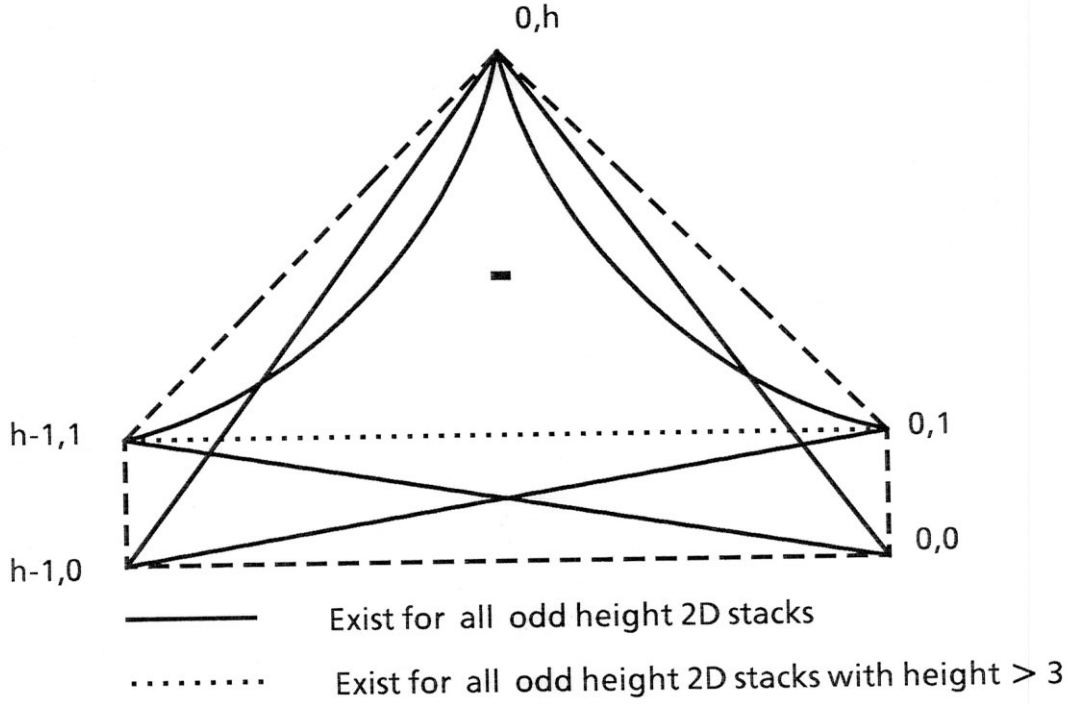
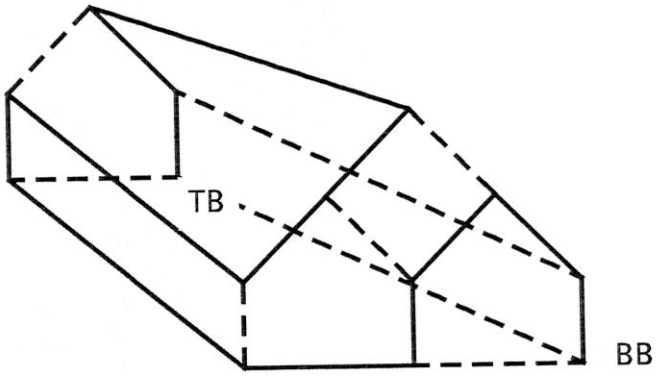
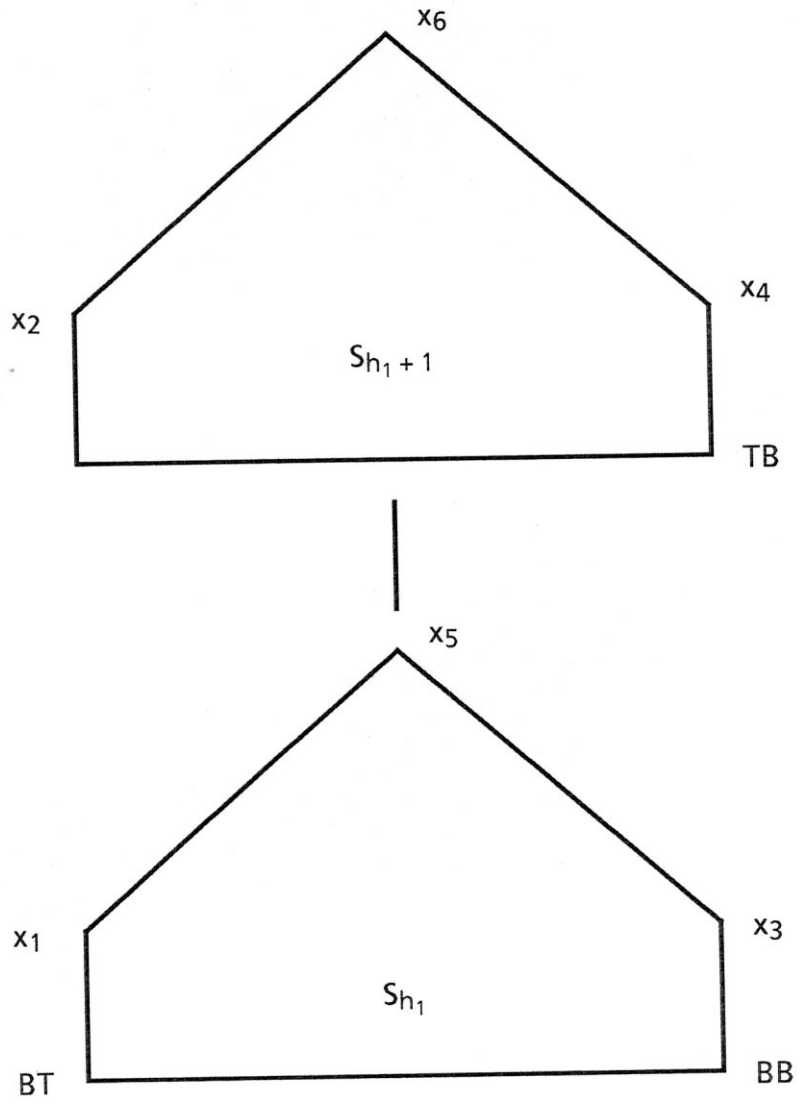


Figure 21
Summary of Corner Node to Corner Node
Hamiltonian Paths in 2D Stacks of Odd Height



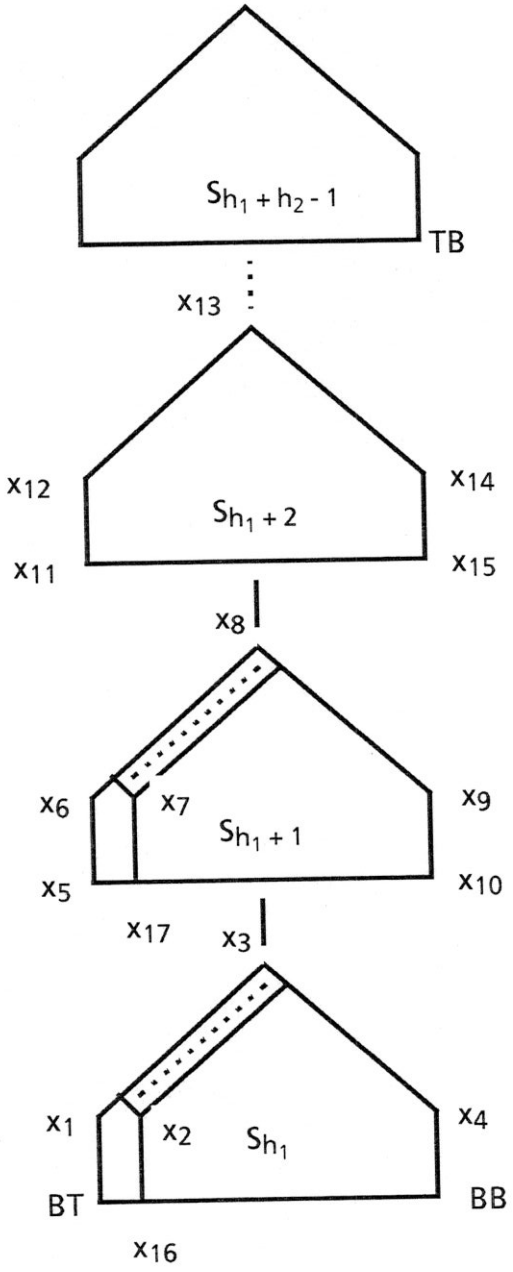
BB to TB in 3D, $h_3 = 0$

Figure 22



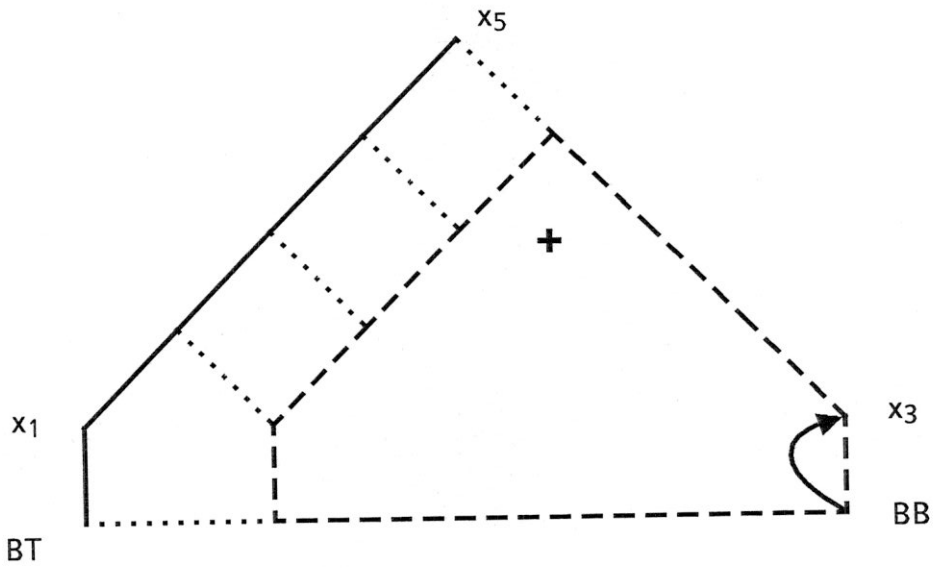
3D Stack for $h_2 = 2$

Figure 23



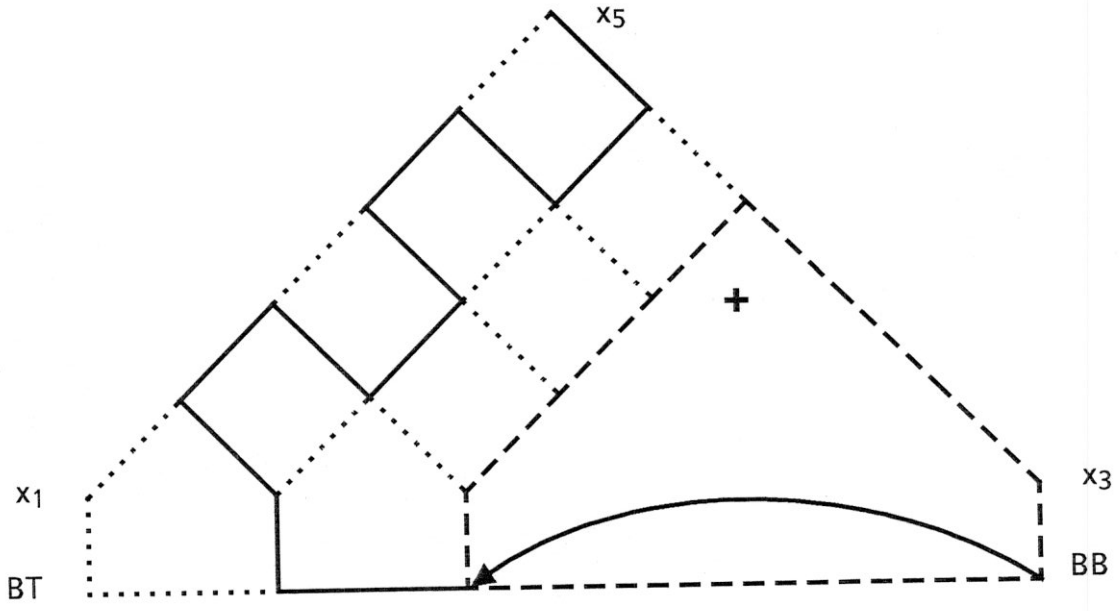
3D Stack for $h_2 > 2$

Figure 24



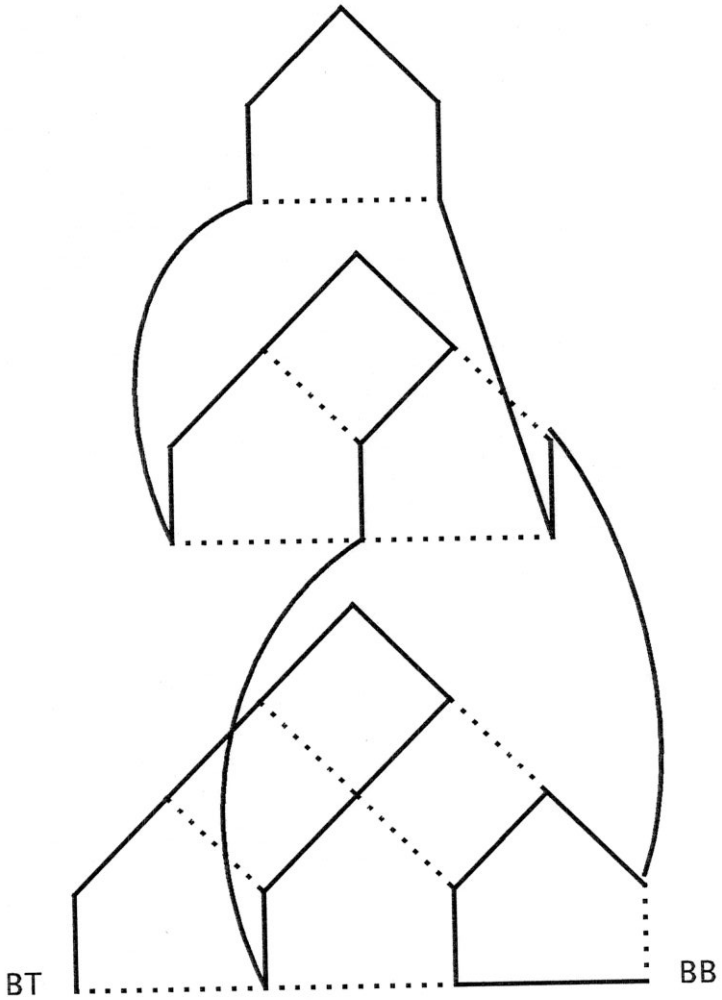
BB to BT for 2D, $h_2 = 2$

Figure 25



BB to BT for 2D, $h_2 = 2$

Figure 26



BB to BT Hamiltonian path for 3D when $h_2 = 3$ and $h_3 = 0$

Figure 27

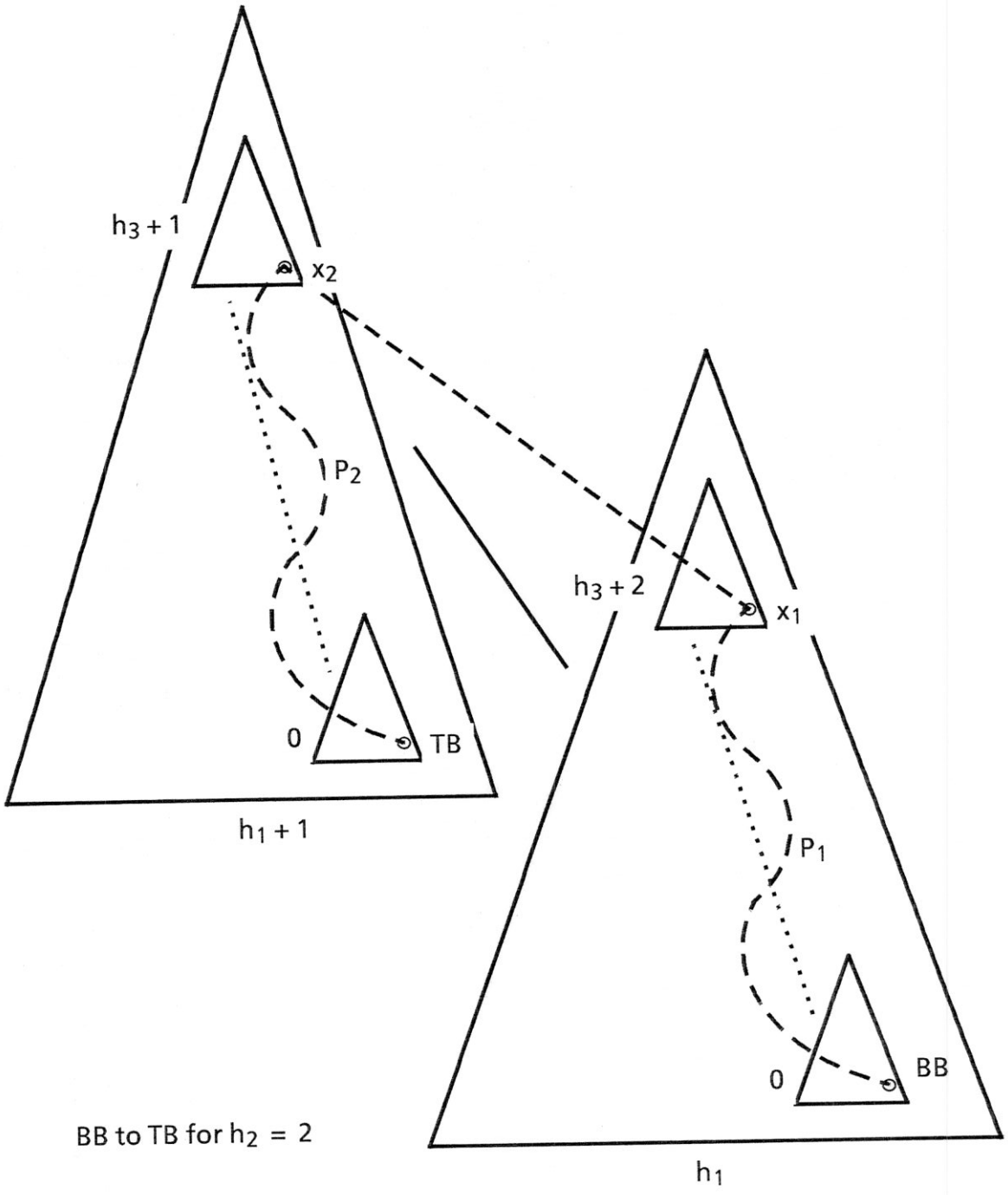


Figure 28

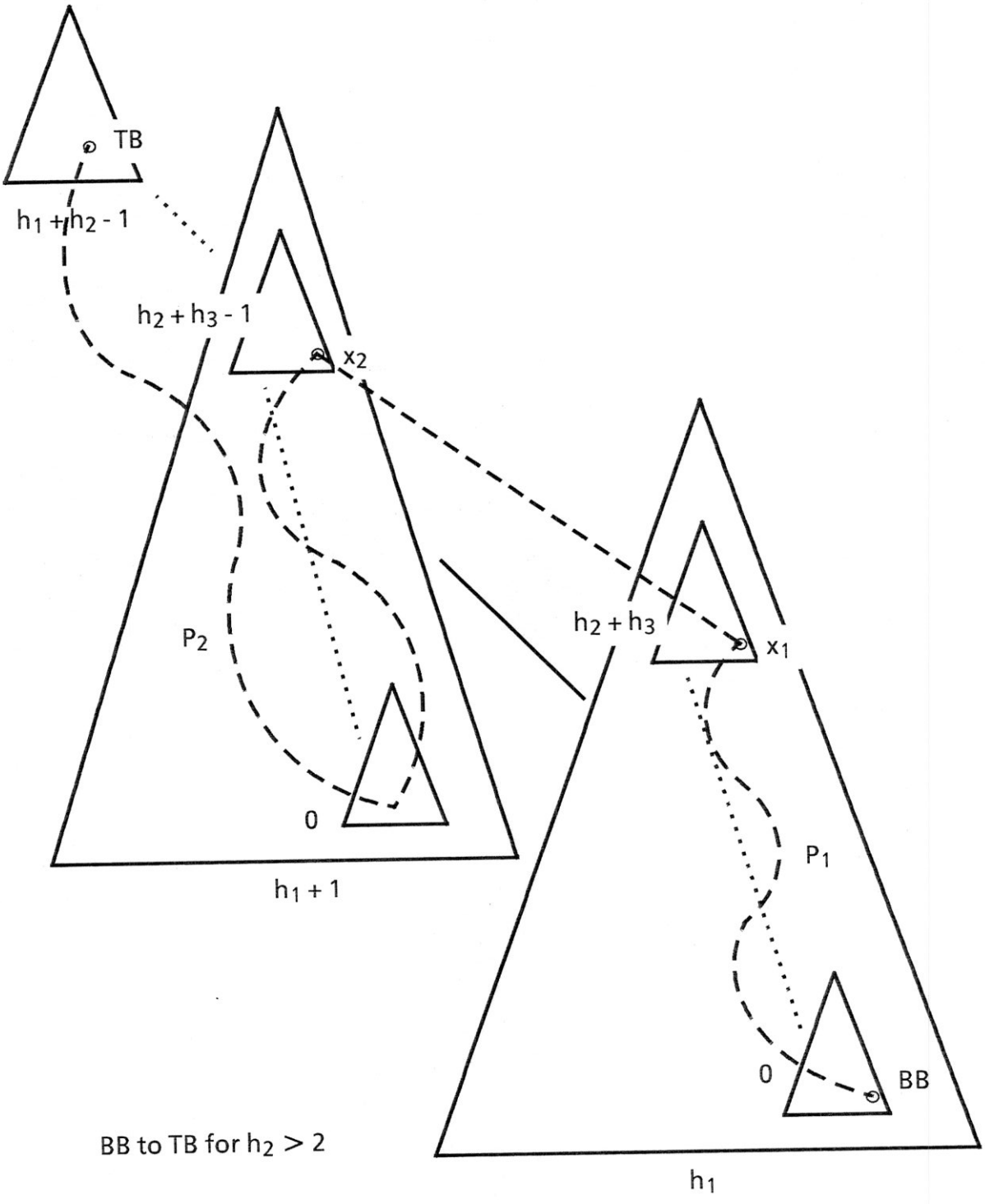


Figure 29

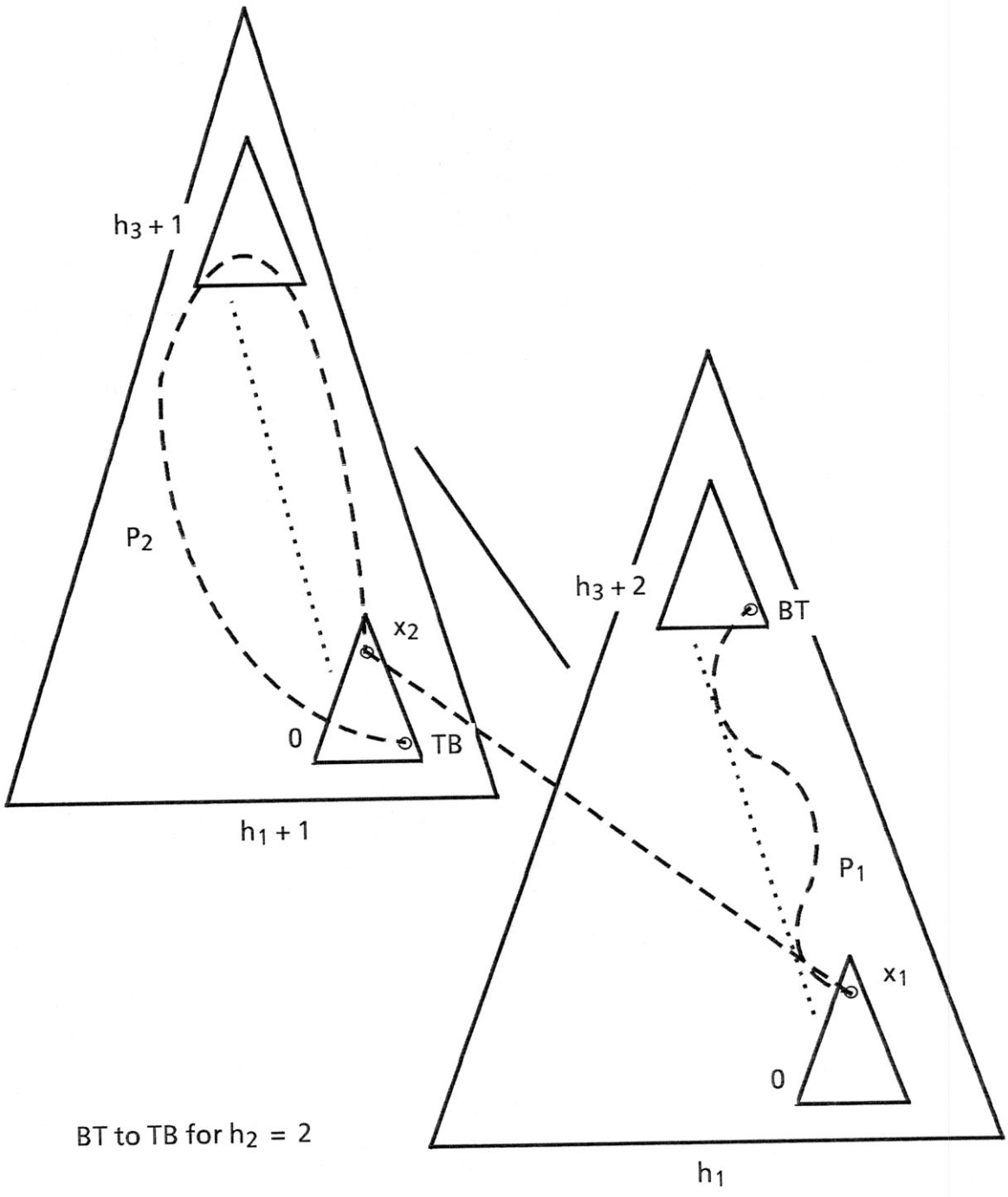


Figure 30

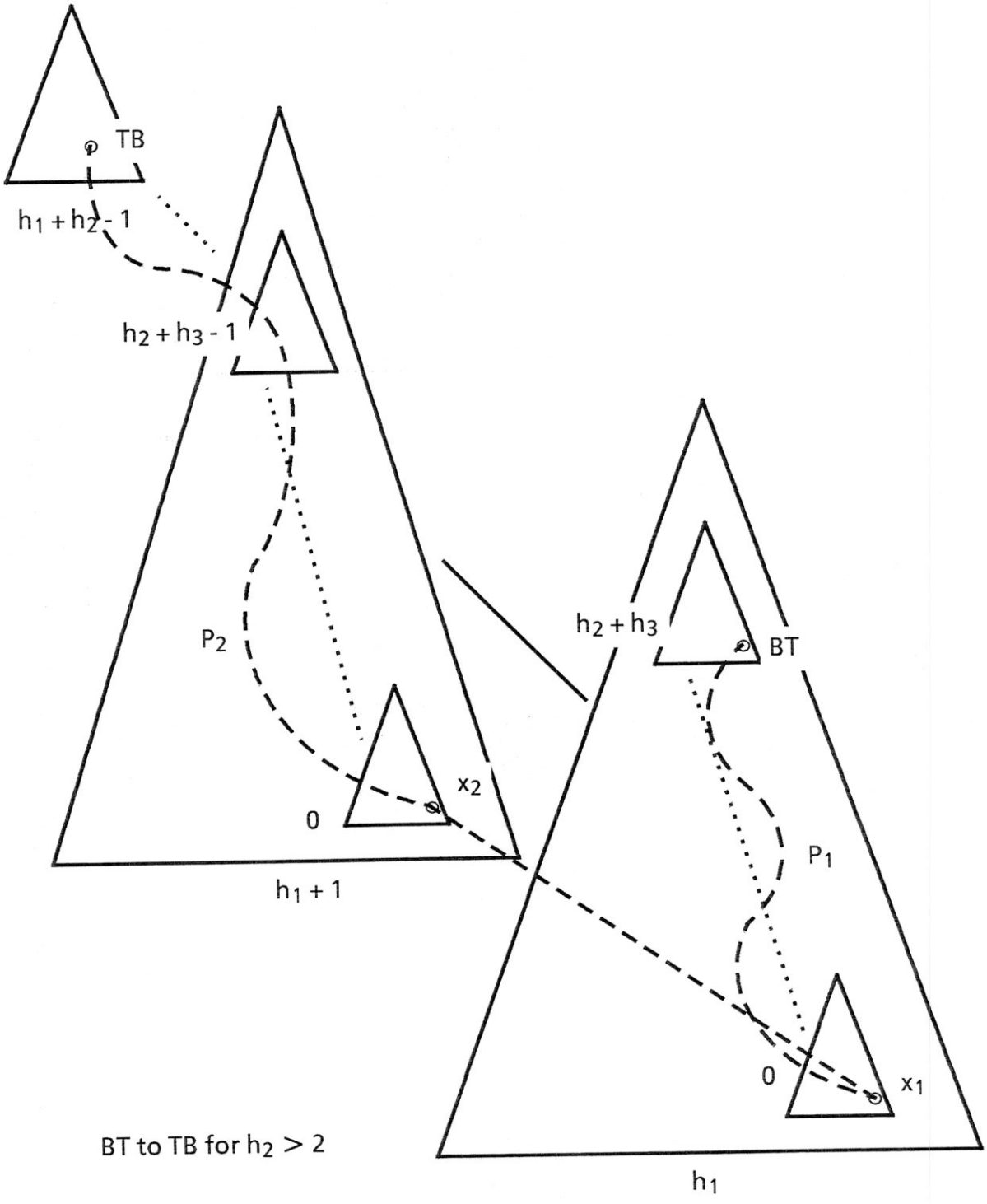


Figure 31

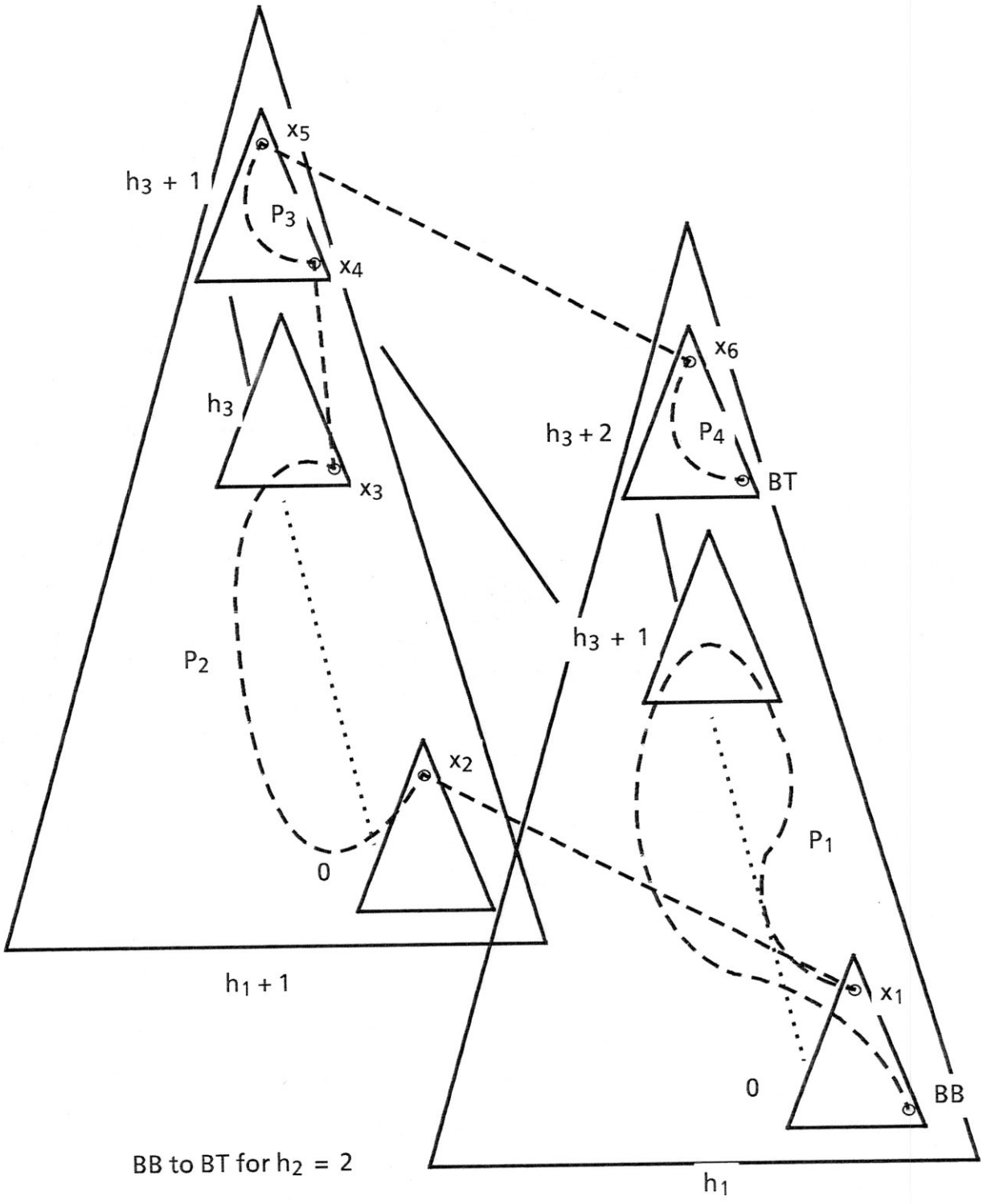

x5
h3 + 1
P3
x4
h3
x3
P2
0
x2
h1 + 1
x6
h3 + 2
P4
BT
h3 + 1
P1
x1
0
BB
BB to BT for h2 = 2
h1

Figure 32

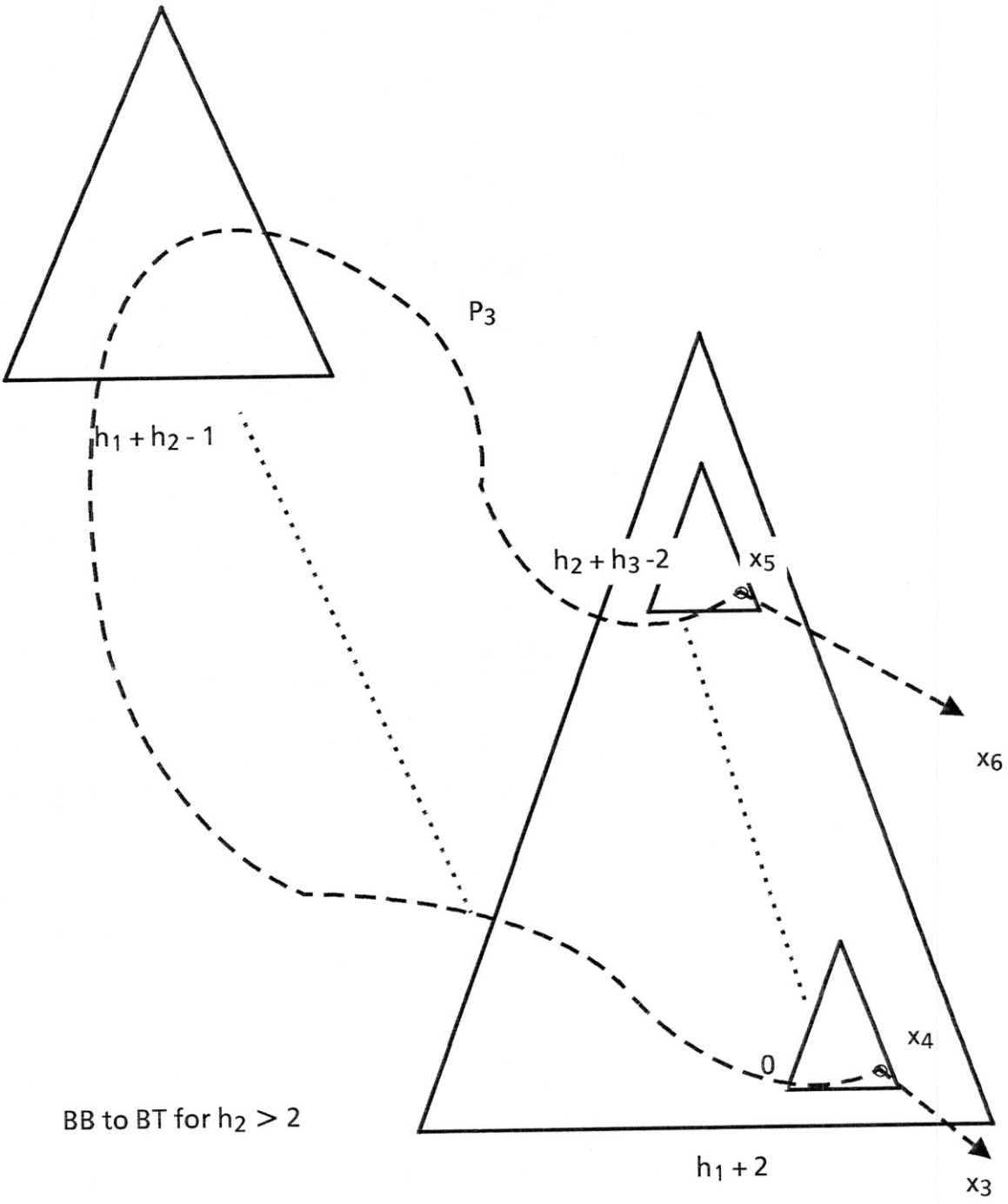


Figure 33

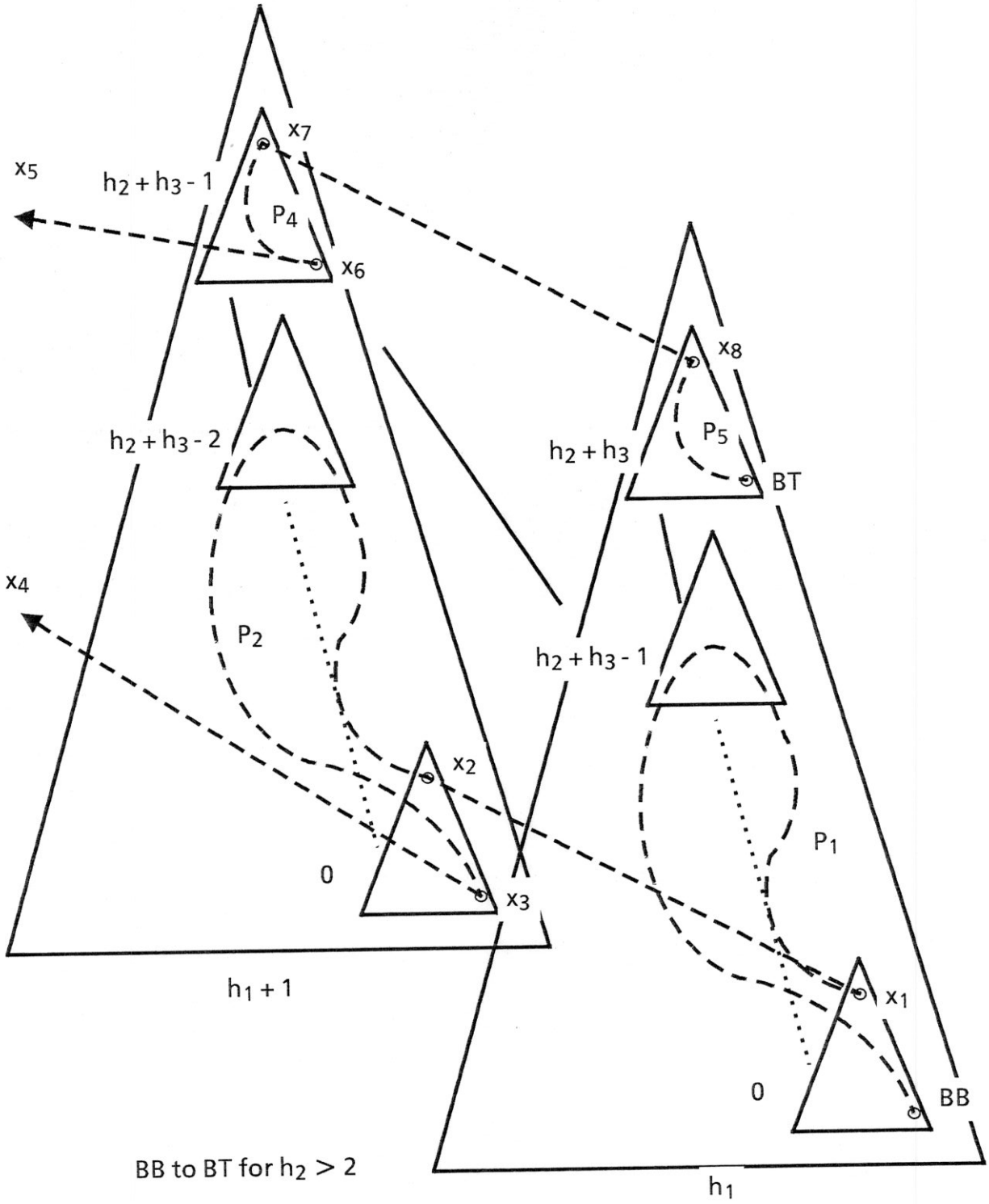
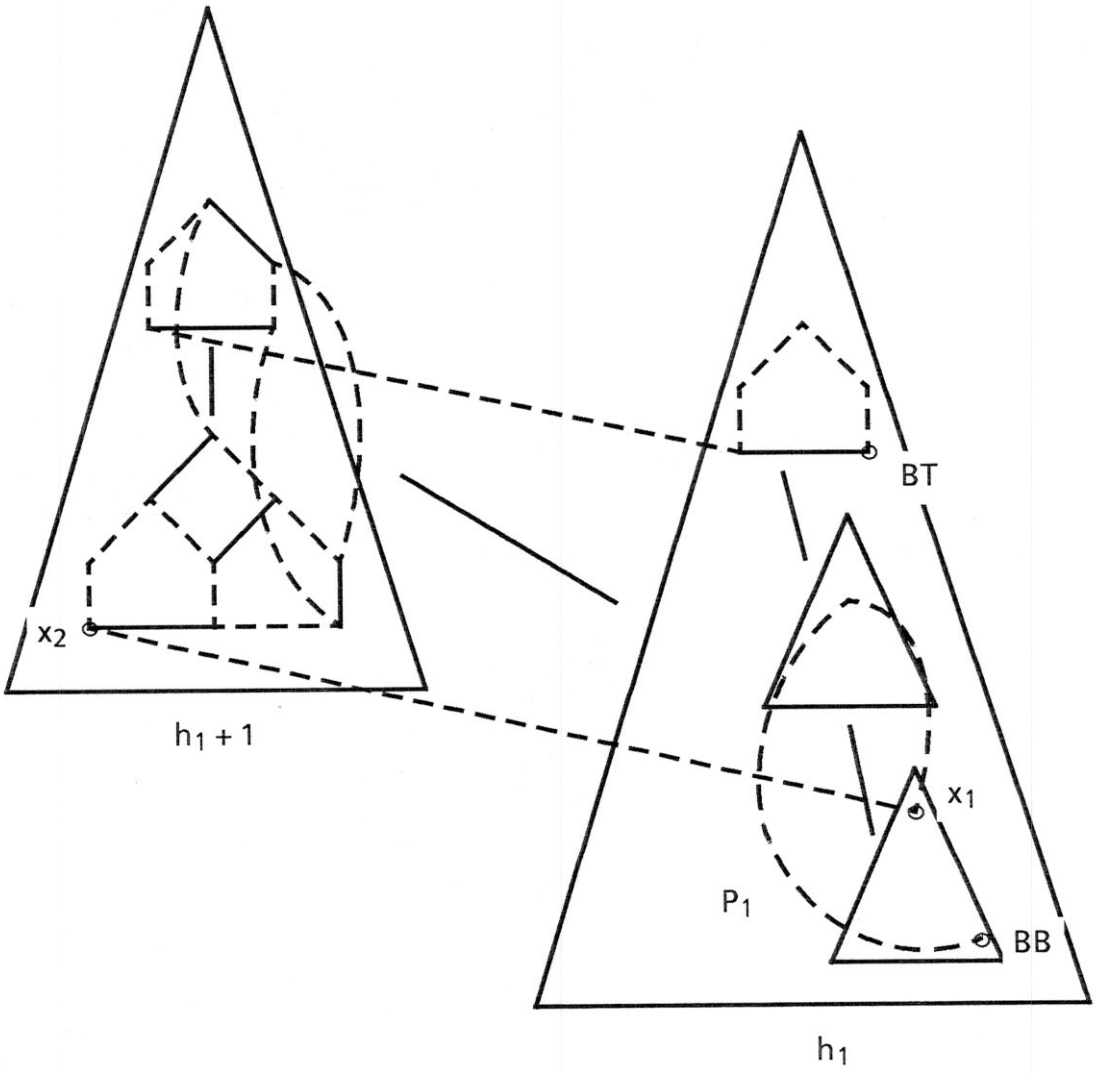

x7
x5
h2 + h3 - 1
P4
x6
x8
h2 + h3 - 2
h2 + h3
P5
BT
x4
P2
h2 + h3 - 1
x2
P1
0
x3
x1
h1 + 1
0
BB
BB to BT for h2 > 2
h1

Figure 34



BB to BT in 4D for $h_2 = 2, h_3 = 0$,
the dotted lines represent the path

Figure 35