Succinct Indices for Path Minimum, with Applications to Path Reporting*

Timothy M. Chan[1], Meng He[2], J. Ian Munro[1], and Gelin Zhou[1]

[1] David R. Cheriton School of Computer Science, University of Waterloo, Canada
{tmchan,imunro,g5zhou}@uwaterloo.ca
[2] Faculty of Computer Science, Dalhousie University, Canada
mhe@cs.dal.ca

Abstract. In the path minimum query problem, we preprocess a tree on n weighted nodes, such that given an arbitrary path, we can locate the node with the smallest weight along this path. We design novel succinct indices for this problem; one of our index structures supports queries in $O(\alpha(m,n))$ time, and occupies $O(m)$ bits of space in addition to the space required for the input tree, where m is an integer greater than or equal to n and $\alpha(m,n)$ is the inverse-Ackermann function. These indices give us the first succinct data structures for the path minimum problem, and allow us to obtain new data structures for path reporting queries, which report the nodes along a query path whose weights are within a query range. We achieve three different time/space tradeoffs for path reporting by designing (a) an $O(n)$-word structure with $O(\lg^{\epsilon} n + occ \cdot \lg^{\epsilon} n)$ query time, where occ is the number of nodes reported; (b) an $O(n \lg\lg n)$-word structure with $O(\lg\lg n + occ \cdot \lg\lg n)$ query time; and (c) an $O(n \lg^{\epsilon} n)$-word structure with $O(\lg\lg n + occ)$ query time. These tradeoffs match the state of the art of two-dimensional orthogonal range reporting queries [8] which can be treated as a special case of path reporting queries. When the number of distinct weights is much smaller than n, we further improve both the query time and the space cost of these three results.

1 Introduction

As one of the most fundamental structures in computer science, trees have been widely used in modeling and representing different types of data in numerous computer applications. In many cases, objects are represented by nodes and their properties are characterized by weights assigned to nodes. Researchers have studied the problems of maintaining a weighted tree, such that various types of *path queries* can be computed efficiently [1, 9, 24, 25, 23, 6, 20, 28, 22, 11]. In this paper, we consider *path minimum (maximum)* queries and *path reporting* queries.

- Path minimum(maximum): Given nodes u and v, return the minimum (maximum) node along the path from u to v, i.e., the node along the path whose weight is the minimum (maximum) one;

* This work was supported by NSERC and the Canada Research Chairs Program. Part of the first author's work was done during his visit to the Department of Computer Science and Engineering, Hong Kong University of Science and Technology.

A. Schulz and D. Wagner (Eds.): ESA 2014, LNCS 8737, pp. 247–259, 2014.

– Path reporting: Given nodes u and v along with a range $[p, q]$, report the nodes along the path from u to v whose weights are between p and q.

When the given tree is a path, the above queries become range minimum (maximum) queries [14, 11] and two-dimensional orthogonal range reporting queries [8], respectively. As stated in [20], the path queries we consider generalize these fundamental range queries to weighted trees.

In this paper, we represent the input tree as an ordinal one, i.e., a rooted tree in which siblings are ordered. The weights of nodes are assumed to be drawn from $[1..\sigma]$. We use lg to denote the base-2 logarithm and use ϵ to denote a constant in $(0, 1)$. Unless otherwise specified, the underlying model of computation is the standard word RAM model with word size $w = \Omega(\lg n)$.

Path Minimum. The problem of supporting path minimum queries has been heavily studied [1, 9, 2, 29, 23, 6, 11]. Unlike our formulation, previous work considers trees on weighted edges instead of weighted nodes. However, it is not hard to see that these two formulations are equivalent.

The *minimum spanning tree verification* problem is a special offline case of the path minimum problem, for which one should determine whether a given spanning tree is minimum. This problem can be solved using $O(n + m)$ comparisons and linear overhead under the word RAM model [24]. The online version of this problem, i.e., the path minimum problem, was considered by Alon and Schieber [1]. In the pointer machine model, they designed a structure that uses super-linear space. Concurrently, Chazelle [9] presented a linear space data structure under word RAM. Both structures require $O(\alpha(n))$ query time to sum up weights along a given path, where $\alpha(n)$ is the inverse-Ackermann function, and weights are drawn from a semigroup. Thus the online version of the path minimum problem can also be supported in the same time and space. More recently, several solutions using $O(n)$ words, i.e., $O(n \lg n)$ bits, with $O(1)$ query time have been designed under word RAM [2, 23, 6, 11]. Pettie [29] studied the lower bound in terms of comparisons of edge weights, and showed that $\Omega(q \cdot \alpha(q, n) + n)$ comparisons are necessary to serve q queries over a tree of size n.

In this paper we present lower and upper bounds for path minimum queries. In Lemma 4 we show that $\Omega(n \lg n)$ bits of space are necessary to encode the answers to path minimum queries over a tree of size n. This distinguishes path minimum queries from range minimum queries in terms of space cost.

We adopt the *indexing model* (also called the *systematic model*) [3, 7, 5] in designing new data structures for path minimum queries. Applying this model to weighted trees, we assume that weights are represented in an arbitrary given raw form; the only requirement is that the given data support access to the weight of a node given its preorder rank. Auxiliary data structures called *indices* are then constructed, and query algorithm uses indices and the access operator provided for the raw data. Not only is this an important theoretical model (its variants are frequently used to prove lower bounds [12, 26, 17]), it is also of practical importance as it addresses cases in which the (large) raw data are stored in slower external memory or even remotely, while the (smaller) indices could be stored in memory or locally. The space of an index is called *additional space*. Note that

the lower bound in the previous paragraph is proved under the encoding model, and thus does not apply to the indexing model.

To present our results, we assume the following definition for the Ackermann function: $A_0(i) = i + 1$ and $A_{\ell+1}(i) = A_\ell^{(i+1)}(i + 8)$, where $A_\ell^{(0)}(i) = i$ and $A_\ell^{(i)}(j) = A_\ell(A_\ell^{(i-1)}(j))$ for $i \geq 1$. This is faster growing than the one defined by Cormen et al. [10]. Let $\alpha(m, n)$ be the smallest L such that $A_L(\lfloor m/n \rfloor) > n$, and $\alpha(n)$ be $\alpha(n, n)$. The following theorem presents our indices for path minimum:

Theorem 1. *An ordinal tree on n weighted nodes can be indexed (a) using $O(m)$ bits of space to support path minimum queries in $O(\alpha(m, n))$ time and $O(\alpha(m, n))$ accesses to the weights of nodes, for any $m \geq n$; or (b) using $2n+o(n)$ bits of space to support path minimum queries in $O(\alpha(n))$ time and $O(\alpha(n))$ accesses to the weights of nodes.*

To better understand variant (a) of this result, we discuss the time and space costs for the following possible values of m. When $m = n$, then we have an index of $O(n)$ bits that supports path minimum queries in $O(\alpha(n))$ time. When $m = O(n(\lg^*)^* n)$, for example, then it is well-known that $\alpha(m, n) = O(1)$, and thus we have an index of $O(n(\lg^*)^* n)$ [1] bits that supports path minimum queries in $O(1)$ time. Previous solutions [2, 23, 6, 11] to the same problem with constant query time occupy $\Omega(n \lg n)$ bits of space in addition to the space required for the input tree. Combining the above results with a trivial encoding of node weights, we obtain the first succinct data structures for path minimum queries. With a little extra work, we can even represent a weighted tree using $n \lg \sigma + 2n + o(n)$ bits only, i.e., within an $o(n)$ additive term of the information-theoretic lower bound, to support queries in $O(\alpha(n))$ time. Considering the construction time is $O(n)$, this variant almost matches the lower bound of Pettie [29].

Path Reporting. Path reporting queries were proposed by He et al. [20]. They obtained two solutions: one uses $O(n)$ words and $O(\lg \sigma + occ \cdot \lg \sigma)$ query time, and the other uses $O(n \lg \lg \sigma)$ words but $O(\lg \sigma + occ \cdot \lg \lg \sigma)$ query time, where σ is the number of distinct weights and occ is output size. For the same problem, Patil et al. [28] designed a succinct structure based on *heavy path decomposition* [31, 18]. Their structure requires only $n \lg \sigma + 6n + o(n \lg \sigma)$ bits but $O(\lg \sigma \lg n + occ \cdot \lg \sigma)$ time. Concurrently, He et al. [22] designed another succinct structure based on a different idea. This structure, requiring $O(\lg \sigma / \lg \lg n + occ \cdot \lg \sigma / \lg \lg n)$ query time, is the best previously known linear space solution.

In this paper, we design three new data structures for path reporting queries:

Theorem 2. *An ordinal tree on n nodes whose weights are drawn from a set of σ distinct weights can be represented using $O(n \lg \sigma \cdot \mathtt{s}(\sigma))$ bits of space, such that path reporting queries can be supported in $O(\min\{\lg \lg \sigma + \mathtt{t}(\sigma), \lg \sigma / \lg \lg n\} + occ \cdot \min\{\mathtt{t}(\sigma), \lg \sigma / \lg \lg n\})$ time, where occ is the size of output, and $\mathtt{s}(\sigma)$ and*

[1] $(\lg^*)^*$ is the number of times $\lg^*$ must be iteratively applied before the result becomes less than or equal to 1.

$\mathtt{t}(\sigma)$ *are: (a)* $\mathtt{s}(\sigma) = O(1)$ *and* $\mathtt{t}(\sigma) = O(\lg^{\epsilon}\sigma)$*; (b)* $\mathtt{s}(\sigma) = O(\lg\lg\sigma)$ *and* $\mathtt{t}(\sigma) = O(\lg\lg\sigma)$*; or (c)* $\mathtt{s}(\sigma) = O(\lg^{\epsilon}\sigma)$ *and* $\mathtt{t}(\sigma) = O(1)$.

These results completely subsume almost all previous results; the only exceptions are the succinct data structures for this problem designed in previous work, whose query times are worse than our linear-space solution. Furthermore, our data structures match the state of the art of 2D range reporting queries [8] when $\sigma = n$, and have better performance when σ is much less than n.

Overview of Techniques. Unlike previous succinct tree structures [16, 28, 22, 19, 13], our approach for path minimum is based on *topological partitions* [15] which transform the input tree into a binary tree and further recursively decomposes it into a hierarchy of clusters with constant external degrees. Our main strategy of constructing path minimum query structures (in Section 3) is to recursively divide the set of levels of decomposition into multiple subsets of levels; with a carefully-defined version of the path minimum query problem which takes levels in the decomposition as parameters, the query over the entire structure can be answered by conquering the subproblems local to the subsets of levels. Solutions to special cases of the query problem are also designed, so that we can present the time and space costs of our solution using recursive formulas. Then, by carefully constructing a number series and using it in the division of levels into subsets, we can prove that our structures achieve the tradeoff presented in Theorem 1 using the inverse-Ackermann function. This approach is novel and exciting, and it does not directly use standard techniques for word RAM at all.

The above strategy would not achieve the desired space bound without a succinct data structure that supports navigations in the input tree, the binary tree that it is transformed into and the clusters in the topological partition. We design such a structure (in Section 4) occupying only $2n + o(n)$ bits, which is of independent interest. Finally, to design solutions to path reporting (in Section 5), we follow the general framework of He et al. [22] to extract subtrees based on the partitions of the entire weight range, and make use of a conceptual structure that borrows ideas from the classical range tree. One strategy of achieving improved results is to further reduce path reporting into queries in which the weight ranges are one-sided, which allows us to apply our succinct index for path minimum queries to achieve the tradeoffs presented in the second half of the abstract. We further apply a tree covering strategy to reduce the space cost for the case in which the number of distinct weights is much smaller than n, and hence prove Theorem 2.

2 Preliminaries

Succinct Data Structures. Bit vectors are one of the main building blocks in many space efficient data structures. Let $B[1..n]$ denote a bit vector of size n. For $\alpha \in \{0, 1\}$, $\mathtt{rank}_{\alpha}(B, i)$ counts α-bits in $B[1..i]$, while $\mathtt{select}_{\alpha}(B, i)$ finds the i-th α-bit in B. The following lemma presents succinct bit vector representations:

Lemma 1 ([30]). *A bit vector with $n-m$ zeros and m ones can be represented using $\lg \binom{n}{m} + O(n \lg\lg n / \lg n)$ bits of space to support $\texttt{rank}_\alpha$, $\texttt{select}_\alpha$, and the access to each bit in constant time.*

The next lemma presents succinct ordinal trees over an alphabet of size $\sigma = o(\lg\lg n)$, and unlabeled trees can be considered as a special case:

Lemma 2 ([16, 19, 13]). *An ordinal tree T on n nodes over an alphabet of size σ can be encoded in $n(\lg\sigma + 2) + O(\sigma n \lg\lg\lg n / \lg\lg n)$ bits of space to support the following operations in $O(1)$ time. Here x and y, which are nodes in T, are identified by preorder ranks. A node is its own 0-th ancestor. In addition, a node with label α is an α-node, and an α-node is an α-ancestor of its descendants.*

- $\texttt{depth}(T, x)$*: the depth of x (i.e., the number of ancestors of x);*
- $\texttt{depth}_\alpha(T, x)$*: the number of α-ancestors of x;*
- $\texttt{parent}(T, x)$*: the parent of x;*
- $\texttt{level\_anc}(T, x, i)$*: the i-th lowest ancestor of x;*
- $\texttt{level\_anc}_\alpha(T, x, i)$*: the i-th lowest α-ancestor of x;*
- $\texttt{LCA}(T, x, y)$*: the lowest common ancestor of x and y.*

Topological Partitions and Topology Trees. Frederickson [15] presented topological partitions for online updating of minimum spanning trees. An input tree T, which may have arbitrary degree, is transformed into a binary tree $\mathcal{B}$; and then $\mathcal{B}$ is partitioned as follows:

Lemma 3 ([15]). *A binary tree $\mathcal{B}$ on n nodes can be partitioned into a hierarchy of clusters with $h+1$ levels for some $h = O(\lg n)$: each cluster is a connected component of $\mathcal{B}$; the only cluster at level h contains all the nodes in $\mathcal{B}$, and each cluster at level 0 contains a single node; a cluster at level $i > 0$ is the disjoint union of at most 4 clusters at level $i-1$; at level i, there are at most $(3/4)^i n$ clusters of size at most 4^i, which form a partition of the nodes in the binary tree; and each cluster has at most 3 nodes that are connected to the outside, which are called its endpoints.*

The hierarchy of clusters is referred to as the topology tree of T and $\mathcal{B}$, which is denoted by $\mathcal{H}$. A node at level $i > 0$ of $\mathcal{H}$ represents a cluster C at level i of the hierarchy, and its children represent the clusters at the lower level that partition C. In particular, the leaf nodes of $\mathcal{H}$, which are at level 0, represent individual nodes of the binary tree $\mathcal{B}$.

Tree Extraction. He et al. [20, 22] introduced tree extraction to support path queries. This technique is based on the deletion operation of tree edit distance [4]. To delete a non-root node u, its children are inserted in place of u into the list of children of its parent, preserving the original left-to-right order. Let T be an ordinal tree and X be a subset of nodes in T. The X-extraction of T, F_X, is defined to be the ordinal forest obtained by deleting all the nodes that are not in X from T. There is a natural one-to-one correspondence between the nodes in X and the nodes in F_X, and the ancestor-descendant and preorder relationships among the remaining nodes are preserved. If X contains the root of T, then F_X consists of a single ordinal tree only, which is denoted by T_X.

3 Path Minimum Queries

We first give a simple lower bound for path minimum queries under the *encoding model*. Due to page limitation, the proof is omitted here.

Lemma 4. *In the worst case, $\Omega(n \lg n)$ bits are required to encode the answers to all possible path minimum queries over a tree on n weighted nodes.*

Unlike the lower bound of Pettie [29], Lemma 4 provides a separation between path minimum and range minimum in terms of space: $\Omega(n \lg n)$ bits are required to encode path minimum queries over a tree on n weighted nodes, while range minimum over an array of length n can always be encoded in $2n$ bits [14].

Now we consider the support for path minimum queries. The space cost of maintaining a weighted tree is dominated by storing the weights of nodes. Thus we represent the input tree as an ordinal one, for which the nodes are identified by their preorder ranks. This does not significantly affect the space cost.

We will assume the indexing model described in Section 1 and develop several novel succinct indices for path minimum queries. Thus the weights of nodes are assumed to be stored separately from the index for queries, and can be accessed with the preorder ranks of nodes. The time cost to answer a given query is measured by the number of accesses to the index and that to node weights.

Let T be an input tree on n nodes. Here T is represented as an ordinal one, and its nodes are identified by preorder ranks. We transform T into a binary tree $\mathcal{B}$ as follows (essentially as in the usual way but with added dummy nodes): For each node u with $d > 2$ children, where $v_1, v_2, \cdots, v_d$ are children of u, we add $d-2$ dummy nodes $x_1, x_2, \cdots, x_{d-2}$. The first and the second child of u are set to be v_1 and x_1, respectively. For $1 \leq k < d-2$, the first and the second child of x_k are set to be v_{k+1} and x_{k+1}, respectively. Finally, the first and the second child of x_{d-2} are set to be v_{d-1} and v_d, respectively. In this way we have replaced u and its children with a right-leaning binary tree, where the leaf nodes are children of u. This transformation does not change the preorder relationship among the nodes in T. In addition, the set of non-dummy nodes along the path between any two non-dummy nodes remain the same after transformation.

We decompose $\mathcal{B}$ and obtain the topology tree $\mathcal{H}$ using Lemma 3. For simplicity, a cluster at level i is called a *level-i cluster*, and its endpoints are called *level-i endpoints*. Since T and $\mathcal{B}$ are both rooted trees, each cluster contains a node that is the ancestor of all the other nodes in the same cluster. This node is referred to as the *root* of the cluster. In the topology tree $\mathcal{H}$, sibling clusters are ordered by the preorder ranks of their roots. Each cluster C is identified by its *topological rank*, i.e., the preorder rank of the node in $\mathcal{H}$ that represents C.

To facilitate the use of topology trees, we define operations relevant to nodes, clusters and endpoints. Here we assume that x and y are nodes in $\mathcal{B}$.

- conversions between nodes in $\mathcal{B}$ and T;
- `level_cluster`$(\mathcal{H}, i, x)$: the level-i cluster that contains node x;
- `LCC`$(\mathcal{H}, x, y)$: the lowest-level cluster C that contains nodes x and y;
- `cluster_root`$(\mathcal{H}, C)$: the root of level-i cluster C;

- `cluster_endpoints`$(\mathcal{H}, C)$: the endpoints of level-i cluster C;
- `closest_endpoint`$(\mathcal{H}, C, x)$: the endpoint of C that is the closest to node x, given that x is outside of C;
- `parent`$(\mathcal{B}, x)$: the parent node of x;
- `LCA`$(\mathcal{B}, x, y)$: the lowest common ancestor of x and y;
- `endpoint_rank`$(\mathcal{B}, i, x)$: the number of level-i endpoints preceding x in preorder of $\mathcal{B}$;
- `endpoint_select`$(\mathcal{B}, i, j)$: the j-th level-i endpoint in preorder of $\mathcal{B}$.

Lemma 5. *Let T be an ordinal tree on n nodes. Then T, the transformed binary tree $\mathcal{B}$, and their topology tree $\mathcal{H}$ can be encoded in $2n + o(n)$ bits of space such that the operations listed above can be supported in $O(1)$ query time.*

To present our key strategy first, we defer the proof of Lemma 5 to Section 4. As the conversion between nodes in T and $\mathcal{B}$ can be performed in $O(1)$ time, we assume that each given query is specified by two nodes in $\mathcal{B}$. Let h denote the highest level of $\mathcal{H}$. The following two query problems are defined in terms of clusters and endpoints, for $0 \leq i < j \leq h$:

- $P_{i,j}(C_0, C_1, C_2)$: find the minimum node along the path from an endpoint of a level-i cluster C_1 to an endpoint of another level-i cluster C_2, where both C_1 and C_2 are contained in the same level-j cluster C_0;
- $P'_{i,j}(C_0, C_1)$: find the minimum node along the path from an endpoint of a level-i cluster C_1 to an endpoint of a level-j cluster C_0, where C_1 is in C_0.

For simplicity, we drop the parameters when referring to these problems in the rest of this section. Thus the original problem is $P_{0,h}$. If $P_{i,j}$ is solved, then $P'_{i,j}$ and $P_{i',j}$ for $i' > i$ are also naturally solved.

Let $h_0 > 0$ be a parameter whose value will be determined later. We will solve P_{0,h_0} using brute-force search, and support $P_{h_0,h}$ using a novel recursive approach as described below. For each cluster C whose level is higher than or equal to h_0, we explicitly store the minimum node on the *bridges* of C, where bridges are paths connecting pairs of endpoints of C (excluding the endpoints). This requires $6i$ bits for a cluster at level i, as it has at most three bridges. The overall space cost is $\sum_{i=h_0}^{h} \left(6i \cdot (3/4)^i n\right) = O(h_0(3/4)^{h_0} n)$ bits, which is $o(n)$ bits when $h_0 = \omega(1)$. We have the following lemmas as base cases of recursion.

Lemma 6. *For $i \geq h_0$, $P_{i,i+8}$ can be solved in $O(1)$ query time and 0 extra bits.*

The correctness of this lemma follows from the fact that each level-$(i+8)$ cluster contains a constant number of level-i clusters, which makes it possible to split a query path into a constant number of subpaths, each being either a level-i endpoint or a bridge of some level-i cluster which have been preprocessed.

Lemma 7. *$P'_{i,j}$ can be solved using $O(1)$ query time and $O((3/4)^i n)$ extra bits.*

Proof. We construct an ordinal tree T_i by extracting all level-i endpoints from $\mathcal{B}$. The size of T_i is $O((3/4)^i n)$. For convenience, we denote a node in T_i by u' iff it corresponds to a level-i endpoint u in $\mathcal{B}$. The conversion between u and u'

can be performed in $O(1)$ time using `endpoint_rank` and `endpoint_select`.

Next we assign labels from alphabet $\{0, 1\}$ to the nodes of T_i. We only consider the case in which the level-j endpoint is the first one in preorder of its cluster; the other cases can be handled similarly. Let u be any level-i endpoint and let v be the first endpoint of $C_0 = \texttt{level\_cluster}(\mathcal{B}, j, u)$, i.e., the level-$j$ cluster that contains u. Like the proof of Lemma 6, the path from u to v in $\mathcal{B}$ can be split into a sequence of level-i endpoints and bridges of level-i clusters. Let x be the next level-i endpoint on the path. We assign 1 to u' in T_i if the minimum node between u and v is smaller than that between x and v; otherwise we assign 0 to u'. We represent this labeled tree in $O((3/4)^i n)$ bits using Lemma 2.

To find the minimum node between u and v, we need only find the closest 1-node to u' along the path from u' to v' in T_i. This can be done in $O(1)$ time by performing $\texttt{level\_anc}_\alpha$ and $\texttt{depth}_\alpha$ operations on T_i. Let x' be such a node. Then the minimum node between u and v must be x or appear on some bridge of the level-i cluster that contains x, and thus can be retrieved in $O(1)$ time. □

Now we turn to consider general $P_{i,j}$, for which we will develop a recursive strategy with multiple iterations. At each iteration, we pick a sequence $i = i_0 < i_1 < i_2 < \ldots < i_k = j$, for which $P_{i_0,i_1}, P_{i_1,i_2}, \ldots, P_{i_{k-1},i_k}$ are assumed to be solved at the previous iteration. By Lemma 7, we solve $P'_{i,i_1}, P'_{i,i_2}, \ldots, P'_{i,i_k}$ using $O(k(3/4)^i n)$ bits of additional space.

Consider the support for a query of $P_{i,j}$, for which level-i endpoints u and t are endpoints of the query path, and u and t are contained in the same level-j cluster. W.l.o.g, we assume that t is an ancestor of u (the case in which neither node is an ancestor of the other can be reduced to this case easily). We compute $C_0 = \texttt{LCC}(\mathcal{B}, u, t)$, which is the lowest level cluster that contains both u and t. Let i' be the level of C_0. Then we determine s such that $i_s < i' \le i_{s+1}$; s can be computed in constant time by precomputing the result for each of the $h + 1 = O(\lg n)$ levels. Let C_1 be the level-i_s cluster that contains u and let x be an endpoint of C_1 that is between u and t. Similarly, let C_2 be the level-i_s cluster that contains t and let z be an endpoint of C_2 that is between u and t. Note that x and z can be found in constant time using `level_cluster` and `closest_endpoint`. Thus the query path can be decomposed into $u \sim x \sim z \sim t$. The minimum node on $u \sim x$ and that on $z \sim t$ can be found by querying P'_{i,i_s}. The minimum node on $x \sim z$ can be found by recursively querying $P_{i_s,i_{s+1}}$.

Summarizing the discussion above, we have the following recurrences. Here ℓ is the number of iterations, and $Q_\ell(i, j)$ and $S_\ell(i, j)$ are time and space costs for solving $P_{i,j}$ at the ℓ-th iteration.

$$S_{\ell+1}(i, j) = \sum_{s=0}^{k-1} S_\ell(i_s, i_{s+1}) + O(k(3/4)^i n) \tag{1}$$

$$Q_{\ell+1}(i, j) = \max_{s=0}^{k-1} Q_\ell(i_s, i_{s+1}) + O(1) \tag{2}$$

Lemma 8. *Given a fixed value L, there exists a recursive strategy and some constant c such that, for $0 \le \ell \le L$, $S_\ell(i, A_\ell(i)) \le c(4/5)^i n$ and $Q_\ell(i, A_\ell(i)) \le c\ell$.*

Proof. At the 0-th iteration, we have $A_0(i) = i + 1$. This can be used as the base case. By Lemma 6, $P_{i,i+1}$ can be supported using constant query time at no extra space cost. Thus the statement holds for $\ell = 0$.

At the $(\ell + 1)$-th iteration, we choose the sequence $i, i + 8, A_\ell(i + 8), A_\ell^{(2)}(i + 8), \ldots, A_\ell^{(i)}(i + 8), A_\ell^{(i+1)}(i + 8)$. The last term is $A_{\ell+1}(i)$. Then by Equation 1:

$$\begin{aligned} S_{\ell+1}(i, A_{\ell+1}(i)) &\le \sum_{0 \le j \le i} S_\ell(A_\ell^{(j)}(i + 8), A_\ell^{(j+1)}(i + 8)) + O(i(3/4)^i n) \\ &\le O(i(3/4)^i n) + \sum_{0 \le j \le i} c(4/5)^{A_\ell^{(j)}(i+8)} n \\ &\le O(i(3/4)^i n) + 5c(4/5)^{i+8} n \;\le\; c(4/5)^i n \end{aligned}$$

for some sufficiently large constant c. This convergence follows because $5(4/5)^8$ is less than 1. Equation 2 implies $Q_{\ell+1}(i, A_{\ell+1}(i)) \le O(1) + \max_{0 \le j \le i} Q_\ell(A_\ell^{(j)}(i + 8), A_\ell^{(j+1)}(i+8)) \le O(1)+c\ell \le c(\ell+1)$ for some sufficiently large c. The induction thus carries through. □

To achieve desired time-space tradeoffs, we recurse one more iteration. Given a parameter $m \ge n$, we set $L = \alpha(m, n)$ and $h_0 = 0$. At the final $(L + 1)$-th iteration, choose the sequence $0, 1, 2, \ldots, \lfloor m/n \rfloor, A_L(\lfloor m/n \rfloor)$. This gives

$$S_{L+1}(0, A_L(\lfloor m/n \rfloor)) \le S_L(\lfloor m/n \rfloor, A_L(\lfloor m/n \rfloor)) + O(\lfloor m/n \rfloor n) \;\le\; O(m)$$

and $Q_{L+1}(0, A_L(\lfloor m/n \rfloor)) \le O(L) = O(\alpha(m, n))$. Thus we have proved part (a) of Theorem 1.

To further decrease the space cost of our index to $2n + o(n)$ bits, we still recurse L levels. We choose $L = \alpha(n)$ and $h_0 = \lceil \log_4 L \rceil$. Note that $h_0 = \omega(1)$ and $A_L(h_0) \ge h$. Therefore we have $S_L(h_0, A_L(h_0)) = O((4/5)^{h_0} n) = o(n)$, and $Q_L(h_0, A_L(h_0)) = O(L) = O(\alpha(n))$. To solve P_{0,h_0} and P'_{0,h_0}, we perform brute-force search on the query path in $O(4^{h_0}) = O(\alpha(n))$ time, as a level-h_0 cluster has at most 4^{h_0} nodes. Thus we have proved part (b) of Theorem 1.

By further constructing the preorder label sequence $[22, 21]$ of T, we have:

Corollary 1. *Let T be an ordinal tree on n nodes, each having a weight drawn from $[1..\sigma]$. Then T can be represented (a) using $n \lg \sigma + O(m)$ bits of space to support path minimum queries in $O(\alpha(m, n))$ time, for any $m \ge n$; or (b) using $n(\lg \sigma + 2) + o(n)$ bits of space to support path minimum queries in $O(\alpha(n))$ time.*

4 Encoding Topology Trees

Let T be an ordinal tree on n nodes. As described in Section 3, we transform T into a binary tree $\mathcal{B}$, and compute the topology tree of $\mathcal{B}$ as $\mathcal{H}$. Let $n_{\mathcal{H}}$ denote the number of nodes in $\mathcal{H}$; clearly $n_{\mathcal{H}} = O(n)$. Let $i_1 = \lceil 8 \lg \lg n \rceil$ and $i_2 = \lfloor (1/2) \lg \lg n \rfloor - 1$. By Lemma 3, there are at most $n_1 = (3/4)^{i_1} n = O(n/(4/3)^{8 \lg \lg n}) = O(n/\lg^{8 \lg(4/3)} n) < O(n/\lg^3 n)$ clusters at level i_1, each

being of size at most $m_1 = 4^{i_1} \leq 4^{8 \lg \lg n + 1} = 4 \lg^{16} n$. Similarly, there are at most $n_2 = (3/4)^{i_2} n = O(n/\lg^{(1/2)\lg(4/3)} n) < O(n/(\lg^{1/5} n))$ clusters at level i_2, each being size of at most $m_2 = 4^{i_2} \leq 4^{(1/2)\lg\lg n - 1} = (\lg n)/4$. Level-$i_1$ clusters are called *mini-clusters*, and level-i_2 ones are called *micro-clusters*. We first store the encodings of micro-clusters.

Lemma 9. *All micro-clusters can be encoded in $2n + o(n)$ bits of space such that given the topological rank of a cluster, its encoding can be retrieved in $O(1)$ time if it is a micro-cluster.*

Proof. Note that $\mathcal{B}$ has at most $2n$ nodes. Given a micro-cluster C, we do not store its encoding directly because it could require about $4n$ bits of space for all micro-clusters. Instead, we define X to be the union of non-dummy nodes and endpoints of C and store only C_X, where C_X is the X-extraction of C as defined in Section 2. We also mark the (at most 3) dummy nodes in C_X, which requires $O(\lg m_2) = O(\lg \lg n)$ bits per node. Encoding C_X as *balanced parentheses* [27], the overall space cost of encoding C is $2n_C + O(\lg \lg n)$ bits, where n_C is the number of non-dummy nodes in C. We concatenate the above encodings of all micro-clusters ordered by topological rank and store them in a sequence, P, of $n' = 2n + O(n \lg \lg n/(\lg^{1/5} n))$ bits. We construct a sparse bit vector, P', of the same length, and set $P'[i]$ to 1 iff $P[i]$ is the first bit of the encoding of a micro-cluster. P' can be represented using Lemma 1 in $\lg \binom{n'}{n_2} + O(n \lg \lg n / \lg n) = O(n \lg \lg n/(\lg^{1/5} n))$ bits to support $\texttt{rank}_\alpha$ and $\texttt{select}_\alpha$ in constant time. We construct another bit vector $B_0[1..n_{\mathcal{H}}]$, in which $B_0[j] = 1$ iff the cluster with topological rank j is a micro-cluster, which has the same asymptotic space cost.

To retrieve the encoding of a cluster, C, whose topological rank is j, we first use B_0 to check if C is a micro-cluster. If it is, let $r = \texttt{rank}_1(B_0, j)$. Then the encoding of C_X is $P[\texttt{select}_1(P', r)..\texttt{select}_1(P', r+1) - 1]$. To recover C from C_X, we need only follow the procedure described at the beginning of Section 3. This can be done in $O(1)$ time using a lookup table F_0 of $o(n)$ bits. □

To support operations, our main strategy is to encode global information at levels on or above i_1, information local to a mini-cluster among levels between i_2 and i_1, and lookup tables for the levels contained in a micro-cluster. Extra care is needed as the topological order of clusters is not the same as the relative order of their roots in preorder. Details are omitted due to page limitation.

5 A Sketch of Supporting Path Reporting Queries

In this section we sketch our improved data structures for path reporting queries. The details of supporting path reporting queries are deferred to the full version of this paper. Our solutions present various algorithmic techniques we developed to prove Theorem 2, as well as a simplified approach to achieve similar time-space tradeoffs for the 2D orthogonal range reporting problem on an $n \times \sigma$ grid.

Following the approach of He et al. [22], we build a conceptual range tree on $[1..\sigma]$ with branching factor $f = \lceil \lg^{\epsilon} n \rceil$. We keep splitting ranges until the bottom level contains σ leaf ranges. This conceptual range tree has $h = \lceil \log_f \sigma \rceil + 1$ levels, among which the top level is the first level, and the bottom level is the h-th level. For each range $[a..b]$ in the conceptual range tree, we obtain $F_{a,b}$ by extracting all nodes whose weights are in $[a..b]$ from T. Each node x in $F_{a,b}$ is assigned a label between 1 and f, which indicates the child range at the lower level that contains the node that corresponds to x. All these $F_{a,b}$'s are maintained in succinct representations such that path reporting queries with respect to labels can be answered in constant time per node.

Let u and v be the endpoints of query path and $[p..q]$ be the query range. The algorithm of He et al. [22] traverses the conceptual range tree from top to bottom, and splits $[p..q]$ into $O(\lg \sigma / \lg \lg n)$ canonical ranges, each being a single range, or the union of consecutive sibling ranges in the conceptual range tree. Thus the original query can be transformed into $O(\lg \sigma / \lg \lg n)$ subqueries on different $F_{a,b}$'s. This relatively simple algorithm requires an overhead of $O(\lg \sigma / \lg \lg n)$ time, in addition to $O(\lg \sigma / \lg \lg n)$ time per node in the output, since we need convert a node in $F_{a,b}$ to that of T level by level.

Our new algorithm avoids traversing the conceptual range tree level by level. We make use of the *ball-inheritance* problem, such that, given a node x in some $F_{a,b}$, we can determine the node in T that corresponds to x much faster than $O(\lg \sigma / \lg \lg n)$ time. To support the given query, we first find the lowest range $[a..b]$ in the conceptual range tree that completely contains $[p..q]$. Let $[a_1..b_1], [a_2..b_2], \ldots, [a_f..b_f]$ be the child ranges of $[a..b]$ in increasing order of left endpoints. We determine $\alpha, \beta \in [1..f]$, such that $[a_\alpha..b_\beta]$ covers $[p..q]$, and $\beta - \alpha$ is minimized. The query range can thus be decomposed into $[p..b_\alpha]$, $[a_{\alpha+1}..b_{\beta-1}]$ and $[a_\beta..q]$. The support for the second subrange is the same as that of He et al. [22]. For the third subrange, we employ the succinct index designed in Theorem 1 with $m = O(n \lg^* n)$, such that we can enumerate the nodes in $F_{a_\beta,q}$ that are on the query path in increasing order of weights. For each of them, we use the auxiliary data structures for the ball-inheritance problem to find the corresponding node in T. This procedure is terminated if the weight of the current node is above q. The support for the first subrange is similar.

References

1. Alon, N., Schieber, B.: Optimal preprocessing for answering on-line product queries. Tech. rep., Tel Aviv University (1987)
2. Alstrup, S., Holm, J.: Improved algorithms for finding level ancestors in dynamic trees. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 73–84. Springer, Heidelberg (2000)
3. Barbay, J., He, M., Munro, J.I., Satti, S.R.: Succinct indexes for strings, binary relations and multilabeled trees. ACM Transactions on Algorithms 7(4), 52 (2011)
4. Bille, P.: A survey on tree edit distance and related problems. Theor. Comput. Sci. 337(1-3), 217–239 (2005)
5. Bringmann, K., Larsen, K.G.: Succinct sampling from discrete distributions. In: STOC, pp. 775–782 (2013)

6. Brodal, G.S., Davoodi, P., Srinivasa Rao, S.: Path minima queries in dynamic weighted trees. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 290–301. Springer, Heidelberg (2011)
7. Brodal, G.S., Davoodi, P., Rao, S.S.: On space efficient two dimensional range minimum data structures. Algorithmica 63(4), 815–830 (2012)
8. Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal range searching on the RAM, revisited. In: Symposium on Computational Geometry, pp. 1–10 (2011)
9. Chazelle, B.: Computing on a free tree via complexity-preserving mappings. Algorithmica 2, 337–361 (1987)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
11. Demaine, E.D., Landau, G.M., Weimann, O.: On cartesian trees and range minimum queries. Algorithmica 68(3), 610–625 (2014)
12. Demaine, E.D., López-Ortiz, A.: A linear lower bound on index size for text retrieval. J. Algorithms 48(1), 2–15 (2003)
13. Farzan, A., Munro, J.I.: A uniform paradigm to succinctly encode various families of trees. Algorithmica 68(1), 16–40 (2014)
14. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput. 40(2), 465–492 (2011)
15. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees, with applications. SIAM J. Comput. 14(4), 781–798 (1985)
16. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. ACM Transactions on Algorithms 2(4), 510–534 (2006)
17. Golynski, A.: Optimal lower bounds for rank and select indexes. Theor. Comput. Sci. 387(3), 348–359 (2007)
18. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. 13(2), 338–355 (1984)
19. He, M., Munro, J.I., Satti, S.R.: Succinct ordinal trees based on tree covering. ACM Transactions on Algorithms 8(4), 42 (2012)
20. He, M., Munro, J.I., Zhou, G.: Path queries in weighted trees. In: Asano, T., Nakano, S.-I., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 140–149. Springer, Heidelberg (2011)
21. He, M., Munro, J.I., Zhou, G.: A framework for succinct labeled ordinal trees over large alphabets. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 537–547. Springer, Heidelberg (2012)
22. He, M., Munro, J.I., Zhou, G.: Succinct data structures for path queries. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 575–586. Springer, Heidelberg (2012)
23. Kaplan, H., Shafrir, N.: Path minima in incremental unrooted trees. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 565–576. Springer, Heidelberg (2008)
24. King, V.: A simpler minimum spanning tree verification algorithm. Algorithmica 18(2), 263–270 (1997)
25. Krizanc, D., Morin, P., Smid, M.H.M.: Range mode and range median queries on lists and trees. Nord. J. Comput. 12(1), 1–17 (2005)
26. Miltersen, P.B.: Lower bounds on the size of selection and rank indexes. In: SODA, pp. 11–12 (2005)

27. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. 31(3), 762–776 (2001)
28. Patil, M., Shah, R., Thankachan, S.V.: Succinct representations of weighted trees supporting path queries. J. Discrete Algorithms 17, 103–108 (2012)
29. Pettie, S.: An inverse-ackermann type lower bound for online minimum spanning tree verification. Combinatorica 26(2), 207–230 (2006)
30. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms 3(4) (2007)
31. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. 26(3), 362–391 (1983)