Vol.2010-AL-131 No.8
2010/9/22

Constant Time Generation of Trees with Degree Bounds

Bingbing Zhuang[†1] and Hiroshi Nagamochi[†1]

Given a number n of vertices, a lower bound d on the diameter, and a capacity function $\Delta(k) \geq 2$, $k = 0, 1, \ldots, \lfloor n/2 \rfloor$, we consider the problem of enumerating all unrooted trees T with exactly n vertices and a diameter at least d such that the degree of each vertex with distance k from the center of T is at most $\Delta(k)$. We give an algorithm that generates all such trees without duplication in $O(1)$-time delay per output in the worst case using $O(n)$ space.

1. Introduction

The problem of enumerating (i.e., listing) all graphs with bounded size is one of the most fundamental issues in graph theory. Time delay of an enumeration algorithm is a time bound between two consecutive outputs. Enumerating graphs with a polynomial time delay would be rather easy since we can examine the whole structure of the current graph anytime. However, algorithms with a constant time delay in the worst case is a hard target to achieve without a full understanding of the graphs to be enumerated, since not only the difference between two consecutive outputs is required to be $O(1)$, but also any operation for examining symmetry and identifying the edges/vertices to be modified to get the next output needs to be executable in $O(1)$ time. One of the common ideas behind efficient enumeration algorithms (e.g.,[5),6),8)]) is to define a unique representation for each graph in a graph class as its "parent," which induces a rooted tree that connects all graphs in the class, called the *family tree* $\mathcal{F}$, where each node in $\mathcal{F}$ corresponds to a graph in the class. Then all graphs in the class will be enumerated one by one according to the depth-first traversal of the family tree $\mathcal{F}$. However, the crucial point to attain an $O(1)$-time delay is to find a "good" parent which enables us to generate each of the children from a graph in $O(1)$ time.

Enumeration of restricted graphs or graphs with configurations has many applications in various fields such as machine learning and chemoinformatics. Enumeration of trees and outerplanar graphs can be used for many purposes including the inference of structures of chemical compounds[2),4)]. For example, the alkane molecular family $\{\mathtt{C}_n\mathtt{H}_{2n+2} \mid n \geq 1\}$ is one of the most fundamental classes of tree-like compounds, where each alkane contains only single bonds (either C-C or C-H bonds). Aringhieri et al.[1)] designed an algorithm that generates all alkane isomers for a given n in $O(n^4)$-time delay on average.

Our research group has been developing algorithms for enumerating chemical graphs that satisfy given various constraints[2)–4)]. We have designed efficient branch-and-bound algorithms for enumerating tree-like chemical graphs[2),4)], which are based on the tree enumeration algorithm[6)], and implementations of these algorithms are available on our web server[⋆1]. Our algorithms can enumerate all alkane isomer in $O(n^2)$-time delay on average[2)], improving the $O(n^4)$-time delay[1)].

Several algorithms to generate all trees with n vertices without repetition have been already known. The best algorithm[9)] runs in time proportional to the number of trees, i.e., the time delay is $O(1)$ on average. Nakano and Uno[7)] gave an $O(1)$-time delay algorithm to generate all trees with exactly n vertices and diameter d without repetition. However, for applications to chemical graph enumerations, we wish to use an efficient algorithm to generate trees with bounded degrees, because each vertex corresponding to an atom in a chemical graph has a small and fixed degree.

In this paper, given a number n of vertices, a lower bound d on the diameter, and a capacity function $\Delta(k) \geq 2$, $k = 0, 1, \ldots, \lfloor n/2 \rfloor$, we consider the problem of enumerating all unrooted trees T with exactly n vertices and a diameter at least d such that the degree of each vertex with distance k from the center of T is at most $\Delta(k)$. We give an algorithm that generates all such trees without duplication in $O(1)$-time delay per output in the worst case using $O(n)$ space. For example, alkane isomers $\mathtt{C}_n\mathtt{H}_{2n+2}$ can be regarded as unrooted trees with exactly

†1 Graduate School of Informatics, Kyoto University

⋆1 http://sunflower.kuicr.kyoto-u.ac.jp/tools/enumol/

n carbon atoms (neglecting hydrogen atoms) such that the degree of each vertex is at most four. Hence our result implies that all alkane isomers $\mathtt{C}_n\mathtt{H}_{2n+2}$ can be generated in $O(1)$-time delay in the worst case without duplication, which is an improvement over the $O(n^2)$-time delay on average[2].

2. Preliminaries

For two sequences A and B over a set of elements for which a total order is defined, let $A > B$ mean that A is lexicographically larger then B, and let $A \geq B$ mean that $A > B$ or $A = B$. Let $A \sqsupset B$ mean that B is a prefix of A and $A \neq B$, and let $A \gg B$ mean that $A > B$ but B is not a prefix of A. Let $A \sqsupseteq B$ mean that $A \sqsupset B$ or $A = B$, i.e., B is a prefix of A.

A graph stands for a simple undirected graph, which is denoted by a pair $G = (V, E)$ of a vertex set V and an edge set E. The set of vertices and the set of edges of a given graph G are denoted by $V(G)$ and $E(G)$, respectively. The *degree* $deg(v; G)$ of a vertex v in a graph G is the number of neighbours of v in G. A path is a sequence of distinct vertices $(v_0, v_1, \ldots, v_k)$ such that (v_{i-1}, v_i) is an edge for $i = 1, 2, \ldots, k$. The *length* of a path is the number of edges in the path. The distance between a pair of vertices u and v is the minimum length of a path between u and v. The *diameter* of G is the maximum distance between two vertices in G.

Unrooted Trees A tree (unrooted tree) is a connected graph without cycles. For two vertices u and v in a tree, let $P_T(u, v)$ be the unique path that connects u and v in T. In an unrooted tree, there are at most two vertices the maximum distance from which to other vertices is minimized. If such a vertex v is unique (i.e., the diameter of T is even), then we call v the *center* of T, and define the depth of a vertex u to be the distance from u to the center. On the other hand, if there are two such vertices v and v' (i.e., the diameter of T is odd), then we call the (v, v') the *center* of T, and define the *depth* $dep(u; T)$ of a vertex u to be the distance from u to the endvertices of the center, i.e., the length of the path from u to the center (v, v') including the edge (v, v').

Let $\Delta : [0, \lfloor n/2 \rfloor] \to [2, n-1]$ denote a capacity function, where $\Delta(k)$ is an upper bound on the degree of a vertex v with depth k in an unrooted tree. An unrooted tree T is called Δ-*bounded* if

$$2 \leq deg(r; T) \leq \Delta(0) \text{ and } deg(v; T) \leq \Delta(dep(v; T)) \ \forall\, v \in V(T) - \{r\}. \quad (1)$$

In this paper, we show the following result.

Theorem 1 For an integers $n \geq 3$ and $d \leq n$, and a capacity function $\Delta : [0, \lfloor n/2 \rfloor] \to [2, n-1]$, all Δ-bounded unrooted trees with exactly n vertices and a diameter at least d can be generated in $O(1)$-time delay in the worst case using $O(n)$ space after an $O(n)$-time initialization.

Let $\mathcal{T}_{odd}(n, \Delta)$ (resp., $\mathcal{T}_{even}(n, \Delta)$) denote the set of all Δ-bounded unrooted trees with exactly n vertices and an odd (resp., even) diameter.

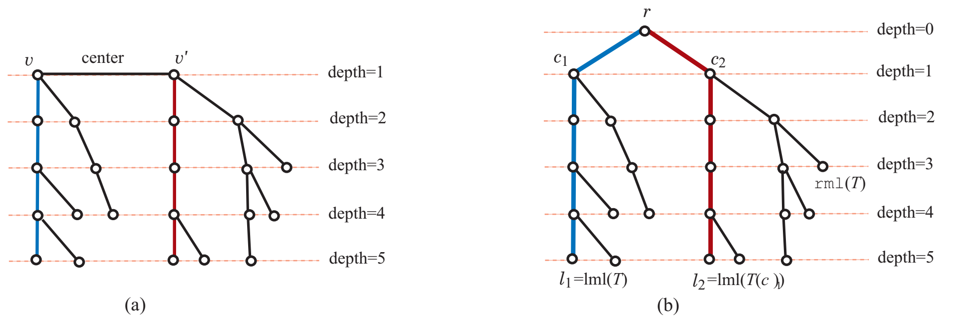

Fig. 1 (a) A tree T' with an odd diameter; (b) the tree T obtained from T' by subdividing the center (v, v') with root r.

Unrooted Rooted Trees We represent unrooted trees as "rooted trees." A *rooted* tree is a tree with one vertex r designated as its root. If $P_T(r, v)$ has exactly k edges then we say that the *depth* $dep(v; T)$ of v is k. The *parent* of $v \neq r$ is its neighbour on $P_T(r, v)$, and the ancestors of $v \neq r$ are the vertices on $P_T(r, v)$. The parent of the root r and the ancestors of r are not defined. We say that if v is the parent of u then u is a child of v, and if v is an ancestor of u then u is a descendant of v. A *leaf* is a vertex that has no child. Note that $P_T(r, v)$ denotes the set of all ancestors of a vertex v in a rooted tree T, where $v \in P_T(r, v)$.

Now we show how to convert the problem of generating unrooted trees in

$\mathcal{T}_{odd}(n,\Delta) \cup \mathcal{T}_{even}(n,\Delta)$ to a problem of generating rooted trees in some classes. Given a capacity function $\Delta : [0, \lfloor n/2 \rfloor] \to [2, n-1]$, let us call a rooted tree T Δ-*bounded* if it satisfies (1).

We call an rooted tree T *centerized* if T has an even diameter and r is the center of T, i.e., there are two children c_1 and c_2 of the root r such that the subtrees T_i at c_i, $i = 1, 2$ attain $dep(T_1) = dep(T_2) = dep(T) - 1$. Let $\mathcal{RT}(n,\Delta)$ denote the set of all Δ-bounded centerized trees. Let $\mathcal{T}^{+}_{odd}(n,\Delta)$ denote the set of trees obtained from each tree $T \in \mathcal{T}_{odd}(n,\Delta)$ by subdividing the center (v, v') with a root r (see Fig. 1(a)-(b)). It is a simple matter to see by definition that the next lemma holds.

Lemma 2 (i) For a given integer $n \geq 3$, $\mathcal{T}_{even}(n,\Delta)$ is given by $\mathcal{RT}(n,\Delta)$.
(ii) For a given integer $n \geq 2$, $\mathcal{T}^{+}_{odd}(n,\Delta)$ is given by $\mathcal{RT}(n+1,\Delta)$, where $\Delta(0) = 2$.

In what follows, we consider only how to generate rooted trees in $\mathcal{RT}(n,\Delta)$.

Ordered Trees Rooted trees are then represented as "ordered trees." An *ordered tree* (o-tree, for short) is a rooted tree with a left-to-right ordering specified for the children of each vertex. For an o-tree T and a vertex in T, let $T(v)$ denote the ordered subtree induced from T by the set of v and descendants of v, preserving the left-to-right ordering for the children of each vertex.

For an o-tree T', a leaf v in T' is called the *leftmost* (resp., *rightmost*) leaf if v is a descendant of the leftmost (resp., rightmost) child of any ancestor of v in T'. Let $\mathrm{lml}(T')$ (resp., $\mathrm{rml}(T')$) denote the leftmost (resp., rightmost) leaf in an o-tree T'. See Fig. 1(b).

Let T be an o-tree with n vertices, and $(v_1, v_2, \ldots, v_n)$ be the list of the vertices of T in preorder, i.e., vertices are indexed in the order of DFS. For two vertices $u = v_i$ and $v = v_j$, we write $u >_T v$ if $i < j$. Consider two vertices $u = v_i$ and $v = v_j$, $i \leq j$ in T. Let $[u, v]$ denote the set of all vertices $v_{i'}$ with $i \leq i' \leq j$, and let $T[u, v]$ denote the graph induced from T by the vertex set $[u, v]$. Also let $\mathrm{lca}(u, v)$ denote the least common ancestor of u and v in T. Let $\mathrm{lca}_{\mathtt{L}}(u, v)$ denote the child w of $\mathrm{lca}(u, v)$ such that $w \in P_T(r, u)$, where we let $\mathrm{lca}_{\mathtt{L}}(u, v) = u$ if $\mathrm{lca}(u, v) = u$. Similarly, $\mathrm{lca}_{\mathtt{R}}(u, v)$ denotes the child w' of $\mathrm{lca}(u, v)$ such that $w' \in P_T(r, u)$, where we let $\mathrm{lca}_{\mathtt{R}}(u, v) = v$ if $\mathrm{lca}(u, v) = v$. We denote the children of the root r in an o-tree by $c_1, c_2, \ldots, c_p$ from left to right. Let $\mathcal{OT}(T)$ denote the set of all o-trees obtained from a rooted tree T.

3. Left-heavy Trees

Since all o-trees in $\mathcal{OT}(T)$ of the same tree T are isomorphic, we choose a particular o-tree as the representative of T. For this, we use "left-heavy trees"[7]. For an o-tree T', we define the *depth sequence* $L(T')$ to be

$$L(T') = [dep(v_1; T'), dep(v_2; T'), \ldots, dep(v_n; T')].$$

If $L(T') > L(T'')$ for two ordered trees T' and T'', then we say that $L(T')$ is *heavier* than $L(T'')$. An o-tree $T' \in \mathcal{OT}(T)$ of a rooted tree T is called *left-heavy* tree if $L(T') \geq L(T'')$ holds for all o-trees $T'' \in \mathcal{OT}(T)$. For two vertices v_i and v_j, $i \leq j$, let $L_{i,j}(T') = [dep(v_i; T'), dep(v_{i+1}; T'), \ldots, dep(v_j; T')]$. It is known that left-heavy trees can be characterized as follows.

Lemma 3 [7] An o-tree $T' \in \mathcal{OT}(T)$ is the left-heavy tree of a rooted tree T if and only if, for a non-root vertex v and its immediate right sibling v' of v (if any) in an o-tree T', it holds $L(T(v)) \geq L(T(v'))$.

For each rooted tree T, a left-heavy tree in $\mathcal{OT}(T)$ is unique up to the isomorphism with respect to the root. In what follows, we assume that unordered rooted trees are represented by left-heavy trees.

By definition of left-heavy trees, we can easily observe that the following inequality on depth also holds.

Lemma 4 For a non-root vertex v and its immediate right sibling v' of v (if any) in a left-heavy tree T, it holds $dep(T(v)) \geq dep(T(v'))$.
In particular, $dep(T(c_1)) = dep(T(c_1)) \geq \cdots \geq dep(T(c_p))$ holds for the children $c_1, c_2, \ldots, c_p$ of the root r in a left-heavy and ceterized tree T. We call a left-heavy and ceterized T *distinguished* if, for each $i = 1, 2$, the number of leaves with the maximum depth in $T(c_i)$ is 1 (i.e., no other leaf than $\mathrm{lml}(T(c_i))$ attains $dep(T(c_i))$).

We consider how to add a new leaf along the rightmost path $P_T(r, \mathrm{rml}(T))$ of a left-heavy tree T so that the resulting o-tree remains left-heavy. This problem has been solved by Uno and Nakano[6]. We here use another solution "competitors" proposed in our companion paper[10], since "competitors" are easier to handle the case where some left part of a left-heavy tree may change.

For an o-tree T and a vertex u in T, let $T+(u,v)$ denote the o-tree obtained from T by appending a new vertex v at u as the rightmost child of u. A vertex u in a left-heavy tree T is called *valid* if $T+(u,v)$ remains left-heavy. Let v be a vertex in an o-tree T. For a descendant v_i of v in T, we define the *pre-sequence* $\mathrm{ps}(v,v_i)$ of v_i to v to be $L_{k,i-1}(G)=[dep(v_k;T), dep(v_{k+1};T),\ldots,dep(v_{i-1};T)]$ for the child v_k of v such that v_k is an ancestor of v_i. For a vertex v_i and a vertex v_j with $j<i$ incomparable v_i, we call v_j *pre-identical* to v_i if $\mathrm{ps}(v,v_j)=\mathrm{ps}(v,v_i)$ holds, and $\mathrm{lca}_{\mathtt{L}}(v_j,v_i)$ is the immediate right sibling $\mathrm{lca}_{\mathtt{R}}(v_j,v_i)$ for $v=\mathrm{lca}(v_j,v_i)$[10]. We define the *competitor* of a vertex v_i to be the vertex v_j pre-identical to v_i which has the smallest index j $(<i)$ among all vertices pre-identical to v_i. A vertex v_i has no competitor if no vertex v_j, $j<i$ is pre-identical to v_i.

Lemma 5 [10] Let T be a left-heavy tree, let $u_0,u_1,\ldots,u_q(=\mathrm{rml}(T))$ denote the rightmost path of T. Then there is an index h^* such that a vertex u_i is valid if and only if $0\le i\le h^*$. Moreover such an index h^* is determined as follows.
(i) u_q has no competitor: Then $h^*=q$.
(ii) u_q has a competitor v_j: Let v_h be the parent of the vertex v_{j+1} next to v_j in T. Then $h^*=dep(v_h;T)$.

Let us call such a vertex v_{h^*} the *lowest valid ancestor* of u_q in T. By maintaining vertices $\{v_1,v_2,\ldots,v_n\}$ in an array and the current tree T in a linked data structure, we can compute v_{h^*} from u_q in $O(1)$ time.

We review how to compute competitors. For each vertex v_i, $i=1,2,\ldots,n$ in this order, we can set the competitor of a vertex v_i to be the vertex v_j, $j<i$ which satisfies one of the next cases holds, where we also compute $\mathrm{lca}(v_j,v_i)$ and $\mathrm{lca}_{\mathtt{R}}(v_j,v_i)$:

(a) $i\ge 2$ and the previous vertex v_{i-1} of v_i has a competitor v_{j-1} and it holds $\mathrm{lca}(v_j,v_i)=\mathrm{lca}(v_{j-1},v_{i-1})$, where $dep(v_{i-1};T)=dep(v_{j-1};T)$ holds: Then the competitor of v_i is given by v_j. We set $\mathrm{lca}(v_j,v_i):=\mathrm{lca}(v_{j-1},v_{i-1})$ and $\mathrm{lca}_{\mathtt{R}}(v_j,v_i):=\mathrm{lca}_{\mathtt{R}}(v_{j-1},v_{i-1})$.

(b) v_i has no such previous vertex v_{i-1} in case (a), and v_i has a left sibling v_j: Then the competitor of v_i is given by v_j. We set $\mathrm{lca}(v_j,v_i)$ to be the parent of v_i and $\mathrm{lca}_{\mathtt{R}}(v_j,v_i):=v_i$.

Lemma 6 [10] In a left-heavy tree T, the competitor of vertex v_i is correctly obtained in cases (a) and (b), if any, if the competitors of all vertices v_t, $t<i$ have been obtained.

In case (a), whether $\mathrm{lca}(v_i,v_j)=\mathrm{lca}(v_{i-1},v_{j-1})$ or not can be tested without knowing the value of $\mathrm{lca}(v_i,v_j)$. For this, we use $\mathrm{lca}(v_{i-1},v_{j-1})$ and $\mathrm{lca}_{\mathtt{R}}(v_{i-1},v_{j-1})$ as follows: $\mathrm{lca}(v_i,v_j)=\mathrm{lca}(v_{i-1},v_{j-1})$ if and only if $j<h$ and $dep(v_h;T)>dep(v_i;T)$ for $v_h=\mathrm{lca}_{\mathtt{R}}(v_{i-1},v_{j-1})$. Hence we can determine the competitor of a new vertex v according to cases (a) and (b) in $O(1)$ time per operation of appending a new leaf.

4. Parent-trees of Δ-bounded Left-heavy Trees

In this section, we define the "parent-tree" of each left-heavy tree T in the class $\mathcal{RT}(n,\Delta)$ of Δ-bounded and centerized trees with n vertices, where $dep(T)-1=dep(T(c_1))=dep(T(c_2))$ holds. For ease of applications of the properties on left-heavy trees, we also define the "parent-tree" of each left-heavy tree in $\mathcal{RT}(n-1,\Delta)\cup\mathcal{RT}(n-2,\Delta)$ *with respect to* n so that the parent-child relationship over these classes forms a family tree $\mathcal{F}$. We will design an algorithm that visits all nodes in family tree $\mathcal{F}$ each in $O(1)$-time. However, we output only trees in $\mathcal{RT}(n,\Delta)$ during the traversal of $\mathcal{F}$.

For the leftmost and second leftmost children c_1 and c_2 of the root r in a left-heavy tree T, let ℓ_i, $i=1,2$ denote the leftmost leaf of the subtree $T(c_i)$ rooted at c_i. We call each vertex in $P_T(r,\ell_1)\cup P_T(r,\ell_2)$ a *core vertex*. Let $v_{\mathtt{last}}$ denote the non-core vertex with the largest preporder index in T.

For an even n, let P_{n-1} denote the o-tree obtained from a path with $n-1$ vertices by choosing its center as the root, and let $P_{n-1}+i$ be obtained from P_{n-1} by adding a new leaf at the ith vertex v_i, $i\in[0,n/2]$. Let $\mathcal{RT}([n,n-2],\Delta)$ denote $\mathcal{RT}(n,\Delta)\cup\mathcal{RT}(n-1,\Delta)\cup\mathcal{RT}(n-2,\Delta)$.

For an odd (resp., even) n, we define the parent-tree of a left-heavy tree $T\in\mathcal{RT}([n,n-2],\Delta)-\{P_n\}$ (resp., $T\in\mathcal{RT}([n,n-2],\Delta)-\{P_{n-1}+i\mid 0\le i\le n/2\}$) with respect to n as follows.

(1) If $T\in\mathcal{RT}(n,\Delta)\cup\mathcal{RT}(n-1,\Delta)$, then the *parent-tree* $\mathcal{P}(T)$ of T is defined to be the o-tree $T-v_{\mathtt{last}}$ obtained from T by removing $v_{\mathtt{last}}$. For example, the parent-tree of T_1 with n vertices (reps., T_2 with $n-1$ vertices) is T_2 (resp.,

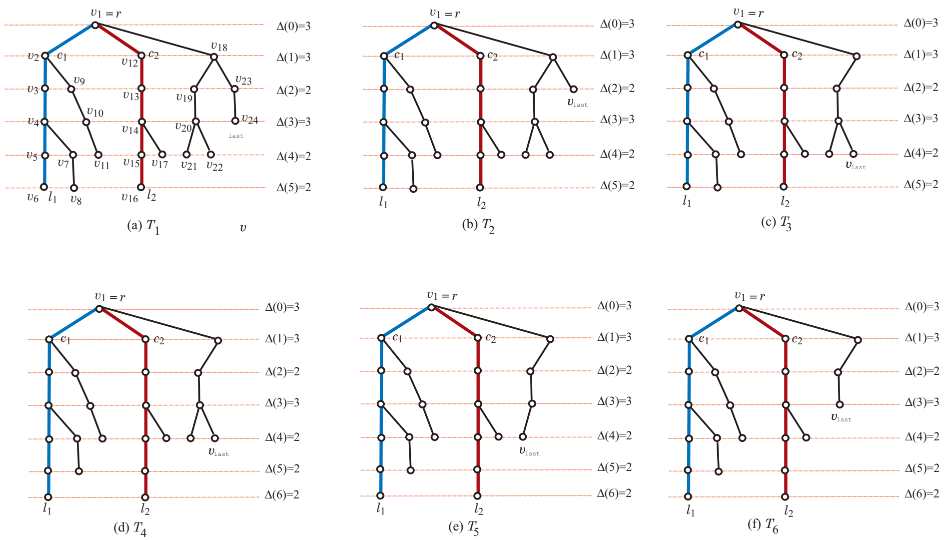

Fig. 2 Examples of left-heavy trees for $n = 24$, where $T_1, T_4 \in \mathcal{RT}(n, \Delta)$, $T_2, T_5 \in \mathcal{RT}(n-1, \Delta)$, and $T_3, T_6 \in \mathcal{RT}(n-2, \Delta)$, and T_{i+1} is the parent-tree of T_i, $i = 1, 2, \ldots, 5$.

T_3) in Fig. 2. The inequalities in Lemma 3 still hold in $T - v_{\texttt{last}}$, and hence $\mathcal{P}(T) = T - v_{\texttt{last}}$ remains left-heavy. Clearly $\mathcal{P}(T) = T - v_{\texttt{last}}$ remains Δ-bounded.

(2) If $T \in \mathcal{RT}(n-2, \Delta)$, then the *parent-tree* $\mathcal{P}(T)$ of T is defined to be the o-tree obtained from T by appending a new leaf to ℓ_i for each $i = 1, 2$. For example, the parent-tree of T_3 with $n-2$ vertices is T_4 in Fig. 2. The inequalities in Lemma 3 still hold in $\mathcal{P}(T) = (T + (\ell_1, v)) + (\ell_2, v')$, since the leftmost paths in $T(c_1)$ and $T(c_2)$ extend. $T - v_{\texttt{last}}$ remains left-heavy. Also $\mathcal{P}(T) = (T + (\ell_1, v)) + (\ell_2, v')$ remains Δ-bounded by $\Delta(dep(\ell_1; T)) = \Delta(dep(\ell_2; T)) \geq 2$.

Lemma 7 For each left-heavy tree $T \in \mathcal{RT}([n, n-2], \Delta) - (\{P_n\} \cup \{P_{n-1} + i \mid 0 \leq i \leq n/2\})$, the parent-tree $\mathcal{P}(T)$ with respect to n is a Δ-bounded left-heavy tree which belongs to $\mathcal{RT}([n, n-2], \Delta)$.

5. Generating Child-trees

A left-heavy tree T' is called a *child-tree* of a left-heavy tree T if T is the parent-

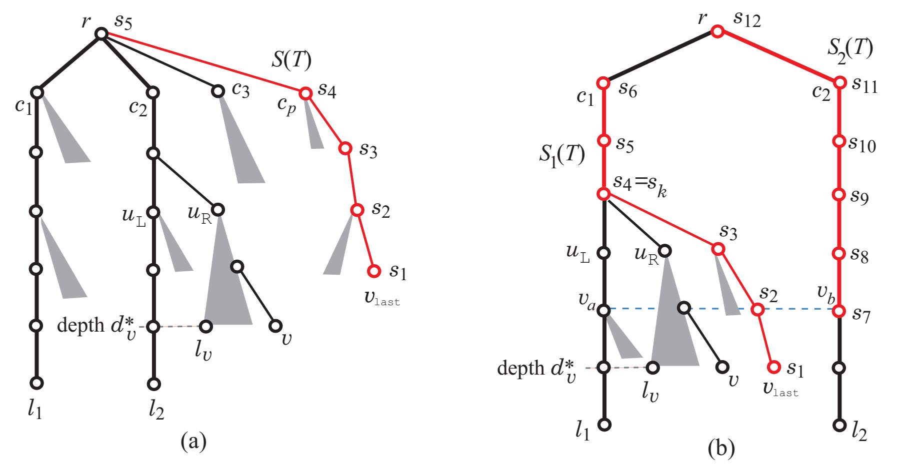

Fig. 3 (a) Spine $S(T) = \{s_1, s_2, \ldots, s_5\}$ in a tree T with $v_{\texttt{last}} \notin V(T(c_1))$; (b) spine $S(T) = S_1(T) \cup S_2(T) = \{s_1, s_2, \ldots, s_{11}\}$ in a tree T with $v_{\texttt{last}} \in V(T(c_1))$.

tree of T', where T' may not be Δ-bounded. A vertex v in T is called *unsaturated* if $deg(v; T) < \Delta(dep(v; T))$. In Sections 5.1 and 5.2, we first characterize the set of all child-tree of a left-heavy tree $T \in \mathcal{RT}([n, n-2], \Delta)$. In Section 5.3, we next describe an entire algorithm Enumerate for enumerating all left-heavy trees $T \in \mathcal{RT}([n, n-2], \Delta)$ by a recursive procedure Gen of generating all child-trees of a given a left-heavy tree $T \in \mathcal{RT}([n, n-2], \Delta)$.

5.1 Appending a Leaf to $T \in \mathcal{RT}(n-1, \Delta) \cup \mathcal{RT}(n-2, \Delta)$

Let $T \in \mathcal{RT}(n-1, \Delta) \cup \mathcal{RT}(n-2, \Delta)$. By definition of parent-trees, any child-tree T' of T has n or $n-1$ vertices, and T is obtained from T' by removing the non-core vertex $v_{\texttt{last}}(T')$ with the largest index in T'. Thus, T' is obtained from T by appending a new leaf (u, v) so that $T + (u, v)$ is left-heavy and Δ-bounded. We here consider when $T + (u, v)$ is left-heavy, i.e., (i) u is a valid vertex in T and (ii) v is the non-core vertex $v_{\texttt{last}}(T + (u, v))$ with the largest index in $T + (u, v)$.

We define the *spine* $S(T)$ of a left-heavy tree $T \in \mathcal{RT}(n-1, \Delta) \cup \mathcal{RT}(n-2, \Delta)$ as the set of vertices u at which appending a new leaf (u, v) results in an o-tree $T + (u, v)$ such that v is the non-core vertex $v_{\texttt{last}}(T + (u, v))$ with the largest index in $T + (u, v)$. By definition of $S(T)$, we observe the next.

Lemma 8 For each left-heavy tree $T \in \mathcal{RT}(n-1,\Delta) \cup \mathcal{RT}(n-2,\Delta)$, the o-tree T' obtained by appending a new leaf (u,v) is a child-tree of T if and only if u is a valid vertex in $S(T)$.

We show how to find all valid vertices in $S(T)$ in the following two cases.

Case-1: The non-core vertex $v_{\mathtt{last}}$ with the largest index in T does not belong to the subtree $T(c_1)$: In this case, $S(T) = (s_1 = v_{\mathtt{last}}, s_2, \ldots, s_m = r)$ is given as the path from the last non-core vertex $v_{\mathtt{last}}$ to the root r. See Fig. 3(a). Since $S(T)$ is the rightmost path of T, all valid vertices in $S(T)$ can be identified by the lowest valid ancestor $v_{h^*} = s_i$ $(1 \le i \le m)$ of $v_{\mathtt{last}}$, and v_{h^*} can be computed in $O(1)$ time in the same manner in Lemma 5 using the competitor of $v_{\mathtt{last}}$.

Case-2: $v_{\mathtt{last}}$ belongs to the subtree $T(c_1)$ (i.e., r has only two children and the subtree $T(c_2)$ is a path): Let v_a denote the lowest core vertex in $T(c_1)$ with $deg(v_a) \ge 3$, and let v_b be the vertex v_b in $T(c_2)$ with $dep(v_b;T) = dep(v_a;T)$. See Fig. 3(b). Then $S(T)$ consists of two sequences $S_1(T)$ and $S_2(T)$, where $S_1(T) = (s_1 = v_{\mathtt{last}}, s_2, \ldots, s_t = c_1)$ is obtained by visiting the path $P_T(v_{\mathtt{last}}, c_1)$ from $v_{\mathtt{last}}$ to c_1, and $S_2(T) = (s_{t+1} = v_b, s_{t+2}, \ldots, s_m = r)$ by visiting $P_T(v_b, r)$ from v_b to r, where $S_1(T)$ is followed by $S_2(T)$ in $S(T)$. We easily see that all vertices in $S_2(T)$ are always valid. For the vertex $s_k = \mathrm{lca}(\ell_1, v_{\mathtt{last}})$, all vertices $s_k, s_{k+1}, \ldots, s_t \in S_1(T)$ are also valid. The valid vertices in $\{s_1, s_2, \ldots, s_{k-1}\}$ can be determined by applying Lemma 5 to the subtree $T(s_K)$. Thus the lowest valid ancestor $v_{h^*} = s_i$ $(1 \le i \le k-1)$ of $v_{\mathtt{last}}$ in $T(s_K)$ can be computed in $O(1)$ time in the same way using the competitor of $v_{\mathtt{last}}$.

Let $\mathrm{spn}(v)$ denote the parent $v' = s_{i+1} \in S(T)$ of a vertex $v = s_i \in S(T)$, where $\mathrm{spn}(r) = \emptyset$.

5.2 Shortening Depth of $T \in \mathcal{RT}(n,\Delta)$

Let $T \in \mathcal{RT}(n,\Delta)$ be a left-heavy tree. By definition of parent-trees, T has at most one child-tree, which is given by $T - \{\ell_1, \ell_2\}$. Note that $T - \{\ell_1, \ell_2\}$ can be a child-tree of T only when T is distinguished, since the parent-tree of any tree $T \in \mathcal{RT}(n-2,\Delta)$ with respect to n is distinguished. In fact, for a distinguished left-heavy tree $T \in \mathcal{RT}(n,\Delta)$, $T - \{\ell_1, \ell_2\}$ is a child-tree of T if and only if $T - \{\ell_1, \ell_2\}$ is left-heavy. We show how to examine the left-heaviness of $T - \{\ell_1, \ell_2\}$ in $O(1)$ time. By definition, we first observe the following case.

Lemma 9 Let $T \in \mathcal{RT}(n,\Delta)$ be a distinguished left-heavy tree. If $T' = T - \{\ell_1, \ell_2\}$ remains distinguished, then T' is left-heavy.

We next show how to check whether T can have a non-distinguished child-tree, i.e., whether the o-tree $T - \{\ell_1, \ell_2\}$ remains left-heavy or not. Let $X_1 = V(T) - \{r\}$ and $X_2 = V(T) - (V(T(c_1) \cup \{r\})$. For each vertex $v \in X_i$, $i = 1, 2$, we define $\mathrm{state}_i(v)$ as follows.

We first define $\mathrm{state}_1(v)$, $v \in X_1$. For the leftmost leaf $\ell_1 = \mathrm{lml}(T(c_1))$ of $T(c_1)$ and a vertex $v \in X_1$, let us denote $u_{\mathtt{L}} = \mathrm{lca}_{\mathtt{L}}(\ell_1, v)$ and $u_{\mathtt{R}} = \mathrm{lca}_{\mathtt{R}}(\ell_1, v)$ (see Fig. 3(b)). We compare subtree $T(u_{\mathtt{L}})$ at $u_{\mathtt{L}}$ and subtree $T_v^{\mathtt{R}} = T[u_{\mathtt{R}}, v]$ in the following way. The depth $dep(T_v^{\mathtt{R}})$ of $T_v^{\mathtt{R}}$ is determined by its leftmost leaf $\ell_v = \mathrm{lml}(T_v^{\mathtt{R}})$. Hence the depth $d_v^* = dep(\ell_v; T)$ of ℓ_v in T is given by

$$d_v^* = dep(\mathrm{lca}(\ell_1, v); T) + dep(T_v^{\mathtt{R}}) + 1.$$

If $T(c_1)$ has a non-core vertex $u' <_T \ell_v$ with $dep(u';T) > d_v^*$ or $u_{\mathtt{R}}$ is not the second leftmost child of $\mathrm{lca}(\ell_1, v)$, then we let $\mathrm{state}_1(v) = \emptyset$. Then the set of vertices $u' <_T \ell_v$ with $dep(u';T) > d_v^*$ consists of core vertices in $T(c_1)$; i.e., it is given by $\{u' \in P_T(r, \ell_1) \mid dep(u';T) > d_v^*\}$. Let L^* be the sequence of depth of the vertices in $\{u' \in P_T(r, \ell_1) \mid dep(u';T) > d_v^*\}$, and $L(T(u_{\mathtt{L}})) - L^*$ denote the sequence obtained from $L(T_v^{\mathtt{L}})$ by eliminating the entries in L^*. We compare the label sequences $L(T(u_{\mathtt{L}})) - L^*$ and $L(T_v^{\mathtt{R}})$, and define

$$\mathrm{state}_1(v) = \begin{cases} (d_v^*, \sqsupseteq) & \text{if } L(T(u_{\mathtt{L}})) - L^* \sqsupseteq L(T_v^{\mathtt{R}}) \\ (d_v^*, \gg) & \text{if } L(T(u_{\mathtt{L}})) - L^* \gg L(T_v^{\mathtt{R}}) \\ (d_v^*, <) & \text{if } L(T(u_{\mathtt{L}})) - L^* < L(T_v^{\mathtt{R}}). \end{cases} \quad (2)$$

We define $\mathrm{state}_2(v)$, $v \in X_2$ analogously with state_1 (see Fig. 3(a)). For $\ell_2 = \mathrm{lml}(T(c_2))$ and a vertex $v \in X_2$, let us denote $u_{\mathtt{L}} = \mathrm{lca}_{\mathtt{L}}(\ell_2, v)$ and $u_{\mathtt{R}} = \mathrm{lca}_{\mathtt{R}}(\ell_2, v)$. Let $T_v^{\mathtt{R}} = T[u_{\mathtt{R}}, v]$ and $d_v^* = dep(\mathrm{lca}(\ell_2, v); T) + dep(T_v^{\mathtt{R}}) + 1$. If $T(c_2)$ has a non-core vertex $u' <_T \ell_v$ with $dep(u';T) > d_v^*$ or $u_{\mathtt{R}}$ is not the second leftmost child of $\mathrm{lca}(\ell_2, v)$, then we let $\mathrm{state}_2(v) = \emptyset$. Otherwise define $\mathrm{state}_2(v)$ by (2).

Lemma 10 Let $T \in \mathcal{RT}(n,\Delta)$ be a distinguished left-heavy tree such that $T' = T - \{\ell_1, \ell_2\}$ is not distinguished. Let $\tilde{d}$ be the current depth $dep(\ell_1;T) = dep(\ell_2;T)$ of $T(c_1)$ and $T(c_2)$. Then $T - \{\ell_1, \ell_2\}$ is left-heavy if and only if, for each $i = 1, 2$, $T(c_i)$ has no non-core vertex v such that $\mathrm{state}_i(v) = (\tilde{d} - 1, <)$.

When a child-tree T is generated from T by appending a new vertex $v_{\mathtt{last}}$

to $\hat{v}$ in the current tree T', state$_i$, $i = 1, 2$ are updated as follows. Let the preorder index of $v_{\mathtt{last}}$ in T be K, i.e., $v_{\mathtt{last}} = v_K$. If $v_K = \mathrm{lml}(T(c_i))$, then state$_i(v_K) := (dep(v_K; T), \sqsupseteq)$. Assume $v_K \neq \mathrm{lml}(T(c_i))$. If the second term in state$_i(v_{K-1})$ is "$\gg$" or "$<$," then state$_i(v_K) :=$ state$_i(v_{K-1})$. Assume state$_i(v_{K-1}) = (d^*, \sqsupseteq)$. Let $v_a = \mathrm{lca}(\ell_1, v_K)$, and h be the integer such that v_K appears as the hth vertex in $T^{\mathtt{R}}_{v_K}$. If vertex v_{a+h+1} belongs to $T^{\mathtt{R}}_{v_K}$, then set state$_i(v_K)$ to be $(d^*, <)$. Otherwise, we set state$_i(v_K)$ to be $(d^*, \gg)$ (resp., $(d^*, \sqsupseteq)$ and $(d^*, <)$) if $dep(v_{a+h+1}; T) > dep(v_K; T)$ (resp., $dep(v_{a+h+1}; T) = dep(v_K; T)$ and $dep(v_{a+h+1}; T) < dep(v_K; T)$).

When we remove ℓ_1 and ℓ_2 in the current tree $T' \in \mathcal{RT}(n, \Delta)$ and add a new non-core vertex to obtain a child-tree $T = T' - \{\ell_1, \ell_2\}$, we need to recompute the competitor of the current last non-core vertex $v_{\mathtt{last}}$ if $T - \{\ell_1, \ell_2\}$ is not distinguished. Only in this case, $v_{\mathtt{last}} \in X_1$ (resp., $v_{\mathtt{last}} \in X_2$) may have a pre-identical vertex in the subtree $T(c_1)$ (resp., $T(c_2)$) of $T = T' - \{\ell_1, \ell_2\}$. We can test whether $v_{\mathtt{last}}$ has a pre-identical vertex in such a subtree $T(c_i)$ or not by checking $state_i(v_{\mathtt{last}})$. Let $\hat{d} = dep(T)$.

(i) $v_{\mathtt{last}} \in X_1$: If $state_1(v_{\mathtt{last}}) = (\hat{d}, \sqsupseteq)$, then set the competitor of $v_{\mathtt{last}}$ to be the vertex $v_{k'}$ in $T(c_1)$ corresponding to $v_{\mathtt{last}}$, i.e., $\mathrm{lca}(\ell_1, v_{\mathtt{last}})$ is a core vertex in $T(c_1)$ and $v_{k'}$ is the $|V(T(\mathrm{lca}_{\mathtt{R}}(\ell_1, v_{\mathtt{last}})))|$th vertex in subtree $T(\mathrm{lca}_{\mathtt{L}}(\ell_1, v_{\mathtt{last}}))$.

(ii) $v_{\mathtt{last}} \in X_2$: If $state_2(v_{\mathtt{last}}) = (\hat{d}, \sqsupseteq)$, then set the competitor of $v_{\mathtt{last}}$ to be the vertex $v_{k'}$ in $T(c_2)$ corresponding to $v_{\mathtt{last}}$, i.e., $\mathrm{lca}(\ell_2, v_{\mathtt{last}})$ is a core vertex in $T(c_2)$ and $v_{k'}$ is the $|V(T(\mathrm{lca}_{\mathtt{R}}(\ell_2, v_{\mathtt{last}})))|$th vertex in subtree $T(\mathrm{lca}_{\mathtt{L}}(\ell_2, v_{\mathtt{last}}))$.

5.3 Entire Algorithm

We are ready to describe the entire algorithm except for showing how to efficiently find unsaturated vertices in the spine $S(T)$. Algorithm ENUMERATE constructs initial Δ-bounded left-heavy trees $T := P_n$ for an odd n and $T := P_{n-1}+i$, $i \in [0, n/2]$ for an even n before it invokes a recursive procedure GEN of generating child-trees.

Algorithm ENUMERATE(n, Δ, d)
Input: An integer $n \geq 3$, a capacity function $\Delta : [0, \lfloor n/2 \rfloor] \to [2, n-1]$, and an integer $d \leq n$.
Output: All Δ-bounded left-heavy trees with exactly n vertices and a diameter at least d.

  if n is odd **then** let T be path P_n rooted at its center; Initialize (T);
    GEN(T)
  else /* n is even */
    Let P_{n-1} be rooted at its center, where $(v_1 = r, v_2, \ldots, v_{n/2} = \ell_1)$
    denotes the path from $\ell_1 = \mathtt{lml}(P_{n-1})$ to the root r;
    for each unsaturated v_i (i.e., v_i with $\Delta(i-1) \geq 3$) **do**
      Let $T := P_{n-1}+(v_i, v)$ be the tree obtained from P_{n-1} by adding
      a new leaf (v_i, v) at v_i; Initialize (T); GEN(T)
    endfor
  endif

The above initialization takes $O(n)$ time. The next procedure GEN generates the child-trees T' of the current Δ-bounded left-heavy T, and calls GEN(T') to enumerate all descendants of T'. We output only trees T with exactly n vertices at every third depth $3a+1$ of recursive call at T during an execution of GEN. To attain an $O(1)$-time delay in the worst case, a generated tree $T \in \mathcal{RT}(n, \Delta)$ is output immediately before or after GEN(T) is executed if a is even (resp., odd).

Procedure GEN(T)
Input: A left-heavy tree $T \in \mathcal{RT}([n, n-2], \Delta)$.
Output: All descendants T'' of T such that T'' is Δ-bounded left-heavy trees with exactly n vertices and a diameter at least d.

  if $|V(T)| = n$ and the current depth of recursive calls is $3a+1$ for an
    even integer a **then** Output T
  endif;
  if $|V(T)| = n$, T is distinguished, $dep(T) \geq d+1$, and $T - \{\ell_1, \ell_2\}$ is
    left-heavy **then** $T' := T - \{\ell_1, \ell_2\}$; GEN$(T')$
  endif;
  if $|V(T)| = n - 2 or n - 1$ **then**
    for each unsaturated and valid vertex u in $S(T)$ **do**
      Let $T' := T + (u, v)$ be obtained from T by adding a new leaf (u, v) ;
      at u; GEN(T')

endfor
endif;
if $|V(T)| = n$ and the current depth of recursive calls is $3a+1$ for an odd integer a **then** Output T
endif

We have shown that each line of GEN for generating a child-tree T' can be performed in $O(1)$ time, except for how to find unsatureted vertices in the spine $S(T)$.

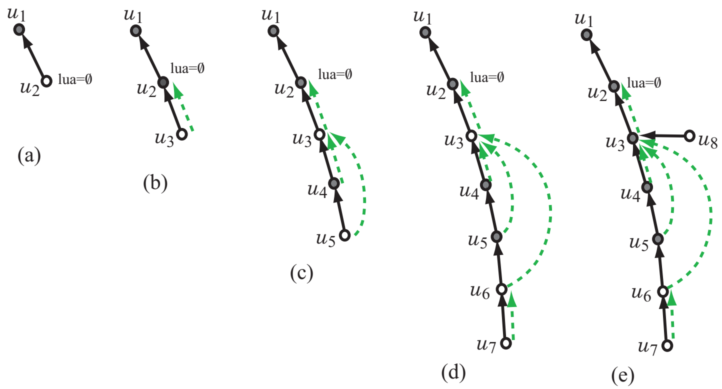

Fig. 4 Illustration for a process of appending new leafs, $u_2, u_3, \ldots, u_8$, where gray vertices and dashed arrows indicate saturated vertices and lua, respectively.

In the rest of the section, we briefly show how to find all valid and unsaturated vertices in the spine $S(T)$ for a left-heavy tree $T \in \mathcal{RT}(n-1, \Delta) \cup \mathcal{RT}(n-2, \Delta)$.

We consider Case-1, i.e., $v_{\texttt{last}}$ belongs to $T(c_1)$ (Case-2 can be treated analogously by applying the following argument to each of $S_1(T)$ and $S_2(T)$). We let lua(v) store the lowest unsaturated ancestor of v in $S(T)$, and lua$(v) = \emptyset$ mean that there is no unsaturated ancestor of v in $S(T)$, for the root lua$(r) = \emptyset$. When we search all valid and unsaturated vertices in the spine $S(T)$, we start with the lowest valid vertex u_{h^*}, which can be determined by the competitor of $v_{\texttt{last}}$ together with lua$(\hat{v})$. Recall that all vertices in $S(T)$ higher than u_{h^*} are valid by Lemma 5.

When a child-tree T' is generated from T by appending a new vertex v to $\hat{v}$, we need to update spn and lua (see Fig. 4), where lua(v) never changes since it is only updated once when v is newly added to the tree. Including the maintenance of all data values, we can find all unsaturated and valid vertices in the spine $S(T)$ in $O(1)$ time per each, where we omit the detail due to space limitation and procedures for updating the data can be found in a full version[11]. This shows that, given integers n and g and a capacity Δ, all left-heavy and centerized trees in $\mathcal{R}(n, \Delta)$ with a diameter at least d can be generated in $O(1)$ time delay using $O(n)$ space, proving Theorem 1

References

1) R. Aringhieri, P. Hansen, F. Malucelli, Chemical trees enumeration algorithms, 4OR: Quart. J. Oper. Resear., 1 (2003) 67-83.
2) H. Fujiwara, J. Wang, L. Zhao, H. Nagamochi, and T. Akutsu, Enumerating tree-like chemical graphs with given path frequency, J. Chem. Inf. Mod., 48, (2008) 1345-1357.
3) T. Imada, S. Ota, H. Nagamochi, and T. Akutsu, Enumerating stereoisomers of tree structured molecules using dynamic programming, LNCS 5878, (2008) 14-23.
4) Y. Ishida, L. Zhao, H. Nagamochi, and T. Akutsu, Improved algorithm for enumerating tree-like chemical graphs, Genome Informatics, 21, (2008) 53-64.
5) S. Nakano, Efficient generation of triconnected plane triangulations, Computational Geometry Theory and Applications, 27(2), (2004) 109-122.
6) S. Nakano and T. Uno, Efficient generation of rooted trees, NII Technical Report (NII-2003-005) (2003).
7) S. Nakano and T. Uno, Constant time generation of trees with specified diameter, LNCS 3353, (2004) 33-45.
8) S. Nakano and T. Uno, Generating colored trees, LNCS 3787, (2005) 249-260.
9) R. A. Wright, B. Richmond, A. Odlyzko, and B. D. McKay, Constant time generation of free trees, SIAM J. Comput., 15, (1986) 540-548.
10) B. Zhuang and H. Nagamochi, Enumerating rooted graphs with reflectional block structures, LNCS 6078, (2010) 49-60.
11) B. Zhuang and H. Nagamochi, Constant time generation of trees with degree bounds, Dept. of Applied Mathematics and Physics, Kyoto University, Technical Report 2010-006 (2010) http://www-or.amp.i.kyoto-u.ac.jp/members/nag/Technicalreport/TR2010-006.pdf