Discrete Mathematics, Algorithms and Applications

Bounding the Number of Reduced Trees, Cographs, and Series-Parallel Graphs by Compression

Takeaki Uno

National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan. uno@nii.jp

Ryuhei Uehara

School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp

Shin-ichi Nakano

Department of Computer Science, Faculty of Engineering, Gunma University, Gunma 376-8515, Japan. nakano@cs.gunma-u.ac.jp

Received Day Month Year
Accepted Day Month Year

We give an efficient encoding and decoding scheme for computing a compact representation of a graph in one of unordered reduced trees, cographs, and series-parallel graphs. The unordered reduced trees are rooted trees in which (i) the ordering of children of each vertex does not matter, and (ii) no vertex has exactly one child. This is one of basic models frequently used in many areas. Our algorithm computes a bit string of length $2\ell - 1$ for a given unordered reduced tree with $\ell \geq 1$ leaves in $O(\ell)$ time, whereas a known folklore algorithm computes a bit string of length $2n - 2$ for an ordered tree with n vertices. Note that in an unordered reduced tree $\ell \leq n < 2\ell$ holds. To the best of our knowledge this is the first such a compact representation for unordered reduced trees. From the theoretical point of view, the length of the representation gives us an upper bound of the number of unordered reduced trees with ℓ leaves. Precisely, the number of unordered reduced trees with ℓ leaves is at most $2^{2\ell-2}$ for $\ell \geq 1$. Moreover, the encoding and decoding can be done in linear time. Therefore, from the practical point of view, our representation is also useful to store a lot of unordered reduced trees efficiently. We also apply the scheme for computing a compact representation to cographs and series-parallel graphs. We show that each of cographs with n vertices has a compact representation in $2n - 1$ bits, and the number of cographs with n vertices is at most 2^{2n-1}. The resulting number is close to the number of cographs with n vertices obtained by the enumeration for small n that approximates $Cd^n/n^{3/2}$, where $C = 0.4126\cdots$ and $d = 3.5608\cdots$. Series-parallel graphs are well investigated in the context of the graphs of bounded treewidth. We give a method to represent a series-parallel graph with m edges in $\lceil 2.5285m - 2 \rceil$ bits. Hence the number of series-parallel graphs with m edges is at most $2^{\lceil 2.5285m-2 \rceil}$. As far as the authors know, this is the first non-trivial result about the number of series-parallel graphs. The encoding and decoding of the cographs and series-parallel graphs also can be done in linear time.

Keywords: Cograph; compact representation; counting; encoding/decoding scheme;

series-parallel graph; unordered reduce tree.

1. Introduction

Tree is one of basic models frequently used in various areas including searching for keys, modeling computation, and parsing a program. Since an explicit storage of a tree of a large size needs huge amount of memory, a compact representation is desired. A typical example is a trie structure that compresses a given huge word dictionary by taking the prefixes of words [11, Chapter 6.3]. In the area of data mining, the other basic models are also used to represent a tons of data having some specific structure. In this context, there are a lot of papers for encoding trees, plane graphs, and plane triangulations. For example, see [9,12,7] for trees, [5,10] for plane graphs, and [1,14] for maximal plane graphs, and referred papers.

In this paper, we first focus on the "unordered reduced" trees, in which the children of each vertex has no ordering and no vertex has exactly one child. The unordered reduced tree is one of important models from both theory and practice. We first give an efficient representation of an unordered reduced tree. The representation requires $2\ell - 2$ bits to represent a given unordered reduced tree with $\ell \geq 2$ leaves, whereas a known folklore representation uses a bit string of length $2n - 2$ which is a representation of an ordered tree with n vertices. Note that $\ell \leq n < 2\ell$ holds for any unordered reduced tree. Computation of the representation can be done efficiently; both of encoding and decoding can be done in $O(\ell)$ time. To the best of our knowledge this is the first non-trivial compact representation designed for unordered reduced trees. It is worth mentioning that we do not use the big-O notation. Hence the length of the representation gives us an upper bound of the number of unordered reduced trees; the number of unordered reduced trees with $\ell \geq 1$ leaves is bounded by $2^{2\ell-2}$.

We next apply the compact representation of unordered reduced tree to the other graph classes; cographs and series-parallel graphs.

For the last decades, many graph classes have been introduced [3]. Among them, cographs form one of basic graph classes. Since they have a simple recursive structure, the class is a subset of many important graph classes, and hence some intractable problems on general graphs become tractable on cographs. The simple recursive structure of a cograph can be represented by a canonical unordered reduced tree. Hence we can estimate that the number of cographs with n vertices is at most 2^{2n-1}. In the context of the implicit representation of the graph class, it is mentioned that the number of cographs is $2^{O(n \log n)}$ [13, Section 8.1]. We exponentially improve this upper bound, and we again mention that we do not use the big-O notation. According to The On-Line Encyclopedia of Integer Sequences (`http://oeis.org/A000084`), the number of cographs with n vertices is estimated as $Cd^n/n^{3/2}$, where $C = 0.4126\cdots$ and $d = 3.5608\cdots$. This value is obtained by the enumeration for small n, and our upper bound is close to this estimation. The encoding to a bit string from a given cograph and the decoding of the bit string to

the original cograph can be done in linear time.

The other graph class is series-parallel graphs. This is also one of basic graph classes, and this class is well investigated in the context of the graphs of bounded treewidth. However, recently, it is revealed that many data obtained from the bioinformatics area can be modeled in this graph class. For example, the E.coli metabolic network has treewidth 3 and more than 90% of pathways of several organisms are series-parallel graphs [4]. Therefore, the importance of this class increases more and more. As far as the authors know, there are no known non-trivial results about the compact representation designed for the class and the number of series-parallel graphs. We give a method to represent a series-parallel graph with m edges in $\lceil 0.528m \rceil$ bits. Hence the number of series-parallel graphs with m edges is at most $2^{\lceil 0.528m \rceil}$. The encoding and decoding of the series-parallel graphs also can be done in linear time.

The rest of the paper is organized as follows. In Section 2, we introduce some definitions. Section 3 gives the algorithms for an unordered reduced tree. Sections 4 and 5 describe compact representations for cographs and series-parallel graphs, respectively. Finally Section 6 is a conclusion.

2. Preliminaries

Let G be a graph. The degree of a vertex in G is the number of vertices adjacent to the vertex. A *tree* is a connected graph with no cycle. A *rooted* tree is a tree in which one vertex r is designated as the *root*. For each vertex v in a tree, let $P(v)$ be the unique path from v to the root r. The *parent* of $v \neq r$ is the unique vertex in $P(v)$ adjacent to v, and the *ancestors* of v are the vertices in $P(v)$. The parent of the root r is not defined. The only ancestor of r is r itself. We say u is a *child* of v if v is the parent of u, and u is a *descendant* of v if v is an ancestor of u. Note that each vertex v is always a descendant and an ancestor of v. The *height* of a vertex v is the number of edges on the longest path from v to a descendant of v. A *leaf* is a vertex having no child. The height of a leaf is always 0, and the height of a vertex is always larger than the height of its child. Precisely, the height of a vertex is the maximum height of its children plus one. A *reduced tree* is a rooted tree in which no vertex has exactly one child. In [2] a reduced rooted tree is called a homeomorphically-irreducible rooted tree. Note that the root of a reduced tree may have degree two, although no vertex has degree two in a similar graph called a *reduced non-rooted tree*, which is a non-rooted tree with no vertex of degree two. A rooted tree is an *ordered tree* if the children of each vertex are linearly ordered, and an *unordered tree* otherwise.

A *full-binary tree* is an ordered (rooted) tree in which each vertex has zero or two children. If vertex v has the ordered children (v_L, v_R), v_L is called the *left child* and v_R is called the *right child* of v. A path $(v_1, v_2, \cdots, v_j)$ in a full-binary tree, satisfying that each v_i is the left child of v_{i+1} for $i = 1, 2, \cdots, j-1$, is called a *left-down path*. A left-down path is *maximal* if it is not a proper subpath of any

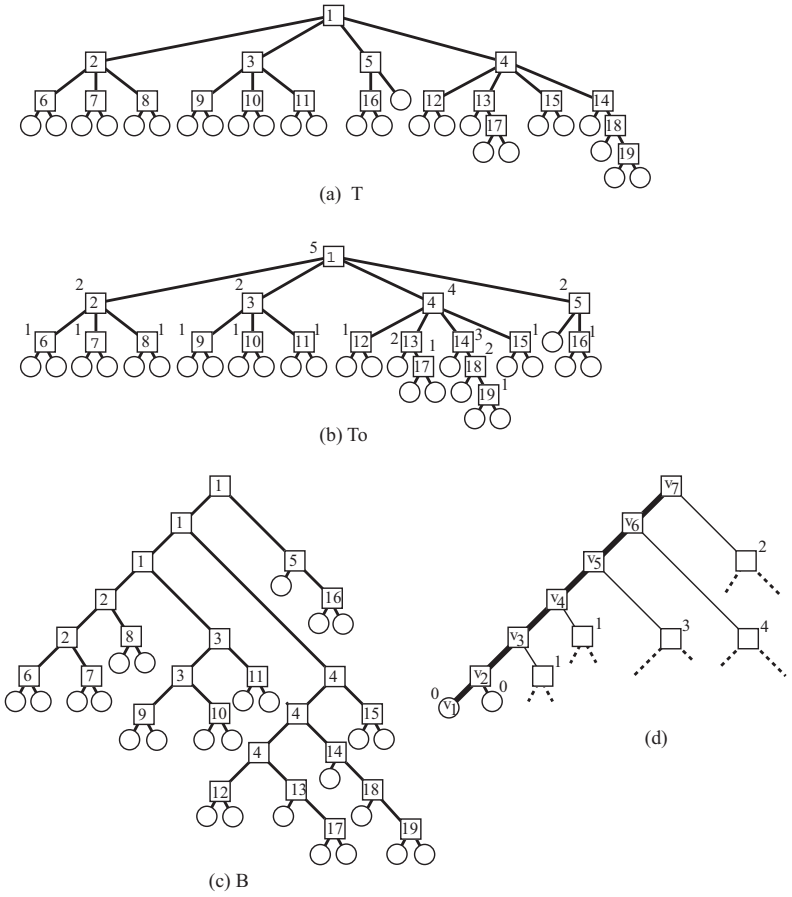

Fig. 1. Outline of the algorithm

other left-down path. A left-down path from a vertex v to its ancestor u is denoted by the sequence of its vertices starting from v and ending with u.

3. Compact Representation of Unordered Reduced Trees

In this section, we assume that the input is an unordered reduced (rooted) tree T with ℓ leaves. See an example in Fig. 1(a).

The outline of our algorithm is as follows. We first give a linear ordering of children of each vertex of T by a simple rule. Now the unordered reduced tree T is transformed into an "ordered" reduced tree T_O (Fig. 1(b)). Then, by replacing each non-leaf vertex of T_O having $k > 0$ children by a left-down path of length $k-1$ (Fig. 2), the ordered reduced tree T_O is transformed into a full-binary tree B. See Fig. 1(c). Note that both T_O and B still have ℓ leaves. Finally, B is encoded into a bit string S of length $2\ell - 1$. The encoding of B is done by a depth first tree

traversal, in which each edge is traversed exactly twice in opposite direction. When an edge is traversed downward, a "0" is appended to bitstring S, and when an edge is traversed upward, a "1" is appended to S. From the bitstring S one can easily reconstruct B.

However, to reconstruct T_O from B, we need to divide each maximal left-down path of B into left-down subpaths so that each subpath corresponds to a vertex in T. Thanks to the linear ordering of children, we can always uniquely divide each left-down path of B into suitable left-down subpaths. By ignoring the linear ordering of children, T is derived from T_O.

3.1. *Encoding an unordered reduced tree*

Now we give an encoding algorithm for unordered reduced trees. We begin with a linear ordering of children, which is the key to our compact representation.

Let T be an unordered reduced tree with ℓ leaves. Let $(v_1, v_2, \cdots, v_k)$ be the sequence of children of a vertex u. We assume the heights of the vertices are increasing order in the sequence, i.e., v_1 has the smallest height and v_k has the largest height (ties are broken in arbitrary way). The linear ordering of children is the sequence obtained by removing v_2 from the sequence and appending v_2 to the last, that is, $(v_1, v_3, v_4, \cdots, v_k, v_2)$. Note that each non-leaf vertex always has two or more children, thereby the sequence above is always defined. The linear orderings define an ordered reduced tree T_O obtained from T. See Fig. 1(b). Intuitively, among the children of each vertex, the leftmost child has the minimum height, the rightmost child has the second minimum height, and the rest of children appear between them with increasing order of heights.

Our idea of a compact representation is to represent T_O by a full-binary tree with the same number of leaves. We replace each non-leaf vertex u of T having $k > 0$ children by a left-down path of length $k - 1$ as shown in Fig. 2. We call the left-down path the *expand path* of u, and denote it by $L(u)$. In the replacement, we keep the linear ordering of the children. When the linear ordering of children of u is $(v_1, \ldots, v_k)$ and $L(u) = (u_1, \ldots, u_{k-1})$, we set v_1 to the left child of u_1, v_2 to the right child of u_1, v_3 to the right child of u_2, ..., v_i to the right child of u_{i-1}, ..., and v_k to the right child of u_{k-1}. For a rooted ordered tree T in Fig. 1(a), the resulting full-binary tree B is drawn in Fig. 1(c). Essentially, our compact representation is a bit string representing B. The algorithm to compute the representation is as follows.

Algorithm (T: an unordered reduced tree)
1. compute the height of each vertex in T;
2. sort the children of each vertex by their heights;
3. compute the linear ordering of the children of each vertex;
4. replace each node having more than two children by the left-down path with keeping the linear ordering of children;
5. remove all leaves from the resulting tree B;

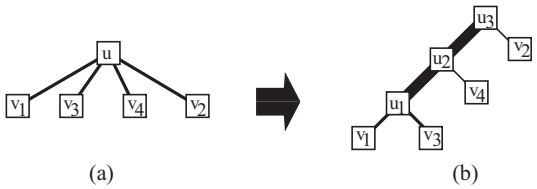

Fig. 2. Replace a vertex v by a left-down path

6. output the bit string representing B (by the depth first tree traversal).

The height of each vertex of T can be computed in $O(\ell)$ time in a bottom up way. The sorting of the children can be done in $O(\ell)$ time, by processing all sortings at once by bucket sort.

We here describe the details of step 6 that encodes the resulting tree B. We put a label 0 to each leaf and 1 to inner nodes. We then compute the pre-order of vertices by a depth-first search going left child first, and right child next. The pre-order is the visiting order by the depth-first search; at the beginning of the search, the sequence is empty, and when a new vertex is visited by the search, the vertex is added to the end of the sequence. The bit string is the sequence of vertex labels ordered by the pre-order. For example, the tree T in Fig. 1(a) is encoded to

$$S = 11111100100100111001001001111001010010101001001010 0.$$

We can see that one can reconstruct B from the bit string S by simulating the traverse. In the next section, we will show that the expand path of each vertex can be extracted from the maximal left-down paths thanks to the linear ordering, therefore the reconstruction of the original unordered reduced tree T from the bit string S can be done uniquely and efficiently. We have the following theorem.

Theorem 3.1. *One can compute in $O(\ell)$ time a bit string of length $2\ell - 1$ representing an unordered reduced tree with ℓ leaves.*

3.2. *Decoding an unordered reduced tree*

The full-binary tree B can be obtained from the bit string S. Since T is an unordered tree, it is enough to show that T_O can be reconstructed from B. The purpose of this section is to establish a way to extract expand paths of all vertices in T_O.

We choose a maximal left-down path and extract all the expand paths included in the path. We start from the rightmost maximal left-down path, and iteratively process maximal left-down paths from right to left. Therefore, when we process a left-down path L, the extraction has been done in descendants of the right child u of any vertex in L. See an example in Fig. 1(d). This implies that we have constructed all the subtrees of T_O rooted at the descendants of u, and hence we can compute

the height of any child x of a vertex v_i in the left-down path L. Set $h(v_i)$ be the height of the right child of v_i.

The extraction of expand paths is done in a bottom up way. Suppose that $L = (v_1, \ldots, v_k)$. We first observe that v_1 itself corresponds to a leaf of T_O, thus the lowest expand path $L(x)$ starts from v_2. We find the other end v_j of $L(x)$ by looking at the heights of right children of $v_2, \ldots, v_k$ one by one, so that cutting L at the vertex will not result the violation of the linear ordering. In precise, v_j is the vertex in the ancestor of v_2 whose right child has the smallest height among the right children of all ancestors of v_2. Then the next lowest expand path starts from v_{j+1}, and in the same way we can find the other end. In this way, we iteratively extract the expand paths until we reach to the top of L. The upside end vertex of an expand path is characterized by the following lemma. This is the key to the extraction of expand paths.

Lemma 3.2. *Suppose that an expand path $L(u)$ in T is $(v_i, \ldots, v_j)$ for some v_j. Then, v_j is the uppermost vertex in the maximal left-down path L including $L(u)$ such that $h(v_j) \leq h(v_i)$.*

Proof. Let v_l is the uppermost vertex in L such that $h(v_l) \leq h(v_i)$. We will prove that $v_l = v_j$. Note that v_l can be v_i, and v_j can be v_i.

Suppose that v_l is a proper descendant of v_j, i.e., $v_l \in L(u)$ and $v_l \neq v_j$. Then, u has at least three children, and the heights of children are $h_0, h(v_j), \ldots, h(v_l)$, where h_0 is the height of the first child of u. From the definition of v_l, $h(v_l) < h(v_j)$, however this contradicts to the linear ordering. Therefore, v_l is not a proper descendant of v_j.

We next suppose that v_l is a proper ancestor of v_j, i.e., $v_l \notin L(u)$. Let $L(u') = (v_{i'}, ..., v_{j'})$ be the expand path of u' which includes v_l. Observe that the height of the left child w of u' is no less than the height of u since w is an ancestor of u. This implies that $h(v_l)$ is smaller than the height of w, since the height of u is strictly larger than $h(v_i)$. This contradicts to the linear ordering. From these, we conclude that $v_l = v_j$. □

Lemma 3.2 claims that the highest vertex v_l in L such that $h(v_l) \leq h(v_i)$ is the other end vertex of $L(u)$. Such a vertex v_l can be found by looking the heights of all vertices in L, thus we obtain the following algorithm to reconstruct T from S.

Algorithm Decode (S: bit string)
1. $B :=$ the full-binary tree obtained from S
2. $V := \emptyset$
3. **for each** maximal left-down path L from right to left
4. remove the leaf v from L, and $V := V \cup \{v\}$
5. compute the height of each vertex in L
6. **while** L is not empty
7. $v_i :=$ the lowest vertex in L

8. v_l := the highest vertex such that $h(v_l) \leq h(v_i)$
9. remove the subpath from v_i to v_l, and insert it to V
10. **end while**
11. **end for**
12. E := edge set induced by the parent-child relation of expand paths and leaves in B

A naive implementation makes this algorithm run in $O(|S|^2)$ time, where $|S|$ is the length of the bit string S. We explain how to compute it in $O(|S|)$ time.

Let $L = (v_1, \ldots, v_k)$ be a left-down path and $L(u) = (v_i, \ldots, v_l)$ be an expand path included in L. We suppose that $v_i \neq v_k$, otherwise $v_l = v_i$. Let $v_j, j > i$ be the lowest vertex satisfying $h(v_{j-1}) > h(v_j)$. If there is no such vertex, v_j is not defined. $v_{j'}$ is the highest vertex satisfying $h(v_{j'}) = h(v_i)$. Note that v_i can be $v_{j'}$, and $v_{j'}$ is always defined. The key observation is that in the linear ordering, children on the middle are sorted in the increasing order of their heights. From the observation, we have the following lemma.

Lemma 3.3. *(1) $v_l = v_j$ holds if v_j is defined and $h(v_j) < h(v_i)$, and (2) $v_l = v_{j'}$ holds if $h(v_j) \geq h(v_i)$ or v_j is not defined.*

Proof. We first observe that v_l never be a proper ancestor of v_j from the linear ordering. Similarly, if v_j is defined, no ancestor of v_j has a height smaller than or equal to v_i; otherwise the linear ordering contains a decreasing point at the middle. This together with the definition of v_l implies that when v_j is defined and $h(v_j) < h(v_i)$, we have $v_l = v_j$. If v_j is defined but $h(v_j) \geq h(v_i)$, we have $v_l = v_{j'}$. In the case that v_j is not defined, heights of the vertices in $(v_i, v_{i+1}, \cdots, v_k)$ are monotonically non-decreasing, thus the highest vertex having the height no more than v_i is $v_{j'}$. This implies $v_l = v_{j'}$, and we conclude the lemma. □

From Lemma 3.3, the task to find an expand path is to find both v_j and $v_{j'}$. This can be done in $O(j - i)$ time as follows: If $v_l = v_j$, then we never access to the vertices $\{v_i, \ldots, v_j\}$. If $v_l = v_{j'}$, we set v_i to $v_{j'+1}$, and find v_j and $v_{j'}$ again to find the next expand path. However, in this case, we know that even for new v_i, the previous v_j is still the lowest vertex satisfying $h(v_{j-1}) > h(v_j)$. Thus, we do not need to compute v_j again. In summary, each vertex in a maximal left-down path is accessed twice; once for finding v_j and once for finding $v_{j'}$. The algorithm maintains two lists. The first list consists of the vertices v_j with $h(v_{j-1}) > h(v_j)$ from descendants, and the second list consists of the vertices of the same height. Both lists can be maintained in linear time, and they admit us to find v_j and $v_{j'}$ in $O(1)$ time. Thus, the extraction of expand paths from a maximal left-down path L can be done in $O(|L|)$ time.

Therefore, any bit string that represents an unordered reduced tree can be decoded uniquely. Thus no two different unordered reduced trees are encoded into the same bit string. Hence we have the following theorems.

Theorem 3.4. *For any $\ell \geq 1$, there is a 1-to-1 mapping from the set of unordered reduced trees of ℓ leaves to the set of bit strings of length at most $2\ell - 1$.*

Theorem 3.5. *From a bit string of length $2\ell-1$ representing an unordered reduced tree with $\ell \geq 1$ leaves, the unordered reduced tree can be reconstructed in $O(\ell)$ time.*

Since the first bit of the bit string is always 1 when the original unordered tree has at least two leaves, we obtain the following theorem.

Theorem 3.6. *The number of unordered reduced trees with $\ell \geq 1$ leaves is at most $2^{2\ell-2}$.*

4. Compact Representation of Cographs

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two arbitrary disjoint graphs. A graph $G = (V, E)$ is the *parallel composition* of G_1 and G_2 if $V = V_1 \cup V_2$ and $E = E_1 \cup E_2$. A graph $G = (V, E)$ is the *series composition* of G_1 and G_2 if $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup \{(x_1, x_2) | x_1 \in V_1, x_2 \in V_2\}$. A *cograph* is a graph composed of one vertex, or a graph obtained from two cographs by one of these two compositions. It is well known that a cograph has a canonical tree representation [6,8], in which (1) each leaf of the tree corresponds to a vertex of the graph, (2) each internal vertex of the tree has a label corresponding to either a series or parallel composition, and (3) on every path in the tree, the labels appear alternatively. We note that (1) implies that the number of leaves of the tree is n for any cograph of n vertices.

Each such canonical tree corresponds to a unique cograph up to isomorphism, thus no two non-isomorphic cographs are made from the same canonical tree and vice versa. Fig. 3 shows an example of a cograph and its canonical tree representation.

In a canonical tree, each non-leaf vertex has at least two children and the ordering of them does not matter. That is, the tree structure is an unordered reduced tree. By (3), an unordered reduced tree corresponds to two cographs; one with the root label of the series composition, and the other with the root label of the parallel composition. Hence the number of cographs of n vertices is twice of the number of unordered reduced trees of n leaves which is equal to 2^{2n-2}. Therefore, the following theorem holds:

Theorem 4.1. *The number of cographs with n vertices is at most 2^{2n-1}.*

It is known that the canonical tree representation of a cograph can be obtained in $O(n+m)$ time, where n is the number of vertices and m is the number of edges in the cograph. From the canonical tree representation one can reconstruct the original cograph in $O(n + m)$ time. We thus have the following theorem.

Theorem 4.2. *A canonical bit string of length $2n - 1$ for a cograph with n vertices and m edges can be computed in $O(n + m)$ time. From the bit string, the corresponding cograph can be retrieved in $O(n + m)$ time.*

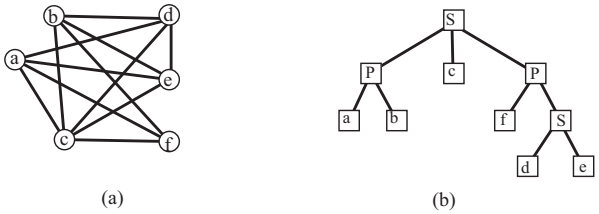

Fig. 3. A cograph and its canonical tree representation

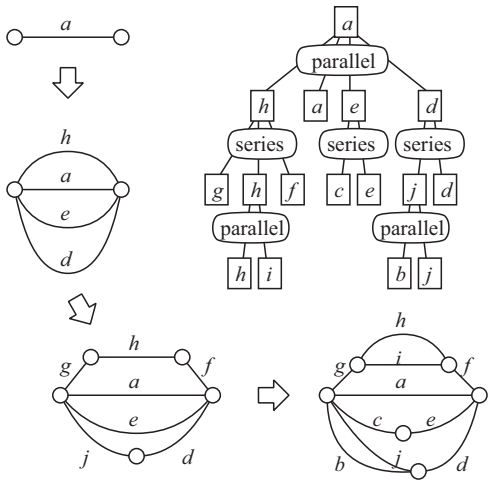

Fig. 4. A connected series-parallel graph and a way to construct it

5. Compact Representation of Series-Parallel Graphs

Let $G = (V, E, s, t)$ be a multi graph with two designated vertices s and t, called the *terminals*. A multi graph is a graph that can contain two or more identical edges having the same endpoints. A graph $G = (V, E, s, t)$ is the *parallel composition* of $G_1 = (V_1, E_1, s_1, t_1)$ and $G_2 = (V_2, E_2, s_2, t_2)$ if $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, $s = s_1 = s_2$, and $t = t_1 = t_2$. A graph $G = (V, E, s, t)$ is the *series composition* of $G_1 = (V_1, E_1, s_1, t_1)$ and $G_2 = (V_2, E_2, s_2, t_2)$ if $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, $s = s_1$, $t_1 = s_2$, and $t = t_2$. The series and the parallel composition can easily be generalized to more than two graphs.

A graph $G = (V, E, s, t)$ is a *series-parallel graph* with terminals s and t if it consists of only one edge $\{s, t\}$ or it can be obtained by a series composition or a parallel composition of two or more series-parallel graphs.

It is well known that any series-parallel graph has a tree structure which describes how the graph is composed [3], and no two non-isomorphic series-parallel

graphs with two terminals are generated from the same tree structure. An example of a series-parallel graph and its construction tree are shown in Fig. 4. Similar to cographs, the structure is a reduced tree with alternative labels. The difference is that the order of children matters for only series composition. Therefore, we have to give care the ordering of children only for series composition, and we cannot use the unordered reduced tree directly. We estimate the extra bits required to remember the ordering of children at each series composition.

We give our linear ordering to parallel compositions in the same way and encode the obtained full-binary tree B to a bit string. To enable us to extract the expand paths, we put a length of the lowest expand path to each maximal left-down path, and the label of the corresponding vertex if necessary. We will show that it is sufficient to reconstruct the original construction tree T.

Let $L = (v_1, \ldots, v_k)$ be a maximal left-down path, and $L(z) = (v_2, \ldots, v_j)$ be the lowest expand path in L. Like for reduced trees, we assume that the extraction has been done in descendants of the right child and the vertices in L. In the following, we observe that several labels are automatically determined by the structure of B. Let v_i be a non-leaf vertex in L, and x be the vertex such that $v_i \in L(x)$. Let u be the right child of v_i, and y be the vertex such that $u \in L(y)$. We first consider the case that u is a non-leaf vertex. In this case, the label of x is automatically determined by the label of y, since they are always different. We next suppose that u is a leaf and v is not in $L(z)$. From the linear ordering, any child of x cannot be a leaf if x is a parallel composition. This together with that u is a leaf implies that x is a series composition.

From the above observation, we can determine the label of $x \neq z$ whose expand path is included in L, by looking at the right children of the vertices in the path. Thus, we cannot get the label of $L(z)$ only when $L(z)$ is the unique expand path in L. In this case, we have to memory the label of z by using additional bits. Let $U(L) = (u_1, \ldots, u_{k'})$ be the vertices in L such that their right children are leaves. We have to distinguish the following $(k' + 1)$ cases.

$$
\begin{aligned}
1 &: u_1, u_2, \ldots, u_{k'} \in L(z) \text{ and their labels are “Series”} \\
2 &: u_1 \in L(z) \text{ and its label is “Parallel”} \\
3 &: u_1, u_2 \in L(z) \text{ and their labels are “Parallel”} \\
&\cdots \\
k' &: u_1, u_2, \ldots, u_{k'-1} \in L(z) \text{ and their labels are “Parallel”} \\
k'+1 &: u_1, u_2, \ldots, u_{k'} \in L(z) \text{ and their labels are “Parallel”}
\end{aligned}
$$

Note that, in the case 2 to $k' + 1$, we can determine that all labels of the other vertices in $U(L)$ are “Series.”

We put the index of the corresponding case to each left-down maximal path, and this information is sufficient to reconstruct T and know the labels of the vertices.

The maximum bits to store the information is equal to

$$\log_2 \max\{\prod_{S\in P}(|S|+1) \mid \ P \text{ is a partition of } \{1,\ldots,n\}\}.$$

By a simple calculation, we can observe that the maximum is attained by the partition $P = \{S_1, S_2, \ldots, S_h\}$ such that $|S_1| = |S_2| = \cdots = |S_{h-1}| = 3$ and $0 < |S_h| \leq 3$. In the case, the maximum length of the bit string is $\lceil (\log_2 3)n/3 \rceil < 0.5284n$. We also need $2n-2$ bits to store the full binary tree of size n. Therefore, we obtain the following theorem:

Theorem 5.1. *The number of series-parallel graphs with two terminals and m edges is at most $2^{\lceil 2.5285m-2 \rceil}$.*

It is known that the construction tree of a series-parallel graph can be obtained in $O(m)$ time, thus encoding can be done in $O(m)$ time. Decoding is also done in $O(m)$ time. Thus, we have the following theorem.

Theorem 5.2. *There is a coding for the class of series-parallel graphs with two terminals and m edges in at most $\lceil 2.5285m - 2 \rceil$ bits with encoding and decoding algorithms running in $O(m)$ time.*

6. Conclusion

In this paper, we designed an algorithm to compute a compact representation of an unordered reduced tree. Our algorithm computes in $O(\ell)$ time a bit string of length $2\ell - 1$ for an unordered reduced tree with ℓ leaves, and also reconstructs in $O(\ell)$ time the original tree from the bit string. We also showed the number of cographs of n vertices is at most 2^{2n-1}, and the number of series-parallel graphs with two terminals of m edges is at most $2^{\lceil 2.5285m-2 \rceil}$. According to The On-Line Encyclopedia of Integer Sequences (`http://oeis.org/A000084`), the numbers of cographs of small $n = 1, 2, \cdots$ vertices are $1, 2, 4, 10, 24, 66, 180, 522, 1532, \cdots$, and it is estimated as $Cd^n/n^{3/2}$, where $C = 0.4126\cdots$ and $d = 3.5608\cdots$. Hence there may still exist a chance to improve the representation.

Acknowledgement

Part of this research is supported by the Funding Program for World-Leading Innovative R&D on Science and Technology, Japan.

References

[1] Aleardi, L.C., Devillers, O., Schaeffer, G.: Succinct Representation of Triangulations with a Boundary. In: WADS 2005. pp. 134–145. Lecture Notes in Computer Science Vol. 3608, Springer-Verlag (2005)

[2] Bergeron, F., Labelle, G., Leroux, P.: Combinatorial Species and Tree-Like Structures. Cambridge University Press (1998)

[3] Brandstädt, A., Le, V., Spinrad, J.: Graph Classes: A Survey. SIAM (1999)
[4] Cheng, Q., Berman, P., Harrison, R., Zelikovsky, A.: Efficient Algorithms of Metabolic Networks with Bounded Treewidth. In: IEEE International Conference on Data Mining Workshops. pp. 687–694. IEEE (2010), `http://www.computer.org/portal/web/csdl/doi/10.1109/ICDMW.2010.150`
[5] Chiang, Y.T., Lin, C.C., Lu, H.I.: Orderly Spanning Trees with Applications. SIAM J. Comput. 34(4), 924–945 (2005)
[6] Corneil, D.G., Perl, Y., Stewart, L.K.: A Linear Recognition Algorithm for Cographs. SIAM Journal on Computing 14(4), 926–934 (1985)
[7] Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. ACM Transactions on Algorithms 2, 510–534 (2006)
[8] Habib, M., Paul, C.: A simple linear time algorithm for cograph recognition. Discrete Applied Mathematics 145(2), 183–197 (2005)
[9] Jacobson, G.: Space-efficient Static Trees and Graphs. In: Proc. 30th Symp. on Foundations of Computer Science. pp. 549–554. IEEE (1989)
[10] Keeler, K., Westbrook, J.: Short Encodings of Planar Graphs and Maps. Discrete Applied Mathematics 58(3), 239–252 (1995)
[11] Knuth, D.: Sorting and Searching, vol. 3 of *The Art of Computer Programming*. Addison-Wesley Publishing Company, 2nd edn. (1998)
[12] Munro, J.I., Raman, V.: Succinct Representation of Balanced Parentheses and Static Trees. SIAM Journal on Computing 31, 762–776 (2001)
[13] Spinrad, J.: Efficient Graph Representations. American Mathematical Society (2003)
[14] Yamanaka, K., Nakano, S.I.: A compact encoding of plane triangulations with efficient query supports. Inf. Process. Lett. 18-19, 803–809 (2010)