Theoretical Computer Science 411 (2010) 3293–3300

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Note

On compact representations of All-Pairs-Shortest-Path-Distance matrices☆

Paolo Ferragina*, Igor Nitto, Rossano Venturini[1]

Dipartimento di Informatica, University of Pisa, Italy

ARTICLE INFO

Article history:
Received 31 March 2008
Received in revised form 1 December 2009
Accepted 19 May 2010
Communicated by G. Italiano

Keywords:
Succinct data structures
Compressed indexes for strings and trees
All-Pairs-Shortest-Path Distances

ABSTRACT

Let G be an unweighted and undirected graph of n nodes, and let $\mathbf{D}$ be the $n \times n$ matrix storing the All-Pairs-Shortest-Path Distances in G. Since $\mathbf{D}$ contains integers in $[n] \cup +\infty$, its plain storage takes $n^2 \log(n+1)$ bits. However, a simple counting argument shows that $n^2/2$ bits are necessary to store $\mathbf{D}$. In this paper we investigate the question of finding a succinct representation of $\mathbf{D}$ that requires $O(n^2)$ bits of storage and still supports constant-time access to each of its entries. This is asymptotically optimal in the worst case, and far from the information-theoretic lower bound by a multiplicative factor $\log_2 3 \simeq 1.585$. As a result $O(1)$ bits per pairs of nodes in G are enough to retain constant-time access to their shortest-path distance. We achieve this result by reducing the storage of $\mathbf{D}$ to the succinct storage of labeled trees and ternary sequences, for which we properly adapt and orchestrate the use of known compressed data structures. This approach can be easily and optimally extended to graphs whose edge weights are positive integers bounded by a constant value.

1. Introduction

The study of succinct data structures has recently attracted a lot of interest in the research arena. A data structure is called *succinct* [12] when its space is *up to* a constant factor from the information-theoretic lower bound, and all of its operations can be supported without any slowdown with respect to the corresponding *plain* (un-succinct) data structure. Nowadays there exist succinct versions of various data structures and data types: binary vectors [16,20,21], dictionaries [11,7], strings [19], (un)labeled trees [3,4,13], binary relations and graphs [17,1], etc.

In this paper we contribute to the design of new succinct data structures by investigating the compact representation of All-Pairs-Shortest-Path-Distance matrices of unweighted and undirected graphs. Formally, let G be an unweighted and undirected graph of n nodes, and let $\mathbf{D}$ be the $n \times n$ matrix that stores in its entry $\mathbf{D}[u, v]$ the length of the shortest path connecting node u to node v in G (or $+\infty$ when u and v are not connected). $\mathbf{D}$ is called the All-Pairs-Shortest-Path-Distance matrix of G, and it is typically stored in $O(n^2)$ memory words, thus taking $n^2 \log(n+1)$ bits in total.[2]

Various authors have investigated the problem of designing succinct graph encodings for supporting the retrieval of either the *adjacency list* of a node (see [17,18] and the references therein), or the *approximate distance* between node pairs in various types of graphs (see [23,22] and the references therein). Specifically, [23] proposed a data structure (distance oracle) for *approximating* shortest-path distances in general weighted graphs up to a multiplicative *stretch factor*. For any fixed k,

☆ A preliminary version of this paper appeared in the Proceedings of the 19th Symposium on Combinatorial Pattern Matching, Springer LNCS 5029, 2008. This work has been partially supported by a Yahoo! Research grant.

* Corresponding address: Dipartimento di Informatica, Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel.: +39 0502212664.
E-mail addresses: ferragina@di.unipi.it (P. Ferragina), nitto@di.unipi.it (I. Nitto), rossano.venturini@isti.cnr.it (R. Venturini).
1 Current address: ISTI-CNR, Pisa, Italy.

2 Throughout this paper we assume that all logarithms are taken to the base 2, whenever not explicitly indicated, and we assume that $0 \log 0 = 0$.

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.05.021

their approximate distance oracle achieved a stretch factor $2k-1$, using $O(kn^{1+1/k}\log n)$ bits of storage and taking $O(k)$ time per distance query. A recent result by Naor and Mendel [15] achieves constant query time, independent of k. Distance oracles with near-linear space complexity exist for planar digraphs, as shown in [22]. For *exact* distances, and thus, for the compact storage of **D**, it is still open whether it is possible to deploy the intrinsic structure of this matrix to devise a representation which is as much close as possible to the information-theoretic lower bound of $n^2/2$ bits, and still takes constant time in the worst case to access any one of its entries.[3]

In our paper we show how to match *asymptotically* the above lower bound by providing a succinct storage scheme for **D** which achieves a bit-space complexity within a factor of $\log_2 3 \simeq 1.585$ from the information-theoretic minimum, and is still able to retrieve in constant time any node-pair distance in **D**. This approach can be easily and optimally extended to graphs whose edge weights are positive integers bounded by a constant value (see Section 7).

Technically, our paper proposes a novel algorithmic reduction (detailed in Theorem 2) which turns the storage of **D** into the succinct storage of (ternary) labeled trees and (ternary) sequences, for which we properly adapt and orchestrate known compressed data structures. In detail, our approach will consist of the following three main steps: (1) We first reduce the storage of **D** into the storage of properly labeled (sub)trees (Theorem 2); then (2) we reduce this latter problem to the succinct storage of ternary sequences and turn any distance query into a constant number of prefix-sum queries over these sequences (Lemma 3); finally (3) we adapt known compressed data structures to execute constant-time prefix sums over those sequences (see Sections 5 and 6).

Using this algorithmic scheme we obtain two main results: a simple compact representation of **D** that requires $(\log_2 3)\ n^2 + o(n^2)$ bits of storage and $O(1)$ access time to anyone of its entries (Corollary 2); and, a more sophisticated solution which reduces the space complexity to $(\frac{1}{2}\log_2 3)\ n^2 + o(n^2)$ bits (Corollary 3) still taking constant time per distance query.

2. Some basic facts

We assume the standard RAM model with memory words of $\Theta(\log n)$ bits, where n is the number of nodes in G.

Let $S[1, n]$ be a sequence drawn from the alphabet $\Sigma = \{a_1, \ldots, a_\sigma\}$. For each symbol $a_i \in \Sigma$, we let n_i be the number of occurrences of a_i in S. Let $\{P_i = n_i/n\}_{i=1}^{\sigma}$ be the empirical probability distribution for the sequence S. The zeroth order *empirical* entropy of S is defined as: $H_0(S) = -\sum_{i=1}^{\sigma} P_i \log P_i$. Recall that $|S|H_0(S)$ provides an information-theoretic lower bound to the output size of any compressor that encodes each symbol of S with a fixed codeword.

The Wavelet Tree [9] is an elegant and powerful data structure that supports efficient rank/select primitives over sequences drawn from arbitrarily large alphabets, and achieves entropy-bounded space occupancy. More precisely, it has been shown in [9] that:

Theorem 1. *Given a sequence $S[1, n]$ drawn from an arbitrary alphabet Σ, the Wavelet Tree built on S takes $nH_0(S) + o(n)$ bits to support the following queries in $O(\log|\Sigma|)$ time:*

- *Retrieve character $S[i]$;*
- *$Rank_c(S, i)$: compute the number of times character $c \in \Sigma$ occurs in $S[1, i]$;*
- *$Select_c(S, i)$: compute the position of the ith occurrence of character $c \in \Sigma$ in S.*

In addition to rank/select primitives, the design of our compact representations for **D** will need to support fast *prefix sums* over integer sequences drawn from potentially large (integer) alphabets. We therefore state the following result which is an easy consequence of [14]:

Lemma 1. *Let $S[1, n]$ be a sequence drawn from the integer alphabet $\Sigma = \{-l, \ldots, 0, \ldots, l\}$. There exists an encoding of S that takes $n\lceil\log(2l+1)\rceil + o(n\log l)$ bits and supports prefix-sum queries in $O(1)$ time.*

A key fact in our techniques will be the availability of a storage scheme for a string S which is space succinct and is able to decode in $O(1)$ time any *short* substring of S having length logarithmic in n. To this aim, we use the following result which is an easy corollary of [6].

Corollary 1. *Given a sequence $S[1, n]$ drawn from a constant-size alphabet Σ, there is a succinct data structure that stores S in $n\log|\Sigma| + o(n)$ bits and supports the retrieval in constant time of any substring of S of length $O(\log n)$ bits.*

Notice that the plain storage of S would have taken $n\lceil\log|\Sigma|\rceil$ bits, which would give a $n\lceil\log_2 3\rceil = 2n$ space bound in our solution of Section 6. By applying the succinct data structure of the above corollary we can store S in at most $n\log_2 3 + o(n)$ bits of space and still guarantee constant time to access any short substring of S (namely, one of length $O(\log n)$).

In the rest of this paper, we will also make use of the following two structural properties of the distance matrix **D**:

Symmetry: $\mathbf{D}[u, v] = \mathbf{D}[v, u]$
Triangle inequality: $|\mathbf{D}[u, v] - \mathbf{D}[w, v]| \leq \mathbf{D}[u, w]$

[3] This lower bound comes from the observation that there is a one-to-one correspondence between unweighted undirected graphs and their distance matrices. Thus the number of $n \times n$ distance matrices is $2^{n(n-1)/2}$.

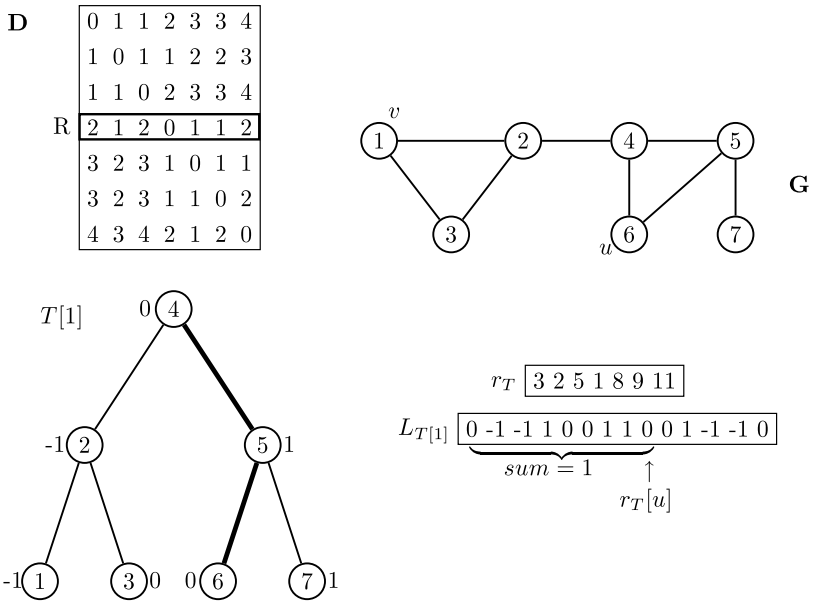

Fig. 1. (Top) A graph G and its distance matrix $\mathbf{D}$. (Bottom) An example of labeled tree $T[1]$, relative to node $1 \in G$, and the associated arrays $L_{T[1]}$ and $r_{T[1]}$. According to Lemma 3 the sum of the labels on $\pi(6)$ is equal to the prefix-sum in $L_{T[1]}[1, r_{T[1]}[6]] = L_{T[1]}[1, 9]$ which correctly returns the value 1.

where u, v, w are any triplet of nodes in the graph G. Note that the triangle inequality has been rewritten in a form that will help future references and intuitions. We finally notice that we can safely assume the graph G to be *connected*. Otherwise we can associate every connected component of G with its distance matrix and then assign proper node labels in a way that takes constant time to check whether two nodes are in the same connected component. The additional storage required by these labels is negligible, because it consists of $O(n \log n) = o(n^2)$ bits.

3. From matrix D to labeled (spanning) trees of G

In this section we show how to reduce the problem of succinctly representing the distance matrix $\mathbf{D}$ into the problem of finding a succinct data structure that encodes a (ternary) labeled tree and supports in constant time a kind of *path-sum query* over its structure. To explain how this algorithmic reduction works, we introduce a useful notation and terminology.

Let T be a spanning tree of the graph G and root T at anyone of its nodes, say r. Given that G is connected, T spans all n nodes of G. For each node u of T (and thus of G), we denote with:

- $\ell(u)$ an integer *label* in $\{-1, 0, 1\}$, associated to u;
- $\mathtt{pre}(u)$ the rank of u in the preorder visit of T (i.e., integer in $[n]$).
- $\pi(u)$ the downward path in T which connects root r to u.
- $f(u)$ the father of node u in T, and with $f^i(u)$ the ith ancestor of u in T (where $f^0(u) = u$).

Among all the possible ternary labelings ℓ of T, we consider the ones induced by the pairwise distances in G. Specifically, for any node $v \in T$ we define a labeling ℓ_v over all nodes $u \in T$, such that $\ell_v(u) = \mathbf{D}[u, v] - \mathbf{D}[f(u), v]$. This is a ternary labeling because of the triangle inequality and the adjacency of u and $f(u)$ in T (and thus in G). The labeled tree resulting by the ternary labeling ℓ_v applied to T is hereafter denoted by $T[v]$. An illustrative example is given in Fig. 1 which shows a graph G and its distance matrix $\mathbf{D}$. In that example we have chosen the vertex $r = 4$ as the root of the spanning tree T, and we have depicted the labeled tree $T[1]$, namely the tree T with vertices labeled according to ℓ_1 and thus with their distance computed from the vertex $v = 1$.
The labeled tree $T[v]$ possesses an interesting property:

Lemma 2 (*Path-sum Query*)**.** *For any node u, the sum of the labels on the downward path $\pi(u)$ in $T[v]$ is equal to $\mathbf{D}[u, v] - \mathbf{D}[r, v]$.*

Proof. The mentioned sum is actually a telescopic sum:

$$\sum_{w \in \pi(u)} \ell_v(w) = \sum_{i=0,\ldots,|\pi(u)|-1} \mathbf{D}[f^i(u), v] - \mathbf{D}[f^{i+1}(u), v] = \mathbf{D}[u, v] - \mathbf{D}[r, v]. \quad \square$$

As an example, consider again Fig. 1 and sum the (ternary) labels on the downward path $\pi(6)$ in $T[1]$. The result is $0+1+0=1$ which is in fact equal to $\mathbf{D}[6,1]-\mathbf{D}[4,1]=3-2=1$.

Lemma 2 can be actually rephrased by saying that the computation of the distance $\mathbf{D}[u,v]$ between any pair of nodes $u, v \in G$, boils down to compute the sum of the value $\mathbf{D}[r,v]$ with the (integer) labels found over the path $\pi(u)$ in $T[v]$, which indeed provide a *path-sum query*. This is the key idea underlying the theorem below which details our reduction from the succinct storage of matrix $\mathbf{D}$ to the succinct storage of a *set* of *path-sum data structures* built upon the labeled trees $T[v]$, one per node $v \in G$.

Theorem 2. *Let T be a tree of n nodes, $E(T)$ be an encoding of its structure, and ℓ be a labeling of its nodes over the ternary alphabet $\{-1,0,1\}$. Suppose that there exists a data structure $D(E(T),\ell)$ that occupies $S(n)$ bits and answers path-sum queries over the labeled tree $\ell(T)$ in $T(n)$ time.*
Then the distance matrix $\mathbf{D}$ of an unweighted undirected graph G of n nodes can be encoded in at most $nS(n)+|E(T)|+o(n^2)$ bits, and the distance between any pair of nodes in G can be computed in $T(n)+O(1)$ time.

Proof. Let T be the spanning tree of G rooted at node r. For each node $v \in T$, we define the labeling ℓ_v such that, for any node u, we set $\ell_v(u)=\mathbf{D}[u,v]-\mathbf{D}[f(u),v]$. We call $T[v]$ the tree T labeled with ℓ_v. We then represent the distance matrix $\mathbf{D}$ of graph G via the following three data structures:

- The array $R[1,n]$ which stores the shortest-path distance between r and every other node in G. Namely, R is the rth row of matrix $\mathbf{D}$.
- The data structures $D(E(T),\ell_v)$, for any node v.
- The tree encoding $E(T)$ of T which allows the constant-time retrieval of the location of $\ell_v(u)$ inside $D(E(T),\ell_v)$, for any node-pair u, v.

The first and the third data structures occupy $|E(T)|+O(n\log n)=|E(T)|+o(n^2)$ bits. The second data structure requires $nS(n)$ bits, because v ranges over all n nodes in T. The claimed space bounds therefore follows.

Now, in order to compute $\mathbf{D}[u,v]$, we execute a path-sum query on $D(E(T),\ell_v)$ and retrieve the sum of the labels along the path $\pi(u)$ in $T[v]$. From Lemma 2, this sum equals $\mathbf{D}[u,v]-\mathbf{D}[r,v]$, so that it suffices to add the value $R[v]=\mathbf{D}[r,v]$ to get the final result. Therefore, any distance query takes $T(n)$ time to compute the path-sum plus a constant number of arithmetic and table-lookup operations. □

4. Path-sum queries boil down to prefix-sum queries

Theorem 2 allows us to shift our attention to the design of an efficient data structure that supports path-sum queries over (ternary) labeled trees. Here we go one step further and show that finding such a data structure boils down to finding an encoding of a *ternary sequence* that supports fast prefix-sum computations.

Let T be an n-node tree and let ℓ be a ternary labeling of its nodes. We visit T in preorder and build the following two arrays (see Fig. 1):

- $L_T[1,2n]$ is the ternary sequence obtained by appending the integer label $\ell(u)$ when the pre-visit of node u starts, and the integer label $-\ell(u)$ when the subtree of u has been completely (pre-)visited.
- $r_T[1,n]$ is the array that maps T's nodes to their positions in L_T. Hence $r_T[u]$ stores the preorder-time instant of u's visit. This way, $L_T[r_T[u]]=\ell(u)$.

The sequence L_T has the following, easy to prove, property (see Fig. 1):

Lemma 3. *Let T be an n-node tree labeled with (positive and negative) integers. For any node u, the sum of the labels on the path $\pi(u)$ in T can be computed as the prefix-sum of the integers in $L_T[1,r_T[u]]$.*

Theorem 2 and Lemma 3 provide us with all the algorithmic machinery we need to succinctly encode the distance matrix $\mathbf{D}$. What we really need now are succinct data structures to perform constant-time prefix-sum queries over integer sequences (namely $L_{T[v]}$, for all $v \in G$). The following two sections will detail two possible solutions, one very simple and already asymptotically optimal, the other more sophisticated and closer to the information-theoretic lower bound.

5. Our first solution

In this section we present our first (and simpler) solution that, given $T[v]$, succinctly represents its corresponding ternary sequence $L_{T[v]}$ and permits to compute efficiently prefix-sum queries over it. This first solution resorts to wavelet tree data structure [10]. This way, the prefix-sum query over $L_{T[v]}[1,r_T[u]]$ can be computed by counting (i.e., *rank*ing) the number of -1 and 1 in the queried prefix of $L_{T[v]}$. By Theorem 1, this counting takes constant time while the space required to store the wavelet tree is $2(\log 3)n+o(n)$ bits (since $|\Sigma|=3$ and $H_0(S)\le\log|\Sigma|$).

We are therefore ready to detail our first simple solution to the succinct encoding of $\mathbf{D}$. For each node $v \in T$, we consider the labeling ℓ_v, the resulting labeled tree $T[v]$, and the corresponding ternary sequence $L_{T[v]}$. We then set the tree encoding $E(T)=r_T$ and build $D(E(T),\ell_v)$ as the wavelet tree of the ternary sequence $L_{T[v]}$. By plugging these data structures into Theorem 2, and exploiting Lemmas 2 and 3, we obtain:

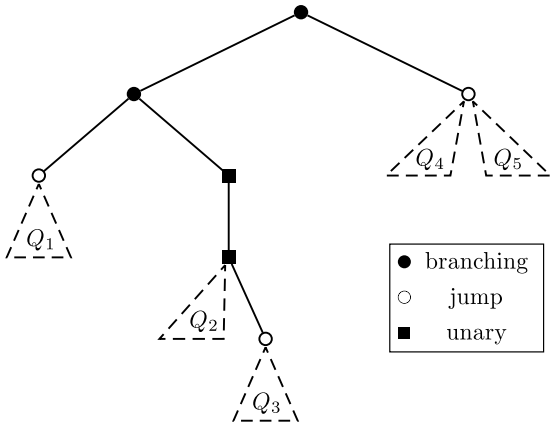

Fig. 2. Macro–micro-tree partition.

Theorem 3. *Let G be an undirected and unweighted graph of n nodes, and let $\mathbf{D}$ be its $n \times n$ matrix storing All-Pairs-Shortest-Path Distances. There exists a succinct representation of $\mathbf{D}$ that uses at most $2n^2(\log 3) + o(n^2)$ bits, and takes constant time to access anyone of its entries.*

For a running example of Theorem 3 we refer the reader to Fig. 1. Assume that we wish to compute $\mathbf{D}[6, 1] = 3$. According to Lemma 2, we need to compute the path-sum over $\pi(6)$ in $T[1]$, which equals to $\mathbf{D}[6, 1] - \mathbf{D}[4, 1] = 1$, and then add to this value $R[1] = \mathbf{D}[4, 1] = 2$ (given that T's root is node 4). By Lemma 3, the path-sum computation boils down to the prefix-sum of $L_{T[1]}[1, r_T[6]] = L_{T[1]}[1, 9]$, which correctly gives the result 1.

In Section 1, we noted that the information-theoretic lower bound for storing the distance matrix $\mathbf{D}$ is $\frac{n^2}{2}$ bits. Therefore the solution proposed in Theorem 3 is asymptotically space and time optimal in the worst case, and away from such lower bound by a multiplicative factor $4 \log_2 3 \simeq 6.34$. This simple approach proves that a succinct encoding taking $O(1)$ bits per pairwise distance of G and $O(1)$ time per distance computation does exist!

A non-trivial issue is now to reduce the amount of bits spent to encode every entry of $\mathbf{D}$, by exploiting some further structural properties of G and T, in order to come as much close as possible to the above lower bound. A first step in this direction is obtained by exploiting the symmetry of matrix $\mathbf{D}$, and thus storing just the prefix $L_{T[v]}[1, r_T[v]]$ for every ternary sequence $L_{T[v]}$. This way, when we query $\mathbf{D}[u, v]$, if $\mathtt{pre}(u) \leq \mathtt{pre}(v)$ we proceed as detailed above (because $r_T[u] \leq r_T[v]$). Otherwise, we swap the role of u and v, and proceed as before. Using this simple trick we halve the space complexity and obtain:

Corollary 2. *There exists a representation for $\mathbf{D}$ that uses at most $n^2(\log 3) + o(n^2)$ bits, and takes constant time to access any one of its entries.*

6. Our second solution

The previous solution is so simple that could be easily implemented. In this section we show how to further halve the space complexity by deploying in a more sophisticated way the structure of T. We proceed in two steps. First, we exhibit a path-sum data structure for an n-node ternary labeled tree that takes $(\log 3)n + o(n)$ bits and supports path-sum queries in $O(1)$ time (Theorem 4). The core of this technique is a well-known approach to the decomposition of arbitrary trees in suitable subtrees, called macro–micro-tree partitioning (see e.g. [2]). Then, we deploy again the "symmetry in $\mathbf{D}$", and get our final result (Corollary 3).

Let T be a tree labeled over $\{-1, 0, 1\}$, and set $\mu = \lceil (\log n)/4 \rceil$. A node $v \in T$ is called a *jump* node, if it has at least μ descendants in T but every child of v has strictly less than μ descendants. A node v is called a *macro* node, if it has at least one jump node among its descendants. The root is assumed to be a macro-node. Any other node of T that is neither jump nor macro is called a *micro*-node. Note that macro and jump nodes can have only macro-nodes as ancestors. Symmetrically, all descendants of micro-nodes are micro-nodes too, so that we define a *micro*-tree as any maximal subtree of micro-nodes in T.

Let $Q_1, \ldots, Q_t$ be the sequence of micro-trees in T ordered by preorder rank of their roots, and let T^* be the subtree of T induced by its macro and jump nodes. Of course, trees $T^*, Q_1, \ldots, Q_t$ form a partition of T (see Fig. 2). Since every micro-node has at most μ descendants, the size of each micro-tree is upper bounded by μ. This decomposition is usually called *macro–micro* partition of T (see e.g. [2]). Below we show how to deploy this decomposition to further reduce the space encoding of $\mathbf{D}$.

Let us concentrate on the subtree T^*, formed by jump and macro-nodes. Note that jump nodes form the leaves of this tree, and are $O(n/\mu) = O(n/\log n)$ in number. The macro-nodes are internal in T^* and can be then divided into *branching* nodes, if they have at least two children in T^*, or *unary* nodes. The number of branching nodes is upper bounded by the

number of leaves in T^*, and thus it is $O(n/\log n)$. To deal with long chains of unary nodes in T^*, we sample them by taking one out of $\lceil \log n \rceil$ consecutive nodes in any maximal unary path of T^*. This way we sample no more than $O(n/\log n)$ unary nodes. The set of nodes formed by jump nodes, branching nodes, and sampled unary nodes is called *breaking nodes*, and has size $O(n/\log n)$. By definition, the distance between any non-breaking node and its closest breaking ancestor in T^* is at most $\lceil \log n \rceil$.

Given the notion of breaking nodes, we define T_F as the tree T^* *contracted* to include only the breaking nodes: i.e., u has parent u' in T_F iff u, u' are breaking nodes and u' is the lowest breaking ancestor of u in T^*. Since we wish to execute path-sum queries over T^* by deploying T_F, we need to reflect the contraction process onto the tree labeling too. This is done as follows. We label every node $u \in T_F$ with the integer $\ell_F(u) = \sum_{w \in \pi(u',u)} \ell(w)$, where u' is the father of u in T_F, $\pi(u', u)$ is the path in T^* connecting u to u', and ℓ is the labeling of T (and thus of T^*). Given the sampling over the unary macro-nodes, and since ℓ is assumed to be a ternary labeling, the label $\ell_F(u)$ is an integer less than $\lceil \log n \rceil$ (in absolute value). At this point, we note that the path-sum leading to any breaking node u can be equally computed either in T or in T_F.

To apply Theorem 2, we need a succinct path-sum data structure that we design here based on the macro–micro decomposition of the ternary labeled tree T. Specifically, let us assume that we wish to answer a path-sum query on a node $u \in T$, we distinguish three cases depending on whether u is micro or not.

(1) Node u is non-micro and breaking. As observed above, we can compute the path-sum over $\pi(u)$ by acting on the contracted tree T_F.
(2) Node u is non-micro and non-breaking. Since u is not a node of T_F, we pick z as the lowest breaking ancestor of u in T^*. Tabulating z for each u takes $O(n \log n)$ bits (see below). Hence $z \in T_F$. The path $\pi(u)$ lies in T^* and can then be decomposed into two subpaths: one connecting T's root r to the breaking node z, and the other being a unary path connecting z to u (and formed by all non-breaking nodes). The first path-sum can be executed in T_F, whereas the other path-sum needs some specific data structure over the unary paths of T^* (formed by non-breaking nodes).
(3) Node u is micro. Let r_j be the root of its enclosing micro-tree Q_j. Tabulating r_j for all micro-nodes u takes $O(n \log n)$ bits (see below). The parent of r_j, say $f(r_j)$, is non-micro, for the maximality of Q_j. Therefore the path $\pi(u)$ can be decomposed in two subpaths: one lies in T^* and connects its root r to $f(r_j)$, the other lies in Q_j and connects r_j to u. Consequently, the first path-sum can be executed in T^*, whereas the other path-sum can be executed in Q_j.

We are therefore left with the design of succinct data structures to support constant-time path-sum queries over the contracted tree T_F, the unary paths in T^*, and the micro-trees Q_js. We detail their implementation below.

Path-sum over the T_F. Given the labeled tree T_F, we build the integer sequence L_{T_F} and the array r_{T_F}, similarly as done in Section 4. Since there are $O(n/\log n)$ breaking nodes, we have that $|L_{T_F}| = O(n/\log n)$ and its elements are in the range $[-\log n, +\log n]$. Now we define K as the data structure of Lemma 1 built on the sequence L_{T_F} (here $l = O(\log n)$), thus taking $O(n \log\log n/\log n) = o(n)$ bits. By Lemma 3, the path-sum query involving a breaking node in T_F can then be answered in constant time using K and r_{T_F}.

Path-sum over the unary paths in T^*. We serialize the unary paths in T^* according to the preorder visit of this tree. Let us denote by P_{T^*} the resulting sequence of ternary labels of those (serialized) nodes. Notice that P_{T^*} is similar in vein to L_{T^*}, but it avoids the double storage of the node labels. Nonetheless path-sum queries over unary paths of T^* can still be executed as prefix-sum queries over P_{T^*}; but with the additional advantage of saving a factor 2 in the space complexity. More specifically, any path-sum query over a unary path in T^* actually boils down to a *range-sum query* over the sequence P_{T^*}, because the paths are unary and node labels are written in P_{T^*} according to a pre-visit of T^*. Additionally, a range-sum query over P_{T^*} can be implemented as a difference of two prefix-sum queries over the same sequence. As a result (see Theorem 1), we can build a wavelet tree on P_{T^*} taking $(\log 3)|P_{T^*}| + o(|P_{T^*}|)$ bits of space (since $|\Sigma| = 3$ and $H_0(P_{T^*}) \leq \log|\Sigma|$). Given this wavelet tree and an array $\mathtt{pre}_{T^*}[1, n]$, which stores the rank of the macro-nodes in the preorder visit of T^*, the path-sum queries over the unary paths in T^* can be answered in constant time.

Path-sum over the micro-trees. Here we exploit the fact that micro-trees are small enough so that we can explicitly store the answer to all possible path-sum queries over all of them in succinct space. We note that any path-sum query over a micro-tree Q can be uniquely specified by a triple $\langle Q, \ell(Q), i\rangle$, where Q denotes the micro-tree structure, $\ell(Q)$ denotes the ternary labeling of Q, and i is the preorder rank in Q of the queried node (hence $i \leq \mu$). We then build a table C that tabulates all possible path-sum queries over micro-trees, indexed by triplets $\langle Q, \ell(Q), i\rangle$. To access C, we need an encoding for the triplet: i.e., we encode the Q's structure via any succinct tree encoding of at most 2μ bits (see e.g. [12,17]), and encode $\ell(Q)$ via the string P_Q which consists of no more than μ ternary labels (obtained by visiting in preorder Q, see above). Consequently, C consists of $2^{2\mu} \times 3^{\mu} \times \mu$ entries, each storing an integer smaller than μ in absolute value. Table C thus takes less than $O(n \log n \log\log n)$ bits. As a result, a path-sum query over a micro-tree Q can be answered in constant time, provided that we have constant-time access to its micro-tree encoding and labeling. To this aim, we store all structural encodings of the Q_i's in one string, thus taking $O(n)$ bits overall. Also, we create the string S_ℓ, obtained by juxtaposing the encodings of the labelings $\ell(Q_i)$ (i.e., the strings P_{Q_i}), for all micro-trees Q_i of T. Note that S_ℓ depends on the labeling ℓ of T. Finally we compress and index S_ℓ via the succinct data structure of Corollary 1. This way, we can retrieve any $\ell(Q_i)$ in constant time, taking a total of $|S_\ell| \log 3 + o(|S_\ell|)$ bits.

To complete the description of our solution we just need to store some other auxiliary arrays which take $O(n \log n) = o(n^2)$ bits overall:

- the array encoding the node type– (non)micro, breaking.
- the array of parent pointers of T's nodes (useful to execute path-sums in micro-trees);
- the arrays storing for each micro-node the root of its micro-tree and its preorder rank inside it (useful to execute path-sums in micro-trees).
- the array storing for each unary non-breaking node the top node in its maximal unary path (useful to execute path-sums of non-micro and non-breaking nodes).

At this point, we are left with the orchestration of all data structures sketched above in order to provide a succinct data structure for performing path-sum queries over the ternary labeled tree T, and then apply Theorem 2. We indeed use the above macro–micro-tree decomposition on T (and its labeling ℓ) and define:

- the succinct data structures $D(E(T), \ell)$, as the combination of data structure K built on T_F, the wavelet tree built on P_{T^*}, and the compressed indexing of S_ℓ. These data structures take $(\log 3)(|P_{T^*}| + |S_\ell|) + o(|P_{T^*}| + |S_\ell| + n) = (\log 3)n + o(n)$ bits.
- the encoding $E(T)$ as the combination of the table C, the encodings of the micro-tree structures, and all other auxiliary arrays mentioned above, for a total of $o(n^2)$ bits.

We then plug this data structure into Theorem 2, and get the following result:

Theorem 4. *There exists a representation for the distance matrix* **D** *that uses at most* $n^2(\log 3) + o(n^2)$ *bits, and takes constant time to access anyone of its entries.*

Proof. The space bound has been proved above. The time bound derives from the three-cases analysis made above and the use of $D(E(T), \ell)$ data structure which guarantees constant-time prefix-sum queries. □

The previous solution does not deploy the symmetry idea sketched at the end of Section 5. We then apply it to further halve the above space occupancy:

Corollary 3. *There exists a representation for* **D** *that uses at most* $n^2(\frac{\log 3}{2}) + o(n^2)$ *bits, and takes constant time to access anyone of its entries.*

7. Coping with weighted graphs

Consider a weighted graph G whose edges have positive integer weights drawn from $[k]$, where k is an arbitrarily large positive constant. By the same argument used in Section 1, the information-theoretic lower bound to store the distance matrix of this graph is $\Omega(n^2 \log k)$ bits. The classic plain representation of G requires $O(n^2 \log(kn))$ bits, which is asymptotically larger.

We can easily extend the algorithmic reduction outlined in Theorem 2 to work on weighted graphs too. The main idea is to turn the representation of the distance matrix **D** into the succinct encoding of a tree labeled on the alphabet $\{-k, \ldots, 0, \ldots, k\}$. Following the same ideas of Sections 5 and 6, we can first derive an encoding taking $n^2 \lceil \log(2k+1) \rceil + o(n^2)$ bits, and then plug this result into the macro–micro technique to halve the space occupancy. The distance query still takes constant time, independent of k.

We mention that the encoding technique for labeled trees remains essentially the same as the one in Section 6, with two small changes: the use of the prefix-sum structure of Lemma 1, rather than the wavelet tree of Theorem 1, and the setting of $w = \log_k n$ in the micro–macro tree partition. The query algorithm and the space complexity analysis remain the same, and are still asymptotically time and space optimal.

Corollary 4. *Let G be a graph with positive integer edge weights, bounded by a constant k. There exists a representation for the distance matrix of G that uses at most* $n^2 \frac{\lceil \log(2k+1) \rceil}{2} + o(n^2)$ *bits, and takes constant time to access anyone of its entries.*

8. Conclusion and open problems

We have studied the problem of succinctly encoding the All-Pair-Shortest-Path matrix **D** of an n-node (un)weighted and undirected graph. We have designed compact representations which are asymptotically time and space optimal, and result close to the information-theoretic lower bound by a small constant factor. Our first solution of Section 5 is so simple that may be easily implemented by adopting any Wavelet Tree implementation available in the literature (see e.g. [5,8]); our second solution of Section 6 is a little bit more involved and closer to the information-theoretic lower bound for the space complexity of the distance matrix **D**.

We leave two interesting open problems. The first one concerns with (dis)proving the existence of a succinct data structure that achieves $n^2/2 + o(n^2)$ bits of space and supports distance queries in constant time. The second question deals with the design of a solution whose space complexity depends on the number m of edges in the graph G, and still

guarantees constant time to compute *exactly* the shortest-path distance between any pair of its nodes. In the case of sparse graphs, the information-theoretic lower bound is $2m \log \frac{n}{m} - \Omega(m) \ll n^2$ bits, which is much lower than $n^2/2$. Such a solution would be of big practical relevance in applications that manage very sparse large graphs, as the ones that occur in Web mining applications.

References

[1] J. Barbay, M. He, J.I. Munro, S. Srinivasa Rao, Succinct indexes for string, bynary relations and multi-labeled trees, in: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms, SODA, 2007, pp. 680–689.
[2] Michael A. Bender, Martin Farach-Colton, The level ancestor problem simplified, Theoretical Computer Science 321 (1) (2004) 5–12.
[3] D. Benoit, E. Demaine, I. Munro, R. Raman, V. Raman, S. Rao, Representing trees of higher degree, Algorithmica 43 (2005) 275–292.
[4] P. Ferragina, F. Luccio, G. Manzini, S. Muthukrishnan, Structuring labeled trees for optimal succinctness, and beyond, in: Proc. 46th IEEE Symposium on Foundations of Computer Science, FOCS, 2005, pp. 184–193.
[5] P. Ferragina, G. Navarro, Pizza&Chili corpus home page. http://pizzachili.dcc.uchile.cl/ or http://pizzachili.di.unipi.it/.
[6] P. Ferragina, R. Venturini, A simple storage scheme for strings achieving entropy bounds, Theoretical Computer Science 372 (1) (2007) 115–121.
[7] Paolo Ferragina, Roberto Grossi, Ankur Gupta, Rahul Shah, Jeffrey Scott Vitter, On searching compressed string collections cache-obliviously, in: PODS, 2008, pp. 181–190.
[8] L. Foschini, R. Grossi, A. Gupta, J. Vitter, Fast compression with a static model in high order entropy, in: Procs of IEEE Data Compression Conference, DCC, 2004, pp. 62–71.
[9] R. Grossi, A. Gupta, J. Vitter, High-order entropy-compressed text indexes, in: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms, SODA, 2003, pp. 841–850.
[10] R. Grossi, A. Gupta, J. Vitter, When indexing equals compression: Experiments on compressing suffix arrays and applications, in: Proc. 15th ACM-SIAM Symp. on Discrete Algorithms, SODA, 2004, pp. 636–645.
[11] A. Gupta, W.K. Hon, R. Shah, J.S. Vitter, Compressed data structures: dictionaries and data-aware measures, Theoretical Computer Science 387 (3) (2007) 313–331.
[12] G. Jacobson, Space-efficient static trees and graphs, in: Proc. 30th IEEE Symposium on Foundations of Computer Science, FOCS, 1989, pp. 549–554.
[13] J. Jansson, K. Sadakane, W.K. Sung, Ultra-succinct representation of ordered trees, in: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms, SODA, 2007, pp. 575–584.
[14] V. Mäkinen, G. Navarro, Rank and select revisited and extended, Theoretical Computer Science 387 (3) (2007) 332–347.
[15] M. Mendel, A. Naor, Ramsey partitions and proximity data structures, in: Proc. of the 47th Annual IEEE Symposium on Foundations of Computer Science, FOCS, 2006, pp. 109–118.
[16] I. Munro, Tables, in: Proceeding of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science, in: LNCS, vol. 1180, Springer-Verlag, 1996, pp. 37–42.
[17] I. Munro, V. Raman, Succinct representation of balanced parentheses, static trees and planar graphs, in: Proc. of the 38th IEEE Symposium on Foundations of Computer Science, FOCS, 1997, pp. 118–126.
[18] I. Munro, V. Raman, Succinct representation of balanced parentheses and static trees, SIAM Journal on Computing 31 (2001) 762–776.
[19] G. Navarro, V. Mäkinen, Compressed full-text indexes, ACM Computing Surveys 39 (1) (2007).
[20] R. Pagh, Low redundancy in static dictionaries with constant query time, SIAM Journal on Computing 31 (2) (2001) 353–363.
[21] R. Raman, V. Raman, S. Srinivasa Rao, Succinct indexable dictionaries with applications to encoding k-ary trees and multisets, in: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms, SODA, 2002, pp. 233–242.
[22] M. Thorup, Compact oracles for reachability and approximate distances in planar digraphs, Journal of the ACM 51 (6) (2004) 993–1024.
[23] M. Thorup, U. Zwick, Approximate distance oracles, in: Proc. of the 33rd Symposium on Theory of Computing, STOC, 2001, pp. 183–192.