On Compact Representations of All-Pairs-Shortest-Path-Distance Matrices*

Igor Nitto and Rossano Venturini

Department of Computer Science, University of Pisa
{nitto,rossano}@di.unipi.it

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 $(n^2 - n)/2$ 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.

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* [9] when its space is *close* to 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. The term "close to" (the information-theoretic lower bound) usually means either "equal plus some low-order terms", or "up to a constant factor from" (the information-theoretic lower bound), where the constant is pretty much close to 1. Nowadays there exist succinct versions of various data structures and data types: bitmap vectors [4,16,17], dictionaries [8], strings [14], (un)labeled trees [3,5,10], binary relations and graphs [12,1], etc.. In this paper we contribute to the design of new succinct data structures by investigating the field of compact representations 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

* This work has been partially supported by the Italian MIUR grants PRIN Main-Stream and Italy-Israel FIRB "Pattern Discovery Algorithms in Discrete Structures, with Applications to Bioinformatics", and by the Yahoo! Research grant on "Data compression and indexing in hierarchical memories".

P. Ferragina and G. Landau (Eds.): CPM 2008, LNCS 5029, pp. 166–177, 2008.

are not connected). $\mathbf{D}$ is called the matrix of All-Pairs-Shortest-Path distances in G (or distance matrix, for brevity) and it is typically stored in $O(n^2)$ memory words, thus taking $n^2 \log(n+1)$ bits in total.[1]

Various authors have investigated the problem of designing succinct graph encodings for supporting the retrieval of either the *adjacency list* of a node (see [12,13] and references therein), or the *approximate distance* between node pairs in various types of graphs (see [19,18] and references therein). When *exact* distances are needed, it is still open whether it is possible to deploy the intrinsic structure of matrix $\mathbf{D}$ to devise a representation which uses $o(n^2 \log n)$ bits and is as much close as possible to the information-theoretic lower bound of $n^2/2$ bits.[2] In our paper we show how to match *asymptotically* the above lower bound, by providing a succinct storage scheme for $\mathbf{D}$ which achieves a bit-space complexity that is far from the information-theoretic minimum by a multiplicative factor $\log_2 3 \simeq 1.585$, and is still able to retrieve in constant time any node-pair distance in G. We remark that the interest in space-efficient representations of shortest path distances for such a simple (undirected and unweighted) graphs is driven by applications in the field of graph layouts via Multi-Dimensional Scaling [15]. Here the distance matrix is deployed to produce a layout of the graph in the plane that closely preserves the shortest-path metric. Technically, our paper is based on an algorithmic reduction (detailed in Theorem 2) which turns the storage of $\mathbf{D}$ into the succinct storage of (ternary) labeled trees and (ternary) sequences, for which we properly adapt and orchestrate known compressed data structures. Using this algorithmic scheme we obtain two results: a simple compact representation of $\mathbf{D}$ requiring $(\log_2 3)n^2 + o(n^2)$ bits of storage and $O(1)$ access time to any of its entry (Corollary 2), and a more sophisticated one which reduces the space complexity to $\left(\frac{1}{2}\log_2 3\right) n^2 + o(n^2)$ bits (Corollary 3) without slowing down the access time.

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 zero-th 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 [7] is an elegant and powerful data structure that supports rank/select primitives over sequences drawn from arbitrarily large alphabets, and achieves entropy-bounded space occupancy.

---

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

[2] 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 $2^{n(n-1)/2}$.

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 i-th occurrence of character $c \in \Sigma$ in S.*

In addition to rank/select primitives, the design of our compact representations 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 [11]:

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.*

An essential fact in our technique will be also 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.*

In the rest of this paper, we will also make use of the following two strong structural properties of the distance matrix $\mathbf{D}$:

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

where u, v, w are any triplets 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 for these labels is $O(n \log n) = o(n^2)$ bits, thus resulting bounded above by the other terms occurring in the space bounds of our representation.

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 some useful notation and terminology.

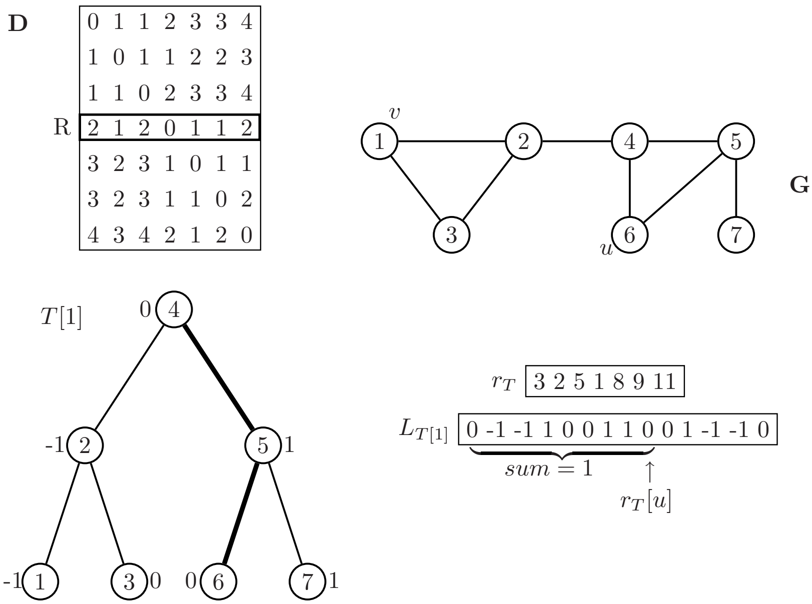

Fig. 1. (Top) A graph G and its distance-matrix $\mathbf{D}$. (Bottom) An example of labeled tree $T[1]$, relative to node $1 \in G$, and the associated arrays $L_{T[1]}$ and $r_{T[1]}$. According to Lemma 3 the sum of the labels on $\pi(6)$ is equal to the prefix-sum in $L_{T[1]}[1, r_{T[1]}[6]] = L_{T[1]}[1,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

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;
- $\texttt{pre}(u)$ the rank of u in the preorder visit of T (i.e., integer in $[n]$).
- $\pi(u)$ the downward path in T which connects 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 labellings 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 such that $\ell_v(u) = \mathbf{D}[u, v] - \mathbf{D}[f(u), v]$, where $u \in T$. This is a ternary labelling because of the triangle inequality and the adjacency of u and $f(u)$ in G. The labeled tree resulting by the ternary labelling ℓ_v applied to T is hereafter denoted by $T[v]$. An illustrative example is given in Fig. 1.

The labeled tree $T[v]$ offers 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. Note that this 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 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 sum the value $\mathbf{D}[r, v]$ to the result of the path sum-query over $\pi(u)$ in $T[v]$. 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 query data structures built upon the labeled trees $T[v]$, for all nodes $v \in G$.

Theorem 2. *Let T be a tree of n nodes, $E(T)$ be an encoding of T's structure, and let ℓ be a labelling of T's nodes over the ternary alphabet $\{-1, 0, 1\}$. Suppose that there exists a succinct data structure $D(E(T), \ell)$ that occupies $S(n)$ bits to store ℓ 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 as detailed above, namely: 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 r-th 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 two data structures occupy $|E(T)| + o(n^2)$ bits. The n path-sum data structures require $nS(n)$ bits, because v ranges over all n nodes in T. The claimed space bounds therefore follows.

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 $O(1)$ arithmetic and table-lookup operations. $\square$

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 pre-visit of node u ends (i.e., its subtree has been completely 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 Figure 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 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 $v \in G$), and suitable succinct encodings of the tree T (namely $E(T)$). 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

The labeled trees we are interested in succinctly encodings, are the ternary-labeled trees $T[v]$ introduced in the proof of Theorem 2, as a result of the ternary labeling ℓ_v. Given $T[v]$, the corresponding sequence $L_{T[v]}$ is drawn from the ternary alphabet $\{-1, 0, 1\}$. In order to compute efficiently the prefix-sum queries over $L_{T[v]}$, we use the wavelet tree data structure (see Theorem 1). 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 and the space required to store the wavelet tree is $2(\log 3)n + o(n)$ bits (since $|\Sigma| = 3$ and $H_0(S) \leq \log |\Sigma|$).

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 2–3, we obtain:

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 any 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 $L_{T[1]}[1, r_T[6]]$, 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 far from such lower bound of a multiplicative factor $4 \log 3 \simeq 6.34$. 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 structural properties of G and T, in order to come as much close as possible to the lower bound 0.5. A first step in this direction is obtained by exploiting the symmetry of matrix $\mathbf{D}$, and thus storing just the suffix $L_{T[v]}[1, r_T[v]]$ 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

In this section we show how to further halve the space complexity by deploying 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]). Second, 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 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 Figure 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.

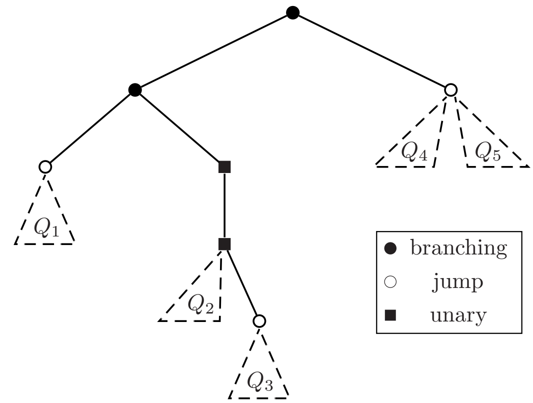

Fig. 2. Macro-micro tree partition

In this section we will 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 $O(n/\log n)$ 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^* (i.e., jump nodes), 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 $O(n/\log n)$ 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 its father 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^*. 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. The parent of r_j, say $f(r_j)$, is a jump node (and thus $f(r_j) \in T_F$), by definition. Therefore the path $\pi(u)$ can be decomposed in two subpaths: one lies in T_F 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_F, 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, $|L_{T_F}| = O(n/\log n)$ and its elements are in the range $[-\log n, +\log n]$. Now we define K as the data structure of Theorem 1 built on 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 pre-order 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, we build a wavelet tree on P_{T^*} (see Theorem 1) 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 pre-order 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. [9,12]), and encode $\ell(Q)$ via the string P_Q which consists of no more than μ ternary labels (obtained by visiting in pre-order 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 labellings $\ell(Q_i)$ (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 pre-order 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, for a total of $o(n^2)$ bits.

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

Theorem 4. *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 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* $\mathbf{D}$ *that uses at most* $n^2(\frac{\log 3}{2}) + o(n^2)$ *bits, and takes constant-time to access any of its entries.*

7 Conclusion and Open Problems

We have studied the problem of succinctly encoding the All-Pair-Shortest-Path matrix of an n-node unweighted 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.

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 occupancy 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 fact, in the case of sparse graphs, the information-theoretic lower bound is $2m \log \frac{n}{m} - \Omega(m) \ll n^2$ bits. Such a solution would be of big practical relevance in applications that manage very sparse large graphs.

Acknowledgments. The authors wish to thank Paolo Ferragina for useful comments and his help in improving the exposition of this paper.

References

1. Barbay, J., He, M., Munro, J.I., Srinivasa Rao, S.: Succinct indexes for string, bynary relations and multi-labeled trees. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA) (2007)
2. Bender, M.A., Farach-Colton, M.: The lca problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
3. Benoit, D., Demaine, E., Munro, I., Raman, R., Raman, V., Rao, S.: Representing trees of higher degree. Algorithmica 43, 275–292 (2005)
4. Brodnik, A., Munro, I.: Membership in constant time and almost-minimum space. SIAM Journal on Computing 28(5), 1627–1640 (1999)
5. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Structuring labeled trees for optimal succinctness, and beyond. In: Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 184–193 (2005)
6. Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. Theor. Comput. Sci. 372(1), 115–121 (2007)
7. Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 841–850 (2003)
8. Gupta, A., Hon, W.K., Shah, R., Vitter, J.S.: Dynamic rank/select dictionaries with applications to XML indexing. Technical Report Purdue University (2006)

9. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 549–554 (1989)
10. Jansson, J., Sadakane, K., Sung, W.K.: Ultra-succinct representation of ordered trees. In: Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA) (2007)
11. Mäkinen, V., Navarro, G.: Rank and select revisited and extended. Theor. Comput. Sci. 387(3) (2007)
12. Munro, I., Raman, V.: Succinct representation of balanced parentheses, static trees and planar graphs. In: Proc. of the 38th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 118–126 (1997)
13. Munro, I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Computing 31, 762–776 (2001)
14. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Comput. Surv. 39(1) (2007)
15. Working Group on Algorithms for Multidimensional Scaling. Algorithms for multidimensional scaling. DIMACS Web Page, `http://dimacs.rutgers.edu/Workshops/Algorithms/AlgorithmsforMultidimensionalScaling.html`
16. Pagh, R.: Low redundancy in static dictionaries with constant query time. SIAM Journal on Computing 31(2), 353–363 (2001)
17. Raman, R., Raman, V., Srinivasa Rao, S.: Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In: Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 233–242 (2002)
18. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. J. ACM 51(6), 993–1024 (2004)
19. Thorup, M., Zwick, U.: Approximate distance oracles. In: STOC, pp. 183–192 (2001)