Simpler, faster and shorter labels for distances in graphs [1]

Jules Bachmann

October 2018

1 Introduction

Distance oracle for graphs. For an undirected and unweighted graph $G = (V, E)$, we want to build a data structure, which for any query $q = \{u, v\}$ efficiently computes and returns the shortest distance $dist_G(u, v)$ between the two nodes u and v. We want this data structure to be as small as possible, while also having a short query time. In the following, we instinctively show a lower bound for the size of such a data structure.
We know that any graph with n nodes can have up to $\frac{n(n-1)}{2} \leq \frac{n^2}{2}$ edges, and hence there are $2^{\frac{n^2}{2}}$ different graphs with n nodes. Since all of these graphs are different, for any two of these graphs, call them G and H, there is an edge $\{u, v\}$ present in only one of them. Without loss of generality, we assume $\{u, v\} \in G$ and obtain the following: $dist_G(u, v) = 1 \neq dist_H(u, v)$. The encoding of G and H in the said data structure must therefore be different which results in us needing $2^{\frac{n^2}{2}}$ different bitstrings. Each bitstring must hence have size at least $\log(2^{\frac{n^2}{2}}) = \frac{n^2}{2}$ which is the lower bound for the size of a distance oracle for general graphs.

Labels for distances in graphs. Instead of having one centralized oracle, we can assign to each vertex v a label l such that for two vertices u and v, we can find the distance $dist_G(u, v)$ by looking at their labels and their labels only. We call this a distance labeling scheme.
Since the rules stated before still hold, we have a lower bound for the label size of $\frac{n}{2}$. For more specific graphs we can achieve better lower bounds. However, the known algorithms for this problem are significantly above the proven lower bound, with the exception of trees. The following table presents the existing best results without second order terms for different graphs:

Graph	Lower bound	Upper bound
General graphs[1]	$O(\frac{n}{2})$	$\frac{\log 3}{2}n + o(n)$
Sparse graphs	$O(\sqrt{n})$[7]	$o(n)$ [6]
Planar graphs	$\Omega(n^{\frac{1}{3}})$ [7]	$O(\sqrt{n})$[4]
Trees[5]	$\frac{1}{4}\log^2 n - O(\log n)$	$\frac{1}{4}\log^2 n + o(\log^2 n)$

Note that the upper bounds given for general graphs is obtained through the method which will be presented here and is a huge improvement to the previous upper bound which was $(\log 3)n$ for a decoding time of $O(n/\log n)$. The labeling scheme presented here will, in addition to its space, need a decoding time of $O(1)$. In this presentation, we do look only at unweighted graphs, but the method presented can be easily adapted to weighted graphs.

Encoding. Since we want to minimize the space used for storing the labels, we will use following encoding method:

Lemma 1. *A table with n elements from an alphabet σ can be represented in a data structure of $\lceil n \ log \ |\sigma| \rceil$ bits.*

The proof for this lemma is trivial.

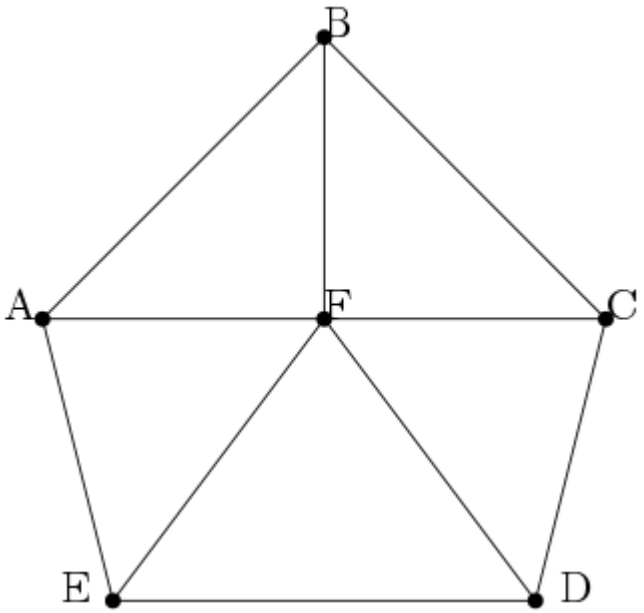
Lemma 2. *A table with n integral entries in [-k,k] can be represented in a data structure of O(n log k) bits to support prefix sums in constant time.*

To prove this lemma, please refer to [3]

Naive algorithm. A very direct approach for a connected, undirected graph $G = (V, E)$ would be for every node $v \in V$ to store the shortest distance to the $n-1$ other nodes in G. Let's now numerate these n nodes from 1 to n, such as $V = \bigcup_{i=1}^{n} v_i$. Instead of storing the distances to all other nodes in the label of each node, for a node v_i we can store only the distances to the $\lfloor n/2 \rfloor$ nodes v_k where $k \in [i+1, i+\lfloor n/2 \rfloor] \pmod n$. For 2 arbitrary node i and j, we can assume $i < j$ without loss of generality, the label of node i will contain the distance to j if $j \leq i + \lfloor n/2 \rfloor$, whereas the label of node j will contain the distance to i if $j > i + \lfloor n/2 \rfloor$.
Since every node $v \in V$ has a label containing the distances to $\lfloor n/2 \rfloor$ nodes, and assuming the largest shortest distance between 2 nodes is $dist_G(x, y), x, y \in V$ we can encode all the labels in space $O(n \cdot \frac{n}{2} \cdot \log(dist_G(x, y))) = O(n^2 \log(dist_G(x, y)))$ using Lemma 1.

2 Labeling schemes

In order to facilitate the understanding of the methods presented here, the following graph $G' = (V', E')$ will be used as an example during this presentation:



Simple algorithm. Let's first define for nodes $x, u, v \in V$:

$$\delta_x(u, v) = dist_G(x, v) - dist_G(x, u)$$

Using the triangle inequality we observe that

$$-dist_G(u, v) \leq \delta_x(u, v) \leq dist_G(u, v)$$

In particular, $\delta_x(u, v) \in [-1, 1]$ whenever u, v are adjacent.
Given a path $v_0, ..., v_t$ of nodes in G, we have : $\delta_x(v_0, v_1)+\delta_x(v_1, v_2)+\delta_x(v_2, v_3)+ ... + \delta_x(v_{t-1}, v_t) = dist_G(x, v_1) - dist_G(x, v_0) + dist_G(x, v_2) - dist_G(x, v_1) + dist_G(x, v_3) - dist_G(x, v_2) + ... + dist_G(x, v_t) - dist_G(x, v_{t-1}) = dist_G(x, v_t) - dist_G(x, v_0)$ which shows that the δ_x - values have a telescoping property. Therefore:

$$\delta_x(v_0, v_t) = \sum_{i=1}^{t} \delta_x(v_{i-1}, v_i)$$

Out of this, we can compute $dist_G(x, v_t)$ by adding $dist_G(x, v_0)$ to the sum computed above:

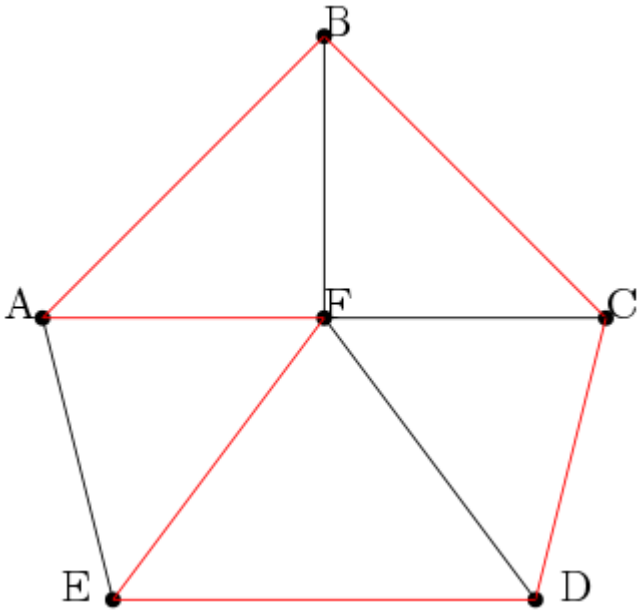
$$dist_G(x, v_t) = dist_G(x, v_0) + \sum_{i=1}^{t} \delta_x(v_{i-1}, v_i)$$

Let's now consider the shortest closed walk when not taking weights into account, the Hamilton walk $v_0, ..., v_{h-1}$ of length h of our graph G, with $n \leq h \leq 2n - 2$. Given nodes $x, y \in V$, we find i, j such that $x = v_i, y = v_j$, assuming $i \leq j$ without loss of generality. We can then compute $dist_G(x, y)$, using the same trick as in the naive algorithm, as the sum of at most $\lfloor h/2 \rfloor \delta_x$ values:

$$\text{for } j \leq i + h/2 : \; dist_G(x, y) = dist_G(v_i, v_j) = \sum_{k=i}^{j-1} \delta_x(v_k, v_{k+1})$$

$$\text{for } j > i + h/2 : \; dist_G(x, y) = dist_G(v_j, v_i) = \sum_{k=j}^{i-1} \delta_x(v_k, v_{k+1}), \text{ with the indices being counted modulo } h \text{ here.}$$

For our example graph G' we have following Hamiltonian walk:



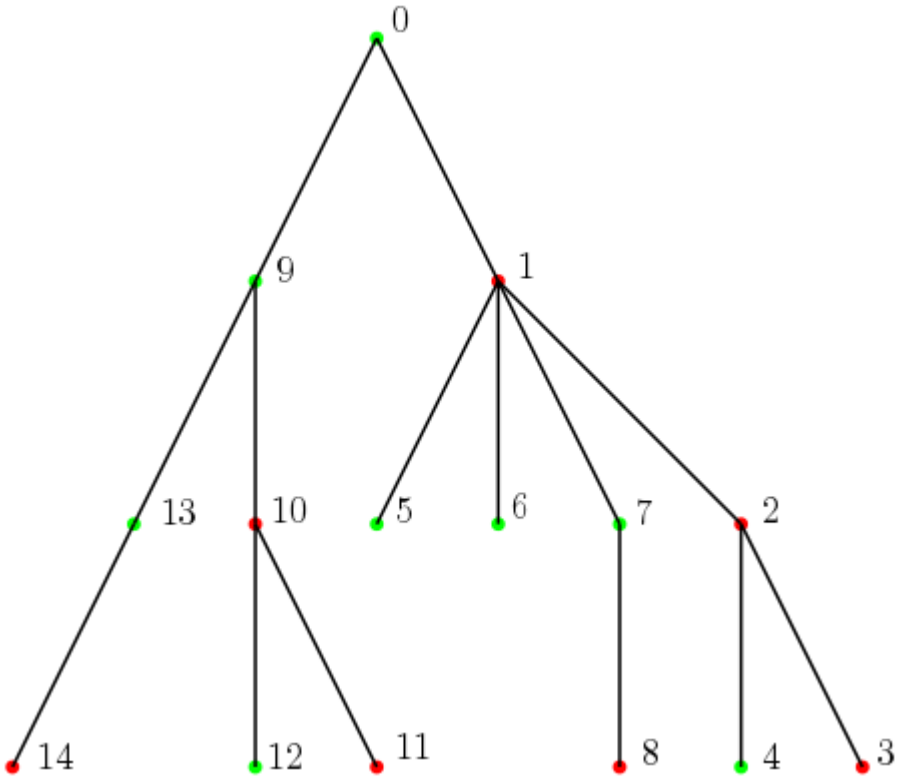
Where $A = v_0$ and $F = v_5$. We take A as our x and E as our y, which correspond to v_0 and v_4 respectively. The distance between A and D is also computed by the following: $dist_{G'}(A, D) = dist_{G'}(v_0, v_3) = \delta_A(v_0, v_1) + \delta_A(v_1, v_2) + \delta_A(v_2, v_3) = (1 - 0) + (2 - 1) + (2 - 2) = 2$.

With this we can construct a first distance labelling scheme where the label $l(x)$ of each node x, consists of the following:
- A number $i \in [0, n - 1]$ such that $x = v_i$
- The $\lfloor h/2 \rfloor$ values $\delta_x(v_k, v_{k+1})$ for $k = i, ..., i + \lfloor h/2 \rfloor \pmod h$

As seen above, $l(x)$ is sufficient to compute $dist_G(x, y)$ for an arbitrary y.
For node E in our example graph G', we would obtain the following $l(E)$:
$4, \delta_E(E, F), \delta_E(F, A), \delta_E(A, B)$
In order to reduce the size of the labels, we encode $l(x)$ as $\lceil \frac{1}{2} h \log 3 \rceil + \lceil \log h \rceil$ using Lemma 1. If G is Hamiltonian, i.e $h = n$ as is our example G', this gives a labelling scheme of size $\lceil \frac{1}{2} n \log 3 \rceil + \lceil \log n \rceil$, while in the general case we have $\lceil (n - 1) \log 3 \rceil + \lceil \log n \rceil$ since $h \leq 2n - 2$.

Heavy-light decomposition of trees.[2] Let's look at something a bit different. Let T be a tree rooted at r. We classify the nodes $v \in T$ as either heavy

of light using the following method:
For every non-leaf node v we look at its children, select one with the biggest subtree and classify that children as heavy. All other children are light; the root r is also light.
We call *apex* of v the nearest light ancestor of a node v. By removing the edges between light nodes and their parents we divide T into a collection of heavy paths. We now enumerate the nodes in t in a depth-first manner with heavy children being visited first and denote the number of a node by $dfs(v)$. This will result in nodes on a heavy path always having numbers in consecutive order, with the root r having the number 0. Now, each node v is assigned a label $l_t(v)$ consisting of the sequence of dfs-values of its first and last ancestor on each heavy path, starting with r and down to the root. The first and last ancestor for a given path can be the same if only one node of that given path is used to attain v.



The graph above will serve as an example, with heavy nodes colored in red, where the label of the node with dfs number 12 will be

$$0, 0, 9, 10, 12, 12$$

In the general case, the label $l_T(v)$ of a node v will look like this:

$$l_1, ..., h_1, l_2, ..., h_2, ..., l_t, ..., h_t$$

Where $l_1 = dfs(r) = 0$ and $h_t = dfs(v)$ and l_i and h_i are the first and last ancestor of the i-th heavy path on the path from r to v. In addition to the label $l_T(v)$, we also store the label $l'_T(v)$ which consists of the distances $dist_G(l_i, v)$ and $dist_G(h_i, v)$.
We know that the number of heavy paths between the root and any leaf is at most $\log n$, since if the leaf is placed deeper than $\log n$, we would have heavy paths spanning multiple nodes on the path r to the said leaf and still end up with $\log n$ or less heavy paths to traverse. Therefore, $l_T(v)$ is a sequence of at most $2 \log n$ numbers from $[0, n)$. We can therefore encode this sequence in $O(\log^2 n)$ bits. $l'_T(v)$ is a sequence of at most $2 \log n$ numbers smaller than n, which allows us to encode $l'_T(v)$ in $O(\log n \cdot \log(n))$ bits. Both labels together can therefore be encoded in $O(\log^2 n)$ bits.

Main algorithm. We now consider a connected graph G again, and build a shortest path tree T rooted at an arbitrary node r for it. Using the enumeration of nodes presented in the heavy-light decomposition of trees, we can now construct a distance labeling scheme similar to the one built for the simple algorithm, but by using the dfs-enumeration of nodes instead of the Hamiltonian path, and instead of saving the multiple $\delta_x(v_k, v_{k+1})$ for a node x as above, we save $\delta_x(parent(v), v)$ for all nodes v with $dfs(x) < dfs(v) \leq dfs(x) + \lfloor n/2 \rfloor \pmod n$. These δ_x values can be encoded in $\lceil \frac{1}{2} n \log 3 \rceil$ bits due to Lemma 1. For each node $x \in V$ we therefore assign a label $l(x)$ consisting of the $n/2$ δ_x just given and the labels $l_T(x)$ and $l'_T(x)$ presented before.
It now remains to show that for two nodes $x \neq y$, $l(x)$ and $l(y)$ are enough to determine the shortest distance between x and y. We assume without loss of generality, that $l(x)$ contains $\delta_x(parent(y), y)$. We define z to be the nearest common ancestor of x and y, which means it has to be the last node of a heavy path used to reach either x or y. It follows of this, that $dfs(z)$ has to be present in either $l_T(x)$ or $l_T(y)$. Additionally, all nodes $v \in T_{(z,y]}$ do have dfs-values such that $\delta_x(parent(v), v)$ is contained in $l(x)$.
T being a shortest path tree, $dist_G(x, z) = dist_T(x, z)$. Since z appears in either $l_T(x)$ or $l_T(y)$, we can obtain either $dist_T(x, z)$ directly from $l'_T(x)$ or compute it as $dist_T(x, z) = dist_T(x, r) - dist_T(y, r) + dist_T(y, z)$. This allows us to compute

$$dist_G(x, y) = dist_G(x, z) + \sum_{v \in T(z,y]} \delta_x(parent(v), v).$$

Since $l(x)$ contains all the needed δ_x and the combination of $l_T(x)$ and $l_T(y)$ allows us to determine which values we need, as explained above.
Hence we have proved the following theorem:

Theorem 1. *There exists a distance labeling scheme for graphs with label size* $\frac{1}{2} n \ log(2W + 1) + O(log\ n \cdot log(n))$.

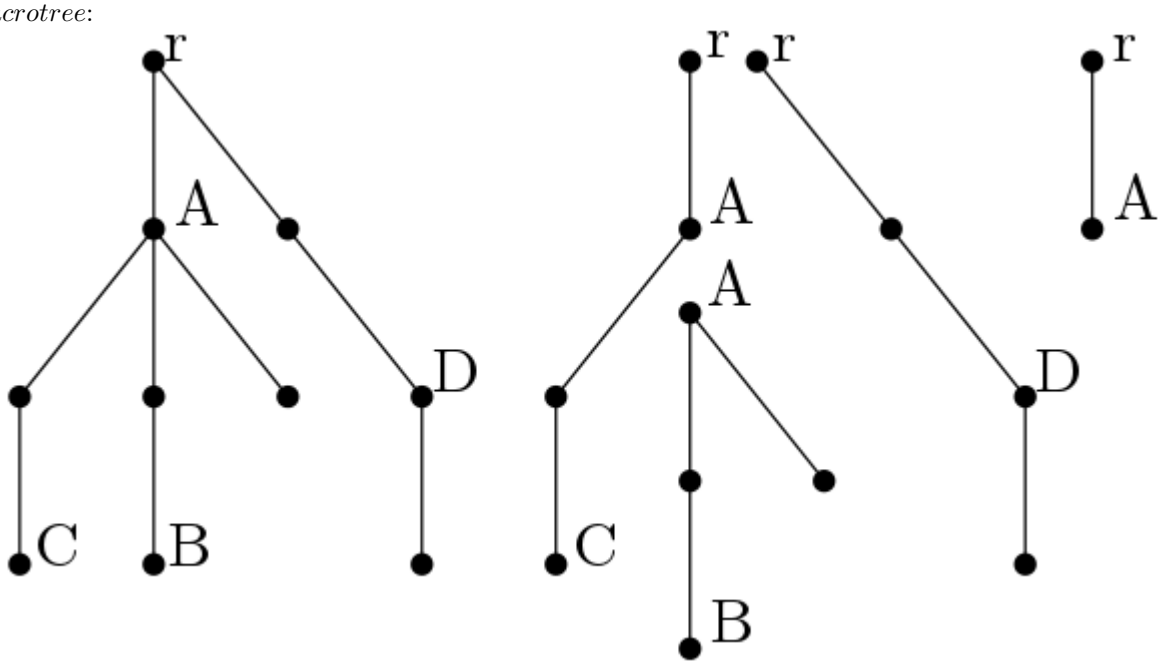
3 Query time

In order to achieve the best possible query time, we will have to rework our labels one more. First, let's take a look at the following:

Lemma 3. *Let k be a positive integer. Every m-edge tree has an edge partition into at most $\lceil m/k \rceil$ trees of at most $2k$ edges*

For proof of this lemma please refer to [1].

Let T be any rooted spanning tree of G. We create an edge-partition $\tau = \{T_1, T_2, ...\}$ of T into rooted subtrees, called $micro - trees$, of at most β edges using lemma 3, with β unknown for now. We define $T^* = T \backslash r$. As the parent relationship in the $micro - trees$ coincides with the one of T, we have $\text{parent}_{T_i}(u) = \text{parent}_T(u)$ for all $u \in T_i^*$. For a node $u \in T^*$, we denote by $i(u)$ the index i such that $u \in T_i^*$ and $\text{MicroRoot}(u) = \text{root}(T_{i(u)}$. For $r = \text{root}(T)$, let $\text{MicroRoot}(r) = r$.
We define the *macrotree* M to have node set $\{\text{Microroot}(u) | u \in G\}$ and an edge between $\text{MicroRoot}(u)$ and $\text{MicroRoot}(\text{MicroRoot}(u))$ for all $u \neq r$. The following example shows the partition of a tree into *microtrees* and the resulting *macrotree*:



In our revisited labeling scheme, the distance x to y will be computed as follows:

$$dist_G(x, y) = dist_G(x, r) + \delta_x(r, \text{MicroRoot}(y)) + \delta_x(\text{MicroRoot}(y), y).$$

Where $dist_G(x, r)$ is saved as part of x's label while $\delta_x(r, \text{MicroRoot}(y))$ is computed as a sum of δ_x values for nodes in the macro tree and is hence referred

as the *macrosum*. As for $\delta_x(\text{MicroRoot}(y), y)$, it is a sum of δ_x values or nodes inside y micro tree and is therefore called the *microsum*.

Macro sum. We have a macro tree M with $O(n/\beta)$ nodes. We take a Hamiltonian walk $v_0, ..., v_{h-1}$ of length $h = O(n/\beta)$ with $v_0 = r$. For given nodes $x, y \in G$, we chose t such as $v_t = \text{MicroRoot}(y)$ and consider the subpath $v_0, ..., v_t$ of the Hamiltonian walk. It is easy to see that

$$\delta_x(r,\text{MicroRoot}(y)) = \delta_x(v_0, v_t) = \sum_{i=0}^{t-1} \delta_x(v_i, v_{i+1})$$

We store these δ_x in a data structure PreFix_x which is stored in x's label, whereas t is stored inside y's label. Since each edge in M connects two nodes in the same micro tree, and the distance inside a micro tree is at most βW, we have $\delta_x(v_i, v_{i+1}) \in [-\beta, \beta]$ for all i. Using this and lemma 2, we can store PreFix_x in $O((n/\beta)\log(2\beta+1)) = O(n\log(\beta)/\beta)$ bits such that prefix sums can be computed in constant time. $t(y)$ in y's label is stored using $O(\log(n/\beta)$ bits.

Micro sum. We now want to store the relevant information needed to compute $\delta_x(\text{MicroRoot}(y), y)$ such that we can decode it in the smallest possible time. We define the shortcut $\delta_x(v) = \delta_x(\text{parent}_T(v), v)$. For each i, we order the nodes in T_i^* in any order. We now define $\delta_x(T_i^*) = (\delta_x(v_1), ..., \delta_x(v_{|T_i^*|}))$, with $v_1, ..., v_{|T_i^*|}$ being the ordered sequence of the nodes of T_i^*. We will store in each x's label the $\delta_x(T_i^*)$ of half the total set of δ_x values, while y's label stores the information for which j's the node v_j lies between $\text{MicroRoot}(y)$ and y.
To store the sequence $\delta_x(T_i^*)$ that consists of $|T_i^*|$ values from $[-1, 1]$ we use an injective function, as described in lemma 1, that maps every sequence of t integers from $[-1, 1]$ into a bit string of length $\lceil t \log 3 \rceil$. In this case, we obtain an encoding of the sequence $\delta_x(T_i^*)$ to a bit string of length $\lceil |T_i^*| \log 3 \rceil \leq \lceil \beta \log 3 \rceil$, as $|T_i^*| \leq \beta$. We denote this encoding by $\text{code}(\delta_x(T_i^*))$.
In order to decode the encoded version of $\delta_x(T_i^*)$ in constant time, we construct a tabulated inverse function code^{-1}.
To store the information about the j's mentioned before in y's label, we save the bit string $\text{mask}(y)$ such that $\text{mask}(y)$ & $\delta_x(T_i^*(y))$ gives back a sequence S similar to $\delta_x(T_i^*(y))$ but with the value of the entries not corresponding to a node v_j being set to 0. The sum of this sequence of integers is the micro sum $\delta_x(\text{MicroRoot}(y), y)$.
We create a tabulated function SumIntegers that sums up the integers of the sequences S, which consist of up to β values in $[-1, 1]$, while the output is a number in $[-\beta, \beta]$.
As long as both input and output can be represented in $O(\log n)$ bits, a lookup in a tabulated function can be done in constant time on the RAM. We therefore set

$$\beta \leq \frac{c \log n}{\lceil \log 3 \rceil}$$

with c being a constant, which does result in the input and output of the tabulated function to be in $O(\log n)$. We will not go into details of space usage of

these tables, just note that as long as $c < 1$, the space used is smaller than the one used by the prefix table, and therefore the code^{-1} and SumIntegers tables can be integrated into the labels without increasing the space asymptotically.

Recall the enumeration of the micro trees done before. Now let $D(x) = \text{code}(\delta_x(T_1^*))$ ... $\text{code}(\delta_x(T_{|\tau|}^*))$ denote the binary string composed of all the $\text{code}(\delta_x(T_i^*))$ in the order $i = 1, 2, ..., |\tau|$ and let $L = |D(x)|$. Note that the length of $\text{code}(\delta_x(T_i^*)$ is the same for all $x \in G$. We therefore denote $p_i \in [0, L)$ the position in the string $D(x)$ where the substring $\text{code}(\delta_x(T_i^*))$ starts.
As before, we will save only half of the δ_x values for a given node x, but we always have to save all the δ_x values of a micro tree. This means that for a node x we will save a table $H(x)$ starting with $\text{code}(\delta_x(T_{i(x)}^*))$ and continuing with the codes of the following micro trees, until it contains at least $n/2\delta_x$ values. We furthermore denote $a(x)$ and $a'(x)$ as the starting and ending positions of the substring containing the code for $T_{i(x)}^*$ in $D(x)$ and $b(x)$ the first bit after the last code contained in $H(x)$.
Hence, $a(x)$, $a'(x)$, $b(x)$ and L are saved in x's label using $O(\log n)$ bits, while $H(x)$ need $\frac{1}{2}n \log 3 + O(fracn \log n \log 3)$ bits. The proof for this won't be detailed here, but can be found in [1]. In one query, we will need to extract at most $\lceil \beta \log 3 \rceil = O(\log n)$ consecutive bits from $H(x)$ in one query, which can be done in consecutive time on the word-RAM.

4 Summary

For the final version of this labeling scheme, the label of a node x is composed of the following:
- $a(x)$, $a'(x)$, mask(x), $t(x)$, $dist_G(x, r)$, L and $b(x)$: $O(\log n)$.
- The table $H(x)$: $\frac{1}{2}n \log 3 + O(\frac{n}{\log n} \log 3)$.
- The prefix table PreFix_x and the tables code^{-1} and SumIntegers: $O(\frac{n}{\log n}((\log 3)^2 + \log \log n \log 3))$.

As for decoding, we can use the following algorithm to find the distance between two nodes x and y in constant time:

Decode$(l(x, G), l(y, G))$
if $(a(x) \leq a(y) < b(x)) \vee (b(x) < a(x) \leq a(y)) \vee (a(y) < b(x) < a(x))$ **then**
  $s = (a(y) - a(x))(modL)$ and $e = (a'(y) - a(x))(modL)$
else
  return Decode$(l(y, G), l(x, G))$
end if
MacroSum = $\text{PreFix}_x(t(y))$
$S = \text{code}^{-1}(H_x[s, ..., e])$& mask$(y)$
MicroSum = SumIntegers(S)
return $\text{dist}_G(x, r)$+ MicroSum + MacroSum

With our labeling scheme we have proven the following:

Theorem 2. *There exists a distance labeling scheme for graphs with edge weights in* $[1, W]$ *using labels of length* $\frac{1}{2}n\log(2W+1)+O(fracn\log n\log 3)(\log\log n))$ *bits and constant decoding time.*

References

[1] Stephen Alstrup, Cyril Gavoille, Esben Bistrup Halvorsen, Holger Petersen *Simpler, faster and shorter labels for distances in graphs* in SODA '16 Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms, pages 338-350
2016

[2] D.D Sleator and R.E Tarjan. *A data structure for dynamic trees.* J. of Computer and System Sciences, 26(3):362 - 391
1983

[3] V. Mäkinen and G. Navarro *Rank and select revisited and extended* Theor. Comput. Sci., 387(3):332-347 November 2007.

[4] P. Gawrychowski and P.Uznański *A note on distance labeling in planar graphs* arXiv:1611.06529 November 2016

[5] O. Freedman, P. Gawrychowski, P. K. Nicholson, O. Weimann *Optimal distance labeling schemes for trees* CoRR abs/1608.00212.

[6] S. Alstrup, S. Dahlgaard, M. B. T. Knudsen, E. Porat *Sublinear distance labeling* in: 24th ESA, 2016, pp. 5:1–5:15.

[7] C. Gavoille, D. Peleg, S. Pérennes, R. Raz *Distance labeling in graphs* J. Algorithms 53 (1) (2004) 85–112.