Addressing, Distances and Routing in Triangular Systems with Applications in Cellular and Sensor Networks

Victor Chepoi,† Feodor Dragan,‡ and Yann Vaxès†

†Laboratoire d'Informatique Fondamentale de Marseille, Université de la Méditerranée,
Faculté des Sciences de Luminy, F-13288 Marseille Cedex 9, France
{chepoi,vaxes}@lidil.univ-mrs.fr
and
‡Department of Computer Science, Kent State University, Kent, Ohio 44242, USA
dragan@cs.kent.edu

Abstract. Triangular systems are the subgraphs of the regular triangular grid which are formed by a simple circuit of the grid and the region bounded by this circuit. We show that each such system with n vertices admits a labeling that assigns $O(\log^2 n)$ bit labels to vertices of the system such that the distance between any two vertices u and v can be determined in constant time by merely inspecting the labels of u and v, without using any other information about the system. Furthermore, there is a labeling, assigning labels of size $O(\log n)$ bits to vertices, which allows, given the label of a source vertex and the label of a destination, to compute in constant time the port number of the edge from the source that heads in the direction of the destination. In constructing these labelings, an isometric embedding of the system into the product of three trees is employed.

Keywords: Cellular networks, sensor networks, triangular systems, distance, routing, location management.

1 Introduction and motivation

Triangular systems are the subgraphs of the regular triangular grid which are formed by a simple circuit (with some vertices visited possibly more than once) of the grid and the region bounded by this circuit. In other words, the triangular systems are the connected planar graphs with inner faces of length 3 and inner vertices of degree 6. Particular instances of triangular systems are *hexagonal networks* considered in [9], which are the isometric subgraphs of the regular triangular grid (e.a., the portion of the regular triangular grid which is formed by a convex polygon and the region bounded by this polygon). Motivated by applications of hexagonal networks in cellular, wireless, sensor and interconnection networks, Nocetti et al. [9] presented a suitable addressing scheme for vertices which allowed to derive a simple formula for distance between vertices and design a very elegant routing algorithm.

Unfortunately, their method works only for triangular systems which are the distance preserving subgraphs of the triangular grid (i.e., for hexagonal networks). However, general triangular systems are more realistic models for cellular, wireless and sensor networks since they address a more general case when receivers or sensors are uniformly located inside some simply connected region (not necessarily convex and not necessarily forming an isometric subgraph of the triangular grid). As possible examples one can consider sensors uniformly distributed in a lake or in a valley surrounded by mountains. In what follows we will outline in some details the application of triangular systems in cellular communications.

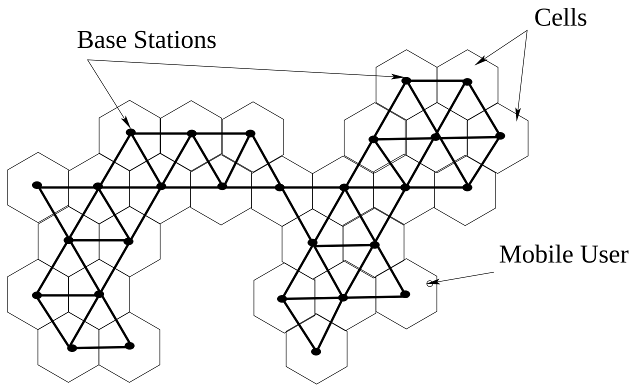

Figure 1: A cellular network modeled by a triangular system.

Cellular communications have experienced an explosive growth recently. *Cellular networks* are commonly designed as triangular systems, where vertices serve as *base stations* (BSs) to which mobile users must connect to make or receive phone calls. Mobile users are normally connected to the nearest BSs and, thus, BSs divide the area such that each BS serves all users that are located inside a *hexagon* (*a cell*) centered at BS (see Figure 1). Mobile users with cellular phones have to register frequently to facilitate their location when phoning them. They move from cell to cell, but do not always contact their new cell to update their position since too many messages may be required and the system may be blocked for regular calls. In [2], Bar-Noy et al. proposed three dynamic location update (or registration) schemes: *time-based, movement based, and distance-based.* It has been shown that the distance-based scheme is the most efficient among the three [2]. In the distance-based location update scheme, a mobile terminal updates its location when the distance in terms of cells it traveled since the last update exceeds a predefined threshold.

The *location management problem* is related to the efficient search for a mobile terminal upon a call arrival. The incoming call is directed toward the last reported position u of mobile user. The search process then continues by paging all cells which are located within cell distance d from u. Here the *cell distance* is the number of cells on a shortest route between the two cells. Cell distance directly corresponds to the number of retransmissions of paging request, which is easily controlled by setting a paging counter. Naor et al. [8] proposed to use *cell identification codes* (CIC) for tracking mobile users. Each cell periodically broadcasts a short message which identifies the cell and its orientation relative to other cells in the network. Mobile users, to efficiently update their location, use this information.

Thus, additional to efficient routing protocols, there is a need in efficient computation of the cell distance between any two cells. However, it has been claimed in [2] that it is hard to compute the distance between two cells, or it requires a lot of storage to maintain the distance information among all cells [1, 7]. Current cellular networks do not provide information that can be used to derive cell distances. In [9], Nocetti et al. proposed a very simple method to compute the cell distance between any two cells in the case when a cellular network is modeled by a hexagonal network. The distance computation of [9] is based on a new cell addressing scheme. The scheme avoids using real geographic coordinates of BSs and, instead, considers relative positions of base stations in a cellular network to arrive at simple representation with three small integers, one of them being 0. This addressing scheme provides also a short and elegant routing protocol.

In this paper, we propose an addressing scheme for triangular systems by employing their isometric embeddings into the Cartesian product of three trees (for benzenoids, which are the dual graphs of triangular systems, a similar result has been established in [3]). This embedding provides a simple representation of any triangular system with only three small integers per vertex, and allows to employ the compact labeling schemes for trees for distance queries and routing. (Note that the addressing scheme of [9] is nothing else than a result of an isometric embedding of a hexagonal network into the product of three paths.) We show that each such system with n vertices admits a labeling that assigns $O(\log^2 n)$ bit labels to vertices of the system such that the distance between any two vertices u and v can be determined in constant time by merely inspecting the labels of u and v, without using any other information about the system. Furthermore, there is also a labeling, assigning labels of size $O(\log n)$ bits to vertices, which allows, given the label of a source vertex and the label of a destination, to compute in constant time the port number of the edge from the source that heads in the direction of the destination.

2 Addressing via isometric embedding into the product of three trees

In a graph $G = (V, E)$ the *length* of a path from a vertex x to a vertex y is the number of edges in the path. The *distance* $d_G(x, y)$ between x to y is the length of a shortest path connecting x and y. Given two connected graphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$ and an integer k, we say that G admits an *isometric embedding* into H if there exists a *mapping*

$$\alpha : V(G) \to V(H)$$

such that

$$d_H(\alpha(x), \alpha(y)) = d_G(x, y)$$

for all vertices $x, y \in V(G)$. If there is a mapping

$$\alpha : V(G) \to V(H)$$

such that

$$d_H(\alpha(x), \alpha(y)) = k\ d_G(x, y)$$

for all vertices $x, y \in V(G)$, then we say that G admits a *scale k isometric embedding* into H.

The *Cartesian product* $H = H_1 \times \cdots \times H_m$ of connected graphs $H_1, \ldots, H_m$ is defined upon the Cartesian product of the vertex sets of the corresponding graphs (called *factors*), i.e.,

$$V(H) = \{u = (u_1, \ldots, u_m) : u_i \in V(H_i), i = 1, \ldots, m\}.$$

Two vertices $u = (u_1, \ldots, u_m)$ and $v = (v_1, \ldots, v_m)$ are adjacent in H if and only if the vectors u and v coincide except at one position i, in which we have two vertices u_i and v_i

adjacent in H_i. The *distance* between two vertices $x = (x_1, \dots, x_m)$ and $y = (y_1, \dots, y_m)$ of H is given by

$$d_H(x, y) = \sum_{i=1}^{m} d_{H_i}(x_i, y_i).$$

To formulate the embedding result we need some further terminology. Let G be a triangular system bounded by a *simple circuit* B. By E_1, E_2 and E_3 denote the edges of G of a given *direction*. Consider a graph $G_i = (V, E_i)$ $(i = 1, 2, 3)$ formed by the edges E_i. Evidently, the connected components of G_i are *paths* of G with end–vertices on B; we call them *i-paths* of G. One can easily show that every i-path P is a shortest path. Moreover, P is the unique shortest path in G that connects the end–vertices of P. Indeed, let vertices $x, y \in P$ be connected by a shortest path Q outside P and suppose, without loss of generality, that $Q \cap P = \{x, y\}$. Let $f = xx'$ be the first edge of Q, say $f \in E_j$. Clearly f is not collinear with the edges of P, i.e., $j \neq i$. For every edge $e \in P$ there is an edge $e' \in Q$ contained in the strip bounded by the j-paths passing via the end–vertices of e. Since $a' \neq b'$ for any two distinct edges $a, b \in P$ and the edge $f \in Q$ is not an image of an edge of P, we conclude that Q is longer than the subpath of P comprised between the vertices x and y, thus yielding a contradiction with the choice of Q.

A straight line segment $c = [p, q]$ is called an *i-cut segment* if p and q are the centers of two edges not belonging to E_i, c is parallel to the i-paths, and the graph obtained from G by removing all edges intersected by c has exactly two connected components. In this case we say that c *separates* any two vertices (or i-paths) from different connected components. Denote by C_i the collection of all i-cut segments. Note that every edge $e \in E_i$ is crossed by exactly two cut-segments (not belonging to C_i).

Define a graph T_i whose nodes are the connected components of G_i and two such components P' and P'' are adjacent in T_i if and only if there exists an edge e in G with one end in P' and the second in P'' (see Figure 2 for an example). Since G is bounded by a Jordan curve B, every T_i is a tree (the existence of a cycle in T_i would imply that G contains a non–triangular interior face). Note that there exists a bijection between the edges of T_i and the cut segments of C_i: if the i-paths P' and P'' are adjacent in T_i, then the edges of G with one end in P' and another one in P'' are crossed by the same i-cut segment. Using this observation and the fact that T_i is a tree, one concludes that, more generally, any two i-paths P' and P'' are separated by exactly $d_{T_i}(P', P'')$ cut segments.

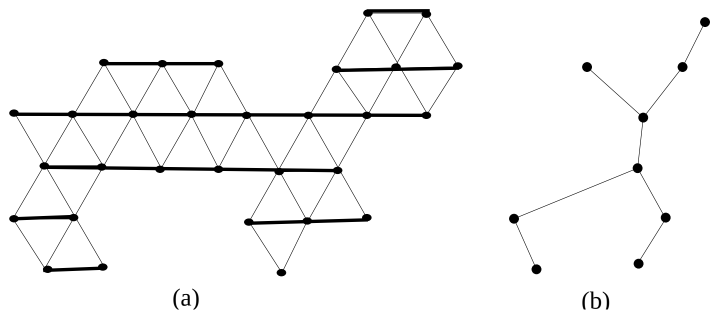

Figure 2: (a) A triangular system and (b) the tree–factor corresponding to the horizontal edges.

We obtain the following canonical embedding α of G into the Cartesian product $H = T_1 \times T_2 \times T_3$. For any vertex v of G put $\alpha(v) = (P, Q, R)$, where P, Q and R are the connected components of the graphs G_1, G_2 and G_3, respectively, sharing the vertex v. We claim that α provides a *scale 2 isometric embedding* of G into H, i.e., for all vertices $x = (\alpha_1(x), \alpha_2(x), \alpha_3(x))$ and $y = (\alpha_1(y), \alpha_2(y), \alpha_1(y))$ of G,

$$2\ d_G(x, y) = \sum_{i=1}^{3} d_{T_i}(\alpha_i(x), \alpha_i(y)) \tag{1}$$

holds. To prove this, pick two arbitrary vertices x and y of G and suppose that $\alpha(x) = (P', Q', R')$, $\alpha(y) = (P'', Q'', R'')$. From what has been shown above one concludes that the vertices x and y are separated by $d_{T_1}(P', P'')$ cut segments from C_1, $d_{T_2}(Q', Q'')$ cut segments from C_2, and $d_{T_3}(R', R'')$ cut segments from C_3. To complete the proof, it suffices to show that the vertices x and y are separated by exactly $2d_G(x, y)$ cut segments. Pick an arbitrary shortest path L between x and y. Any cut segment c separating x and y necessarily intersects at least one edge of L. If, say, $c \in C_i$ intersects two edges $u'v'$ and $u''v''$ of L, then we arrive at a contradiction. Indeed, in this case, if the vertices u' are u'' are taken from the same i-path P, then u' and u'' would be connected by more than one shortest path, in contradiction with what has been shown about P. Thus, every cut segment separating x and y intersects exactly one edge of L. Since every cut segment crossing an edge of L also separates the vertices x, y and since every edge of L is crossed by exactly two cut segments, we obtain (1).

Hence, α is a scale 2 isometric embedding of G into H. To define three integer addresses of the vertices of G one can do the following. For a given edge direction i ($i = 1, 2, 3$) first find the corresponding edge set E_i. Then define the graphs G_i and find their connected components. Having these connected components, it is easy to construct the trees T_i ($i = 1, 2, 3$) and index their nodes from 1 to k_i ($k_i \leq n$) in *depth-first-search* order. Now the i–th coordinate ($i = 1, 2, 3$) of a vertex v of G is the index of that connected component of G_i which contains v. Clearly, if G consists of n vertices, then the trees T_1, T_2, T_3 and the three integer addresses of the vertices of G can be computed in total $O(n)$ time.

Summarizing the discussion of this section, we conclude

Theorem 2.1 *The map α provides a scale 2 isometric embedding of a triangular system G with n vertices into the graph $H = T_1 \times T_2 \times T_3$. The factors T_1, T_2, T_3 as well as the corresponding three integer addresses of the vertices of G can be computed in total $O(n)$ number of operations.*

3 Distance decoder

A graph family $\mathcal{F}$ is said (see [11]) to have an $l(n)$ *distance labeling scheme* if there is a function L labeling the vertices of each n-vertex graph in $\mathcal{F}$ with distinct labels of up to $l(n)$ bits, and there exists an algorithm, called *distance decoder*, that given two labels $L(v), L(u)$ of two vertices v, u in a graph from $\mathcal{F}$, decides the distance between v and u in time polynomial in the length of the given labels. Note that the algorithm is not given any additional information, other that the two labels, regarding the graph from which the vertices were taken.

In this section, we show that triangular systems with n vertices enjoy a distance labeling scheme with labels of size $O(\log^2 n)$ bits and a constant time distance decoder.

Let G be a triangular system with n vertices and assume that the tree–factors T_1, T_2, T_3 and the three integer addresses of the vertices of G are given. The distance formula (1) reduces the problem of computing $d_G(x, y)$ to three similar problems on factors. A major advantage is that all three factors are trees. A distance labeling scheme for n-node trees that uses only $O(\log^2 n)$ bit labels but an $O(\log n)$ time distance decoder has been given in [10]. This result is complemented by a lower bound proven in [4], showing that $\Omega(\log^2 n)$ bit labels are necessary for the class of all n-node trees with inner vertices of degree at most 3.

Here we slightly revise the distance labeling scheme for trees of [10] and show that the modified distance decoder runs in constant time.

Let T be an arbitrary tree. It is well known that any tree T with n nodes has a node v (called a *median node*) the removal of which breaks T into disconnected subtrees $T^1, \ldots, T^k$, each with at most $n/2$ nodes. Using this fact, first we construct a decomposition tree $\widehat{T}$ for the tree T in the following recursive way. Find a median node v of T and let $T^1, \ldots, T^k$ be the connected components of $T - v$. For each T^j $(j = 1, \ldots, k)$ construct a decomposition tree $\widehat{T^j}$ recursively and build $\widehat{T}$ by taking v to be the root and connecting the root of each tree $\widehat{T^j}$ as a child of v. It is easy to see that a decomposition tree $\widehat{T}$ of a tree T with n nodes has depth at most $\log_2 n$ and can be constructed in $O(n \log n)$ time since a median node of a tree can be found in linear time [5]. Indeed, in each level of recursion we need to find median nodes of current subtrees and, since the tree sizes are reduced by a factor 1/2, the recursion depth is $O(\log n)$.

For the tree $\widehat{T}$ we need also a labeling scheme for depths of nearest common ancestors (*NCA-depth labeling scheme*). NCA-depth labeling scheme for a tree $\widehat{T}$ with the root v is a scheme that labels the nodes of $\widehat{T}$ with short labels in such a way that the distance from v to the nearest common ancestor of two nodes x and y of $\widehat{T}$ can be determined efficiently by merely inspecting the labels of x and y, without using any other information. In [11] such a scheme with $O(\log^2 n)$ bit labels but with $O(\log n)$ query time was presented for any tree with n nodes. One can use here the fact that $\widehat{T}$ has the $O(\log n)$ depth and get constant query time in this case. To do this one can simply translate the technique of Harel and Tarjan [6] to a labeling scheme. Note that whenever they access global information, it is associated with an ancestor in a tree. Since the depth of our tree is $O(\log n)$, one can copy this ancestor information down to each descendant and get the desired label of $O(\log^2 n)$ bits. Thus, tree $\widehat{T}$ can be preprocessed in $O(n \log n)$ time for depths of nearest common ancestors. This preprocessing step creates for $\widehat{T}$ an NCA-depth labeling scheme with $O(\log^2 n)$ bit labels and constant query time.

Now, for each node x of a tree T, let A_x be the label of x in the NCA-depth labeling scheme of $\widehat{T}$. Let also $v_0, v_1, \ldots, v_h$ be the nodes of the path of $\widehat{T}$ from the root v (which is v_0) to the node $x = v_h$. Clearly, $h \leq \log_2 n$. In the distance labeling scheme for T, the label $L_T(x)$ of x will be the concatenation of A_x and $h + 1$ distances $d_T(x, v_0), d_T(x, v_1), \ldots, d_T(x, v_h)$. Since the depth of $\widehat{T}$ is $O(\log n)$, $L_T(x)$ is of length $O(\log^2 n)$ bits for any node x. Clearly the computation of all labels $L_T(x), x \in V(T)$, takes $O(n \log n)$ total time. To decode the distance in T between x and y, one can use the following function. Note that, since the nearest common ancestor $nca_{\widehat{T}}(x, y)$ of nodes x and y lies on the path of T between x and y, we have $d_T(x, y) = d_T(x, nca_{\widehat{T}}(x, y)) + d_T(y, nca_{\widehat{T}}(x, y))$.

function distance_decoder_trees($L_T(x), L_T(y)$)

extract from $L_T(x)$ and $L_T(y)$ the entries A_x and A_y;
use A_x and A_y to find the depth l in $\widehat{T}$ of the nearest common ancestor of x and y;
extract from $L_T(x)$ and $L_T(y)$ the distances $d_T(x, v_l)$ and $d_T(y, v_l)$;
return $d_T(x, v_l) + d_T(y, v_l)$.

For a triangular system G with the tree–factors T_1, T_2, T_3, the label $L(x)$ of a vertex x will be the concatenation of $L_{T_1}(\alpha_1(x)), L_{T_2}(\alpha_2(x))$ and $L_{T_3}(\alpha_3(x))$. Then the distance between vertices x and y of G can be computed in constant time using the following function.

function distance_decoder_triang_syst($L(x), L(y)$)

return (distance_decoder_trees($L_{T_1}(\alpha_1(x)), L_{T_1}(\alpha_1(y))$)+
distance_decoder_trees($L_{T_2}(\alpha_2(x)), L_{T_2}(\alpha_2(y))$)+
distance_decoder_trees($L_{T_3}(\alpha_3(x)), L_{T_3}(\alpha_3(y))$))/2.

Thus, we have proved the following result.

Theorem 3.1 *The family of triangular systems with at most n vertices admits a distance labeling scheme with labels of size $O(\log^2 n)$ bits and a constant time distance decoder. Moreover, the scheme is constructable in time $O(n \log n)$.*

4 Routing

Following [12], one can give the following formal definition. A family $\Re$ of graphs is said to have an $l(n)$ routing labeling scheme if there is a function L labeling the vertices of each n-vertex graph in $\Re$ with distinct labels of up to $l(n)$ bits, and there exists an efficient algorithm, called the *routing decision*, that given the label of a source vertex v and the label of the destination vertex (the header of the packet), decides in time polynomial in the length of the given labels and using only those two labels, whether this packet has already reached its destination, and if not, to which neighbor of v to forward the packet.

In this section, we establish that triangular systems with n vertices enjoy a routing labeling scheme with labels of size $O(\log n)$ bits and a constant time routing decision. For this, we build up the tree-factors T_1, T_2, T_3 provided by Theorem 2.1, and to each factor we employ the routing scheme of Thorup and Zwick [13].

To make this note self-contained, first we briefly describe the routing labeling scheme of [13]. Let T be an arbitrary n-node tree rooted at node r. The *weight* s_v of a node v is the number of its descendants in the tree (a node is considered to be a descendant of itself). A child of a node v is said to be *heavy* if its weight is highest among all children. We let each non-leaf node have a single heavy child (ties can be broken arbitrarily). All other children of v are called *light*. For convenience, we define r, the root of the tree, to be heavy. The *light level* ℓ_v of a node v is defined as the number of light nodes on the path from r to v, including v if it is light. If v is a non-leaf node, we let v' be its heavy child, and $v_0, v_1, \ldots, v_{d-1}$ be its light children in non-increasing order of weight, i.e., $s_{v'} \geq s_{v_0} \geq \cdots \geq s_{v_{d-1}}$. It is easy to see that $s_{v_i} \leq s_v/(i+2)$, for $0 \leq i < d$. Therefore, each time an edge from a node to one of its light children is descended, the number of descendants in the corresponding subtree decreases by a factor of at least 2. Thus, the light level ℓ_v of every node v is at most $O(\log_2 n)$.

We assign the edge vv_i, for $0 \le i < d$, port number i, and assign the edge vv' port number d. We enumerate the nodes of the tree in *depth first order*, where all the light children of a node are visited before its heavy child. We identify a node v with the number assigned to it, and let f_v be the *largest descendant* of v. We let P_v be an array containing in its first element $P_v[0]$ the port number corresponding to the edge from v to its parent, and then in its second element $P_v[1]$ the port number corresponding to the edge from v to its heavy child. Let also $L_v = (q_1, q_2, \ldots, q_{\ell-1})$ be the port numbers of the edges leading to the light nodes on the path from r to v. Instead of storing each port number q in a separate word, one can use only $\lfloor \log_2 q \rfloor + 1$ bits, or a single bit if $q = 0$, and concatenate all these bit strings. For example, the sequence (2, 0, 5, 3) would yield the string 10'0'101'11 (the quotes are, of course, not part of this sequence and were added for illustration purposes only). Instead of the quotes, one can use a *mask*. Each '1' in this mask would mark the end of a string representing a number. Thus, the mask corresponding to our string above would be 01'1'001'01 (again, without the quotes). With each node v we therefore associate a bit string L_v and a masking bit string M_v. The length of each one of them is $O(\log_2 n)$ as shown in [13].

Now, the routing label $L_T(x)$ stored at node x consists of $(x, x', f_x, P_x, L_x, M_x, k_x)$. We store with x also the total length k_x of first $\ell_x - 1$ numbers in L_x. The total size of $L_T(x)$ is $O(\log_2 n)$ bits. The header of a packet headed towards destination y will consist only of (y, L_y, M_y), i.e., $H_T(y) = (y, L_y, M_y)$. The routing algorithm should now be obvious. Suppose that a packet with the header (y, L_y, M_y) arrives at x. If $x = y$, we are done. Otherwise, we check whether $y \in [x, f_x]$ (where $[x, f_x]$ are the integers from x to f_x). If not, then y is not a descendant of x and the packet is forwarded to the parent of x using port $P_x[0]$. Next, we check whether $y \in [y', f_x]$. If so, then y is a descendant of a heavy child of x, and the packet is forwarded to the heavy child of x using port $P_x[1]$. Otherwise, y is a descendant of a light child, in which case we need to extract the ℓ_x-th number coded in L_y. We only have to do that, however, when y is a descendant of x, in which case we know that the first $\ell_x - 1$ numbers coded in L_y are exactly the same as those in L_x. Thus, we can use the number k_x stored at x to extract the required port number from L_y. If the indices in the bit string L_y start from 0, we need to extract the bit substring of L_y starting with $L_y(k_x)$ and ending with $L_y(k_x + j)$, where j is the smallest non-negative integer such that $M_y(k_x + j) = 1$. A formal description of this constant time routing algorithm is given below.

function routing_decision_trees($L_T(x), H_T(y)$)

if $x = y$ then return "packet reached its destination";
if $y \notin [x, f_x]$ then return $P_x[0]$;
if $y \in [y', f_x]$ then return $P_x[1]$;
else
  extract from L_y the bit substring starting with $L_y(k_x)$ and ending with $L_y(k_x + j)$,
    where j is the smallest non-negative integer such that $M_y(k_x + j) = 1$, and
  return the obtained port number.

Returning to a triangular system $G = (V, E)$, recall that $(\alpha_1(x), \alpha_2(x), \alpha_3(x))$ is the address of a vertex $x \in V$ in the graph $T_1 \times T_2 \times T_3$ defined in Theorem 2.1. Denote by $L_{T_i}(\alpha_i(x))$ the Thorup-Zwick label of the node $\alpha_i(x)$ in the tree T_i, $i = 1, 2, 3$. Let $\Delta_i(x)$ be the set of all i-paths different from $\alpha_i(x)$ which pass via a neighbor of the vertex x in G. One can easily see that $\delta_i(x) := |\Delta_i(x)| \le 4$. Set $\Delta_i(x) := \{\alpha_i^1, ..., \alpha_i^{\delta_i(x)}\}$. For each $x \in V$ and each index $i = 1, 2, 3$, we keep an array $O_i(x)$ having $\delta_i(x)$ entries as well as $\delta_i(x)$ arrays

$Q_i^j(x)$, $j = 1, \ldots, \delta_i(x)$, with one or two entries each. The array $O_i(x)$ contains the port numbers corresponding to the edges of T_i from $\alpha_i(x)$ to the nodes of $\Delta_i(x)$. Each $Q_i^j(x)$ contains the port numbers corresponding to the edges of G from x to its neighbors in the i-path α_i^j. The label $L_G(x)$ of a vertex x of G is the concatenation of the Thorup-Zwick labels $L_{T_1}(\alpha_1(x)), L_{T_2}(\alpha_2(x)), L_{T_3}(\alpha_3(x))$, of the arrays $O_i(x)$, $i = 1, 2, 3$, and of the arrays $Q_i^j(x)$, $i = 1, 2, 3$ and $j = 1, \ldots, \delta_i(x)$.

The header $H_G(y)$ of a packet with destination y will consists of the corresponding Thorup-Zwick headers of the paths $\alpha_1(y), \alpha_2(y)$, and $\alpha_3(y)$. Suppose that such a packet arrives at a vertex x. For routing decision at x, we use an auxiliary array $\mathbf{A}$ indexed by the port numbers of the edges incident to x in G and whose entries are initially all set to 0. For each $i \in \{1, 2, 3\}$, employing the Thorup-Zwick algorithm, we compute the port number of the edge of T_i incident to $\alpha_i(x)$ and lying on the path between $\alpha_i(x)$ and $\alpha_i(y)$. If this port corresponds to an entry of the array $O_i(x)$ (this can be checked in $O(1)$ time), say the jth entry, then we increment by 1 all entries of the array $\mathbf{A}$ which correspond to port numbers occurring in the array $Q_i^j(x)$. This operation is repeated for each $i \in \{1, 2, 3\}$ until one of the entries of $\mathbf{A}$ becomes equal to 2. Then the packet is forwarded to the neighbor of x via the corresponding port.

function routing_decision_triang_syst$(L(x), H(y))$
if $(\alpha_1(x), \alpha_2(x), \alpha_3(x)) = (\alpha_1(y), \alpha_2(y), \alpha_3(y))$ then return "packet reached its destination";
set $\mathbf{A} \leftarrow \mathbf{0}$;
for each $i \in \{1, 2, 3\}$ do
  $p \leftarrow$ routing_decision_trees$(L_{T_i}(\alpha_i(x)), H_{T_i}(\alpha_i(y)))$;
  for each $j \in \{1, ..., |O_i(x)|\}$ do
    if $p = O_i(x)[j]$ then
      for each entry port_G of the array $Q_i^j(x)$ do
        $\mathbf{A}[\mathsf{port}_G] \leftarrow \mathbf{A}[\mathsf{port}_G] + 1$;
        if $\mathbf{A}[\mathsf{port}_G] = 2$ then
          return port_G.

The correctness of this simple algorithm is a consequence of the following fact: a neighbor z of x is closer to the destination y than the vertex x if and only if in exactly two of the trees $T_i, i = 1, 2, 3$, the vertex $\alpha_i(z)$ belongs to the path between $\alpha_i(x)$ and $\alpha_i(y)$. Indeed, by Theorem 2.1, two adjacent vertices x, z of G have one identical coordinate (say the third one) and two coordinates corresponding to adjacent nodes of the first and the second tree factors. Hence $d_G(x, y) = d_G(z, y) + 1$ if and only if $\alpha_1(z)$ is between $\alpha_1(x)$ and $\alpha_1(y)$ in T_1 and $\alpha_2(z)$ is between $\alpha_2(x)$ and $\alpha_2(y)$ in T_2. Only such vertices z will have entries in $\mathbf{A}$ equal to 2. Thus, we obtain the following result.

Theorem 4.1 *The family of triangular systems with at most n vertices admits a routing labeling scheme with labels of size $O(\log n)$ bits and a constant time routing decision. Moreover, the scheme is constructable in linear $O(n)$ time.*

References

[1] I.F. Akyildiz, J.S.M. Ho, and Y.B. Lin, Movement-based location update and selective paging for PCS networks, *IEEE/ACM Trans. Networking*, vol. 4, no. 4, pp. 629-638, Aug. 1996.

[2] A. Bar-Noy, I. Kessler, and M. Sidi, Mobile users: to update or not to update, *Wireless Networks,* vol. 1, no. 2, pp. 175-185, 1994.

[3] V. Chepoi, On distances in benzenoid systems, *J. Chemical Information and Computer Sciences* vol. 36, pp. 1169–1172, 1996.

[4] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz, Distance labeling in graphs, in *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001)*, January 7-9, 2001, Washington, DC, USA, ACM-SIAM, pp. 210–219, 2001.

[5] A.J. Goldman, Optimal center location in simple networks, *Transportation Science,* vol. 5, pp. 212–221, 1971.

[6] D. Harel, R. Tarjan, Fast algorithms for finding nearest common ancestor, *SIAM J. Comput.,* vol. 13, pp. 338–355, 1984.

[7] J. Li, H. Kameda, and K. Li, Optimal dynamic location update for PCS networks, *IEEE/ACM Trans. Networking,* vol. 8, no. 3, pp. 319-327, 2000.

[8] Z. Naor, H. Levy, and U. Zwick, Cell identification codes for tracking mobile users, *Proc. INFOCOM 1999,* pp. 28-35, 1999, also available in *Wireless Networks,* vol. 8, no. 1, pp. 73-84, 2002.

[9] F.G. Nocetti, I. Stojmenovic, and J. Zhang, Addressing and routing in hexagonal network with application for tracking mobile users and connection rerouting in cellular networks, *IEEE Trans. on Parallel and Distributed Systems,* vol. 13, no. 9, pp. 963-971, 2002.

[10] D. Peleg, Proximity-preserving labeling schemes and their applications, in *Proceedings of the 25th International Workshop "Graph-Theoretic Concepts in Computer Science" (WG '99)*, Ascona, Switzerland, June 17-19, 1999, *Lecture Notes in Computer Science 1665*, Springer, pp. 30-41, 1999.

[11] D. Peleg, Informative labeling schemes for graphs, in *Proceedings of the 25th International Symposium "Mathematical Foundations of Computer Science" (MFCS 2000)*, Bratislava, Slovakia, August 28 - September 1, 2000, *Lecture Notes in Computer Science 1893*, Springer, pp. 579-588, 2000.

[12] D. Peleg, *Distributed Computing – A Locality-Sensitive Approach*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[13] M. Thorup, U. Zwick, Compact routing schemes, in *Proc. 13th Ann. ACM Symp. on Par. Alg. and Arch. (SPAA 2001)*, ACM, pp. 1–10, 2001.