Distributed Algorithm of Self-transformation of the Distributed Network Topology in Order to Minimize the Wiener Index

Igor Burdonov[0000-0001-9539-7853]

Ivannikov Institute for System Programming of the RAS, Alexander Solzhenitsyn st., 25, 109004, Moscow, Russia
igor@ispras.ru

Abstract. We consider a distributed network, the topology of which is described by a undirected tree without multiple edges and loops. The network itself can change its topology using special "commands" supplied by its nodes. The paper proposes an extremely local atomic transformation: the addition of an edge connecting the different ends of two adjacent edges, and the simultaneous removal of one of these edges. This transformation is performed by a "command" from the common vertex of two adjacent edges. It is shown that any other tree can be obtained from any tree using only atomic transformations. If the degrees of the tree vertices are limited by the number d ($d \geq 3$), then the transformation does not violate this restriction. As an example of the goal of such a transformation, the tasks of maximizing and minimizing the Wiener index of a tree with a limited degree of vertices without changing the set of its vertices are considered. The Wiener index is the sum of the pairwise distances between the vertices of the graph. The maximum Wiener index has a linear tree (a tree with two leaf vertices). For a root tree with a minimum Wiener index, its type and a method of calculating the number of vertices in the branches of the neighbors of the root is proposed. Two distributed algorithms are proposed: the transformation of a tree into a linear tree and the transformation of a linear tree into a tree with a minimum Wiener index. It is proved that both algorithms have complexity not higher than $2n - 2$, where n is the number of tree vertice.

Keywords: Distributed Network, Transformation of Graphs, Wiener Index.

1 Introduction

The Wiener index [1] is a topological index of molecular graphs used in many applications, especially in mathematical and computer chemistry and chemoinformatics.

We consider a distributed network whose topology is a dynamically changing tree. Dynamic graphs [2] model self-organizing networks [3–5], including social networks, neural networks [6] and swarm intelligence [7]. A feature of these networks is their homogeneity, without dividing the nodes into switches, hosts, and controllers. The focus is on routing, bandwidth, noise immunity, security, load balancing and network resources, etc. A change in the network topology is understood as an external factor that must be taken into account, but which the network itself does not control or only partially controls [8, 9].

On the other hand, there are many works in the literature devoted to precisely targeted transformation of the graph, in particular, trees in order to optimize according to certain criteria. The article proposes an atomic transformation that is extremely local, affecting a minimum of vertices and edges that are as close as possible to each other. Other tree transformations were considered earlier [10, 11, 12], but they are not local enough and are reduced to chains of atomic transformations.

The algorithms proposed in the article are distributed and parallel. A tree transforms itself according to "commands" from computing units correlated with vertices. For this, we need the locality of transformation.

The structure of the article is as follows. Section 2 defines a distributed network model and an atomic transformation. Section 3 contains basic concepts and statements related to the Wiener index. In Section 4, we propose an algorithm for transforming a tree into a linear tree, and in Section 5, an algorithm is proposed how to move from a linear tree to a tree with a minimum Wiener index under a given restriction on the degrees of vertices. The complexity estimates are given.

2 Model

Let G be an undirected tree that underlies the distributed network. Let $d \geq 3$ be the upper bound on the degrees of the vertices. The edges incident to the vertex are assigned various nonzero numbers; in this article we will call such a tree an *ordered tree*. The edge ab has two numbers: e_{ab} at vertex a and e_{ba} at vertex b.

Vertices are identified with computational units that send messages to each other along the edges of the graph. Vertex memory is a set of variables. First, at each vertex a, the variable $E(a)$ is initialized by the set of numbers of edges incident to the vertex a. Further, during the transformation of the graph, the vertex a itself adjusts the variable $E(a)$.

The message is set by type and parameters: $Type(p_1, ..., p_k)$. Vertex a, sending a message along edge ab, indicates its number e_{ab}. Vertex b receives the message along with the edge number e_{ba}.

The tree is transformed by "commands" from its vertices. The atomic transformation $a \to c \to b$ is the replacement of the edge ac by the edge ab in the presence of the edge cb (fig. 1). It is executed by the command ***CHANGE***(e_{ca}, e_{cb}, P), which is given by vertex c, where P are additional parameters. The edge ab receives at the vertex a the same number that the deleted edge ac had, i.e. e_{ac}, and at the vertex b it gets any "free" e_{bc} number. In order for vertex b to "recognize" this number, the message $Change(P)$ is automatically sent along the edge from a to b. Other messages transmitted along the variable edge are not lost, but a message sent to vertex c will be received by vertex b. Vertex c itself removes the number e_{ca} from $E(c)$, and vertex b itself adds the number e_{ba} to $E(b)$.

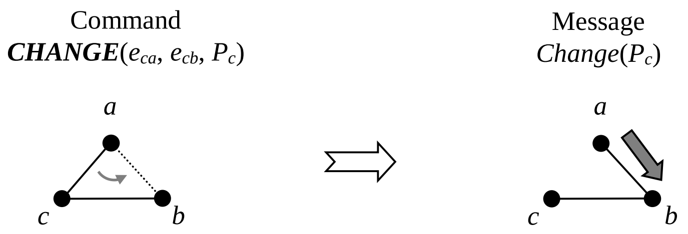

Fig. 1. The atomic transformation $a \to c \to b$ is the replacement of the edge *ac* by the edge *ab* in the presence of the edge *cb*.

To estimate the running time of the algorithm, we will neglect the computation time at the top and assume that the time for sending messages about changes, including forwarding messages to Change, does not exceed one clock cycle.

The following statements are true for atomic transformation[1]:

Proposition 1. An atomic transformation does not change the vertices of the tree and the tree remains a tree.

Proposition 2. Any tree can be transformed into a linear tree by a chain of atomic transformations preserving the set of vertices and the upper bound *d* on the degrees of the vertices.

The atomic transformation is reversible: after the transformation $a \to c \to b$, we can make the inverse transformation $a \to b \to c$. If the transformation $a \to c \to b$ preserves the upper bound *d* on the degrees of the vertices, then the transformation $a \to b \to c$ also preserves the upper bound *d* on the degrees of the vertices. If the chain of transformations converts tree *A* into tree *B*, then the reverse chain of inverse transformations converts tree B into tree *A*. Therefore, the following statement is true.

Proposition 3. Any tree can be obtained from a linear tree with the same set of vertices by a chain of atomic transformations preserving the upper bound *d* on the degrees of the vertices.

As a corollary of Propositions 2 and 3, we have the following proposition:

Proposition 4. Any tree can be transformed into any other tree with the same set of vertices using a chain of atomic transformations that preserve the upper bound *d* on the degrees of the vertices.

3 Wiener Index

The Wiener index is the sum of all pairwise distances between vertices. For a given number of vertices, the maximum Wiener index has a linear tree (a tree with two leaves) (A000292 in [14]). The type of a tree with maximum degree *d* ($d \geq 3$) and the minimum Wiener index was defined in [15]. This is a type of a balanced tree (leaf heights differ by at most 1) with a strict requirement for the degree of the vertex, which distinguishes it from B-trees in which all leaves are at the same height, and the degrees of the vertices can be different, and from AVL trees which are binary.

[1] For proofs of these and other statements see [13].

A *rooted tree* is a tree in which one vertex has been distinguished as the *root*. The *height of the vertex* is the distance from it to the root. The *tree height* is the maximum height of the vertex. The *branch of a vertex v* is a subgraph $G(v)$ generated by the set of vertices connected to the root by a path passing through v. For the edge ab, vertex a is the *father* of vertex b, and vertex b is the *son* of vertex a if the path from the root to b passes through a. Each vertex, except the root, has exactly one *father*.

In an ordered root tree, vertices of the same height are linearly ordered: the vertex v is to the *left* of the vertex w (w is to the *right* of v) if, after the maximum common prefix of the paths leading from the root to v and w, the number of the next edge on the path to v is less than the number of the next edge on the path to w.

A root tree of height h with n vertices is *almost good* if the root degree is $min\{d - 1, n - 1\}$, for $h \geq 3$ all vertices at a height of 1 .. h - 2 have degree d, and the tree can be ordered so that for $h \geq 2$ at a height h - 1, the rightmost inner vertex u has degree at most d, the vertices to the left of u have degree d, and the vertices to the right of u are leaves. A *good tree* differs only in the degree of root, it is equal to $min\{d, n - 1\}$. Examples are in Fig. 2.

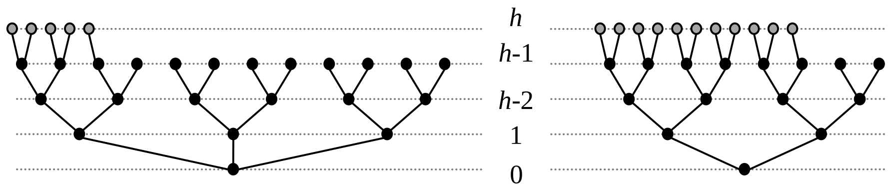

Fig. 2. Good tree (left) and almost good tree (right).

<u>Proposition 5</u> (Theorem 2.2. in [15]). A tree with the vertex degree at most d ($d \geq 3$) has a minimal Wiener index if and only if it is a good tree.

<u>Proposition 6</u>. For every n, there exists a unique good tree up to isomorphism preserving the root with the number n of vertices and an almost good tree up to isomorphism preserving the root with the number of n vertices.

<u>Proposition 7</u>. In a (almost) good tree G, the branch $G(v)$ of the vertex v adjacent to the root is an almost good tree.

Let an almost good tree have height h, the degree of the root be 0 or d - 1, and the degrees of all vertices at height h - 1 be d. The number of vertices of this tree is denoted by $N(d, h) = 1 + (d - 1) + (d - 1)^2 + ... + (d - 1)^h = ((d - 1)^{h+1} - 1) / (d - 2)$. Let a good tree have height h, the root degree be 0 or d, and the degrees of all vertices at height h - 1 be d. The number of vertices of this tree is denoted by $M(d, h)$: $M(d, 0) = 1$ and $M(d, h) = 1 + d + d(d - 1) + ... + d(d - 1)^{h-1} = 1 + dN(d, h - 1)$ for $h \geq 1$. Examples are in Fig. 2 when removing “gray” peaks at a height h.

Let a (almost) good tree with n vertices be given. The branch of the root neighbor is almost a good tree. We order the neighbors of the root by not increasing the number of vertices in their branches and denote these numbers:

for an almost good tree: $N(d, n, 1) \geq ... \geq N(d, n, min\{d - 1, n - 1\})$,

for a good tree: $M(d, n, 1) \geq ... \geq M(d, n, min\{d, n - 1\})$.

Proposition 8. Let $L(d, i) = N(d, i)$ if G is an almost good tree, and $L(d, i) = M(d, i)$ if G is a good tree. Let the number of vertices $n = L(d, h - 1) + m(d - 1) + s < L(d, h)$, where $0 \leq s < d - 1$, and $m = p(d - 1)^{h-2} + q$, where $0 \leq q < (d - 1)^{h-2}$. Then for p left-most neighbors of the root, their branches have $N(d, h - 1)$ vertices, for the next right neighbor of the root, its branch has $N(d, h - 2) + q(d - 1) + s$ vertices, and for the remaining neighbors of the root, each branch has $N(d, h - 2)$ vertices.

4 Algorithm $\mathcal{A}$: Transformation to a Linear Tree

4.1 Informal Description

If for a vertex v the branch of its son w is a linear tree, then the path from v through w to the leaf will be called a *line* from v. A *starlike tree* is a root tree consisting of lines leading from a root.

Let G be a tree with n vertices and with root r. For convenience, we assume that the root has a fictitious edge with number 0, leading to a fictitious father. The algorithm starts when the message *Start*() is received by the root from his (fictitious) father and ends with sending the message *Line*(n) from the root along this edge. The algorithm is recursive, at the recursion level h, the algorithm works on each branch of $G(v)$, where v has height h, and consists of three stages.

Stage 1 (Fig. 3). The vertex v receives a message *Start* from his father,stores the number of the edge leading to his father, and sends *Start* to all his sons.

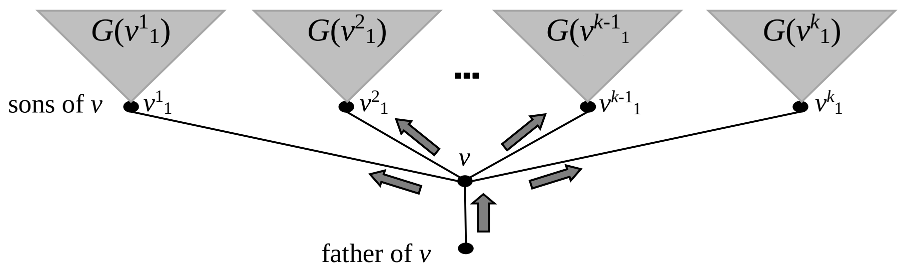

Fig. 3. Stage 1: Messages *Start* and tree branches.

Stage 2 (Fig. 4). Vertex v expects to receive messages *Line* from its sons, counting the number of vertices of branch $G(v)$ as 1 + the sum of the parameters of received messages *Line*.

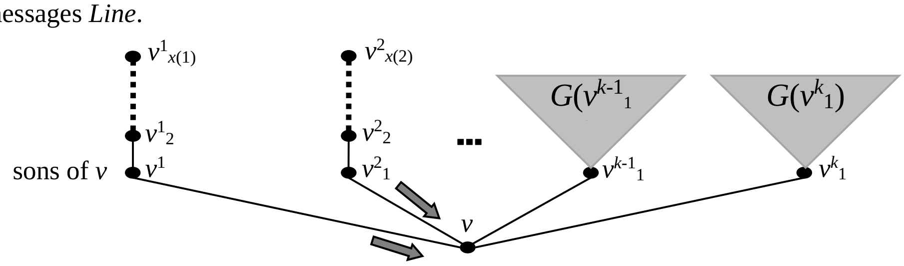

Fig. 4. Stage 2: Messages *Line* and lines.

At the beginning of Stage 3 (Fig. 5), the branch $G(v)$ is a starlike tree with k lines. The length of the i^{th} line is $x(i)$. The vertex v starts k - 1 parallel chains of atomic transformations so that for i = 1 .. k - 1 at the first edge of the $(i + 1)^{\text{th}}$ line, its end v^{i+1}_1 is fixed, and the other end moves along the i^{th} line from v to the leaf $v^i_{x(i)}$. To do this, the vertex v performs k - 1 transformations $v^{i+1}_1 \rightarrow v \rightarrow v^i_1$ simultaneously in the sequence $i = k - 1, ..., 1$. Each of these transformations is the first in the chain of transformations along the i^{th} line: $v^{i+1}_1 \rightarrow v \rightarrow v_i 1$, $v^{i+1}_1 \rightarrow v^i_1 \rightarrow v^i_2$, ..., $v^{i+1}_1 \rightarrow v^i_{x(i)-1} \rightarrow v^i_{x(i)}$. These transformation chains are performed in parallel along k - 1 lines. When the chain of transformations along the i^{th} line ends in its leaf, the i^{th} and $(i + 1)^{\text{th}}$ lines are concatenated. When this happens for all i = 1 .. k - 1, the branch $G(v)$ becomes a linear tree. Vertex v is notified of this by a message *Finish*(). It is sent after the k^{th} and $(k - 1)^{\text{th}}$ lines are concatenated, and then passes along the lines k - 1, ..., 1, and the i^{th} line goes from the end $v^i_{x(i)}$ to the beginning v^i_1. If *Finish* transfers to the $(i - 1)^{\text{th}}$ line before concatenating it with the i^{th} line, the transfer is suspended until the concatenation is completed. At the end, *Finish* is sent along the edge (v^1_1, v). Vertex v sends the message *Line*, to its father completing the work on branch $G(v)$.

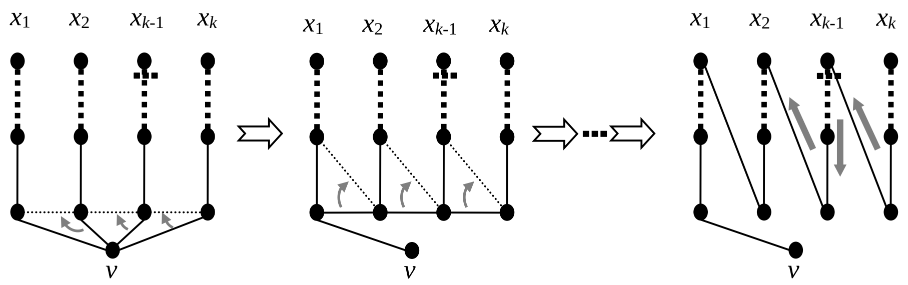

Fig. 5. Stage 3: Messages *Finish* and transformations.

4.2 Formal Description

Algorithm $\mathcal{A}(G, root)$.

Variables at vertex v:

- $E(v)$ is the set of edge numbers incident to the vertex v;
- $fat(v)$ is the number at the vertex v of the edge leading from the vertex v to its father;
- $line(v)$ is the flag (***Bool***) that the vertex v has sent the message *Line*;
- $nson(v)$ is the number of messages *Line* expected by vertex v from its sons;
- $sson(v)$ is the sum of the length of the lines from the vertex v in the received messages *Line*;
- $f(v)$ is the flag (***Bool***) of receiving *Finish* before *Change*.

Notation: $element(S)$ is the function to select an arbitrary element from a nonempty set S.

```
while it is not the end of the algorithm {
wait () { /* receive message */
    if Start() on the edge i is received { /* stage 1 */
```

```
    fat(v) := i; line(v) := false;  /* initialization */
    nson(v) := | E(v) \ { fat(v) } |; sson(v) := 1; f(v) := 0;
    send Start() on each edge j ∈ E(v) \ { fat(v) };
    if nson(v) = 0 { /* v is leaf or isolated vertex */
        send Line(sson(v)) on the edge fat(v); line(v) := true;
        if fat(v) = 0 { end of algorithm; } /* v is isolated vertex */
} }
if Line(l) on the edge i is received { /* stage 2 */
    nson(v) := nson(v) - 1; sson(v) := sson(v) + l; /* new line from v */
    if E(v) \ { fat(v) } = { i } { /* the vertex v has one son */
        send Line(sson(v)) on the edge fat(v); line(v) := true;
        if fat(v) = 0 { end of algorithm; }
    }
    else { /* vertex degree is greater than 2 or it is the root of degree 2 */
        if nson(v) = 0 { /* beginning of Step 3: G(v) is starlike tree */
            j := element( E(v) \ { fat(v), i } ); /* select edge */
            /* transformation a→v→b, где e(v, a) = i, e(v, b) = j */
            command CHANGE(i, j, true);
            E(v) = E(v) \ { i }; /* edge removed */
            while | E(v) \ { fat(v) } | > 1 { /* transformation cycle  */
                m := j;
                j := element( E(v) \ { fat(v), m } ); /* select edge */
                /* transformation a→v→b, where e(v, a) = m, e(v, b) = j */
                command CHANGE(m, j, false);
                E(v) = E(v) \ { m }; /* edge removed */
 }  }  }  }
if Change(f) on the edge i is received { /* stage 3: transformation a→c→v */
    f(v) := f(v) ∨ f; /* f(v) = true if there is a line v, a, ... */
    if | E(v) | = 2 { /* the transformation chain along the line is not finished */
        j := element(E(v) \ { fat(v) }; /* select the next edge */
        /* transformation a→v→b, where e(v, a) = i, e(v, b) = j */
        command CHANGE(i, j, f(v)); f(v) := false;
    }
    else { /* | E(v) | = 1, the transformation chain along the line is finished */
        E(v) = E(v) ∪ { i }; /* edge added */
        if f(v) { /* two lines concatenated */
            send Finish() on the edge fat(v); f(v) := false;
}    }  }
if Finish() on the edge i is received { /* Stage 3 */
    if i ∉ E(v) { f(v) := true; } /* vertex v received Finish before Change */
    else {
        if  line(v) { /* vertex v sent Line */
            send Finish() on the edge fat(v);
        }
        else { /* vertex v did not send Line */
```

```
                send Line(sson(v)) on the edge fat(v); line(v) := true;
                if fat(v) = 0 { end of algorithm; }
}}   }  }  }
```

Proposition 9. Algorithm $\mathcal{A}$ transforms a tree with n vertices into a linear tree without violating the restriction d on the degree of vertices with the time complexity $t(n) \leq 2n - 2$. The message *Line*(n) is sent from the root to his fictitious father.

The bound $2n - 2$ is reachable for a linear tree when the root is one of the leaves. The bound cannot be improved: in order to find out that the tree is linear, the message must go from the root to another leaf and vice versa, i.e. there is a walk through a path of length $2n - 2$.

5 Algorithm $\mathcal{B}$: Transformation of a Linear Tree into a Good Tree

5.1 Informal Description

Let G be a linear tree with n vertices and a root r being a leaf. The algorithm starts when the root receives the message *Begin*(d, n) on the fictitious edge number 0, and ends with the message *End*() sending from the root along this edge.

The algorithm is performed recursively, the recursion level is equal to the height of the vertex in the target good tree. At the recursion level h, a part of a good tree is constructed at heights from 0 to h. Initially, $h = 0$, and the constructed part consists of one root. At level h, the algorithm runs on each branch of $G(v)$, where the vertex v has height h in G and consists of two stages.

A starlike tree in which the degree of the root and the number of vertices of the branches of the neighbors of the root are the same as those of an (almost) good tree with the same number of vertices, we will call a (*almost*) *good starlike tree*.

Stage 1. The branch $G(v)$ is a line from v. We construct a starlike tree with a root in v: a good starlike tree if $v = r$, or an almost good starlike tree if $v \neq r$. The number of vertices of the branch $G(v)$ is the parameter of the message *Begin*, from which Stage 1 starts on the branch $G(v)$.

Stage 2. The branch $G(v)$ is a good ($v = r$) or almost good ($v \neq r$) starlike tree. The vertex v sends each son w the message *Begin*(d, l), where l is the number of vertices on the branch $G(w)$, initiating the algorithm at the next level of recursion. This can be done as soon as a line of the desired length from w is constructed. Vertex v is waiting for messages *End*() from all its sons and then sends *End*() to its father. If $v = r$, the algorithm ends.

How to build a starlike tree in Stage 1? We use the concept of the *current vertex* (first it is the vertex v) and two operations: *MOVING* and *TRANSFORMATION*. The parameter t is the number of transformations that remain to be done to build a line in a starlike tree.

MOVING $c \dashrightarrow b$: c is the current vertex, there is an edge $\{c, b\}$. The vertex c sends to the vertex b the message *Move*(t). Upon receiving the message, vertex b becomes current vertex.

TRANSFORMATION $a \to c \to b$: c is the current vertex; there are edges $\{a, c\}$ and $\{c, b\}$. The value of t decreases by 1: $t := t - 1$. Vertex c gives the command ***CHANGE***(e_{ca}, e_{cb}, t). After receiving the message *Change*(t), vertex b becomes current.

The construction is shown in Fig. 6: the gray arrow indicates the current vertex, the white circle indicates the vertex v. Let $l > 2$ be the number of vertices of the branch $G(v)$. First there is a line $v = v_1, v_2, ..., v_l$ for the current vertex v. Use the following notations:

$x = min\{d, l - 1\}$ is the degree of the vertex v in a good starlike tree;

$SM(d, l, 0) = 1$;

$SM(d, l, j) = 1 + M(d, l, 1) + M(d, l, 2) + ... + M(d, l, j)$, for $j = 1 .. x$, is the number of vertices in a good star tree on the first j lines plus one (the root);

$v^j_i = v_{SM(d, l, j - 1) + i}$, for $j = 1 .. x - 1$ and $i = 1 .. M(d, l, j)$, is the i^{th} vertex of the j^{th} line;

$v^x_i = v_{SM(d, l, x) - i + 1}$, for $i = 1 .. M(d, l, x)$, is i^{th} vertex of the x^{th} line.

We build the lines from right to left, starting from the x^{th} line and ending with the 2^{nd} line.

Construction of the j^{th} line in a good star tree for $j = x .. 2$:

1. $t = M(d, l, j)$.
2. *MOVING* $v \dashrightarrow v^j_1$.
3. The chain $M(d, l, j) - 1$ TRANSFORMATIONS:

$v \to v^j_1 \to v^j_2$, $\quad v \to v^j_2 \to v^j_3$, $\quad ...,\quad v \to v^j_{M(d, l, j) - 1} \to v^j_{M(d, l, j)}$;

$t = M(d, l, j) - 1$, $\quad t = M(d, l, j) - 2$, $\quad ...,\quad t = 1$.

4. $M(d, l, j)^{th}$ TRANSFORMATION: $v^{j+1}_1 \to v^j_{M(d, l, j)} \to v$.
5. Starting the algorithm for constructing an almost good starlike tree on the j^{th} branch: the vertex v sends the message *Begin*(d, $M(d, l, j)$) to the vertex $v^j_{M(d, l, j)}$.

When the 2^{nd} line is built, then the first line is built, so the algorithm on the first ruler is simultaneously launched: the vertex v sends the *Begin*(d, $M(d, l, 1)$) to the vertex $v^1_{M(d, l, 1)}$.

An almost good starlike tree is constructed in the same way, but the number of lines x is not $min\{d, l - 1\}$ but $min\{d - 1, l - 1\}$, and the number of vertices in the j^{th} line is not $M(d, l, j)$ but $N(d, l, j)$.

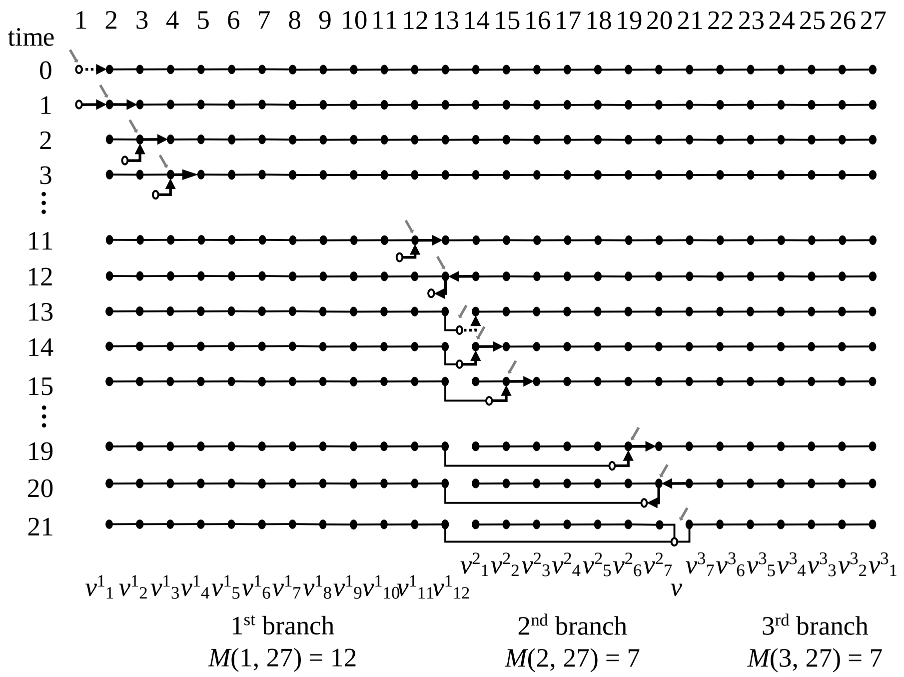

Fig. 6. Building a good starlike tree for $n = 27$ and $d = 3$.

5.2 Formal Description

Algorithm $\mathcal{B}(G, root)$.

Variables at vertex v:

- $E(v)$ is the set of edge numbers incident to the vertex v;
- $fat(v)$ is the number at the vertex v of the edge leading from the vertex v to its father;
- $sson(v)$ is the number of vertices in the branch $G(v)$;
- $d(v)$ is upper bound on degrees of vertices;
- $j(v)$ is the number of the starlike tree line under construction;
- $first(v)$ is the number of the first edge of the starlike tree line under construction;
- $nson(v)$ is the number of messages *End* expected by vertex v from its sons.

```
while is not the end of the algorithm {
wait () { /* receive message */
    if Begin(d, l) on the edge i is received {
        d(v) := d; sson(v) := l; fat(v) := i; /* initialization */
        if fat(v) = 0 { nson(v) := min{d, sson(v) - 1}; }
        else { nson(v) := min{d - 1, sson(v) - 1}; }
        j(v) := nson(v); /* start with the last line of ?a shortest? length */
        if sson(v) ≥ 3 { /* number of vertices ≥ 3, transformation needed */
            if fat(v) = 0 { t := M(d, sson(v), j(v)); } else { t := N(d, sson(v), j(v)); }
```

```
        first(v) := element( E(v) \ { i } ); /* select edge */
        send Move(t) on the edge first(v);
    }
    else { /* number of vertices ≤ 2, no transformation needed */
        send End() on the edge fat(v);
        if fat(v) = 0 { end of algorithm; } /* v = root */
} }
if Move(t) on the edge i is reseived {
    e := element( E(v) \ { i } ); /* select edge */
    if t > 1 { /* first transformation of the transformation chain */
        /* transformation a→v→b, где e(v, a) = i, e(v, b) = e */
        command CHANGE(i, e, t);
        E(v) := E(v) \ { i }; /* edge removed */
    }
    else { /* final transformation */
        /* transformation a→v→b, где e(v, a) = e, e(v, b) = i */
        command CHANGE(e, i, t);
        E(v) := E(v) \ { e }; /* edge removed */
} }
if Change(t) on the edge i is received {
    if t > 2 { /* continue the chain of transformations */
        e := element( E(v) \ { fat(v), i } ); /* select edge */
        /* transformation a→v→b, где e(v, a) = i, e(v, b) = e */
        command CHANGE(i, e, t - 1);
    }
    if t = 2 { final transformation */
        e := element( E(v) \ { fat(v), i } ); /* select edge */
        /* transformation a→v→b, где e(v, a) = e, e(v, b) = i */
        command CHANGE(e, i, t - 1);
    }
    if t = 1 { all transformations are finished */
        E(v) := E(v) ∪ { i }; /* edge added */
        /* start the algorithm on the constructed line */
        if fat(v) = 0 { t := M(d, sson(v), j(v)); } else { t := N(d, sson(v), j(v)); }
        if t ≥ 3 { send Begin(d(v), t) on the edge first(v); }
        else { nson(v) := nson(v) - 1; }
        j(v) := j(v) - 1; /* go to the next line */
        if fat(v) = 0 { t := M(d, sson(v), j(v)); } else { t := N(d, sson(v), j(v)); }
        if j(v) > 1 { /* start building the next line */
            first(v) := i; send Move(t) on the edge first(v);
        }
        if j(v) = 1 { /* all lines are constructed */
            /* start the algorithm on the last line */
            if t ≥ 3 { send Begin(d(v), t) on the edge i; }
            else { nson(v) := nson(v) - 1; }
```

```
}    }  }
if End() on the edge i is reseived {
     nson(v) := nson(v) - 1;
     if nson(v) = 0 { /* branch is constructed */
          send End() on the edge fat(v);
          if fat(v) = 0 { end of algorithm; } /* v = root */
}}    }  }
```

Proposition 10. Algorithm $\mathcal{B}$ transforms a linear tree with n vertices into a good tree without violating the restriction of d on the degree of vertices with the time complexity $t(n) \leq 2n - 2$.

This work was supported by RFBR project 17-07-00682-a.

References

1. Wiener, H.: Structural determination of paraffin boiling points. J. Am. Chem. Soc, 69 (1), 17–20 (1947).
2. Kochkarov, A.A., Sennikova, L.I., Kochkarov, R.A.: Nekotorye osobennosti primeneniia dinamicheskikh grafov dlia konstruirovaniia algoritmov vzaimodeistviia podvizhnykh abonentov. Izvestiia IuFU. Tekhnicheskie nauki, razdel V, cistemy i punkty upravleniia, 1, 207–214 (2015).
3. Proskochilo, A.V., Vorobev, A.V., Zriakhov, M.S., Kravchuk, A.S.: Analiz sostoianiia i perspektivy razvitiia samoorganizuiushchikhsia setei. Nauchnye vedomosti, seriia ekonomika, informatika, vypusk 36/1, 19 (216), 177–186 (2015) [in Russian].
4. Pathan, A.S.K. (ed.): Security of self-organizing networks: MANET, WSN, WMN, VANET. CRC press, (2010).
5. Boukerche, A. (ed.): Algorithms and protocols for wireless, mobile Ad Hoc networks. John Wiley & Sons (2008).
6. Chen, Z., Li, S., Yue, W.: SOFM Neural Network Based Hierarchical Topology Control for Wireless Sensor Networks. Hindawi Publishing Corporation, Journal of Sensors, vol. 2014, article ID 121278 (2014), http://dx.doi.org/10.1155/2014/121278.
7. Mo, S., Zeng, J.-C., Tan, Y.: Particle Swarm Optimization Based on Self-organizing Topology Driven by Fitness. In: International Conference on Computational Aspects of Social Networks, CASoN 2010, Taiyuan, China, 10.1109/CASoN, 13, 23–26 (2010).
8. Wen, C.-Y., Tang, H.-K.: Autonomous distributed self-organization for mobile wireless sensor networks. Sensors (Basel, Switzerland) 9 (11), 8961–8995 (2009).
9. Llorca, J., Milner, S.D., Davis, C.: Molecular System Dynamics for Self-Organization in Heterogeneous Wireless Networks. EURASIP Journal on Wireless Communications and Networking (2010), http://dx.doi.org/10.1155/2010/548016.
10. Wai-kai, C.: Net Theory and Its Applications: Flows in Networks. Imperial College Press (2003).
11. Wang, H.: On the Extremal Wiener Polarity Index of Hückel Graphs. Computational and Mathematical Methods in Medicine, vol. 2016, article ID 3873597, http://dx.doi.org/10.1155/2016/3873597.
12. Xu, X., Gao, Y., Sang, Y., Liang, Y.: On the Wiener Indices of Trees Ordering by Diameter-Growing Transformation Relative to the Pendent Edges. Mathematical Problems in Engineering, vol. 2019, article ID 8769428, https://doi.org/10.1155/2019/8769428.

13. Burdonov, I.B.: Samotransformatsiia derevev s ogranichennoi stepeniu vershin s tseliu minimizatsii ili maksimizatsii indeksa Vinera. Trudy Instituta sistemnogo programmirovaniia RAN, 31 (4), 189–210 (2019), https://doi.org/10.15514/ISPRAS-2019-31(4)-13. [in Russian].
14. The On-Line Encyclopedia of Integer Sequences (OEIS), http://oeis.org/, last accessed 2019/11/21.
15. Fischerman, M., Hoffmann, A., Rautenbach, D., Székely, L., Volkmann, L.: Wiener index versus maximum degree in trees. Discrete Applied Mathematics 122 (1–3), 127–137 (2002).