Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3
© 2005 - 2014 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PROBABILISTIC DISTRIBUTED ALGORITHM FOR MINIMUM SPANNING TREE CONSTRUCTION

[1]ISMAIL HIND, [2]EL MEHDI STOUTI, [3]ABDELAAZIZ EL HIBAOUI, [4]ALI DAHMANI
FS– Abdelmalek Essaâdi University
P.O. Box. 2121 M'Hannech II
93030 Tetuan Morocco
E-mail: [1]ismailhind@gmail.com, [2]stouti@uae.ma, [3]hibaoui@uae.ma, [4]alidahmani@hotmail.com

ABSTRACT

In this paper we introduce and analyze a probabilistic distributed algorithm for minimum spanning tree construction. Our algorithm is based firstly on the handshake algorithm that produces firstly *k* sub-spanning trees, where *k* is the size of the maximal matching produced. Secondly, the merged step of our algorithm is executed in a distributed manner and following some roles to reduce the total number of those sub-spanning from *k* to 1. We proof that the residual graph is acyclic and all vertices belong to it. A detailed analysis of the number of exchanged messages is carried on to validate the effectiveness of our algorithm.

Keywords: *Distributed Algorithms, Randomized Algorithm Analysis, Handshake, Maximal Matching, Minimum Spanning Tree Construction.*

1. INTRODUCTION

The minimum spanning tree problem is a well-known combinatorial optimization problem concerned in linking all the vertices of an undirected and connected graph without creating any cycle. The methods for finding a minimum spanning tree have played a central role in the design of computer algorithm [1].

The earliest known algorithm for finding a minimum spanning tree (MST) was given by Otakar Borüvka back in 1926 [2]. In every step of the algorithm each vertex selects its smallest adjacent edge. These edges are added to the MST without creating any cycle.

Kruskal's algorithm was given by Joseph Kruskal in 1956 [3]. It creates a forest where each vertex in the graph is initially a separate tree. For each edge *(u, v)* in sorted order, this algorithm does the following: If the vertices *u* and *v* belong to two different trees, then add *(u, v)* to the forest, combining two trees into a single tree. It proceeds until all the edges have been processed.

Prim's algorithm was conceived by Robert Prim in 1957 [4]. It starts from an arbitrary vertex, and builds upon a single partial minimum spanning tree, at each step adding an edge connecting the vertex nearest to, but not already in the current partial minimum spanning tree. It grows until the tree spans all the vertices in the input graph.

In this paper, we present and analyze a probabilistic distributed algorithm to construct a minimum spanning tree without a specific criterion. Our algorithm is based on the handshake algorithm presented in [5] which is a probabilistic distributed algorithm for finding a maximal matching.

We consider that all edges, where the handshakes are occurred, are in the minimum spanning tree. Those edges constitute the base for our algorithm to construct a minimum spanning tree. In a distributed manner, we join those edges and all the isolated vertices in a connected graph avoiding any cycle. The residual graph is the minimum spanning tree that we are looking for.

The paper is organized as follows: Section 2 presents some notifications and definitions necessary for understanding the rest of the document. It exposes also our model and assumptions. Section 3 is devoted to describe our algorithm whereas Section 4 shows the analysis of the introduced algorithm. Finally, Section 5 concludes the paper and presents our further work.

2. DEFINITIONS AND MODEL

2.1 Definitions and notations

In graph theory, given a graph *G=(V,E)*, a matching *M* in *G* is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex where *V* is the set of all the vertices of *G* and *E* is the set of the edges linking those vertices.

A vertex is matched (or saturated) if it is an endpoint of one of the edges in the matching. Otherwise the vertex is unmatched [6].

A *maximal matching* is a matching *M* of a graph *G* with the property that if any edge not in *M* is added to *M*, it is no longer a matching, that is, *M* is maximal if it is not a proper subset of any other matching in graph *G*. In other words, a matching *M* of a graph *G* is maximal if every edge in *G* has a non-empty intersection with at least one edge in *M*. The following figure shows an example of maximal matching (edges in bold) in the graph.

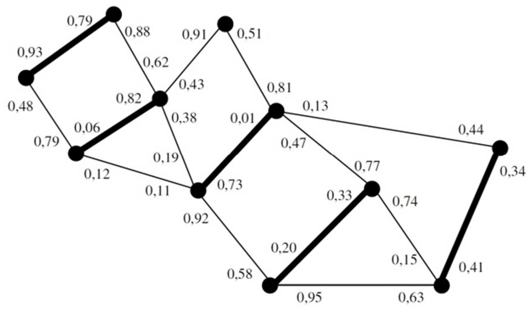

Figure 1: Example Of Maximal Matching Where The Endpoints Of The Bold Edges Are Matched.

On the other side, a spanning tree *T* of a connected and undirected graph *G* is a tree composed by all vertices and a sub-set of the set *E*. A minimum spanning tree of *G* is a selection of edges of *G* that form a minimum spanning tree [7]. That is, every vertex lies in the tree, but no cycles (or loops) are formed. The following figure shows an example of the minimum spanning tree obtained from the graph in Figure 1.

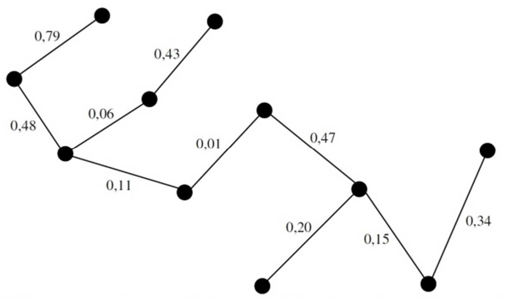

Figure 2: Minimum Spanning Tree Of The Graph Of The Figure 1.

2.2 Model and Assumptions

In this paper, we consider the standard model of communication networks point-to-point [8]-[9]. A network is described by a undirected and connected graph *G=(V,E)* where *V* is the set of all vertices of *G* and *E* is the set of edges linking those vertices. The vertices represent the nodes or network processes, the edges represent the communication links between processes.

We assume that each process has a local unshared memory with a bounded capacity and at least one processor. It can only communicate directly with its neighbours by message exchanges. Also, it is able to distinguish between its ports (i.e. a port from which a message is received or sent). The identities (IP addresses on the Internet) of other processes, in particular those of its neighbours, are unknown (local orientation).

For our study, processes communicate in the network only by message exchanges. The exchange of messages is done without loss or alteration or modification. How exchanges between processes are carried out determines the type of network.

A global clock is common to all processes, and we admit that every event that takes place at a certain time is of zero duration, and we suggest that communications take no time (i.e. the period between the decision of a vertex and notification to its neighbours is equal to 0).

We used graph theory to represent and analyze our network. So, a network is described by a undirected and connected graph. The vertices represent the nodes or network processes, whereas the edges represent the communication links between those processes.

3. ALGORITHM

The main aim behind this study is to suggest and analyze a distributed algorithm for creating a minimum spanning tree based on handshake algorithm.

Many researches had trait the maximal matching problem such as *MSZ* algorithm [10], Kuhn algorithm [11], Heuberger algorithm [12] and *HS* algorithm [5]. In *MSZ* algorithm the authors proposed and analyzed a randomized algorithm to get handshakes between neighbours in an anonymous graph, and they also showed the efficiency of their algorithm in several types of graphs.

In *HS* algorithm, the authors had introduced and studied a probabilistic distributed algorithm to find a maximal matching. This algorithm is based on random delays which are generated uniformly in the real interval [0, 1]. A handshake takes place between a pair of neighbour processors if both processors are free at the proposed time.

In our proposed algorithm, we use the *HS* algorithm because the expected number of handshakes found by this algorithm is substantially greater than that obtained in other known probabilistic distributed algorithms.

So, our algorithm for finding a minimum spanning tree merges two algorithms. The first one is for finding the handshakes between a set of pairs of vertices, whereas the second one is executed to link the founded handshakes with the isolated vertices to make one acyclic graph. This algorithm can be considered as a distributed various of the *Borüvka* algorithm [2].

However, we enrich the *HS* algorithm by some instructions in order to construct an acyclic graph which links all the handshakes and the remaining isolated vertices. As proven in [5], the *HS* algorithm provides a maximal number of handshakes. The informal description of this algorithm is as follows:

- For a given graph *G*, each vertex $v \in G$ generates uniformly a random variable for each neighbour. Those random variables generated by a vertex for its neighbours belong to the real interval [0, 1]. They represent the waiting times to get a handshake with its neighbours. The number of waiting times generated by all this vertices is two-times the number of the edges (i.e., one waiting times for each half-edge). The vertex agenda is the set of all waiting times proposed by this vertex to its neighbours.
- When the clock reaches the smallest time of all the generated times, a handshake occurs between the vertex that provides this time and the neighbour who is assigned to it. So the two vertices remove all other edges with the other neighbours. Indeed, their neighbours remove from their agendas the times that they offer to these two vertices which had a handshake.
- The algorithm continues to run through the remaining process equipped with modified agendas until the lap time (unit time) expired.

The formal description of our algorithm is given in Algorithm 1.

We define the following parameters for each vertex $v \in G$:

- $t_{u,v}$ the waiting time for a vertex v to get a handshake with a neighbour u. It is a random variable uniformly chosen by the vertex v in the real interval [0, 1] for a neighbour u.
- A_v the agendas of *v*. It is the set of all values generated by *v* to its neighbours.
- $A_{v,u} = A_v \cup A_u \setminus \{t_{v,u}, t_{u,v}\}$. The set of shared agendas between the two neighbours *v* and *u*.
- t_{min} the minimum waiting time of the vertex *v*. This value is equal to: $t_{min} = \min(A_v)$.
- T_{min} the minimum waiting time of all the value generated by the neighbours u and *v*. This value is equal to: $T_{min} = \min(A_{v,u})$.
- $S_{v,u}$ the state of the edge $\{v, u\}$, it is a Boolean variable initiated to *false* (i.e. the edge $\{v, u\}$ is not initially in the minimum spanning tree). When it becomes *true*, the edge $\{v, u\}$ is in the minimum spanning tree.
- S_v the state of the vertex *v*. If it is true, the vertex *v* is in the minimum spanning tree.
- S_{min}^{v} is the set of the minimum waiting times proposed by vertices. This set is used to be sure that no cycle is formed.
- $msg < value, set1, set2 >$ the form of the exchanged messages by vertices.

Algorithm 1: Minimum Spanning Tree Algorithm

```
A vertex $v$ waits until one of the following events happened:
$\mathcal{A}'_v := \mathcal{A}_v$;
$S^v_{min} := \emptyset$;
$t_{min} := min(\mathcal{A}_v)$;
while $((t < t_{min}))$ or $(t \neq 1))$ do
    recieve msg$< n, \mathcal{A}_u, S^u_{min} >$ From $u$;
    $\mathcal{A}_{u,v} := \mathcal{A}_u \cup \mathcal{A}_v \setminus \{t_{u,v}, t_{v,u}\}$;
    if $(\mathcal{A}_{v,u} \neq \emptyset)$ then
        $T_{min} := min(\mathcal{A}_{u,v})$;
        switch $(n)$ do
            case 0
                $\mathcal{A}_v := \mathcal{A}_v \setminus \{t_{v,u}\}$;
            case 1
                $S_v := true$;
                $S_{v,u} := true$;
                if $(T_{min} \in \mathcal{A}_v)$ then
                    send $msg < 2, \emptyset, S^v_{min} >$ to $u'$
                    such that $t_{v,u'} := T_{min}$;
                else
                    send $msg < 3, \emptyset, S^v_{min} >$ to $u$;
                    send $msg < 0, \emptyset, \emptyset >$ to others;
            case 2
                if $(S^v_{min} \cap S^u_{min} = \emptyset)$ then
                    send $msg < 4, \emptyset, \emptyset >$ to $u$;
                    $S_{v,u} = true$;
            case 2
                if $(S^v_{min} \cap S^u_{min} = \emptyset)$ then
                    send $msg < 4, \emptyset, \emptyset >$ to $u$;
                    $S_{v,u} = true$;
            case 3
                $S^v_{min} := S^v_{min} \cup S^u_{min}$;
                send $msg < 2, \emptyset, S^v_{min} >$ to $u'$ such
                that $t_{v,u'} = T_{min}$;
            case 4
                $S_{v,u} = true$;
if $(t = t_{min})$ and $(S_v \neq true)$ then
    $S_v := true$;
    $S_{v,u} := true$;
    send $msg < 1, \mathcal{A}_v, S^v_{min} >$ to $u$ such that
    $t_{v,u} := t_{min}$;
    send $msg < 0, \emptyset, \emptyset >$ to others ;
else
    $(* \ t = 1 \ *)$ select $u$ such that $t_{v,u} = min(\mathcal{A}'_v)$;
    send $msg < 2, \emptyset, \emptyset >$ to $u$;
    $S_v := true$ ;
    $S_{v,u} := true$;
```

Our algorithm works as follows:

Each vertex v waits until one of the following events happened:

- If v receives the message $msg < 0, \emptyset, \emptyset >$ from a neighbour u then it removes the value $t_{v,u}$ from its set A_v.
- If v receives the message $msg < 1, A_u, S^u_{min} >$ from a neighbour u then it changes the states S_v and $S_{v,u}$ to *true*. There is a handshake between v and u. Consequently, the edge {v, u} belongs to the minimum spanning tree. In this case, one of the two following cases occurs:

  1. If $T_{min} \in A_v$ where $T_{min} = \min(A_{v,u})$, then the edge *{v, u'}* will belong to the minimum spanning tree. The message $msg < 2, \emptyset, S^v_{min} >$ will be sent to u' to inform it that the edge *{v, u'}* will belong to the minimum spanning tree. Then $t_{v,u'}$ affects the value of T_{min}.
  2. Otherwise, v sends the message $msg < 3, \emptyset, S^v_{min} >$ to u to give it the option of choosing the edge that will connect with the edge *{v, u}* in the minimum spanning tree. In the same time, v sends the message $msg < 0, \emptyset, \emptyset >$ to others.

  If v receives the message $msg < 2, \emptyset, S^u_{min} >$ from a neighbour u then if $S^v_{min} \cup S^u_{min} = \emptyset$, which means that the edge *{v, u}* doesn't construct any cycle, v sends the message $msg < 4, \emptyset, \emptyset >$ to u and changes its state $S_{v,u}$ to *true*.
- If v receives the message $msg < 3, \emptyset, S^u_{min} >$ from a neighbour u then v adds the set S^u_{min} to S^v_{min} and sends the message $msg < 2, \emptyset, S^v_{min} >$ to a neighbour u' which verify $t_{v,u'} = T_{min}$.
- If v receives the message $msg < 4, \emptyset, \emptyset >$ from a neighbour u then v turns its state $S_{v,u}$ to *true*. However, this test is to guarantee that no cycle will be created by adding the edge *{v, u}* to the minimum spanning tree.
- If $t = t_{v,u}$ and $S_v \neq true$ then v turns the states S_v and $S_{v,u}$ to *true* and sends the message $msg < 1, A_v, S^v_{min} >$ to u such that $t_{v,u} = t_{min}$. In addition, v sends $msg < 0, \emptyset, \emptyset >$ to all other neighbours. There is a handshake between v and u.
- Otherwise (* *t = 1* which means that v is an isolated vertex *), v sends the message $msg < 2, \emptyset, \emptyset >$ to u such that $t_{v,u} = \min(A'_v)$. Consequently, v changes its states the states S_v and $S_{v,u}$ to *true*.

Journal of Theoretical and Applied Information Technology
30th September 2014. Vol. 67 No.3
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. ALGORITHM ANALYSES

The algorithm starts by generating uniformly $2|E|$ real independent random variables in the real interval [0, 1]: a random variable for each half edge. We assume that (2|E|)! permutations on the set of these real numbers have the same probability. This is the main assumption on which is based the result of our analysis.

In order to maintain this hypothesis, we simply assume that, for each edge $e = \{u, v\} \in G$, the algorithm will produce two continuous r.v. $t_{v,u}$ and $t_{u,v}$, we assume that those r.v. associated to the edges are independents. The probability to generate the same values tends to zero.

The first handshake takes place on an edge $e = \{u, v\}$, if at least one of the two associated random variables. $t_{v,u}$ or $t_{u,v}$ is minimal in the whole graph. Thus, for the first handshake of *G*, each edge has the same chance $1/|E|$ to be chosen.

The attribution of the first handshake on the edge *{u, v}* implies that the incident edges of the vertices *u* and *v* are removed from the graph except the edge *{u, v}*. The algorithm continues to run on the residual graph (preserving the random generation of the remaining edges) until only the edges on the handshakes occur remain.

The number of handshakes in a round is simply the total number of edges on which handshake are assigned. We denote by *N(G)* this number. It takes the value 0 with probability 1 if $E = \emptyset$, the value 1 with probability 1 if $|E| = 1$. Generally, it takes a value of the set of all maximal matching cardinalities in *G*, with a certain probability.

4.1 Sub-Spanning Trees

Proposition 4.1: Let *G= (V, E)* be a graph with $|V|=n$ and $|E|=m$, the probability to have initially *k* sub-spanning trees is:

$$\Pr(N(G) = k) = \frac{1}{m} \sum \Pr(N(G_e) = k - 1),\ \forall k \geq 1,$$

where G_e is a residual graph that contains all the edges of *G* except *e* and it incident edges.

4.2 Expected Number of Sub-Spanning Trees

In a randomized distributed algorithm, an important parameter which measures the efficiency is the expected number of events which can take place in a unit of time. We study this parameter for our new algorithm. So, let *𝔼(G)* be the mathematical expectation of *N(G)* and let $M_e(G)$ a r.v. that equals 1 if the edge *e={v, u}* is in the minimum spanning tree and 0 otherwise, for the edge *e* in the graph *G = (V, E)*.

Proposition 4.2: Initially, the number of the sub-spanning trees can be written as:

$$N(G) = \sum_{e \in E} M_e(G).$$

Proposition 4.3: The mathematical expectation of the number of sub-spanning trees, noted $\mathbb{E}(N(G))$, is given by the following formula:

$$\mathbb{E}(N(G) = \sum_{k} kPr\,(N(G) = k)$$
$$= \sum_{e \in E} \mathbb{E}(M_e(G)).$$

It is clear that the construction of the spanning tree of the graph *G* is done by merging several sub-spanning trees. The sub-spanning trees are basically produced by the edges where the handshakes are occurred. Initially the number of sub-spanning trees is equal to the number of maximal matching. Through the time the algorithm adds to those sub-spanning trees the isolated vertices and sometimes merges some one of them. In the end of the algorithm all sub-spanning trees are merged on one minimum spanning tree. We recall that the edges that used to merge the handshake edges are not used in the *HS* algorithm.

4.3 Minimality Proof

It is easy to see that the spanning tree produced is minimal. Indeed, the *HS* algorithm selects the edges, as a sub-spanning tree, which has the minimum values in the whole graph. However, the second step uses the minimum edges to merge them with the spanning tree. Consequently, the produced spanning tree is minimal.

4.4 Message Complexity

One of the parameters of complexity measures, we found the number of exchanged messages. In the next we will calculate the number of the maximum exchanged messages for the algorithm of the spanning tree construction. The number of exchanged messages depends on each topology. So we will analyze this number in some topologies.

4.4.1 Star Graph

In graph theory, a star graph S_k is a complete bipartite graph with one internal node and *k-1* leaves. Let the graph *G=(V, E)* be a star graph such that *V* represents the set of its vertices and *E* represents it edges. In this case the maximal matching is equal to 1, for odd or even number of vertices.

- For an odd star, let |*V*|=*n* and |*E*|= *n-1* where $n \in \mathbb{N}$. The cardinal of maximal matching is equal to 1. So we need exactly another *n*-2 edges to reconstruct the minimum spanning tree.
- For an even star, let |*V*|=*n* and |*E*|=*n*-1 where $n \in \mathbb{N}$. We need exactly *n*-2 edges to reconstruct the minimal spanning tree since the cardinal of maximum matching is equal to 1.

The Figure 3 shows an example of a maximal matching in a star graph with |*V*|=13.

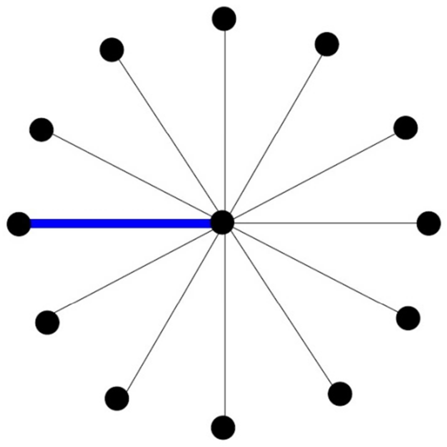

Figure 3: Example Of A Maximal Matching In A Star Graph.

Proposition 4.4: The upper-bound of the number of maximal exchanged messages in the star graph, noted $N_{msg}(n)$, is given by the following formula:

$$\begin{cases} N_{msg}(1) = 0 \\ N_{msg}(2) = 2 \\ N_{msg}(n) \leq 2n - 1, \ \ \forall n \geq 3, \end{cases}$$

where *n* is the size of the graph (|*V*|=*n*).

Proof: Let the graph *G=(V, E)* be a star graph (|*V*|=*n*, |E|=*m*) and let U_n be the arithmetic sequence that verify: $U_3 = 5$, and $U_n = 2n - 1$.

For *n*=3, it is clear that the number of maximal exchanged messages for |*V*|=3 is less than or equal to 5 ($N_{msg}(3) \leq 5$). By adding a new vertex this number will increase by 2. However, for *n*=4, the number of maximal exchanged messages $N_{msg}(4)$ is equal to 7. Let propose now that $N_{msg}(n) \leq 2n - 1$ and prove that $N_{msg}(n + 1) \leq 2n + 1$. In the worst case by adding a new vertex to *G*, the number of maximal exchanged messages will increase by 2.

So, $N_{msg}(n + 1) \leq 2n - 1 + 2 = 2n + 1$ which proves the proposition.

4.4.2 Chain Graph

In graph theory, a chain graph is a sequence of edges that connect a set of vertices which, by most definitions, are all distinct from one to another and they are not creating any cycle. Let the graph *G=(V, E)* be a chain such that *V* represents the set of its vertices and *E* represents it edges.

We distinguish between several cases:

- For an odd chain, let |*V*|=*n* and |*E*|=*n-1* such that $n \in \mathbb{N}$. The cardinal of maximum matching is equal to $\left\lfloor\frac{n}{2}\right\rfloor$. So we need exactly another $\left\lfloor\frac{n}{2}\right\rfloor$ edges to reconstruct the minimum spanning tree.
- For an even chain, let |*V*|=*n* and |*E*|=*n*-1 such that $n \in \mathbb{N}$.
  - In the bets case the cardinal of maximal matching is equal to $\frac{n}{2}$. So we need exactly another $\frac{n}{2} - 1$ edges to reconstruct the minimum spanning tree.
  - In the worst case, the cardinal of maximal matching is $\frac{n}{2} - 1$. So we need exactly another $\frac{n}{2}$ edges to reconstruct the minimum spanning tree.

The Figure 4 presents an example of a maximal matching in a chain with |*V* | = 7.

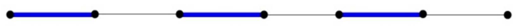

Figure 4: Example Of A Maximal Matching In A Chain Graph.

Proposition 4.5: The upper-bound of the number of maximal exchanged messages in the chain graph, noted $N_{msg}(n)$, is given by the following formula:

$$\begin{cases} N_{msg}(1) = 0 \\ N_{msg}(n) \le 3n - 4, \ \forall n \ge 2, \end{cases}$$

where n is the size of the graph (|*V*| = *n*).

Proof: Let the graph *G = (V, E)* be a chain (|*V* | = *n* and |*E*| = *n* - 1) and let U_n be the arithmetic sequence that verify:

$U_2 = 2$, and $U_n = 3n - 4, \ \forall n \ge 2.$

For *n*=2, it is clear that the number of maximal exchanged messages for is less than or equal to 2 ($N_{msg}(2) \le 2$). By adding a new vertex this number will increase by 3. However, for *n*=3, the number of maximal exchanged messages ($N_{msg,3}$) is equal to 5. Let propose now that $N_{msg}(n) \le 3n - 4$ and prove that $N_{msg}(n+1) \le 3n - 1$. In the worst case, the addition of a new vertex to *G*, the number of maximal exchanged messages will increase by 2. So, $N_{msg}(n+1) \le 3n - 4 + 3 = 3n - 1$ this proves the proposition.

4.4.3 Ring Graph

In graph theory, a ring is a graph consists of a sequence of vertices starting and ending at the same vertex, with each two consecutive vertices in the sequence adjacent to each other in the graph.

We distinguish between several cases (even, odd, best case, worst case):

- For an odd ring, let |*V*|=|*E*|= *n*, such that $n \in \mathbb{N}$. The cardinal of maximum matching is equal to $\left\lfloor \frac{n}{2} \right\rfloor$. So we need exactly another $\left\lfloor \frac{n}{2} \right\rfloor$ edges to reconstruct the spanning tree.
- For an even ring, let |*V*|=|*E*|=*n*, such that $n \in \mathbb{N}$. In the bets case, the cardinal of maximum matching is equal to $\frac{n}{2}$. So we need exactly another $\frac{n}{2} - 1$ edges to reconstruct the spanning tree.

The Figure 5 shows an example of a maximal matching in a ring with |*V*| = *5*.

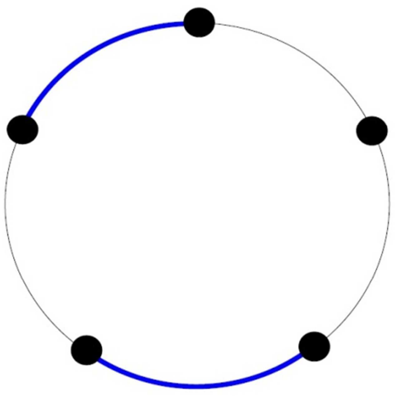
Figure 5: Example Of A Maximal Matching In A Ring Graph.

Proposition 4.6: The upper-bound of the number of maximal exchanged messages in the ring graph, noted $N_{msg}(n)$, is given by the following formula:

$$\begin{cases} N_{msg}(1) = 0 \\ N_{msg}(n) \le 4n - 6, \ \forall n \ge 2, \end{cases}$$

where n is the size of the graph.

4.4.4 Double-Star Graph

Let *G=(V, E)* be a double-star graph where |*V*|=*n* and |*E*|= *n-1*. In this case the maximal matching is equal to $\left\lfloor \frac{n}{2} \right\rfloor$. So we need exactly another $\left\lfloor \frac{n}{2} \right\rfloor$ edges to reconstruct the minimum spanning tree.

The Figure 6 presents an example of a maximal matching in a double-star graph with |*V*|=*25*.

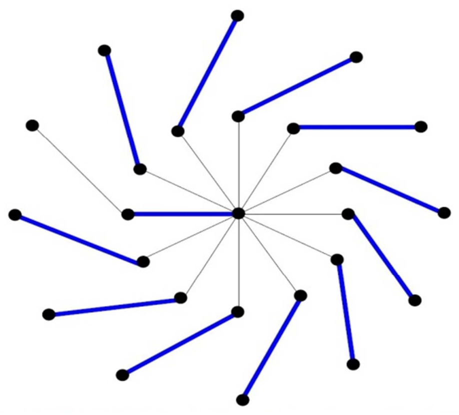
Figure 6: Example Of A Maximal Matching In A Double-Star Graph.

Proposition 4.7: The upper-bound of the number of maximal exchanged messages in a double-star graph, is given by the following formula:

$$\begin{cases} N_{msg}(1) = 0 \\ N_{msg}(n) \leq 3n - 4, \ \ \forall n \geq 2 \end{cases}$$

where n is the size of the graph.

5. CONCLUSION

In this paper, we have introduced and analyzed a probabilistic distributed algorithm for the minimum spanning tree construction based on a handshake algorithm. According to our algorithm, each vertex executes, firstly, a set of instructions and decides if it is matched or not.

The edges on which the handshakes occurred constitute a set of sub-spanning trees. Moreover, the second step of the proposed algorithm is lunched to merge those sub-spanning trees until reduce their number to one. The residual graph obtained is the minimal spanning tree that we are looking for.

As perspective of this work, we attend to calculate both the lower and upper bounds of the expectation time duration of the minimum spanning tree construction, and this for any type of graph. In addition, we attend to modelize the process of the minimum spanning tree construction by a Markov-chain process in continuous time.

REFRENCES:

[1] Y. Matsumoto, N. Kamiyama, and K. Imai, "An approximation algorithm dependent on edge-coloring number for minimum maximal matching problem," *Information Processing Letters,* vol. 111, no. 10, pp. 465 – 468, 2011.

[2] G. Bergantiños and J. Vidal-Puga, "The folk solution and Borüvka's algorithm in minimum cost spanning tree problems," *Discrete Applied Mathematics,* vol. 159, no. 12, pp. 1279 – 1283, 2011.

[3] Z. Ning and W. Longshu, "The complexity and algorithm for minimum expense spanning trees," *Procedia Engineering,* vol. 29, no. 0, pp. 118 – 122, 2012, 2012 International Workshop on Information and Electronics Engineering.

[4] C. Martel, "The expected complexity of prim's minimum spanning tree algorithm," *Information Processing Letters,* vol. 81, no. 4, pp. 197 – 201, 2002.

[5] A. El Hibaoui, Y. Métivier, and N. Z. A. Robson, J.M.and Saheb, "Analysis of a randomized dynamic timetable handshake algorithm," 2009.

[6] R. Diestel, Graph Theory, second edition,electronic ed. Berlin: *Springer*, February 2000.

[7] M. C. Golumbic, Algorithmic graph theory and perfect graphs, 2nd ed. *Elsevier*, 2004.

[8] G. Tel, Introduction to distributed algorithms. *Cambridge University Press*, 2000.

[9] C. Lavault, Évaluation des algorithmes distribués : analyse, complexité, méthodes, ser. *Collection Informatique*. Paris: Hermès, 1995 (53-Mayenne.

[10] Y. Métivier, N. Saheb, and A. Zemmari, "Analysis of a randomized rendezvous algorithm," *Inf. Comput*, vol. 184, no. 1, pp. 109–128, 2003.

[11] D. Kühn, D. Osthus, and T. Townsend, "Fractional and integer matchings in uniform hypergraphs," *European Journal of Combinatorics,* vol. 38, no. 0, pp. 83 – 96, 2014.

[12] C. Heuberger and S. Wagner, "The number of aximum matchings in a tree," *Discrete Mathematics*, vol. 311, no. 21, pp. 2512 – 2542, 2011.