 doi:10.20944/preprints201911.0194.v1

A Robot's Response Acceleration using the Metric Dimension Problem

E. M. Badr[1] and Khalid Aloufi[2]

[1]Scientific Computing Department, Faculty of Computers & Artificial Intelligence, Benha University, Benha, Egypt. badrgraph@gmail.com

[2]College of Computer Science & Engineering Taibah University Saudi Arabia koufi@taibahu.edu.sa

Abstract

Consider a robot that is navigating in a space modeled by a graph, and that wants to know its current location. It can send a signal to determine how far it is from each landmark among a set of fixed landmarks. We study the problem of computing the minimum required number of landmarks, and where they should be placed so that the robot can always determine its location. Since the problem is an *NP-complete* problem, the robot's responses to the actions are slow. To accelerate this response, we can use the parallel version of this problem. In this work, we introduce a new parallel implementation for determining the metric dimension of a given graph. We run the proposed algorithm on a symmetric multi-processing (SMP) cluster using C programming language and the Message Passing Interface (MPI) library. Finally, we run our implementation on four categories of graphs (the tracks in which the robot moves): a cycle graph C_n, a path graph P_n, a triangular snake graph Δ_k and a ladder graph L_n. Preliminary computational results indicate that the metric dimension problem is an *NP-complete* problem and prove the ability of the proposed algorithm to achieve a speedup of 6 for 8 processors.

Keywords - Parallel Processing; MPI cluster; metric dimension; resolving set; NP-complete.

 doi:10.20944/preprints201911.0194.v1

1. Introduction

The metric dimension of a graph has demonstrated to be useful and has many applications such as Robotic Navigation [1], [2], Chemistry [3], [4] and Combinatorial Search and Optimization [5].

Consider a robot that is navigating a space modeled by a graph and that wants to know its current location. It can send a signal to determine how far it is from each landmark among a set of fixed landmarks. We study the problem of computing the minimum required number of landmarks and where they should be placed so that the robot can always determine its location. The problem is an *NP-complete* problem [6], which means that the robot's responses to the actions are slow. To accelerate this response, we can use the parallel version of this problem.

For a definition of the metric dimension, let G be a connected graph and $d(u, v)$ be the distance between the vertices u and v. A subset of vertices $W = \{w_1, \ldots , w_k\}$ is called a resolving set for G if for every two distinct vertices $u, v \in V(G)$ there is a vertex $w_i \in W$ such that $d(u, w_i) \neq d(v, w_i)$. The metric dimension $md(G)$ of G is the minimum cardinality of a resolving set for G. Harary and Melter [6] and Slater [7] independently defined the metric dimension problem. The metric dimension problem is known as the *locating number* or *rigidity* problem and as *Harary's problem*.

There are three reasons to study this problem. First, very little is known about the time complexity of this problem. Second, the serial implementation of this problem takes significant time to get the final result (e.g., 994 seconds for graph G with 20 vertices). Third, we hope to accelerate the robot's responses to the actions. Gerey and Johnson [8] showed that finding the metric dimension of a given graph is an *NP-complete* problem. Recently, Josep Diaz *et al*. [9] proved that the metric dimension of planar graphs with bounded degree is NP-complete. In addition, they introduced a polynomial-time algorithm for finding the metric dimension of outerplanar graphs.

E. M. Badr and K. Aloufi [10] proposed an exponential algorithm for finding the metric dimension of a given graph, which has a time complexity of $O(n^2 \text{ x } 2^n)$. They also introduced the results of computer calculations that found the metric dimension of various classes of networks by using an approximate algorithm, namely, integer linear programming. E. M. Badr *et al*. [11] introduced polynomial algorithms for special graphs, namely, mirror graphs, square graphs, Tortoise graphs, $Z\text{-}(P_n)$ graphs and middle graphs.

Integer linear programming is an efficient approach for determining the metric dimension of graphs with big orders. We can see how to formulate graph problems for mathematical models in [12]. Chartrand et al. [13] introduced the metric dimension problem as an integer programming problem. James Daniel Currie and Ortrud R. Oellermann [14] proposed another efficient mathematical model as an integer programming problem.

There are two approaches for distributing the serial code into more than one processor. One is the row distribution scheme and the other is the column distribution scheme. Badr *et al.* [15] used the row distribution scheme and presented a well-designed, quite valuable implementation for eight loosely coupled processors (interconnected with Fast Ethernet and the Scalable Coherent Interface (SCI)), targeting and achieving significant speedup (up to five) when solving small random dense linear programming problems.

Figure 1 shows how the different signals determine the robot's location. Suppose the robot moves on the path graph (straight line) and there is a landmark vertex (v_4) that sends different signals (the distances are 3, 2, 1, and 0 between v_1, v_2, v_3, and v_4 and the landmark vertex v_4, respectively) to the robot to determine its location.

In this paper, we introduce a new parallel implementation for determining the metric dimension of a given graph. We run the proposed algorithm on a symmetric multi-processing (SMP) cluster using C programming language and the Message Passing Interface (MPI) library. Finally, we run our implementation on four categories of graphs (the tracks in which the robot moves): a cycle graph C_n, a path graph P_n, a triangular snake graph Δ_k and a ladder graph L_n. Preliminary computational results ensure that the metric dimension problem is an *NP-complete* problem and prove the ability of the proposed algorithm to achieve a speedup of 6 for 8 processors.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 November 2019 doi:10.20944/preprints201911.0194.v1

2. A serial exponential algorithm for finding the metric dimension of a graph

In this section, we introduce some definitions of graphs (the tracks in which the robot moves).

Definition 1 [16]:

A triangular snake (or Δ_k *-snake*) is a connected graph in which all blocks are triangles and the block-cut-point graph is a path.

Definition 2: The ladder graph L_n is defined by $L_n = P_n$ x K_2, where P_n is a path with n vertices, x denotes the Cartesian product and K_2 is a complete graph with two vertices.

The main aim of this section is to introduce an algorithm that gives a metric dimension for a given graph G. The time complexity of this algorithm is exponential and is $O(n^2 \text{x} 2^n)$ [10]. Recall that Gerey and Johnson [6] showed that determining the metric dimension of an arbitrary graph is an *NP-complete* problem. Recently, Josep Diaz et al. [9] proved that the metric dimension of planar graphs with bounded degree is NP-complete. In addition, they introduced a polynomial-time algorithm for finding the metric dimension of outerplanar graphs.

The parameters of Algorithm 1 are the following:

$A[i][j]$: the adjacency matrix of graph G,

S[i]: all of subsets of $V(G)$ (power set with n vertices),

$D[i][j]$: the distance matrix of a graph G,

$E[i][j]$: the distance matrix that corresponds to the subset $S[i]$,

Cardinality: the order of subset $S[i]$, and

MetricDimension: the metric dimension of graph G.

The main function of the proposed algorithm (Algorithm1) includes four procedures, Floyd –Warshall, Initialization, Get-Next-Subset and Check-Subset, as follows:

Algorithm 1: A serial algorithm for finding the metric dimension of a given graph

Input: An adjacency matrix $A[n][n]$ of an n-vertex simple connected graph G.
Output: A metric dimension of G.
Begin
1: Call Procedure1 (Floyd -Warshall's algorithm)
2: Call Procedure 2 (Initialization)
3: **while** Not Done **do**
4:Call Procedure 3 (Get-Next-Subset)
5:**for** i = n to 1 **do**
6: **for** j = n to 1 **do**
7:    **if** $S[j]$ = 1 **then**

doi:10.20944/preprints201911.0194.v1

```
            E[i][,j] ← D[i][,j]
        end
    end
end
Call Procedure 4 (Check-Subset)
if(E[i][:] or E[j][:])and(metric_dimension> cardinality)then
            MetricDimension = cardinality;
end
End /* Begin
```

The Floyd -Warshall procedure determines the distance matrix of a graph G. It is known that the Floyd -Warshall algorithm has the time complexity of $O(n^3)$ because it has three inner loops.

Procedure 1: Floyd -Warshall algorithm for finding the distance matrix of a graph

```
for i =1 to n do
    for j = 1 to n do
        A(i,j) ← inf;
        A(i,i) ← 0;
    end
end
D ← A
minn ← inf;
    for k = 1 to n do
        for i =1 to n do
            for j =1 to n do
                x ← D[i][j]
                y ← D[i][k] + D[k][j]
              if x < y then
                  minn ← x;
              else
                  minn ← y;
              end
              D[i][j] ← minn;
            end
        end
    end
```

The Initialization procedure initializes the initial subset of $V(G)$ as array $S[j] = 0$ and the distance matrix $E[i][j] = 0$, which corresponds to the subset $S[i]$. We outline the initialization procedure that considers the empty subgraph as the first subset as follows.

Procedure 2: Initialization

```
for i = n to 1 do
    S[i] ← 0
    for j= n to 1 do
        E[i][,j] ← 0
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 November 2019 doi:10.20944/preprints201911.0194.v1

```
      end
  end
  cardinality ← 0
  MetricDimension ← 0
  Done ← false
```

The Get-Next-Subset procedure generates all subsets $S[i]$ of $V(G)$ using the binary counting representation method.

Procedure 3: Get-Next-Subset

```
j ← n + 1
repeat
    j ← j - 1
until (( S[j] = 0) or ( j = 0)
if j ≠ 0 then
  S[j] ← 1
  MAX ← j
  for i = MAX +1 to n  do
        S[i] ← 0
  end
else
  Done ← True
end if
```

The Check-Subset procedure verifies whether the current subset $S[i]$ is a resolving set or not. A precise description of this process is the following.

Procedure 4: Check-Subset

```
for i = 1 to n-1 d
    for j= i+1 to 1 do
        if E[i][:] = E[j][:] then
           break
        end
    end
    if ( E[i][:] = E[j][:] ) then
        break
    end
    cardinality = non-zero elements of S[i]
end
```

3. A parallel algorithm for finding the metric dimension of a graph

In this section, we introduce a new parallel algorithm that determines the metric dimension of an arbitrary graph. The main idea in Algorithm 2 is that each processor generates $2^n/NPRS$ subsets, where *NPRS* is the number of processors and *n* is the order of graph *G*.

Algorithm 2: MPI Parallel algorithm for finding the metric dimension of a graph

```
Begin
1-/* All processors read the adjacency matrix A[ ] [ ] */
      for 1 ≤ i ≤ n do
            for 1 ≤ j ≤ do
                  Read A[i][j]
            end
      end
2-/*All processors initialize S[] and E[] [ ]*/
      for 1 ≤ i ≤ n do
            Set S[i]=0
      end
      for 1 ≤ j ≤ n do
            for 1 ≤ k ≤ n do
                  Set E[j][k]=0
            end
      end
3-/* Each processor generates q=2^n/NPRS subsets */
      for  0 ≤ i ≤ NPRS  do
            Each processor generates q subsets only.
      end
4-/* Each processor constructs E[][] for every S[][] */
     for i = n to 1 do
            for j = n to 1 do
                  if S[j] = 1 then
                        E[i][,j] = D[i][,j]
                  end
            end
     end
5-/* Each processor checks their subsets */
      for i = n to 1 do
                  for j= i+1 to 1 do
                        if E[i][:] = E[j][:] then
                              break
                        end
                  end
                   if ( E[i][:] = E[j][:] ) then
                              break
                   end
                   local_cardinality=non-zero elements of S[][]
      end

6-/* Each processor determines the local metric dimension */
  if (E[i][:] or E[j][:])and(metric_dimension>cardinality)then
      metric_dimension = cardinality;
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 November 2019 doi:10.20944/preprints201911.0194.v1

```
   end
7-/*Each processor sends local-MetricDimension to the processor 0 */
    Send Local-MetricDimension to the processor 0
8-/* Proc 0 receives the Local-MetricDimension from the all */
      ▪ Receive Local-MetricDimension from the all processors.
      ▪ Search the minimum value among Local-MetricDimension
        and assign in MetricDimension.
      ▪ Print MetricDimension
End /* Begin
```

4. Computational Results

We run the proposed algorithm implementation on a symmetric multi-processing (SMP) cluster using C programming language and the Message Passing Interface (MPI) library. We run the numerical experiments on the computer cluster of the Faculty of Science [17] at Cairo University. It is a homogeneous PC cluster that consists of 9 nodes, including one master and eight slaves. Each slave node has an Intel(R) core (TM)2 Duo CPU E7400 @ 2.80 GHz and 4 Gb of DDR2 RAM and the master node has an Intel(R) core(TM)2 Quad CPU Q6700 @ 2.66 GHz and 4 Gb of DDR2 RAM. The interconnection among the processors uses Fast Ethernet and the Scalable Coherent Interface (SCI).

We have run our implementation on four categories of graphs (the tracks in which the robot moves): a cycle graph C_n, a path graph P_n, a triangular snake graph Δ_k and ladder graph L_n.

From Table 1, we note that the order of the cycle graph starts with fifteen vertices because if we use a smaller problem size, we cannot get a good speedup because the communication time is greater than the computation time.

First, it is understandable that for 1 CPU, since a separate source code was developed, the communication time does not exist because the master does not send anything to a worker. It does all the computation work on its own. It had to be done using this pattern to get accurate results and the reason was that the same computer architecture should be used. If the code for 1 CPU was executed on machine with a newer CPU, the times should be quite different. As a result, this should be executed on the head of the cluster called the "master".

Looking at Table 4, comparing the times for 1 CPU and the times for 8 CPUs, we can assess the time differences and how many times faster the results are with parallelization. To be more specific, with 8 CPUs and the ladder graph with 20 vertices, we can achieve a speedup of 5.721 times ($time_{cpu[1]}$ / $time_{cpu[8]}$).

4. Conclusion:

We introduced a new parallel implementation for determining the metric dimension of a given graph. We run the proposed algorithm on a symmetric multi-processing (SMP) cluster using C programming language and the Message Passing Interface (MPI) library. Finally, we run our implementation on four categories of graphs (the tracks in which the robot moves): a cycle graph C_n, a path graph P_n, a triangular snake graph Δ_k and a ladder graph L_n. Preliminary computational results indicated that the metric dimension problem is an *NP-complete* problem and proved the ability of the proposed algorithm to achieve a speedup of 6 for 8 processors.

Table 1: Cycle graph C_n where $15 \le n \le 20$

n	*Cycle*										
	d	*md*	1 *Proc.*		2 *Proc.*		4 *Proc.*		8 *Proc.*		
			CPU	*Speedup*	*CPU*	*Speedup*	*CPU*	*speedup*	*CPU*	*speedup*
15	7	2	7.490	1	5.722	1.309	4.895	1.530	3.808	1.967
16	8	2	16.026	1	11.521	1.391	10.169	1.576	6.480	2.473
17	8	2	42.243	1	30.152	1.401	23.455	1.801	14.076	3.001
18	9	2	103.548	1	67.152	1.542	43.746	2.367	28.146	3.679
19	9	2	235.593	1	132.133	1.783	81.239	2.900	49.988	4.713
20	10	2	501.269	1	269.499	1.860	139.358	3.597	98.269	5.101

Table 2: Path graph P_n where $15 \le n \le 20$

n	*Path*									
	d	*md*	1 *Proc.*		2 *Proc.*		4 *Proc.*		8 *Proc.*	
			CPU	*speedup*	*CPU*	*speedup*	*CPU*	*speedup*	*CPU*	*speedup*
15	14	1	7.451	1	5.611	1.328	4.789	1.556	3.729	1.998
16	15	1	18.248	1	13.025	1.401	11.498	1.587	7.037	2.593
17	16	1	43.470	1	27.203	1.598	23.497	1.850	13.513	3.217
18	17	1	105.984	1	63.388	1.672	28.112	2.561	28.112	3.770
19	18	1	241.557	1	141.344	1.709	80.761	2.991	53.395	4.524
20	19	1	530.335	1	281.047	1.887	143.295	3.701	97.184	5.457

Table 3: Δ_k *-snakes graph* where $5 \le k \le 10$

k	Δ_k *-snakes graph*									
	d	***md***	**1 *Proc.***		**2 *Proc.***		**4 *Proc.***		**8 *Proc.***	
			CPU	***speedup***	***CPU***	***speedup***	***CPU***	***speedup***	***CPU***	***speedup***
5	5	2	0.207	1	0.1992	1.039	0.192	1.078	0.189	1.098
6	6	2	1.195	1	1.087	1.099	1.034	1.156	0.598	1.998
7	7	2	6.005	1	4.725	1.271	4.259	1.410	1.870	3.211
8	8	2	31.784	1	21.375	1.487	18.214	1.745	8.756	3.630
9	9	2	167.522	1	104.636	1.601	58.248	2.876	36.697	4.565
10	10	2	994.015	1	529.013	1.879	262.342	3.789	177.503	5.600

Table 4: *Ladder graph* L_n where $5 \le n \le 9$

k	***Ladder graph***									
	d	***md***	**1 *Proc.***		**2 *Proc.***		**4 *Proc.***		**8 *Proc.***	
			CPU	***speedup***	***CPU***	***speedup***	***CPU***	***speedup***	***CPU***	***speedup***
5	6	2	0.629	1	0.534	1.178	0.538	1.169	0.299	2.105
6	7	2	3.717	1	2.870	1.295	2.653	1.401	1.082	3.354
7	8	2	19.964	1	14.488	1.378	8.099	2.465	5.359	3.725
8	9	2	107.231	1	60.378	1.776	29.778	3.601	22.523	4.761
9	10	2	651.142	1	343.429	1.896	173.130	3.761	113.816	5.721

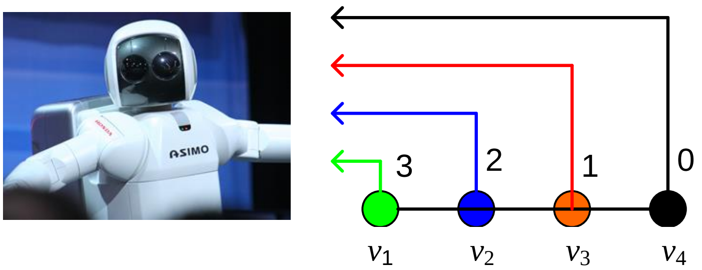

Figure 1: how the different signals determine the robot's location

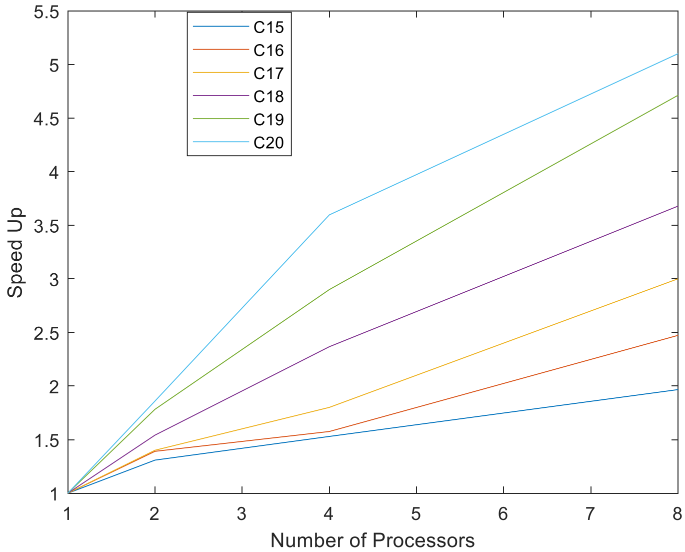

Figure 2: Algorithm 2 on a cycle graph with different sizes (C_{15}, C_{16}, C_{17}, C_{18}, C_{19}, and C_{20})

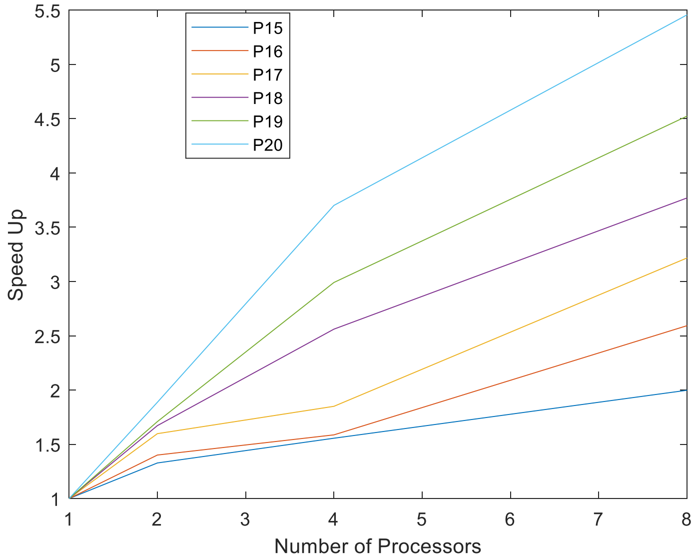

Figure 3: Algorithm 2 on a path graph with different sizes (P_{15}, P_{16}, P_{17}, P_{18}, P_{19}, and P_{20})

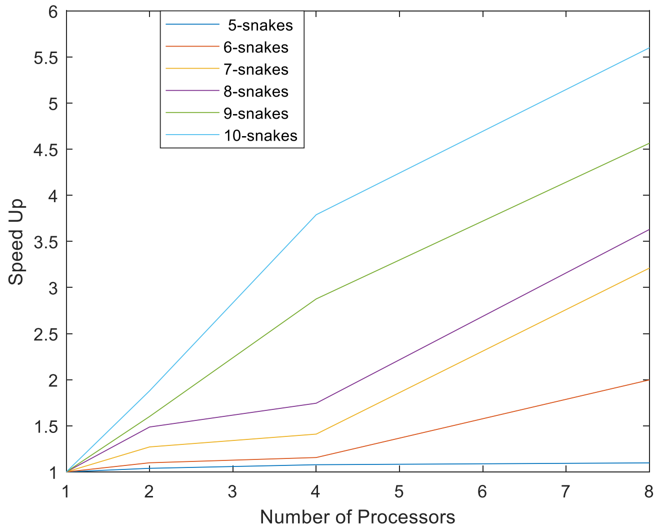

Figure 4: Algorithm 2 on a Δ_k *-snakes graph* with different sizes Δ_5, Δ_6, Δ_7, Δ_8, Δ_9, and Δ_{10}

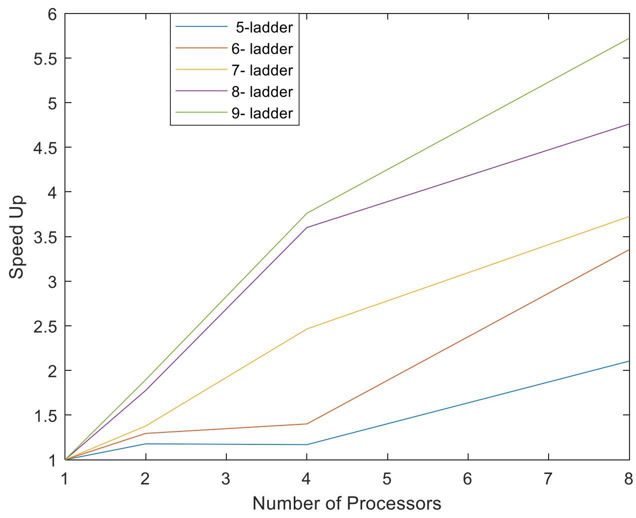

Figure 5: Algorithm 2 on a ladder graph with different sizes (L_5, L_6, L_7, L_8, and L_9)

References:

[1] S. Khuller, B. Raghavachari and A. Rosenfeld, *Landmarks in graphs*, Disc. Appl. Math. 70 (1996) 217–229.

[2] B. Shanmukha, B. Sooryanarayana and K. S. Harinath, *Metric dimension of wheels*, Far East J. Appl. Math. 8 (3) (2002) 217–229.

[3] G. Chartrand, D. Erwin, G. L. Johns and P. Zhang, *Boundary vertices in graphs*, Discrete Math. 263 (2003) 25-34.

[4] C. Poisson and P. Zhang, *The metric dimension of unicyclic graphs*, J. Comb. Math Comb. Comput. 40 (2002) 17–32.

[5] A. Seb¨o and E. Tannier, *On metric generators of graphs*, Math. Oper. Res. 29 (2) (2004) 383–393.

[6] Harary, F., Melter, R.A.: The metric dimension of a graph. Ars Combinatoria 2, 191–195 (1976)

[7] Slater, P.: Leaves of trees. Congressus Numerantium 14, 549–559 (1975)

[8] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, 1979.

[9] Josep Diaz, Olli Pottonen, Maria Serna, and Erik Jan van Leeuwen, On the Complexity of Metric Dimension, ESA 2012: 419-430

[10] E. M. Badr and K. Aloufi, Polynomial, Exponential and Approximate Algorithms for Metric Dimension Problem, submitted (2019) Heliyon Journal..

[11] E. M. Badr, A. Elrokh and B. Mohamed, On the metric dimension of Some Families of cyclic graphs, IT MUST Conference 29-30 April 2019, 6 October Giza, Egypt.

[12] Elsayed M. Badr, Mahmoud I. Moussa (2019), An upper bound of radio *k*-coloring problem and its integer linear programming model, March 2019, Wireless Networks

[13] G. Chartrand, L. Eroha, M. Johnson and O. Oellermann, Resolvability in graphs and the metric dimension of a graph. Discrete Applied Mathematics, 105:99-113, 2000.

[14] James Daniel Currie and Ortrud R. Oellermann, The Metric Dimension and Metric Independence of a Graph, Journal of Combinatorial Mathematics and Combinatorial Computing · January 2011.

[15] E.S. Badr, M. Moussa, K. Paparrizos, N. Samaras, and A. Sifaleras, Some computational results on MPI parallel implementation of dense simplex method, World Academy of Science, Engineering and Technology (WASET), 23, 2008,778–781.

[16] A. Rosa (1967), Cyclic steiner Triple Systems and Labelings of Triangular Cacti, Scientia, 5, 87-95.

[17] Sweilam NH, Moharram HM, Sameh Ahmed. On the parallel iterative finite difference algorithm for 2-D Poisson's equation with MPI cluster. In: The 8th international conference on INFOrmatics and systems (INFOS2012), IEEE Explorer; 2012.

[18] E. S. Badr, K. Paparrizos, N. Samaras, and A. Sifaleras (2005), On the Basis Inverse of the Exterior Point Simplex Algorithm, in Proc. of the 17th National Conference of Hellenic Operational Research Society (HELORS), 16-18 June, Rio, Greece, pp. 677-687.

[19] E. S. Badr, K. Paparrizos, N. Samaras, and A. Sifaleras (2005), On the Basis Inverse of the Exterior Point Simplex Algorithm, in Proc. of the 17th National Conference of Hellenic Operational Research Society (HELORS), 16-18 June, Rio, Greece, pp. 677-687.

[20] E.S. Badr, K. Paparrizos, Baloukas Thanasis and G. Varkas (2006), Some computational results on the efficiency of an exterior point algorithm, in Proc. of the

18th National Conference of Hellenic Operational Research Society (HELORS), 15-17 June, Rio, Greece, pp. 1103-1115