A Robot's Response Acceleration using the Metric Dimension Problem

E. M. Badr\textsuperscript{1} and Khalid Aloufi\textsuperscript{2}

\textsuperscript{1}Scientific Computing Department, Faculty of Computers & Artificial Intelligence, Benha University, Benha, Egypt. badrgraph@gmail.com
\textsuperscript{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 $\Delta_k$ 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.
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 \textit{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 \textit{locating number} or \textit{rigidity} problem and as \textit{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 \textit{NP-complete} problem. Recently, Josep Diaz \textit{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 $O(n^2 \times 2^n)$. 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 \textit{et al.} [11] introduced polynomial algorithms...
for special graphs, namely, mirror graphs, square graphs, Tortoise graphs, $Z(P_n)$ 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 $\Delta_k$ 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.
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 $\Delta_k$-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 $L_n = P_n \times K_2$, where $P_n$ is a path with $n$ vertices, $\times$ 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 $O(n^2 \times 2^n)$ [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
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**

1. for $i = 1$ to $n$ do
2.   for $j = 1$ to $n$ do
3.     $A(i,j) \leftarrow \infty$;  
4.     $A(i,i) \leftarrow 0$;
5.   end
6. end
7. $D \leftarrow A$
8. $\text{minn} \leftarrow \infty$;
9. for $k = 1$ to $n$ do
10.   for $i = 1$ to $n$ do
11.     for $j = 1$ to $n$ do
12.       $x \leftarrow D[i][j]$
13.       $y \leftarrow D[i][k] + D[k][j]$
14.       if $x < y$ then
15.         $\text{minn} \leftarrow x$;
16.       else
17.         $\text{minn} \leftarrow y$;
18.     end
19.   end
20. $D[i][j] \leftarrow \text{minn}$;
21. end
22. 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**

1. for $i = n$ to $1$ do
2.   $S[i] \leftarrow 0$
3. for $j = n$ to $1$ do
4.   $E[i][,j] \leftarrow 0$
The Get-Next-Subset procedure generates all subsets $S[i]$ of $V(G)$ using the binary counting representation method.

**Procedure 3: Get-Next-Subset**

1: $j \leftarrow n + 1$
2: repeat
3: $j \leftarrow j - 1$
4: until $(S[j] = 0)$ or $(j = 0)$
5: if $j \neq 0$ then
6: $S[j] \leftarrow 1$
7: $MAX \leftarrow j$
8: for $i = MAX + 1$ to $n$ do
9: $S[i] \leftarrow 0$
10: end
11: else
12: Done $\leftarrow$ True
13: 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**

1: for $i = 1$ to $n-1$ do
2: for $j= i+1$ to $n$ do
3: if $E[i][:] = E[j][:]$ then
4: break
5: end
6: end
7: if $(E[i][:] = E[j][:])$ then
8: break
9: end
10: cardinality = non-zero elements of $S[i]$
11: 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 $2^n/\text{NPRS}$ subsets, where $\text{NPRS}$ 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 $1 \leq i \leq n$ do
      for $1 \leq j \leq n$ do
         Read $A[i][j]$
      end
   end

2-/* All processors initialize $S[]$ and $E[] [ ] */
   for $1 \leq i \leq n$ do
      Set $S[i]=0$
   end
   for $1 \leq j \leq n$ do
      for $1 \leq k \leq n$ do
         Set $E[j][k]=0$
      end
   end

3-/* Each processor generates $q=2^n/\text{NPRS}$ subsets */
   for $0 \leq i \leq \text{NPRS}$ 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 $(E[i][] \text{ or } E[j][])$ and $\text{metric_dimension} > \text{cardinality}$ then
      metric_dimension = cardinality;
   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
$\Delta_k$ 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 $\Delta_k$ 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>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>7</td>
<td>2</td>
<td>7.490</td>
<td>1</td>
<td>5.722</td>
<td>1.309</td>
<td>4.895</td>
<td>1.530</td>
<td>3.808</td>
<td>1.967</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>2</td>
<td>42.243</td>
<td>1</td>
<td>30.152</td>
<td>1.401</td>
<td>23.455</td>
<td>1.801</td>
<td>14.076</td>
<td>3.001</td>
</tr>
<tr>
<td>18</td>
<td>9</td>
<td>2</td>
<td>103.548</td>
<td>1</td>
<td>67.152</td>
<td>1.542</td>
<td>43.746</td>
<td>2.367</td>
<td>28.146</td>
<td>3.679</td>
</tr>
<tr>
<td>19</td>
<td>9</td>
<td>2</td>
<td>235.593</td>
<td>1</td>
<td>132.133</td>
<td>1.783</td>
<td>81.239</td>
<td>2.900</td>
<td>49.988</td>
<td>4.713</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>2</td>
<td>501.269</td>
<td>1</td>
<td>269.499</td>
<td>1.860</td>
<td>139.358</td>
<td>3.597</td>
<td>98.269</td>
<td>5.101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>1</td>
<td>7.451</td>
<td>1</td>
<td>5.611</td>
<td>1.328</td>
<td>4.789</td>
<td>1.556</td>
<td>3.729</td>
<td>1.998</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>1</td>
<td>18.248</td>
<td>1</td>
<td>13.025</td>
<td>1.401</td>
<td>11.498</td>
<td>1.587</td>
<td>7.037</td>
<td>2.593</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>1</td>
<td>43.470</td>
<td>1</td>
<td>27.203</td>
<td>1.598</td>
<td>23.497</td>
<td>1.850</td>
<td>13.513</td>
<td>3.217</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>1</td>
<td>105.984</td>
<td>1</td>
<td>63.388</td>
<td>1.672</td>
<td>28.112</td>
<td>2.561</td>
<td>28.112</td>
<td>3.770</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>1</td>
<td>241.557</td>
<td>1</td>
<td>141.344</td>
<td>1.709</td>
<td>80.761</td>
<td>2.991</td>
<td>53.395</td>
<td>4.524</td>
</tr>
<tr>
<td>20</td>
<td>19</td>
<td>1</td>
<td>530.335</td>
<td>1</td>
<td>281.047</td>
<td>1.887</td>
<td>143.295</td>
<td>3.701</td>
<td>97.184</td>
<td>5.457</td>
</tr>
</tbody>
</table>
Table 3: $\Delta_k$-snakes graph where $5 \leq k \leq 10$

<table>
<thead>
<tr>
<th>$k$</th>
<th>$d$</th>
<th>$md$</th>
<th>1 Proc.</th>
<th>2 Proc.</th>
<th>4 Proc.</th>
<th>8 Proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU</td>
<td>speedup</td>
<td>CPU</td>
<td>speedup</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
<td>0.207</td>
<td>1</td>
<td>0.1992</td>
<td>1.039</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2</td>
<td>1.195</td>
<td>1</td>
<td>1.087</td>
<td>1.099</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>2</td>
<td>6.005</td>
<td>1</td>
<td>4.725</td>
<td>1.271</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2</td>
<td>31.784</td>
<td>1</td>
<td>21.375</td>
<td>1.487</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>2</td>
<td>167.522</td>
<td>1</td>
<td>104.636</td>
<td>1.601</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>2</td>
<td>994.015</td>
<td>1</td>
<td>529.013</td>
<td>1.879</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$k$</th>
<th>$d$</th>
<th>$md$</th>
<th>1 Proc.</th>
<th>2 Proc.</th>
<th>4 Proc.</th>
<th>8 Proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU</td>
<td>speedup</td>
<td>CPU</td>
<td>speedup</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>2</td>
<td>0.629</td>
<td>1</td>
<td>0.534</td>
<td>1.178</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>2</td>
<td>3.717</td>
<td>1</td>
<td>2.870</td>
<td>1.295</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>2</td>
<td>19.964</td>
<td>1</td>
<td>14.488</td>
<td>1.378</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>2</td>
<td>107.231</td>
<td>1</td>
<td>60.378</td>
<td>1.776</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>2</td>
<td>651.142</td>
<td>1</td>
<td>343.429</td>
<td>1.896</td>
</tr>
</tbody>
</table>

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 (P₁₅, P₁₆, P₁₇, P₁₈, P₁₉, and P₂₀)

Figure 4: Algorithm 2 on a Δₖ-snares graph with different sizes Δ₅, Δ₆, Δ₇, Δ₈, Δ₉, and Δ₁₀
Figure 5: Algorithm 2 on a ladder graph with different sizes (L_5, L_6, L_7, L_8, and L_9)

References:


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