PROJECT REPORT
VISUALLY OPTIMISED GRAPH DRAWING ALGORITHM AND SOFTWARE

JOHN C. MCCABE-DANSTED (00980994)

Undergraduate Mathematics Project Report
For Curtin University of Technology

Date: 15/02/2001.
1. Introduction and Background

1.1. subsection

1.1.1. subsubsection

Introduction

The primary goal of this project is to develop an algorithm for drawing graphs. We shall define a graph as a set of vertices V, and a set of edges E. No element can belong to both V and E, but every element in E is of the form (u,v), where u and v are members of V. It is traditional for mathematicians to draw graphs as dots on a page, and the edges as lines between them. There are a large number of ways that different types of graphs are drawn; most of these ways are very similar. For example, the only difference between a flowchart and a traditionally drawn graph, is that vertices are represented by ovals, rectangles, and diamonds rather than dots.

There are an increasing number of circumstances where one may wish to draw a graph. In computer science, it is expected that Entity-Relationship diagrams will be drawn when designing a database. When identifying bottle necks in a computer network, the network administrator may wish to draw a graph of the network.

There are a number of issues which must be considered to draw a graph effectively. It is particularly important the number of edges crossing other edges is reduced. Other criteria such as the symmetry and the length of the edges in the drawn graph should also be considered. Certain software packages do not even attempt to draw graphs effectively. Many Australian universities use cnet, a software package that emulates a computer network, to teach their students about
communication protocols. One of the networks it provides as example is the major network links of Australia (shown below)

In the diagram above, the positions of the vertices have been manually entered to reflect their geographic position. If the positions of the vertices are not set, cnet will simply draw the vertices in a straight line as shown below.

Cnet...

There is a considerable amount of existing research into drawing graphs. Force-directed and simulated annealing based algorithms have proven to be effective for drawing small graphs. Force-directed algorithms can result in few edge crossings, without special logic for this purpose[1]. The drawing of large graphs, in the order of 700 to 7000 vertices, is an entirely different problem to drawing small graphs.

Traditional optimality criteria are less important for large graphs. Even if the graph is readable, it may be impossible to comprehend[2]. If a large graph is drawn in its entirety, the viewer may comprehend only the structure of the graph[2]. Alternatively, less important vertices may be ghosted, hidden, or grouped[3]. The computational complexity of drawing large graphs also makes $O(n^3)$ force-directed algorithms unsuitable. Herman, Melancon and Marshall[2] instead recommend use of a $O(n \log n)$spanning-tree algorithm for laying out large graphs. Spanning-tree algorithms involve first finding a spanning tree of the graph, and then laying out the graph in a tree structure. We will only consider the problem of laying out small graphs.

Although it is not necessary for an undergraduate project to be original, a little bit of originality can make the project more rewarding for the people involved. Due to the amount of existing research, we decided that it would be necessary to put a practical slant on the project. The project would involve implementing an algorithm as a computer program; suggesting and developing practical uses for the program. The algorithm was also developed independently from prior research. After the algorithm was developed, it was compared to prior solutions.

There are a number of software packages that are able to draw graphs effectively. For example, WWWPal is designed to help in the analysis and synthesis of Web documents[4][5]. It offers a number of different algorithms to draw the graphs, such as radial, spring loaded, and incremental[5]. It is designed to be capable of drawing huge graphs, even up to 200,000 vertices[6][5].

TSGraph[7] is another program capable of formatting graphs. It only provides the integer vector method. This method is unable to layout certain types of graphs effectively. For example the (x,y) coordinates of a standard pentagon are inherently irrational, and thus cannot be placed on an integer plane.
2. **Initial Prototype: Simple Force Directed Algorithm**

2.1. **Discussion of Rule based Algorithm.** When developing the algorithm, the possibility of laying out the graph according to certain rules was first considered. For example, drawing all the vertices on the largest cycle in the graph as a standard polygon, as shown in G1 and G2.

![G1](image1.png) ![G2](image2.png)

G1 looks better than G2, so perhaps the remaining vertices should be placed outside the cycle. However, as shown in G3 and G4, it may be more appropriate to place a vertex within the cycle. But what if vertex is not a cut edge? now we should put vertices on inside of cycle.

![G1](image3.png) ![G2](image4.png)

This opens up the question of how many rules and exceptions we would need. To handle all possible graphs appropriately we may need many rules. It would be difficult to test that all of
these rules work together appropriately. Where two rules may apply to the same graph, it must be decided which rule takes precedence. Clearly a more simple model would be desirable.

2.2. Discussion of Physical Model. One approach to any problem, is to see how such a problem is handled in nature. A Chemical compound can be thought of as a graph, with atoms represented by vertices, and bonds represented as edges. Chemicals also have a physical structure, which can be modeled as balls joined by flexible sticks.

The structure of a chemical compound has certain desirable aesthetic properties. No two atoms can be on the same spot, there must be some distance between the nuclei. Furthermore, a bond between two atoms will resist being stretched, preventing unnecessarily long bonds. This suggests that a physical model may be used to produce an acceptable drawing of a graph.

With a physical model, forces are used in place of rules. In any physical system, if two forces apply they are simply added. No exceptions need to apply. We want unrelated vertices to be as far away from each other as possible. Very long or short edges are also undesirable. Thus, we also want edges to approximate a standard length, for example one unit. To this end, four forces were considered, stretching, compression, repulsion, and air resistance.

<table>
<thead>
<tr>
<th>Force</th>
<th>Description</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stretching</td>
<td>Pulls any two linked vertices closer together</td>
<td>Ensure that edges are not unnecessarily long</td>
</tr>
<tr>
<td>Compression</td>
<td>Pushes two linked vertices apart</td>
<td>Ensure that edges are not too short</td>
</tr>
<tr>
<td>Repulsion</td>
<td>Each vertex pushes every other vertex away</td>
<td>Ensure that two vertices are not too close together</td>
</tr>
<tr>
<td>Resistance</td>
<td>Gradually slow down vertices</td>
<td>Ensure that system reaches an equilibrium point</td>
</tr>
</tbody>
</table>

This model can be further simplified. The repulsion force between all vertices can be used to ensure that edges are not too short. An object falling though an atmosphere will reach a terminal velocity. If we assume that resistance dominates inertia, this velocity will be reached immediately so the system will effectively be inertia-less. Where \( T \) is the total force excluding air resistance, \( R \) is the force of air resistance and \( v \) is the velocity.

In standard physics the formula \( F = m \times a \) applies. \( F = T + R \) so \( T + R = m \times a \). \( R \gg m \), so we can simplify this to \( T + R = 0 \). Let the resistance force be defined as \( R = -v \), where \( v \) is the velocity of the vertex. Hence \( T + (-v) = 0 \), so \( v = T \).

Lemma 2.1. \( v = T \)

To calculate \( T \) and thus \( v \), only the stretching and compression forces need to be considered. Hence to simulate the physical system only these forces need to be considered, as summarised in the diagram below.
These forces attempt to assure that edges are of length approximately equal to one and unrelated vertices are far apart.

2.3. Implementation of Physical Model. For this section, let us define \( P(i, t) \) as ‘The position of vertex i at time t’. Then define \( E \) as the adjacency matrix of the graph.

\[
E[i, j] = \begin{cases} 
1 & \text{if vertices } i \text{ and } j \text{ are adjacent} \\
0 & \text{if vertices } i \text{ and } j \text{ are not adjacent} 
\end{cases}
\]

The distance \( d \) between two vertices \( i \) and \( j \) is equal to \( \frac{P(i, t) - P(j, t)}{|P(i, t) - P(j, t)|} \). The total attraction force \( T(i, j) \) between two vertices is the attraction \( E[i, j]d \) minus the repulsion \( \frac{1}{d} \).

**Lemma 2.2.** \( T(i, j) = E[i, j]d - \frac{1}{d} \).

The total attraction force between two vertices is

In the previous section we established that in our physical model we would use a Repulsion force and an Attraction force. The direction of the force \( T(i, j) \) on the vertex \( j \) is \( \frac{P(i, t) - P(j, t)}{|P(i, t) - P(j, t)|} \). Hence for the total directed force \( T \) on a single vertex \( j \), from all vertices \( 0, 1, \ldots, n \):

\[
T = \sum_{j=0}^{n} \frac{P(i, t) - P(j, t)}{|P(i, t) - P(j, t)|} \left( E[i, j]d - \frac{1}{d} \right)
\]

from lemma 2.1 on the preceding page. \( v = T \). Also, in any physical system the change in position over time is the velocity, i.e. \( \frac{dP(j, t)}{dt} = v(j, t) \). Thus the following system of ODEs (Ordinary Differential Equations) results.

\[
\forall j \in 0..n \quad \frac{dP(j, t)}{dt} = \sum_{j=0}^{n} \frac{P(i, t) - P(j, t)}{|P(i, t) - P(j, t)|} \left( E[i, j]d - \frac{1}{d} \right)
\]

The software then uses Euler’s method to solve these Equations. In general, Euler’s method is a very poor method for solving ODEs\([4, 708]\). However, with a small step size the Euler method provided sufficient accuracy and reasonable performance on Pentium computers.

These ODEs form an initial value problem. The initial positions are obtained simply generating a random starting position for each vertex.
2.4. **Energy in the Physical Model.** The resistance force is constantly reducing the energy remaining in the physical system. *Energy = Force * Distance*, hence the potential energy $e$ due to a force is:

$$e = \int_b^d F(x) dx$$

where $d$ is the distance between the object and the source of the force, $F(x)$ is the attraction force the source exerts on the object at distance $x$, and $b$ is the base distance from the object which is defined to have zero energy. From lemma 2.2 on the preceding page we know that the total attraction force between two vertices is $E[i,j]d - \frac{1}{d}$, let us also define the base distance $b$ to be 1. Hence the energy equation becomes

$$e = \int_b^d E[i,j]x - \frac{1}{x}dx = \left[E[i,j]\frac{1}{2}x^2 - \ln x\right]_b^d$$

It would be intuitive to believe that there is some ‘true’ value for the base distance $b$ which would measure that absolute amount of energy in the system. In fact no such values exist. If either $0$ or $\infty$ was chosen as such a value $\ln x$ would become undefined. Thus 1 is the simplest value for $b$ that gives a useful result.

As the time $t$ increases, the total amount of energy decreases. This is intuitive as the resistance force drains energy from the system, and the system has no energy source. Due to the number of variables, a rigorous proof of this could be complex. A simple demonstration follows.

Note that

$$\frac{de}{dt} = \frac{dd}{dt} \int_b^d E[i,j]x - \frac{1}{x}dx$$

$$= E[i,j]d - \frac{1}{d}$$

$$= T(i,j)$$

Lemma 2.1 states that $v = T(i,j)$. As $T(i,j)$ is the total attraction force, $\frac{dd}{dt} = -v = -T(i,j)$.

$$\frac{de}{dt} = \frac{dd}{dt} \frac{de}{dd} \text{ by the chain rule}$$

$$= T(i,j).(-T(i,j))$$

$$= v.(-v)$$

$$= -v^2$$

Thus if $v \neq 0$, $\frac{dv}{dt} < 0$. In other words, the system continues to lose energy until it stops moving. Where $E[i,j] = 1$, the global minimum for the energy equation $E[i,j]d - \frac{1}{d}$ is 1. Thus, in a low energy system the edge length would be approximately one unit. Where $E[i,j] = 0$, the global minimum for the energy equation $E[i,j]d - \frac{1}{d}$ is $\infty$. In a low energy system, unrelated vertices would be far apart. This meets the aims of the physical system as described in section 2.2.

This algorithm can also be thought of as solving an optimisation problem. As discussed in the previous paragraph, a physical representation of the graph with low energy has desirable properties. The energy in the system continually decreases with time, approaching a local minima.
3. **Vertex Swapping**

Frick, Ludwig and Mehldau[1] suggested that force directed algorithms can reduce the number of crossovers in a drawing without any special logic specific to this purpose. Our force-directed algorithm minimises the energy of the physical system. As will be discussed latter, a low energy system will have few crossovers. However the system can get trapped in at a local minimum. In some cases, switching two vertices can escape this local minimum.

### 3.1. Reduction in cross-overs due to reduction of energy in system.

Where two edges crossover two paths $P_1$ and $P_2$ can be defined as shown above. The total length of the edges is equal to the length of $P_1$ and $P_2$. Switching the positions of two vertices will result in a graph with no crossover, and the two new paths $P_1'$ and $P_2'$. The total length of $P_1'$ and $P_2'$ clearly adds up to the total length of the edges in $G_2$. $P_1'$ and $P_2'$ have the same beginning and end points as $P_1$ and $P_2$ respectively. However $P_1'$ and $P_2'$ are straight, and by the definition of a straight line are thus shorter than $P_1$ and $P_2$.

**Lemma 3.1.** The total edge length of $G_2$ is less than that of $G_1$.

A longer edge length will generally mean more energy. Note that the positions occupied by the vertices $v_1$, $v_2$, $v_3$ and $v_4$ in $G_1$, are the same as the positions as the vertices $v_1$, $v_4$, $v_3$ and $v_2$ in $G_2$. Since the repulsion force does not distinguish between vertices, this means that the total potential energy due to the repulsion force is the same in $G_1$ and $G_2$. Due to lemma 3.1, we would expect the total potential energy due to the tension in the edges to be lower than that of $G_1$. Thus, in general, a system with less energy will have fewer total crossovers.

### 3.2. Local Minima.

It is possible for the system to become trapped at a local minimum. To see how consider the following diagram.
As discussed in the previous section, a crossover increases the energy of the system. If the system could escape from the configuration of G1 this would reduce the energy of the system. However, for this to happen one vertex, such as v1, would have to pass through the other edge. As shown in G2, v1 must come closer to v2 and v4 to do so. This increases the repulsion force, and forces v1 back to its position in G1.

3.3. How vertex swapping algorithm eliminates crossovers. The vertex swapping algorithm (below) cannot remove all crossovers from the graph. Some graphs are simply not planar, in other cases the algorithm is simply not able to escape the local minimum. However we will demonstrate that it works for the case of two edges.

For each pair of vertices \((v, u)\)
  
  Swap position of \(v\) and \(u\).
  
  If total edge length does not decrease
  Swap the positions of \(v\) and \(u\) back to their original positions.
  End If
End For

Consider any two edges in a graph. Swapping the position of two vertices in the graph could have one of three consequences; remove a crossover between the edges; cause a crossover between the edges; neither cause nor remove a crossover between the edges.

Remove a crossover between two edges. (Case 1) From lemma 3.1, switching these two vertices reduces the total length of these two edges.

Cause a crossover between the edges. (Case 2) From lemma 3.1, switching these two vertices back would reduce the total length of these two edges. Hence, swapping them in the first place increases the length.

Neither cause nor remove a crossover between the edges. (Case 3) It does not matter whether these two vertices are swapped, it has no effect on whether there is a crossover or not.

From considering cases 1, 2 and 3, we can state the following two lemmas.

Lemma 3.2. If swapping two vertices eliminates a crossover it reduces the total length of the two edges.

Lemma 3.3. If swapping two vertices reduces the total edge length of the two edges, it does not increase cause a crossover.

The vertex swapping algorithm attempts to swap every two vertices. If the edges are crossed over, then in at least one case, swapping the vertices will uncross the edges. Lemma 3.2 shows
that this will reduce the total edge length, so the vertices will not be swapped back. Thus if the
edges are crossed the algorithm will uncross them.

If two edges cross, it is possible to switch two vertices in such a way that these two edges
no longer cross-over. From lemma 3.1, switching these two vertices reduces the total length of
these two edges.

If the two edges do not cross, the algorithm will not cause them to cross. The algorithm will
swap the edges back unless the swap reduces the total edge length. From lemma 3.3, if the swap
reduced the total edge length it will not cause a crossover.

Hence the algorithm can ensure that no cross-over exists in a graph with two edges.

4. Utility of Software

4.1. Basic Use of Software. The core program developed for this project is the GraphApplet.
This applet is embedded in a number of sample html documents. The applet can be run simply
by opening these documents in a Java enabled browser. The most direct way to run this applet
is to type: appletviewer <htmlfile>.html on a system that has the JDK (Java Development Kit)
installed. Remember that the CLASSPATH environment variable must include the GraphApplet
directory. The layouts that the applet produces can then be captured by typing appletviewer
<htmlfile>.html > output.layout. Note that some versions of Java have difficulty displaying the
animation in real time.

On a system without the JDK installed, the PlaceVertices class can be used to produce layouts.
As a simple example a triangle can be described as the edges (0,1) (1,2) (0,2). The PlaceVertices
class can be run by typing Java PlaceVertices. To enter the edges (0,1) (1,2) (0,2), type 0 1 1 2
0 2. The program will then output the positions of vertices 0, 1, 2 3 in the form

\[
x_0 \ x_1 \ x_2 \ldots \\
y_0 \ y_1 \ y_2 \ldots
\]

4.2. Suggest Alternative Layouts. As the algorithm randomly decides the starting positions
of the vertices, the algorithm is non-deterministic. On different occasions it can layout the exact
same graph different ways. This could be an advantage, as it could provide the user with a
number of different alternative layouts. A human may instinctively choose one way of drawing
a graph, and ignore other possible ways of drawing the same graph. These alternate methods
of drawing the graph may allow the human to see a problem in a different way, or may even be
better in some sense that the layout the human would have drawn. For example, consider the
three drawings of the Peterson graph below.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G1: 5 crossovers</td>
<td>G2: 3 crossovers</td>
<td>G3: 6 crossovers</td>
</tr>
</tbody>
</table>
A human will almost always draw the Peterson graph as the first graph, with 5 crossovers. This layout has the advantage that it is easy to remember, and a viewer will be immediately able to identify the graph as the Peterson graph. However, as discussed in the introduction, Purchase [5] suggests that the most important criterion for drawing a graph is the number of crossovers. This may suggest that the layout G2 is superior. On the other hand G3 allows the viewer to easily identify the largest cycle in the graph, which may be important in certain contexts.

One possible use for this project is to produce a number of different layout of graphs and provide them to the user so that they can pick the graph most suitable to their needs. To realise this possibility a EliminateDuplicates class has been developed. The user can run the GraphApplet for some while, perhaps overnight. The output can be piped through the EliminateDuplicates class to eliminate all duplicated classes, leaving only a handful of unique layouts for the graph. For example, it is possible to record only the unique graphs produced by GraphApplet, by typing `appletviewer <htmlfile>.html | java EliminateDuplicates > uniques.layout`.

This system works by keeping statistics on the longest, shortest and average lengths of edges in the layout. A table of these statistics for different layouts of the Peterson graph is provided in section 6.1 on the next page. From this table it can be seen that for the same layout these statistics are accurate to about four significant figures. Different layouts seem to differ by more than one percent. The algorithm for EliminateDuplicates assumes that the layouts are identical if these three statistics are equal to within one percent.

Simply comparing the vertex positions between the graphs would not be sufficient to determine whether the layouts are the same. A ten degree rotation does not change the the structure of the layout, but it does significantly change the positions of the vertices.

4.3. Automatic Layout of Graphs for \LaTeX. \LaTeX is a popular tool for producing mathematical documents. \LaTeX's philosophy is to separate the content of the document from the form. The author of the content decides what the text and headings of the document are. However the author does not decide how the document is laid out, for example the size and font used for the headings.

\LaTeX is only capable of formatting documents. To place a graph in a \LaTeX document, the graph must first be created in another application. If the author uses a drawing tool to create the graph they must concern themselves with the layout of the graph as well as the actual graph itself.

This program could be used to construct the layout of the graph, extending \LaTeX's philosophy to graphs. To realise this potential a utility has been developed as part of this project to export the graphs as EPS (Encapsulated Post Script) files. These files can then be imported into \LaTeX.

To create the EPS files from an exists list of layouts, input.layouts, type `/graphlist2eps input.layouts`. This will extract the unique layouts contained in the input.layouts file and output them as OUTPUT_input.layouts.0.eps, OUTPUT_input.layouts.1.eps, OUTPUT_input.layouts.2.eps, ... . On UNIX systems the EPS files can be viewed through the `gv` command.

Generally an author may wish to maintain control over the form as well as the content of the graphs in their documents. However, this feature has been useful in producing this report. Many of the graphs in this document, such as the graphs in section 4.2 on the facing page, are EPS files produced by this project. As discussed in that section it would be possible for the author choose the layout of the graph from a number of alternatives, even though they did not need consider the layout the graph when they first inputted it.

4.4. Web Toy. As the physical system simulation component of this project was written as a Java applet, it is easy to embed it in web documents. The GraphApplet is only several kilobytes in size, and will load within a few seconds with a modern modem. Any Java enabled web browser can display the simulation simply by opening the web page. For an example of this,
there are animated simulations in the slides for the presentation on this project, available at www.cs.curtin.edu.au/~mccabedj.

4.5. Efficient Graph Description Format. Some of the static graphs in the slides are also displayed using the Graph Applet. The HTML (Hyper Text Markup Language) does not have any special features for displaying diagrams or graphs. Typically when a diagram is displayed on the web, it is embedded in a HTML document as a image file. Saving a graph as a compressed image file may take about two and a half kilobytes, depending on the size and complexity of the graph. An uncompressed applet tag that describes the same graph may take only half a kilobyte or less. If a web site was to display a number of graphs, it would require a lower total bandwidth to display a graphs if they were embedded as a Java applet tag rather than as image files.

5. Further Research

6. Appendixes

6.1. Edge Length Tables. Tables describing the shortest, longest, and average edge length for each layout generated. The highlighted column table indicates which column the table is sorted by.
<table>
<thead>
<tr>
<th>Min length</th>
<th>Avg Length</th>
<th>MaxLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9796702</td>
<td>1.6648399</td>
<td>2.6954589</td>
</tr>
<tr>
<td>0.8402851</td>
<td>1.6255572</td>
<td>2.8654544</td>
</tr>
<tr>
<td>0.8402963</td>
<td>1.6255474</td>
<td>2.8657024</td>
</tr>
<tr>
<td>0.8403011</td>
<td>1.62555</td>
<td>2.8650365</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023778</td>
<td>2.7652979</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416172</td>
<td>1.6023884</td>
<td>2.7652664</td>
</tr>
<tr>
<td>0.8402851</td>
<td>1.6255572</td>
<td>2.8654544</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.8402963</td>
<td>1.6255474</td>
<td>2.8657024</td>
</tr>
<tr>
<td>0.8403011</td>
<td>1.62555</td>
<td>2.8650365</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023778</td>
<td>2.7652979</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416172</td>
<td>1.6023884</td>
<td>2.7652664</td>
</tr>
<tr>
<td>0.8402851</td>
<td>1.6255572</td>
<td>2.8654544</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.8402963</td>
<td>1.6255474</td>
<td>2.8657024</td>
</tr>
<tr>
<td>0.8403011</td>
<td>1.62555</td>
<td>2.8650365</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023778</td>
<td>2.7652979</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416172</td>
<td>1.6023884</td>
<td>2.7652664</td>
</tr>
<tr>
<td>0.8402851</td>
<td>1.6255572</td>
<td>2.8654544</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.8402963</td>
<td>1.6255474</td>
<td>2.8657024</td>
</tr>
<tr>
<td>0.8403011</td>
<td>1.62555</td>
<td>2.8650365</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023778</td>
<td>2.7652979</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416172</td>
<td>1.6023884</td>
<td>2.7652664</td>
</tr>
<tr>
<td>0.8402851</td>
<td>1.6255572</td>
<td>2.8654544</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.8402963</td>
<td>1.6255474</td>
<td>2.8657024</td>
</tr>
<tr>
<td>0.8403011</td>
<td>1.62555</td>
<td>2.8650365</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023778</td>
<td>2.7652979</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416172</td>
<td>1.6023884</td>
<td>2.7652664</td>
</tr>
<tr>
<td>0.8402851</td>
<td>1.6255572</td>
<td>2.8654544</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.8402963</td>
<td>1.6255474</td>
<td>2.8657024</td>
</tr>
<tr>
<td>0.8403011</td>
<td>1.62555</td>
<td>2.8650365</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023778</td>
<td>2.7652979</td>
</tr>
<tr>
<td>0.9416147</td>
<td>1.6023778</td>
<td>2.7652945</td>
</tr>
<tr>
<td>0.9416126</td>
<td>1.6023856</td>
<td>2.7652366</td>
</tr>
<tr>
<td>0.9416172</td>
<td>1.6023884</td>
<td>2.7652664</td>
</tr>
</tbody>
</table>
REFERENCES


