Great Deluge Algorithm for the Linear Ordering Problem: The Case of Tanzanian Input-Output Table

Amos Mathias  
Department of Science Mathematics and Technology Education, University of Dodoma, Box 523, Dodoma, Tanzania  
Email: ms_mathias@yahoo.co.uk

Allen R. Mushi  
Department of Mathematics, University of Dar es salaam, Box 35062, DSM, Tanzania  
Email: allenmushi66@gmail.com

Abstract—Given a weighted complete digraph, the Linear Ordering Problem (LOP) consists of finding and acyclic tournament with maximum weight. It is sometimes referred to as triangulation problem or permutation problem depending on the context of its application. This study introduces an algorithm for LOP and applied for triangulation of Tanzanian Input-Output tables. The algorithm development process uses Great Deluge heuristic method. It is implemented using C++ programming language and tested on a personal computer with 2.40GHZ speed processor. The algorithm has been able to triangulate the Tanzanian input-output tables of size 79x79 within a reasonable time (1.17 seconds). It has been able to order the corresponding economic sectors in the linear order, with upper triangle weight increased from 585,481 to 839,842 giving the degree of linearity of 94.3%.

Index Terms — Optimization, Linear Ordering, Input-Output Tables, Great Deluge Algorithm

I. INTRODUCTION

Linear Ordering Problem (LOP) involves finding an acyclic tournament in a complete weighted digraph with maximum weight. This is equivalent to finding a permutation of rows and columns (linear order) of the associated matrix such that the sum of the weights of the upper triangle is maximized, Mushi [1]. It is one of the combinatorial optimization problems that are classified as NP-hard (Non-deterministic Polynomial time) problems as discussed in Marti and Reinelt [2]. It is so classified because there is no general deterministic polynomial time algorithm for its solution that is known to date, Schiavinotto and Stützle [3].

Formally, given a complete digraph \( D_n = (V_n, A_n) \) with \( n \) nodes and arc weights \( x_{ij} \), for all \( i, j \in V_n \) (nodes set) and \( (i, j) \in A_n \) (arcs set) defined on a positive weight function \( x: A \mapsto \mathbb{R}_+ \), find an acyclic tournament \( T \subseteq A \) such that the weight \( x(T) = \sum_{(i,j)\in T} x_{ij} \) is the maximum possible [2].

The ideas on the solution of LOP existed since 1958, from the work done by Chenery and Watanable [4]. Since then, it has received considerable attention by many researchers and hence becoming a problem of interest to date. LOP is found in a number of applications, such as Triangulation of Input-Output matrices in economics, Archaeological seriation, Minimizing total weighted completion time in one-machine scheduling, Aggregation of Individual preferences, Grötschel, Junger and Reinelt [5], ordering of teams in sports tournaments, Mushi [1] as well as Machine translation, Tromble [6].

A number of algorithms for the LOP solution have been developed, however each algorithm works with some weaknesses depending on the level of complexity of the problem. Grötschel, Junger and Reinelt [5], introduced an exact algorithm which combines heuristics, cutting plane and branch and bound techniques basing on investigations of the LOP polytope. The algorithm was able to solve up to 60x60 Input-Output tables from the Linear Ordering Library (LOLIB) instances [7], with 83.185% degree of linearity and running time of 13 minutes and 25 seconds. Likewise, an exact algorithm focused on cutting plane and branch and bound procedures has been used to LOP in Mushi [1]. The approach relaxed integer constraints and solved the problem as a continuous Linear Programming problem with normal simplex algorithm and then applying cutting planes with available facets followed by branch and bound technique to get integral solution. The algorithm was able to solve the 41x41 Irish input-output table to optimality within reasonable time.

As pointed earlier, this is an NP-Hard problem and therefore no optimal algorithm is known that can solve a general problem within reasonable time. Consequently, a lot of effort has been devoted into finding good heuristic solutions. Although heuristics cannot give a guarantee of optimal solutions, they have been widely used to give good solutions within reasonable time. Marti and Reinelt [2] reports on the development of a heuristic, commonly referred to as Beckers method, based on calculations of quotients which were used as basis for ranking the economic sectors of an input-output table. However, the authors admit that their algorithm does not necessarily lead to good approximate solution to their triangulation problem.
Chanas and Koblański [8] designed a heuristic algorithm which could determine the solution for both maximization and minimization of the given problem criterion function, basing on the resulting permutation by sorting through insertion and permutation reversals. Experimental results show that the algorithm was able to give near optimal solutions in most instances. However, the method requires the problem to be decomposable into components, where methods for effective decomposition are difficult to find.

Garcia et al [9] developed a Variable Neighbourhood Search (VNS) algorithm. The algorithm combines various neighbourhoods for effective exploration of search space. The VNS results competed with well known heuristics such as Tabu Search but require further extensive studies. Campos, et al [10] presented meta-heuristic approaches for LOP, using scatter search technique. The approach used combined solution vectors that had proved effective in a number of problem settings, but was able to match only 30 out of 49 solutions of world problem instances in the LOLIB.

Recent work includes a publication by Celso and Matsunori [11] on local algorithms. They provided improvements into the local search by introducing two algorithms namely LIST and TREE for neighbourhood structure. Computational experiments with random problems showed good results for sparse instances with density up to 10%. Even more recent is the 2014 publication by Tao et al [12], on Multi-Parent Memetic algorithm, denoted by MPM. The algorithm incorporates particular features such as multi-parent recombination operator, a diversity based parent selection strategy and a quality-and-diversity based pool updating strategy. Computation results shows that the MPM is an efficient algorithm and outperforms previous Memetic algorithm in detecting lower bounds for challenging instances.

This paper introduces another heuristic algorithm for the LOP particularly for a triangulation problem using Great Deluge Algorithm (GDA) described in Doeck [13]. We are interested in the use of GDA particularly for Tanzanian Input-Output economic tables since the technique has been effective in other combinatorial optimization problems, including timetabling, Mush [14], [15], Landa-Silva and Obit [16], and Turabie, Abdullah and McCollum [17]. Other successful applications include Dynamic Scheduling on Grid environment problem reported in McMullan and McCollum, [18].

The paper is organized as follows; first we give a mathematical formulation of the problem; secondly we present the Great Deluge Algorithm with the specific adaptation to the Linear Ordering Problem, including development of initial solution, selection of moves and neighbourhood structure. We then provide our analysis of results as applied to the Tanzanian Input-Output tables; and lastly we present conclusion and suggest areas for further research.

II. MATHEMATICAL FORMULATION

The LOP is formulated as a binary Integer Programming problem. The input-output table decision variable is defined as follows:

Let

\[ x_{ij} = \begin{cases} 1 & \text{if there is an arc } (i, j) \text{ between nodes } i \text{ and } j \\ 0 & \text{Otherwise} \end{cases} \quad \forall (i, j) \in A_n \]

Thus, the Linear Programming model as defined by Marti and Reinelt [2] is:

Maximize \[ \sum_{(i, j) \in A_n} w_{ij} x_{ij} \] (1)

Subject to:

\[ x_{ij} + x_{ji} = 1, \forall i, j \in V_n \]
\[ x_{ij} + x_{ik} + x_{kj} \leq 2, \forall i, j, k \in V_n, j \neq k \]
\[ x_{ij} \in \{0,1\}, \forall i, j \in V_n \]

Where \( w_{ij} \) represents the actual weights from the input-output table.

In Graphical representation, we define a digraph \( D_n := (V_n, A_n) \) and find an acyclic tournament \( T \subseteq A_n \) with the linear ordering \( <V_1, V_2, ..., V_m> \) that gives the maximum sum of weights assigned to its corresponding arcs [2].

The most common exact approach in solving LOP is the use of a branch and cut algorithm that combines both cutting planes and branch and bound methods. This is facilitated by the development of deepest possible cutting planes known as facets that can prune infeasibilities from the relaxation of the associated linear programming problem. The known facets include the 3-dicycles, 3-fences, Möbius ladders of type \( M \) and type \( M' \) [2]. These facets together with the minimal equation system give the following Linear Programming relaxation:

Maximize \[ \sum_{i,j} C_{ij} y_{ij} \] (2)

Subject to:

\[ x_{ij} + x_{ji} = 1, \text{ for all } 1 \leq i \leq j \leq n, \]
\[ x_{ij} \geq 0, \text{ for all } 1 \leq i, j \leq n, i \neq j, \]
\[ \alpha(C) \leq 2, \text{ for all } 3 \text{-dicycles } C \in A_n, \]
\[ \alpha(F) \leq 7, \text{ for all } 3 \text{-Fences } D = (V, F) \text{ in } D_n, \]
\[ \alpha(M) \leq 8, \text{ for all Möbius ladders } D = (V, M) \text{ of type } M_2 \text{ or } M_2' \text{ in } D_n \]

It has been shown in [2] that this problem has \( \binom{n}{2} \) equations, \( n(n-1) \) non-negativity constraints, \( \binom{n}{3} \) 3-
dicycle inequalities, \(120\binom{n}{6}\) 3-fence inequalities, and
\(360\binom{n}{6}\) Möbius ladder inequalities.

Due to this enormous number of constraints, it is impractical to list all the constraints and solve the linear program using available computer code. Instead, a cutting plane algorithm is applied where facets are added to the problem one at a time until the solution is found or the problem size has been reduced to an extent that it can be solved by branch and bound method.

However, this approach has been only successful in solving specific instances to optimality due to large size of the problem. The use of heuristics is therefore a preferred approach when it comes to large problem sizes and hence the choice of Great Deluge heuristic approach in this work.

A. Linear Ordering as a Triangulation Problem

As described by Marti et al in [7], given an \((n, n)\) matrix \(C = (c_{ij})\), the triangulation problem involves the determination of a simultaneous permutation of the rows and columns of the matrix \(C\) such that the sum of upper-diagonal entries is as large as possible.

By setting arc weights \(w_{ij} = c_{ij}\) for the complete digraph \(D_n\), the triangulation problem for \(C\) can be solved as Linear Ordering Problem for \(D_n\). Conversely, a linear ordering problem for \(D_n\) can be transformed into a triangulation problem for an \((n, n)\) matrix \(C = (c_{ij})\) by setting \(c_{ij} = w_{ij}\) and the diagonal entries \(c_{ii} = 0\).

Example

Consider the following directed compete graph with weights on arcs (Fig. 1):

The associated matrix is as shown in Table 1:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

Sum of upper weight = 2+3+1 = 6

An acyclic tournament with the highest weight in this simple example is achieved by picking arcs with highest weight which does not violate the acyclic condition and covers all arcs. The graph below gives the best solution (Fig. 2):

![Fig. 2. Acyclic tournament with highest weight example](image)

Ordering procedure

The first node is the one with no entering arc (i.e. 3); the last node is the one with no leaving arc (i.e. 1). The order is therefore: 3, 2, 1 and the associated matrix is as shown in Table 2:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Sum of upper weight = 9 + 8 + 7 = 24

Our case study problem is the input-output table for Tanzanian Economy with 79 sectors of economy and cannot be solved by such a simple procedure. Triangulation is an important factor in understanding complex series of interactions among sectors of any county’s economy [19].

III. GDA ALGORITHM FORMULATION

One of the main challenges associated with global heuristic techniques such as Genetic Algorithms, Tabu Search, Simulated Annealing and many others is the sheer number of parameters that have to be selected and their sensitivity towards the choice of the best solution. Great Deluge algorithm was designed to address this problem of multiple parameters by minimizing the number of parameter requirements without jeopardising the quality of solution. The algorithm was introduced by Dueck, G. [13] and in general it requires only one parameter.

The idea is a simulation of an object in a mountainous space which is under pouring rain. The object wonders randomly on the space, but there is water level below which it can’t go because of water. If this level is \(L\), then the object accepts any area that has a value greater than \(L\). As time goes on, \(L\) rises slowly and finally forced up onto a peak (and then the rain stops). The idea can easily be defined for the minimization case as shown in the pseudo code in Fig:

Copyright © 2015 MECS
Great Deluge Algorithm

Set the Initial Solution $S_o$;
Calculate Initial cost $f(S_o)$;
Initial Level $L = f(S_o)$;
Specify input parameter $\Delta L$;
While no further improvement possible {
  Define neighbourhood $N(S_o)$
  Randomly select a candidate solution $S \in N(S_o)$;
  If $f(s) \leq f(S_o)$
    Accept new solution ($S_o = S$);
  Else if $f(s) \leq L$
    Accept new solution ($S_o = S$);
  Else
    Reject new solution
  Lower Level $L = L - \Delta L$;
}

$S_o$ is the best solution;
End Great Deluge

Fig. 3. Great Deluge Algorithm’s Pseudo-code

In this minimization case the algorithm allows a reduction in solution values according to their improvement. However, the approach also accept worse candidate solution if its value is less than or equal to the given upper limit $L$. The main function of $L$ during search process is to restrict some of the search space and thereby forces the current solution to escape into the remaining feasible space. It can be noted that the user only needs to decide one input parameter i.e. $\Delta L$ which controls the reduction of level $L$.

Initially, the controlling process is slow, and in this case the value of $L$ does not exceed the current solution, but only prohibits the longest backward moves [20]. The situation progresses until the value of $L$ exceeds the current solution, and in this lower level, only good moves are accepted.

A. Initial Solution

Initially, we need a quick feasible solution as an input to the GDA. The initial solution in this case is easily found by picking all upper triangle values of the solution matrix. In this case we are guaranteed that the solution does not contain any dicycles and covers all nodes of the associated graph. Therefore the initial solution is set as $S_o = \left( x_{ij}^0 \right)$ such that:

$$x_{ij}^0 = \begin{cases} 
1 & \text{for all } i < j \\
0 & \text{otherwise}
\end{cases}$$

Objective function value for the initial solution is the sum of the product of weights and the associated binary values in the solution matrix and is represented by

$$f(S_o) = \sum_{i,j=0}^{n-1} \sum_{j=0}^{n-1} w_{ij} x_{ij}^0$$

Where $w_{ij}$ is the weight of arc $(i, j)$ in the input-output table.

B. Moves and Neighbourhood structure

The algorithm uses swap moves which involve swapping of arcs in the graph. Two nodes $i$ and $j$ are selected at random in the current solution and swapped into $j$ and $i$ in the order. The corresponding binary values are therefore swapped as well, since only one of the $(i,j)$ and $(j,i)$ can have a value of 1 at the same time in the solution. After this choice the algorithm check if there is violation of constraints. If no violation, the swapping is confirmed otherwise no swapping is done. The process continues until a swap with no violation of constraints is found.

C. Checking violation of constraints

Given the choice of the initial solution, it guarantees that only one of $x_{ij} = 1$ at the same time, and therefore the constraint $x_{ij} + x_{ji} = 1, \forall i, j \in V$ is satisfied. Furthermore, the selection of a swap move guarantees that the constraint will always be satisfied. The only constraint to be checked for violation is the dicycle constraint. Given two nodes say $i$ and $j$ which have been picked randomly for swapping, and index $k = \{1, 2, ..., n-2\}$ is defined and the following is checked for any violation:

$$x_{ij} + x_{jk} + x_{ki} \leq 2, \forall i, j, k \in V_n, j \neq k.$$ The algorithm stops once the first violation is detected and returns a violation indicator; otherwise the swapping process is confirmed.

D. Increasing level rate ($\Delta L$)

As shown in Fig. 3, Great Deluge algorithm is designed to accept good solutions but can accept bad moves only when the function value is greater than a specified level value (Level). Given an initial solution $f(S_o)$, the increasing level rate is calculated as follows:

$$\Delta L = \frac{f(S_o)}{N_{mov}},$$

where $(N_{mov})$ is the pre-determined number of moves and the only input parameter. Initially Level is assigned the values of the initial solution, it is then steadily increased by $\Delta L$ at each iteration.

IV. ANALYSIS OF RESULTS

The algorithm code was implemented on a C++ programming language and tested on the personal computer with 2.4GHz speed processor. It has been tested on the Tanzanian input-output table. Tanzanian input-output economic table is classified into 79 main sectors of economy based on the type of product produced in each unit according to Tanzanian Central Bank (CB) data of 1992 [21]. This therefore gives a $79 \times 79$ size of a matrix. The algorithm was tested by different values of $(N_{mov})$ and the solution obtained after a number of iterations (MaxIters) as shown in Table 3. The number of MaxIters was varied from 0 to 44,000 with fixed interval of 2,000. The results were recorded.
three times at different number of moves ($Nmov$) i.e. 10,000, 30,000, and 60,000.

Table 3. Algorithm Performance Results

<table>
<thead>
<tr>
<th>MaxIters</th>
<th>S1(10,000)</th>
<th>S2(30,000)</th>
<th>S3(60,000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>585,481</td>
<td>585,481</td>
<td>585,481</td>
</tr>
<tr>
<td>2,000</td>
<td>677,018</td>
<td>623,765</td>
<td>608,260</td>
</tr>
<tr>
<td>4,000</td>
<td>753,655</td>
<td>682,803</td>
<td>641,191</td>
</tr>
<tr>
<td>6,000</td>
<td>802,245</td>
<td>702,740</td>
<td>649,524</td>
</tr>
<tr>
<td>8,000</td>
<td>816,925</td>
<td>741,428</td>
<td>664,371</td>
</tr>
<tr>
<td>10,000</td>
<td>830,979</td>
<td>778,743</td>
<td>683,898</td>
</tr>
<tr>
<td>12,000</td>
<td>838,002</td>
<td>813,608</td>
<td>704,031</td>
</tr>
<tr>
<td>14,000</td>
<td>839,787</td>
<td>833,948</td>
<td>727,310</td>
</tr>
<tr>
<td>16,000</td>
<td>839,842</td>
<td>839,730</td>
<td>833,667</td>
</tr>
<tr>
<td>18,000</td>
<td>839,842</td>
<td>839,842</td>
<td>838,126</td>
</tr>
<tr>
<td>20,000</td>
<td>839,842</td>
<td>839,842</td>
<td>838,168</td>
</tr>
<tr>
<td>22,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,695</td>
</tr>
<tr>
<td>24,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,838</td>
</tr>
<tr>
<td>26,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>28,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>30,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>32,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>34,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>36,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>38,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>40,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>42,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
<tr>
<td>44,000</td>
<td>839,842</td>
<td>839,842</td>
<td>839,842</td>
</tr>
</tbody>
</table>

The results are as shown in Fig. 5, where the solution values are fluctuating above the level line. Initially there is high fluctuation showing the high acceptance of bad moves but later the solution stabilizes and finally converges regardless to the increase the level values. This clearly demonstrates the expected performance of Great Deluge and the influence of the level parameter in the quality of solution.

Fig. 5. Solution-Level relationship

A. Degree of Linearity and Order of sectors

Degree of linearity is an index that shows the extent of triangulation of the matrix. This is given by

$$\lambda = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} W_{ij}}{\sum_{j=1}^{n} \sum_{i=j+1}^{n} W_{ij}}$$

(3)

This is basically the ratio of the sum of the weights above the diagonal to the sum of all weights in the matrix (except diagonals). The value of $\lambda=1$ for a perfectly linear economy, Leontief [22]. The computation of the degree of linearity ($\lambda$) for our solution is shown in Table 5.

Table 5. Measure of Linearity

<table>
<thead>
<tr>
<th>Sum of I/O table entries for $i &lt; j$</th>
<th>Sum of I/O table entries for $i \neq j$</th>
<th>Degree of Linearity</th>
</tr>
</thead>
<tbody>
<tr>
<td>839,842</td>
<td>890,629</td>
<td>94.3%</td>
</tr>
</tbody>
</table>

The degree of linearity attained so far (94.3%) shows how well the input-output table for this particular problem is triangulated. The original sectors of the economy in the input-output tables were ordered as shown in the table 6.

After running the Great Deluge Algorithm the results are shown in Table 7 where the order has been completely changed to reflect a maximization of the upper diagonal entries of the input-output table.
which have been tested with other algorithms and compare results. However fast the algorithm is, it is still not guaranteed that the obtained solution is optimal. A lot of effort has been devoted to development of exact methods especially in the identification of unique facets for the problem. It is therefore worth investigating further the use of exact methods for the Tanzania Input-output table.

REFERENCES


Authors’ Profiles

**Amos Mathias**: A postgraduate student in the Masters of Science in Mathematical modeling programme at the University of Dar es Salaam in Tanzania. Mathias is also working in the Department of Science Mathematics and Technology Education at the University of Dodoma in Tanzania.

**Allen Rangia Mushi**: An Associate Professor of Mathematics from the Department of Mathematics of the University of Dar es Salaam in Tanzania. His area of interest is Operations Research specifically the Combinatorial Optimization Problems. Major research work includes Timetabling, Linear Ordering, Resource Leveling in manufacturing industries, Municipal Solid Waste Disposal models and Graph Theory.