African Journal of Science and Technology (AJST)
Science and Engineering Series Vol. 6, No. 1, pp. 51 - 64

THE LINEAR ORDERING PROBLEM: AN ALGORITHM FOR THE OPTIMAL SOLUTION

Mushi, A. R.

Mathematics Department, University of Dar es salaam

ABSTRACT:- *In this paper we describe and implement an algorithm for the exact solution of the Linear Ordering problem. Linear Ordering is the problem of finding a linear order of the nodes of a graph such that the sum of the weights which are consistent with this order is as large as possible. It is an NP - Hard combinatorial optimisation problem with a large number of applications, including triangulation of input - output matrices in Economics, aggregation of individual preferences and ordering of teams in sports. We implement an algorithm for the exact solution using cutting plane and branch and bound procedures. The program developed is then applied to the triangulation problem for the input - output tables. We have been able to triangulate input - output matrices of size up to 41 x 41.*

INTRODUCTION

The Linear Ordering Problem (LOP)

The Linear Ordering Problem can be stated as follows;

Given a complete digraph $D_n = (V_n,A_n)$ of n nodes with a non-negative weight function $C : A \to \Re_+$, find an acyclic sub digraph of maximum total weight. Expressed equivalently, we want to find a linear order of the nodes of D_n such that the sum of the arc weights, which are consistent with this order, is as large as possible.

LOP is an NP-Hard problem, with many applications, including archaeological seriation, aggregation of individual preferences in sports, and the triangulation of input-output tables.

Since LOP is NP-Hard, no exact algorithm is known for the general solution. General algorithms exists but can only be used for small size instances due to the complexity of the solutions space which grows exponentially with the size of the problem. However, efforts have been made to obtain improvements on the exact methods with some success, where considerably large instances have been solved to optimality (see [1] and [2] for the work done on the Travelling Salesman Problem).

The idea in this approach is to relax the integer constraints in the formulation of the problem and solve it as a continuous problem. Algorithms such as simplex can solve such continuous problems to optimality. The challenge therefore is to develop facets, a set of inequalities which describes the polytope associated with the problem. In terms of the formulation of the problem, these are the deepest cuts to the relaxation ([3] and [4])). Once these are known, they can be applied to the relaxation as constraints and solved by simplex methods to obtain an optimal solution with guaranteed integer optimal solution.

It is not possible though, to obtain all facets associated with an NP-hard problem. So the relaxation is solved by using the known facets. If the optimal solution is not found, a branch and bound procedure can be used to finish up the problem. The algorithm is therefore known as branch and cut.
We present the known facets of the Linear Ordering Problem, the general cutting plane and branch and bound algorithm and present our implementation on the triangulation of the input-output economic matrices.

Facets of the LOP

The theory of polyhedral combinatorics is used to describe the structure of the linear ordering problem. This is done by describing the Linear Ordering Polytope (hereby denoted as P_{LO}^{n}) through its facets.

Since the solution is an acyclic sub-digraph say D_n we define a variable $x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in D_n \\ 0 & \text{Otherwise} \end{cases}$

Minimal Inequalities [5]

Let n > 2. Then the system $x_{ij} + x_{ji} = 1$ for all $i, j \in V_n$, $i < j$ is a minimal equation system for P_{LO}^{n}.

Facets Induced by Dicycles

These are inequalities which excludes dicycles into the solution vector. If C is a dicycle in D_n, $n \geq 3$, consisting of 3 arcs, then the 3-dicycle inequality $x(C) \leq 2$ defines a facet of P_{LO}^{n}. [5]

Facets Induced by k-Fences

Definition:

A digraph D = (V, A) is called a k-fence if it has the following properties;

(i) $|V| = 2k$, $k \geq 3$,
(ii) V can be partitioned into two disjoint subsets $U = \{u_1, \ldots U_k\}$ and $L = \{l_1, \ldots, l_k\}$ such that

$$A = \bigcup_{i=1}^{k} (\{(u_i, l_i)\} \cup \{(l_i, u_j)\} \mid j \in \{1, \ldots, k\}, j \neq i)$$

Let D = (V, A) be a k-fence contained in D_n, $n \geq 2k$. Then the k-fence inequality $x(A) \leq k^2 - k + 1$ defines a facet of P_{LO}^{n} where $x(C)$ is the sum of arcs of C [5]. The arcs (u_i, l_i) are called pales, while arcs (l_j, u_j), $j \neq i$ are called pickets.

Facets Induced by Möbius Ladder

Definition:

Let D = (V, M) be a sub-digraph of D_n which is generated by the k-dicycles $C_1, \ldots, C_k$ i.e.

$V = \cup V(C_i)$, $M = \cup C_i$ D is called a Möbius Ladder if it satisfies the following properties;

(i) $k \geq 3$ and odd
(ii) The length of C_i is three or four, $i = 1, \ldots, k$
(iii) The degree of each node $u \in V(M)$ is at least three
(iv) If two dicycles C_i and C_j, $2 < i+1 < j \leq k$ have a node, say v in common, then C_j is either left-adjacent or right-adjacent to C_i but not both
v) Given any dicycles C_j, $j \in \{i, \ldots, k\}$, set (i)$J = \{1, \ldots, k\} \cap \{j-2, j-4, \ldots\} \cup \{j+1, j+3, \ldots\}$. Then the set $M \setminus \{e_i \mid i \in J\}$ contains exactly one dicycle namely C_j.

Let D = (V, M) be a Möbius Ladder in the complete digraph D_n generated by the k-dicycles $C_1, C_2 \ldots C_k$. Then the Möbius Ladder inequality $x(M) \leq |M| - \frac{k+1}{2}$ defines a facet of P_{LO}^{n} for $n \geq |V|$. [6]

Branch and Cut Algorithm

We now describe how the algorithm for solving the Linear Ordering problem was developed. The Linear Ordering problem that we are interested in generally takes the following form:

Given a complete digraph $D = (V_n, A_n)$ and a vector $c \in \Re^{n(n-1)}$ then,

$$\boxed{\begin{array}{ll} \text{Maximise } c^T x & \text{(LO)} \\ \text{Subject to } x \in P_{LO}^{n} & \end{array}}$$

We would like to solve a relaxation of this problem which will contain as many facet defining inequalities as possible. Considering the 3-dicycles, 3-fences, Möbius ladders, together with the minimal equation system we have the following relaxation:

(RLO)

$$\begin{aligned}
&Maximize \sum_{\substack{i,j\\ i\neq j}} c_{ij}x_{ij}\\
&subject\ to\\
&x_{ij}+x_{ji}=1, \text{ for all } 1\le i\le j\le n,\\
&x_{ij}\ge 0, \text{ for all } 1\le i, j\le n,\ i\neq j,\\
&x(C)\le 2, \text{ for all 3-Dicycles C in } A_n,\\
&x(F)\le 7, \text{ for all 3-fences D=(V,F) in } D_{n,}\\
&x(M)\le 8, \text{ for all Möbius ladders D=(V,M) in } D_n
\end{aligned}$$

This is a zero-one problem with n(n-1) variables. If we drop the integral condition of the variables, we can solve the linear programming problem by the normal simplex algorithm. If the result of this initial simplex step is integral no more needs to be done; otherwise we must further solve this initial solution (by using an integer programming algorithm) so as to obtain an integer solution.

However, it has been shown that this problem has $\binom{n}{2}$ equations, n(n-1) nonnegativity constraints, 2 $\binom{n}{3}$ 3-dicycle inequalities, 120 $\binom{n}{6}$ 3-fence inequalities, and 360 $\binom{n}{6}$ Möbius ladder inequalities [6]. 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 we apply the cutting plane and branch & bound algorithms as shown by the following pseudo code.

Procedure cutting plane

```
{Solves RLO using cutting planes}
$P:\{x\in\Re^{n(n-1)} \,|\, x_{ij}+x_{ji}=1 \text{ for all } 1\le i\le j\le n$
$x_{ij}\ge 0 \text{ for all } 1\le i, j\le n\}$
Found := True;
Do While (Found)
Solve Max $\{c^T x \,|\, x\in P\}$ and let $\overline{x}$ be the optimal solution;

If there exists a facet defining inequality $a^T x\le a_o$ such that $a^T\overline{x}\ge a_o$ then do;
$P:=P\cap\{x\in\Re^{n(n-1)} \,|\, a^T x\le a_o\}$
                Found := True;
   Else Found := False;
 End;
    if x is integral then $\overline{x}$ solves the linear ordering problem;
    Else start Integer programming algorithm (Branch and Bound).
End cutting-plane;
```

Transformation of P_{LO}^n

The linear program of P_{LO}^n has n(n-1) variables. We can halve the number of variables by the following transformation;

Since the minimal equation is $x_{ij}+x_{ji}=1$, for all $1\le i\le j\le n$, , then we can substitute x_{ij}, j < i, by $1 - x_{ji}$ in all inequalities and in the objective function.

The 3-dicycle inequalities $x_{ij}+x_{jk}+x_{ki}$ £ 2, are transformed into $x_{ij}+x_{jk}-x_{ik}\le 1$ if i<j<k, or into $-x_{ji}-x_{kj}+x_{ki}\le 0$ if i>j>k. The trivial inequalities and equations change to $0\le x_{ij}\le 1$ for all $1\le i<j\le n$.

The original P_{LO}^n has now been replaced by its projection denoted by $\overline{P}_{LO}^n$ into the real vector space $\Re^{\binom{n}{2}}$ which is of full dimension and has the same number of vertices as P_{LO}^n Also the objective function is transformed as follows:

$$\sum_{\substack{i,j\\ i\neq j}} c_{ij}x_{ij} = \sum_{i<j} c_{ij}x_{ij} + \sum_{i>j} c_{ij}(1-x_{ji})$$
$$= \sum_{i<j} c_{ij}x_{ij} + \sum_{i<j}(-c_{ij}x_{ij}) + \sum_{i>j} c_{ij}$$

Since the last term is a constant, which does not affect optimal solution, then we

$$\text{maximize: } z = \sum_{i<j} c_{ij}x_{ij} + \sum_{i<j}(-c_{ij}x_{ij})$$

The optimal value from $\overline{P}_{LO}^{n}$ differs from that of P_{LO}^{n} by a constant value $\sum_{i>j} c_{ij}$. The objective function will be denoted by $\overline{c}^{\,T}x$ with $\overline{c} \in \Re^{\binom{n}{2}}$ and $\overline{c}_{ij} = (c_{ij} - c_{ji})$ for all $1 \leq i < j \leq n$. The initial solution is therefore just the trivial inequalities stated above.

Implementation

Looking back to our general algorithm, the following questions remain to be answered:

1. How can we detect the violated inequalities?
2. Which inequalities are to be added to the RLO if more than one violation is found?
3. Should one class of facets have preference over another?

We firstly answer the second question. It is noted that for the case of 3-dicycles, a large number of constraints (violated inequalities) may sometimes be generated in a single pass of the algorithm. This may create a storage problem. There are three strategies which can be used to overcome this:

(i) **All violated:** If only a few inequalities have been generated (a fixed number have to be set), then all are inserted to the RLO,

(ii) ***k* most violated:** If large number of inequalities have been generated, then a fixed number of them say *k* are chosen and added to the RLO. The choice is based on the most violated criteria. That is those with larger right hand sides.

(iii) **Arc disjoint**: This case is also applied in the case where a large number of constraints are generated. A subset of the violated inequalities is chosen with the property that no two corresponding 3-dicycles have an arc in common. This is based on the idea (not theoretically verified) that one inequality may be sufficient to locally decrease the infeasibility of the current solution. [5]

To answer the third question, the 3-dicycles are preferred because they have to be present to exclude infeasible integer solutions and because they can be detected more efficiently than the other classes of inequalities. The order therefore is to check 3-dicycles first until no more are detected, then detect the k-fences, and then Möbius ladders.

More specifically the algorithm then looks as follows:

Algorithm cutting plane2

```
$P := \{\Re^{\binom{n}{2}} \mid x_{ij} \leq 1,$ for all $1 \leq i \leq j \leq n,$
                $-x_{ij} \leq 0,$ for all $1 \leq i \leq j \leq n.$
Found := True;
Do While(Found)
Solve Max $\{c^T x \mid x \in P\}$ and let $\overline{x}$ be the optimum solution for the current LP;
        Check_3_dicycles(P, $\overline{x}$ ,Found);{ If found, add new constraints and } {return True to Found }
        If Not(Found) then
        Check_k_fence(P, $\overline{x}$ ,Found);
        If Not(Found) then
        Check_M_ladder(P, $\overline{x}$ ,Found);
Endwhile;
If $\overline{x}$ is integral then done;
Else Solve integer programming algorithm;
End algorithm cutting plane2
```

The following section answers the first of the above three questions:

Detection of violated inequalities:

Since there is a direct correspondence between the facet-defining inequalities for P_{LO}^{n} and $\overline{P}_{LO}^{n}$, we now describe how to detect the facets of P_{LO}^{n} but during our implementation we will use the transformation $\overline{P}_{LO}^{n}$. Since the initial linear program includes all trivial inequalities, these are satisfied by all other subsequent solutions as these solutions use the constraints of the initial program.

Detection of violated 3-dicycle inequalities:

There are $2\binom{n}{3}$ different 3-dicycles in A_n. We enumerate all possible violations. This procedure has complexity of order O (n^3).

Procedure Check_3_Dicycle(P, $\overline{x}$,Found);

```
Found := False; {will change to True if any inequality is found}
For i := 1 to n-1 do
        For j := i+1 To n-1 do
                For k := j+1 To n do
                        IF($x_{ij} + x_{ji} - x_{ik} > 2$) then
                          InsertH(i,j,k,H,S);{in
                           sorted ascending order}
                                Found := True;
                        Endif;
                        IF($-x_{ij} - x_{jk} + x_{ik} > 2$) then
                                InsertH(k,j,i,H,S)
                                Found := True;
                        Endif;
                Endfor;
        Endfor;
Endfor;
IF FOUND Then
   Add_k_most_violated(H) {to the LP},
EndIf;
EndProcedure Check_3_Dicycle;
```

An array H with four sections of equal size T is used to store the violated 3-dicycles: If i_1, j_1, k_1 are nodes of the violated 3-dicycle, and i_1 is stored in position *l*, then j_1 is stored in position *l*+T, and k_1 in *l*+2T.

This dicycle may represent the inequality (i) $x_{ij} + x_{jk} - x_{ki} \le 1$,

or (ii) $-x_{ij} - x_{jk} + x_{ik} \le 0$. Thus position *l*+3T stores the status of this arc, (0 if of type (i) and 1 otherwise).

This array is sorted in decreasing order of magnitude of the violations. The violations are stored in the array S. We select the first K of them, but if less than K are generated then we select all of them.

Detection of k-fence violated inequalities:

There are $120\binom{n}{6}$ k-fence inequalities for k = 3, while there are more of such inequalities for higher values of k. Clearly enumeration over all inequalities is no longer feasible. We thus resort to a heuristic which is formulated after a careful study of the polytope P_C^n (Constructed from 3-dicycles and trivial inequalities).

The projected polytope is as follows;

$$\begin{aligned} P_C^n = \{\Re^{\binom{n}{2}} \mid\ & x_{ij} + x_{jk} - x_{ik} \le 1,\ \text{for all } 1 \le i < j < k \le n, \\ & -x_{ij} - x_{jk} + x_{ik} \le 0,\ \text{for all } 1 \le i < j < k \le n, \\ & x_{ij} \le 1,\ \text{for all } 1 \le i < j \le n, \\ & -x_{ij} \le 0,\ \text{for all } 1 \le i < j \le n\} \end{aligned}$$

This polytope has a matrix (denoted by B_n) with $2\binom{n}{3}$ rows from 3-dicycles and $2\binom{n}{2}$ rows from the trivial inequalities. Each vertex of this polytope is one of the solutions of the linear program (associated with P_C^n) and must have $\binom{n}{2}$ elements. If we let C_n denote the portion of B_n corresponding to the 3-dicycle inequalities, then the following theorem shows that the 3-dicycle inequalities are not sufficient to characterise a vertex of $\overline{P}_C^n$.

We are interested in solution vectors (vertices) violating the 3-fence inequality. These can be found by looking at the properties of some of these violations. Consider a 3-fence D = (V, F) in a complete digraph D_6 and $c_F \in \Re^{30}$ as its incidence vector. Maximising its objective function $c_F^T x$ over P_C^6 an optimum solution y* was found as depicted by Figure 1.

Optimum solution y* over P_C^6

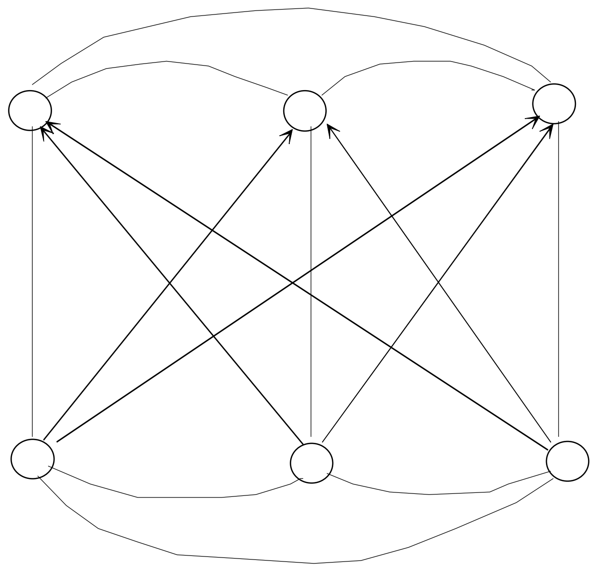

Figure 1: Möbius Ladders

Note: The broken lines correspond to fractional components with value 0.5 in both directions (i.e. $x_{ij} = 0.5$ and $x_{ji} = 0.5$). The solid lines correspond to components having value 1 (i.e. $x_{ij} = 1$ and $x_{ji} = 0$ or $x_{ij} = 0$ and $x_{ji} = 1$) in the direction of the arrow.

Thus we note that the components indexed by arcs anti parallel to the pickets, have the value 0, and all pickets have the value 1. Moreover we can see that all pales have value 0.5. If we sum the values of all arcs in the 3-fence F, we clearly see that the 3-fence inequality $x(F) \leq 7$ is violated (since the sum of F is 7.5).

Heuristic:[5]

The following heuristic was described by Reinelt [6]. The heuristic is based on the uniqueness of the solution in Figure 1 for any 3-fence. Therefore it is assumed that the common property of all inequalities violating a 3-fence is the nonintegrality of the pales.

If y is the solution of the current cutting plane algorithm, then we define a corresponding graph (undirected) $G_y = (A(A_y),A_y)$ by setting $A_y = \{ij \mid y_{ij}$ is fractional$\}$. In this graph the fractional components may be different from 0.5 whenever the current polytope is not the 'pure' polytope. Moreover in order to speed up the running time, instead of A_y we choose the arc set

$$Ay = \{(i, j) \big| \varepsilon \leq y_{ij} \leq 1 - \varepsilon \text{ where } \varepsilon \text{ was set to 0.1.}$$

We then separate all 3-fences and test them as follows. We enumerate the triple edges of G which are non adjacent, i.e. edges which have no node in common. Each of these 3-pairs is assumed to be a pale of a 3-fence. For each of these pairs, there are 8 possible orientations of their edges for a 3-fence and at most one of them will violate a 3-fence inequality. We check this violation by constructing all possible orientations. We note that once a violation is found there is no need to go on with the construction of the remaining orientations since that violation is unique.

The general algorithm is:

```
Procedure Check_3_fence(P, $\overline{x}$ ,Found)
Found := False;
(i)   Find_fractional_components of  and store them in
      an array say F;
(ii)  Identify_non-adjacents on F {remove the rest from
      F}
         Number F in order 1,2,...,u.
         If u £ 2 then Return; {No 3-fence}
         Else {u >2}
(iii) {Enumerate all possible k-fences}
         While there are more 3-pair permutations do
                  While not(Found) and there are more
                  3-fence orientations do
(iv)  Check_3_fence_Violation(Found); {Returns True if
      Found}
         If(Found) then Storeconstraint(F);
                  Endwhile;
         Endwhile;
         Endif;
         AddFences(F)
End Procedure Check_3_fence;
```

Graph data structure

The graph D_n in this approach is represented as an array of length 2n. The first n cells store the *i* part of the arcs and the following n cells store the *j* part. For instance if (i,j) is an arc in the graph D_n, and *i* is stored in position $l \leq n$ in the array, then *j* will be stored in position l+n.

(i) Finding fractional components

```
Procedure Find_fractionals(F, x, u);
u := 0

For each arc (i,j) with i ≠ j do {with $\varepsilon$ = 0.1}

IF $(\varepsilon \le x_{ij} \le 1-\varepsilon)$. Then
        Increment(u) {number of fractional
        components}
                F(u) := i;
                F(u+n) := j;
        Endif;
Endfor;
End_algorithm; {At the end the graph F will contain u
components}.
```

(ii) Finding non-adjacent components

Two arcs are non-adjacent if they have no node in common. Thus if we have two different arcs (i1,j1) and (i2,j2), they can be adjacent in the following four possible ways :

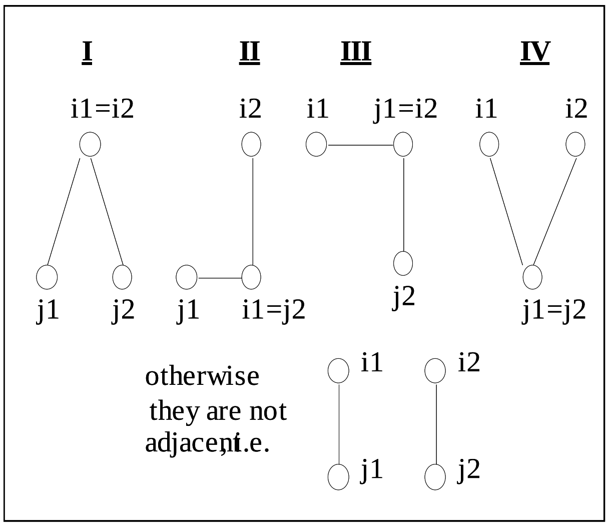

Figure 2: Finding non-adjacent components

All possible 2-pair arcs from fractional components F are enumerated and checked for adjacency. We note that, if two arcs in the set F are adjacent, then one must leave the set as they can not both be members of the set of non-adjacent arcs.

Thus we start with the first node in F and eliminate all arcs in F adjacent to it by setting their indices in F to zero. We then move to the next non-zero node in F and repeat the above process. At the end we collect all non-zero arcs in F as non-adjacent fractional components.

```
Procedure Find_Non-Adjacents(F,u,x);

For i := 1 to u do
        If (F(i) ≠ 0) Then
          For j := i+1 to u do
                If(F(j) ≠ 0) Then
                        i1 := F(i); j1 := F(i+u);
                        i2 := F(j); j2 := F(j+u);
                        If(i1=i2) OR (i1=j2) OR (j1-i2)
                        OR (j1=j2) Then
                                F(j) := 0;
                                F(j+u) := 0;
                        EndIf;
                EndIf;
          Endfor;
        EndIf;
Endfor;
    For each i with F(i) ≠ 0 do
        Pull_Non_adjacents(F(i)) {pull non-adjacents
together}
   Endfor;
Endalgorithm;
```

(iii) Enumerating the K-fences

We enumerate all possible subsets of k elements from u elements of F (fractional non-adjacent components). This is done by a call to subroutine NEXKSUB which returns an array of k elements different to the previous call, until all subsets are enumerated (see Nijenhuis, A 1978 [7]). When all subsets have been exhausted, NEXKSUB returns False to a logical variable MTC (More To Come).

Each of these k element subsets is considered to be a set of pales of a k-fence, and is checked for k-fence violation.

(iv) Checking for k-fence violation

A set with k pairs of non-adjacent fractional arcs from F is checked for all possible orientations of a k-fence. For instance for a 3-fence, there are eight possible orientations for the pales of a 3-fence. Suppose the arcs chosen are (i1,j1), (i2,j2), (i3,j3), (i4,j4). The following are the eight possible orientations of a 3-fence pales:

We want to find the general algorithm which will generate all possible subsets for any k. First we note that we can obtain the next subset from the previous one by swapping an arc of the previous subset. For example, set II is obtained from set I by swapping arc (i1,j1), and set III is obtained from II by swapping arc (i2,j2).

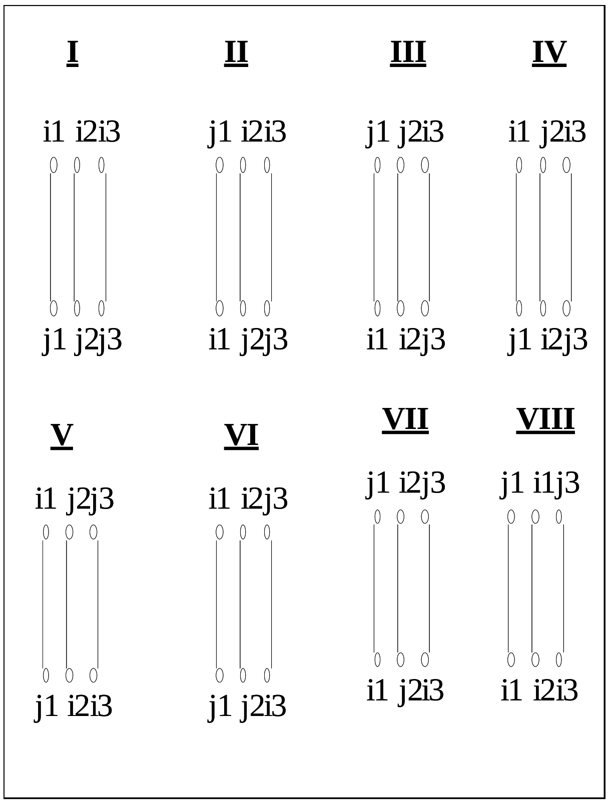

Figure 3: Checking for k-fence violation

Now suppose we have a list of k elements L_k consisting of binary values (0-1). We wish to find a successor to a subset S of L_k. In other words, we want to find the index *j* of the single element a_j in S which is to be changed (in our case swapped) in order to form the successor.
For example the list L_3 is as follows: **0**00, 1**0**0, **1**10, 01**0**, 0**1**1, **0**01, 1**0**1, 111. The sequence of indices *j* of the changed co-ordinates (indicated by bold numbers) is : 1,2,1,3,2,1,2.

This sequence of *j* values is generated by a subroutine NEXSUB (Nijenhuis, A [7]). Each index is then used to generate the next subset until all subsets are exhausted. At the end NEXSUB returns False to MTC. However if a selected subset is found to violate a k-fence inequality the algorithm stops as this k-fence is unique. The violation is added as a new constraint of an LP.

Procedure Check_F_violation(F)

```
MTC := True;
Found := False;
While(MTC and NOT Found) Do
        NEXSUB(j)
        Swap(F(j),F(j+n));
        IF(x(F) > $k^2$-k+1)Then
                Found := True;
                Add(F) {to LP}
        Endif;
Endwhile;
Endalgorithm;
```

Detection of violated Möbius Ladders

Again enumeration of all possible Möbius Ladders is not possible. The heuristic is based on finding one of the smallest Möbius Ladders, as candidates for a Möbius Ladder separation routine Given a Möbius Ladder D = (V,M) and it's incidence vector $c_M \in \Re^{30}$ if we maximise c^T_Mx over P_C^6 we get the optimal solution y* as shown by the figure below. It's value is *x*(M) = 8.5 which clearly violates Möbius Ladders inequality (i.e. *x*(M) ≤ (8).

Note: Faint lines correspond to the value 0.5 (in both directions) and solid lines correspond to a value of 1. Components opposite to the solid lines (not shown) have value 0.

$\bar{M}$ resembles Figure 4 but with the solid line components reversed. The heuristic exploits this structure and works as follows. Given y as the solution to the current problem, we develop a graph $G_y = (V(A_y),A_y)$ of fractional components as done for the k-fences problem. We then enumerate all 4-cycles without diagonals in G_y corresponding to the four nodes 1,2,3,4, as shown. For each of these 4-cycles, we try to find another node *w* which is adjacent to the four nodes (node 5 of Figure 4). If we are successful then we enumerate all nodes adjacent to *w* as possible candidates for the role of node 6 above. The role of node 6 is that such that when a node is in position 6 it creates a new Möbius ladder. Each of the 6 nodes identified in the enumeration are treated as nodes of a Möbius Ladder isomorphic to M or $\bar{M}$ and are checked for Möbius Ladder inequality violations.

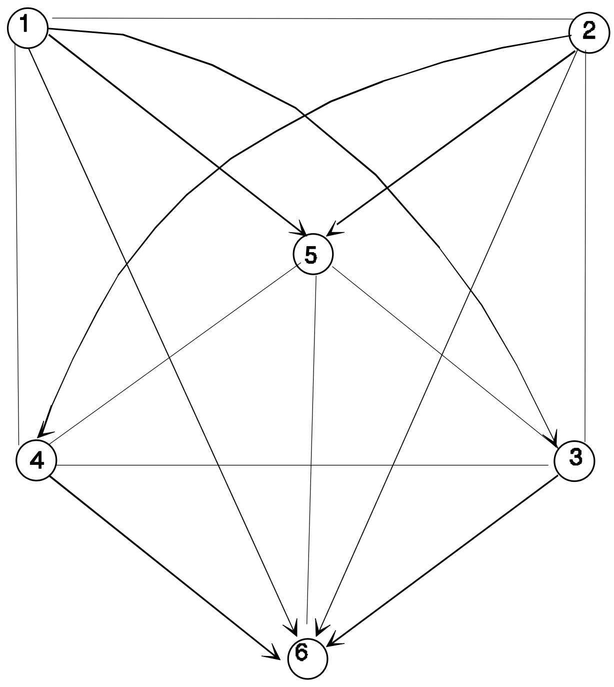

Figure 4: Detection of violated Möbius Ladders

Algorithm_Möbius_Ladder;

```
Find_fractionals(y,F) {enumerate all 2-pairs of F which
        forms a 4-cycle, these are the non-adjacent arcs}
    For each 4-cycle($a,b,c,d$) do
        Find_node_w($w$) { adjacent to the 4-cycle}
                {Since this is a complete
                digraph, then it is sufficient to find a
                node which is not one of the 4-cycle
                nodes}
            If ($w$_found) then
                List_6_node($q$) {enumerate all nodes
                adjacent to $w$ as possible candidates
                for node 6 role. That is not among the
                5 nodes selected so far}
                For each of ($a,b,c,d,w,q$) do
                    Check_Mobius_ d_
                    Violation ($a,b,c,d,w,q$)
                    {add new constraint if
                    found}
                Endfor
            Endif
    Endfor
Endalgorithm Möbius_Ladder;
```

Summary of the Cutting Plane algorithm

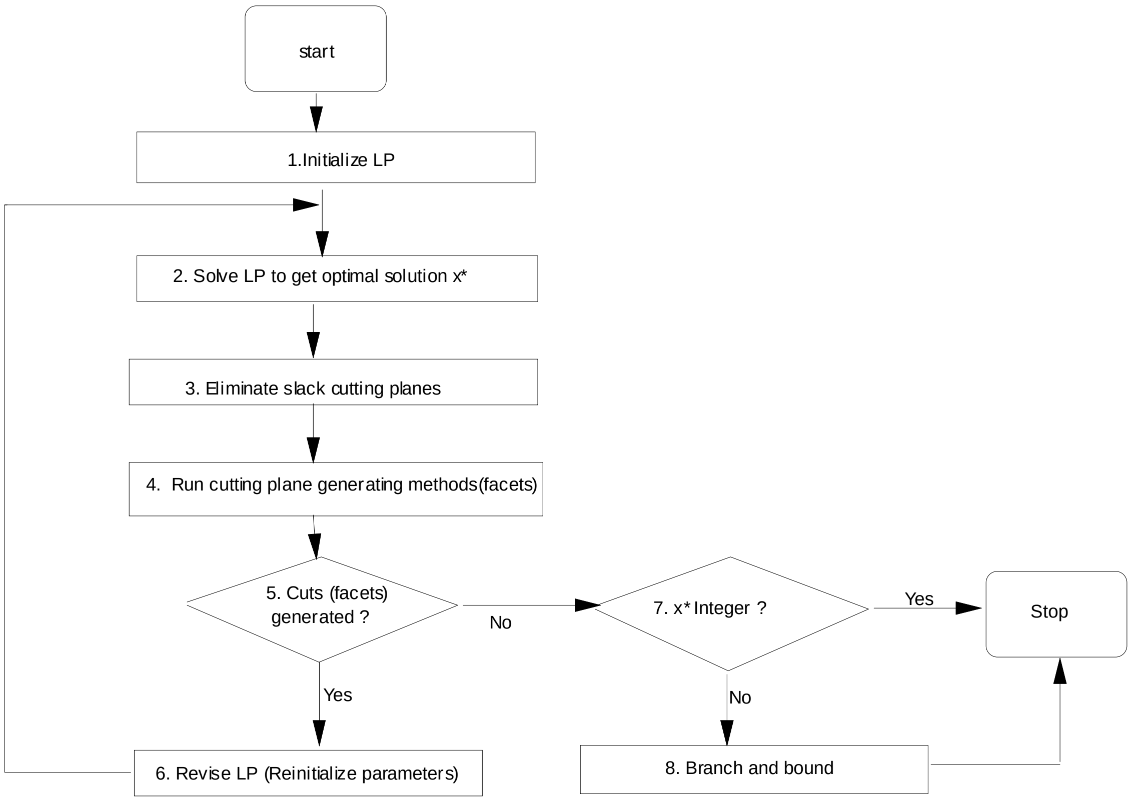
start
1.Initialize LP
2. Solve LP to get optimal solution x*
3. Eliminate slack cutting planes
4. Run cutting plane generating methods(facets)
5. Cuts (facets) generated ?
No
7. x* Integer ?
Yes
Stop
Yes
No
6. Revise LP (Reinitialize parameters)
8. Branch and bound

Figure 5: Cutting Plane Algorithm

1. Preparation of the LP with the trivial inequalities.
2. Solve LP to get optimal solution, using an LP package (LP solver)
3. Eliminate cutting planes: We delete all constraints having positive slack variables, so as to decrease the size of the linear program. These deleted constraints may be reinserted during optimisation but practically these results in smaller programs without considerably affecting the speed of optimisation.
   We find these slack variables by checking all basic variables with indices greater than the number of variables n. These variables will correspond to slack variables which remained basic (i.e. positive slacks) in the optimal solution.
4. The identification of the facets as explained previously.
5. If cuts were generated, we go to step 6 for revision of LP, otherwise we go to step 7 for integrality checking.
6. Revision of LP: Any cutting planes generated are added to the LP and then the algorithm is repeated from step 2.
7. If x* is integral we have found a solution, otherwise go to step 8.
8. Perform branch and bound to get an integral solution using an LP solver.

Summary of Results

The 20x20 tables were extracted from the Input/Output tables. The k most violated inequalities for 3-dicycles were chosen between 100 and 500. In the following table we list the running time of the algorithms, the degree of linearity and density of each table used. The running times include the time used in generating MPS format files and the PRIMAL and DUAL algorithms. The first two numbers of the table names represent the size of the table and the next two represent the year of the table from which it was extracted (the first letter is used to distinguish one table from another).

Table 1: Summary of Results for 20x20 tables

Table (k=100)	Density	Degree of Linearity	CPU time (seconds)
A2085.d	64.73	62.95	11.81
B2085.d	59.47	93.21	11.76
C2085.d	35.53	62.00	19.70
D2085.d	39.74	90.29	10.94
E2085.d	56.05	53.43	16.34
F2085.d	52.63	88.83	12.22
G2085.d	30.78	79.99	17.37
H2085.d	33.42	91.57	11.52
A2064.d	60.53	65.57	11.09
B2064.d	65.53	90.74	9.65
C2064.d	64.21	76.19	18.23
D2064.d	71.32	85.63	11.68
A2069.d	52.63	67.34	22.45
B2069.d	60.79	88.11	9.65
C2069.d	62.37	84.36	17.70
D2069.d	70.26	92.50	11.81
A2075.d	55.00	72.52	12.16
B2075.d	50.00	95.83	14.75
C2075.d	34.74	63.06	12.18
D2075.d	46.05	90.91	12.28

Table 2: Summary of Results for 25x25 tables

Table (k = 200)	Density	Degree of Linearity	CPU time (Seconds)
A2585.d	59.00	62.67	20.81
B2585.d	51.00	94.39	33.37
C2585.d	41.00	63.75	17.59
D2585.d	49.67	86.75	23.82
E2585.d	51.83	55.96	21.56
F2585.d	44.67	87.12	24.31
G2585.d	35.83	55.89	30.75
A2564.d	66.50	67.55	28.20
A2569.d	58.33	69.66	31.36
A2575.d	48.83	70.60	29.50
B2575.d	42.00	95.14	22.16

Table 3: Summary of Results for 33x33 tables

Table (k = 500)	Density	Degree of Linearity	CPU time (seconds)
A3385.d	52.18	70.33	97.45
B3385.d	45.36	65.21	63.88
A3364.d	70.45	71.60	50.16
A3369.d	67.23	74.27	93.73
A3375.d	41.10	75.65	67.84

Table 4: Summary of Results for 41x41 tables

Table (k = 500)	Density	Degree of Linearity	CPU time (seconds)
A4185.d	48.66	69.79	237.14
B4185.d	42.62	71.97	173.41
C4175.d	41.83	75.42	146.17

The results show that tables with up to 41 sectors can be triangulated within reasonable time scale. It can also be observed that there is very little relationship between the degree of linearity, density and time. This shows that the density of the problem can hardly affect the optimization time. This is due to the fact that the zero values in the table will only affect the objective row of the simplex tableau which is a very small part of the whole tableau.

We now illustrate the optimization process in a few cases. The number of iterations in the simplex algorithm is listed together with the facets generated, the objective function value and the degree of linearity for each phase of the problem.

Table 5: Optimisation Process; 20x20 tables

Table (k=100)		PRIMAL Iterations	Facets generated	Objective value	Size	Degree of Linearity	CPU Time
I	A2085.d	66	-	2541.50	381	23.65	2.87
II		47	118 Dicycles	2463.70	481	58.74	4.63
III		8	39 Dicycles	2463.40	520	62.95	4.31
I	A2064.d	61	-	173648.00	381	47.55	2.75
II		33	68 Dicycles	171608.00	449	58.81	4.10
III		11	39 Dicycles	171531.00	452	65.57	4.24
I	A2069.d	62	-	241189.00	381	45.67	2.89
II		52	129 Dicycles	238944.00	481	45.48	4.72
III		36	54 Dicycles	238070.00	535	66.66	5.19
IV		14	19 Dicycles	238063.00	554	67.34	4.92
V		1	7 Dicycles	238063.00	561	67.34	4.73

Table 6: Optimization Process; 33x33 tables

Table (k=500)		PRIMAL iterations	Facets generated	Objective value	Size (number of rows)	Degree of Linearity	CPU time (sec.)
I	A3385.d	157	-	3594	1057	52.45	9
II		233	670 Dicycles	3409.2	1557	74.19	25.44
III		201	321 Dicycles	3376.2	1878	75.58	30.49
IV		11	31 Dicycles	3376	1909	75.65	16.74
V		1	1 Dicycle	3376	1910	75.65	15.78

Table 7: Optimisation Process; 41x41 tables

Table (k=500)		PRIMAL iterations	Facets generated (Dicycles)	Objective value	Size (number of rows)	Degree of Linearity	CPU time (Sec.)
I	A4185.d	251	-	4743.50	1641	45.78	17.48
II		211	1358	4541.90	2141	59.27	33.37
III		215	999	4431.10	2641	67.30	45.53
IV		362	231	4394.20	2872	68.30	82.31
V		41	132	4393.10	3004	68.89	31.00
VI		18	13	4389.10	3017	69.79	28.20

Observations from Table 5 - Table 7;

- In all tables only the 3-dicycles were needed to find an optimal solution.
- Branch and bound stage was not needed in any case.
- Only a small number of phases were necessary to reach an optimum solution. Thus our semi-automatic algorithm was able to solve these real life problems in a very convenient way.
- Generally, the addition of facets showed an improvement in the degree of linearity and the objective value, which shows the effectiveness of facet defining inequalities (3-dicycles in this case) in improving the solution.

The number of iterations of the DUAL procedure and the time taken are highest in the middle phases but decrease towards the last phase. Since the size of the problem showed an increase with the number of phases, it was expected that the time would increase too (together with the number of iterations). This was not the case, and is due to the fact that the DUAL simplex procedure greatly improves the performance of the algorithm. To see this, we compare the results obtained by the application of the PRIMAL procedure alone and those from the DUAL procedure for the 41 sector table A4185.d.

Table 8: Comparisons of PRIMAL and DUAL performances

PHASE	PRIMAL		DUAL	
	Size	Time	Size	Time
I	1641	17.48	1641	17.48
II	2141	43.74	2141	33.37
III	2641	92.35	2641	45.53
IV	2774	232.69	2872	82.31
V	2803	302.48	3004	31.00
VI	2817	426.17	3017	28.20
VII	2823	309.11		
Total	**time**	**1424.02**	**237.89**	

Since the Dual Simplex algorithm works by removing the infeasibilities introduced to the previous basis due to the added cuts, the fewer the cuts added the faster the DUAL procedure will be. From the Table 5 - Table 7, the last phases involve only a few facets and hence few cuts are added. This makes the DUAL procedure very fast compared to PRIMAL. We compare the performance of the 3-dicycles procedure in relation to various k-violation strategies. This is illustrated by table A2085.d for k = 50,100 and all.

Table 9: Performance of 3-dicycles in relation to k-violations

Table A2085	Phase	PRIMAL iterations	Facets generated	Objective value	Size (number of rows)	CPU time (seconds)
	I	66	-	2541.5	381	2.87
K=50	II	30	118 Dicycles	2471.4	431	4.05
	III	16	117 Dicycles	2471	481	4.26
	IV	19	29 Dicycles	2463.4	510	5.36

Total time 15.54

	I	66	-	2541.5	381	2.87
k=100	II	47	118 Dicycles	2463.7	481	4.63
	III	8	39 Dicycles	2463.4	520	4.31

Total time 11.81

	I	66	-	2541.5	381	2.87
all	II	52	118 Dicycles	2463.7	499	4.77
	III	4	24 Dicycles	2463.4	523	4.47

Total time 12.11

In the first case (k=50), 118 dicycles were found in phase one. But only 50 were introduced to the tableau in phase II. This resulted in a relatively small sized problem and was solved faster (4.05 seconds). The next phase generated 117 dicycles, and the algorithm took four phases to reach optimality.

In the second case (k = 100), 100 dicycles were added to the tableau in phase II. This resulted in a relatively large problem (compared to the first case). The time taken to solve it was also relatively long (4.63 Seconds), but only three phases were required to reach optimality.
In the third case, all 118 dicycles were added to the tableau in phase II. The time taken to solve it was longest (4.77 Seconds) and again three phases were needed to reach optimality.

We may conclude that, for large values of k the LP becomes large and time consuming, while for small values of k, a cut generation routine is called quite often and thus gives more phases of the algorithm. But, on the other hand, we have seen that the size of the LP does not necessarily affect the time. It depends on the number of cuts added.

In general, there is no specific criterion for the selection of k. It is a better idea to do some test for each size in order to determine the best value of k. In our table (Table 9) the overall time was best for the second case (k=100). We thus used K= 100 for the 20 nodes tables. Using the same idea we selected k = 200 for 25 nodes and 500 for 41 nodes.

CONCLUSION

The aim of the study was to use the theory of polyhedral combinatorics to develop a computer algorithm for the solution of the Linear Ordering problem. This was done by describing the facet structure of the Polytope associated with the Linear Ordering Problem. Linear Programming methods were applied to develop a cutting plane algorithm. The algorithm implemented the Polytope descriptions (in terms of equations and inequalities) to find the exact solution to our problem.

It was also the intention of this project, to apply the developed algorithm to the practical problem, 'Triangulation of Input - Output tables', and hence to investigate the importance of the facets of this Polytope. We developed a semi - automatic algorithm which involved the use of cutting plane procedure and branch and bound. We then presented and analyzed the computational results. The following can be concluded;

- The k-Fence and Möbius Ladder facets did not appear on any of the problems tested. Only 3-dicycles were found and these were sufficient to get an optimal solution. We conclude that the 3-dicycles are the most important facets.

- The Dual - Simplex procedure is a very useful tool in this cutting plane algorithm, its use decreased the solution times for large problems to a great extent.
- The algorithm performed well on the problems tested. We can thus conclude that polyhedral combinatorics is a useful tool for attacking hard combinatorial optimization problems. It seems to be worthwhile to relate the polytope to combinatorial optimization problems whose vertices correspond to the feasible solutions, and then try to describe the facet structure of this Polytope.
- It has been demonstrated that it is possible to use a linear programming approach to the problem Polytope as a means of formulating algorithms for the exact solutions of some of the NP - Hard combinatorial optimization problems.

REFERENCES

1. Grötschel, M. and Padberg, M W. (1983a): "*Polyhedral aspects of the Travelling Salesman Problem I. Theory*", European Institute for Advanced Studies in Management. W. Germany.
2. Grötschel, M. and Padberg, M W. (1983b): "*Polyhedral aspects of the Travelling Salesman Problem II. Computation.*" European Institute for Advanced Studies in Management. W. Germany.
3. Bachem,A and Grötschel, M (1982): "*New aspects of Polyhedral Theory*" in Korte, B (ed): Modern Applied Mathematics: " Optimization and Operations Research", North-Holland Publishing Co. Amsterdam - New York -. Oxford, 1982, 51-106
4. Grötschel, M (1982): "*Approaches to Hard Combinatorial Optimization Problems*" in Korte,B (ed) : Modern Applied Mathematics: " Optimization and Operations Research", North-Holland Publishing Co. Amsterdam - New York -. Oxford, 1982, 437-575.
5. Reinelt, G (1985): "*The Linear Ordering Problem: Algorithms and Applications*", Herman Verlag Berlin.
6. Grötschel, M, Junger, M and Reinelt, G (1983) : "A cutting plane algorithm for the Linear Ordering Problem", European Journal of Operations Research 6 (1984), 1195-1220.
7. Nijenhuis, A and Wilf, H, S (1978): "*Combinatorial Algorithms for Computers and Calculators*", Academic Press Inc. New York - San Francisco - London.