Lagrangian Heuristics to Linear Ordering

Alexandre Belloni
Instituto de Matemática Pura e Aplicada
Rua Dona Castorina 110, Rio de Janeiro, RJ, 22460-320, Brazil
belloni@impa.br

Abilio Lucena
Departamento de Administração
Universidade Federal do Rio de Janeiro
Av. Pasteur 250, Rio de Janeiro-RJ, 22290-240, Brazil
lucena@openlink.com.br *†

October 2001

Abstract

Two heuristics for the Linear Ordering Problem are investigated in this paper. heuristics are embedded within a Lagrangian Relaxation framework and are initiated with a construction phase. In this process, some Lagrangian (dual) information is used as an input to guide the construction of initial Linear Orderings. Solutions thus obtained are then submitted to local improvement in an overall procedure that is repeated for every Subgradient Optimization iteration. On the Lagrangian relaxation side of the algorithm, Relax and Cut is used instead of a straightforward implementation of the Subgradient Method. The reason being the exceedingly large number of inequalities that must be relaxed to generate (Lagrangian) dual bounds. The proposed heuristics are tested for instances from the literature and also for some new *hard to solve exactly* ones. In particular, optimal solutions are obtained for all 79 instances taken from the literature. As a by product, optimality is also proven for 72 of them.

1 Introduction

A *linear ordering* (or a permutation) of the n elements of a finite set N is a bijective mapping $\pi : \{1, 2, \ldots, n\} \to N$. For an element $i \in N$ and an ordering π, the *position* of i in π is given by $\pi^{-1}(i)$. The ordering cost $d(\pi)$

*Corresponding author.

†Research supported by Fundação de Amparo a Pesquisa do Estado do Rio de Janeiro (FAPERJ) under grant E26/71.906/00.

is computed from a set of *precedence* costs, $\{c_{ij} : i, j \in N, i \neq j\}$. Every pair of distinct elements $i, j \in N$ must contribute with either c_{ij} or else c_{ji} to $d(\pi)$. If $\pi^{-1}(i) < \pi^{-1}(j)$ then c_{ij} is incurred. Otherwise, c_{ji} is incurred. The Linear Ordering Problem (LOP) is to find an ordering π of N with as large a $d(\pi)$ as possible.

LOP can be cast in graph theoretical terms by assuming the elements of N to be the nodes of a complete directed graph $D = (N, A)$. Accordingly, $A = \{(i, j) : i, j \in N, i \neq j\}$ is defined as the arc set and the precedence costs introduced above are thus taken to be arc costs. A subgraph $D' = (N, A')$, $A' \subseteq A$, is denoted a tournament if, for every pair of distinct nodes $i, j \in N$, exactly one of the arcs, (i, j) or (j, i), is contained in A'. An acyclic tournament D' with the largest sum of arc costs implies an optimal LOP solution to N.

LOP has a number of relevant practical applications. Among these, undoubtedly, the most commonly cited one is the *triangulation* of input-output matrices in Economics [31]. A reduced list of LOP applications is found in [14] where references to additional applications can be found.

An exact solution algorithm to LOP, based on the use of polyhedral cutting planes, is found in [14]. More recently, [26] proposed an exact solution algorithm where Linear Programming (LP) relaxations are partially solved by an Interior Point algorithm (complemented, in the end, with the Simplex Method). Families of facet defining inequalities for the LOP polytope can be found in [15], [29], [25], [30], [19], and [2].

Some recent heuristics to LOP are [6], [18], [5] and [13]. The first one uses a sorting through insertion pattern together with permutation reversals. The second one uses Tabu Search [10]. Scatter Search [11] is used in [5]. Finally, Variable Neighborhood Search [23] is used in [13]. A very preliminary version of this paper appears in [1].

Two different LOP heuristics are proposed and computationally tested here. Heuristics are embedded within a Lagrangian Relaxation (LR) framework and are repeatedly called under different inputs generated from Lagrangian relaxation outputs (either Lagrangian modified costs or else Lagrangian subproblem solutions). As a result, a series of feasible LOP solutions are generated which attempt to incorporating LR upper bounding information. On implementing such a scheme, an exceedingly large number of inequalities would normally have to be dualized (in order to generate Lagrangian LOP upper bounds). This, in turn, usually brings convergence problems to a straightforward implementation of, say the Subgradient Method (SM) of [16] (see Section 2 for details). As a result, a Relax and Cut algorithm, in the style proposed in [21, 22], is used instead.

Heuristics proposed in this study have been tested for the 49 instances of LOLIB [28], the 30 instances proposed in [26] and 21 new instances introduced here. LOLIB instances originate from the input-output matrices for some European countries. They range in size from 44 up to 60 economic

sectors. Instances in [26] were randomly generated to resemble input-output matrices. They range in size from 100 up to 250 elements. Finally the third test set was randomly generated with the aim of obtaining *hard to solve exactly* LOP instances. Instances in this set range in size from 30 up to 500 elements.

This paper is organized as follows. In Section 2, Relax and Cut is briefly reviewed. In Section 3, Lagrangian Relaxation LOP upper bounds are discussed. Lagrangian based LOP heuristics are introduced in Section 4. In Section 5, some test instances are attempted to be solved to proven optimality with Mixed Integer Programming (MIP) solver CPLEX 7.1 [7]. This is done in order to put computational results for the proposed heuristics (see Section 6) into a more balanced perspective. Finally, some concluding remarks in Section 7 close the paper.

2 Relax and Cut Algorithms

Assume that a (binary 0-1, for simplicity) formulation of a $\mathcal{NP}$-hard combinatorial optimization problem is given. Assume as well that exponentially many inequalities may be included in it. Such a formulation can be generically described as

$$\max\{cx: \ \ Ax \leq b, \ \ x \in X\}, \tag{1}$$

where $x \in \mathbb{B}^m$ is a vector of variables, $c \in \mathbb{R}^m$, $b \in \mathbb{R}^p$, $A \in \mathbb{R}^{p \times m}$ and, finally, $X \subseteq \mathbb{B}^m$ is an associated feasibility region. Assume, as it is customary in Lagrangian relaxation, that

$$\max\{cx: \ \ x \in X\} \tag{2}$$

is an easy (polynomial time) problem to solve. On the other hand, in what is unusual for the application of Lagrangian relaxation, let p be exponential in m. In spite of that, assume one wishes to dualize

$$\{a_i x \leq b_i: \ \ i = 1, 2, \ldots, p\} \tag{3}$$

in a Lagrangian fashion and let $\lambda \in \mathbb{R}^p_+$ be the corresponding vector of Lagrangian multipliers. Subgradient Optimization (SO) could then be used to solve

$$\min_{\lambda \geq 0} \big\{ \max\{(c - \lambda A)x + \lambda b: \ \ x \in X\} \big\}. \tag{4}$$

Optimization is typically conducted here in an interactive way with multipliers being updated so that the optimal value of (4) is attained. For the sake of completeness, let us briefly review how the Subgradient Method (SM) [16], as implemented in [9], attempts to solve (4).

At any given iteration of the SM, for given feasible values of the Lagrangian multipliers λ, let $\overline{x}$ be an optimal solution to

$$\max\{(c - \lambda A)x + \lambda b : \ \ x \in X\}. \tag{5}$$

Denote by z_{ub} the corresponding solution value and let z_{lb} be a known lower bound on (1). Additionally, let $g \in \mathbb{R}^p$ be a vector of subgradients associated with the relaxed constraints. Given an optimal solution $\overline{x}$ to (5), subgradients g are evaluated as

$$g_i = (b_i - a_i\overline{x}), \ \ i = 1, 2, \ldots, p. \tag{6}$$

In the literature (see [9], for instance) Lagrangian multipliers are usually updated by firstly determining a "step size" θ,

$$\theta = \frac{\alpha(z_{ub} - z_{lb})}{\sum\limits_{i=1,\ldots,p} g_i^2}, \tag{7}$$

where α is a real number assuming values in $(0, 2]$. One would then proceed to computing

$$\lambda_i \equiv \max\{0; \lambda_i - \theta g_i\}, \ \ i = 1, \ldots, p, \tag{8}$$

and move on to the next iteration of the SM.

Under the conditions imposed here, a straightforward implementation of updating formulas (6)–(8) is not as simple as it might appear. The reason being the exceedingly large number of inequalities that, typically, would have to be dualized (recall that p is assumed to be exponential in m).

Inequalities in (3), for every SM iteration, may be divided into three groups. The first one contains inequalities that are violated by $\overline{x}$. The second group is for those inequalities that have nonzero multipliers currently associated with them. Notice that an inequality may be, simultaneously, in the two groups just defined. Finally, the third group consists of the remaining inequalities and evaluating their subgradients would account for most of the computational burden at a SM iteration.

One should notice that, under the classification proposed above, inequalities may change groups from one SM iteration to another. It should also be pointed out that the only multipliers that may directly contribute to the Lagrangian costs $(c-\lambda A)$, at any given SM iteration, are the ones associated with inequalities in groups one and two. These inequalities are thus denoted *active inequalities*. Conversely, we denote inequalities in group three *inactive inequalities*. Finally, it is important to stress that, from (8), multipliers for inequalities in group three will not be changing their null values at the end of the current SM iteration.

Clearly, inactive inequalities will not directly contribute to Lagrangian costs (since their multipliers will remain zero valued at the end of the current SM iteration). On the other hand, they do play a decisive role in

determining the value of θ. Typically, for the application above, the number of strictly positive subgradients, amongst inequalities in group three, tends to be huge. Consequently, the value of θ would, typically, result extremely small, leaving multiplier values virtually unchanged from iteration to iteration. Bearing this in mind, one may choose to apply (6)–(8) exclusively to active inequalities. This was suggested in [21, 22] and is implemented here. That results in a dynamic scheme where the set of active inequalities may continuously change. Notice, in association, that an inequality may become active at one given SM iteration, then become inactive at a subsequent one and become, yet again, active at a later iteration.

The scheme suggested above is, in a sense, very much akin to cutting planes generation. It has been firstly proposed and successfully used in [21, 22] for the Steiner Problem in Graphs. Later on, it has been used for the Edge-Weighted Clique Problem in [17], for the Vehicle Routing Problem in [24] and for the Rectangular Partitioning Problem in [4].

The term *Relax and Cut* was coined in [8] for an algorithm that, although different from the one in [21, 22], shares a number of points in common with it. In this paper, we use the term Relax and Cut in a broader sense to denote the whole class of LR algorithms where inequalities are dualized *on the fly* (as they become violated at the solution to a LR subproblem). This class therefore encompasses the algorithm in [21, 22] as well as the one in [8].

3 Lagrangian Relaxation Upper Bounds

We have used the Linear Integer Programming formulation of LOP found in [14]. It involves a set of binary $0-1$ variables $\{x_{ij} : (i,j) \in A\}$. These variables are used to impose an ordering precedence between any two distinct elements i and j of N. Accordingly, for any $i, j \in N$, $i \neq j$, whenever i preceeds j, then both $x_{ij} = 1$ and $x_{ji} = 0$ must hold. Otherwise, $x_{ij} = 0$ and $x_{ji} = 1$ must hold. Consider, in association, a polyhedral region, $\mathcal{R}$, described as

$$x_{ij} + x_{ji} = 1, \text{ for all } i, j \in N, i \neq j \tag{9}$$

$$x_{ij} + x_{jk} + x_{ki} \leq 2, \text{ for all } (i,j),(j,k),(k,i) \in A \tag{10}$$

$$0 \leq x_{ij} \leq 1, \text{ for all } (i,j) \in A. \tag{11}$$

A valid LOP formulation (see [14]) is then given by

$$\max \{ \sum_{(i,j)\in A} c_{ij} x_{ij} : x \in \mathcal{R} \cap \mathbb{Z}^{|A|} \}, \tag{12}$$

where (9) enforces a tournament to be formed while the 3-Cycle inequalities (10) ensure the tournament to be cycle free. Attaching multipliers $\lambda \geq 0$ to

the inequalities in (10) and denoting by $\mathcal{R}_L$ the polyhedral region described by (9) and (11), a valid upper bound for LOP is given by

$$UB(\lambda) = \max \{ \sum_{(i,j)\in A} c'_{ij}x_{ij} + 2 \times \sum_{i,j,k\in N, i\neq j, i\neq k, j\neq k} \lambda_{ijk} : x \in \mathcal{R}_L \cap \mathbb{Z}^{|A|}\}, \tag{13}$$

where $\{c'_{ij} = c_{ij} - \sum_{k\in N\setminus\{i,j\}} \lambda_{ijk} : (i,j) \in A\}$ is the set Lagrangian modified costs. An optimal solution to (13) is obtained by setting to one, for every pair $i, j \in N, i \neq j$, that variable, x_{ij} or x_{ji}, with the largest Lagrangian cost (the remaining variable being set to zero). Ties are broken arbitrarily. The best possible Lagrangian bound for this application is

$$\mathrm{Min}_{\lambda\geq 0} UB(\lambda) \tag{14}$$

and can, in principle, be attained by using Subgradient Optimization methods (see [16] for the Subgradient Method (SM) and [3] for Bundle Methods). Notice however that the $\subset^3_{|N|}$ inequalities in (10) could number as many as $20,708,499$ for an LOP with 500 elements. Dualizing such a large number of inequalities, as explained before, would make the use of a straightforward implementation of, say the SM, quite ineffective. In this case, an acceptable compromise is then reached by using Relax and Cut.

In Section 2 a generic description of a Relax and Cut algorithm for Combinatorial Optimization problems is given. In what follows that algorithm is specialized for LOP.

3.1 Relax and cut for LOP

In our implementation of a Relax and Cut algorithm for LOP only 3-Cycle inequalities (10) are dualized. Preliminary computational results indicated that, at an optimal solution $\overline{x}$ to (13), the number of violated 3-Cycle inequalities is typically very large. This appears in agreement with what has been reported for LP based LOP lower bounding algorithms. For these algorithms 3-Cycle inequalities are normally introduced as cutting planes. Faced with too many violated inequalities at a given reoptimization round, only a fraction of these (i.e. the most violated ones) are normally used as cutting planes [26]. In our context, unfortunately, a similar approach can not be pursued. This applies since all 3-Cycle inequalities that violate $\overline{x}$ do so by the same amount (i.e. one unit).

For an optimal solution $\overline{x}$ to (13), consider all violated 3-Cycle inequalities that currently have a zero valued multiplier associated with them. These inequalities form group one inequalities of Section 2. As pointed out above, a very large number of such inequalities typically exist. In order to keep down the number of active inequalities, only at most $\eta > 0$ group one inequalities per variable are introduced into the set of active inequalities (as

they are identified) at a given SM iteration. Remaining group one inequalities are simply ignored. Acting as suggested above tends to evenly spread active inequalities amongst variables. Overall this appears to be an adequate strategy which does not seem to negatively affect dual bound quality.

Let $\zeta \geq 1$ be the maximum number of iterations allowed for SM. In our computational experiments, the value of parameter α (see Section 2) is initially set at 2.0 and is halved after $\lfloor \zeta/20 \rfloor$ iterations without an overall improvement on the best dual bound (upper bound for LOP) so far generated. All other aspects of our LOP Relax and Cut algorithm follow the generic algorithm of Section 2.

4 Lagrangian based heuristics

Two basic LOP heuristics are introduced next. Their use as stand alone, single pass procedures, is not very attractive. Nevertheless, solution quality tends to be greatly enhanced with the use of *local search* procedures. This overall scheme, i.e. a basic LOP heuristic followed by local search, tends to be enhanced even further by carrying over some dual information into the single pass procedures. Accordingly, primal bound generation is incorporated within the LR framework of Section 3. In such a scheme, LOP heuristics are called, for every iteration of the SM, either under Lagrangian modified costs or else using, directly, LR solutions as an input. At every iteration of the SM, feasible LOP solutions are attempted to be improved through local search.

4.1 Procedure Position Cost

For any set $S \subset N$, denote by $\overline{S}$ the complement of S over N. In association, compute, for every $j \in \overline{S}$, *position costs* $q_j = \sum_{k \in \overline{S} \setminus \{j\}} c_{jk}$. The basic idea behind the first heuristic, denoted *Position Cost* (PC), is to use position costs to sequentially build a linear ordering of the elements of N. Accordingly, at step $p = |S| + 1$ of the procedure (note that S is the empty set for $p = 1$), the elements in $S \subset N$ would have been placed in the first $p - 1$ ordering positions. At this stage, that element j, $j \in \overline{S}$, for which $q_j \geq q_k$, for all $k \in \overline{S} \setminus \{j\}$, is to be placed at the p-th ordering position. The step is concluded by setting $S := S \cup \{j\}$. The procedure is finished after $|N|$ steps and has complexity $O(|N|^2)$.

Procedure PC is called for every SM iteration. Lagrangian modified costs are used to compute position costs. Once a feasible solution is generated (from the position costs), one resorts back to the original LOP costs and applies local improvement procedures (see Subsection 4.3).

4.2 Procedure Node Degree

The second heuristic uses, as input, LR solutions (instead of Lagrangian modified costs). For a given iteration of the SM, let λ be the corresponding set of Lagrangian multipliers and assume $\overline{x}$ to be an optimal solution to (13). Clearly, such a solution must be a tournament, though not necessarily an acyclic one. For this tournament compute, for every node $i \in N$, the out degree $\sum_{(i,j)\in A} \overline{x}_{ij}$ (i.e. the number of tournament arcs pointing outwards of i). Notice that if $\overline{x}$ were an acyclic tournament (and therefore a feasible solution to LOP), exactly one of the nodes in N would have an out degree of $|N|-1$. Accordingly, exactly one other node in N would have an out degree of $l \in \{0, \ldots, |N|-2\}$. The basic idea behind the second heuristic, denoted *Node Degree* (ND), is then to (linear) order the elements of N in decreasing value of their $\overline{x}$ outdegrees (ties are broken arbitrarily). Heuristic ND has complexity $O(|N|^2)$.

Procedure ND is called, at every iteration of SM, for the corresponding solution, $\{\overline{x}_{ij} : (i,j) \in A\}$ of (13).

4.3 Local Search

Let π be a feasible LOP solution. The concept of k-optimality (see [20]) can be applied over π in an attempt to generating better quality LOP solutions. In a $2-opt$ move, two distinct elements of N, say i and j, are to exchange (among themselves) positions in π, provided that results in an overall increase in ordering cost. In a $3-opt$ move, 3 distinct elements of N, say i, j and l, are to exchange (among themselves) positions in π, if an overall increase in ordering cost results. Given i, j and l, only two possibilities of a $3-opt$ exchange exist that are not themselves simply 2-opt moves. Clearly, the concept can be generalized into a $k-opt$ move, $4 \leq k \leq |N|$, involving k distinct elements of N.

Bearing in mind the high computational cost of either 2-opt or 3-opt, one should consider restricting the associated search neighborhoods. In order to illustrate the idea, consider a linear ordering π of N, an element $i \in N$ (placed in position $\pi^-(i) = p, 1 \leq p \leq |N|$) and an integral valued parameter $\delta > 0$. In a restricted 2-opt move, one searches for candidate elements to exchange (ordering) positions with i in the range of positions defined by $[\max\{1, p-\delta\}, \min\{|N|, p+\delta\}]$. Amongst competing candidates, that element j leading to the largest overall increase in (linear ordering) cost is chosen for the exchange. Search neighborhoods for 3-opt moves, involving element $i \in N$, are defined likewise.

Another improvement procedure, denoted here *sequence shifts*, has proven a better alternative to either 2-opt or 3-opt (in terms of both CPU time and bound quality). Assume that a linear ordering π of N is given and consider an element $i \in N$ such that $\pi^-(i) = p$, $1 \leq p \leq |N|$. The basic idea of a se-

quence shift operation is to find the best position in π to move element i to. As an illustration, assume that an overall increase in ordering cost results if i is moved to, say position $(p+k)$, where $k > 0$. On implementing this move, the sequence of elements in positions $(p+1)$ through to $(p+k)$ must be shifted one position backwards. Accordingly, if i is moved, instead, to position $(p-k)$, the sequence of elements placed in positions $(p-k)$ through to $(p-1)$ must be shifted one position forwards.

Sequence shifts can be implemented quite efficiently by dividing the associated search neighborhood in two. The first half involves those positions located ahead of p. The second one is for positions placed behind p. Let $d(\pi)$ be the cost of π and consider first the case where one is searching amongst positions located ahead of p. For the simplest possible scenario, assume that element j is currently placed at position $(p+1)$. Moving i one position forwards to $(p+1)$ while shifting j one position backwards to p, leads to a new ordering π^1 with a cost $d(\pi^1) = d(\pi) - c_{ij} + c_{ji}$. Consider now the option of moving i, not to position $(p+1)$, but to position $(p+2)$ (where element l is currently assumed to be placed). This action would result in a shift of the sequence of elements placed in positions $(p+1)$ and $(p+2)$, one position backwards (j being moved to p while l is moved to $p+1$). Denoting by π^2 the resulting ordering, notice that $d(\pi^2) = d(\pi^1) - c_{il} + c_{li} = d(\pi) - c_{ij} - c_{il} + c_{ji} + c_{li}$. In general terms, assume that i is to be moved $k \geq 2$ positions ahead, to a position currently occupied by l. The cost $d(\pi^k)$ of the resulting ordering π^k can then be computed as $d(\pi^k) = d(\pi^{k-1}) - c_{il} + c_{li}$. Sequence shifts are thus computed, one at a time, using as input the results from the previous operation. It should be noticed that a procedure, similar to the one outlined above, can be devised for a search involving positions located behind p. Among all possibilities investigated, that position (if any) leading to the largest overall increase in ordering cost is chosen for i to be moved to. Implemented as outlined above, sequence shifts are computationally cheap and no restriction on the associated search neighborhood need be imposed.

The idea of sequence shifts can be generalized by attempting to move, in one piece, two or more contiguously placed elements of an ordering π. The implementation suggested above for single element moves can be readily adapted for this general case.

A combination of heuristic PC with 2-opt moves has been used in [14]. Single element shifts, under a different name, have been used in [5, 18].

For the computational results in Section 6 only sequence shifts are used. Moves involving up to seven contiguously placed elements are implemented.

5 Test Instances

In order to test the proposed heuristics, three different sets of LOP instances are used. Two of these come from the literature while the third one is introduced in this study. Solver CPLEX 7.1 is used in an attempt to solve some of these instances to proven optimality. CPLEX 7.1 generates, if necessary, Gomory cuts [12] at the root node of the enumeration tree. As a result, the bound thus produced dominates the one given by the LP relaxation of (12).

Before moving on to computational experiments, it is important to highlight a detail which is associated with LOP duality gaps. As we shall explain next, percentage duality gaps for LOP, as formulated in (12), are normally much smaller than the duality gaps derived from a very natural reformulation of (12). Indeed duality gaps associated with this reformulation can be argued to be more acceptable parameters to measure problem difficulty against.

Let LOP be modeled as in (12). Notice from (9) that variables x_{ji} with $j > i$ can be rewritten as $x_{ji} = 1 - x_{ij}$. In this process, objective function (12) becomes

$$\sum_{i,j \in N, i<j} c_{ji} + \sum_{i,j \in N, i<j} (c_{ij} - c_{ji}) x_{ij} \tag{15}$$

where

$$\sum_{i,j \in N, i<j} c_{ji} \tag{16}$$

is a constant. Typically (16) tends to be a very large proportion of overall optimal LOP solution values. In general, LP based exact solution approaches for LOP use the variable reformulation above. As a result, percentage duality gaps between optimal solution values and their LP relaxation counterparts tend to be masked by constant (16). Duality gaps that include (16) represent an "underestimate" of the inherent difficulty of solving exactly an LOP instance. It thus appears more appropriate to compute duality gaps that exclude constant (16). In this section, duality gaps are computed both ways. This, in our view, gives a more balanced account of the true difficulty of solving a given LOP instance to proven optimality. Instances acknowledged as being difficult to solve exactly are prime candidates for applying non exact solution approaches.

5.1 The LOLIB instances

The first test set contains the 49 instances from LOLIB [28]. LOLIB instances relate with input-output matrices compiled for the economies of different European countries. For this test set $44 \leq |N| \leq 60$.

Solver CPLEX, under default settings (except for parameter *mingap* which was lowered to 10^{-6}), was used in an attempt to generating proven

$\|N\|$	b&b nodes	time (secs)
48.1	1.08	3.99

Table 1: CPLEX statistics for LOLIB instances: average results

optimal solutions for all LOLIB instances. LOP upper bounds are generated from the LP relaxation of (12) (all 3-Cycle inequalities included right from the start) reinforced, if necessary, with Gomory cuts. Table **??** in Appendix **??** shows the results obtained. Entries on that table give, respectively, instance identification, number of elements in N, optimal solution value, percentage (duality) gap between LP relaxation and optimal solution value, percentage duality gap excluding constant (16), number of branch and bound nodes and CPU times (in seconds of a 933 Mhz Pentium III based machine, under 512 Mb of RAM memory). Average results are shown on Table 1.

As one may appreciate from the results on Tables 1 and **??**, all LOLIB instances are solved exactly under fairly low CPU times. With the exception of only two instances, remaining ones are solved at the root node of the enumeration tree. Furthermore, the two instances possessing a duality gap (after Gomory cuts were introduced) required only three branch and bound nodes to have optimality proven. As far as the LP relaxation of (12) is concerned, only 5 LOLIB instances do not have naturally integral relaxations.

In view of the results quoted above, one may conclude that the LOLIB instances are *quite easy* to solve exactly.

5.2 Instances proposed by Mitchell and Borchers

The second test set contains instances randomly generated in [26] to resemble input-output matrices. The 30 instances in this set range in size from 100 up to 250 elements. Attempting to solve them exactly, just as done before for the LOLIB instances, although possible, does not seem a very attractive proposition. Model (12), even after the variable reformulation suggested above, would involve a large number of $3-Cycle$ inequalities. In this case, a more appropriate course of action would be to initially exclude all $3-Cycle$ inequalities from the LP relaxation of (12) and only introduce them, as cutting planes, once they become violated. This is done in [26] (in a more elaborated way) where an Interior Point LP solver is combined with CPLEX 4.0 to obtain the LP relaxation of (12). This combination has proved a much more effective alternative to either the Interior Point algorithm or CPLEX 4.0 acting in isolation.

In what follows we quote, mostly, the computational results in [26]. These were obtained on a Sun SPARC 20/71 machine (CPU times in sec-

onds). All instances were solved to proven optimality. Only instances r100b2 and r200d1 do not possess a naturally integral LP relaxation of (12). Solution times ranged from 65 seconds for instance r150e0 up to 1666 seconds for instance r200d1. Duality gaps for r100b2 are 0.0043%, including (16), and 0.0136%, excluding (16). Corresponding figures for r200d1 are, respectively, 0.00039% and 0.0014%.

As one may appreciate from the results quoted above, instances in [26] do not differ fundamentally from the LOLIB ones. They invariably have very strong LP relaxations of (12). Duality gaps exist for only a very small proportion of the instances involved and are not very large. On the other hand, instances in [26] have larger dimensions. That and also having to deal with highly degenerated LP relaxations of (12) constitute the main challenges in solving the Mitchell and Borchers instances to proven optimality. Overall, these instances can be considered much more difficult to solve exactly than the LOLIB ones.

5.3 Randomly generated LOP instances

Based on the results above, duality gaps appear to be almost always equal zero for input-output matrices (or for instances randomly generated to resemble them). By contrast, instances to be introduced next tend to have *large* duality gaps and appear to be *difficult* to solve exactly. This applies even for $|N| \leq 50$. They are generated as follows. One firstly defines the number n of elements in N, an integral valued perturbation parameter $\epsilon > 0$ and the fraction $\mu \in (0, 1]$ of variables with nonzero costs. Then, n distinct points, within a square of side l (lying on the Euclidian plane), are randomly generated. Each of these points is associated with a different element of N. Euclidian distances are then computed for every pair of points. Let $d(i, j)$ be the (rounded off) Euclidian distance for points associated, respectively, with elements i and j. Precedence costs c_{ij} and c_{ji} are then computed as $c_{ij} = d(i, j) + \epsilon_{ij}$ and $c_{ji} = d(i, j) + \epsilon_{ji}$, where ϵ_{ij} and ϵ_{ji} are randomly generated in the range $[0, \epsilon]$. The fraction $(1 - \mu)$ of variables with zero valued costs is sequentially met as follows. Assume a pair of distinct points i and j is randomly chosen. If fraction $(1 - \mu)$ has not yet been reached, cost c_{ij} is set to 0 and the procedure is repeated. Otherwise, a valid LOP instance has been generated.

Test instances are generated by setting $l = 100$, $\epsilon = 30$ and using 3 different sets of values for $(1 - \mu)$. Series A instances have $(1 - \mu) = 0.6$. Series B instances have $(1 - \mu) = 0.4$. Finally, series C instances have $(1 - \mu) = 0.2$.

Once again CPLEX 7.1 is used in an attempt to generating optimal solutions to the instances described above. Table 2 shows the results obtained. Only those instances with $|N| = 30$ were found to be easy to solve exactly. Instances with as little as $|N| = 50$ turned out unexpectedly hard

instance	$\|N\|$	optimal	% gap1	% gap2	b&b nodes	time (secs)
e30A.mat	30	17490	0.0	0.0	1	1.49
e30B.mat	30	18116	0.0	0.0	1	4.78
e30C.mat	30	25369	0.0	0.0	1	1.31
e50A.mat	50	46439	0.67	3.52	759	70018
e50B.mat	50	$\leq$ 58656	$\leq$ 1.51	$\leq$ 9.7	$>$ 700	$>$ 128346

Table 2: CPLEX statistics for Euclidian instances

to solve. Experiments for larger instances indicate that difficulty appears to increase with the increase in instance dimension. It is thus reasonable to conclude that (12) must be strengthened with additional families of strong valid inequalities if optimality is to be attained for instances with $|N| \geq 50$.

From the computational evidence obtained, one may conclude that LOP instances introduced in this study are *hard to solve exactly*. They therefore constitute a good test bed for LOP heuristics.

6 Computational Experiments

Feasible LOP solutions are obtained by firstly defining the maximum number of iterations allowed for the SM. Experiments were conducted, in separate, for each one of the two proposed heuristics. For all test instances with $|N| < 300$, heuristics are called for every Relax and Cut iteration, for runs with ζ fixed at, respectively, 1, 200, and 3000 (see Subsection 3.1). One additional run with $\zeta = 400$ was implemented only for heuristic ND. For instances with $|N| \geq 300$, heuristics are only called whenever an overall improvement on the dual bound occurs during a Relax and Cut run. In all experiments η is initially fixed at 15 (see yet again Subsection 3.1). Once a gap of less than one percent is reached between upper and lower bounds, that value is set to $\eta = 100 + \lfloor \zeta/100 \rfloor$. Obviously, whenever LOP optimality could be proven before the iterations limit is reached, the run was immediately stopped. Given that, for all test instances, precedence costs are integral valued, one possibility for proving optimality is to reach a difference of less than one unit between upper and lower bounds. Another (more rare) possibility arises when, at a given iteration of the SM, an acyclic tournament turns out to be an optimal solution to (13). In this case, if the corresponding Lagrangian upper bound equals the tournament cost under the original precedence costs, optimality is again proven.

Tables 3 through 8 summarize the results obtained. Tables 3 and 4 are for the LOLIB instances. Tables 5 and 6 are for the instances introduced in [26]. Finally, Tables 7 and 8 are for the instances proposed in this study. For any of these tables, entries give, respectively, the maximum number of SM iterations allowed, the average percentage gap between optimal solution values and best primal bounds generated, number of instances for which

optimal feasible solutions are obtained, number of instances for which optimalty was proven, and the average CPU times involved (in milliseconds of a Pentium III 933 MHz machine). All the coding was done in C++. For the test set introduced in this study optimal solution values are known for only 4 instances. Some experiments were then conducted with heuristic ND for $\zeta = 20000$ in order to generate *high quality* feasible solutions for these instances. Feasible solutions thus obtained replace optimal solution values, whenever applicable, for the statistics quoted above.

Tables **??** through **??** detail the statistics from which Tables 3 through 8 are compiled. Entries on these tables differ from single iteration SM runs to 200 and 3000 iterations runs. For single SM iteration runs, entries give, respectively, instance identification, number of elements in N, Relax and Cut upper bound, Relax and Cut percentage gap from optimal solution value, heuristic bound, heuristic percentage gap from known optimal solution value, and CPU time in milliseconds. For the remaining tables, entries give, respectively, instance identification, Relax and Cut upper bound, maximum number of active 3-cycle inequalities throughout the SM run, heuristic bound, heuristic percentage gap from optimal solution value, number of times an optimal solution is generated for the SM run, and CPU time in seconds. As mentioned before, best known feasible solution values for the test instances introduced in this study are used instead of optimal solution values, whenever applicable.

The two different heuristics appear to perform very well for the LOP instances considered. However ND appears to have a clear edge over PC (generates better quality LOP solutions in less CPU time). In particular, ND (under $\zeta = 200$) was capable of generating optimal solutions for all LOLIB instances. For the parameter settings used, $\zeta = 400$ is a very good upper bound on the least number of Relax and Cut iterations necessary to obtain similar results for the instances in [26]. In relation with the instances introduced in this study, little can be said. Optimal solution values are known for only 4 out of the 21 instances involved. Furthermore, duality gaps appear to be very large and therefore Lagrangian relaxation upper bounds ought to be far from corresponding optimal solution values. For the 4 instances where optimal solutions are known, ND with $\zeta = 3000$ managed to find them all.

Dual bounds generated by the Relax and Cut algorithm also proved very good. They equaled optimal LOP solution values (rounding off dual bounds if necessary) for 72 out of the 79 combined instances of LOLIB and [26]. Each of the seven instances for which Relax and Cut could not prove optimality do possess a duality gap (between the LP relaxation of (12) and corresponding optimal solution values). Therefore best possible Lagrangian bounds (14) are known to be above optimal solution values anyway and thus LR alone is not enough to reach optimality.

In the computational experiments, at most $6,560$ inequalities, out of a

iterations	% PC-gap	nb. opts	nb. opt. proofs	time (ms)
1	0.07154	17	0	39
200	0.01028	41	1	1508
3000	0.00443	42	37	8254

Table 3: LOLIB instances: PC

iterations	% ND-gap	nb. opts	nb. opt. proofs	time (ms)
1	0.09933	15	0	37
200	0.00000	49	3	1432
3000	0.00000	49	44	6157

Table 4: LOLIB instances: ND

maximum of over $34,220$ inequalities in (10), were explicitly dualized, at any iteration of Relax and Cut, for any of the LOLIB instances. For the instances in [26] the corresponding figures are, respectively, $94,442$ and $2,576,000$. Finally, for the newly introduced instances, the figures are $210,000$ and $20,708,499$.

7 Conclusions

Two Lagrangian based LOP heuristics are proposed in this study. Heuristics are embedded within a Lagrangian Relax and Cut framework where some local search procedures are also used. Very good quality solutions have been generated for LOP instances associated with input-output matrices. Computational experiments with some new, randomly generated LOP instances (introduced in this paper), appear to indicate these instances to be *hard to solve exactly*.

The proposed heuristics present some features that may be used to advantage. In particular, dual bounds are generated together with primal ones. Combining these two bounds, an indication of solution quality is obtained. Furthermore, primal bounds appear to benefit from the good quality dual bounds that are generated. In this respect, a point to mention is the fact that additional families of valid LOP inequalities can be incorporated into

iterations	% PC-gap	nb. opts	nb. opt. proofs	time (ms)
1	0.01622	0	0	171
200	0.00163	9	6	29732
3000	0.00118	17	17	257367

Table 5: Instances from [26]: PC

iterations	% ND-gap	nb. opts	nb. opt. proofs	time (ms)
1	0.01219	2	0	209
200	0.00034	24	10	25763
400	0.00000	30	21	35602
3000	0.00000	30	28	54013

Table 6: Instances from [26]: ND

iterations	% PC-gap	nb. opts	nb. opt. proofs	time (ms)
1	0.55591	1	0	420
200	0.13129	1	0	64862
3000	0.06128	2	2	1009084

Table 7: New instances: PC

our Relax and Cut framework. Indeed, as explained in Section 2, exponentially many such inequalities may be treated as suitable candidates to Lagrangian dualization. Since a number of different families of facet defining inequalities for the LOP polytope have already been proposed in the literature, a potential exists for further improvements in the dual bounds. Such an improvement, in turn, should have a positive impact on primal bound quality.

It appears that the best LOP heuristics in the literature are the ones in [18] and [13]. A direct comparison with our best heuristic, i.e. ND, appears difficult to carry out. For instance, LOLIB is the only common test set for all three algorithms. In terms of CPU times, [13] appears to quote only the times for runs where best solutions are obtained (and not overall CPU times). From our part, overall CPU times are quoted. Overall CPU times appear to be quoted in [18] as well.

In a very superficial comparison, for the LOLIB intances, ND, together with [13], managed to find optimal solutions for all instances involved. On the other hand, [18] found optimal solutions for 47 out of 49 instances. Changing the focus to CPU times, [18] appears to have an edge over ND (and almost certainly over [13]).

iterations	% ND-gap	nb. opts	nb. opt. proofs	time (ms)
1	0.63228	0	0	392
200	0.08788	2	0	63955
3000	0.03622	6	2	989603

Table 8: New instances: ND

References

[1] A. Belloni and A. Lucena. Lagrangian Based Heuristics to the Linear Ordering Problem. *Proceedings of MIC'2001-4th Metahruristics International Conference*, pp. 445–450, Porto, Portugal, 2001.

[2] G. Bolotashvili, M. Kovalev, and G. Girlich. New facets of the linear ordering polytope. *SIAM J. Discrete Math.*, 3:326–336, 1999.

[3] J.F. Bonnans, J.Ch. Gilbert, C. Lemaréchal, and C. Sagastizábal. *Optimisation numérique: aspects théoriques et pratiques*. Springer Verlag, 1997.

[4] F. Calheiros, A. Lucena, and C. de Sousa. Optimal Rectangular Partitions. relatório técnico, Laboratório de Métodos Quantitativos, Departamento de Administração, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, 2001.

[5] V. Campos, M. Laguna, and R. Marti. Scatter Search for the Linear Ordering Problem. Graduate School of Business Report, University of Colorado, Boulder, CO 80309, USA, 1998.

[6] S. Chanas and P. Kobylánski. A New Heuristic Algorithm Solving the Linear Ordering Problem. *Computational Optimization and Applications* 6:191–205, 1996.

[7] CPLEX 7.1 ILOG, Inc. CPLEX Division, 2001.

[8] L. Escudero, M. Guignard, and K. Malik. A lagrangian relax and cut approach for the sequential ordering with precedence constraints. *Annals of Operations Research*, 50:219–237, 1994.

[9] M.L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Mangement Science*, 27:1–18, 1981.

[10] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publisher, 1997.

[11] F. Glover. A Template for Scatter Search and Path Relinking. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, *Artificial Evolution*, Lecture Notes in Computer Science 1363, Springer, pages 13–54, 1998.

[12] R.E. Gomory. An Algorithm for the Mixed Integer Problem. RM-2597, The Rand Corporation, 1960.

[13] C.G. González and D. Pérez-Brito. A Variable Neighborhood Search for Solving the Linear Ordering Problem. *Proceedings of MIC'2001-4th*

Metahruristics International Conference, pp. 181–185, Porto, Portugal, 2001.

[14] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195-1220, 1984.

[15] M. Grötschel, M. Jünger, and G. Reinelt. Facets of the linear ordering polytope. *Mathematical Programming*, 33:43-60, 1985.

[16] M. Held, P. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6, pages 62–88,1974.

[17] M. Hunting, U. Faigle, and W. Kern. A Lagrangian relaxation approach to the edge-weighted clique problem. working paper, Department of Applied Mathematics, Twente University, 1998.

[18] M. Laguna, R. Marti, and V. Campos. Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem. Graduate School of Business Report, University of Colorado, Boulder, CO 80309, USA, 1998.

[19] J. Leung and J. Lee. More facets from fences for linear ordering and acyclic subgraphs polytopes. *Discrete Applied Mathematics*,50:185–200, 1994.

[20] S. Lin and B.W. Kerningham. An effective heuristic for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[21] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting-planes. *COAL Bulletin*, 21:2–8, 1992.

[22] A. Lucena. Tight bounds for the Steiner problem in graphs. *Proceedings of NETFLOW93*, pages 147–154, 1993.

[23] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[24] C. Martinhon, A. Lucena, and N. Maculan. A relax and cut algorithm for the vehicle routing problem. relatório técnico, Laboratório de Métodos Quantitativos, Departamento de Administração, Universidade Federal do Rio de Janeiro, 2000.

[25] A. McLennan. Binary Stochastic Choice, in *Preferences, Uncertainty and Optimality*, J. S. Chipman, D. McFadden, and M. K. Richter, eds, Westview Press, Boulder, 1990.

[26] J.E. Mitchell and B. Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm, Rensseler Institute Report, Troy, NJ 12180, USA, 1997.

[27] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[28] G. Reinelt. LOLIB http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/LOLIB/LOLIB.html.

[29] G. Reinelt. The Linear Ordering Problem: Algorithms and Applications. *Res. Exp. Math. 8*, Heldermann Verlag, Berlin, 1985.

[30] G. Reinelt. A note on samll linear ordering polytope. *Discrete Computational Geometry*, 10:67–78, 1993.

[31] H. Wessels. Triagulation und Blocktriangulation von Input-Output-Tabellen. *Deutches Institut für Wirtschaftsforschung*, Heft 63, Berlin, 1981.