CORE DISCUSSION PAPER
2005/XX

LS-LIB: a Library of Reformulations, Cut Separation Algorithms and Primal Heuristics in a High-Level Modeling Language for Solving MIP Production Planning Problems

Yves POCHET[1], Mathieu VAN VYVE[2] and Laurence A. WOLSEY[3]

June 2005

Abstract

Much progress has been made in recent years in solving certain classes of production planning problems using mixed integer programming. One of the major challenges is how to make this expertise available and easy to use to the non-specialist and to the practitioners. Here we describe a modeling approach and tool LS-LIB, and report on computational results.

LS-LIB is a library of primitives to declare procedures/subroutines/global constraints in a high-level modeling language that we believe offers an interesting partial answer to this challenge. LS-LIB provides routines for problem reformulation, cut generation, and heuristics to find good feasible solutions quickly. The user must provide an initial formulation of his problem in the modeling language MOSEL. Then using his knowledge of the problem he must first classify each product or sku according to a simple three field scheme: [production type, capacity type, variant] proposed recently. Then it is a simple matter to use the global constraints of LS-LIB by adding a few lines to his initial MOSEL formulation to get a tightened formulation and/or call the appropriate cut separation routines. The heuristic procedures are called in a similar fashion.

We illustrate the use of LS-LIB on an intractable two-level problem, and a hard multi-level problem.

Keywords: Lot-Sizing, Production Planning, Mixed Integer Programming, Modeling Language, Extended Formulation, Cut Separation, Heuristics.

[1]Center of Operations Research and Econometrics (CORE) and Institut d'Administration et de Gestion (IAG), Université catholique de Louvain. 34, Voie du Roman Pays. 1348 Louvain-la-Neuve, Belgium. Email: pochet@core.ucl.ac.be

[2]Département d'Informatique, Université Libre de Bruxelles, Bruxelles, Belgium. Email: mathieu.van.vyve@ulb.ac.be

[3]Center of Operations Research and Econometrics (CORE) and Département d'Ingénierie Mathémathique (INMA), Université catholique de Louvain. 34, Voie du Roman Pays. 1348 Louvain-la-Neuve, Belgium. Email: wolsey@core.ucl.ac.be

This text presents research results of the Belgian Program on Interuniversity Poles of Attraction initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. The scientific responsibility is assumed by the authors.

1 Motivation

Much progress has been made in recent years in solving certain classes of production planning problems using mixed integer programming. One of the major challenges is how to make this expertise available and easy to use to the non-specialist. In this paper we present a library LS-LIB of primitives used to declare subroutines and global constraints in a high-level modeling language that we believe offers an interesting partial answer to this challenge.

We suppose that the user has a new production planning problem to solve, either on a one-off or regular basis, and that he believes that mixed integer programming is an appropriate approach. The scenario is typically as follows:

- he builds an initial MIP model and, using a modeling and optimization language,
- he runs his MIP branch-and-cut system (commercial or other) in default mode. Either he is happy with the results (end of the story), or
- the results are unsatisfactory (i.e. finding reasonable solutions takes far too much time, and/or nothing is known about the quality of the solutions because the dual bounds are too weak).

He then has several options:

a) try different parameter settings of the MIP system,

b) identify relaxations for which "good/improved" formulations or cutting planes are known and either use them to modify his initial model or to code separation routines so as to get stronger linear programming bounds, and hopefully faster MIP solution times or better solutions,

c) use LP-based (or other) heuristics to find good feasible solutions quickly.

He then resolves the MIP, and iterates through a), b) and c) several times, until an appropriate solution strategy is found.

LS-LIB provides primitives for problem reformulation, cut generation, and heuristics to find good feasible solutions quickly, and this addresses steps b) and c). The user must provide an initial formulation of the problem in the modeling language MOSEL. Then using his knowledge of the problem he first must classify each product or sku according to a simple three field scheme (production type, capacity type, variant) proposed recently in Wolsey [38]. Then it is a simple matter to use the new primitives offered by LS-LIB to get a tightened formulation and/or call the cut separation routines. The heuristic procedures are called in a similar fashion.

The research on reformulations and cutting planes for single item lot-sizing problems has involved numerous researchers over the last 30 years. Among others, significant contributions on reformulations have come from Krarup and Bilde [15], Eppen and Martin [12], Pochet and Wolsey [21, 22], Van Hoesel and Kolen [31], Van Vyve [35], and on valid inequalities from Barany, Van Roy and Wolsey [5], Van Hoesel, Wagelmans and coauthors [32, 30], Pochet [19], Constantino [9, 10], Atamturk and Munoz [3]. Surveys of this area up to 2000 can be found in Pochet and Wolsey [23] and Pochet [20].

In addition to these polyhedral results, the recent classification scheme of Wolsey [38] is an attempt to provide easy access to this literature, and to the useful results available, and is crucial to the correct use of LS-LIB. A second important factor for LS-LIB is the development of approximate reformulations which lead to smaller linear programs without significant weakening of the bounds, see Stadtler [27] and van Vyve and Wolsey [36].

Other research has dealt with the treatment and solution of multi-item and multi-level problems. An important aspect of the solution of multi-level problems is the use of echelon stocks, see Clark and Scarf [8], and their use in Afentakish and Gavish [1]. The use of this research to provide an effective solution approach for a variety of problems using mixed integer programming has been demonstrated in Belvaux and Wolsey [7].

There is also an extended literature on heuristics for multi-item lot-sizing, based on Lagrangian relaxation, column generation, etc., see for instance Tempelmeier and Derstoff [29]. Work on heuristic approaches for lot-sizing problems based on MIP and improved formulations is limited, one exception is the work of Stadtler [28] on relax-and-fix heuristics. However new local or neighborhood search heuristic ideas for general MIP have appeared recently, see Fischetti and Lodi [13] and Dana et al. [11].

The outline of the paper now follows. In Section 2 we discuss the reformulation approach for production planning problems modeled as MIPs, and the iterative solution process outlined above. We describe and analyze the current tools available to support the process. Based on this discussion, we list system requirements that are necessary for a new high level modeling tool.

In Section 3 we present LS-LIB: the facilities offered, how they are implemented and how they satisfy the requirements listed at the end of Section 2.

In Section 4 we present computational results using LS-LIB for an untractable two-level problem, and a hard multi-level problem.

Finally in an appendix we present in detail the use of LS-LIB on a simple multi-item problem Clorox that has already appeared in the literature. We describe the approach the non-expert would follow to solve the problem with the aid of LS-LIB. He starts with the MOSEL formulation of the original problem, and classifies all the items produced. Given this, a check of the LS-LIB Tables tells him what is known and available as reformulations or cuts. We show how he modifies his MOSEL formulation to include the reformulations/cuts he wants, and the results obtained. A development of the production planning context, including an introduction to mixed integer programming, the application of mixed integer programming to lot-sizing problems, the classification of lot-sizing problems, the theoretical results on algorithms, valid inequalities and formulations, as well as several detailed case studies can be found in Pochet and Wolsey [24].

2 The MIP solution process for production planning problems

In general, MIPs are solved using reformulation techniques leading to branch-and-cut algorithms. Reformulation with new constraints, or both new constraints and new variables, is typically done in two phases:

1. Identification of a relaxation of the problem,
2. Derivation of valid inequalities or extended reformulations for the relaxation.

The strength of the reformulation obtained depends, on the one hand on the closeness of the relaxation relative to the original problem, and on the other hand on the strength of the valid inequalities or extended formulations for the relaxation itself.

The applicability of this *reformulation procedure* depends essentially on the availability of relaxations for which theoretical results (i.e. valid inequalities or extended reformulations) are known. Consequently, research has focussed on the study of simple mathematical structures that constitute strong relaxations of classes of MIPs. A useful distinction is between low-level relaxations, which are useful for general MIPs, and high-level relaxations, which contain more structure and give rise to more powerful reformulations or cuts, but are only applicable to specific problem classes.

The most important low-level relaxations are the simple mixed integer set and the associated MIR inequality, see Nemhauser and Wolsey [17], and the flow cover set and inequalities, see Padberg et al. [18], Van Roy and Wolsey [25], and Gu at al. [14]. These constitute the major reformulation building blocks of the best commercial MIP solvers such as XPRESS or CPLEX. We suppose here that these low-level inequalities are added automatically by the MIP solver used, and thus they do not constitute as such the main topic of this text.

High-level relaxations for production planning problems include, among others, a wide variety of single-item lot-sizing models for which many results concerning reformulations and cutting planes are known.

So we are interested in the development of software or tools to help in using high-level relaxations to solve complex production planning problems.

Many approaches can be taken to develop or implement a branch-and-cut system for a class of (production planning) problems, depending on

- whether the class of problems to be handled by the new system is large or narrow,
- the programming/modeling tools used by the system developer, and
- the division of the work between the end-user and the system.

A general MIP solver, such as XPRESS or CPLEX, is designed to handle the whole range of MIP problems. C/C++ library versions of these solvers are available, which makes it possible to code specific separation routines and/or heuristics. The main drawback of this approach is the very low-level, unstructured information about the model that is available to the system: just linear and integrality constraints. While this is sufficient to specify the model, this makes it very hard to (automatically) recognize the structure of the problem (multi-level? capacitated? with set-up times?). This is redundant work, as the modeler has this higher level and structured information at the time he writes the model (at least privately).

Recognizing this has led to the development of specialized versions of the modeling language itself. This gives access directly to the information contained in the model. Data, variables and constraints are accessible through their names and indexing sets. Thus some structural information contained in the model is available for the branch-and-cut system, making the automatic identification of high-level relaxations much easier. BC-PROD, a branch-and-cut system for production planning, makes use of this mechanism through modeling conventions, see Belvaux and Wolsey [6]. For example, $x(i, k, t)$ is the reserved name for the decision variable representing the amount of item i produced on machine k in period t.
There are several problems associated with this modeling convention approach.

- It is nearly unavoidable that soon after the development of the system, interesting new problems will arise that do not fit into the modeling conventions. For example, a multi-site problem would require a fourth index for the production variable so as to indicate the site.
- In an industrial environment, it is not easy to impose modeling conventions. In particular, rewriting an existing model is costly and error-prone, and different naming conventions might already be in place. Thus modeling conventions might themselves be problematic.
- What to do with the relaxations that are detected is left entirely to the system. For example, suppose that a single-item lot-sizing subproblem with n time periods, constant capacity and backlogging is identified. There exists an extended formulation of size $O(n^3)$ variables and constraints for this mixed-integer set. However, it might be computationally better to add the weaker, but more compact, extended formulation for the uncapacitated version of the same problem ($O(n)$ variables and $O(n^2)$ constraints). Our experience is that this kind of decision is very hard to automate, and is better seen as the outcome of the trial-and-error process a)-b)-c) outlined in the introduction.

Our general critique so far is that the modeler/developer has information about the structure of the problem that is potentially useful to the optimizer. But passing this information through modeling conventions is not always convenient or possible. On the other hand, he does not acquire (or does not want to acquire) the technical or scientific knowledge about separation algorithms, extended formulations or heuristic design, and therefore he is unable to exploit this structural information directly.
Below we give our choice of desirable features for an efficient branch-and-cut system for production planning.

- The end-user is responsible for building the initial model and formulation, and is therefore the best person to identify sub-structures contained in the model, and their corresponding relaxations.
- The end-user should be able to control what extended formulations are added, and what separation routines are used based on the relaxations that he has identified.
- The decision to call heuristics should be easily integrated into the system but left to the user.
- The writing of complex extended formulations, heuristics or separation routines should be done by the system.
- The system should be computationally efficient and easy-to-use.
- The system should be easy to extend, upgrade and maintain.

3 LS-LIB

In this section, we discuss what LS-LIB does and how it is implemented. The general idea is to provide new primitives allowing the modeler to declare global constraints. These primitives essentially define relaxations for which theoretical knowledge is available. The new primitives are implemented as procedures.

The notion of global constraint used here is very close to what is referred to as a global constraint in the Constraint Programming (CP) community. In CP, global constraints are also structures or sets of linear and non-linear constraints linking specified variables. This information is used during enumeration to reduce the range of feasible values of these variables at a node. Here, global constraints are modeled automatically as a set of linear and non-linear (integrality) constraints linking specified variables, and are used to

- strengthen the model with extended formulations, or to
- generate valid inequalities during the branch-and-bound enumeration.

The approach that we advocate would not be possible without the facilities offered by the recently developed high-level modeling language MOSEL that allows the integration of programming, modeling and optimization. Whether a similar library could be developed in other modeling languages such as OPL, AMPL, LINGO or GAMS merits further investigation.

The features of MOSEL that are essential for LS-LIB are the following:

- MOSEL is a typed language. Elementary types are boolean, integers, reals, strings. Compound types are arrays, sets. Optimization types are variables, linear expressions and linear constraints.
- MOSEL is a modeling-and-programming language. Basic programming constructs such as assignments, loops, conditional statements, procedures/functions and includes are available.
- MOSEL is fully interfaced with the XPRESS-MP optimizer. In particular, XPRESS callbacks for branch and cut are directly available in MOSEL.

LS-LIB appears to the end-user as a collection of three types of procedures, for the addition of extended formulations, cutting planes using separation routines, and primal heuristics, respectively.

3.1 Extended Formulations and Cutting Planes

For many problem variants of single-item lot-sizing problems, there exist better or tight formulations, either *extended formulations* involving new variables and constraints, or *cut formulations* involving additional constraints in the original variables added as cuts during the optimization process.

Practically the existence of a tight extended formulation of size $O(n^2)$ variables and constraints for a single item problem with n time periods means that there is a linear program of this size whose solution, for any linear objective function and data, always solves the single item problem.

Adding such an extended formulation is done through the call of a MOSEL procedure. The type of relaxation (e.g. uncapacitated single-item lot-sizing with backlogging) to be declared is encoded in the name of the procedure and the choice (if any) of extended formulation (e.g. multi-commodity or shortest path) is indicated by an additional integer. The arguments specify the variables (stock, backlog, production, set-up) involved and the data defining the instance (demands, capacity). The last three arguments are always the size of the instance (e.g. the number of periods), the approximation parameter (controlling the tradeoff between the size of the reformulation and its tightness, see van Vyve and Wolsey [36]), and a boolean specifying if the constraints of the extended formulation should be added as model cuts. For example, the call

```
XFormWWUSCB(S,R,Y,Z,W,D,n,k,false)
```

declares a single-item Wagner-Whitin cost uncapacitated lot-sizing problem with backlogging and start-ups/switch-offs over n times periods with demand vector D, where S is the vector of stock variables, R is the vector of backlog variables, Y are the setup variables, and Z and W are the start-up and switch-off variables. The procedure adds an appropriate extended formulation of size $O(n)$ variables and $O(kn)$ constraints. Procedure calls to add extended formulations for other relaxations are similar, and are listed in the Appendix in Table 5.

The implementation of the extended formulations in LS-LIB is straightforward, as the scope of the objects of type MIP variables and linear constraints is always global in MOSEL. Thus new variables and constraints declared inside the procedure are still valid after its termination.

Avoiding the difficulty of the large size of extended formulations, another approach is to stay in the original variable space and add valid inequalities as cutting planes in the course of optimization.

The corresponding MOSEL procedures are very similar to these generating extended formulations. The names of the procedures begin with `XCut` instead of `XForm`, and the approximation parameter and the model cuts flag are not defined. The available procedures with their parameters and signification are listed in the Appendix in Table 6.

The implementation of the cutting planes in LS-LIB involves two phases. The procedure `XCutXX` only declares a global structure XX which holds all the information necessary to define the corresponding relaxation, i.e. its type, size, data and MIP variables, and a procedure `XCut_ini` controls a few parameters of cut generation (depth and frequency of the cut generation,...). At optimization time, within a callback, all such global constraints are considered automatically and sequentially and an associated cut separation algorithm is invoked depending on the control parameters.

3.2 Primal Heuristics

The third and last type of procedures defines and calls primal heuristics. The goal of these subroutines is to generate good feasible solutions quickly. The names of the procedures all begin with `XHeur`.

Traditionally, there are two types of heuristics, *construction heuristics* that produce a feasible solution from scratch, and *improvement heuristics* that try to improve a given feasible solution. Here we briefly present those heuristics implemented in LS-LIB, that appear particularly well adapted for production planning problems.

A trivial *construction heuristic* consists in running the default MIP solver for a fixed amount of time.

Relax-and-Fix is a construction heuristic using the idea of decomposing the problem into smaller MIP problems that are much easier to solve, i.e. problems involving many less integer variables. In its simplest version, the integer variables are partitioned into R disjoint sets $Q^1, \ldots, Q^R$ of decreasing importance. At iteration k, $k = 1, \ldots, R$, the variables in $Q^1, \ldots, Q^{k-1}$ have already been fixed, the variables in Q^k are considered as integer, and the variables in $Q^{k+1}, \ldots, Q^R$ are linearly relaxed. Each of the smaller MIP problems is solved - to optimality or for a fixed amount of time - using the default MIP system, and for subsequent iterations the variables in Q^k become fixed at their best or optimal values obtained at the end of iteration k.

The LP-based *improvement heuristics* use some information from the formulation of a problem and from the best available integer solution, and try to improve the latter by searching its neighborhood. The neighborhood is typically defined in such a way that solving the MIP problem over it is relatively easy or fast. Then if a better (or even worse) feasible solution is found, the step can be iterated.

For example, in *Local-Branching* the neighborhood consists of the solutions in which at most k integer variables take different values from those in the current best solution, see Fischetti and Lodi [13].

In *Relaxation-Induced-Neighborhood-Search*, the neighborhood consists of the solutions in which the 0-1 variables having the same value in the linear relaxation and in the best integer solution are fixed at this common value, see Dana et al. [11].

Finally, we propose an *Exchange* or Fix-and-Relax heuristic that uses the same type of decomposition as relax-and-fix. Here, at each iteration, all integer variables are fixed except those in one set Q^k, for some $k \in \{1, \ldots, R\}$. Then, whether or not a better feasible integer solution is found, the exchange step can be iterated.

Clearly LP-based heuristics can be applied either just at the top node, or else at chosen nodes within the branch-and-cut tree. In particular RINS and Local Branching, as well as other simple heuristics such as Diving [11], are available as options in certain MIP systems.

As an example of the calling parameters, the relax-and-fix heuristic procedure from LS-LIB is called by

```
XHeurRF(CY,SOL,NI,NT,MAXT,PAR).
```

Here CY is a $NI \times NT$ array of constraints, each one enforcing the integrality constraint on one binary variable. By modifying these constraints inside the procedure, it is easy to relax the integrality requirement on individual variables, or to fix these variables to specific values and re-optimize. These operations are the only ones needed to implement relax-and-fix.

The set of binary variables will be partitioned in relax-and-fix by taking columns of CY. Usually, these columns correspond to time periods of the production planning model. SOL is the name of the $NI \times NT$ array in which the heuristic solution, specified as values of the integral variables referenced by CY, will be found at the output of the procedure.

Moreover, there are a number of parameters to control the behavior of the procedure. $MAXT$ indicates the maximum time to be spent on each sub-MIP solved during the heuristic, and PAR indicates into how many intervals R the time horizon $1..NT$ will be divided, which is also the number of smaller MIPs to be solved sequentially in the relax-and-fix heuristic.

The available procedures with their parameters and signification are listed in the Appendix in Table 7.

3.3 The $LS-LIB$ Modeling and Optimization Process

Here we formalize somewhat the modeling and optimization prototyping process presented in the introduction.

Algorithm 1: Improved Formulations or Dual Bounds

- User builds his formulation in MOSEL. Suppose that the names chosen for the data and variables are denoted by sn^i for the stock vector, yn^i for the set-up vector, dn^i for the data vector, etc, where i refers to the item i.
- User classifies each item in his model as suggested in Wolsey [38].
- For each item, User decides on one or more LS-LIB reformulations $XFormPROD^i - CAP^i - VAR^i()$ and/or cut routines $XCutPROD^i - CAP^i - VAR^i()$ to try.
- Using LS-LIB, User adds these to the model, i.e $XFormPROD^i - CAP^i - VAR^i(sn^i, yn^i, \ldots, dn^i, \ldots)$
- User optimizes.

Algorithm 2: Improved Solutions or Primal Bounds

- User takes his original, or improved formulation.
- User adds a construction heuristic $XHeurCONSTRUCT()$ model plus any number of calls to improvement heuristics $XHeurIMPROVE()$.
- The best feasible solution is kept, and possibly used in Algorithm 1. Alternatively another reformulation/cut/heuristic combination is tested.

4 Computational Results using LS-LIB

Below we present results using LS-LIB for two intractable multi-level problems. The first is a two-level problem on which we were unable to make any progress for several years. The second are the four test instances with 78 and 80 items respectively of a multi-level assembly problem that have been discussed in several recent papers, and where the two 80 item instances have not yet been solved to optimality (with best gaps so far ranging from 3 to 15%).

4.1 Powder Production and Packing

4.1.1 Problem Description and Context

This problem is a simplified version of a powder production/packing problem. There are 60 types of packed powder, which are the end products. There is a known demand to be met for each of these 60 end products for 30 periods. Backlogging is allowed. The set of end products is partitioned, on the one hand, into 7 different groups sharing a common production line (resource). They are also partitioned, on the other hand, into two distinct groups sharing a common manpower resource.

The production of each end product consumes one given type among 17 available powders (bulk products). The production of the bulk product is part of the problem. There is a common resource shared by all bulks. Other complicating constraints at the bulk level are perishability (the powder must be packed at the latest one period after production), and a maximum total stock level for the powders.

None of the resources in the problem involve set-up times. Finally, there is a time-independent lower bound on production for all end and bulk products. The objective is to minimize stock and backlog, with a higher penalty or weight put on backlog.

4.1.2 Classification and Testing the Initial Formulation

After making minor improvements to the starting formulation provided by the company, we arrived at a starting formulation *pb1b.mos*.

We note that it has two-level product structure, so that reformulations with echelon stocks are appropriate. From this, we observed that the classification of the items is $\{LS-U-B, LB\}$ relative to the parameters useful for LS-LIB. This suggests the use of two relaxations $\{WW-U-B\}$ and $\{WW-U-B, LB\}$.

Using MOSEL, it is not necessary to change the original model in order to define the echelon stock reformulation. It suffices to define each echelon stock variable as the linear expression

$$E(i,t) := s(i,t) + \sum_{j \in Suc(i)} e(j,t),$$

and use $E(i,t)$ as arguments of the LS-LIB procedures. It is worthwhile pointing out that this last equation is not an additional constraint of the model. $E(i,t)$ will be automatically replaced by its expression in terms of the original variables.

We first carried out two runs to get some idea of the difficulty of the problem. We ran the initial problem *pb1b*, and the model *pb1c* with the LS-LIB reformulation of $\{WW-U-B\}$ with $Tk = 5$ and with model cuts ($MC = 1$). Each run was for 900 seconds. The results are shown in Table 1.

For the latter run it takes 114 seconds to obtain the LP value with the model cuts from our reformulation, 193 seconds to obtain the XLP value with the automatically generated system cuts of Xpress (Covercuts=20,Gomcuts=2), and no feasible solution (IP value) is found within 900 seconds. We then reduced the parameter to $Tk = 3$, but still no solution is found within 900 seconds. In the Table, LB gives the lower bound obtained at the end of the computing time (i.e. after 900 seconds), and Gap measures the relative deviation from optimality and is defined as Gap $= 100 \times \frac{IP-LB}{IP}$ %.

instance	m	n	int	LP	XLP	IP	LB	Gap
pb1b	8308	8878	2321	470.2	1062.9	2422.1	1080.4	55%
pb1c	26653	12598	2321	1547.2	1585.4	-	1585.9	-

Table 1: pb1: Initial runs with MAXTIME=900

We see that with the weak formulation *pb1b* feasible solutions are found, but the final gap is more than 50%, so that we have no idea how good our best feasible solution really is. On the other hand with the strengthened formulation *pb1c*, we obtain a significantly stronger lower bound, but no feasible solutions are found within 900 seconds because the solutions of the linear program at each node are taking a long time due to the size of the formulation. The gap obtained by combining *pb1b* and *pb1c* is 33%.

Our tentative conclusion at this stage is that proving optimality for this problem is probably out of the question. So we decided to address the question of finding a feasible solution guaranteed to be within 25% of optimality or better within say 30 minutes. Specifically we consider how good a lower bound we can obtain in up to 15 minutes, and separately how good an upper bound we can obtain in the same time.

4.1.3 Finding Lower Bounds

Here there was no choice but to use the linear programming bounds provided by the extended formulations, as improving the lower bounds in the branch-and-cut tree was far too slow.

In Table 2 we show the results of runs for pb1c with different values of TK for $WW-U-B$ (see column $TK1$), and in addition the reformulation $WW-U-B, LB$ for items with significant demands using $TK = 4$ (see column $TK2$).

instance	TK1	TK2	m	n	LP	XLP	Secs
pb1c	3	0	20423	12598	1398.8	1452.2	110
pb1c	5	0	26653	12598	1547.2	1581.0	171
pb1c	8	0	35089	12598	1657.3	1713.2	282
pb1c	15	0	52948	12598	1692.1	1728.9	403
pb1c	8	4	134106	76094	1682.2	1717.3	3915

Table 2: pb1: Lower Bounds by Reformulation and LP (no branching)

4.1.4 Finding Upper Bounds

Our initial run with the tightened formulation indicated that if we wanted to use *pb1c* for heuristics, we were obliged by the time constraint to work with a decomposition heuristic such as relax-and-fix or exchange, which divided the problem up into significantly smaller subproblems.

In Table 3 we present results obtained first with the weak formulation *pb1b*, and then with the tightened formulation *pb1c*. In each case using LS-LIB, we ran relax-and-fix followed by two full runs of exchange, i.e. two runs of exchange through all $NS = R$ blocks of variables. On the tightened formulation, the time to find a feasible solution even for the smaller subproblems is long and somewhat unpredictable. Thus $MAXT = 60$ indicates that a run stops after 60 seconds if a solution has been found, and otherwise continues until a first feasible solution is discovered.

instance	MAXT	Ns	RF val.	EXCH val.	EXCH val.	RF Secs	EXCH Secs	EXCH Secs
pb1b	-60	6	2254.3	2102.6	2083.3	360	360	360
pb1b	-30	7	2297.6	2140.7	2117.6	210	189	187
pb1c(TK=8)	60	6	2222.5	2089.1	2073.6	1569	360	360
pb1c(TK=5)	60	6	2158.1	2095.4	2061.2	1504	360	360

Table 3: pb1: Primal Heuristics - Upper Bounds

Thus we can obtain a lower bound with $TK = 8$ of 1713.2 in 282 secs, and an upper bound of 2083.3 in 360 secs. This gives a gap of 17.8%. The best bounds 1728.9 in 403 secs and 2061.2 in about 2200 secs give a gap of 15.1%.

4.2 Assembly of Bottling Racks

These problems are multi-level problems with assembly product structure. Many items can be produced in each time period, and the capacity constraints limiting production on each resource in each time period involve both production rates per item and set-up times for families of items. The objective is to satisfy demand without backlogging, and to minimize a combination of inventory holding costs and family set-up costs.

For these problems we adopted the initial echelon stock reformulation (44)-(50) from Wolsey [38] for each problem instance. Then using the classification $WW - U$ for each item, we used the same parameters $TK = 5$ for the 78 item problems, and $TK = 9$ for the 80 item problems.

We adapted a similar default strategy to the one used for *pb*1. We applied relax-and-fix followed by exchange to get a good starting solution. Then we fed the heuristic value to the optimizer as a cutoff, and ran for a limited amount of time to improve the lower bound. For the 78 item instances, we set MAXTIME=60 for each MIP during the heuristic (maximum 240 seconds in total), and then we ran the optimizer for the same 240 seconds. For the more difficult 80 item problems, the times were doubled. The results are shown in Table 4, where Gap measures the relative deviation

from optimality as Gap $= 100 \times \frac{heurval - LB}{heurval}$ %, and the last column indicates the optimal or best known value for each problem instance.

instance	heurval	XLP	LB	Gap(LB)	Optimal value
cl3-78-1b	11505.4	10840.6	11164.6	3.0	11503.4
cl3-78-2b	10889.1	10515.8	10643.5	2.3	10889.1
cl3-80-1b	24913.4	21526.8	21906.0	12.1	[23238, 24544]
cl3-80-2b	26005.6	22152.8	22385.4	13.9	[22385, 25740]

Table 4: cl3-Multi-Level Assembly

5 Conclusions

Our goal has been to demonstrate how the arrival of high-level modeling and optimization languages, such as MOSEL, has finally enabled us to develop an easily useable platform for solving various lot-sizing problems by mixed integer programming. As researchers and consultants, we finally have a tool that can be easily maintained and extended, allows us to tackle and advise on new problems formulated in MOSEL within minutes, incorporates a considerable amount of the knowledge accumulated over the last twenty years, and does not depend on sophisticated programmers or doctoral students for its survival. For end-users we hope that it is a useful tool, particularly for prototyping.

Our approach has been to enrich the modeling vocabulary of MIP (linear and integrality constraints) with higher-level lot-sizing primitives. The structured information, encoded by the modeler in these new primitives, is used by the system to automatically improve the initial MIP formulation. We feel that the division of the work between the end-user and the computer is at its right place: the end-user only works with a modeling language and has full control over extended reformulations, cutting planes and primal heuristics. However, he does not need to know, and even less to program, their mathematical or algorithmic details.

Whether a similar system could be built using languages such as AMPL, OPL or GAMS is an interesting question. For example, constraints and linear expressions are passed as arguments of all procedures in LS-LIB. It is not clear whether something similar can be done, or another implementation solution can be found, in these languages.

Computationally all the results we report in Section 4 can be obtained without LS-LIB, and similar results for the set of multi-level instances have been reported earlier in Belvaux and Wolsey [6, 7]. However what previously required the study and correct translation of an extended formulation or the programming of a separation routine is now immediately available and ready for use. It is perhaps worth adding that the idea of the conceptually trivial EXCHANGE heuristic came after implementing the relax-and-fix heuristic for LS-LIB, and, as shown on the *pb*1 problem, it quickly leads to high quality solutions that previously took twelve or more hours to find.

Several extensions to LS-LIB can obviously be envisaged, including treatment of various single-item variants, such as sales, piecewise concave production costs, etc. No routines for multi-item models have been incorporated into LS-LIB, partly because the resulting relaxations are NP-hard, including knapsack or simple mixed integer constraints for which the commercial branch-and-cut MIP systems already have separation heuristics, and partly because highly effective reformulations or cutting planes have not yet been discovered.

It is an obvious question whether a similar library might be useful for other classes of MIP problems, such as network design, location, or more general supply chain problems. To date we have seen that the LS-LIB relax-and-fix routine works well on certain location problems where time intervals are replaced by geographical regions.

Finally given the sort of results that can now be obtained – good lower bounds from reformulations and cutting planes – good upper bounds from a variety of heuristics, we believe that the

developers of MIP systems can now no longer avoid a crucial question: How to use a good feasible solution effectively so as to significantly speed up the branch-and-cut process?

An initial version of the LSLIB library, consisting of the file "lslibwww.bim" and user instructions are available at www.core.ucl.ac.be/PPbyMIP/, or from the authors.

References

[1] P. Afentakis, B. Gavish, and U. Karmarkar. Computationally efficient optimal solutions to the lot-sizing problem in multistage assembly systems. *Management Science*, 30:222–239, 1984.

[2] A. Agra and M. Constantino. Lotsizing with backlogging and start-ups: the case of Wagner-Whitin costs. *Operations Research Letters*, 25:81–88, 1999.

[3] A. Atamtürk and J.C. Munoz. A study of the lot-sizing polytope. *Mathematical Programming*, 99:443–466, 2004.

[4] I. Barany, J. Edmonds, and L.A. Wolsey. Packing and covering a tree by subtrees. *Combinatorica*, 6:245–257, 1986.

[5] I. Barany, T.J. Van Roy, and L.A. Wolsey. Uncapacitated lot sizing: the convex hull of solutions. *Mathematical Programming*, 22:32–43, 1984.

[6] G. Belvaux and L.A. Wolsey. Bc-prod: A specialized branch-and-cut system for lot-sizing problems. *Management Science*, 46:724–738, 2000.

[7] G. Belvaux and L.A. Wolsey. Modelling practical lot-sizing problems as mixed integer programs. *Management Science*, 47:993–1007, 2001.

[8] A.J. Clark and H. Scarf. Optimal policies for multi-echelon inventory problems. *Management Science*, 6:475–490, 1960.

[9] M. Constantino. A cutting plane approach to capacitiated lot-sizing with start-up costs. *Mathematical Programming*, 75:353–376, 1996.

[10] M. Constantino. Lower bounds in lot-sizing models: a polyhedral study. *Mathematics of Operations Research*, 23:101–118, 1998.

[11] E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.

[12] G.D. Eppen and R.K. Martin. Solving multi-item lot-sizing problems using variable definition. *Operations Research*, 35:832–848, 1987.

[13] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–48, 2003.

[14] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85:439–467, 1998.

[15] J. Krarup and O. Bilde. Plant location, set covering and economic lot sizes: an O(mn) algorithm for structured problems. In L. Collatz et al., editor, *Optimierung bei Graphentheoretischen und Ganzzahligen Probleme*, pages 155–180. Birkhauser Verlag, Basel, 1977.

[16] A. Miller and L.A. Wolsey. Tight MIP formulations for multi-item discrete lot-sizing problems. *Operations Research*, 51:557–565, 2003.

[17] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988.

[18] M.W. Padberg, T.J. Van Roy, and L.A. Wolsey. Valid inequalities for fixed charge problems. *Mathematical Programming*, 33:842–861, 1985.

[19] Y. Pochet. Valid inequalities and separation for capacitated economic lot-sizing. *Operations Research Letters*, 7:109–116, 1988.

[20] Y. Pochet. Mathematical programming models and formulations for deterministic production planning problems. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization*, volume 2241 of *LNCS*, pages 57–111. Springer, 2001.

[21] Y. Pochet and L.A. Wolsey. Lot-size models with backlogging: Strong formulations and cutting planes. *Mathematical Programming*, 40:317–335, 1988.

[22] Y. Pochet and L.A. Wolsey. Polyhedra for lot-sizing with Wagner-Whitin costs. *Mathematical Programming*, 67:297–324, 1994.

[23] Y. Pochet and L.A. Wolsey. Algorithms and reformulations for lot sizing problems. In W. Cook, L. Lovasz, and P. Seymour, editors, *Combinatorial Optimization*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1995.

[24] Y. Pochet and L.A. Wolsey. Formulations for single item lot-sizing with Wagner-Whitin costs and nondecreasing capacities. Discussion Paper 2005/xx, CORE, Université catholique de Louvain, 2005.

[25] T.J. Van Roy and L.A. Wolsey. Valid inequalities for mixed $0-1$ programs. *Discrete Applied Mathematics*, 14:199–213, 1986.

[26] H. Stadtler. Reformulations of the shortest-route model for dynamic multi-item multi-level capacitated lot-sizing. *OR Spektrum*, 19:87–96, 1997.

[27] H. Stadtler. Improved rolling schedules for the dynamic single-level lot-sizing problem. *Management Science*, 46:318–326, 2000.

[28] H. Stadtler. Multilevel lot sizing with setup times and multiple constrained resources: internally rolling schedules with lot-sizing windows. *Operations Research*, 51:487–502, 2003.

[29] H. Tempelmeier and M. Derstoff. A lagrangean-based heuristic for dynamic multilevel multiitem constrained lotsizing with setup times. *Management Science*, 42:738–757, 1996.

[30] C.A. van Eijl and C.P.M. van Hoesel. On the discrete lot-sizing and scheduling problem with Wagner-Whitin costs. *Operations Research Letters*, 20:7–13, 1997.

[31] C.P.M. van Hoesel and A.W.J. Kolen. A linear description of the discrete lot-sizing and scheduling problem. *European Journal of Operational Research*, 75:342–353, 1994.

[32] C.P.M. van Hoesel, A. Wagelmans, and L.A. Wolsey. Polyhedral characterization of the economic lot-sizing problem with start-up costs. *SIAM Journal of Discrete Mathematics*, 7:141–151, 1994.

[33] M. Van Vyve. Lot-sizing with constant lower bounds on production. draft 2, CORE, Université catholique de Louvain, 2003.

[34] M. Van Vyve. The continuous mixing polyhedron. *Mathematics of Operations Research*, 00:0–1, 2005.

[35] M. Van Vyve. Formulations for the single item lot-sizing problem with backlogging and constant capacity. *Mathematical Programming*, 00:0–1, 2005.

[36] M. Van Vyve and L.A. Wolsey. Approximate extended formulations. *Mathematical Programming B*, 00:0–1, 2005.

[37] L.A. Wolsey. *Integer Programming*. John Wiley and Sons, 1998.

[38] L.A. Wolsey. Solving multi-item lot-sizing problems with an MIP solver using classification and reformulation. *Management Science*, 48:1587–1602, 2002.

6 Appendix: LS-LIB Procedures and Functions

In this section we present the procedures available in $LS-LIB$.

6.1 Reformulations - XForm

Each reformulation concerns a single item subproblem. In Table 5, Columns 1 and 2 indicate the problem classification, respectively $PROB-CAP$ and VAR.

Row 1 contains the possible procedure parameters:

S denotes the stock vector $s_0, s_1, \ldots, s_{NT}$,
R denotes the backlog vector $r_1, \ldots, r_{NT}$,
X denotes the production vector $x_1, \ldots, x_{NT}$,
Y denotes the set-up vector $y_1, \ldots, y_{NT}$,
Z denotes the start-up vector $z_1, \ldots, z_{NT}$,
W denotes the switch-off vector $w_1, \ldots, w_{NT}$,
D denotes the demand vector $d_1, \ldots, d_{NT}$,
C denotes the constant capacity C, or the capacity vector $C_1, \ldots, C_{NT}$,
NT denotes the number of time periods $n = NT$,
TK is the approximation parameter controlling the size and quality of the reformulation,
MC indicates if constraints are added as Model Cuts ($MC = 1$), or are added a priori to the formulation ($MC = 0$).

A "Y" in the Table indicates that the corresponding parameter is present,
a "-" indicates that the parameter is not present,
a "0" in the "S" column indicates that just s_0 is present,
a "1/LB" in the "C" column indicates that the constant capacity is assumed to be $C = 1/$ the constant lower bound is LB, and no capacity parameter is passed to the routine.

		S	R	X	Y	Z	W	D	C	NT	TK	MC	Size (cons × vars)	ref
LS-U		Y	-	Y	Y	-	-	Y	-	Y	Y	Y	$O(n^2)\times O(n^2)$	[15]
LS-U		Y	-	Y	Y	-	-	Y	-	Y	Y	Y	$O(n)\times O(n^2)$	[12]
LS-U	B	Y	Y	Y	Y	-	-	Y	-	Y	Y	Y	$O(n^2)\times O(n^2)$	[4, 21]
WW-U		Y	-	-	Y	-	-	Y	-	Y	Y	Y	$O(n^2)\times O(n)$	[22]
WW-U	B	Y	Y	-	Y	-	-	Y	-	Y	Y	Y	$O(n^2)\times O(n)$	[22]
WW-U	SC	Y	-	-	Y	Y	-	Y	-	Y	Y	Y	$O(n^2)\times O(n)$	[22]
WW-U	SC,B	Y	Y	-	Y	Y	Y	Y	-	Y	Y	Y	$O(n^2)\times O(n)$	[2]
WW-U	LB	Y	Y	-	Y	-	-	Y	LB	Y	Y	Y	$O(n^3)\times O(n^2)$	[33]
WW-U	B,LB	Y	Y	-	Y	-	-	Y	LB	Y	Y	Y	$O(n^3)\times O(n^2)$	[33]
WW-CC		Y	-	-	Y	-	-	Y	Y	Y	Y	Y	$O(n^2)\times O(n^2)$	[22]
WW-CC	B	Y	Y	-	Y	-	-	Y	Y	Y	Y	Y	$O(n^3)\times O(n^2)$	[35]
WW-C		Y	-	-	Y	-	-	Y	Y	Y	Y	Y	$O(n^2)\times O(n^2)$	[24]
DLSI-CC		0	-	-	Y	-	-	Y	Y	Y	Y	Y	$O(n)\times O(n)$	[22, 16]
DLSI-CC	B	0	Y	-	Y	-	-	Y	Y	Y	Y	Y	$O(n^2)\times O(n)$	[16, 34]
DLS-CC	B	-	Y	-	Y	-	-	Y	Y	Y	Y	Y	$O(n)\times O(n)$	[16]
DLS-CC	SC	-	Y	-	Y	Y	-	Y	1	Y	Y	Y	$O(n^2)\times O(n)$	[30]

Table 5: XForm

N.B. In all the $LS-LIB$ procedures, it is assumed that the time horizon is represented in MOSEL

as the range $1..NT$ or $0..NT$, and the set of items/skus/products as the range $1..NI$. If the time periods are represented as a set of strings, or as sets of integers, an appropriate translation is needed before calling the procedures.

Example 1 *Examination of the row $WW-U-SC,B$ in Table 5 indicates that we need to have declared the variables and constants marked with a "Y", and then call the reformulation for each item. We assume that the variables and data in the MOSEL problem formulation are called "sname(i,t), dname(i,t), etc".*

Now, given a formulation written in the modeling language MOSEL, we show what needs to be added to call the reformulation procedure for problem $WW-U-SC,B$ for each item.

```
! upper part, definition of the initial model:
!    data, variables, constraints, objective
!

!STEP1: declaration of the arguments of the procedure
begin-declaration
   NT: integer   ! Usually already declared in the model definition
   S: array(0..NT) of linctr
   R: array(1..NT) of linctr
   Y: array(1..NT) of linctr
   Z: array(1..NT) of linctr
   D: array(1..NT) of real
   TK: integer
   MC: integer
end-declaration

!REFORMULATION LOOP OVER THE ITEMS
forall(i in 1..NI) do

!STEP2: computation of the arguments (for each single item i)
   S(0):= sname(i,0)
   forall (t in 1..NT) do
     S(t):= sname(i,t)
     R(t):= rname(i,t)
     Y(t):= yname(i,t)
     Z(t):= zname(i,t)
     W(t):= wname(i,t)
     D(t):= dname(i,t)
   end-do

!STEP3: call of the reformulation procedure  (for each single item i)
   XFormWWUSCB(S,R,Y,Z,W,D,NT,TK,MC)

!END OF DO-LOOP OVER THE ITEMS
end-do

!
! lower part, solve instructions
!
```

6.2 Cutting Plane Separation - XCut

To call cutting plane separation routines, the procedure arguments are shown in Table 6, and are essentially identical to those in Table 5.

		S	R	X	Y	Z	W	D	C	NT	TK	Sep	ref
LS-U		Y	-	Y	Y	-	-	Y	-	Y	-	$O(n^2)$	[5]
LS-C		Y	Y	Y	-	-	-	Y	Y	Y	-	Pr alper	[3]
WW-U		Y	-	-	Y		-	Y	-	Y	-	$O(n)$	[22]
WW-U	B	Y	Y	-	Y	-	-	Y	-	Y	-	$O(n^3)$	[22]
WW-CC		Y	-	-	Y	-	-	Y	Y	Y	-	$O(n^2 \log n)$	[22]
DLSI-CC		0	-	-	Y	-	-	Y	Y	Y	Y	$O(n \log n)$	[16]
DLSI-CC	B	0	Y	-	Y	-	-	Y	Y	Y	Y	$O(n^3)$	[34]

Table 6: XCut

The call to one of these cut generation routines passes the names of data and variables, and sets up the separation routines. It is implemented in the MOSEL language exactly in the same way as for extended reformulations.

In addition, there is one additional routine

$$XCut\_init(xrows, xelems, depth, freq, npassmax, eps)$$

that must be called once, before the instruction to solve the problem, to activate the cut generation routines during the optimization phase. The parameters of this procedure mean the following:

"xrows" denotes the number of extra rows to store cuts,
"xelems" denotes the number of extra matrix elements to store cuts,
"depth" denotes the maximum depth in the branch-and-cut tree
at which cuts are generated,
"freq" denotes the frequency of cut generation,
i.e. cut routines are called at the root node and at depths that are multiples of freq,
"npassmax" denotes the maximum number of re-optimization passes
at each iteration of cut generation,
"eps" defines the tolerance for positive cut violation.

There are default values for all these parameters, and the non expert may simply write

$$XCut\_init$$

to activate the cut routines, which are then generated at the root node only.

6.3 Heuristics - XHeur

In Table 7 we indicate the calling parameters for the heuristics. Remember that

CY denotes the constraints indexed over $1..NI, 1..NT$ defining the y variables as binary variables,
SOL indexed over $1..NI, 1..NT$ contains as input an initial feasible solution if it is an improving heuristic, and as output the heuristic solution found (if any).

	CY	SOL	NI	NT	MAXT	PAR	Ref
RF	Y	Y	Y	Y	Y	NS	[26, 37]
MIP	Y	Y	Y	Y	Y	-	
CF	Y	Y	Y	Y	Y	-	
RINS	Y	Y	Y	Y	Y	-	[11]
LB	Y	Y	Y	Y	Y	-	[13]
EXCH	Y	Y	Y	Y	Y	NS	Section 3.2

Table 7: XHeur

Here RF is a relax-and-fix heuristic, MIP a trivial heuristic consisting of running default branch-and-cut for a given time, and CF is a cut-and-fix heuristic in which, given the LP solution, all integer variables that take integer values and at least one taking a fractional value are fixed at each iteration. The RINS, Local Branching (LB) and Exchange (EXCH) heuristics have been briefly described in Section 2.

7 Appendix: A Detailed Example

We consider a multi-item problem with start-up and switch-off costs in which at most one item can be produced in each period.

7.1 The Mathematical Formulation

This has the formulation

$$\begin{gathered}
\min \sum_{i,t} h_t^i s_t^i + \sum_{i,t} f_t^i y_t^i + \sum_{i,t} g_t^i z_t^i + \sum_{i,t} q_t^i w_t^i \\
s_{t-1}^i + x_t^i = d_t^i + s_t^i \quad \forall i,t \\
x_t^i \leq C^i y_t^i \quad \forall i,t \\
x_t^i \geq L^i y_t^i \quad \forall i,t \\
z_t^i - w_{t-1}^i = y_t^i - y_{t-1}^i \quad \forall i,t \\
z_t^i \leq y_t^i \quad \forall i,t \\
\sum_i y_t^i \leq 1 \quad \forall t \\
x, s \geq 0, y, z \in \{0,1\}
\end{gathered}$$

Here x_t^i denotes the amount of item i produced per hour, and s_t^i and r_t^i are also measured in production hours. w_t^i is a switch-off variable. Note that w_t^i is precisely the slack variable in the usual constraint $z_{t+1}^i \geq y_{t+1}^i - y_t^i$ used to define z_{t+1}^i.

Each item can be classified as $WW-CC-SC$. Inspection of the XForm and Xcut Tables suggests the possibility of using a reformulation of the relaxation $WW - U - SC$, and/or a reformulation or cutting planes for the relaxation $WW - CC$.

7.2 The Initial MOSEL Formulation

```
!-------------------------------------------------------------------
model 'clbooko'
uses 'mmetc','mmxprs','mmsystem';
setparam("xprs_colorder",1)
setparam("XPRS_VERBOSE",1)

!============================
! SECTION 1: INDICES and DATA
!============================
declarations
   NI=4            ! # of Families
   NT=30           ! # of Time Periods

   RMIN: array (1..NI) of real      ! Production rate [units/hour]
   H: array (1..NI) of real         ! Invent. cost of Prod in fam i [euro/unit,period]
   CAP: array (1..NT) of real       ! Number of production hours in period t [hours]
   F: array (1..NT) of real         ! Set-up cost in period t [euro]
   DEM: array (1..NI,1..NT) of real ! Demand per family per period [hours]
end-declarations

diskdata(ETC_IN,'cldemand.dat',DEM)

RMIN(1):=[807,608,1559,1622]
H(1):=[0.0025,0.0030,0.0022,0.0022]
F(1):=[100,100,100,4600,100,100,100,100,100,4600,100,100,100,100,100,
       100,100,100,4600,100,100,100,100,100,4600,100,100,100,100,100]

forall(t in 1..NT) CAP(t):=16

!=====================
! SECTION 2: VARIABLES
!=====================
declarations
   x: array(1..NI,1..NT) of mpvar ! Production of fam i in hours
   s: array(1..NI,1..NT) of mpvar ! stock of fam i in hours at the end or per t
   y: array(1..NI,1..NT) of mpvar ! =1 if family i produced in per t
   z: array(1..NI,1..NT) of mpvar ! = 1 if family i produced in per t
                                  !   but not in per t-1
   w: array(1..NI,1..NT) of mpvar ! =1 if family i produced in per t
                                  !   but not in per t+1
end-declarations

!====================================
! SECTION 3: OBJECTIVE and CONSTRAINTS
!====================================

COST:= SUM(i in 1..NI, t in 1..NT) H(i)*RMIN(i)*s(i,t) +
       SUM(i in 1..NI, t in 1..NT) F(t)*y(i,t) +
       SUM(i in 1..NI, t in 1..NT) 50*z(i,t) +
       SUM(i in 1..NI, t in 1..NT) 50*w(i,t)

! DEMAND SATISFACTION
```

```
forall(i in 1..NI, t in 1..NT)
  AGD(i,t):= x(i,t)+ IF(t>1,s(i,t-1),0) = DEM(i,t) +s(i,t)

! VUB FOR SETUPS
forall(i in 1..NI, t in 1..NT)
  SA(i,t):= CAP(t)*y(i,t) - x(i,t)>= 0

! VLB FOR SETUPS
forall(i in 1..NI, t in 1..NT)
  LB(i,t):= 7*y(i,t) <= x(i,t)

! ONE PRODUCTION PER PERIOD
forall(t in 1..NT)
  mode(t):= SUM(i in 1..NI) y(i,t)<= 1

! START-UPS AND SWITCH-OFFS
forall(i in 1..NI, t in 2..NT)
  SYS(i,t):= y(i,t)-y(i,t-1)= z(i,t)-w(i,t-1)
forall(i in 1..NI)
  SYT(i):= y(i,1) = z(i,1)

! VARIABLE TYPES AND BOUNDS
forall(i in 1..NI,t in 1..NT)
  y(i,t) is_binary
forall(i in 1..NI,t in 1..NT)
  w(i,t)<= 1
forall(i in 1..NI,t in 1..NT)
  z(i,t)<= 1


!====================
! SECTION 4: SOLUTION
!====================

minimize(COST)
exit(0)
end-model
!----------------------------------------------------------------------
```

7.3 Adding a Reformulation

In the LS-LIB-Reformulation Table 5 of Section 3, we see that we can get the extended formulation for $WW - U - SC$ by calling

$$XFormWWUSC(S, Y, Z, NT, TK, MC).$$

Thus we insert the following block just before the line

```
minimize(COST)
```

```
!----------------------------------------------------------------------

include 'D:\\lotsizing\LSLIB-XForm.mos'
```

```
declarations
  Y: array(1..NT) of linctr
  Z: array(1..NT) of linctr
  S: array(0..NT) of linctr
  D: array(1..NT) of real
end-declarations

S(0):=0
forall(i in 1..NI) do
    forall(t in 1..NT) Y(t):=y(i,t)
    forall(t in 1..NT) Z(t):=z(i,t)
    forall(t in 1..NT) S(t):=s(i,t)
    forall(t in 1..NT) D(t):=DEM(i,t)
    XFormWWUSC(S,Y,Z,D,NT,15,0)
end-do
!-------------------------------------------------------------------
```

Here the first statement calls the LS-LIB library, the declarations block defines which variables and data need to be passed for each item. Then the do loop over the items, passes the names used in the MOSEL file, i.e the set-up variable y_t^i is $y(i,t)$, the demand d_t^i is $DEM(i,t)$, etc.

7.4 Adding a Cut Separation Routine

In the LS-LIB-Cut Table 6 of Section 3, we see that we can get the cutting plane routine for $WW-CC$ by calling

$$XCutWWCC(S,Y,C,NT).$$

Thus we insert the following block just before the line

```
 minimize(COST)
```

```
!-------------------------------------------------------------------

include 'D:\\lotsizing\LSLIB-XCut.mos'
 declarations
  Y: array(1..NT) of linctr
  Z: array(1..NT) of linctr
  S: array(0..NT) of linctr
  D: array(1..NT) of real
end-declarations

S(0):=0
forall(i in 1..NI) do
    forall(t in 1..NT) Y(t):=y(i,t)
    forall(t in 1..NT) S(t):=s(i,t)
    forall(t in 1..NT) D(t):=DEM(i,t)
    XCutWWCC(S,Y,D,16,NT)
end-do

XCut_init
!-------------------------------------------------------------------
```

Here the block inserted is almost identical to that used for the reformulation. The call to $XCutWWCC(S, Y, D, 16, NT)$ passes the names of the names of the data and variables, and sets up the separation routines. The final line $XCut\_init$ activates the routines when optimization occurs.

Note that to add both the reformulation and cutting plane routines *simultaneously*, it suffices to replace the last two lines used for the reformulation by

```
    XFormWWUSC(S,Y,Z,D,NT,15,0)
    XCutWWCC(S,Y,D,16,NT)
end-do
XCut_init
```

The results for the original formulation and the three reformulations are shown in Table 8, where XLP indicates the relaxation value at the root node after the addition of the Xpress and LS-LIB cuts.

instance		m	n	int	LP	XLP	IP	secs	nodes
cl-run1		510	720	120	1509.1	3972.8	4404.5	23	8155
cl-run2	XForm	1438	720	120	3775.1	4333.1	4404.5	12	196
cl-run3	XCut	510	720	120	1509.1	3972.1	4404.5	27	7659
cl-run4	XForm & XCut	1438	720	120	3775.1	4344.7	4404.5	9	45

Table 8: Results with reformulation (TK=15) and/or cuts

7.5 Adding some Heuristics

Even though heuristics are not necessary for this problem, we apply a relax-and-fix heuristic dividing the time horizon into two, and allowing up to 3 seconds for the optimization of each partial problem, followed by the RINS heuristic for up to 10 seconds. We take the initial formulation of the problem and replace the $minimize(COST)$ line by the following block.

```
!-------------------------------------------------------------------

include 'D:\\lotsizing\LSLIB-XHeur.mos'
declarations
   CY:    array(1..NI,1..NT) of linctr
   HEURSOL:array(1..NI,1..NT) of integer
   MAXT:  integer
end-declarations

forall(i in 1..NI,t in 1..NT)
   CY(i,t):= y(i,t)  is_binary
MAXT  := 3
NS    := 2
hval  := XHeurRF(CY,HEURSOL,COST,NI,NT,MAXT,NS)

MAXT  := 10
hval :=XHeurRINS(CY,HEURSOL,COST,NI,NT,MAXT)

!-------------------------------------------------------------------
```

Here *hval* contains the value of the heuristic solution returned by the functions $XHeurRF$ and $XHeurRINS$, CY is the name of the linear constraints indicating the 0-1 variables, $HEURSOL$ contains the heuristic solution found, which is also the initial solution passed to the improvement heuristic function $XHeurRINS$, COST is the name of the linear constraint containing the objective function, the time allowed for each MIP solved during the heuristic is MAXT seconds, and NS is the number of segments into which the time horizon is divided.

Relax-and-fix finds the optimal solution after less than 1 second, and RINS cannot improve on it, terminating after 3 seconds.