Hybrid Tabu Search for Lot Sizing Problems

João Pedro Pedroso[1] and Mikio Kubo[2]

[1] DCC-FC and LIACC, Universidade do Porto,
Rua do Campo Alegre, 823, 4150-180 Porto, Portugal
jpp@ncc.up.pt
[2] Supply Chain Optimization Lab.,
Tokyo University of Marine Science and Technology,
2-1-6 Etsuujima Koutou-ku, Tokyo 135-8533, Japan
kubo@e.kaiyodai.ac.jp

Abstract. This paper presents a hybrid tabu search strategy for lot sizing problems. This strategy allows the exploitation of the quality of the well-known relax-and-fix heuristic, inside a tabu search framework which enforces diversity.

The computational results show an advantage of this strategy when compared to a version of the relax-and-fix heuristic and to time constrained branch-and-bound.

1 Introduction

Lot sizing is a class of combinatorial optimization problems with applications in production planning. In these problems there is a fixed number of periods, and in each period production of items can occur in machines. A machine has to be appropriately setup for being able to produce, and this setup implies, e.g., the payment of a fixed cost, or the reduction of the machine working time by some fixed amount.

The amount produced in a given period can be used to satisfy the demand of that period, or remain in inventory. When production can also be used to satisfy demand of preceding periods, the models are said to allow backlogging.

Lot sizing problems can be classified into small-bucket or big-bucket models. On small bucket models, each machine can produce at most one item per period; on big-bucket models, several items can be manufactured in each period.

Good surveys on lot sizing are provided in [3] and [6].

2 The Lot Sizing Model

The problem that we deal with in this paper is a lot sizing problem belonging to the big bucket class: more than one setup is allowed per period, as long as the machine capacities are respected.

The costs that are to be taken into account are setup costs, variable production costs, and inventory and backlog costs. Unitary values for each of them can vary from period to period.

The decision variables in this lot sizing problem concern the manufacture or not of a product in each period, as well as the amount to produce. The setup, binary variable y_{pmt} is 1 if product p is manufactured in machine m during period t, and 0 otherwise. The continuous variable x_{pmt} is the corresponding manufactured amount.

Let T be the number of periods and $\mathcal{T} = \{1, \ldots, T\}$. Let $\mathcal{P}$ be the set of products and $\mathcal{M}$ be the set of machines. Let furthermore $\mathcal{M}^p$ be the subset of machines that are compatible with the production of p. The setup costs are then determined by:

$$F = \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}^p} \sum_{t \in \mathcal{T}} f_{pmt}\, y_{pmt}, \tag{1}$$

where f_{pmt} is the cost of setting up machine m on period t for producing p. Similarly, variable costs are

$$V = \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}^p} \sum_{t \in \mathcal{T}} v_{pmt}\, x_{pmt}, \tag{2}$$

where v_{pmt} is the variable cost of production of p on machine m, period t. If h_{pt} is the amount of product p that is kept in inventory at the end of period t, the inventory costs can be determined by

$$I = \sum_{p \in \mathcal{P}} \sum_{t \in \mathcal{T}} i_{pt}\, h_{pt}, \tag{3}$$

where i_{pt} is the unit inventory cost for product p on period t. Analogously, if g_{pt} is the amount of product p that failed to meet demand at the end of period t, the backlog costs can be determined by

$$B = \sum_{p \in \mathcal{P}} \sum_{t \in \mathcal{T}} b_{pt}\, g_{pt}, \tag{4}$$

where b_{pt} is the unit backlog cost for product p on period t. The lot sizing objective can now be written as

$$\text{minimise } z = F + V + I + B. \tag{5}$$

If the demand of a product p in period t is D_{pt}, the flow conservation constraints can be written as

$$h_{p,t-1} - g_{p,t-1} + \sum_{m \in \mathcal{M}^p} x_{pmt} = D_{pt} + h_{pt} - g_{pt} \quad \forall\, p \in \mathcal{P},\ \forall\, t \in \mathcal{T}. \tag{6}$$

The initial inventory and backlog for each product p should be assigned to h_{p0} and g_{p0}, respectively (and possibly equivalent assignments might be made for h_{pT} and g_{pT}).

There is a limit on the time that each machine is available on each period; this implies that

$$\sum_{p \in \mathcal{P}:m \in \mathcal{M}^p} \left(\frac{x_{pmt}}{\gamma_{pm}} + \tau_{pmt}\, y_{pmt} \right) \leq A_{mt} \quad \forall\, m \in \mathcal{M},\ \forall\, t \in \mathcal{T}. \tag{7}$$

In this equation, γ_{pm} is the total capacity of production of product p on machine m per time unit, τ_{pmt} is the setup time required if there is production of p on machine m during period t, and A_{mt} is the number of time units available for production on machine m during period t.

Manufacturing of a given product can only occur on machines which have been setup for that product:

$$x_{pmt} \leq \gamma_{pm} \, A_{mt} \, y_{pmt} \quad \forall \, p \in \mathcal{P}, \, \forall \, m \in \mathcal{M}^p, \, \forall \, t \in \mathcal{T}. \tag{8}$$

The problem can be summarized as the following mixed-integer program (MIP):

$$
\begin{aligned}
\text{minimise} \quad & z = F + V + I + B \\
\text{subject to :} \quad & F = \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}} \sum_{t \in \mathcal{T}} f_{pmt} \, y_{pmt} \\
& V = \sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}} \sum_{t \in \mathcal{T}} v_{pmt} \, x_{pmt} \\
& I = \sum_{p \in \mathcal{P}} \sum_{t \in \mathcal{T}} i_{pt} \, h_{pt} \\
& B = \sum_{p \in \mathcal{P}} \sum_{t \in \mathcal{T}} b_{pt} \, g_{pt} \\
& h_{p,t-1} - g_{p,t-1} + \sum_{m \in \mathcal{M}^p} x_{pmt} = D_{pt} + h_{pt} - g_{pt}, \quad \forall \, p \in \mathcal{P}, \, \forall \, t \in \mathcal{T} \\
& \sum_{p \in \mathcal{P}:m \in \mathcal{M}^p} \left(\frac{x_{pmt}}{\gamma_{pm}} + \tau_{pmt} \, y_{pmt} \right) \leq A_{mt}, \quad \forall \, m \in \mathcal{M}, \, \forall \, t \in \mathcal{T} \\
& x_{pmt} \leq \gamma_{pm} \, A_{mt} \, y_{pmt} \quad \forall \, p \in \mathcal{P}, \, \forall \, m \in \mathcal{M}^p, \, \forall \, t \in \mathcal{T} \\
\\
& F, V, I, B \in \mathbb{R}^+ \\
& h_{pt}, \, g_{pt} \in \mathbb{R}^+, \quad \forall \, p \in \mathcal{P}, \, \forall \, t \in \mathcal{T} \\
& x_{pmt} \in \mathbb{R}^+, \, y_{pmt} \in \{0,1\}, \quad \forall \, p \in \mathcal{P}, \forall \, m \in \mathcal{M}, \, \forall \, t \in \mathcal{T}
\end{aligned}
\tag{9}
$$

3   Construction: Relax-and-Fix-One-Product

For the construction of a solution to the problem defined by Problem 9, we consider partial relaxations of the initial problem, in a variant of the classic relax-and-fix [8,9] heuristic.

In the basic form of the relax-and-fix heuristic, each period is treated independently. The strategy starts by relaxing all the variables except those of period 1, thus keeping y_{pm1} integer and relaxing integrity for all other y_{pmt}. This MIP is solved, determining the heuristic values for variables $\bar{y}_{pm1}$ (i.e., the binary variables of the first period). The approach then moves to the second period. The variables of the first period are fixed at $y_{pm1} = \bar{y}_{pm1}$, the variables y_{pm2} are integer, and all the other y_{pmt} relaxed; this determines the heuristic

value for y_{pm2}. These steps are repeated, until all the y variables are fixed, as described in Algorithm 1.

Algorithm 1: Relax-and-fix heuristic.
RELAXANDFIX()
(1)   relax $y_{pmt}, \forall\, p \in \mathcal{P},\ \forall\, m \in \mathcal{M},\ \forall\, t \in \mathcal{T}$, as continuous variables
(2)   **for** $t = 1$ **to** T
(3)   **foreach** $p \in \mathcal{P}$
(4)   **foreach** $m \in \mathcal{M}^p$
(5)   set y_{pmt} as integer
(6)   solve Problem 9, determining $\bar{y}_{pmt},\ \forall p \in \mathcal{P},\ \forall m \in \mathcal{M}^p$
(7)   **foreach** $p \in \mathcal{P}$
(8)   **foreach** $m \in \mathcal{M}^p$
(9)   fix $y_{pmt} := \bar{y}_{pmt}$
(10)   **return** $\bar{y}$

This approach is reported to provide very good solutions for many lot sizing problems. However, for large instances the exact MIP solution of even a single period can be too time consuming. Therefore, we propose a variant were each MIP determines only the variables of one period *that concern a single product*. We call this approach *relax-and-fix-one-product*; it is outlined in Algorithm 2 (were a random permutation of a set $\mathcal{P}$ is denoted by $\mathscr{R}(\mathcal{P})$).

Algorithm 2: Relax-and-fix-one-product variant.
RELAXANDFIXONEPRODUCT()
(1)   relax $y_{pmt}, \forall\, p \in \mathcal{P},\ \forall\, m \in \mathcal{M},\ \forall\, t \in \mathcal{T}$, as continuous variables
(2)   **for** $t = 1$ **to** T
(3)   **foreach** $p \in \mathscr{R}(\mathcal{P})$
(4)   **foreach** $m \in \mathcal{M}^p$
(5)   set y_{pmt} as integer
(6)   solve Problem 9, determining $\bar{y}_{pmt},\ \forall m \in \mathcal{M}^p$
(7)   **foreach** $m \in \mathcal{M}^p$
(8)   fix $y_{pmt} := \bar{y}_{pmt}$
(9)   **return** $\bar{y}$

3.1   Solution Reconstruction

As we will see in the next section, the relax-and-fix-one-product construction mechanism can be interestingly used in a different context: that of completing a solution that has been partially destructed. For this purpose, all that is required is to check if incoming $\bar{y}_{pmt}$ variables are initialized or not; if they are initialized, they should be fixed in the MIP at their current value. Otherwise, they are treated as in previous algorithm: their are made integer if they belong to the period and product currently being dealt, and relaxed otherwise. These ideas are described in Algorithm 3 (we denote a random permutation of a set $\mathcal{P}$ by $\mathscr{R}(\mathcal{P})$).

Algorithm 3: Relax-and-fix for solution reconstruction

RECONSTRUCT($\bar{y}$)

(1)   **for** $t = 1$ **to** T
(2)   **foreach** $p \in \mathcal{P}$
(3)   **foreach** $m \in \mathcal{M}^p$
(4)   **if** $\bar{y}_{pmt}$ is not initialized
(5)   relax y_{pmt}
(6)   **else**
(7)   fix $y_{pmt} := \bar{y}_{pmt}$
(8)   **for** $t = 1$ **to** T
(9)   **foreach** $p \in \mathcal{R}(\mathcal{P})$
(10) $\mathcal{U} := \{\}$
(11)   **foreach** $m \in \mathcal{M}^p$
(12)   **if** $\bar{y}_{pmt}$ is not initialized
(13)   set y_{pmt} as integer
(14) $\mathcal{U} := \mathcal{U} \cup \{(p, m, t)\}$
(15)   solve Problem 9, determining $\bar{y}_{pmt}, \forall (p, m, t) \in \mathcal{U}$
(16)   **foreach** $(p, m, t) \in \mathcal{U}$
(17)   fix $y_{pmt} := \bar{y}_{pmt}$
(18)   **return** $\bar{y}$

4   A Hybrid Tabu Search Approach

In this section we present a hybrid metaheuristic approach devised for tackling the lot sizing problem. The approach is a two-fold hybrid, were relax-and-fix-one-product is used to initialize a solution, or complete partial solutions, and tabu search [2] is responsible for creating diverse points for restarting relax-and-fix. Before each restart, the current tabu search solution is partially destructed; its reconstruction is made by means of the relax-and-fix-one-product procedure presented in Algorithm 3.

4.1   Solution Representation and Evaluation

In what concerns tabu search, the subset of variables of the Problem 9 which is necessary to store is the set of y_{pmt} variables; all the continuous variables can be determined in function of these. Thus, a tabu search solution will consist of a matrix of the $\bar{y}_{pmt}$ binary variables.

The evaluation of a solution can be made through the solution of the Problem 9, with all the binary variables fixed at the values $\bar{y}_{pmt}$. As all the binary variables are fixed, this problem is a linear program (LP). The value of z at the optimal solution of this LP will provide the evaluation of the quality of $\bar{y}_{pmt}$. The values of all the other variables x, h and g corresponding to $\bar{y}_{pmt}$ are also determined through this LP solution.

4.2   Tabu Search

The tabu search framework, presented in Algorithm 4, is based only on short term memory. This procedure has a parameter, *tlim*, which is the limit of CPU

to be used in the search (as an alternative to the usual stopping criterion, based on the number of iterations). The remaining arguments are a seed for initializing the random number generator, and the name of the instance to be solved.

Algorithm 4: Hybrid tabu search.

TABUSEARCH(*tlim, seed, instance*)
(1)   store *instance* information as global data $\mathcal{T}, \mathcal{P}, \mathcal{M}, f, g, \ldots$
(2)   initialize random number generator with *seed*
(3) $\bar{y} :=$ RELAXANDFIXONEPRODUCT()
(4) $\bar{y}^* := \bar{y}$
(5) $n := |\mathcal{T}| \times |\mathcal{P}|$
(6) $\Theta := ((-n, \ldots, -n), \ldots, (-n, \ldots, -n))$
(7) $i := 1$
(8)   **while** CPUTIME() $< tlim$
(9) $\bar{y} :=$ TABUMOVE($\bar{y}, \bar{y}^*, i, \Theta$)
(10)   **if** $\bar{y}$ is better than $\bar{y}^*$
(11) $\bar{y}^* := \bar{y}$
(12) $i := i + 1$
(13)   **return** $\bar{y}^*$

4.3 Neighborhood and Candidate Selection

In the course of a tabu search iteration, the neighborhood of the current solution is searched as presented in sAlgorithm 5. The arguments of this algorithm are the current solution $\bar{y}$, the best solution found $\bar{y}^*$, the current iteration i, and the tabu matrix Θ.

Tabu information is kept in the matrix Θ, where Θ_{pm} holds the iteration at which some variable y_{pmt} has been updated. The tabu tenure is a random value, drawn in each iteration between 1 and the number of integer variables and stored in the variable d, on line 6 of Algorithm 5 ($\mathscr{R}[a, b]$ is the notation used for a random integer with uniform distribution in $[a, \ldots, b]$). If the current iteration is i, a move involving product p and machine m will be tabu if $i - \Theta_{pm} \leq d$; otherwise (i.e., if $i - \Theta_{pm} > d$) it is not tabu. Making the tabu tenure a random value simplifies the parameterization of the algorithm.

The neighborhood used consists of solutions where manufacturing a product in a given period and machine is stopped, and its manufacture is attempted in different machines, on the same period. Hence, for a given solution y we start by checking, in a random order, what are the products which are being manufactured in the first period. Let us suppose that a product p is being manufactured in machine m, i.e., $y_{pm1} = 1$. The first neighbor is a solution where $y_{pm1} = 0$, all the other elements being equal to their equivalent in y. Other neighbors have $y_{pm'1} = 1$ for all the machines $m' \neq m$ where p was *not* being produced. After checking the first period, we check sequentially the periods $2, \ldots, T$, as detailed in lines 2 to 20 of Algorithm 5. This is, therefore, a composed neighborhood, where one or two moves are allowed. On line 3, $\mathscr{R}(\mathcal{P})$ is the notation used for a random permutation of the set of products $\mathcal{P}$.

Algorithm 5: Move during each tabu search iteration

TABUMOVE($\bar{y}, \bar{y}^*, i, \Theta$)

(1) $\bar{y}' := \bar{y}$
(2)   **for** $t = 1$ **to** T
(3)   **foreach** $p \in \mathscr{R}(\mathcal{P})$
(4) $\mathcal{S} := \{m \in \mathcal{M}^p : \bar{y}_{pmt} = 1\}$
(5) $\mathcal{U} := \{m \in \mathcal{M}^p : \bar{y}_{pmt} = 0\}$
(6) $d := \mathscr{R}[1, |\mathcal{P}| \times |\mathcal{M}| \times |\mathcal{T}|]$
(7)   **foreach** $m \in \mathcal{S}$
(8)   fix $\bar{y}_{pmt} := 0$
(9)   **if** $\bar{y}$ is better than $\bar{y}^*$ **or** $(i - \Theta_{pm} > d$ **and** $\bar{y}$ is better than $\bar{y}')$
(10)   **return** $\bar{y}$
(11)   **if** $i - \Theta_{pm} > d$ **and** $(\bar{y}^c$ is not initialized **or** $\bar{y}$ is better than $\bar{y}^c)$
(12) $\bar{y}^c := \bar{y}$, $m_1 := (p, m, t)$
(13)   **foreach** $m' \in \mathcal{U}$
(14)   fix $\bar{y}_{pm't} := 1$
(15)   **if** $\bar{y}$ is better than $\bar{y}^*$ **or** $(i - \Theta_{pm} > d$ **and** $\bar{y}$ is better than $\bar{y}')$
(16)   **return** $\bar{y}$
(17)   **if** $i - \Theta_{pm} > d$ **and** $(\bar{y}^c$ is not initialized **or** $\bar{y}$ is better than $\bar{y}^c)$
(18) $\bar{y}^c := \bar{y}$, $m_1 := (p, m, t)$, $m_2 := (p, m', t)$
(19)   restore $\bar{y}_{pm't} := 0$
(20)   restore $\bar{y}_{pmt} := 1$
(21) $\alpha := \mathscr{R}$
(22)   un-initialize $\alpha\%$ of the $\bar{y}^c$ variables
(23)   **if** $\bar{y}^c$ is not initialized
(24)   select a random index (p, m, t)
(25) $\bar{y}^c := \bar{y}$, $\bar{y}^c_{pmt} := 1 - \bar{y}_{pmt}$, $\Theta_{pm} := i$
(26)   **else**
(27) $(p, m, t) := m_1$, $\Theta_{pm} := i$
(28)   **if** m_2 is initialized
(29) $(p, m, t) := m_2$, $\Theta_{pm} := i$
(30) $\bar{y} := $RECONSTRUCT$(\bar{y}^c)$
(31)   **return** $\bar{y}$

If a neighbor improving the best solution could be found, it is returned immediately. A neighbor solution is immediately returned also if it is not tabu and it improves the input solution (lines 9–10 and 15–16).

In the case that no improving move could be found in the whole neighborhood, we force a diversification: the solution is partially destructed, as shown in lines 21 and 22 ($\mathscr{R}$ is the notation used for a random real number with uniform distribution in $[0, 1]$).

The best found move is then applied and made tabu (lines 27 to 29), and the solution is reconstructed (line 30). Notice that this move is excluded from the parts of the solution that are to be reconstructed.

Lines 23 to 25 prevent the case where the search is blocked, all moves being tabu; in such a case, a random move is taken.

5   Computational Results

The implementation of the hybrid tabu search is somewhat tricky, as it involves the interaction with LP and MIP solvers; but for the MIP class of problems there is no alternative. The programs were implemented in the Python language [7], making use of an interface to the GLPK optimization library [4]. The programming the code was written in just a few pages of code; it is available, together with the mathematical programming model, in [5].

5.1   Practical Benchmarks

The strategies described in this paper were tested on a series of benchmark instances. These instances have a random component, but are derived from a real-world problem, where there are 12 products involved and the number of periods is 12. We have produced an instance based on this full-size problem, and smaller instances by reducing the number of periods and randomly selecting a subset of products. The machines made available on smaller instances are those compatible with the selected products, and the demand is randomly drawn, the average being the true estimated demand.

The characteristics of the selected instances are presented in table 1; these instances are available for downloading in [5].

Table 1. Names and characteristics of the instances used for benchmarking

Name	Number of periods	Number of products	Number of integers	Number of variables	Number of constraints
fri-02	2	2	20	56	45
fri-05	5	5	135	334	235
fri-07	7	7	210	536	369
fri-09	9	9	306	796	554
fri-12	12	12	492	1300	857

The results obtained both by the branch-and-bound solver available in the GLPK kit and by the metaheuristics presented in this paper are presented in table 2. In both the situations, the search was limited to a CPU time of one hour in a computer running Linux 2.4.27-2-686, with an Intel Pentium 4 processor at 1.6 GHz.

Notice that as there is a random component on the relax-and-fix-one-product heuristic, the solutions found can be different from run to run. However, due to the structure of the costs on these instances, the solution is virtually always the same.

5.2   Algorithm Behavior

For providing a deeper insight on the behavior of the algorithm, we also present graphics of the evolution of the current solution with respect to the number of

Table 2. Results obtained by the relax-and-fix-one-product heuristic, by the hybrid tabu search, and by time-constrained branch-and-bound. Branch-and-bound and the hybrid tabu search were limited to 3600 seconds of CPU time for each instance. (* *fri-02* was solved by branch-and-bound to optimality in less that one second.)

Name	Relax-and-fix (average)		Hybrid tabu search sol.			branch-and-
	time (s)	solution	worst	average	best	bound best sol.
fri-02	< 1	13.897	13.897	13.897	13.897	13.897*
fri-05	1.4	49.904	48.878	48.878	48.878	48.878
fri-07	2.3	131.095	126.865	126.197	126.030	127.604
fri-09	4.6	213.405	209.201	208.303	207.640	235.125
fri-12	12.4	277.451	275.004	274.681	273.963	431.660

iterations, for a typical run. We have selected the instance *fri-12*, and analyzed the evolution of the objective value of the current solution, z, as well as that of the best solution, z^*. To illustrate the importance of the destruction/reconstruction phase, we have plotted in figure 1 this evolution for the hybrid tabu search algorithm, and also for pure tabu search, without that phase.

These graphics show the importance of the destruction/reconstruction phase, as without it tabu search quickly moves from the initial (relax-and-fix-one-product) solution into poor areas, and cannot easily reach good solutions again. This is due to the fact that a large number of moves is required to change a good solution into another good solution. Partial destroying the solution and reconstructing it with the relax-and-fix-one-product heuristic can do the large number of changes in the solution that is required to bring it to good places. This phase is used whenever tabu search cannot find an neighbor improving the current solution; if improving neighbors are found, the destruction/reconstruction cycle is skipped.

On this run of the hybrid metaheuristic the number of iterations allowed in the 3600 seconds CPU time was 178; on 164 of these the solution was destructed and reconstructed with relax-and-fix-one-product. There were 10 improvements on the best solution found; 1 was found stopping production in a machine on the current solution, 7 were found starting production in a different machine, and 2 were found on reconstruction. Although these values vary widely from run to run and with the instance, in general there can be observed (as in this case) solution improvements on both the neighborhood search and the destruction/reconstruction phase.

5.3   Other Benchmarks

Although the algorithm was designed for a specific application, as it encompasses most of the relevant issues in big bucket lot sizing, it can be used with no modification for other applications. We have selected some of the relevant (big-bucket) benchmarks from the LOTSIZELIB [10], adapted them to allow backlog (though at a very high penalty) and used hybrid tabu search to solve these instances on the same computational setup used in section 5.1.
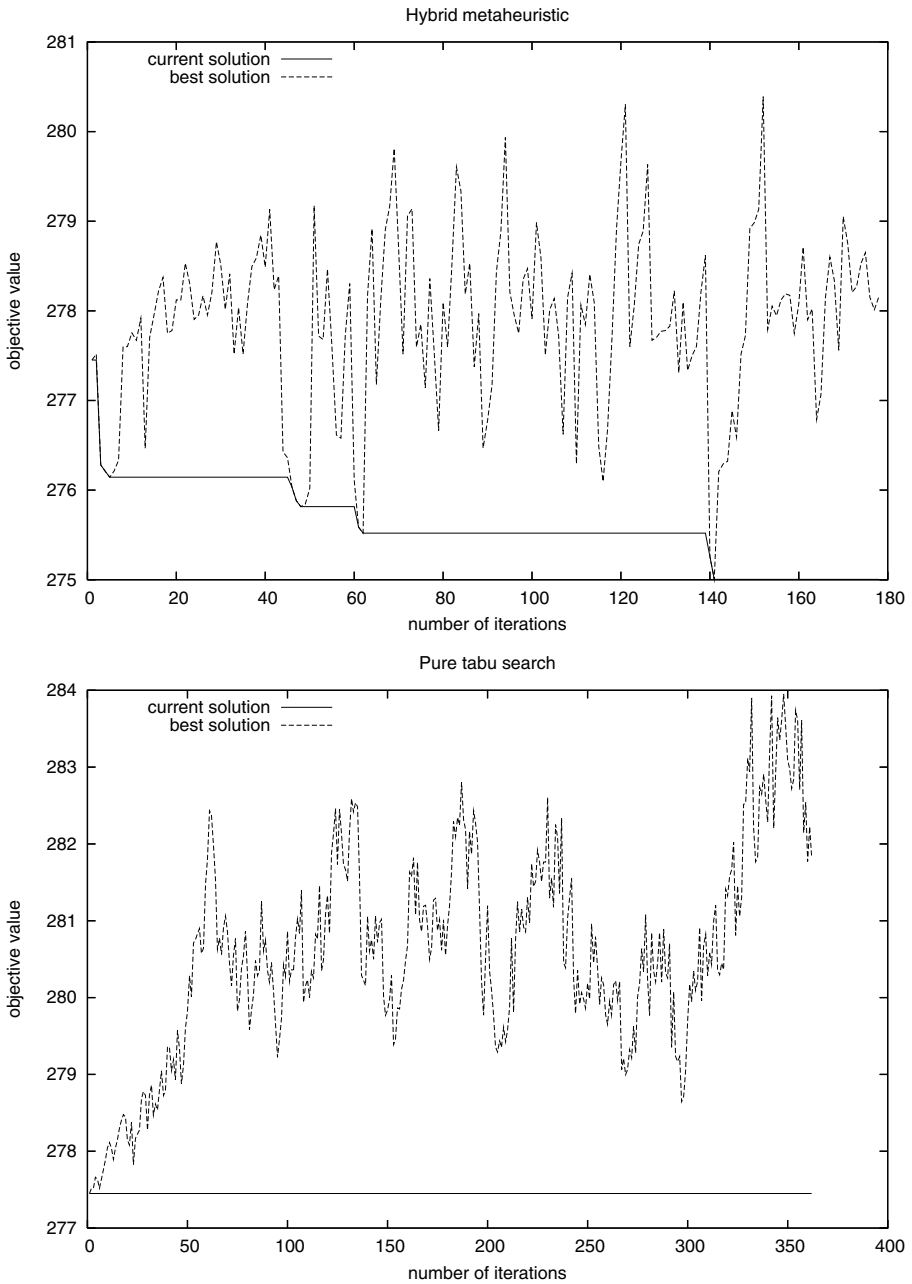
Fig. 1. Objective value as a function of the iteration number for the *fri-12* instance. The top graphic corresponds to the hybrid meta-heuristic; the bottom one corresponds to pure tabu search, without the destruction/reconstruction steps. Both algorithms run 3600 seconds.

Notice that by allowing backlog, we are making these benchmarks more difficult than the original ones.

Table 3. Results obtained by the relax-and-fix heuristic, by the hybrid tabu search, and by time-constrained branch-and-bound on LOTSIZELIB instances. Branch-and-bound and the hybrid tabu search were limited to 3600 seconds of CPU time for each instance. Optimal solutions are as reported in [1]. (* indicate best known solutions, optimality is not proven on these cases.)

Name	Relax-and-fix (average)		Hybrid tabu search sol.			branch-and-	optimal
	time (s)	solution	worst	average	best	bound best sol.	solution
pp08a	<1	7638.0	7380	7374	7360	7350	7350
rgna	<1	82.2	82.2	82.2	82.2	82.2	82.2
set1ch	13.4	56024.3	55243.5	55089.6	54950	60517.7	54537
tr6-15	1.3	40767.6	38357	38238	38054	39388	37721
tr6-30	5.1	67057.0	63422	63246.2	63132	63711	61746*
tr12-30	69.1	143014.0	137371	136762.8	136299	1940337	130599*

6   Conclusion

The main motivation for this work was the exploitation of the quality of the well-known relax-and-fix heuristic for lot sizing problems, in a setup which enforced diversity. This setup was provided by a tabu search mechanism, which was responsible for imposing some changes on the solution. After these changes were made, a part of the solution (not involving the latest changes) was destructed, and relax-and-fix was used to rebuild it.

The reason why this was required as a complement to tabu search is that non-improving moves made by tabu search rapidly force the solution into rather poor regions, because a large number of moves is required to change a good solution into another good solution. These "moves" were done by the relax-and-fix heuristic whenever tabu search could not find an neighbor improving the current solution. When improving neighbors were found, the destruction/reconstruction cycle were skipped.

The computational results obtained with hybrid tabu search on a series of benchmarks show a clear advantage of this strategy, as compared to the simple relax-and-fix-one-product heuristic and to time-limited branch-and-bound.

This work has raised several issues, which remain as topics for future research. The first is the assessment of the quality of the hybrid algorithm using a specialized branch-and-cut system, as the one provided in [1], for the solution of MIPs, instead of branch-and-bound. Another open question concerns limiting the CPU used on each MIP solution; as most of the CPU is used for proving optimality (which is not required in this context), limiting it would probably lead to significant improvements.

The tabu search framework was designed with only short term memory. This provided solutions which are good enough for the practical application to which

the algorithm was designed, but if a deeper search is required it might be useful to implement more sophisticated methods, including long term memory and periodic restart from elite solutions.

References

1. G. Belvaux and Laurence A. Wolsey. Modelling issues and a specialized branch-and-cut system bc-prod. Discussion Paper 9849, Center for Operations Research and Econometrics, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1998.
2. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Boston, 1997.
3. R. Kuik and M. Salomon. Batching decisions: Structure and models. *European Journal and Operational Research*, 75:243–260, 1994.
4. Andrew Makhorin. *GLPK – GNU Linear Programming Kit.* Free Software Foundation, http://www.gnu.org, 2005. Version 4.8.
5. João P. Pedroso. Hybrid tabu search for lot sizing problems: an implementation in the Python programming language. Internet repository, version 0.1, 2005. http://www.ncc.up.pt/~jpp/lsize.
6. Yves Pochet and Laurence A. Wolsey. Algorithms and reformulations for lot sizing problems. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 20, pages 245–293, 1995.
7. Guido van Rossum et al. *Python Documentation.* PythonLabs, http://www.python.org, 2005. Release 2.3.5.
8. Mathieu Van Vyve and Yves Pochet. General heuristics for production planning problems. *INFORMS Journal on Computing*, 16:316–32, 2004.
9. Laurence Wolsey. *Integer Programming.* John Wiley & Sons, 1998.
10. Laurence A. Wolsey. LOTSIZELIB. Internet repository, version 3.0, 1996. http://www.core.ucl.ac.be/wolsey/lotsizel.htm.