A Tabu-Search Heuristic for the Capacitated Lot-Sizing Problem with Set-up Carryover

Mohan Gopalakrishnan • Ke Ding • Jean-Marie Bourjolly • Srimathy Mohan
School of Management, Arizona State University West, MC 2451, PO Box 37100, Phoenix, Arizona 85069-7100
Prestige Telcomm. Ltd., 575 Boul. Morgon, Baie d'Urfe, PQ, Canada H9X 3T6
Department of Decision Sciences & MIS, Concordia University, 1455 de Maisonneuve Blvd. West Montreal, PQ, Canada H3G 1M8
School of Management, Arizona State University West, MC 2451, PO Box 37100, Phoenix, Arizona 85069-7100
mohan@asu.edu • kding@prestige-tel.com • brjolly@vax2.concordia.ca • srimathy.mohan@asuwest-online.asu.edu

This paper presents a tabu-search heuristic for the capacitated lot-sizing problem (CLSP) with set-up carryover. This production-planning problems allows multiple items to be produced within a time period, and setups for items to be carried over from one period to the next. Two interrelated decisions, sequencing and lot sizing, are present in this problem. Our tabu-search heuristic consists of five basic move types—three for the sequencing decisions and two for the lot-sizing decisions. We allow infeasible solutions to be generated at a penalty during the course of the search. We use several search strategies, such as dynamic tabu list, adaptive memory, and self-adjusting penalties, to strengthen our heuristic. We also propose a lower-bounding procedure to estimate the quality of our heuristic solution. We have also modified our heuristic to produce good solutions for the CLSP without set-up carryover. The computational study, conducted on a set of 540 test problems, indicates that on average our heuristic solutions are within 12% of a bound on optimality. In addition, for the set of test problems our results indicate an 8% reduction in total cost through set-up carryover.
(*Lot Sizing; Tabu Search; Set-up Carryover*)

1. Introduction

In this paper, we present a tabu-search heuristic for a single-level, multi-item, capacitated lot-sizing problem (CLSP) with set-up carryover. The lot-sizing problem we are considering involves the production of P items on a single machine for a planning horizon of T time periods. In each time period, the multiple items compete for limited production capacity. Production of item i in period t, X_{it}, incurs a unit production cost c_{it} and consumes b_i units of capacity. End-of-period inventory I_{it} is carried into the next period at a per-unit cost of h_{it}. Production changeovers from one item to another incur a product-dependent set-up time, st_i and set-up cost, sc_i. Set-up carryover from one period to the next can take place under the following circumstances:

a) the same item is produced last in period t and first in period $t+1$;

b) a sufficiently large idle time at the end of period t is utilized to set up the machine to produce the item scheduled first in period $t+1$; and

c) an idle period t is utilized to set up the machine to produce the item scheduled first in period $t+1$, or to maintain the state of the machine so that it is ready to produce the item scheduled first in period $t+1$.

The objective of the CLSP with set-up carryover is to obtain a feasible production schedule that minimizes the sum of the production, inventory, and set-up costs

0025-1909/01/4706/0851$5.00
1526-5501 electronic ISSN

while allowing set-ups to be carried over from one period to another.

In the CLSP with set-up times, a set-up has to be performed whenever an item is produced (Trigeiro et al. 1989). Hence, it is a simple task to keep track of the number of set-ups in a period, which is equal to the number of products produced in that period. However, in our problem, we have to keep track of the set-up-carryover information from one period to the next by partially sequencing the items produced in each period, specifying the items produced first and last and keeping track of the machine state (i.e., knowing which item the machine is ready to produce) at the end of a period. Thus, the CLSP with set-up carryover is composed of two interrelated decisions, i.e., lot sizing and partial sequencing.

One of the main issues arising in CLSP research is the modeling of set-up times. In the literature, we find two distinct methods to incorporate set-up times. The first method uses set-up costs in the objective function as a surrogate for set-up times. The second method includes both set-up costs in the objective function and set-up times in the capacity constraints. In addition, incorporating set-up carryover is critical for many manufacturing environments in which production changeovers take significant time (Trigeiro et al. 1989). Besides, when production capacity is tight, set-up carryover may make the difference between obtaining a feasible schedule or not.

The CLSP is known for its computational intractability. Florian et al. (1980) have shown that the general case of the single-item CLSP is NP-hard. Trigeiro et al. (1989) state that when set-up times are introduced in the multi-item CLSP, even the feasibility problem becomes NP-complete. When set-up carryover is also included, the resulting problem becomes even harder.

For the CLSP with set-up times, a number of researchers have developed heuristic solution procedures. We present below a few of the heuristics most relevant to our research. Aras and Swanson (1982) have developed a heuristic algorithm to determine both lot sizes and job sequencing for the CLSP with set-up times. Their formulation does not include set-up costs, and the model minimizes inventory costs subject to capacity and demand constraints. Their framework considers inventory costs incurred during the production period also, and hence, they have to determine the sequence of products in each period. This sequencing information is used to carry set-ups between periods. However, the main driver of the algorithm is the total inventory cost and not the number of set-ups.

Trigeiro et al. (1989) have developed a Lagrangian relaxation-based heuristic for the CLSP with set-up times. They relax the capacity constraints, and the resulting problem decomposes into a set of independent, uncapacitated, single-item problems. Their algorithm uses dynamic programming to solve the single-item problems, a production-smoothing heuristic to fix capacity violations, and a subgradient optimization procedure to strengthen the lower bound. The solution gap between the heuristic and the lower bound for a large number of test problems is 2.6% on average. Lozano et al. (1991) have solved the Lagrangian dual problem of the CLSP with set-up times using a primal-dual algorithm that provides monotonously convergent solutions to the Lagrangian optimum. They ensure the feasibility of the production plan through a heuristic that performs both lot splitting and lot shifting. Diaby et al. (1992) have proposed another Lagrangian relaxation-based heuristic procedure for solving CLSP with set-up times, limited regular time, and limited ovetime. A similar work by Anderson and Cheah (1993) solves the CLSP with minimum batch size and set-up times.

Salomon et al. (1993) report a procedure based on column generation for solving the single-level CLSP with set-up times. They use simulated annealing or tabu search to generate the new columns to be added to the master problem. They have tested their procedure on test problems in Trigeiro et al. (1989), and have shown that their solution quality is similar to that of Trigeiro et al., but the computational times are much larger.

There are a few formulations for modeling set-up carryovers in a small time-bucket setting. Here, the length of each time period is very small and the entire period is used for a set-up or for producing at most one item, or the machine is idle. In this setting modeling set-up carryover is relatively straightforward. Cattrysse et al. (1993) have developed a dual-ascent

and column-generation-based heuristic to schedule products in a small time-bucket setting. Applying such a modeling approach to the large time-bucket setting would tremendously increase the number of integer variables. In such scenarios, mathematical programming-based heuristics developed for the small time-bucket setting would be very inefficient. Hence, it is necessary to model set-up carryovers in a large time-bucket setting and develop solution procedures for the same.

Gopalakrishnan et al. (1995) have developed a mixed-integer linear-programming (MILP) framework for the CLSP with constant set-up time and set-up carryover in a large time-bucket setting. This model was motivated by a scheduling problem in a large paper mill, where items (custom embossed napkins) were produced on multiple machines. Changeovers between items incurred a large constant set-up time. The paper mill's problem was solved using a branch-and-bound procedure. Details of the application are contained in Gopalakrishnan et al. (1995). Gopalakrishnan (2000) has extended the MILP model to the product-dependent set-up time scenario (model CLSPSC). This framework is applicable in a wider variety of manufacturing situations compared to the constant set-up time framework, thus improving the relevance to management.

In this paper, we present TABU-CLSPSC, a tabu-search-based heuristic for the CLSP with setup carryover. Tabu search is a metaheuristic that makes use of memory structures and exploration strategies based on information stored in memory to search beyond local optima. The procedure repeatedly moves from a solution to the best among its neighboring solutions. To prevent cycling, the procedure stores recently visited solutions in a continuously updated "tabu list" for a given number of iterations and does not visit them as long as they are in the list. Several strategies such as diversification and intensification are used to make tabu search an effective and robust procedure (Hertz et al. 1997, Glover and Laguna 1993, Glover et al. 1993, 1998). Researchers have successfully developed tabu-search heuristics for combinatorially difficult problems in several domains such as vehicle routing (Gendreau et al. 1994) and flow-shop scheduling (Dell'Amico and Trubian 1993).

TABU-CLSPSC moves from a given solution to a neighboring solution using one of the five basic moves that define the neighborhood of the current solution. The moves involve swapping items within a period, moving set-ups from one period to either an earlier or a later period, and moving production lots from one period to an earlier or a later period. Our algorithm allows solutions that violate the capacity constraints but penalizes such solutions. The algorithm also incorporates an adaptive memory scheme proposed by Rochat and Taillard (1995) that has probabilistic intensification and diversification features. We have also developed TABU-CLSP, a tabu-search-based heuristic for the CLSP with set-up times. We have modified TABU-CLSPSC and have developed a move specifically for TABU-CLSP. We have tested both our heuristics on a large set of test problems used by Trigeiro et al. (1989). Our results indicate that we have good heuristics for the CLSP with set-up times and set-up carryover.

The rest of the paper is organized as follows. Section 2 describes the mechanics of the five basic moves in detail. We describe the different components of the TABU-CLSPSC and TABU-CLSP heuristics in §3. Section 4 provides details and results of our computational study, and §5 provides a summary and conclusion.

2. Description of the Basic Moves

This section contains the description of the five moves that help the search process move from a current solution to a neighboring solution. We introduce the following notation before describing the solution representation and the moves.

- X_{it} – production quantity of item i in period t;
- α_t – item type $(1, 2, \ldots, P)$ produced first in period t ($\alpha_t = 0$, if period t is idle);
- β_t – item type produced last in period t ($\beta_T = 0$, if period t is idle);
- γ_t – machine state (item type that the machine is ready to produce) at the end of period t;

where $i = 1, \ldots, P$, and $t = 1, \ldots, T$.

Using the solution representation defined above, we can determine the number of items produced in

period t by counting the number of X_{it} variables that are nonzero. The number of items produced in period t together with the values of γ_{t-1}, α_t, β_t, γ_t, and α_{t+1} determine the set of items that are set up in period t. For example, suppose Items 1 and 2 are produced in period t. If $\gamma_{t-1}=1$, $\alpha_t=1$, $\beta_t=2$, $\gamma_t=3$, and $\alpha_{t+1}=3$, then Items 2 and 3 are set up in period t. We determine the ending inventory of item i in period t as: $I_{it}=X_{it}+I_{i(t-1)}-D_{it}$, where D_{it} is the demand for item i in period t, and $I_{i(t-1)}$ is the inventory of item i at the end of period $t-1$.

Given a solution X, we can move to a neighboring solution X' by using one of the following five moves. The first three moves operate on the partial-sequencing variables α_t, β_t, γ_t, and the last two moves operate on the lot-sizing variables, X_{it}.

2.1. Swap Items Within a Period (SWAP)

The first type of move, SWAP, swaps the items produced first and last in a given period, or swaps one of these with some other item produced in that period. We can perform a SWAP move for any period in which at least two items are produced. A SWAP move could reduce the number of set-ups in two adjacent periods through a set-up carryover. Following SWAP, the values of α_t and β_t, as well as γ_{t-1} and γ_t, are updated.

Assume all P items are produced in each of the T period. Then, in each period, one can perform $P-2$ swap moves between some item produced in between and the item produced first, $P-2$ similar swaps involving the item produced last, and *one* swap move between the items produced first and last. Hence, the size of the subneighborhood corresponding to the swap move is at most $((P-2)+(P-2)+1)T=(2p-3)T=O(PT)$.

2.2. Move Set-up at the Beginning of a Period to an Earlier Period (SETUP1)

This move shifts the first set-up in period t to the end of period $t-1$, when the machine state at the end of period $t-1$ is different from the item produced first in period t. This move creates the "end-of-period" set-up and can affect the capacities in two adjacent periods. If period t is not idle, SETUP1 simply assigns the value of α_t to γ_{t-1}. If it is idle, it assigns the value of γ_t to γ_{t-1}. The motivation behind this move is to help carry a set-up over a series of one or more adjacent idle periods. The size of the subneighborhood associated with this move is at most $T-1=O(T)$, because we can perform at most one move in each of the first $T-1$ periods.

2.3. Move End-of-Period Set-up to the Following Period (SETUP2)

The third move, SETUP2, is the inverse of the second move. This move shifts the end-of-period set-up in period t to the beginning of period $t+1$, when the item produced last in period t is different from the machine state at the end of that period. SETUP2 removes an end-of-period set-up in period t by moving it to the following period, thereby reducing or avoiding a possible capacity violation in period t. If period t is not idle, SETUP2 assigns the value of β_t to γ_t; if period t is idle it assigns the value of γ_{t-1} to γ_t. The size of the subneighborhood associated with SETUP2 is also $O(T)$.

2.4. Move Production Lot to an Earlier Period (LOT1)

The fourth move, LOT1, shifts a certain amount of production of an item (say, i) from period t to an earlier period s, where $1\le s\le t-1$. Depending on the available capacity in period s, we can move either the entire lot of item i or just a fraction of it to period s. LOT1 could reduce capacity violations in period t through set-up reduction or lot shifting.

Before a lot is moved, we check period s for the available capacity. If item i is already produced in period s, no new set-up is needed and all the slack capacity available in period s can be used to produce more of item i. If sufficient slack capacity does not exist in period s to absorb enough of item i from period t, we may allow the entire lot to be transferred to period s. We consider the tradeoff between the penalty for violating the capacity in period s along with the inventory carrying cost for item i resulting from the lot shift, and the reduction in the set-up cost and capacity violation in period t, while performing LOT1.

If item i is not originally produced in period s, in addition to considering the cost trade-off described above, we need to consider the following two

situations:

- Two or more items are already produced in period s, and the new item i is produced inbetween. Hence, a new set-up is needed, and the amount of extra production capacity in period s equals the amount of slack capacity minus the time to perform one set-up for item i. We maintain the values of the partial-sequencing variables in periods t and s by not producing item i first or last in period s.
- Fewer than two items are produced in period s. Here, we need to examine the partial sequences in periods s and $s-1$ to determine the number of set-ups for period s. Note that the value of some of the partial-sequencing variables, γ_{s-1}, γ_s, α_s, and β_s might also change. Suppose, for example, that only one item (say, j) is produced in period s originally, and that the machine is already set up to produce item i at the end of period $s-1$ (i.e., $\gamma_{s-1}=i$). Following the transfer, if we produce item i first in period s (to take advantage of the set-up carryover), α_s changes from j to i, and the value of β_s remains the same (j).

Assume all P items are produced in each of the T periods. Because we can shift the lot for any item in period t to at most $t-1$ previous periods ($t=2,\ldots,T$), the size of the subneighborhood corresponding to the LOT1 move is at most $(1+2+\cdots+(T-1))P=O(PT^2)$.

2.5. Move Production Lot to a Later Period (LOT2)

The fifth move, LOT2, is the inverse of LOT1. Here, a certain amount of production for an item i in period t is shifted to a later period s ($t+1\le s\le T$), provided there is a positive ending inventory for item i in period t and zero ending inventory for the same item in period s. Also, the amount of production shifted (whole lot or a fraction) depends on available capacity in period s, determined as in LOT1. Note that for LOT1 and LOT2, if we shift a whole batch for item i out of period t and item i was scheduled to be produced first or last in period t, then we need to change the values of some of the partial-sequencing variables for periods t and $t-1$. Because we can move the lot for an item in period t to at most $T-t$ later periods ($t=1,\ldots,T-1$), the size of the subneighborhood is at most $(1+2+\cdots+(T-1))P=O(PT^2)$.

3. Description of the Heuristic

The overall TABU-CLSPSC algorithm contains the following three components.

- An *initial solution generator* that provides twelve different initial solutions.
- A *basic tabu search (BTS)* that works on each of the initial solutions using the five moves described earlier for a given number of iterations and develops improved production schedules. We break up each one of these improved solutions into single-item schedules and load them into a set that we call the adaptive memory.
- A *diversification and intensification phase* that probabilistically selects and combines single-item schedules found in the adaptive memory to form new solutions.

During the diversification and intensification phases, we form new solutions, improve these solutions using the BTS, and load the associated single-item schedules onto the adaptive memory until a stopping criterion is met. Next, we describe each of the components in detail.

3.1. Initial-Solution Generator

For each initial solution, we need to determine production levels for individual items and values for the partial-sequencing variables in each period. We use the following six methods to determine the production levels.

- Use the lot-for-lot approach, where the production level of an item in a given period is its demand in that period.
- Use the Wagner-Whitin method (Wagner and Whitin 1958) of generating a single-item schedule.
- Produce the total demand for an item in the first period with nonzero demand for that item.
- Divide the planning horizon into two halves and produce all the demand for an item in a given half of the planning horizon, in the first period of that horizon.
- Divide the planning horizon into four equal parts, and produce all the demand for an item in a given quarter of the planning horizon, in the first period of that horizon.
- Produce the combined demand of every two periods in the first of the two periods.

Next, we assign initial values to the partial-sequencing variables using one of the following two methods. Both methods assign values to the variables one period at a time. For each period, consider all the items with nonzero production quantity in that period and sort them in ascending order of the production quantities. The first method produces the first item on the list as the first product in the period and the second item on the list as the last product. The second method produces the first item on the list as the first product in the period and the last item on the list as the last product. The end-of-period set-up is always equal to the last product produced in that period (i.e., $\gamma_t = \beta_t$). For idle periods, $\gamma_t = \gamma_{t-1}$. Thus, the six production schedules, together with two different sets of partial-sequencing variables, give twelve initial solutions.

3.2. Basic Tabu-Search Procedure

In this section we describe the important elements of the basic tabu search and then provide a step-by-step description of the search heuristic.

3.2.1. Objective Function. For a given feasible solution X, the objective function $F_1(X)$ consists of inventory costs for carrying inventory across periods, set-up costs incurred during a period, and the production cost for producing item i in period t. This can be expressed as:

$$F_1(X) = \sum_i \sum_t h_{it} I_{it} + \sum_i \sum_t sc_i N_{it} + \sum_i \sum_t c_{it} X_{it}$$

where N_{it} is the integer-variable counting the number of set-ups for item i in period t. Because we allow violations of the capacity constraints during the search process, we define another objective function $F_2(X)$ for solutions violating the capacity constraints as follows:

$$F_2(X) = F_1(X) + p \sum_t \left[\sum_i b_i X_{it} + \sum_i st_i N_{it} - C_t \right]^+,$$

where $[x]^+ = max(0, x)$, p is the penalty rate, and C_t is the capacity available in period t. Note that when a solution X is feasible, $F_1(X) = F_2(X)$. Let F_1^* and F_2^* denote, respectively, the lowest values of $F_1(X)$ and $F_2(X)$ obtained during the search, X^* be the best-known solution (feasible or not) and $\tilde{X}^*$ be the best-known feasible solution.

3.2.2. Penalty Rate. The penalty rate p is related to capacity violations. For any infeasible solution, we penalize the amount of capacity violation in any period by this rate. This penalty rate is initially set to an arbitrary value of 50. The rate is varied during the search, keeping in mind that a high rate may be able to drive out infeasibility quickly, but it may also prevent searching other good regions. Inspired by the work of Gendreau et al. (1994), we use the concept of self-adjusting penalties to produce a mix of feasible and infeasible solutions during the search process. In our basic tabu search, we keep track of the number of infeasible solutions, n, obtained during the previous h iterations. If n is less than $h/2$, we reduce the value of p to $(0.5+n/h)p$, because more than half the solutions were feasible. If n is greater than $h/2$, we increase the value of p to $(2n/h)p$ and do not change the value of p if $n = h/2$. Based on initial experimentation, we set the value of h to 10 in our basic tabu search. We also set minimum and maximum values of 0.5 and 10,000 for p.

3.2.3. Neighborhood Structure. We generate the neighborhood of a given solution X by invoking each one of the five types of moves described in the previous section. Note that when many items, but not the same ones, are produced in any two adjacent periods, the SWAP move can produce many neighboring solutions that are neither better nor worse than the given solution X. This is because many items can be produced first or last, and the objective-function value remains the same for each of these swaps. If none of the other types of moves yield an improving solution, the search process will choose the next solution from the list of solutions produced by the SWAP move. This is because of the existence of a large number of non-tabu SWAP moves even with the use of a long tabu list. Hence, the search process could stagnate. Our initial experimentation confirmed this. To avoid this problem, we invoke the swap move once in three iterations. Thus, we generate the neighborhood of a given solution using all the five moves one-third of the time, and using the last four moves two-thirds of the time.

3.2.4. Tabu-Move Attributes and Tabu List. We represent a tabu move by a set of attributes defined

for each type of move. For the SWAP move, we record the ordinal number of the period t and the item produced first or last (α_t or β_t) in that period before the move. Any SWAP move involving the initial values of α_t or β_t in period t is *tabu* for L iterations after the current one. For SETUP1, the associated move attributes are the ordinal number of the previous period, $t-1$, and the item type involved in the move, α_t. Similarly for SETUP2, the attributes are the ordinal number of the following period, $t+1$, and the machine state at the end of the period, γ_t. Also, because these two moves are the inverse of one another, each time one of them is performed, the reverse move is *tabu* for L iterations after the current one. Similarly, when a lot is shifted to an earlier or to a later period (LOT1 and LOT2), the corresponding move attributes are the item shifted, i, and the ordinal number of the target period to which the lot is shifted (i.e., s). Again, these two moves are inverse of each other, and if one is executed the reverse is *tabu* for L iterations after the current one.

TABU-CLSPSC uses a tabu list with variable length. Based on the work of Dell'Amico and Trubian (1993), the algorithm adjusts the list length L in the following manner: Decrease the length by one if the solution obtained after the move is better than the current one or else increase it by one. Also, the length of the list is maintained within a range $[L_{\min}, L_{\max}]$. After some initial experiments, we have set the values of $L_{\min}$ and $L_{\max}$ to $0.25(P+T)$ and $1.75(P+T)$, respectively.

3.2.5. Stopping Rule. If the number of iterations during which the search progresses without any improvement in the objective function is greater than a prespecified value $k_{\max}$, we stop the basic tabu search. Based on initial experimentation, we set the value of $k_{\max}$ to $1.5(300+10P)$.

3.2.6. Description of the Basic Tabu Search. We now provide a step-by-step description of the basic tabu-search process that tries to improve a given solution X.

Step 0: Initialization

Let $F_1^* \leftarrow \infty$, $F_2^* \leftarrow \infty$. Set the iteration counter $k \leftarrow 1$.

Consider the given solution X. Set $F_2^* = F_2(X)$ and $X^* = X$. If X is feasible, set $F_1^* = F_1(X)$ and $\widetilde{X}^* = X$. Tabu list is empty.

Tabu list length range: Set $L_{\min} \leftarrow 0.25(P+T)$, $L_{\max} \leftarrow 1.75(P+T)$

Penalty rate: Set $p \leftarrow 50, p_{\min} \leftarrow 0.1, p_{\max} \leftarrow 10000, h \leftarrow 10$

Stopping rule: $k_{\max} \leftarrow 1.5(300+10\ P)$

Step 1: Move Evaluation and Best Move Selection

Call SETUP1, SETUP2, LOT1, LOT2, and if ($k \bmod 3) = 0$, call SWAP. Generate all the neighbors X' of X as described in §3.2.3. Do not consider a neighboring solution X' generated by a tabu move unless $F_2(X') \le F_2^*$.

Set $\widetilde{X} = \operatorname{argmin}\{F_2(X')\}$.

Step 2: Tabu List and Solution Update

If $F_2(\widetilde{X}) < F_2^*$, empty the existing tabu list and add the attributes of the move yielding $\widetilde{X}$ (the current move attributes) to the tabu list. If $F_2(\widetilde{X}) \ge F_2^*$, do the following to update the tabu list as explained in §3.2.4.

- If the current list length is less than $L_{\min}$, add the current move attributes to the list.
- Otherwise,
  - If $F_2(\widetilde{X}) \le F_2(X)$: Delete two elements from the beginning and add the current move attributes to the tabu list if list length $> L_{\min}$; delete one element from the beginning and add the current move attributes to the tabu list if list length $= L_{\min}$;
  - If $F_2(\widetilde{X}) > F_2(X)$: and the current list length is less than $L_{\max}$, add the current move attributes to the tabu list if list length $< L_{\max}$; delete one element from the beginning and add the current move attributes to the tabu list if list length $= L_{\max}$.

Set $X \leftarrow \widetilde{X}$. If $F_2(X) \le F_2^*$, set $X^* \leftarrow X$, $F_2^* \leftarrow F_2(X)$.

If $F_1(X) \le F_1^*$, set $\widetilde{X}^* \leftarrow X, F_1^* \leftarrow F_1(X)$.

Step 3: Penalty Rate Update

If ($k \bmod h) = 0$, update the penalty rate p as explained in §3.2.2.

Step 4: Termination Check

If F_2^* has not decreased for the last $k_{\max}$ iterations, stop. Otherwise, go to Step 1.

3.3. Diversification and Intensification

The third component of TABU-CLSPSC is a probabilistic diversification and intensification scheme

motivated by the work of Rochat and Taillard (1995) for the Vehicle Routing Problem. The purpose of this procedure is to provide new starting solutions for the basic tabu search through probabilistic selection and combination of parts of different previously obtained solutions. We break up the previously visited solutions that are available for combination into single-item schedules and store these schedules in a set D that is called the adaptive memory.

To start with, the adaptive memory is made up of single-item schedules obtained by decomposing the twelve initial solutions to the CLSP with set-up carryover, produced by the initial-solution generator and improved by the basic tabu search. Along with each single-item schedule, we also store the set-up-carryover information associated and the objective function value of the solution to which it belongs. Specifically, the set-up-carryover information indicates in which period a set-up is carried over for that particular item. We sort the single-item schedules in the adaptive memory in ascending order based on the objective function values. All single-item schedules belonging to the same solution are then sorted in descending order based on the number of set-up carryovers for each of them. Hence, if we consider two single-item schedules from the same solution, the schedule with more set-up carryovers will be stored before the other single-item schedule. We store a copy of this pool of single-item schedules in set D' and construct a new incumbent solution from D'.

The selection of a single-item schedule for creating a new solution is probabilistic and is biased in favor of "better" schedules (i.e., schedules with lower objective function value and higher number of set-up carryovers). The probability of selecting the ith worst single-item schedule from the set D' is $2^*i/|D'||D'+1|$ (Rochat and Taillard 1995). Once we select a single-item schedule, we try to add it to the new incumbent solution, X. While adding new single-item schedules to the incumbent solution, we check for capacity violation in each period. In order not to be too restrictive and eliminate good single-item schedules, we allow a minimal amount of capacity violation in the incumbent solution. If the selected single-item schedule does not violate capacity in each period by more than $\alpha\%$ ($\alpha = 5$ in our computation), we add it to X. If not, we discard it. After adding the single-item schedule to X, we update the remaining capacity in each period. Also, we remove from D' all schedules from the different solutions corresponding to the item just added to X.

Next, we select another single-item schedule from D' and repeat the process until either all items are scheduled or the set D' is empty. In the latter case, if the new incumbent solution X does not contain schedules for all the items, we construct a complete solution as follows. For every item that is not covered in X, we add to X a corresponding lot-for-lot single-item schedule, i.e., the production level of that item in each period is equal to its demand in that period. This new incumbent solution then serves as a starting solution for the basic tabu-search process. The BTS tries to obtain an improved solution, and the single-item schedules associated with the improved solution are added back into the adaptive memory. This process is repeated until a stopping criterion is met.

We limit the size of the adaptive memory to contain only 8^*P single-item schedules so that it does not grow continuously as the diversification and intensification phases progress. Every time we sort the single-item schedules in D using the scheme described earlier, we also count the number of single-item schedules in D. If the number of single-item schedules is greater than 8^*P, we remove the last $|D| - 8^*P$ schedules from this sorted list. As explained in Rochat and Taillard (1995), once we perform the diversification and intensification process several times, the search tends to concentrate in promising regions of the solution space. This is due to the fact that we choose the schedules probabilistically with a bias towards "better" schedules and the worst schedules are removed from the adaptive memory to maintain its size. Also note that because we allow identical single-item schedules (from different solutions) to be added to the adaptive memory, the "better" schedules are more often used to form the new incumbent solution, and the process slowly changes from a diversification phase to an intensification phase.

3.3.1. Overall Algorithm. We provide below a detailed description of the overall algorithm that

employs the adaptive memory structure to diversify and intensify the search.

Initialization Phase

0. Let $F_1^* \leftarrow \infty$, $F_2^* \leftarrow \infty$, $X^* \leftarrow \emptyset$, and $\tilde{X}^* \leftarrow \emptyset$.

1. Generate 12 initial solutions as described in §3.1.

2. For each initial solution X, perform the basic tabu search to obtain an improved solution X'. If $F_2(X') < F_2^*$, set $X^* = X'$, and $F_2^* = F_2(X')$. Similarly update F_1^* and $\tilde{X}^*$.

3. Decompose X' into single-item schedules, and for each item record the associated objective function value and the number of set-up carryovers.

4. Insert each single-item schedule into the adaptive memory D, which is sorted in an ascending order by objective function value. All single-item schedules belonging to the same solution are sorted in descending order based on the number of setup carryovers for each of them.

5. If the number of schedules in D is greater than 8^*P, remove the last $|D| - 8^*P$ schedules from D.

Diversification and Intensification Phase

Step 0: Initialization

Set the number of nonimproving basic tabu search runs, $c \leftarrow 0$.

Step 1: New Solution

Set $D' \leftarrow D$ and $X \leftarrow$ *empty*.

While D' is not empty

- Choose a single-item schedule $d \in D'$ with a probability of $2^*i/|D'||D'+1|$ for selecting the ith worst single-item schedule, and remove d from D'.
- If adding d to the new incumbent solution X does not violate capacity in each period by more than α% of the capacity limit in that period, add d to X. Otherwise, discard d.
- If d is added to X, update the capacity in each period, and remove from D' all the single-item schedules that correspond to the item d.

If some items are not covered by the schedules in X, construct a complete solution by adding to X the lot-for-lot schedules for the uncovered items. For the new incumbent solution X, perform the basic tabu search to obtain a new improved solution X'.

Step 2: Update

Update the adaptive memory using Steps 3–5 of the initialization phase. If $F_1(X') < F_1^*$, set $\tilde{X}^* = X'$. If $F_2(X') < F_2^*$, set $X^* = X'$, $F_2^* = F_2(X')$, and go to Step 0; otherwise set $c \leftarrow c+1$.

Step 3: Termination Check

If c < 3, go to Step 1. Otherwise stop. If $F_1^* < \infty$, $\tilde{X}^*$ is the best feasible solution; otherwise no feasible solution was found.

3.4. TABU-CLSP Heuristic

We have modified TABU-CLSPSC and have added an additional tailored move to develop TABU-CLSP, a heuristic for the CLSP with set-up times. A comparison between the performance of this heuristic and Trigeiro et al.'s (1989) TTM heuristic would help understand the efficiency of tabu-search techniques for the CLSP.

We use the LOT1 and LOT2 moves from TABU-CLPSC that deal with lot sizing. However, we do not allow set-up carryovers. Whenever an item is produced, it incurs a set-up. We have developed an additional move, LOTSWAP that swaps lots for two different items between two periods. The LOTSWAP move shifts a certain amount of production of an item (say i) from period t to a later period $s(t+1 \le s \le T)$ and simultaneously moves some amount of production of an item (say j, $j \neq i$) from period s to period t. Essentially, this move is a combination of the moves LOT1 and LOT2. The difference is that the amount of an item i that can be moved to a target period is determined by considering the capacity available in the target period after moving out another item j. We move either the entire lot or a part of it, depending on available capacity. The LOTSWAP move attempts to reduce capacity violations of lot merging and lot shifting.

The move attributes are the ordinal numbers of the two periods (t and s) involved in the lot swap. Also, if the move is executed between the two periods, then any other move that attempts to swap items between these periods is considered tabu for the next L iterations, where L is the tabu list length. Assume all P items are produced in each of the T time periods. We can shift the lot or a part of it for any item i in time period t to at most $T-t$ later periods. Because we can shift any of the other $P-1$ items to period t, the size of the subneighborhood corresponding to this move is at most $(1+2+\cdots(T-1))P(P-1) = O(P^2T^2)$.

The TABU-CLSP heuristic is similar to the TABU-CLSPSC heuristic described in §3.3 with the only difference being the definition of the neighborhood in a solution. In the initialization phase, we use only the LOT1 and LOT2 moves for the basic tabu search. In the diversification and intensification phase, we use the LOT1, LOT2, and LOTSWAP moves in the basic tabu search. However, because the subneighborhood of the LOTSWAP move is relatively large, we invoke this move only in 2 out of 10 iterations. All the parameter values are the same as in TABU-CLSPSC.

We tried the LOTSWAP move in TABU-CLSPSC also. After some initial testing, we decided not to use this move for TABU-CLSPSC. Due to the size of the subneighborhood, the additional computation time was excessive and the marginal improvement in solution quality was minimal. In fact, in some cases, the solution produced with LOTSWAP was worse.

4. Computational Study

The purpose of the computational study was threefold. The first objective was to quantify the effectiveness of set-up carryover by using the TABU-CLSPSC heuristic; the second was to test the efficiency of the heuristic; and the third was to compare the performance of TABU-CLSP heuristic with that of Trigeiro et al. (1989). We used the 751 test problems obtained from Trigeiro et al. to calibrate and test our heuristics.

To quantify the effectiveness of set-up carryover, we solved all the problem instances, first using the TABU-CLSPSC heuristic and then the TTM procedure. We define the *effectiveness measure* as the percentage difference between the TTM solution value and the TABU-CLSPSC solution value. This measure gives us an indication of the potential cost savings by incorporating set-up carryovers for the same set of problems.

The quality of the heuristic solution and execution times determine the efficiency of the heuristic. Because optimal solutions are not easy to find, we have developed a lower-bounding procedure for determining the solution gap. This procedure takes advantage of the following fact: We can carry at most one set-up for each of the T time periods and, hence, reduce the number of set-ups in any period by one using at most the CLSPSC framework (Gopalakrishnan 2000) as opposed to the Trigeiro et al. (1989) framework (CLSP).

We can incorporate the reduction in the number of set-ups indirectly into the CLSP model by increasing its capacity in each period. Because at most one setup can be carried, we use the maximum set-up time across all items as the value for increasing the capacity in the CLSP model. Also note that, because we are reducing the number of set-ups in each time period by one, the objective function of the CLSP model has to be reduced accordingly. Here again, because we need a lower bound we use the maximum set-up cost across all items.

Hence, if we add the maximum set-up time to the capacity of each period in the CLSP model and reduce its objective function by (T^* *the maximum setup cost*), the optimal solution to this modified CLSP model is a lower bound for the CLSPSC. However, the CLSP itself is a hard problem, and it is not practical to expect optimal solutions for reasonably sized instances. However Trigeiro et al.'s (1989) Lagrangian procedure provides a lower bound for the CLSP, and hence for the modified CLSP. This lower bound for the modified CLSP serves as a lower bound for the CLSPSC. Now the performance measure is the *optimality gap*, which is the percentage difference between the heuristic solution and the lower bound.

We have also solved all the problem instances using the TABU-CLSP heuristic. We compare the performance of TABU-CLSP with that of the TTM heuristic using the solution values and computation times.

Trigeiro et al. generate their problem instances in three phases. Phase 1 has 70 problems and was used for fine-tuning the parameters of the TTM algorithm, while Phase 2 generates 141 problems by analyzing the different problem characteristics, and Phase 3 has 540 problems that were used to test the algorithm. Table 1 gives the problem sizes and the number of problem instances for each category.

The TABU-CLSPSC and TABU-CLSP heuristics were coded in C++ and run on a PC with a Pentium III, 550 MHz microprocessor. The TTM procedure (provided by the authors) is coded in FORTRAN. It was recompiled and run on the same machine. We used the 70 problems from Phase 1 for initial testing and for fixing the values for the various

Table 1 Number of Problems in the Experimental Design

		Number of periods		
	Number of items	15	30	20
Phase 2	6	116	5	—
	12	5	5	—
	24	5	5	—
Phase 3	10	—	—	180
	20	—	—	180
	30	—	—	180

parameters in our algorithm. Next, we tested the two heuristics on the Phase 2 and Phase 3 problems and present the results below.

4.1. Computational Results

Effectiveness of Set-up Carryover. TABU-CLSPSC found lower-cost solutions when compared to TTM for all problem instances except one in Phase 3. Table 2 summarizes the results for measuring the effectiveness of set-up carryover. As expected, the results show that on average, the total cost of the TABU-CLSPSC solutions is lower than the total cost of the solutions found by the TTM procedure. The average reduction in cost is approximately 24% for the Phase 2 problems, and 8% for the Phase 3 problems. The difference in cost savings is due to the problem sizes in terms of the number of items. Most of the problems (116 out of 141) in Phase 2 contain 6 items and 15 periods, whereas the problems in Phase 3 have 10, 20, and 30 items and 20 periods. The average reduction in total cost becomes smaller as the number of items in a problem instance increases. The reason for this is that at most one set-up can be carried over in each period, and hence for the problems with more items, the reduction in cost due to the set-up carryover accounts for a smaller proportion of the total cost.

Table 2 Effectiveness of Set-Up Carryover

	Effectiveness Measure (%)	Number of items	Number of periods 15	30	20
Phase 2	24.39	6	27.34	23.19	—
		12	11.29	9.48	—
		24	5.59	4.04	—
Phase 3	7.49	10	—	—	14.57
		20	—	—	5.90
		30	—	—	3.38

Efficiency of the TABU-CLSPSC Heuristic. TABU-CLSPSC found a feasible solution for all problem instances. We present the computational results related to measuring the efficiency of our heuristic in Table 3. The first row presents the average statistics for the TTM heuristic, the second row for TABU-CLSP, and the third row for TABU-CLSPSC. The average optimality gap for TABU-CLSPSC is 25.6% for the Phase 2 problems and 12.4% for the Phase 3 problems. Again, the increase in the optimality gap for the Phase 2 problems is due to the large number of problems with 6 items. The average gaps for TABU-CLSPSC are 27.8%, 13.9%, and 6.0% for problems with 6, 12, and 24 items, respectively. The sensitivity of the heuristic to the number of items explains the wide variation in the optimality gaps for Phase 2 and Phase 3. In general, we feel that the slightly large optimality gaps are probably due to the quality of the lower bounds. Our bound is actually a lower bound on a lower bound, and hence the two effects cumulate.

To evaluate the usefulness of the adaptive memory, we kept track of the optimality gap at the end of the initialization procedure. For the Phase 2 problems, TABU-CLSPSC found a feasible solution for all 141 problems at the end of the initialization procedure, and the average optimality gap was 27.6%. This reduced to 25.6% after the probabilistic diversification and intensification procedure. For the Phase 3 problems, initial average optimality gap was 14.6% and 4 problem instances (out of 540) had infeasible solutions. The final gap was 12.4% and all problem instances had feasible solutions. These results indicate

Table 3 Efficiency of the Heuristics

	Phase 2		Phase 3	
	Average Optimality Gap (%)	Average CPU Times (in sec.)	Average Optimality Gap (%)	Average CPU Times (in sec.)
TTM	3.96	0.26	2.29	0.59
TABU-CLSP	3.2	24.8	4.01	97.21
TABU-CLSPSC	25.56	20.76	12.38	81.66

that our neighborhood definition and the basic tabu-search procedures are pretty robust and the adaptive memory helps fine-tune and direct the search process towards more promising regions.

The average computation time is 20.8 seconds for Phase 2 and 81.7 seconds for Phase 3. Computational time tends to increase with an increase in problem size, especially with the number of periods. This is primarily due to the exhaustive search of the neighborhood.

Comparison of TABU-CLSP and TTM Heuristic. Table 3 presents the average optimality gaps and execution times for the two heuristics. The average optimality gap for Phase 2 is 3.2% for TABU-CLSP and 4.0% for TTM. The results from this phase indicate that TABU-CLSP seems to perform better than TTM on average for problems with fewer items.

The average optimality gap for Phase 3 is 4.0% for TABU-CLSP and 2.3% for TTM. TTM produces slightly better solutions as the number of items increases. Because Phase 3 contains a large number of problems with 20 and 30 items, TTM outperforms TABU-CLSP on average. However, TABU-CLSP solutions are equal to or better than TTM solutions for 138 instances in Phase 3. The results indicate that TABU-CLSP compares reasonably with TTM, but the computation times for TABU-CLSP are much larger on average.

5. Conclusion

In this paper, we have described two tabu-search heuristics. The first one, TABU-CLSPSC, is for the CLSP with set-up carryover, which involves both lot-sizing and partial-sequencing decisions. The second heuristic, TABU-CLSP, produces solutions for the CLSP with only set-up times. Results from extensive computational testing indicate that our heuristics are effective in producing good solutions to these problems. TABU-CLSPSC is the only heuristic that we know of, to address set-up carryover for the CLSP in a large time-bucket setting.

The comparable optimality gaps for TTM and TABU-CLSP help us ascertain that our basic tabu-search scheme is robust and quite efficient. Because a very similar scheme is used in TABU-CLSPSC, we feel that our heuristic finds good solutions for the CLSP with set-up carryover. One of the reasons for the slightly large optimality gaps for the TABU-CLSPSC is probably due to the quality of the lower bounds.

Even though our computation times are higher compared to TTM, it is quite a challenge to develop tailored mathematical programming-based heuristics for complicated models such as the CLSP with set-up carryover. In such situations, it is practical to use metaheuristics like tabu search. Another advantage is that this heuristic can be easily extended to handle situations like scheduling on multiple machines.

Acknowledgments

The authors would like to thank the anonymous reviewers and the associate editor for their thoughtful review and valuable suggestions. The first author's work was supported in part by Grant OGP0138205 of NSERC, Canada.

References

Anderson, E. J., B. S. Cheah. 1993. Capacitated lot-sizing with minimum batch sizes and setup times. *Internat. J. Production Econom.* **30–31** 137–152.

Aras, O. A., L. A. Swanson. 1982. A lot sizing and sequencing algorithm for dynamic demands upon a single facility. *J. Oper. Management.* **2** 177–185.

Cattrysse, D., M. Salomon, R. Kuik, L. N. Van Wassenhove. 1993. A dual ascent and column generation heuristic for the discrete lot sizing and scheduling problem with setup times. *Management Sci.* **39** 477–486.

Dell'Amico, M., M. Trubian. 1993. Applying tabu search to the job-shop scheduling problem. *Ann. Oper. Res.* **41** 231–252.

Diaby, M., H. C. Bahl, M. H. Karwan, S. Zionts. 1992. A Lagrangean relaxation approach for very-large-scale capacitated lot-sizing. *Management Sci.* **38** 1329–1340.

Florian, M., J. K. Lenstra, A. H. G. Rinnooy Kan. 1980. Deterministic production planning: algorithms and complexity. *Management Sci.* **26** 669–679.

Gendreau, M., A. Hertz, G. Laporte. 1994. A tabu search heuristic for the vehicle routing problem. *Management Sci.* **40** 1276–1290.

Glover, F., M. Laguna. 1993. Tabu search. C. Reeves, ed. *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publications, Oxford, U.K., 70–150.

———, G. Kochenberger, B. Alidaee. 1998. Adaptive memory tabu search. *Management Sci.* **44** 336–345.

———, E. Taillard, D. de Werra. 1993. A users guide to tabu search. *Ann. Oper. Res.* **41** 3–28.

Gopalakrishnan, M. 2000. A modified framework for modeling setup carryover in the capacitated lotsizing problem. *Internat. J. Production Res.* **38** 3421–3424.

———, D. M. Miller, C. P. Schmidt. 1995. A framework for modeling setup carryover in the capacitated lot sizing problem. *Internat. J. Production Res.* **33** 1973–1988.

Hertz., A., E. Taillard, D. de Werra. 1997. Tabu search. E. Aarts, J.K. Lenstra, eds. *Local Search in Combinatorial Optimization.* John Wiley & Sons, New York, 121–136.

Lozano, S., J. Larraneta, L. Onieva. 1991. Primal-dual approach to the single level capacitated lot-sizing problem. *Euro. J. Oper. Res.* **51** 354–366.

Rochat, Y., E. Taillard. 1995. Probabilistic diversification and intensification in local search for vehicle routing *J. Heuristics* **1** 147–167.

Salomon, M., R. Kuik, L. N. Van Wassenhove. 1993. Statistical search methods for lotsizing problems. *Ann. Oper. Res.* **41** 453–468.

Trigeiro, W. W., L. J. Thomas, J. O. McClain. 1989. Capacitated lot sizing with setup times. *Management Sci.* **35** 353–366.

Wagner, H. M., T. M. Whitin. 1958. Dynamic version of the economic lot size model. *Management Sci.* **5** 89–96.

Accepted by Luk Van Wassenhove; received June 15, 1998. This paper was with the authors 22 months for 2 revisions.