A Tabu Search Heuristic for the Capacitated Lot Sizing Problem with Setup Times and Setup Carryovers

Ke Ding

A Thesis

in

The Faculty

of

Commerce and Administration

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Administration at
Concordia University
Montreal, Quebec, Canada

September 1996

ISBN 0-612-18385-8

ABSTRACT

A Tabu Search Heuristic for the Capacitated Lot Sizing Problem with Setup Times and Setup Carryovers

Ke Ding

The single level capacitated lot sizing problem (CLSP) with setup times and setup carryovers is a class of production planning problems in which multiple items can be produced within a time period, and setups are considered to be carried over from one period to the next. Explicitly handling the setup carryovers is important for those production systems in which production changeovers from one item to another incur significant setup times and/or setup costs. This research focuses on developing a tabu search heuristic for the CLSP with setup times and setup carryovers. In essence, the decision problem is composed of two interrelated decisions, i.e., partial sequencing decision for setup carryovers, and lot sizing decision. In our heuristic, three move types are designed to handle the issue of partial sequencing, and two move types are designed to handle the issue of lot sizing. Together, the five move types define the move mechanisms used for local search. During the course of the search, infeasible solutions are allowed. Global search strategies, such as dynamic tabu list, intensification, and diversification, are employed in our heuristic. A bounding procedure for the problem is also proposed. The computational study conducted on a large number of test problems indicates that our heuristic performs well in solving this computationally hard problem.

ACKNOWLEDGMENTS

First of all, I wish to express my sincerest gratitude to my two thesis supervisors, Dr. Mohan Gopalakrishnan and Dr. Jean-Marie Bourjolly. I am grateful to Dr. Bourjolly for his guidance in algorithm design and heuristic development. I am grateful to Dr. Gopalakrishnan for his guidance in understanding many issues in production planning. Their insight and research experience in developing solution procedures for production management problems greatly benefited me in the development of this heuristic solution procedure. Their support, encouragement and patience demonstrated during the course of this thesis research are greatly appreciated.

I also sincerely thank Dr. Martin Kusy for spending his valuable time with us on discussing this research work.

Finally, I thank my wife for her love and support with which the completion of this thesis becomes possible.

Contents

List of Tables

List of Figures

Chapter 1
INTRODUCTION

Managerial decisions in a manufacturing organization can be represented as a hierarchy of three levels of decisions, i.e., strategic planning, tactical planning, and operational planning and control, as shown in Figure 1 (Bahl et al. 1987). Lot sizing is one of the decisions made at the second level, i.e., at the tactical planning stage of the decision hierarchy. The lot sizing is concerned with the trade-offs between production levels, inventory levels, and frequency of setups. The goal is to optimally determine the production levels and production timings of every individual item to meet all the demand requirements over a given planning horizon.

Different classes of lot sizing problems have been studied in the literature. In this study, we deal with a class of single level, multi-item, capacitated, dynamic lot sizing problems with setup times and setup carryovers on a single facility, and we focus on developing a solution procedure for this class of problems. A Mixed Integer Linear Programming (MILP) model was proposed in a recent study by Gopalakrishnan et al. (1995) to address the Capacitated Lot Sizing Problem (CLSP) with setup times and setup carryovers. A good solution procedure for the model is essential as this would facilitate the application of the model to real world decision problems of this kind. The CLSP is known to be computationally intractable, and hence in this research, we develop a heuristic solution procedure based on tabu search framework. The tabu search heuristic has the elements for both local and global search. The heuristic allows infeasible solution to be visited during the course of the search, and contains five move types that define the move mechanisms used for local search. The main global search elements employed in the heuristic include a tabu list with variable length, an intensification strategy, and two diversification strategies. We also conduct an extensive computational experimentation to evaluate the heuristic's performance.

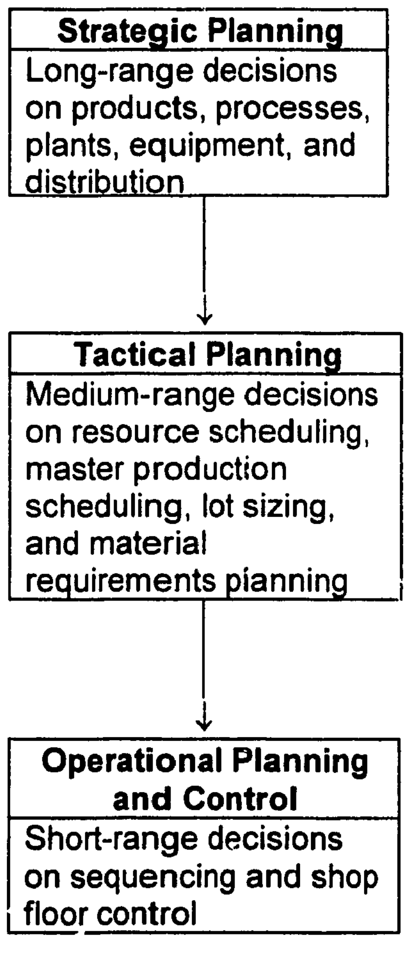
Strategic Planning
Long-range decisions on products, processes, plants, equipment, and distribution
Tactical Planning
Medium-range decisions on resource scheduling, master production scheduling, lot sizing, and material requirements planning
Operational Planning and Control
Short-range decisions on sequencing and shop floor control

Figure 1. Hierarchy of Decisions in a Manufacturing Organization (taken from Bahl et al. 1987)

In this chapter, we introduce the lot sizing problem and its mathematical formulation. We also describe briefly the tabu search technique and the main characteristics of our heuristic, and finally present an outline of this thesis.

1.1 The Capacitated Lot Sizing Problem with Setup Times and Setup Carryovers

Based on two important characteristics, i.e., *demand type* and *resource constraints*, previous research work on the lot sizing can be classified into four categories, as shown in Figure 2 (Bahl et al. 1987). Lot sizing problems can be first classified into *single-level* and *multiple-level* problems based on demand type. Product demands in a single-level environment are *independent*, since they do not depend on lot sizing decisions made for other items. The demands for all the items can be determined by considering customer orders or market forecasts. In contrast, there exist *dependent* demands in a multiple-level environment wherein end products are made from intermediate items, and intermediate items may themselves depend on lower level items. Although, demands for end products can be determined externally so that they are independent, end products generate demands for items at other levels so that for all lower-level items, dependent demands arise. Besides the demand type distinction, lot sizing problems can be further distinguished in terms of the resource constraints, i.e., *capacitated* or *uncapacitated* resources. They are termed respectively as the capacitated, and uncapacitated lot sizing problems.

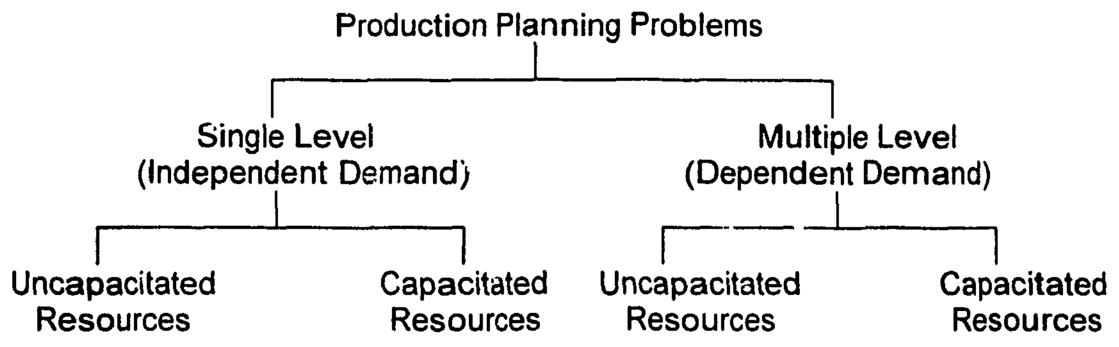

Figure 2. A Classification of Production Planning Problems (taken from Bahl et al. 1987)

The single level, multi-item lot sizing problem with capacity constraints is referred to as the capacitated lot sizing problem (CLSP). There are two approaches to modeling *setups* in the CLSP. One approach handles setups exclusively through *setup costs*. Another approach models setups through incorporating both *setup costs* and *setup times*, i.e., downtimes for performing setups that incur the loss of resource time. For a manufacturing environment in which setup times are not significant compared with production times, it is sufficient to model them with surrogate setup costs. However, for an environment in which setups incur significant times, it becomes necessary to explicitly model the setup times.

In addition, lot sizing problems are addressed in two different time bucket settings. In the *small time bucket* setting, the length of time period is short, and hence it is assumed that at most one item can be produced in any given period (the time periods can be used to either produce, or do a setup, or remain idle). However, in the *large time bucket* setting, the length of time period is assumed to be relatively long so that multiple items can be produced in any given period. In such an environment, if the production changeover from one item to another incurs a significant setup time, it is very advantageous to carry the setups from one period to the next. That is, if the item produced last in a given period is the same as that produced first in the immediately following period, then carrying the setup for this item into the following period will save significant capacity time and setup cost. Furthermore, in certain instances, a feasible lot sizing can be developed only if the setups are carried over from one period to the next.

Taking the above issues into consideration, Gopalakrishnan et al. (1995) proposed a mixed integer linear programming (MILP) model for the CLSP with setup times and setup carryovers in the large time bucket setting. In the model, setup times and setup costs were assumed to be constant across items and time periods. The MILP formulation is presented as follows:

Model M1

Minimize $Z = \sum_i \sum_t H_{it} I_{it} + A \sum_t N_t + \sum_i \sum_t F_t Y_{it}$ (1)

Subject to

$I_{i,t-1} + X_{it} - I_{it} = D_{it}$ $i = 1,\dots,P;\ t = 1,\dots,T$ (2)

$\sum_i b_i X_{it} + q N_t \le C_t$ $t = 1,\dots,T$ (3)

$X_{it} - M Y_{it} \le 0$ $i = 1,\dots,P;\ t = 1,\dots,T$ (4)

$N_t = \sum_i Y_{it} + \sum_i S_{it} + \sum_i V_{it} + \sum_i O_{it} - 1$ $t = 1,\dots,T$ (5)

$S_{it} \ge \gamma_{i,t-1} - \alpha_{it}$ $i = 1,\dots,P;\ t = 1,\dots,T$ (6)

$V_{it} \ge \beta_{it} - \gamma_{it}$ $i = 1,\dots,P;\ t = 1,\dots,T$ (7)

$O_{it} \ge \gamma_{it} - \gamma_{i,t-1} - \omega_t$ $i = 1,\dots,P;\ t = 1,\dots,T$ (8)

$$Y_{it} \leq \omega_t \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (9)$$

$$\sum_i Y_{it} - 1 \leq (P-1)\,\delta_t \qquad t = 1,\dots,T \quad (10)$$

$$\omega_t \leq \sum_i \alpha_{it} \leq 1 \qquad t = 1,\dots,T \quad (11)$$

$$\omega_t \leq \sum_i \beta_{it} \leq 1 \qquad t = 1,\dots,T \quad (12)$$

$$\alpha_{it} \leq Y_{it} \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (13)$$

$$\beta_{it} \leq Y_{it} \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (14)$$

$$\alpha_{it} + \beta_{it} \leq 2 - \delta_t \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (15)$$

$$\sum_i \gamma_{it} = 1 \qquad t = 1,\dots,T \quad (16)$$

$$X_{it},\ I_{it},\ N_t,\ S_{it},\ V_{it},\ O_{it},\ \omega_t \geq 0 \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (17)$$

$$0 \leq \delta_t \leq 1 \qquad t = 1,\dots,T \quad (18)$$

$$Y_{it},\ \alpha_{it},\ \beta_{it},\ \gamma_{it} \in \{0, 1\} \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (19)$$

$$I_{i0} = 0 \qquad i = 1,\dots,P \quad (20)$$

The parameters in the model are:

P	=	the number of items,
T	=	the number of planning periods,
H_{it}	=	inventory holding cost for item i in period t,
A	=	setup cost per setup,
F_t	=	fixed charge for production of an item in period t,
D_{it}	=	demand for item i in period t (units),
b_i	=	capacity consumed per unit production of item i (hours/unit),
q	=	setup time per setup (hours/setup),
C_t	=	machine capacity in period t (hours),

$$M_{it} = \min\left(\sum_{r=t}^{T} D_{ir}, C_t/b_i\right).$$

The decision variables in the model are:

X_{it} = lot size quantity for item i in period t,

I_{it} = inventory for item i at the end of period t,

N_t = number of setups in period t,

$$Y_{it} = \begin{cases} 1, & \text{if item i is produced in period t} \\ 0, & \text{otherwise,} \end{cases}$$

$$\alpha_{it} = \begin{cases} 1, & \text{if item i is produced first in period t} \\ 0, & \text{otherwise,} \end{cases}$$

$$\beta_{it} = \begin{cases} 1, & \text{if item i is produced last in period t} \\ 0, & \text{otherwise,} \end{cases}$$

$$\gamma_{it} = \begin{cases} 1, & \text{if the facility is set up for item i at the end of period t} \\ 0, & \text{otherwise,} \end{cases}$$

$$S_{it} = \begin{cases} 1, & \text{if } \gamma_{it-1} = 1 \text{ and } \alpha_{it} = 0 \\ 0, & \text{otherwise,} \end{cases}$$

$$V_{it} = \begin{cases} 1, & \text{if } \beta_{it} = 1 \text{ and } \gamma_{it} = 0 \\ 0, & \text{otherwise,} \end{cases}$$

$$O_{it} = \begin{cases} 1, & \text{if an idle period t is used to perform a setup} \\ 0, & \text{otherwise,} \end{cases}$$

$$\omega_t = \begin{cases} 1, & \text{if at least one item is produced in period t} \\ 0, & \text{otherwise,} \end{cases}$$

$$\delta_t = \begin{cases} 0, & \text{if exactly one item is produced in period t} \\ 0 < \text{and} \leq 1, & \text{otherwise.} \end{cases}$$

The objective is to minimize the sum of inventory costs, setup costs, and fixed production charge costs, as shown in equation (1). Unit production costs are assumed to be constant for all the items in each period, and hence are ignored. Equations (2) capture the relationship between the opening and closing inventories, production quantity and demand. The capacity constraints are represented by inequalities (3), in which the setup times reduce the available capacity of the constrained resource. Constraints (4) require that $Y_{it} = 1$ whenever item i is produced in period t. Constraints (5) through (8) are *setup counting* constraints. In constraints (5), the number of setups in period t is calculated by considering the values of variables Y_{it}, S_{it}, V_{it}, and O_{it}. S_{it}, V_{it}, and O_{it}, $i = 1,...,P$, are used to track whether or not, in period t, a *setup is carried over from period t-1, an end-of-period setup is done, or a setup is made in an idle period.*

Constraints (9) through (16) are *partial sequencing* constraints. These constraints capture the interrelationships among the variables α_{it}, β_{it}, γ_{it}, ω_t, and δ_t, whose values are used to determine the values of S_{it}, V_{it}, and O_{it} in the setup counting constraints. For any given period, this set of the constraints determines *which item is produced first, which item is produced last, and the machine state at the end of the period.* Constraints (17) through (19) model the nonnegative and binary requirements on the decision variables. The nonnegative requirement on the inventory variables means that backlogging is not allowed. Finally, equations (20) indicate zero starting inventory position.

1.2 Heuristic Solution Procedure for the CLSP with Setup Times and Setup Carryovers

The CLSP is known to be computationally intractable, and it is shown that for the CLSP with setup times, even finding a feasible solution is NP-complete (Maes et al. 1990). Hence, for the CLSP with setup times, most solution procedures reported in the literature are based on some approximate approaches (e.g., Trigeiro et al. 1989, Diaby et al. 1992). When setup carryovers are explicitly modeled in the formulation of the problem, the resulting model becomes even more complex. Therefore, we have attempted to develop a heuristic based on tabu search framework.

Tabu search, developed by Glover (1986), and independently by Hansen (1986), is a *meta-heuristic* that integrates a number of *anti-cycling* mechanisms into a local search procedure to lead the search to explore beyond local optima. In tabu search, moves causing deteriorating objective function values are allowed to be executed under certain circumstances in order to move away from local optima. But this may result in cycling. Tabu search accomplishes anti-cycling by *memorizing* the recently visited solutions in *tabu lists* (sometimes called *recency-based memory*), and forbidding them to be revisited for a number of iterations.

Tabu search can be viewed as a variant of neighborhood search methods. Give a function f(x) to be minimized over a solution space X. Let N(x) be the neighborhood of a solution x, and let L_k be the set of solutions memorized in tabu lists at iteration k. Let N(x, k) = N(x) - L_k, i.e., the restricted neighborhood at iteration k. A generic tabu search heuristic can be outlined as follows:

```
i. choose an initial solution x in solution space X,
      $x^* \leftarrow x$, and $k \leftarrow 1$
ii. **while** stopping criterion is not met **do**
      $k \leftarrow k+1$,
      evaluate $f(x')$ for all $x' \in N(x, k)$,
      choose the best solution, say $x''$, in $N(x, k)$,
      $x \leftarrow x''$,
      **if** $f(x) < f(x^*)$ **then** $x^* \leftarrow x$,
      update tabu list,
   **end while**
iii. return $x^*$
```

Certain essential elements in tabu search include tabu lists, *aspiration criteria*, and *candidate list strategies.* Tabu lists can be implemented with fixed list length (*static tabu list*), or variable list length (*dynamic tabu list*). The *dynamic tabu lists* vary the tabu list length during the course of the search, and usually produce more robust and superior solutions. Aspiration criteria override the tabu status of a move under certain conditions, and hence add flexibility to tabu search. Candidate list strategies limit the number of neighbors evaluated at each iteration, and hence are useful for handling the situation where the neighborhood is large, or the evaluation of its elements is costly.

Sometimes *short-term memory* (or *recency-based memory*) based strategies alone may not be sufficient to produce high quality solutions. *Intensification* and *diversification* are two strategies that operate on *longer-term memory* structure, and are often useful for achieving high quality solutions in a shorter time span, or increasing the probability of finding better solutions for a longer time. To achieve good results, an important issue is to keep a balance between intensification and diversification in a tabu search procedure. A comprehensive overview of tabu search can be found in Glover (1994).

The following issues were dealt with in developing an effective tabu search heuristic for the CLSP with setup times and setup carryovers. (i) The solution space was enlarged by relaxing the capacity constraints, and any violations of the capacity are dealt through penalties applied to the objective function. Our heuristic thus allows infeasible solutions to be visited during the course of the search, and has the flexibility to start a search from any initial solution that meets demand requirements. (ii) Five move types were designed to define the local search. The moves allow setups for products to be moved within and across periods, and lot sizes for products to be moved across periods. (iii) The global search was accomplished through the use of a dynamic tabu list, an intensification approach using an *elite solution list*, a diversification approach using *transition frequency memories* combined with *aspiration by search direction*, a diversification approach using *full restart* from other initial solutions, and a variable penalty rate approach.

1.3 Thesis Outline

This chapter introduced the CLSP with setup times and setup carryovers along with the mathematical formulation of the problem. It also briefly introduced our heuristic approach to solving this problem.

Chapter 2 presents a literature review on the single level CLSP with setup time, and the tabu search applications to production scheduling and planning problems.

In chapter 3, we present the elements relating to local search implemented in our heuristic. These include the solution representation used in our heuristic, the solution space explored during the course of the search, the modified objective function, and the move mechanisms.

In chapter 4, we describe the global search strategies implemented in our heuristic. These include the changing neighborhood at different iterations, the dynamic tabu list, the intensification using an elite solution list, the diversification using transition frequency memories combined with the aspiration by search direction, the diversification with full restart from other initial solutions, and the variable penalty rate applied to violation of capacity constraints. We also propose in this chapter a bounding procedure for finding the lower bounds on the optimal objective value of our problem so that we can have a relative measure for evaluating our heuristic's solution quality.

Chapter 5 presents a computational study in which our heuristic is tested on a large number of problems constructed from the test problems created by Trigeiro et al. (1989). We analyze the gap between the objective value of our tabu search solution and our lower bound. We also analyze the cost difference, which is the difference between the cost of our tabu search solution for the CLSP with setup times and setup carryovers and the cost of the solution found by Trigeiro et al.'s procedure (1989) for the CLSP with setup times. Based on the overall computational results for both the solution gap and the cost difference, we comment on the solution quality achieved by our tabu search solution procedure. Finally, computation times required by our heuristic, and by Trigeiro et al.'s procedure are reported.

Chapter 6 summarizes the research results, and offers some directions for future research.

Chapter 2
LITERATURE REVIEW

In this chapter, we review in the first part the work on the single level, multi-item CLSP with setup times in the large time bucket setting. As mentioned before, the CLSP in the large time bucket setting has been approached in two different ways with regard to modeling setups: setups are introduced as setup costs only (e.g., Eppen and Martin 1987, and Cattrysse et al. 1990), and setups are modeled through both setup costs and setup times. The review here is confined to the literature on the CLSP with setup times, since we aim to develop a solution procedure for this class of the problems. Tabu search is a meta-heuristic technique that has proved to be successful in solving a host of computationally hard problems. In the second part, we review the work of some tabu search applications to production scheduling and planning problems.

2.1 The CLSP with Setup Times

The CLSP with setup times has received continuing research attention due to its practical importance and theoretical challenge. In this section, we present the review on the CLSP with setup times and setup carryovers in the following three sub-sections: the earlier work, the more recent work, and Trigeiro et al.'s work (1989).

2.1.1 The Earlier Work

Modeling setups as downtimes incurred at the capacitated work center in addition to its costs can be traced to Manne (1958). One important issue in modeling has been the presence of binary variables for tracking the setups for each item in each period. Manne proposed a formulation in which binary variables are defined for production schedules for the items, rather than for setups. This formulation approach has one advantage, i.e., a linear programming solution to the model provides a close approximation to the solution of the original problem when the number of items is large compared with the number of time periods in a problem instance.

Dzielinski and Gomory (1965) continued the work by applying the Dantzig-Wolf decomposition to Manne's formulation so that larger problem instances can be solved. Lasdon and Terjung (1971) developed a solution procedure to Manne's formulation using a column generation technique and a generalized bounding method. They demonstrated that their solution procedure was more efficient and provided better bounds than earlier approaches. These researches showed that very large problems could be solved using this modeling approach. The problem with the modeling approach is that the approximation may results in incorrect costs, and infeasible solutions due to setup times.

Newson (1975 a, b) proposed a heuristic for solving the CLSP with setup times that structures the problem as a network of unlimited capacity, and uses an arc-cutting criterion to find feasible integer solutions. The difficult of the heuristic is that it is not very effective in finding feasible solutions when the number of items is not very large, or the capacity constraints are tight. The methods for determining the optimal lower bounds for the CLSP with setup times based on generalized duality theory were discussed in Kleindorfer and Newson (1975). For certain problems, the dual solutions can be used to generate feasible solutions that may be optimal or near optimal.

2.1.2 The More Recent Work

The MILP models for the CLSP in which setup times are explicitly modeled contain a structure that can be exploited by the Lagrangian relaxation technique, and the research in this direction can be found in a number of more recent articles. In general, for a given set of Lagrangian multipliers, Lagrangian relaxation-based heuristic procedures relax the capacity constraints to decompose the models into a set of independent uncapacitated single item lot sizing problems, which then can be solved repeatedly using the efficient Wagner-Whitin dynamic programming algorithm.

Trigeiro et al. (1989) developed a heuristic procedure of this type for solving the CLSP with setup times as well as setup costs. The overall Lagrangian dual problem was solved using subgradient optimization. A smoothing routine that shifts and splits scheduled lot sizes obtained from the Wagner-Whitin algorithm was designed to generate feasible solutions. We will review the article in more detail in sub-section 2.1.3.

Lozano et al. (1991) solved the Lagrangian dual problem of the CLSP with setup times using a primal-dual algorithm that provides monotonously convergent solutions to the Lagrangian optimum. A heuristic performing lot split and lot shift operations at each iteration was proposed in order to eliminate the infeasibility that might appear in the production plans produced by the primal-dual algorithm. It was reported that for a wide range of problems tested, this approach provided better solutions than other methods, including subgradient method, but required greater computation times. The average gap between their solutions and lower bounds was as small as 1.79 percent.

Another Lagrangian relaxation-based heuristic procedure for solving the very large scale CLSP was proposed in Diaby et al. (1992). They considered setup costs, setup times, limited regular time, and limited overtime in their model formulation. Their procedure solved the Lagrangian dual problem using the subgradient optimization. Transportation problems were then formulated using the setup decisions, rather than the production plans, found in the Wagner-Whitin solutions. A procedure called primal perturbation that operates on transportation tableaus was designed to obtain feasible production plans for the original problem. It was reported that very large problems with as many as 500 items and 30 periods were solved within one percent of their lower bounds in a reasonable amount of time.

A similar work by Anderson and Cheah (1993) solved the CLSP with minimum batch size and setup times. Also in their model, overtime is allowed, but no setup costs are considered. The Lagrangian relaxation-based heuristic procedure consists of a dynamic programming algorithm extended from the Wagner-Whitin procedure, a modified subgradient algorithm, and a minimum-cost network algorithm and a network-based heuristic for achieving primal feasibility. High quality solutions were obtained with an average gap of only 0.44 percent.

A procedure for solving the single level CLSP with setup times based on column generation was discussed in Salomon et al. (1993). The master problem in this column generation heuristic is a set partitioning problem, and the sub-problems are a set of single item CLSPs. Embedded in the procedure, simulated annealing or tabu search procedure is used to solve the sub-problems of generating new columns that are added to the master problem. The computational results from testing the procedures on Trigeiro et al.'s test problems show that the solution quality of the column generation procedure is almost the same as that reported in Trigeiro et al. (1989). However, the solution times required by the procedure are much longer than that required by the Trigeiro et al.'s procedure.

Two other studies handled the lot sizing problem in a more general way in that they incorporate both lot sizing and sequencing decisions. Aras and Swanson's heuristic (1982) addressed sequencing of jobs within periods so that the first and last item produced in a period can be determined. If one item is to be produced at the end of one period and at the beginning of next, only one setup is considered to take place. Smith-Daniel and Ritzman (1988) described a mixed integer linear programming formulation for the lot sizing and sequencing decision on a series of several capacitated work centers.

None of the above-mentioned studies consider the setup carryovers in forming their modeling framework. However, in a recent study, Gopalakrishnan et al. (1995) proposed a modeling framework for the CLSP with setup times and setup carryovers. The framework also extends the CLSP model to include multiple machines and tool requirements planning. The MILP model for the single machine case (model M1) introduced in Chapter 1 is based on Gopalakrishnan et al.'s work.

The computational complexity of CLSP has been studied by many researchers. For example, Florian et al. (1980) showed that the general case of single item CLSP is NP-hard. Some special cases of single item problem, such as the problem with nondecreasing setup cost, zero holding cost, nondecreasing production cost, and nonincreasing capacities, are solvable in a polynomial time, but the problem with two items under the conditions similar to the single item problem becomes NP-hard (Bitran, and Yanasse 1982). When setup times are introduced in the multi-item CLSP, even the feasibility problem becomes NP-complete (Maes et al. 1991).

2.1.3 Trigeiro et al.'s Work

We give a more detailed review here on the article by Trigeiro et al. (1989), because, in this study, (i) a bounding procedure for our problem is developed based on the model studied by these authors, (ii) with minor modification on the values of setup times and setup costs, the test problems created by these authors are used in the experimentation of our tabu search heuristic, and (iii) the costs found by our heuristic for the CLSP with setup times and setup carryovers are compared with that found by Trigeiro et al.'s procedure (referred to below as the TTM procedure) for the CLSP with setup times.

The problem studied by these authors is the single level, multi-item CLSP with setup times. Using the notation similar to that in model M1, we can present the problem formulation as follows:

Model M2

Minimize $Z = \sum_i \sum_t H_{it} I_{it} + \sum_i \sum_t A_{it} Y_{it} + \sum_i \sum_t B_{it} X_{it}$ (1)

Subject to

$I_{i,t-1} + X_{it} - I_{it} = D_{it}$ $i = 1,\ldots,P;\ t = 1,\ldots,T$ (2)

$\sum_i b_i X_{it} + \sum_i q_i Y_{it} \leq C_t$ $t = 1,\ldots,T$ (3)

$X_{it} - M\, Y_{it} \leq 0$ $i = 1,\ldots,P;\ t = 1,\ldots,T$ (4)

$X_{it},\ I_{it} \geq 0$ $i = 1,\ldots,P;\ t = 1,\ldots,T$ (5)

$Y_{it} \in \{0, 1\}$ $i = 1,\ldots,P;\ t = 1,\ldots,T$ (6)

$I_{i0} = 0$ $i = 1,\ldots,P$ (7)

where A_{it} is the setup cost for item i in period t, q_i is the setup time for item i, B_{it} is the unit cost for producing item i in period t, and the remaining parameters and variables have the same meaning as in model M1. Note that the cost for a setup depends on the item and the period, and the time taken by a setup varies with the item, so that A_{it}, and q_i are used rather than a constant A, and q. Also note that if the unit costs are stationary across periods, the variable production costs in the objective function can be ignored. One major restriction of the model is that one setup is required for the production of each item in each period, and no setup carryover between any two adjacent periods is accounted for.

In this formulation, if constraints (3) are dualized, for a given set of Lagrangian multipliers, the resulting Lagrangian relaxation of the problem decomposes into a set of independent, single item uncapacitated lot sizing problems that can be solved efficiently using the Wagner-Whitin dynamic program algorithm.

A procedure based on the Lagrangian relaxation was developed by these authors. The procedure consists of a primal, a dual, and a smoothing procedure. At each iteration, the multipliers are obtained in the dual procedure using the subgradient optimization which guarantees that the multipliers converge to their optimal values so that the best lower bound for the relaxation will be attained. And then a solution and a lower bound are found in the primal procedure using the dynamic programming algorithm. The lower bound was used as a relative quality measure for the solutions found by the heuristic.

The solutions found in the primal procedure are usually near-feasible with respect to the capacity constraints, so that a smoothing procedure was designed to modify the plan in order to fit the capacity. The smoothing procedure moves a whole batch, or splits a batch among other periods in order to eliminate excess capacity requirements in the primal plan. The smoothing procedure stops whenever a feasible solution is reached. If the infeasibility still exists after four passes (two for shifting lots to earlier periods, and two for shifting lots to later periods), the attempt is given up, and one goes on to the next dual and primal iteration. If the resulting plan is feasible, it is stored until a better one is found later. When the procedure terminates at a fixed number of iterations, the stored feasible solution, which is the best one ever found, is reported. If no feasible solutions were ever found, it is reported that the problem is considered infeasible by the procedure.

The experimentation of the procedure was conducted in three phases on a variety of randomly generated problems that capture different problem characteristics. The characteristics considered in the testing problems are the demand variability, the variability of capacity consumption per unit production, the setup time level, the setup cost / holding cost ratio, the capacity tightness, and the problem size.

The procedure was first tested on 70 problems in phase 1 for fine tuning the smoothing procedure and subgradient optimization procedure. It was then tested on 141 problems with different characteristics in phase 2. Based on the results obtained in phase 2, it was further tested on 540 problems with five chosen characteristics. A description of how these test problems are generated will be given in chapter 5 when we look at how our test problems are prepared based on the test problems from these authors.

The results for the overall solution quality of the procedure show that the average gap between the TTM solution and the TTM lower bound is less than 4 percent. The quality of TTM lower bound appears good for it deviates from the best lower bound computed using a column generation program by less than 0.08 percent.

The solution gaps for the problems with tight capacity, high setup costs, few items, or small number of periods, are larger. The characteristics that have little effects on the solution gap are the variability of capacity consumption per unit production, the variability of setup time across items, and the variability of demand. In phase 2, the solution gap for the problems with higher level of setup times is smaller, whereas the level of setup times have little effect on the solution gap in phase 3. This mixed effect is due to the different ways in which the capacity levels are generated for the test problems. It is noted at the end of the article that measuring an algorithm's ability to find feasible solutions for any given problem can be difficult.

In the next section, we will focus on the tabu search applications to production scheduling and planning problems.

2.2 Tabu Search Applications to Production Scheduling and Planning

Tabu search was first proposed by Glover (1986) as a general heuristic procedure for solving integer programming, and has been applied to many types of combinatorial optimization problems. A comprehensive description of the method and the characterization of its elements can be found in, e.g., Glover et al. (1993), and Hertz et al. (1992). Based on the extensive computational results in many applications and the late developments in the literature, Glover (1994) discussed the principal tabu search features that had contributed to the success of the method, and offered some directions for future improvement.

Impressive results have been achieved by many tabu search applications to production scheduling and planning problems. We present the review on these applications in the following two sub-sections: tabu search applications to production scheduling, and tabu search applications to production planning.

2.2.1 Tabu Search Applications to Production Scheduling

A flow shop consists of n jobs to be processed on m machines in the same order. The objective of the flow shop scheduling problem is to find a process schedule that minimize the completion time of the last job. An early application of tabu search to flow shop scheduling was developed by Widmer and Hertz (1989). The procedure first finds an initial starting sequence using insertion method, and then improves the sequence using tabu search technique. The type of move is defined as swaps of positions between two jobs. The simple rules used in the procedure include: the best solution produced by a non-tabu move is selected, tabu list length is set to 7, and it is terminated when a maximum number of iterations is reached. The procedure is effective in that, when compared with the best previous heuristic for 500 test problems, it returns better solutions for 58 percent of the problems, and the best solutions for 92 percent of the problems.

Reeves (1993) proposed an improved tabu search algorithm for solving the flow shop scheduling problem. The number of moves for evaluation at each iteration is reduced using a candidate list strategy. To limit the size of the neighborhood explored at each iteration, the consideration of moves is restricted to job set {J1, . . . , Jp} first, where p < n. Then, job set {Jp+1, . . ., J2p} is tried at the next iteration, and so on. When job Jn is reached, the process is restarted. The idea underlying the strategy is to find a balance between searching the solution space and making use of the information obtained during the course of the search. Computational efficiency was improved, and the results showed that, when implemented efficiently, tabu search performs better than simulated annealing on a wide range of test problems.

Tabu search was applied to a generalized flow shop problem, i.e., flexible-resource flow shop scheduling problem, by Daniels and Mazzola (1993). In their problem, process times of job on machines depend on the amount of resource allocated to the operations. The problem consists of the decisions in three levels, i.e., job sequencing, resource assignment, and operation start times. A nested strategy that decomposes the problem into three main components corresponding to the three decision levels was used. The nested tabu search procedure proves to be significantly superior when it is compared with other heuristics on a large number of test problems.

The job shop scheduling is another domain where tabu search has been applied. The problem has n jobs, and each job consists of a number of operations to be processed on some of m machines. Each job has its own precedence ordering of the job operations. The objective is to find a schedule that minimizes the makespan (the completion time of the last job).

One of the tabu search applications to the problem was reported in Dell'Amico and Trubian (1993). Their algorithm starts with finding a good initial solution by constructing from both ends of a partial schedule. The moves are defined as swaps that exchange two, or three arcs expressed on a directed network corresponding to two, or three operations. A tabu list with variable length is employed, and the minimum and maximum lengths are revised periodically using the random values from some number intervals. When the search fails to find better solutions for a number of iterations since the last best solution is found, a simple intensification sets the best solution to become the current solution, and continues the search starting from the best solution. Computational results indicate that the tabu search method is highly robust, and outperforms two simulated annealing algorithms.

Taillard (1994) presented an implementation of tabu search in both serial and parallel execution for solving the job shop problem. Based on two structural properties of a directed graph formulation with conjunctive arcs (for job operation precedence) and disjunctive arcs (for machine operation ordering), moves are defined as a reversal of a critical disjunctive arc, and the value of a move is the change on the longest path of the graph. The tabu list length is determined using random drawings from a range of values, and the length is changed for every *l* iterations, where *l* is the current tabu list length. In this implementation, the simplest aspiration criterion (accepting the move that produce a solution better than the best solution encountered so far regardless of its tabu status) is employed. The article reports that tabu search performs better than other heuristics for problems of size 10 x10, but shifting bottleneck heuristic produces better solutions for larger problems. It is also noted that tabu search has the advantage of being able to be extended easily to handle more complicated problems, such as the job shop scheduling problem with dynamic setup times.

A tabu search method was developed by Widmer (1991) to solve job shop scheduling problem with tooling constraints, which is a scheduling problem for flexible manufacturing. The problem deals with scheduling job shop with sequence dependent setup times for the operations due to tooling constraints.

The tabu search method uses insertion moves, and the evaluation of moves is done with respect to a weighted cost function composed of the makespan, the number of tool changes, the number of late jobs, and the number of machines failing to complete operations within a planning horizon. A single tabu list with fixed length is used in this basic tabu search application. The computational results show that larger problems make it difficult to find the best solutions with this basic approach. With this implementation, the adaptability of tabu search to solving very complex practical problems was demonstrated.

Tabu search application to single machine scheduling problem with delay penalties and setup costs are found in Laguna et al. (1991). In this problem, if job j is processed immediately after job i, a setup cost s_{ij} is incurred. The problem is a permutation problem with n! possible sequences, where n is the number of jobs. In this implementation, insertion and swap are the two types of moves. To limit the number of neighbors explored at each iteration, the moves involving jobs no more than d positions apart are considered, where d is $\lfloor n/2 \rfloor - 1$. Laguna and Glover (1993) integrated into the procedure a diversification component that discourages frequently executed moves to be selected during non-improving periods. The resulting procedure is robust in producing high quality solutions, and for many problem instances, it produces the solutions better than the best solutions found by some previous heuristics.

Woodruff and Spearman (1992) solved a general sequencing problem on a single machine that includes two classes of jobs with setup times, setup costs, holding costs, and deadlines, using tabu search approach. Insertion moves are used in the procedure. The procedure features a strategic oscillation that allows the infeasible solutions to be visited, a candidate list to reduce the number of solutions examined at each iteration, a diversification element using a parameter in the objective function, and a tabu list controlled by some hashing function values. This procedure is very effective for solving the problem in that it produces optimal solutions for 17 problems in a set of 20 test problems, and the average deviation from optimality is 3 percent.

2.2.2 Tabu Search Applications to Production Planning

In production planning area, tabu search applications to lot sizing problems are found in two articles. Kuik et al. (1993) compared linear programming, simulated annealing, and tabu search heuristics for solving a multi-level capacitated lot sizing problem for assembly production systems with one bottleneck. Setup times were not considered in the model. In the tabu search heuristic, moves are defined as the transitions from a given setup pattern y to another setup pattern y'. To evaluate a neighbor, an approximate algorithm that can quickly solve the problem with a fixed setup pattern is used. The best solution is chosen from a solution list of size 10 that contains the randomly generated non-tabu neighbors using the transition mechanism. The implementation contains a tabu list with length of 10, and terminates at 500 iterations. A procedure that uses the result from the LP-relaxation in the tabu search procedure was also described. The conclusions based on test experiments are that simulated annealing, and tabu search heuristic are preferable to the LP-based (rounding) heuristic. The combination of LP-based heuristic with simulated annealing, or tabu search enhances the performance.

As we have seen in the previous section, another application of tabu search to the lot sizing problem is found in Salomon et al. (1993). In this application, tabu search heuristic is used repeatedly to solve the single item CLSPs with the dual cost multipliers passed from a master problem. The tabu search heuristic consists of a move mechanism that generates the neighbor setup patterns by changing the value of one setup variable in a given setup pattern, and a special algorithm that can efficiently solve the sub-problem with a given setup pattern. This column generation approach with embedded tabu search heuristic turns out requiring more computational efforts than the dual based heuristic developed by Trigeiro et al. (1989). One of the reasons for this is said to be that the tabu search heuristic is not effective enough in producing high quality solutions to the single item CLSP.

2.3 Summary

We focused our literature review on research work on the CLSP with setup times, and on the tabu search applications to production scheduling and planning problems. It is found that for the CLSP with setup time alone, many mathematical programming-based techniques were employed by researchers. Among various mathematical programming-based techniques, Lagrangian relaxation is one approach that takes advantage of the special structures presented in the models for the CLSP with setup times, and successful results have been achieved by a number of Lagrangian relaxation-based heuristics. Ignoring setup carryovers in the CLSP modeling had been a gap, and a framework that explicitly modeling setup carryovers in the CLSP was proposed to address the gap. However, a good solution procedure for the CLSP with setup times and setup carryovers is still missing.

Tabu search has been successfully applied to solving many production scheduling and planning problems. The review shows that more tabu search applications, and better results for production scheduling problems can be found in the literature. Two tabu search applications to the CLSPs have been developed in line with one method for solving mixed integer programming model with tabu search, as mentioned in Glover (1990). In the two applications, the moves are defined by the setup pattern transitions (i.e., changing values of some binary variables). Then, with a given setup pattern, the procedures go about to solve the resulting problem with an optimal or approximate approach.

We develop a tabu search heuristic for the CLSP with setup times and setup carryovers, which, in addition to the move mechanisms that handle the setup carryovers, contains the move mechanisms that directly operate on production quantities, rather than on setup patterns. The design of our heuristic will be presented in the next two chapters.

Chapter 3
LOCAL SEARCH ELEMENTS

In the earlier chapters, we introduced the general formulation for the CLSP with setup times and setup carryovers. We also presented a brief description of our heuristic solution procedure based on the tabu search framework, along with a review of the literature pertinent to the CLSP with setup times and tabu search applications to the production scheduling and planning. In this chapter, we present the elements for implementing the local search in our heuristic. First, we present the solution representation, followed by the explanation of the enlarged solution space searched in our heuristic. Secondly, we describe in detail the different move mechanisms designed for carrying out the local search in our heuristic.

3.1 Solution Representation

In our heuristic procedure, a solution to the CLSP with setup times and setup carryovers is represented a little differently from that in the MILP model M1. We have the flexibility to represent the solutions differently, since our heuristic is not based on a mathematical programming approach. As we will see, the modified solution representation fits in better with the move operations implemented in our heuristic, and has a smaller number of variables.

Let P be the number of items to be produced, and let T be the number of time periods to be planned in a given problem instance. We adopt the following solution representation:

X_{it} -- production quantity for item i in period t;

α_t -- item type (expressed as a number from 1 to P) produced first in period t ($\alpha_t = 0$ if period is idle);

β_t -- item type produced last in period t ($\beta_t = 0$ if period t is idle);

γ_t -- machine state (expressed as the item type for which the machine is ready to produce) at the end of period t;

where i = 1,..., P, and t = 1,...,T.

In addition to the information explicitly reflected in the solution representation, other information can be derived. We can determine (i) the end inventory quantity for item i in period t by examining the demand quantity, X_{it}, and the end inventory quantity for item i in period t-1, (ii) the number of items produced in period t by counting nonzero X_{it}, and (iii) the number of setups done in period t by checking the number of the items produced in period t and the values of γ_{t-1}, α_t, β_t, and γ_t. These information will be used to evaluate the objective function for a given solution.

The solution representation defined above does not contain those variables such as S_{it}, V_{it}, O_{it}, ω_t, and δ_t found in the setup counting and partial sequencing constraints in model M1, but the relationships captured by these constraints can be observed in our heuristic implicitly with the structure of the solution representation and the operations in the move mechanisms on the variables X_{it}, α_t, β_t, and γ_t. The process of determining the number of setups in a given period described in the last paragraph is equivalent to observing the setup counting constraints (5) through (8) in model M1. The variables α_t, β_t, and γ_t express just what the partial sequencing constraints (9) through (16) in model M1 attempt to determine, i.e., the item produced first, the item produced last, and the machine state at the end of a period. In designing move operations, we will apply certain rules for changing the values of variables α_t, β_t, and γ_t to ensure that after a move operation, the partial sequences are correctly reflected by these variables .

3.2 Solution Space Explored

We let the solution space searched by our heuristic to be composed of all the solutions that satisfy the demand constraints in model M1. Since as mentioned earlier, the relationships captured by the setup counting and partial sequencing constraints in model M1 can be dealt with by the solution representation and the move operations, we need only to consider the remaining constraints in model M1 for relaxation. The remaining constraints are the demand constraints and capacity constraints. Because it is proved that finding a feasible solution with respect to constraints (2) through (4) in model M1 is NP-complete due to the inclusion of setup times in the capacity constraints, we choose to relax the capacity constraints, and to meet the demand constraints during the course of the search. Thus, the solution space explored by our procedure consists of all the solutions satisfying the demand constraints:

$$I_{i,t-1} + X_{it} - I_{it} = D_{it} \qquad i = 1,\ldots,P;\ t = 1,\ldots,T.$$

Finding a solution that satisfies the demand constraints alone is easy. One example is the lot-for-lot solution. Another solution can be the Wagner-Whitin solution. These solutions plus the initial values for α_t, β_t, and γ_t, $t = 1,\ldots,T$, can serve as the initial solutions for starting the search. The initial values for α_t, β_t, and γ_t, where $t = 1,\ldots,T$, can be obtained through identifying any items with nonzero production quantity in period t.

However, a solution that satisfies demands alone may be infeasible in terms of meeting capacity constraints. We deal with the infeasibility problem by the means of applying a penalty to the objective function for any capacity violation in a given solution, and the hope is that the infeasibility will be driven out eventually during the course of the search.

For a given solution, the penalty for the violation of the capacity constraints is defined as the product of a penalty rate and the sum of the resource requirement exceeding the capacity limit in each period. The penalty rate is a parameter in our procedure. To determine the sum of the resource requirement exceeding the capacity limit, the capacity constraint in each period needs to be examined. That is, for $t = 1,\ldots,T$, if

$$\sum_i b_i X_{it} + q N_t > C_t,$$

the amount of resource requirement exceeding the capacity limit in period t

$$e_t = \sum_i b_i X_{it} + q N_t - C_t;$$

otherwise, e_t is zero.

Thus, when the penalty is applied, the objective function becomes

$$f = Z + p\sum_t[\sum_i b_i X_{it} + q N_t - C_t]^+,$$

where $[x]^+ = \max\{0, x\}$, and p is the penalty rate. This is the objective function associated with any solution visited during the course of the search.

3.3 Move Mechanisms

We now present the design of move mechanisms used for local search. The moves operate on the variables X_{it}, α_t, β_t, and γ_t, and generate trial solutions that meet demand requirements. In general, if a solution in a solution space is viewed as a node in a graph representing the solution space, one desired property for the move mechanisms is that given any solution x, the search process is able to generate a path from x to an optimal solution. In designing our move mechanisms, trying to achieve this property is one guide line. Five move types are designed, and they are presented in the following five sub-sections.

3.3.1 Swap of Positions between the Item Produced First or Last and Some Other Item within a Given Period

This move type is designed to realize possible setup carryovers in case there are same items produced in any two adjacent periods. A move of this type (called swap move) changes a partial sequence within a period. The notion of partial sequencing refers to distinguishing the items produced in a given period, say period t, into the item produced first, α_t, the item produced last, β_t, and hence the items produced in-between, and identifying the machine state at the end of the period, γ_t. For a given solution, the position of any produced item in the partial sequence in any period is known. A swap move changes a partial sequence in a given period in such a way that an item i in the position of the item produced first or last and another item j in other position exchange their positions in the partial sequence, so that after the move, item j takes the position of the item produced first or last, and item i takes the position originally taken by item j. The moves of this type can be performed for any period in which at least two items are produced.

To implement a swap move in period t, in addition to changing the value of α_t, or β_t, the value of γ_{t-1}, or γ_t needs to be changed too under certain conditions. We can determine whether there is a setup carryover between any two adjacent periods by looking at the information on a partial sequence. Condition ($\beta_t = \gamma_t$ and $\gamma_t = \alpha_{t+1}$) means a setup carryover between period t and t+1 without an end-of-period setup in period t, condition ($\beta_t \neq \gamma_t$ and $\gamma_t = \alpha_{t+1}$) means a setup carryover between period t and t+1 with an end-of-period setup in period t, and condition ($\beta_t = \gamma_t$ and $\gamma_t \neq \alpha_{t+1}$) indicates that there is no setup carryover between period t and t+1. But condition ($\beta_t \neq \gamma_t$ and $\gamma_t \neq \alpha_{t+1}$) means an unnecessary setup between the period t and t+1. Thus, in addition to change the value of α_t, or β_t, we change the value of γ_{t-1}, or γ_t too under the condition ($\beta_{t-1} \neq \gamma_{t-1}$ and $\gamma_{t-1} = \alpha_t$), or condition ($\beta_t = \gamma_t$ and $\gamma_t \neq \alpha_{t+1}$), respectively, so that the undesired situation ($\beta_{t-1} \neq \gamma_{t-1}$ and $\gamma_{t-1} \neq \alpha_t$), or ($\beta_t \neq \gamma_t$ and $\gamma_t \neq \alpha_{t+1}$) does not take place at any time during the course of the search.

Now suppose that item i produced in-between is considered for a swap of position with the item produced first or last in period t. For the swap involving the item produced first, the following rules apply.

Swap Between an Item Produced In-between and the Item Produced First

```
if $\beta_{t-1} = \gamma_{t-1}$ then
  if $\gamma_{t-1} \neq \alpha_t$ then
    $\alpha_t \leftarrow i$
  else
    $\alpha_t \leftarrow i$
     if penalty increase due to adding one setup in period t-1 < penalty
       increase due to adding one setup in period t then
       $\gamma_{t-1} \leftarrow i$
    end if
  end if
else                    ($\beta_{t-1} \neq \gamma_{t-1}$)
  $\alpha_t \leftarrow i$
  $\gamma_{t-1} \leftarrow i$
end if
```

For the swap involving the item produced last, the following rules apply.

Swap Between an Item Produced In-between and the Item Produced Last

```
if $\beta_t = \gamma_t$ then
  if $\gamma_t \neq \alpha_{t+1}$ then
    $\beta_t \leftarrow i$
    $\gamma_t \leftarrow i$
  else
    $\beta_t \leftarrow i$
    if penalty increase due to adding one setup in period t+1 < penalty
      increase due to adding one setup in period t then
      $\gamma_t \leftarrow i$
    end if
  end if
else                    ($\beta_t \neq \gamma_t$)
  $\beta_t \leftarrow i$
end if
```

If a swap of positions is between the item produced first and the item produced last in period t, the conditions about α_t, β_t, γ_{t-1}, and γ_t at the both ends of period t need to be examined, and a combination of the rules for the swap of positions with the item produced first and the rule for the swap of positions with the item produced last described earlier must be used. The following rules apply for the swaps between the item produced first and the item produced last.

Swap between the Item Produced First and the Item Produced Last

```
if $\beta_{t-1} = \gamma_{t-1}$ and $\beta_t = \gamma_t$ then
    if $\gamma_{t-1} \neq \alpha_t$ and $\gamma_t \neq \alpha_{t+1}$ then
        $i \leftarrow \alpha_t$
        $\alpha_t \leftarrow \beta_t$
        $\beta_t \leftarrow i$
        $\gamma_t \leftarrow i$
    else if $\gamma_{t-1} \neq \alpha_t$ and $\gamma_t = \alpha_{t+1}$ then
        $i \leftarrow \alpha_t$
        $\alpha_t \leftarrow \beta_t$
        $\beta_t \leftarrow i$
        if penalty increase due to adding one setup in period t+1 < penalty
            increase due to adding one setup in period t then
            $\gamma_t \leftarrow i$
        end if
    else if $\gamma_{t-1} = \alpha_t$ and $\gamma_t \neq \alpha_{t+1}$ then
        $i \leftarrow \alpha_t$
        $\alpha_t \leftarrow \beta_t$
        $\beta_t \leftarrow i$
        $\gamma_t \leftarrow i$
        if penalty increase due to adding one setup in period t-1 < penalty
            increase due to adding one setup in period t then
            $\gamma_{t-1} \leftarrow \alpha_t$
        end if
    else                    ($\gamma_{t-1} = \alpha_t$ and $\gamma_t = \alpha_{t+1}$)
        $i \leftarrow \alpha_t$
        $\alpha_t \leftarrow \beta_t$
        $\beta_t \leftarrow i$
        pen-inc $\leftarrow$ penalty increase due to adding two setups in period t
        **if** penalty increase due to adding one setup in period t-1 and one
          setup in period t < pen-inc **then**
          $\gamma_{t-1} \leftarrow \alpha_t$
          pen-inc $\leftarrow$ penalty increase due to adding one setup in period t-1
                      and one setup in period t
        **end if**
        **if** penalty increase due to adding one setup in period t and one
          setup in period t+1 < pen-inc **then**
          $\gamma_t \leftarrow i$
          pen-inc $\leftarrow$ penalty increase due to adding one setup in period t
                      and one setup in period t+1
        **end if**
        **if** penalty increase due to adding one setup in period t-1 and one
          setup in period t+1 < pen-inc **then**
          $\gamma_{t-1} \leftarrow \alpha_t$
          $\gamma_t \leftarrow i$
        **end if**
      **end if**
    **else if** $\beta_{t-1} = \gamma_{t-1}$ and $\beta_t \neq \gamma_t$ **then**
      **if** $\gamma_{t-1} \neq \alpha_t$ **then**
        $i \leftarrow \alpha_t$
        $\alpha_t \leftarrow \beta_t$
        $\beta_t \leftarrow i$
      **else**
```

```
            $i \leftarrow \alpha_t$
            $\alpha_t \leftarrow \beta_t$
            $\beta_t \leftarrow i$
            if penalty increase due to adding one setup in period t-1 < penalty
              increase due to adding one setup in period t then
              $\gamma_{t-1} \leftarrow \alpha_t$
            end if
          end if
        else if $\beta_{t-1} \neq \gamma_{t-1}$ and $\beta_t = \gamma_t$ then
          if $\gamma_t \neq \alpha_{t+1}$ then
            $i \leftarrow \alpha_t$
            $\alpha_t \leftarrow \beta_t$
            $\gamma_{t-1} \leftarrow \beta_t$
            $\beta_t \leftarrow i$
            $\gamma_t \leftarrow i$
          else
            $i \leftarrow \alpha_t$
            $\alpha_t \leftarrow \beta_t$
            $\gamma_{t-1} \leftarrow \beta_t$
            $\beta_t \leftarrow i$
            if penalty increase due to adding one setup in period t+1 < penalty
              increase due to adding one setup in period t then
              $\gamma_t \leftarrow i$
            end if
          end if
        else                    ($\beta_{t-1} \neq \gamma_{t-1}$ and $\beta_t \neq \gamma_t$)
          $i \leftarrow \alpha_t$
```

```
    $\alpha_t \leftarrow \beta_t$
    $\gamma_{t-1} \leftarrow \beta_t$
    $\beta_t \leftarrow i$
end if
```

To accommodate the situation of idle period in period t-1, and/or in period t+1, we apply some measures to the swap move procedures described above. In all the above procedures, γ_{t-2} is used in the places of β_{t-1} in case period t-1 is idle, and γ_{t+1} is used in the places of α_{t+1} in case period t+1 is idle.

The cost items in the objective function affected by a swap move can be the setup costs and penalties. The objective function evaluation for a trial solution produced by a swap move can be performed by finding out the relevant changes in setup costs and penalties in the two adjacent periods, i. e., period t-1 and t, or period t and t+1, for a swap between the item produced in between and the item produced first, or last, respectively. And it can be done by finding out the relevant changes in the three adjacent periods, i. e., period t-1, t, and t+1, for a swap between the item produced first and the item produced last. The actual changes of the cost items in the objective function depend on the conditions about α_t, β_t, γ_{t-1} and γ_t before and after a move. The computations in evaluating a move are rather simple, but the detailed condition checkings are required to determine the specific conditions before and after the move.

Instead of going through the detailed condition checkings for move evaluation under different situations, we describe an example below for illustration.

Suppose that a swap is between the item produced in-between and the item produced first, condition ($\beta_{t-1} = \gamma_{t-1}$ and $\gamma_{t-1} \neq \alpha_t$) holds before the move, and it becomes ($\beta_{t-1} = \gamma_{t-1}$ and $\gamma_{t-1} = \alpha_t$) after the move. In this case, setup cost for one setup is saved, and penalty is reduced if there is any in period t before the move. The objective function value of the trial solution given by this move is

$$f_{trial} = f - A - B_t,$$

where f is the objective function value before the move, A is the setup cost per setup, and B_t is the possible penalty decrease in period t. Here, the value of B_t depends on the situation of capacity constraint feasibility in period t before the move. Let p be the penalty rate, let q be the setup time per setup, and let e_t be the amount of resource requirement exceeding the capacity limit in period t before the move. The value of B_t can be determined according to the following formula·

$$B_t = \begin{cases} 0 & \text{if } e_t = 0, \\ p.e_t & \text{if } 0 < e_t < q, \\ p.q & \text{if } q \leq e_t. \end{cases}$$

Associated with a swap between an item produced in-between and the item produced first or last, the attributes recorded in the tabu list are the ordinal number of the period, t, and the item produced first or last, α_t or β_t. Associated with a swap between the item produced first and the item produced last, the move recorded in the tabu list are the ordinal number of the period, t, and the item produced first, α_t. In our current implementation, after the current iteration at which a swap move involving α_t or β_t in period t is executed, any swap move involving the original α_t or β_t in period t is considered a tabu move for L iterations, where L is the tabu list length. This tabu restriction is more strict than simply forbidding a reverse move, and hence cycling will be less likely to happen.

Assume all the P items are produced in each of the T periods. Then, there will be P-2 swap moves between the item produced in-between and the item produced first, P-2 swap moves between the item produced in-between and the item produced last is P-2, and 1 swap move between the item produced first and the item produced last that can be performed in each period. Thus the size of a solution's sub-neighborhood created by this move type is at most $((P-2)+(P-2)+1)T = (2P-3)T = O(PT)$.

3.3.2 Setup Move to the End of the Previous Period

A move of this type moves the first setup in period t to the end of period t-1, if the machine state in the end of period t-1 is different from the first setup in period t, i.e., $\gamma_{t-1} \neq \alpha_t$ in case at least one item is produced in period t, or $\gamma_{t-1} \neq \gamma_t$ in case period t is idle. This move type helps to realize the end-of-period setup, which is desired when the items produced in period t are different from that in period t-1, and setting up the machine for item α_t at the end of period t-1, in stead of at the beginning of period t, reduces or eliminates the capacity violation.

Suppose period t is under consideration. If period t is not idle, the implementation of a move of this type is as follows:

Setup Move to the End of the Previous Period

```
if $\gamma_{t-1} \neq \alpha_t$ then
  $\gamma_{t-1} \leftarrow \alpha_t$
end if
```

If period t is idle, γ_t is used in the places of α_t in the above procedure. Moving setup in an idle period to the end of the previous period is to help carry a setup over a series of idle periods. The presence of a series of consecutive idle periods in solutions may not be very possible for practical instances. We nonetheless provide measures for handling the situation.

The cost items in the objective function to be affected by a move of this type can be the penalties. The objective function value of a trial solution reached by a move of this type can be efficiently evaluated by considering only the changes in penalties in the periods t-1 and t. The objective function value of a trial solution generated by a move of this type is

$$f_{trial} = f + B_{t-1} - B_t,$$

where f is the objective function value before the move, B_{t-1} is the possible penalty increase in period t-1, and B_t is the possible penalty decrease in period t. The value of B_t can be determined in the same way as that described earlier for swap move evaluation. To determine the value of B_{t-1}, let p be the penalty rate, let q be the setup time per setup, and let u_{t-1} be the amount of slack capacity in period t-1 before the move. The value of B_{t-1} can be determined according to the following formula:

$$B_{t-1} = \begin{cases} p.q & \text{if } u_{t-1} = 0, \\ p.(q - u_{t-1}) & \text{if } 0 < u_{t-1} < q, \\ 0 & \text{if } q \le u_{t-1}. \end{cases}$$

For a move of this type in period t, the associated move attributes recorded in the tabu list are the ordinal number of the previous period, t-1, and the item type involved in the move, α_t. The reverse move of a move of this type is a move of type 3 (setup move to the beginning of the next period) to be described below involving α_t in period t-1. A reverse move is considered tabu for L iterations after the current iterations, where L is the tabu list length.

At most one move of this type is possible in any period from period 2 to period T, and hence the size of the sub-neighborhood created by this move type is at most T-1 = O(T).

3.3.3 Setup Move to the Beginning of the Next Period

A move of this type moves the end-of-period setup in period t to the beginning of period t+1, if the machine state at the end of period t is different from the item produced last in period t, i.e., $\beta_t \neq \gamma_t$, in case period t is not idle, or if it is different from the machine state at the end of period t-1, i.e., $\gamma_{t-1} \neq \gamma_t$, in case period t is idle. This move type helps to remove undesired end-of-period setup in period t, when setting up the machine for item α_{t+1} at the end of period t incurs more capacity violation than setting up the machine for item α_{t+1} at the beginning of period t+1.

Suppose a move of this type is performed in period t. In case period t is not idle, a move of this type is a simple operation as shown below:

```
*Setup Move to the Beginning of the Next Period*
**if** $\beta_t \neq \gamma_t$ **then**
    $\gamma_t \leftarrow \beta_t$
**end if**
```

If period t is idle, β_t is replaced with γ_{t-1} everywhere in the move operation. Again, the consideration given to the case of idle period is to facilitate the realization of setup carryover over a series of consecutive idle periods, though most often a series of consecutive idle periods is not very likely to happen in practical instances.

The cost items in the objective function affected by a move of this type can be the penalties in period t and t+1. To evaluate a trial solution reached by a move of this type, we can perform the computations efficiently by considering only the changes in penalty in the two adjacent periods. The objective function value of a trial solution from a move of this type is

$$f_{trial} = f - B_t + B_{t+1},$$

where f is the objective function value before the move, B_t is the possible penalty decrease in period t, and B_{t+1} is the possible penalty increase in period t+1. The values of B_t and B_{t+1} can be determined using the same formulae as that described earlier for move type 1 and 2.

For a move of this type in period t, the associated move attributes recorded in the tabu list are the ordinal number of the next period, t+1, and the machine state at the end of period t, γ_t. The reverse of a move of this type is a move of type 2. A move of type 2 involving γ_t in period t+1 is considered a tabu move for L iterations after the current iterations, where L is the tabu list length.

As at most one move of this type can be performed in any period from period 1 to period T-1, the size of the sub-neighborhood associated with this move type is at most T-1 = O(T).

3.3.4 Lot Shift to an Earlier Period

This move type is designed to shift certain amount of production for an item in period t to an earlier period s, where $1 \le s \le t-1$, and a move of this type can result in a solution that yields lower objective function value due to a better trade-off between the inventory costs, setup costs, fixed production charges, and penalties for capacity violation. When a part of a batch is shifted to an earlier period, a move of this type can help to eliminate or reduce the capacity violation. When a whole batch is shifted to an earlier period, in addition to being able to eliminate or reduce the capacity violation, it can also possibly reduce the setup costs and fixed production charges. But, in any case, the inventory costs will rise.

Suppose item i is produced in period t, and is under consideration for being shifted from period t to period s, where $1 \le s \le t-1$. Depending on whether item i is produced in period s or not, a lot is shifted either without a setup or with a setup. To determine the amount of available capacity in period s for taking in production of item i, the situation in period s has to be examined. In case item i is originally produced in period s, the amount of available capacity for taking in production equals the amount of slack capacity in period s since no new setup needs to be added when a lot is shifted into period s. In case item i is not originally produced in period s, the amount of available capacity in period s depends on the conditions in period s. One example can help to illustrate this point. Suppose only one item is produced in period s originally, and item i is the same as the machine state at the end of period s-1. If we make item i to be produced first in period s, the setup for item i will not be needed, and hence the amount of available capacity in period s will be equal to the amount of slack capacity in period s.

Some rules are designed for determining under what circumstance a lot shift should be performed, and in what size a lot should be shifted. We describe these rules below separately for the case that item i is produced originally in period s, and for the case that item i is not produced originally in period s.

Let X_{it} and X_{is} be the production quantities for item i in period t and period s respectively, let b_i be the capacity consumption coefficient of item i, let u_s be the amount of slack capacity in period s, and let e_t be the amount of resource requirement exceeding capacity limit in period t. For the case that item i is originally produced in period s, we have the following rules:

Lot Shift to an Earlier Period when Item i Is Produced in that Earlier Period

```
1 if $e_t > 0$ and $u_s > 0$ then
    if $u_s \geq b_iX_{it}$ then
```

```
          $X_{is} \leftarrow X_{is} + X_{it}$
          $X_{it} \leftarrow 0$          (*whole batch is shifted*)
        else
          if $u_s \geq e_t$ then
            $X_{is} \leftarrow X_{is} + e_t/b_i$
            $X_{it} \leftarrow X_{it} - e_t/b_i$     (*part of a batch is shifted*)
          else
            $X_{is} \leftarrow X_{is} + u_s/b_i$
            $X_{it} \leftarrow X_{it} - u_s/b_i$     (*part of a batch is shifted*)
          end if
        end if
      end if
    2 if $e_t > 0$ and $u_s = 0$ then
        $X_{is} \leftarrow X_{is} + X_{it}$
        $X_{it} \leftarrow 0$          (*whole batch is shifted*)
      end if
    3 if $e_t = 0$ and $u_s > 0$ then
        if $u_s \geq b_i X_{it}$ then
          $X_{is} \leftarrow X_{is} + X_{it}$
          $X_{it} \leftarrow 0$          (*whole batch is shifted*)
        else
          $X_{is} \leftarrow X_{is} + u_s/b_i$
          $X_{it} \leftarrow X_{it} - u_s/b_i$     (*part of a batch is shifted*)
        end if
      end if
```

In the above procedure, a certain amount of production quantity is shifted in step 1 where the capacity in period t is exceeded, and a slack capacity exists in period s. A certain amount of production quantity is shifted too in step 3 where both period t and period s have slack capacity. Step 2 is included because we want to cover as more trial solutions which might be favorable in certain cases as possible. The moves carried out in step 2 can produce favorable trial solutions when the penalty rate (which is varied in the main procedure as described in the next chapter) and inventory holding costs for item i are not high so that the sum of the possible penalty increase and inventory cost increase is not as large as the setup cost saving due to a lot merge.

When item i is not produced in period s, before a lot is moved into period s, condition checkings in period s need to be done in order to determine the amount of available capacity in period s. Specifically, if two or more items are produced in period s, as a rule, the new item i is made to be produced in-between so that a new setup needs to be added into period s, and hence the amount of available capacity in period s equals the amount of slack capacity minus one setup time. However, if only one item or no item is produced in period s, the conditions of the production's partial sequence in period s and the machine state at the end of period s-1 and period s need to be examined in order to determine how many setups need to be added in period s. The example mentioned earlier illustrates one of the possible situations. Once the number of added setups is known, the amount of available capacity in period s can be determined.

Assume the condition checkings are done, and the available capacity in period s for item i is known. Let X_{it} and X_{is} be the production quantity for item i in period t and period s respectively, let b_i be the capacity consumption coefficient of item i, let w_s be the amount of available capacity in period s for item i, and let e_t be the amount of resource requirement exceeding capacity limit in period t. For the case that item i is not originally produced in period s, we have the following rules:

Lot Shift to an Earlier Period when Item i Is Not Produced in that Earlier Period

```
1 if $e_t > 0$ and $w_s > 0$ then
    if $w_s \geq b_i X_{it}$ then
      $X_{is} \leftarrow X_{is} + X_{it}$,
      $X_{it} \leftarrow 0$          (whole batch is shifted)
    else
      if $w_s \geq e_t$ then
        $X_{is} \leftarrow X_{is} + e_t/b_i$,
        $X_{it} \leftarrow X_{it} - e_t/b_i$     (part of a batch is shifted)
      else
        $X_{is} \leftarrow X_{is} + w_s/b_i$,
        $X_{it} \leftarrow X_{it} - w_s/b_i$     (part of a batch is shifted)
      end if
    end if
  end if
2 if $e_t = 0$ and $w_s > 0$ then
    if $w_s \geq b_i X_{it}$ then
      $X_{is} \leftarrow X_{is} + X_{it}$,
      $X_{it} \leftarrow 0$          (whole batch is shifted)
    else
```

```
        $X_{is} \leftarrow X_{is} + w_s/b_i$,
        $X_{it} \leftarrow X_{it} - w_s/b_i$        *(part of a batch is shifted)*
    **end if**
**end if**
```

Here, the above procedure is similar to that for the case item i is originally produced in period s except that the available capacity is used in stead of using the slack capacity. However, it does not consider shifting a lot when the capacity limits in both period t and s are exceeded.

Note that in the above procedures, in case a whole batch is shifted out of period t, and the item being shifted is the item originally produced first or last in period t, the possible changes in the values of the variables α_t, β_t, and γ_t for the partial sequence need to be made in period t, and the corresponding possible cost changes need to be reflected in the evaluation of a move.

This move type can have the effect of reducing the setup costs, fixed production charges, and penalties when a whole batch is moved, and the effect of reducing penalties when a part of a batch is moved. But the inventory costs are certain to rise. The change of objective function value for a trial solution depends on the specific conditions before and after a move, and can be determined after some detailed condition checkings are done. For example, assume that before the move, item i is neither an item produced first nor an item produced last in period t, the amount of resource requirement exceeding capacity limit in period t e_t is greater than 0, item i is produced in period s, two or more items are produced in period s, and the amount of slack capacity in period s u_s is greater than b_iX_{it}. When a whole batch of item i is shifted from period t to period s, the objective function value of the trial solution reached by this move becomes

$$f_{trial} = f - A - F_t - B_t + K,$$

where A is the setup cost for one setup, F_t is the fixed production charge for one item produced in period t, B_t is the penalty decrease in period t, and K is the inventory cost increase due to the move. Here, B_t can be determined according to the following formula:

$$B_t = \begin{cases} p.e_t & \text{if } e_t < q + b_i X_{it}, \\ p.(q + b_i X_{it}) & \text{if } e_t \geq q + b_i X_{it}, \end{cases}$$

where p is the penalty rate, and q is the setup time for one setup. And K is equal to $\sum_{r=s}^{t-1} H_{ir} X_{it}$, where H_{ir} is the inventory holding cost for item i in period r.

This example shows that the evaluation of a trial solution requires simple calculations, but very detailed condition checkings. With the above example given for the illustration of move evaluation, we will not cover the detailed condition checking process for many other situations, as we did earlier in presenting the swap moves.

If a move of this type is executed, the move attributes recorded in the tabu list are the item shifted, i, the ordinal number of the target period of the lot shift, s. The reverse of a move of this type is a move of type 5 (lot shift to a later period) to be described below. In the current implementation, a move of type 5 shifting item i out of period s is considered tabu for L iterations after the current iteration, where L is the tabu list length.

Assume all the P items are produced in each of the T period. Since the lot for any item in period t, t = 2,...,T, can be shifted to at most t-1 previous periods, the sub-neighborhood corresponding to this move type has the size of at most $(1+2+\ldots+(T-1))P=\frac{1}{2}T(T-1)P=O(PT^2)$.

3.3.5 Lot Shift to a Later Period

This move type is designed to shift a certain amount of production for an item in period t to a later period s, where $t+1 \le s \le T'$, where T' is determined in a condition checking operation described below. A move of this type can be performed if there is an end inventory for an item in period t, and the later period s has available capacity for taking in the production of the item. This move type has the effect of reducing the unnecessary inventory built up in the earlier periods, which can happen in an initial solution, or as the result of executing some moves of lot shift to an earlier period in the previous iterations.

Suppose item i is produced in period t, and is being considered for shift from period t to a later period. In order to guarantee the demand requirements are met after a lot shift, we need to determine which late period can be a candidate target period for taking in a lot, i.e., period s, and the maximum production quantity allowed to be shift from period t to period s. It is found that any later period before and up to the first period with zero end inventory level for item i can be a candidate target period s for taking in certain amount of production of item i. The maximum production quantity of item i allowed to be shifted from period t to any target period s is determined by $Q = \min \{I_{it}, I_{i,t+1}, \ldots, I_{i,s-1}\}$ so that the demands for item i in periods between period t and period s-1 will be still satisfied after the move.

Lot shift to a later period can also be accompanied with or without a new setup in period s, depending on the conditions in period s. If item i is originally produced in period s, the amount of available capacity equals the amount of slack capacity in period s. If item i is not originally produced in period s, the amount of available capacity usually equals the amount of slack capacity minus one setup time accompanying the introduction of production of item i in period s. To determine the number of setups to be added in period s due to a lot move, conditions in period s need to be examined. The discussion presented earlier for move type 4 for determining the number of added setup applies here too.

For this move type, after the amount of available capacity period s is determined, the same rules are used for both the cases that item i is originally produced in period s, and item i is not originally produced in period s. Let X_{it}, and X_{is} be the production quantity for item i in period t, and period s respectively, let b_i be the capacity consumption coefficient for item i, let u_s be the amount of available capacity in period s, and let Q be the maximum allowed production quantity of item i to be shifted from period t to period s. Assume also u_s is greater than zero. The following rules apply to this move type:

Lot Shift to a Late Period

```
1 if $u_s/b_i \geq Q$ then
      if $X_{it} \leq Q$ then
         $X_{is} \leftarrow X_{is} + X_{it}$
         $X_{it} \leftarrow 0$          (whole batch is shifted)
      else
         $X_{is} \leftarrow X_{is} + Q$
         $X_{it} \leftarrow X_{it} - Q$      (part of a batch is shifted)
      end if
  else
      if $b_iX_{it} \leq u_s$ then
         $X_{is} \leftarrow X_{is} + X_{it}$
         $X_{it} \leftarrow 0$          (whole batch is shifted)
      else
         $X_{is} \leftarrow X_{is} + u_s/b_i$
         $X_{it} \leftarrow X_{it} - u_s/b_i$      (part of a batch is shifted)
      end if
  end if
```

As noted earlier for move type 4, when a whole batch of item i is shifted out of period t, and item i is originally produced first or last in period t, the possible changes in the values of αt, βt, and γt need to be implemented, and the corresponding changes in costs need to be reflected in the objective function evaluation.

In addition to the main effect of reducing inventory costs, a move of this type can also affect other cost items in the objective function. Similar to that discussed earlier for move type 4, the calculations in move evaluation are simple, but extensive condition checkings are required in order to determine which cost items are affected.

If a move of this type is executed, the associated attributes memorized in the tabu list are the item shifted, i, and the ordinal number of the target period of the lot shift, s. The reverse of a move of this type is a move of type 4. In the current implementation, we consider a move of type 4 shifting item i out of period s a tabu move for L iterations after the current iteration, where L is the tabu list length.

Assume again all the P items are produced in each of the T period. Since the lot for an item in period t, t = 1,...,T-1, can be shifted to at most T-t later periods, the sub-neighborhood corresponding to this move type has the size of at most $((T-1) + (T-2) \ldots +1)P= \frac{1}{2}T(T-1)P=O(PT^2)$.

3.3.6 The Routine Procedures for the Move Mechanisms

To implement the move mechanisms presented above , five routine procedures are created for the five respective move types. Each routine procedure generates and evaluates all the possible moves of the corresponding type. The routines are named as SWAP, SPMVAHD, SPMVBAK, LTMVFWD, and LTMVBWD for the move types of position swap, setup move to the end of the previous period, setup move to the beginning of the next period, lot shift to an earlier period, and lot shift to a later period, respectively. Either all or a part of the five routines will be invoked one by one at each iteration in the main procedure. Thus, the neighborhood of a solution x is the set of all the solutions generated by the moves of the types invoked at each iteration. After the execution of the move procedures at each iteration, a best trial solution in the neighborhood is identified.

3.4 Summary

This chapter presents the design of local search elements used in our heuristic. These elements include the solution representation, the solution space explored, and the move mechanisms. Among the five move types designed, three of them help to achieve good partial sequencing, i.e., realize setup carryover across periods, and two of them help to achieve good lot sizing. Together, the five procedures that perform the corresponding move operations fulfill the local search function in our tabu search heuristic.

Chapter 4
GLOBAL SEARCH ELEMENTS AND A BOUNDING PROCEDURE

Chapter 3 presented the local search elements designed for our tabu search heuristic for solving the CLSP with setup times and setup carryovers. A tabu search heuristic contains as well the global search elements for searching beyond local optima. In this chapter, we present the global elements used in our heuristic, and a bounding procedure. First, we discuss our approach to invoking the five routine procedures performing the move operations, followed by a description of a data structure for implementing the tabu list, and the rules for varying the tabu list length. Secondly, we describe the intensification, and diversification approaches adopted in our heuristic. Some rules for changing the penalty rate during the course of the search are also described. Thirdly, we present our heuristic's main procedure in which all the search elements (local and global) are integrated. Lastly, we present a procedure for finding the lower bounds on the optimal value of our problem, which will serve as one of the relative measures for evaluating the solution quality of our heuristic.

4.1 Neighborhood Structure

How the five move types are used in our heuristic is the issue we addressed first. If all the five procedures performing the five move types are invoked at each iteration, the neighborhood of a solution x will be the union of the five sub-neighborhoods generated by the moves of five types.

One problem with the swap moves was found after some preliminary testings during the algorithm development phase. In case many items, but not the same items are produced in any two adjacent periods, the swap moves can produce many neighbor solutions that are neither better nor worse than the current solution because many items can be made to be produced first or last, and the objective function value remains the same. If no improving solution can be found by the moves of other types, the next solution at each iteration will be always chosen from the solutions produced by swap moves because of the existence of a large number of non-tabu swap moves even with the use of a long tabu list. The search process, therefore, stagnate at this stage. The problem happens often when the problem instances have a large size in terms of the number of items.

This stagnation problem is resolved by restricting the use of swap moves during the course of the search. Specifically, the routine that performs the swap moves is called roughly one third of the times. The approach is implemented at each iteration in the following fashion:

```
1 call SPMVAHD
2 call SPMVBAK
3 call LTMVFWD
4 call LTMVBWD
5 **if** (k mod P) $\leq$ (P / 3) **then**
      call SWAP
```

```
end if
```

where k is the current value of an iteration counter, and P is the number of items. Hence, the neighborhood searched changes alternatively during the course of the search. It is the one generated by all the moves of the five types in one third of the time, or by the moves of four types in two third of the time. This strategy proves working well since it eliminates the stagnation problem for those instances with a large number of items, and achieves the same quality solutions with less computational effort for those instances that did not experience the stagnation problem before the strategy is used.

In our current implementation, a neighborhood is searched exhaustively at each iteration. The best non-tabu solution among all the neighbor solutions is chosen as the new current solution. The non-tabu requirement on the neighbor solutions is overruled by a basic aspiration criterion that accepts a solution in spite of its tabu status if it is better than the best solution encountered so far.

4.2 Implementation of the Tabu List

In this section, we describe the data structure used in our algorithm for implementing the tabu list, and the rules for varying the tabu list length.

4.2.1 The Linked List with Multiple Links for the Tabu List Operations

As described in Chapter 3, associated with a move of any type, there is a set of attributes to be memorized in the tabu list. In order to implement the tabu list operations, a data structure for the tabu list needs to be designed. In our algorithm, we use one single list to record the attribute sets of L recently executed moves of different types.

The single tabu list memorize the attributes of moves of multiple types is represented as a *linked list with multiple links*. In addition to the key of an element which represents the set of move attributes, there are two pointers attached to each element on the list. One pointer points to the next element on the list regardless of its move type, and another pointer points to the next element of the same move type. The link consisting of the first type pointers is for adding and deleting operations performed on the linked list, and the links of the second type pointers are for checking the tabu status. The linked list operates in a FIFO manner. At each iteration, according to the rule for varying the tabu list length (to be described below), a new set of the attributes is inserted at the tail of the list, and one set, or two sets of the attributes are removed from the head of the list, or no removal takes place. The adding and deleting operations are typical linked queue operations with the update done on the two sets of the head and tail pointers, one for the whole list link, and another for one move type link. In order to determine the tabu status of a trial move, we need to check the elements corresponding to the moves of same type on the list. This tabu status checking can be done quite efficiently for we only need to traverse one link for the moves of same type on the list. The tabu status checking is the searching operation on the linked list along the link for a specific move type. Figure 3 is an illustration of the linked list with multiple links.

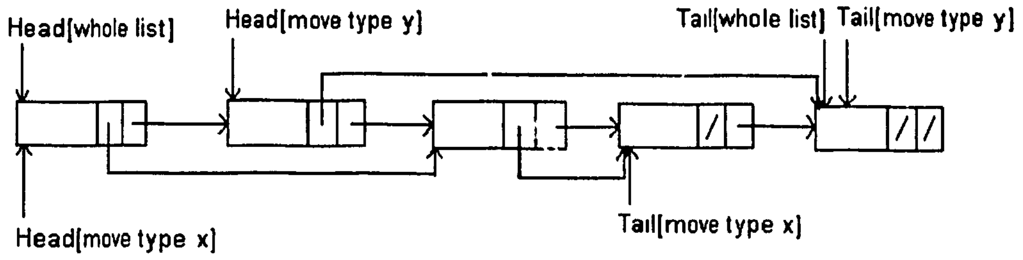

Figure 3. Illustration of the Linked List with Multiple Links

4.2.2 The Tabu List with a Variable Length

Another design issue is the list length. Using *tabu lists with a fixed length* is a strategy easy to implement, but may not be as robust as using *tabu lists with a variable length* (Glover 1994). The experiments on our heuristic using fixed length tabu list also show that finding an appropriate value for the list length that works well for most of test problems is hardly possible. We then choose to use variable length tabu list.

The variable length tabu list is usually implemented with two parameters: a minimum length L_{min}, and a maximum length L_{max}. The actual length within the interval of $[L_{min}, L_{max}]$ during the course of the search is determined by using either the random or deterministic rules. One example of the random rules used by Taillard (1994) is that the list length is a value of a random variable uniformly distributed on the interval [0.8L, 1.2L], where L is determined based on the problem size. In our current implementation, since no attempt is made to introduce a probabilistic element, we use a deterministic rule for the variable list length , which is a simplified version of the work due to Dell'Amico and Trubian (1993). Let f_{cur} and f_{pre} be the objective function values for the current solution and the last solution respectively, let L_{min} and L_{max} be the minimum and maximum tabu list length respectively, and let L_{cur} be the current tabu list length. The rules for varying the list length are given as follows:

```
**if** $f_{cur} < f_{pre}$ **then**
  **if** $L_{cur} > L_{min}$ **then**
    $L_{cur} \leftarrow L_{cur} - 1$
  **end if**
**else**
  **if** $L_{cur} < L_{max}$ **then**
    $L_{cur} \leftarrow L_{cur} + 1$
  **end if**
**end if**
```

With the knowledge gained from testing on a number of problem instances, we set the minimum length L_{min} to (1-3/4)(P+T), and the maximum length L_{max} to (1+6/4)(P+T), where P and T are the number of items and the number of periods respectively. A tabu list with longer length not only has the effect of preventing cycling, but it also has the effect of diversifying the search process. The minimum and maximum lengths are set to be proportional to the problem size, so that the diversifying effect provided by longer tabu lists is exploited in larger problem instances.

4.3 Intensification and Diversification Approaches

Intensification and diversification are important strategies, especially for large problems, in that they can enhance a procedure's ability to achieve high quality solutions with less computational effort, or better solutions for larger computation times. In our case, some preliminary testing results with our heuristic using short term memory alone indicated that improvements were needed in order to yield better solution quality for some instances. We then choose to incorporate both intensification and diversification components into our heuristic. The approaches to intensification and diversification in the current implementation are described in the following sub-sections.

4.3.1 The Intensification Using Restart from Good Solutions Found in Earlier Phases

The intensification approach in our heuristic is an application of the *elite selection* strategy, as discussed in Glover (1994). This approach uses a list to record solutions thought to be good during the course of the search. When the best solution is not improved after a number of iterations, a solution recorded in the list is removed from the list, and the search is resumed from the solution.

In the current implementation, we record in a list any solution that is better than the best ever found before. The list is designed to accommodate at most five objects, so that at most five good solutions are memorized. The list operates like a stack: a new good solution is always added at the end of the list, and the one at the end of the list is always removed first for re-starting the search. However, it is a stack that never overflows, for when a new solution needs to be added, and the list is full with five members, the object at the beginning of the list is thrown away, and the new one is pushed in at the end of the list. The attributes for the move that is executed at the good solution to reach the next solution is saved together with the solution itself in the list. When resuming the search from a solution from the list, the tabu list is emptied first, and then the set of move attributes accompanying the solution is added to the tabu list, so that a different search path can be launched. To be sure the solutions in the elite solution list are somewhat different, a new best solution is recorded only when no more best solutions are found in the next 30 iterations after the new best solution is found.

The intensification approach can be outlined as follows:

1. search starting from an initial solution and update the elite solution list during the search until no new best solution is found for a number of iterations

```
2. while the elite solution list not empty do
        remove the solution and move attributes from the end of the elite
        solution list,
        empty the tabu list,
        add the move attributes accompanying the solution to the tabu list,
        search starting from the removed elite solution and update the elite
        solution list during the search until no new best solution is found for
        a number of iterations
   end while
```

4.3.2 The Diversification Using Frequency-Based Memory in Combination with Aspiration by Search Direction

One diversification element in the current implementation is based on the idea of penalizing frequently executed moves. To implement the idea, some long term memory structures need to be defined so that the frequencies of the moves executed throughout the search can be recorded. In our case, for the swap move type, we record the number of times item i in period t is swapped to become the item produced first or last. With the use of an array swpfrq[P][T], where P and T are the number of items and the number of periods respectively, the frequency count of a swap move executed on item i in period t is stored in swpfrq[i][t].

For lot shift moves (both to an earlier and to a later period), we record the number of times period t and s are involved in shifting item i. An array ltmvfrq[P][T], where P and T are the number of items and the number of periods respectively, is used to record the frequency counts of the lot shift moves. Thus, if a lot move that shift item i from period t to period s is executed, the counts ltmvfrq[i][t], and ltmvfrq[i][s] are increased by one.

These frequency counts on moves are then used in a penalty term applied to the objective function so that the frequently performed moves will be discouraged during the diversification periods. The penalty term is obtained by multiplying the frequency count and a penalty factor. Let c be the frequency count, and let w be the penalty factor. The modified objective function for evaluating a move during the diversification periods is

$$g = f + w*c,$$

where f is the original objective function. However, the penalty term is not applied to moves that lead to solutions better than the best solution found so far.

To evaluate a move with respect to the modified objective function g, we need to further define the penalty factors corresponding to the two types of frequency counts. Based on the discussion in Glover et al. (1993) and our experiments, we set the penalty factor for the swap move frequent counts to be the product of the square root of the neighborhood size of the swap moves and a multiplier whose value is determined through experiments. The multiplier is set to 3 after some experiments. If we use the maximum possible neighborhood size (discussed in previous chapter) in the penalty factor, the penalty factor for swap move is $3\sqrt{(2P-3)T}$. Therefore, the penalty term applied to a swap move involving item i in period t is $3\sqrt{(2P-3)T}$swpfrq[i][t].

As for the lot shift moves, the penalty factor is also set to be the product of the square root of the neighborhood size of the lot shift moves and a multiplier whose value is determined through experiments. The difference here is that the multiplier takes one of the two values, 0.01 and 10, alternatively depending on what search phase, improving or non-improving, the search process is currently in. Thus, the penalty factor for a lot shift move is $m\sqrt{T(T-1)P}$, where m is the multiplier taking value of either 0.01 or 10, and the maximum possible neighborhood size for the two types of lot moves are used. The penalty term applied to a move that shift item i from period t to period s is $m\sqrt{T(T-1)P}$(ltmvfrq[i][t] + ltmvfrq[i][s]).

For the lot shift moves, we use 0.01 for m in the improving phases and 10 in the non-improving phases to vary the diversification effect in different search phases. Also an aspiration criterion is combined in the process. Larger penalties resulting from the higher multiplier value, 10, create a stronger diversification effect in the non-improving phases. Cnce a solution better than its immediate predecessor is found during a non-improving phase, an improving phases begins. In an improving phase, in addition to lessening the diversification effect by applying a smaller penalty term resulting from the lower multiplier value, 0.01, we employ a special aspiration criterion that allows improving moves to be executed regardless of their tabu status so that a true local optimum can be reached when an improving phase ends. Once a true local optimum is reached, the search enters a non-improving phase. Then, the special aspiration criterion is discontinued, and larger penalties begins to apply. Thus, stronger diversification and aspiration alternate for the non-improving and improving phases. This approach is a case that contains *aspiration by search direction,* as mentioned in Glover (1994).

Let m be the multiplier for penalty factor for moves of type 4 and 5, let r be the threshold for triggering the start of the diversification with aspiration by search direction, let f_{cur} be the objective function value of the current solution, and let $f_{trial\text{-}best}$ be the objective function value of the best neighbor solution. The rules for changing the multiplier's value for the lot moves, and triggering the aspiration criterion are outlined as follows:

```
if the iteration counter < r then
    search with neither diversification, nor aspiration criterion
else if the iteration counter = r then
    start aspiration criterion,
    $m \leftarrow 0.01$
else
    if aspiration criterion in effect then
        if $f_{cur} > f_{trial\text{-}best}$ then
            continue the aspiration criterion,
        else                (the current solution is a local optimum)
            discontinue the aspiration criterion,
            $m \leftarrow 10$,
            identify the best move again with $m = 10$
        end if
    else  (aspiration criterion not in effect means heavier diversification)
        if $f_{cur} > f_{trial\text{-}best}$ then
            start aspiration criterion,
            $m \leftarrow 0.01$
        else
            continue the search with $m = 10$
        end if
      end if
    end if
```

4.3.3 The Diversification with Full Restart from Other Initial Solutions

Our heuristic contains another diversification strategy, i.e., *full restart from other initial solutions*. The first initial solution we used is the lot-for-lot solution. Then, the Wagner-Wintin's solution is used as one restart initial solution. Lastly, the solution in which the sum of the demands in all the periods for an item is put in the first period with a non-zero demand entry is used as another restart initial solution. Lot for lot solution incurs no inventory costs, whereas the second restart initial solution incurs the lowest setup costs. The Wagner-Wintin's solution produces an optimal cost when capacity restriction is disregarded. We found the strategy of full restart from the other initial solution to be useful in the current implementation for each type of the initial solutions tends to produce better solutions for instances with certain characteristics. However, the strategy results in a longer computation time.

4.4 Variable Penalty Rate

Here, we comment on the penalty rate applied to the violation of capacity constraints. In the current implementation, the rate is allowed to vary between a minimum value and a maximum value. We vary the rate's value because a high rate may be able to drive out infeasibility fast, but it may also prevent the search from reaching the regions other than the one containing the current local optima. Allowing infeasibility during the course of the search is necessary to help the search to reach the other good regions. The minimum and maximum penalty rate's values in the current implementation are set to 5, and 100 respectively. Let f_{cur} and f_{pre} are the objective function values for the current solution and the last solution respectively, and let p be the penalty rate. The penalty rate is altered during the course of the search according to the following rules:

```
if $f_{cur} < f_{pre}$ then
  if $p > 5$ then
    $p \leftarrow p - 5$
  end if
else
  if $p < 100$ then
    $p \leftarrow p + 5$
  end if
end if
```

4.5 The Main Procedure

Now we can describe the main procedure, in which all the search elements we adopted in our current implementation are integrated. The main procedure CLSPTABU invokes a procedure called SEARCH. Before presenting SEARCH, we define the parameters used in our procedure as follows:

k': the threshold of iteration count for starting diversification with aspiration by search direction;

m1: the multiplier for penalty factor of the move frequency applied to moves of type 1;

m2: the multiplier for penalty factor of the move frequency applied to moves of type 4 and 5;

p: the penalty rate for capacity violation;

L_{min}: the minimum tabu list length;

L_{max}: the maximum tabu list length;

r_{max}: the maximum number of the iterations before terminating after the last best solution is found.

Also, let P and T be the number of items and the number of periods respectively, let X and X* be the current solution and the best solution respectively, let $X_{trial\text{-}best}$ be the best trial solution at each iteration, let f(X) and f(X*) be the current objective function value and the best objective function value respectively, let k and k* be the iteration counter and the value of iteration counter at which the last best solution is found respectively, and let r be the iteration counter after the last best solution is found. The procedure SEARCH is outlined as follows:

SEARCH (X, X*, f(X*), Move Attributes)

Step 0 (initialization):

Set $k \leftarrow 1$, $r \leftarrow 1$, $m1 \leftarrow 0$, $m2 \leftarrow 0$, and $p \leftarrow 50$. Empty the tabu list. If X is a solution from the elite solution list, add the move attributes associated with X to the tabu list.

Step 1 (move evaluation and best move selection):

Call SPMVAHD, SPMVBAK, LTMVFWD, LTMVBWD, and if (k mod P) $\leq$ (P / 3), call SWAP to evaluate all the moves, and identify the best move for the next solution. The moves of types 1, 4 and 5 are evaluated with respect to the modified objective function g, which includes the penalty term for move frequency recorded in previous search. The moves of type 2 and 3 are evaluated with respect to the objective function f. If the special aspiration criterion is in effect, the move that yields the smallest value for the corresponding objective function is selected regardless of its tabu status. Otherwise, the best move selection is done in the usual manner by considering the tabu status of a move.

Step 2 (determination of parameter values for diversification and aspiration):

If k equals k', start the aspiration criterion, set m1 $\leftarrow$ 3, and set m2 $\leftarrow$ 0.01.

If k is greater than k', execute the following rules:

if the aspiration criterion is in effect and the best move is an improving one, continue the aspiration criterion, and keep m2 = 0.01;

if the aspiration criterion is in effect and the best move is a non-improving one, discontinue the aspiration criterion, set m2 $\leftarrow$ 10 for a heavier diversification, and go to step 1;

if the aspiration criterion is not in effect and the best move is a non-improving one, keep m2 = 10;

if aspiration criterion is not in effect and the best move is an improving one, start the aspiration criterion, and set m2 $\leftarrow$ 0.01 for a lighter diversification.

Step 3 (solution update and identification of good solution for intensification):

Update current solution by setting $X \leftarrow X_{trial\text{-}best}$. Update frequency counts for moves of type 1, 4, or 5. If $f(X) < f(X^*)$, set $X^* \leftarrow X$, $f(X^*) \leftarrow f(X)$, and $k^* \leftarrow k$. Set $k \leftarrow k+1$. If $f(X) < f(X^*)$, set $r \leftarrow 1$; otherwise set $r \leftarrow r+1$. If $(k-k^*) = 1$, memorize the attributes of the move executed. If $(k-k^*) = 30$, store the last best solution together with the attributes of the move that leads to the next solution from the last best solution into the elite solution list.

Step 4 (tabu list update):

If the current list length is less than L_{min}, add the move attributes to the list. Otherwise, if the move executed is an improving one and the current list length is greater than L_{min}, delete two elements from the list and add the move attributes to the tabu list;

if the move executed is an improving one and the current list length is equal to L_{min}, delete one element from the list and add the move attributes to the tabu list;

if the move executed is a non-improving one and the current list length is less than L_{max}, add the move attributes to the tabu list;

if the move executed is a non-improving one and the current list length is equal to L_{max}, delete one element from the list and add the move attributes to the tabu list.

Step 5 (penalty rate update):

If the move executed is an improving one and the current penalty rate is greater than 5, set $p := p - 5$; if the move executed is a non-improving one and the current penalty rate is less than 100, set $p := p + 5$.

Step 6 (termination check):

If r is greater than r_{max}, stop. Otherwise, go to step 1.

The main procedure CLSPTABU is presented as follows:

CLSPTABU

Step 0 (initialization):

Set $L_{min} \leftarrow (1-3/4)(P+T)$, $L_{max} \leftarrow (1+3/4)(P+T)$, $r_{max} \leftarrow 1.5(300 + 10P)$, and $k' \leftarrow 0.5(300 + 10P)$. Set the initial solution count $I \leftarrow 1$.

Step 1 (initial solution):

If $I = 1$, find the lot for lot solution; if $I = 2$, find the Wagner-Wintin's solution; if $I = 3$, find the solution in which all the demanded quantity for an item is produced in first period with non-zero demand; if $I = 4$, stop.

Make the current solution X equal to the initial solution. Call SEARCH (X, X*, f(X*), Null Move Attributes).

Step 2 (elite solution list check):

If the elite solution list is not empty, remove the first solution and the associated move attributes from the list, and set the current solution X to equal the solution removed from the elite solution list. Otherwise, set $I \leftarrow I + 1$, go to step 1.

Step 3 (intensification):

Call SEARCH (X, X*, f(X*), Move Attributes). Go to step 2.

4.6 A Bounding Procedure

Our experiments on solving the CLSP with setup times and setup carryovers with exact solution procedure using CPLEX (a linear optimization software package) show that it is usually impossible to find an optimal solution even for a small size problem, say, 6 items and 8 periods, within a reasonable time span. Therefore, a bounding procedure that finds lower bounds of our problem is required so that we can have a relative measure of our heuristic's solution quality. Developing a procedure that produces tight lower bounds for the problem is still a subject under study. Here, we propose a bounding procedure for finding lower bounds that may be poor for some instances.

We observe intuitively that the best result we can achieve by allowing setups to be carried over from one period to the next in the CLSP with setup times and setup carryovers is that one setup carryover is realized in each period. It follows that in the best of circumstances, the number of setups in each period is equal to the number of the items produced in that period minus one. Using the notations in model M1, it is

$$N_t = \sum_i Y_{it} - 1 \qquad t = 1,\ldots,T \qquad (1)$$

Since equations (1) give the best possible condition for lowering the overall cost, if we use them as surrogates for all the setup counting and partial sequencing constraints in model M1, the optimal value cannot be increased. Thus, the optimal value of the objective function under condition (1), the original demand and capacity constraints, is a lower bound on that of model M1.

Now we derive a lower bound for our problem through analytical reasoning. If we remove the constraints (6) through (16), and constraints (18) from model M1, we obtain the following:

Model M3

Minimize $Z = \sum_i \sum_t H_{it} I_{it} + A \sum_t N_t + \sum_i \sum_t F_t Y_{it}$ (1)

Subject to

$I_{i,t-1} + X_{it} - I_{it} = D_{it}$ $i = 1,\ldots,P;\ t = 1,\ldots,T$ (2)

$\sum_i b_i X_{it} + q N_t \le C_t$ $t = 1,\ldots,T$ (3)

$X_{it} - M Y_{it} \le 0$ $i = 1,\ldots,P;\ t = 1,\ldots,T$ (4)

$N_t = \sum_i Y_{it} + \sum_i S_{it} + \sum_i V_{it} + \sum_i O_{it} - 1$ $t = 1,\ldots,T$ (5)

$X_{it}\ I_{it}, N_t, S_{it}, V_{it}, O_{it} \ge 0$ $i = 1,\ldots,P;\ t = 1,\ldots,T$ (6)

$Y_{it} \in \{0, 1\}$ $i = 1,\ldots,P;\ t = 1,\ldots,T$ (7)

$I_{i0} = 0$ $i = 1,\ldots,P$ (8)

Since removing constraints cannot increase the optimal value, the optimal value of model M3 is a lower bound on the optimal value of model M1.

Further, if we substitute N_t in the objective function (1) and in inequalities (3) with equations (5), we have model M4 expressed as follows:

Model M4

Minimize

$$Z = \sum_i \sum_t H_{it} I_{it} + A \sum_t \left(\sum_i Y_{it} + \sum_i S_{it} + \sum_i V_{it} + \sum_i O_{it} - 1 \right) + \sum_i \sum_t F_t Y_{it} \qquad (1)$$

Subject to

$$I_{i,t-1} + X_{it} - I_{it} = D_{it} \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (2)$$

$$\sum_i bX_{it} + q\left(\sum_i Y_{it} + \sum_i S_{it} + \sum_i V_{it} + \sum_i O_{it} - 1\right) \le C_t \qquad t = 1,\dots,T \quad (3)$$

$$X_{it} - M\,Y_{it} \le 0 \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (4)$$

$$X_{it},\ I_{it},\ N_t,\ S_{it},\ V_{it},\ O_{it} \ge 0 \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (5)$$

$$Y_{it} \in \{0, 1\} \qquad i = 1,\dots,P;\ t = 1,\dots,T \quad (6)$$

$$I_{i0} = 0 \qquad i = 1,\dots,P \quad (7)$$

Observing the above formulation, we can show that the values of all the variables S_{it}, V_{it}, and O_{it} in an optimal solution for model M4 must be zero. Assume for the sake of contradiction that in an optimal solution for model M4, the value of some S_{it}, V_{it}, and O_{it} variables are nonzero. We can always construct another solution out of the optimal solution by changing all those nonzero S_{it}, V_{it}, and O_{it} into zero, and keeping the values for other variables. Since constraints (3) are still satisfied after the change, the constructed solution is a feasible solution. But the objective function value of this new solution will be lower than that of the original solution. This contradicts the assumption of the optimality of the original solution.

Since variables S_{it}, V_{it}, and O_{it} will be zero in an optimal solution, we can set S_{it}, V_{it}, and O_{it} to equal zero to obtain the following equivalent model:

Model M5

Minimize $Z = \sum_i \sum_t H_{it} I_{it} + \sum_i \sum_t (A+F_t) Y_{it} - TA$ (1)

Subject to

$I_{i,t-1} + X_{it} - I_{it} = D_{it}$ $\quad i = 1,\ldots,P;\ t = 1,\ldots,T$ (2)

$\sum_i b_i X_{it} + q \sum_i Y_{it} \leq C_t + q$ $\quad t = 1,\ldots,T$ (3)

$X_{it} - M Y_{it} \leq 0$ $\quad i = 1,\ldots,P;\ t = 1,\ldots,T$ (4)

$X_{it}, I_{it} \geq 0$ $\quad i = 1,\ldots,P;\ t = 1,\ldots,T$ (6)

$Y_{it} \in \{0, 1\}$ $\quad i = 1,\ldots,P;\ t = 1,\ldots,T$ (7)

$I_{i0} = 0$ $\quad i = 1,\ldots,P$ (8)

Since the optimal value of model M5 is the same as that of model M3, it is also a lower bound on the optimal value of model M1.

If the constant term TA in the objective function in model M5 is disregarded, this new model is just model M2 for the CLSP with setup times, which was studied in Trigeiro et al. (1989). The constant setup time in model M5 makes it the special case of model M2. Although there are no known efficient solution procedures for solving this problem optimally, the TTM procedure produces a lower bound on the optimal value of the CLSP with setup times. This lower bound minus the term TA is a lower bound on the optimal value of our problem.

To obtain a lower bound for our problem, we first prepare a lower bound problem for model M2 by changing the capacity values and setup cost values in our test problems. The capacity value for period t now is C_t+q, and the setup cost in period t is $A+F_t$. We then solve the lower bound problem using the TTM procedure. A lower bound for our problem can be found by subtracting the constant item TA from the lower bound value produced by the TTM procedure.

The review in chapter 2 on the TTM procedure found that the quality of TTM lower bound appears good for it is within 0.08 percent deviation from the best lower bound. Thus, the tightness of the lower bounds on the optimal value of our problem, for the most part, depends on the magnitude of the setup cost since there is a constant item TA to be subtracted from the TTM lower bound, and the level of setup time since the capacities in all the period are increased by q units of the resource. Also, the number of setup carryovers in an optimal solution affects the tightness of a bound because the bounding procedure is based on the assumption that in each period, one setup carryover is realized.

4.7 Summary

The approaches to adapting various global search strategies, such as intensification and diversification, to our heuristic to improve its overall performance are discussed. We then present our heuristic's main procedure in which all the search elements (local and global) are put together. Also, a procedure for finding the lower bounds for the CLSP with setup times and setup carryovers is proposed.

Chapter 5
COMPUTATIONAL STUDY

In chapter 4, after giving an explanation of the global search elements, we presented CLSPTABU, our heuristic's main procedure for solving the CLSP with setup times and setup carryovers, along with a bounding procedure. This chapter presents the computational study in which our tabu search (TS) heuristic is tested, and its performance is evaluated. First, we present the testing results obtained during the heuristic development phase where we compare the solutions found by the basic version of our heuristic with the optimal solutions for a set of small size problems. Secondly, we describe the experimentation scheme through which our heuristic is tested on a large number of problems. We then present the analysis of solution gap. Following that, we present the analysis of cost difference. Based on the results for both the solution gap and the cost difference, we comment on the solution quality achieved by our heuristic. Lastly, we present the results for computation times.

5.1 Preliminary Testing Results

In the early phase of heuristic development, we tested the basic version of our heuristic on a set of small size problems. The test problems are randomly generated. The tabu search solutions are compared with the optimal solutions if an optimal solution can be found for these problems using CPLEX within eight hours. We present the computational results in Table 1. In Table 1, the computation times required by tabu search heuristic are CPU seconds on a PC 486DX-33 machine, and the computation times required by CPLEX are CPU times on a SUN 4 (Sparc 1) workstation. The solution gaps are in percentage term. The basic version of our tabu search heuristic produced impressive results for these small size problems.

Table 1
Tabu Search Solution vs. Optimal Solution

No. of Items	No. of Periods	TS Solution	Optimal Solution	Solution Gap	TS Sol. Time sec.	Optimal Sol. Time
12	4	3624	3624	0	1.60	3.83 sec.
12	4	6285	6096	3.1	1.82	70 sec.
12	4	5030	5003	0.5	2.34	226 sec.
12	4	6337	6311	0.4	2.12	102 sec.
12	4	6056	5788	4.6	2.12	11.7 min.
8	6	5206	5166	0.7	2.45	18.3 min.
8	6	5874	5576	5.3	2.50	4.9 hr.
8	6	5555	5500	1.0	2.21	4.0 hr.
8	6	5698	5239	8.7	2.23	7.3 hr.
4	12	2485	2153	15.4	3.12	35 min.
4	12	2125	1972	7.8	2.87	75 min.

After some global search strategies were incorporated into our tabu search heuristic, an extensive computational study was conducted to test the heuristic on a large number of test problems of larger size.

5.2 Experimentation Scheme

The test problems used in the computational study are prepared by making minor modifications on the existing test problems from Trigeiro et al. (1989). In this section, we present the experimentation scheme in the following order: Trigeiro et al.'s test problem, our test problems, and performance measures.

5.2.1 Trigeiro et al.'s Test Problems

The test problems created by these authors are available to anyone who is interested in the research on the CLSP, and are featured with various problem characteristics whose effect on problem difficulty might be significant. The characteristics considered in the test problems are the demand variability, the variability of capacity consumption per unit production, the setup time level, the setup cost / holding cost ratio, the capacity tightness, and the problem size. Since the effect of these characteristics on our heuristic's solution quality will be analyzed below, we give a description about how these test problems were created by these authors as follows:

Demand. For every item in each period, the demand value is generated from a uniform distribution with a mean of 100. 25% of the items in each of the first four periods are randomly chosen to be given a zero value for the demands, and the demands in later periods are augmented to maintain the overall demand of 100 per period. This is to simulate the demand pattern with increasing trend that can exist in manufacturing settings where some demands are already satisfied by batching decision made in the past. The effect of demand variability can be examined by varying the coefficient of variation of the demand values.

Capacity Consumption per Unit Production. The amount of capacity consumed by a unit of production either is set to one, or drawn uniformly from an interval of [0.5, 1.5] for the case of high variability of this parameter.

Setup Time. Setup times are item dependent. For each item, a setup time is randomly generated from a uniform distribution. The distribution parameter is defined by the requirement on average value and variability of setup times. For the case of low mean and high variability of setup times, the values are drawn from [10, 50].

Setup Cost/Holding Cost Ratio. Both the setup costs and inventory holding costs in the test problems are set to be item independent, but not to vary with periods. For a given mean demand, the setup cost/holding cost ratio determines the time-between-order (TBO) in an Economic Order Quantity (EOQ) model. For each item, the values of setup cost, and inventory holding cost are drawn from the streams of uniformly distributed random values. For example, the ratio with a low mean and a high variability is created with the setup cost value from [200, 1000], and the holding cost value from [1, 5].

Capacity. Capacity tightness is controlled through a target average utilization of capacity. In phase 1, and 2, the required capacity levels are calculated based on the lot-for-lot solution plus a setup time for each item produced. First, the total amount of resource required by the lot-for-lot solution for the whole planning horizon is averaged over all periods. The capacity then is determined by dividing the resulting number by a capacity utilization factor, either 0.75, or 1.00, or 1.10. In phase 3, the capacity available in each period is generated in the same fashion except that an EOQ solution is used, and capacity utilization factors are set to 0.75, 0.85, and 0.95. An EOQ solution results in fewer setups in the case of larger TBO, so that available capacity generated with the measure on EOQ lot size is lower for higher TBO.

Problem Size. Problem size is measured by both the number of items, and the number of periods. Problems in phase 1 have 4, 6, or 8 items, and 15 periods. All the test problems except 25 larger problems in phase 2 have 6 items and 15 periods. The 25 larger problems have the size of 12x15, 24x15, 6x30, 12x30, or 24x30. In phase 3, problems have the size of 10x20, 20x20, or 30x20.

Infeasibility For those test problems that were reported as infeasible by the TTM procedure, new problems were generated until enough number of problems considered feasible are found for each combination of the problem characteristic's parameter.

5.2.2 Our Test Problems

The setup times and setup costs in the original problems vary across items, whereas the setup times and setup costs in the model for the CLSP with setup times and setup carryovers are assumed to be constant for all the items. Therefore, to use these test problems in our experimentation, we have to change the variable setup times and setup costs to the constant setup times and setup costs.

Two sets of test problems are generated based on the original problems through the two different ways in establishing the constant values for setup time and setup cost. In one set of the test problems, the constant values of setup time and setup cost are obtained by setting them to be equal to the averages of the original setup time values and setup cost values for all the items (referred to below as the problems with the average setup related values). In another set of the problems, they are generated by setting them to be equal to the least values among the original setup times values and setup costs values for all the items (referred to below as the problems with the least setup related values). The values of other parameters in the test problems remain intact.

The test problems with the average setup related values are equivalent to the original problems in that, on the average, the original value settings for the capacity tightness, setup time, and setup cost are kept roughly unchanged. The problems with the least setup related values are tested because we are interested in examining the effect of considering setup carryovers on the total cost under different setup related values. Also, the expectation that our lower bounds are tighter for the problems with lower setup related values can be verified.

5.2.3 Performance Measures

In this computational study, two measures, i.e., the solution gap and the cost difference, are used for solution quality evaluation. The test problems are solved first using our tabu search heuristic. To obtain the solution gap, the lower bounds are found in two steps using the bounding procedure discussed in the previous chapter. First, the lower bound problems are constructed through modifying the values of capacity, and setup cost in our test problems. Secondly, the TTM procedure is applied to solving the lower bound problems, and the lower bounds found by the TTM procedure are used to calculate the lower bounds of our problems. To obtain the cost differences, the test problems are solved using the TTM procedure. The difference between the cost of our tabu search solution and that of TTM solutions allow us to examine the effect of explicitly considering setup carryovers in modeling and solution procedure on the overall production cost.

Our tabu search procedure is coded in C, and the program is run on a SUN 4 (Sparc 1) workstation under the operating system Sun OS5.4. The TTM procedure is coded in FORTRAN, and its source code was obtained along with the test problems from the original authors. It was re-compiled using the f77-FORTRAN compiler, and run on the same workstation.

5.3 Analysis of Solution Gap

The Trigeiro et al's test problems are arranged in three phases by the authors. The 70 problems in phase 1 are not controlled in terms of systematically setting the values for problem characteristics. In phase 2, which contains 141 problems, the problem characteristics are varied one at a time by changing their values from the nominal values so that the effects of the problem characteristics on solution gap can be examined. In phase 3, the 540 problems are the result of a full factorial design on the five selected characteristics with 2 or 3 levels. Since the same structure presents in out test problems, we analyze the solution gaps in a fashion similar to that used in Trigeiro et al. (1989).

Table 2
Summary of the Solution Gaps (in %)

	Test Problems		
	phase 1 70 problems	phase 2 141 problems	phase 3 540 problems
Average Solution Gap:			
problems with the average setup related values	11.954	11.475	9.653
problems with the least setup related values	3.748	4.241	5.332

The overall computational results for the solution gap for the problems in each phases are summarized in Table 2. The solution gap is computed as:

$$Gap = \frac{TS\ objective\ value\ -\ Lower\ bound}{Lower\ bound}.$$

The first row in the table presents the average solution gaps for the problems in the three phases with the average setup related values, and the second row presents the average solution gaps for the problems in the three phases with the least setup related values. The solution gaps for the problems with lower levels of setup times and setup costs are consistently smaller than that for the problems with higher levels of setup times and setup costs. The largest average gap in the table is 11.954 percent. A large gap is due to either a large deviation of tabu search solution from optimal solution, or a weak bound. It is not possible to judge which is the case here. However, as we discussed at the end of the previous chapter about the quality of the lower bound, the lower bounds are expected to be poor when setup related values are high.

The effect of the various problem characteristics on the solution gap can be evaluated by analyzing the computational results for the test problems in phase 2. The problems in phase 2 include a set of problems with nominal values for all the problem characteristics. Other problems are created by varying the values for one characteristic at a time. The nominal values are 30 units for the average setup time, 600 for the average setup cost, range [1, 5] for the holding costs, 0.244 for the coefficient of demand variation, 1 for unit production capacity consumption for all the items, 100 percent for the lot-for-lot capacity utilization, 6 for the number of items, and 15 for the number of periods.

Table 3, and 4 present the analysis of solution gap with respect to the problem characteristics for the problems with the average setup values, and for the problems with the least setup values, respectively. For the problems with the least setup related values, the mean values of the parameters for controlling the levels of setup time, setup cost, and capacity utilization no longer be the same as the original ones, but the levels of the characteristics can still be distinguished since the least setup related values change proportionally with the mean values in the original test problems. So, in Table 4, the levels are indicated as low, medium, and high, instead of the original mean values.

The results in the two tables show a similar pattern for the solution gaps with respect to the problem characteristics. The observation is summarized as follows:

- There is a decrease in the solution gap, when setup time level is high. This can be explained by the fact that the setup times are included in the design of capacity measure when the problems are created, and hence savings in setups due to batching and setup carryovers loosen the capacity restriction more for the case of higher setup time values.
- High setup cost gives rise to a significantly large solution gap. This effect is probably in part because the gap between the lower bound and the optimal objective value is larger when setup cost is higher.
- The solution gap increases when the demand variation coefficient becomes high at 0.608, whereas the solution gap seems not affected when the demand variation coefficient is increased from low to medium.
- Introducing variability in the capacity consumption coefficient yields a larger solution gap.

- Tightening the capacity constraints to high level significantly increases the solution gap, whereas a small increase in capacity tightness has little effect on the solution gap. As discussed in Trigeiro et al. (1989), one of the reason for this effect is that high interdependency among the items competing for the limited resources makes problem hard to solve.
- Increase in the number of items results in a great decrease in solution gap. One of the explanation for this effect given in Trigeiro et al. (1989) applies here too. Lot sizing with setup times bears an analogy to bin packing. To solve a lot sizing problem with more products is like to pack the bins with more items of different size, which cab be done more easily. The number of period seems to have little effect on the solution gap.

Table 3
Analysis of Solution Gap (in %), phase 2 problems with the average setup related values

Setup Time	
Low (mean 30 units of capacity)	High (mean 90 units of capacity)
10.184	8.550

Ratio of Setup Cost to Holding Cost	
Low (mean 280)	High (mean 560)
10.212	23.457

Coefficient of Demand Variation		
Low (0.090)	Medium (0.244)	High (0.608)
9.454	8.496	13.263

Variability of Capacity Consumption Coefficient	
None (1.0)	High (0.5 - 1.5)
8.496	12.791

Lot-for-lot Capacity Utilization		
Low (0.75)	Medium (1.00)	High (1.15)
9.125	8.496	16.992

Problem Size		No. of Items		
		6	12	24
No. of Periods	15	8.496	4.259	1.996
	30	9.965	4.438	1.890

Table 4
Analysis of Solution Gap (in %), phase 2 problems with the least setup related values

Setup Time	
Low	High
3.141	1.524

Ratio of Setup Cost to Holding Cost	
Low	High
4.654	13.335

Coefficient of Demand Variation		
Low (0.090)	Medium (0.244)	High (0.608)
1.289	1.342	6.439

Variability of Capacity Consumption Coefficient	
None (1.0)	High (0.5 - 1.5)
1.342	3.622

Lot-for-lot Capacity Utilization		
Low	Medium	High
0.991	1.342	5.463

Problem Size		No. of Items		
		6	12	24
No. of Periods	15	1.342	0.257	0.138
	30	1.104	0.531	0.274

The problems in phase 3 allow us to further analyze the solution gap with regard to the following problem characteristics: the capacity tightness measured by EOQ capacity utilization, the number of items, the ratio of setup cost to holding cost measured by the time between orders, the demand variability, and the setup time. Table 5, and 6 summarize the analysis of solution gap with respect to these characteristics.

Table 5
Further Analysis of Solution Gap (in %), phase 3 problems with the average setup related values

	Low	Medium	High
EOQ Capacity Utilization (75%, 85%, 95%)	6.871	7.376	14.714
Problem Size (10, 20, 30 items)	15.358	7.946	5.657
Time Between Orders (TBO) (1, 2, 4 periods)	3.237	6.367	19.356
Demand Variability (CV = 0.35, 0.59)	---	9.325	9.982
Average setup Time (11, 43 units of capacity)	10.309	---	8.998

Table 6
Further Analysis of Solution Gap (in %), phase 3 problems with the least setup values

	Low	Medium	High
EOQ Capacity Utilization	3.399	3.688	8.907
Problem Size (10, 20, 30 items)	8.579	4.242	3.174
Time Between Orders (TBO)	1.860	2.916	11.219
Demand Variability (CV = 0.35, 0.59)	---	4.761	5.902
Average setup Time	6.254	---	4.409

The solution gaps are affected by the problem characteristics in a similar fashion in the above two tables. The demand variability and setup time seem to have a minor effect on the solution gap. There is a great increase in solution gap for problems with few items, and high setup cost. When capacity constraints are far tight, the solution gaps become larger.

The effect of demand variability seems less evident in phase 3 than in phase 2. The reason for this might be due to the fact mentioned in Trigeiro et al. (1989) that for each item, the demand values were generated independently, and the variability of aggregate demand is lessened by the combined effect of many independently generated demand for each item. When the number of the items becomes larger in the problems in phase 3, it is lessened more.

The findings in phase 2 and 3 indicate that the solution gap is large for problems with high setup costs, tight capacity constraints, and few items. The setup time level, the demand variability, and the variability in capacity consumption per unit production do not affect the solution gap significantly.

5.4 Analysis of Cost Difference

We compare in this section the total cost found by our tabu search heuristic for the CLSP with setup times and setup carryovers with the total cost found by the TTM procedure for the CLSP with setup times alone. The cost difference used in the following discussion is defined as:

$$\text{Cost difference} = \frac{\text{TTM objective value} - \text{TS objective value}}{\text{TTM objective value}}.$$

The computational results for the cost difference are presented in the same way as that for the solution gap. We first present the overall results for the three phases of the test problems, and then present the cost differences for the problems with the different characteristics.

Table 7 summarizes the computational results for the cost difference for the problems in each of the three phases. The results reveal that on the average, the costs found by our tabu search heuristic for the CLSP with setup times and setup carryovers are lower than that found by the TTM procedure for the CLSP with setup times for all the three phases. The costs are reduced by over 20 percent in phase 1, 17 percent in phase 2, and 5 percent in phase 3.

The differences in cost savings among the three phases are because that the number of items in the test problems in the three phases are different. The effect of the number of items on the percentage cost saving will be discussed below.

In phase 1 and 2, our heuristic produces lower cost solutions for all the problems. For the problems in phase 3, our heuristic failed to find the lower cost solutions for 39 problems out of the 540 test problems in case the average setup related values are used in our test problems, and for 33 problems out of the 540 test problems in case the least setup related values are used in our test problems. By examining the test problems individually, it is found that our procedure failed to produce lower cost solutions for those problems that have highly tight capacity constraints, low setup time, low or medium setup cost, and high demand variability. The detailed computational results also indicate that for most of those cases, the costs of tabu search solutions exceed that of TTM solution by less than 2 percent, and for many problems, our heuristic finds nearly the same cost. That is why on the average, the costs of tabu search solutions are lower than that of TTM solutions.

Another observation is that the costs saved for the problems with the average setup related values are nearly the same as that for problems with the least setup related values. This means that the cost saving in percentage term is not much affected by the levels of setup times and setup costs.

Table 7
Summary of the Cost Differences (in %)

	Test Problems		
	phase 1 70 problems	phase 2 141 problems	phase 3 540 problems
Average Cost Difference :			
problems with the average setup related values	20.929	18.276	5.310*
problems with the least setup related values	20.294	17.149	5.511**

*TS objective value is higher than TTM objective value for 39 problems.
** TS objective value is higher than TTM objective value for 33 problems.

The cost differences for the problems with different characteristics in phase 2 are shown in Table 8 and 9. Table 8 is for the problems with the average setup related values, and Table 9 is for the problems with the least setup related values. Unlike the results for the solution gap, the results for the cost difference show that except the number of items, the problem characteristics appear to have very minor effect on the cost saving. The only problem characteristic that exhibits a major effect on the cost saving is the number of items. The cost saving becomes smaller as the number of items in a problem instance increases. The reason for this is that at most one setup can be carried over in each period when setup carryovers are allowed, and hence the cost saved due to the setup carryovers accounts for a smaller proportion of the total cost for the problems with more items provided the cost coefficients associated with the items are roughly the same for the problems with different number of items in them. Comparing the results in the two tables, we observe that the cost savings for problems with the average setup related values appears to be slightly higher than that for problems with the least setup related values.

Table 8
Analysis of Cost Difference (in %), phase 2 problems with the average setup related values

Setup Time	
Low (mean 30 units)	High (mean 90 units)
20.690	19.588

Ratio of Setup Cost to Holding Cost (mean)	
Low (mean 280)	High (mean 560)
19.640	18.747

Coefficient of Demand Variation		
Low (0.090)	Medium (0.244)	High (0.608)
21.839	20.594	22.426

Variability of Capacity Consumption Coefficient	
None (1.0)	High (0.5 - 1.5)
20.594	19.677

Lot-for-lot Capacity Utilization		
Low (0.75)	Medium (1.00)	High (1.15)
19.490	20.594	20.186

Problem Size		No. of Items		
		6	12	24
No. of Periods	15	20.594	9.873	4.728
	30	19.142	9.411	4.529

Table 9
Analysis of Cost Difference (in %), phase 2 problems with the least setup related values

Setup Time	
Low	High
18.825	19.091

Ratio of Setup Cost to Holding Cost	
Low	High
17.500	18.187

Coefficient of Demand Variation		
Low (0.090)	Medium (0.244)	High (0.608)
18.905	17.329	19.773

Variability of Capacity Consumption Coefficient	
None (1.0)	High (0.5 - 1.5)
17.329	19.215

Lot-for-lot Capacity Utilization		
Low	Medium	High
19.217	17.329	19.215

Problem Size		No. of Items		
		6	12	24
No. of	15	17.329	9.416	4.839
Periods	30	17.755	8.911	4.691

The cost difference analysis is carried out further on the problems in phase 3, and the results for the cost difference are presented in Table 10 and 11 for the problems with the average setup related values and the least setup related values respectively. The findings show that the number of items has a big impact on the cost saving. The costs appear to be reduced a little less for the problems with high capacity tightness, high setup cost, or low setup times. The cost saving patterns presented in the two tables for the two sets of problems do not exhibit much differences.

Table 10
Further Analysis of Cost Difference (in %), phase 3 with the average setup related values

	Low	Medium	High
EOQ Capacity Utilization (75%, 85%, 95%)	5.746	5.967	4.215
Problem Size (10, 20, 30 items)	10.049	3.844	2.037
Time Between Orders (TBO) (1, 2, 4 periods)	5.920	5.674	4.388
Demand Variability (CV = 0.35, 0.59)	---	5.344	5.274
Average setup Time (11, 43 units of capacity)	4.779	---	5.839

Table 11
Further Analysis of Cost Difference (in %), phase 3 problems with the least setup related values

	Low	Medium	High
EOQ Capacity Utilization	6.339	6.358	3.839
Problem Size (10, 20, 30 items)	9.712	4.305	2.517
Time Between Orders (TBO)	6.067	5.977	4.491
Demand Variability (CV = 0.35, 0.59)	---	5.434	5.589
Average setup Time	4.962	---	6.061

Based on the results for the problems in phase 2 and 3, it is found that the cost saving is inversely related to the number of items, and appears to be a little smaller for problems with high capacity tightness, high setup cost, or low setup time.

Comparing the results for the cost difference with that for the solution gap, we observe that all the testing problem characteristics except the number of items have a much weaker effect on the cost saving than on the solution gap. This suggests that with regard to saving the total cost in percentage term, our heuristic is not sensitive to the various problem characteristics.

Examining the overall results for the cost difference in Table 7, and for the solution gap in Table 2, some comments on the effectiveness of our heuristic can be made. For the problems with higher levels of setup times and setup costs, other things held equal, the cost savings due to setup carryovers are expected to be greater since the reduction in the number of setups reduces a larger amount of setup costs and generates more available capacity times which in turn help to reduce more inventory costs. On the other hand, for the problems with higher levels of setup times and setup costs, the total cost incurred in a CLSP model without considering setup carryovers will be greater too. Thus, other things held equal, the cost savings in percentage term due to the setup carryovers are expected to be within a close range regardless of the levels of setup times and setup costs.

The cost savings achieved by our heuristic are consistent with this expectation in that the computational results for the cost difference show that for the problems with different levels of setup times and setup costs, the cost savings realized by our procedure account for a nearly same percentage of the total costs found by the TTM procedure for the CLSP with setup times alone. In contrast, the solution gaps for the problems with higher levels of setup times and setup costs are much larger than that for the problems with lower levels of setup times and setup costs. A large solution gap means that either the objective value of a heuristic solution is not close to the optimal objective value, or the lower bound is not tight. In our case, since the percentage cost savings achieved by our procedure for the problems with higher setup related values are nearly the same as that for the problems with lower setup related values, we suspect that the large solution gaps for the problems with higher setup related values are, for the most part, due to the large gaps between the optimal objective values and our lower bounds.

5.5 Computation Times

The computation times required by the current implementation of our heuristic are presented in Table 12, and are relatively long. Computation time tends to increase quickly with the increase in problem size, especially with the increase in the number of periods. The reasons for this are because of the use of the exhaustive neighborhood search at each iteration, and the multiple restarts form other initial solutions. Candidate list strategy is a way to limit the amount of neighbors evaluated at each iteration, and hence is often effective in reducing computation times. The approach to applying a candidate list strategy to our heuristic remains to be worked out.

Table 12
Average Tabu Search Computation Times (CPU seconds on a SUN 4 workstation)

phase 1 and phase 2

		Number of Items				
		4	6	8	12	24
Number of Periods	15	42.17	54.87	88.59	141.70	438.30
	30		277.50		712.80	2125.40

phase 3

		Number of Items		
		10	20	30
Number of Periods	20	168.50	440.50	800.50

For a comparison, the computation times required by TTM procedure are captured for problems in phase 2, and are presented in Table 13. It can be seen that the TTM procedure requires much less computation times.

Table 13
Average Computation Times Used by the TTM Procedure (CPU seconds on a SUN 4 workstation)

phase 2

		Number of Items		
		6	12	24
Number of Periods	15	2.71	5.40	10.76
	30	14.38	31.72	57.10

5.6 Summary

Our heuristic is tested on a large number of test problems, and evaluated with two relative measures, the solution gap and the cost difference. In this computational study, two sets of the test problems are created from Trigeiro et al.'s problems using the average setup related values and the least setup related values respectively. Computational results show that the solution gaps for the problems with the average setup related values are larger (average 10.21 percent) than that for problems with the least setup related values (average 4.98 percent). The solution gap is larger for the problems with higher setup cost, tighter capacity constraints, and smaller number of items. The variability of demand, the variability of capacity consumption per unit production, and the level of setup time have a relatively minor effect on the solution gap. Computational results reveal that on the average, the cost saving accounts for a roughly same proportion of the total costs found by the TTM procedure for solving the CLSP with setup times alone, for the problems with the average setup related values and the problems with the least setup related values. The cost savings in percentage term appears not sensitive to the various problem characteristics except the number of items. Based on the fact that our heuristic reduces the total costs in almost the same percentage for the problems with higher or lower level of setup related values, we suspect that the large gaps for the problems with the higher level of setup times and setup costs result from the poor lower bounds for these problems. The computation times required by the current implementation of our heuristic are quite long, and the work to reduce them remains to be done.

Chapter 6
CONCLUSION

This chapter concludes the thesis. First, we summarize the research results, and then, we suggest some directions for future research.

6.1 Research Results

The setup carryover issue was explicitly addressed in a modeling framework for the CLSP with setup times and setup carryovers in the large time bucket setting by Gopalakrishnan et al. (1995) so that the total cost of a lot sizing plan is reduced, or the possibility of finding feasible plans is increased in case there exists no feasible plan for the CLSP with setup time alone. Our research here was focused on developing a tabu search heuristic for the CLSP with setup times and setup carryovers on purpose to facilitate the application of the model to the real world decision problems of this kind. Based on the results obtained from our computational study, the viability of the tabu search heuristic approach for solving this class of the CLSP can be determined.

By nature, tabu search is a meta-heuristic which must be tailored to the specific of the particular problem under study. For our problem, first, we allowed infeasible solutions that violate the capacity constraints to be visited through a penalty term applied to the objective function. This provides a flexibility in our heuristic for the search can be started from any solution as long as it meets the demand requirements. In addition, this strategy helps to lead the search toward the regions not covered before when the penalty rate is varied in our heuristic.

Then, we designed five move types to respond to the nature of the decisions to be made in the problem. The problem is composed of two types of interrelated decisions, i.e., partial sequencing decision for setup carryovers, and lot sizing decision. In our heuristic, three move types are designed to handle the issue of partial sequencing. The three move types are the position swap between the item produced first or last and the other item within a period, the setup move to the previous period, and the setup move to the next period. Also, two move types are designed to handle the issue of lot sizing. The two move types are the lot shift to an earlier period, and the lot shift to a later period. Together, the five move types allow us to implement the local search in our heuristic.

To enhance the overall performance, various global search strategies and the associated memory structures were employed in our heuristic. The main search strategies include the tabu list with variable length, the intensification using an elite solution list, the diversification using the move frequency memories combined with the aspiration by search direction, the diversification with full restart from other initial solutions, and the variable penalty rate for capacity violation.

A procedure that finds the lower bounds for our problem was proposed too. The bounding procedure works according to the following steps: constructing the lower bound problems, solving them with the TTM procedure, and calculating the lower bounds of our problem based on the TTM bounds.

An extensive numerical experimentation was conducted. With small modification made on the setup related values in the original problems from Trigeiro et al. (1989), two sets of test problems, one with the average setup related values, and another with the least setup related values, were prepared. The computational results were analyzed with regard to two measures, i.e., the gaps between our tabu search solution and the lower bound, and the difference between the cost of our tabu search solution for the CLSP with setup times and setup carryovers and the cost of TTM solution for the CLSP with setup times. Regarding the effect of problem characteristics on the solution gap, larger gaps are found for the problems with higher setup cost, tighter capacity constraints, and smaller number of items. The variability of demand, the variability of capacity consumption per unit production, and the setup time level have a relatively minor effect on the solution gap. The percentage cost saving achieved by our heuristic appears insensitive to the various problem characteristics except the number of items.

The overall computational results revealed that although the solution gaps for the problems with higher setup related values are larger, the percentage cost savings are found to be almost the same for the problems with higher or lower setup related values. Based on the fact that our heuristic reduces the total costs in a nearly same percentage for the problems with higher or lower level of setup related values, and the solution gaps for the problems with the least setup related values average less than 5 percent, we can conclude that our heuristic performs well in producing good near optimal solutions for this computationally hard problem.

The computation times required by the current implementation of our procedure were found to be relatively long, and the way to reduce them needs to be worked out.

One contribution of this research is that a heuristic solution procedure for solving the CLSP with setup times and setup carryovers is developed, and its performance is encouraging. The work remains to be done is to reduce computation times, and improve the solution quality for those problems for which our procedure failed to find a better solution.

6.2 Future Research Directions

The directions for future research are presented with regard to the following three aspects: improvement of the current implementation of our heuristic, application of statistical bounding approaches, and possible extension of our heuristic to the related problems.

6.2.1 Improvement of Our Heuristic

In the current implementation, our heuristic failed to produce lower cost solutions than the TTM procedure for a small number of the test problems. For these instances, the comparisons between the detailed production plans generated by our heuristic and that by the TTM procedure will help us to work out good strategies for finding better solutions for these problems. We can even execute a search starting from the TTM solutions, and examine the outcomes for these problems.

It was found during the experimentation that the current approach to vary the penalty rate during the course of the search appears having minor effect on improving the solution quality. Therefore, another possible way to improve heuristic is to test other approaches to vary the penalty rate. For example, we can try the way described in Gendreau et al. (1994). The idea is to double the penalty rate if a certain number of previous solutions is found to be infeasible, and halve the penalty rate if a certain number of previous solutions is found to be feasible.

The computational times required by the current implementation is relatively long. One way to alleviate the problem is to implement a candidate list strategy, which usually can help to reduce computational effort while maintaining the effectiveness of a search procedure. The way to incorporate a candidate list strategy in our heuristic needs to be identified and implemented.

6.2.2 Statistical Bounding Procedure

The bounding procedure used in this study may not produce a tight enough bound for some problem instances since it is affected by the levels of setup times and setup costs. When the available deterministic bounding procedure fails to produce tight bounds, the use of statistical bound procedure may be a way to resolve the problem of measuring the solution quality of an approximate solution procedure. A statistical lower bound is defined as a constant z satisfying $\text{Prob}(z^* \geq z) \geq 1-\alpha$ for a fixed $\alpha \in (0,1)$, where z^* is the (unknown) optimal objective value of an optimization problem. Statistical procedures for generating the lower confidence limit z on z^* based on a sample of n local optimum solution $z_1, \ldots, z_n$ found by a local search procedure are available, and Derigs (1985) reported some extensive computational results on the statistical lower bounds for the traveling salesman problem and the quadratic assignment problem. The application of statistical bound to our problem is worth investigating.

6.2.3 Extensions of Our Heuristic to the Related Problems

Our heuristic currently solves the setup carryover model in which setup times and setup costs across items are assumed to be constant. One extension of the heuristic procedure is to make it to solve the setup carryover model with the item-dependent setup times and setup costs. The modification to accommodate the item-dependent setup related values in our heuristic algorithm does not require much effort, because each time a trial solution is evaluated, the item and the periods involved in a move are known, and hence the corresponding cost items and setup time associated with the item can be easily identified and used in move evaluation. It would be interesting to see the heuristic's solution quality for this class of problems, when some measures for solution quality could be available.

Extending our heuristic to solving the multi-machine CLSP model is another goal in future research. What we need to do is to extend the move mechanisms designed for the single machine CLSP to the multi-machine CLSP, but this can be challenging, and requires much more research effort.

References

Anderson, E. J., and B. S. Cheah, 1993, Capacitated lot-sizing with minimum batch sizes and setup times. International Journal of production Economics, 30-31, 137-152.

Aras, O. A., and L. A. Swanson, 1982, A lot sizing and sequencing algorithm for dynamic demands upon a single facility. Journal of Operations Management, 2, 177-185.

Bahl, H. C., L. P. Ritzman, and J. N. Gupta, 1987, Determining lot sizes and resource requirements: a review. Operations Research, 35, 329-345.

Bitran, G. R., and H. H. Yanasse, 1982, Computational complexity of the capacitated lot size problem. Management Science, 28, 1174-1186.

Cattrysse, D, J. Maes, and L. N. Van Wassenhove, 1990, Set partitioning and column generation heuristics for capacitated dynamic lotsizing. European Journal of Operations Research, 46, 38-47.

Daniels, R. L., and J. B. Mazzola, 1993, A tabu search heuristic for the flexible-resource flow shop scheduling problem. Annals of Operations Research, 41, 207-230.

Dell'Amico, M., and M. Trudian, 1993, Applying tabu search to the job-shop scheduling problem. Annals of Operations Research, 41, 231-252.

Derigs, U., 1985, Using confidence limits for the global optimum in combinatorial optimization. Operations Research, 33, 1024-1049.

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

Dzielinski, B. P., and R. E. Gomory, 1965, Optimal programming of lot sizes, inventory and labor allocations. Management Science, 11, 874-890.

Eppen, G. D., R. K. Martin, 1987, Solving multi-item capapcitated lot-sizing problems using variable redefinition. Operations Research, 35, 832-848.

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

Gendreau, M., A. Hertz, and G. Laporte, 1994, A tabu search heuristic for the vehicle routing problem. Management Science, 40, 1276-1290.

Glover, F., 1986, Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 5, 533-549.

Glover, F., 1990, Tabu search - part II. ORSA Journal on Computing, 2, 4-32.

Glover, F., E. Taillard, and D. de Werra, 1993, A users guide to tabu search. Annals of Operations Research, Vol. 41, 3-28.

Glover, F., and M. Laguna, 1993, Tabu search. Modern Heuristic Techniques for Combinatorial problems, C. Reeves (editor), Blackwell Scientific Publications, Oxford, United Kingdom.

Glover, F., 1994, Tabu search fundamentals and uses. Personal Communication.

Gopalakrishnan, M., D. M. Miller, and C. P. Schmidt, 1995, A framework for modeling setup carryover in the capacitated lot sizing problem. International Journal of Production Research, 33, 1973-1988.

Hansen, P., 1986, The steepest ascent, mildest descent heuristic for combinatorial programming. Congress on Numerical Methods in Combinatorial optimization, Capri, Italy.

Herz, A., E. Taillard, and D. de Werra, 1992, Tabu search. Local Search in Combinatorial Optimization, J. K. Lenstra (editor), and also ORWP 92/18, Dep. de Mathematiques, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland.

Kleindorfer, P. R., and E. F. P. Newson, 1975, A lower bounding structure for lot-size scheduling problems. Operations Research, 23, 299-311.

Kuik, R., M. Salomon, L. N. Van Wassenhove, and J. Maes, 1993, Linear programming, simulated annealing and tabu search heuristics for lotsizing in bottleneck assembly systems. IIE Transaction, 25, No. 1, 62-72.

Laguna, M., and F. Glover, 1993, Integrating target analysis and tabu search for improved scheduling system. Expert Systems with Applications: An International Journal, 6, 287-297.

Laguna, M., J. W. Barnes, and F. Glover, 1991, Tabu search methods for a single machine scheduling problem. International Journal of Manufacturing, 2, 63-73.

Lasdon, L. S., and R. C. Terjung, 1971, An efficient algorithm for multi-item scheduling. Operations Research, 19, 946-969.

Lozano, S., J. Larraneta, and L. Onieva, 1991, Primal-dual approach to the single level capacitated lot-sizing problem. European Journal of Operations Research, 51, 354-366.

Maes, J., J. O. McClain, and L. N. Van Wassenhove, 1991, Multilevel capacitated lotizing complexity and LP-based heuristics. European Journal of Operations Research, 53, 131-148.

Manne, A. S., 1958, Programming of economic lot sizes. Management Sciences, 4, 115-135.

Mooney, E. L., and R. L. Rardin, 1993, Tabu search for a class of scheduling problem. Annals of Operations Research, 41, 253-278.

Newson, E. F. P. 1975a, Multi-item lot size scheduling by Heuristic - part I: with fixed resources. Management Science, 21, 1186-1193.

Newson, E. F. P. 1975a, Multi-item lot size scheduling by Heuristic - part II: with variable resources. Management Science, 21, 1194 -1203.

Reeves, C. R., 1993, Improving the efficiency of tabu search for machine sequencing problem. Journal of Operations Research Society, 44, 375-382.

Salomon, M., R. Kuik, and L. N. Van Wassenhove, 1993, Statistical search methods for lotsizing problems. Annals of Operations Research, 41, 453-468.

Smith-Daniels, V. L., and L. P. Ritzman, 1988, A model for lot sizing and sequencing in process industries. International Journal of Production Research, 26, 647-674.

Taillard, E., 1994, Parallel taboo search techniques for the job shop scheduling problem. ORSA Journal on Computing, 6, 108-117.

Taillard, E., 1991, Robust taboo search for the quadratic assignment problem. Parallel Computing, 17, 443-455.

Trigeiro, W. W., L. J. Thomas, and J. O. McClain, 1989, Capacitated lot sizing with setup times. Management Science, 35, 353-366.

Wagner, H. M., and T. M. Whitin, 1958, Dynamic version of the economic lot size model. Management Science, 5, 89-96.

Widmer, M., and A. Hertz, 1989, A heuristic method for the flow shop sequencing problem. European Journal of Operations Research, 41, 186-193.

Widmer, M., 1991, Job shop scheduling with tooling constraints: a tabu search approach. Journal of Operations Research Society, 42, 75-82.

Woodruff, D. L., and M. L. Spearman, 1992, Sequencing and batching for two class of jobs with deadlines and setup times. Production and Operations Management, 1, 87-102.