SOLVING THE DISCRETE LOTSIZING AND SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SET-UP COSTS AND SET-UP TIMES USING THE TRAVELLING SALESMAN PROBLEM WITH TIME WINDOWS

96/02/TM
(Revised version of 94/46/TM)

by

M. SALOMON*
M. M. SOLOMON**
L. N. VAN WASSENHOVE†
Y. DUMAS††
AND
S. DAUZERE-PERES‡

* Professor, at Erasmus University, Rotterdam, The Netherlands.

** Professor, at Northeastern University, Boston, USA and at GERAD, Montréal, Canada.

† Professor of Operations Management and Operations Research at INSEAD, Boulevard de Constance, Fontainebleau 77305 Cedex, France.

†† AD-OPT Technologies, Montréal, Canada.

‡ Ecole des Mines, Nantes, France.

Printed at INSEAD, Fontainebleau, France

Solving the Discrete Lotsizing and Scheduling Problem with sequence dependent set-up costs and set-up times using the Travelling Salesman Problem with time windows

Marc Salomon, Marius M. Solomon, Luk N. Van Wassenhove,
Yvan Dumas, and Stephane Dauzère-Pérès,

Erasmus University, Rotterdam, The Netherlands
Northeastern University, Boston, USA and GERAD, Montreal, Canada
INSEAD, Fontainebleau, France
AD-OPT Technologies, Montreal, Canada
Ecole des Mines, Nantes, France

Abstract

In this paper we consider the Discrete Lotsizing and Scheduling Problem with sequence dependent set-up costs and set-up times (DLSPSD). DLSPSD contains elements from lotsizing and from job scheduling, and is known to be NP-Hard. An exact solution procedure for DLSPSD is developed, based on a transformation of DLSPSD into a Travelling Salesman Problem with Time Windows (TSPTW). TSPTW is solved by a novel dynamic programming approach due to Dumas et al. (1993). The results of a computational study show that the algorithm is the first one capable of solving DLSPSD problems of moderate size to optimality with a reasonable computational effort.

Keywords: Lotsizing, Sequencing, Travelling Salesman Problem with Time Windows, Dynamic Programming.

1 Introduction

The Discrete Lotsizing and Scheduling Problem (DLSP) is the problem of determining a minimal cost production schedule, such that machine capacity restrictions are not violated, and demand for all products is satisfied without backlogging. Here we consider the single machine variant of this problem with sequence dependent set-up costs, sequence dependent set-up times, and inventory holding costs. In the sequel we denote the variant of DLSP with sequence dependencies by DLSPSD. Mathematically, DLSPSD is formulated as,

DLSPSD

$$Z_{DLSPSD} = \min \sum_{i=1}^{N} \sum_{t=1}^{T} \left(\sum_{j=0}^{N} S_{j,i} w_{j,i,t} + h_i I_{i,t} \right) \tag{1}$$

subject to

$$I_{i,t-1} + y_{i,t} - d_{i,t} = I_{i,t} \qquad i = 1, \ldots, N; t = 1, \ldots, T \tag{2}$$

$$\sum_{\tau=1}^{t} y_{i,\tau} \geq D_{i,t} \qquad i = 1, \ldots, N; t = 1, \ldots, T-1 \tag{3a}$$

$$\sum_{\tau=1}^{T} y_{i,\tau} = D_{i,T} \qquad i = 1,\ldots,N \tag{3b}$$

$$y_{j,\tau} \leq 1 - y_{i,t} \qquad \begin{cases} i = 1,\ldots,N;\ j = 0,\ldots,N;\ i \neq j; \\ a_{j,i} > 0;\ \tau = t - a_{j,i},\ldots,t-1; \\ t = 1,\ldots,T \end{cases} \tag{4}$$

$$v_{j,i,\tau} \geq y_{i,t} + y_{j,t-a_{j,i}-1} - 1 \qquad \begin{cases} i = 1,\ldots,N;\ j = 0,\ldots,N;\ i \neq j; \\ a_{j,i} > 0;\ \tau = t - a_{j,i},\ldots,t-1 \\ t = 1,\ldots,T \end{cases} \tag{5}$$

$$w_{j,i,t} \geq v_{j,i,t} - v_{j,i,t-1} \qquad \begin{cases} i = 1,\ldots,N;\ j = 0,\ldots,N;\ i \neq j; \\ a_{j,i} > 0;\ t = 1,\ldots,T \end{cases} \tag{6a}$$

$$w_{j,i,t} \geq y_{i,t} + y_{j,t-1} - 1 \qquad \begin{cases} i = 1,\ldots,N;\ j = 0,\ldots,N;\ i \neq j; \\ a_{j,i} = 0;\ t = 1,\ldots,T \end{cases} \tag{6b}$$

$$y_{j,t-1} \geq w_{j,i,t} \qquad \begin{cases} i = 1,\ldots,N;\ j = 0,\ldots,N;\ i \neq j; \\ t = 1,\ldots,T \end{cases} \tag{7}$$

$$y_{i,t+a_{j,i}} \geq w_{j,i,t} \qquad \begin{cases} i = 1,\ldots,N;\ j = 0,\ldots,N;\ i \neq j; \\ t = 1,\ldots,T \end{cases} \tag{8}$$

$$\sum_{i=0}^{N} y_{i,t} + \sum_{\{i,j|a_{j,i}>0\}} v_{j,i,t} = 1 \qquad t = 1,\ldots,T \tag{9}$$

$$y_{i,t} \in \{0,1\} \qquad i = 0,\ldots,N;\ t = 1,\ldots,T \tag{10}$$

$$v_{j,i,t} \in \{0,1\} \qquad \begin{cases} i = 1,\ldots,N;\ j = 0,\ldots,N;\ i \neq j; \\ a_{j,i} > 0;\ t = 1,\ldots,T \end{cases} \tag{11}$$

$$w_{j,i,t} \in \{0,1\} \qquad \begin{cases} i = 1,\ldots,N;\ j = 0,\ldots,N;\ i \neq j; \\ t = 1,\ldots,T \end{cases} \tag{12}$$

In this model the decision variables y indicate production, i.e., when product i $(= 0,\ldots,N)$ is produced in period t $(= 1,\ldots,T)$, the decision variable $y_{i,t}$ equals one, and $y_{i,t}$ equals zero otherwise. Here, 'product' $i = 0$ is an artificial product, indicating that the machine is idle in period t. The decision variables I represent the on-hand inventory, i.e., $I_{i,t}$ is the on-hand inventory of product i at the end of period t.
Set-up time is represented by the decision variables v. If the machine produces product i while it previously produced product j, the decision variables $v_{j,i,\tau}$ equal one during each time period τ used for the set-up. In the first period τ of a set-up the decision variable $w_{j,i,\tau}$ equals one. Here, the first period of a set-up is defined as the period in which set-up time starts if $a_{j,i} > 0$, and it is defined as the first period in which production starts if $a_{j,i} = 0$.

The objective, minimizing the sum of set-up costs and inventory holding costs, is expressed by

(1). To explain the set-up cost structure, we first define a *production batch* of product i as an uninterrupted sequence of periods in which production takes place for product i. Between two subsequent production batches of product j and product i, set-up costs $S_{j,i}$ are incurred in the first period of a set-up. Holding costs for product i are h_i per item per period that the item is kept in inventory. The inventory positions are defined by constraints (2). Here, $d_{i,t}$ is the demand for product i in period t. Without loss of generality we assume throughout this paper $d_{i,t} \in \{0,1\}$ (see e.g. Salomon et al. 1991). Constraints (3a) guarantee that for each product i and for each period $t = 1,\ldots,T-1$ the cumulative production is no less than the cumulative demand $D_{i,t} = \sum_{\tau=1}^{t} d_{i,\tau}$. Similarly, constraints (3b) guarantee that cumulative production equals cumulative demand in the last planning period. As a consequence of constraints (3a) and (3b) the end-of-period inventory position is non-negative in each period of the planning horizon, i.e. $I_{i,t} \geq 0$ for $t = 1,\ldots,T$.
Constraints (4) ensure that if product i is produced in period t, no other product j with set-up time $a_{j,i} > 0$ can be produced in periods $[t - a_{j,i}, t-1]$, since these periods need to be reserved either for production of product i or for set-up time. In this way it is implicitly assumed that the triangle inequality holds with respect to the set-up times, i.e., $a_{j,i} \leq a_{j,k} + a_{k,i}$. Constraints (5) force that if production takes place for product i in period t and for product j in period $t - a_{j,i} - 1$, then periods $[t - a_{j,i}, t-1]$ need to be reserved for set-up time if $a_{j,i} > 0$. Constraints (6a) force $w_{j,i,t} = 1$ if the first period of a set-up between product j and product i is period t ($a_{j,i} > 0$). Similarly, for the case of zero set-up time constraints (6b) force $w_{j,i,t} = 1$ if the first production period is t. If $w_{j,i,t} = 1$ constraints (7) ensure that product j is produced in period $t-1$, whereas constraints (8) ensure that product i is produced in period $t + a_{j,i}$. The restriction that the machine can never be in production and/or set-up for more than one product at the same time is represented by (9). The binary character of the production and set-up variables is represented by (10), (11), and (12).

Lemma 1 *Formulation DLSPSD forces that,*

(A) *variable $v_{j,i,t} = 1$ if and only if there exist at least one period $\tau \in [t - a_{j,i}, t-1]$ for which $y_{j,\tau} = 1$ and $y_{i,\tau+a_{j,i}+1} = 1$,*

(B) *variable $w_{j,i,t} = 1$ if and only if,*

- *period t is the first set-up period required to change-over production from product j to product i ($a_{j,i} > 0$),*
- *period t is the first production period after a production change-over from product j to product i ($a_{j,i} = 0$).*

(C) *if period t_1 is the last production period of a batch for product j, and period t_2 is the first production period of a production batch for product i, and the two batches are immediate successors, then $t_2 = t_1 + a_{j,i} + 1$,*

Proof.

(A) The 'if' part follows immediately from (5). To proof the 'only if' part, let u be the largest numbered period smaller then or equal to t for which $v_{j,i,u} = 1$ and $v_{j,i,u-1} = 0$. Note that such period u must always exist. Constraint (6a) now implies that $w_{j,i,u} = 1$, whereas (7) and (8) force $y_{j,u-1} = 1$ and $y_{i,u+a_{j,i}} = 1$. Suppose, ad absurdum, $u \leq t - a_{j,i}$. Then

$v_{j,i,u+a_{j,i}} = 1$ while at the same time $y_{i,u+a_{j,i}} = 1$, which is impossible due to constraints (9). Thus, $u > t - a_{j,i}$. By taking $\tau = u - 1$ it is forced that $y_{i,\tau} = 1$ and $y_{j,\tau+a_{j,i}+1} = 1$ for at least one period $\tau \in [t - a_{j,i}, t - 1]$.

(B) The 'if' part follows from (6a) in case of non-zero set-up time and from (6b) in case of zero set-up time. To proof the 'only if' part for the case $a_{j,i} > 0$, we show that the situation in which $w_{j,i,t} = 1$ and $v_{j,i,t} - v_{j,i,t-1} \leq 0$ can not occur. If $w_{j,i,t} = 1$, then (7) forces $y_{j,t-1} = 1$ and (8) forces $y_{i,t+a_{j,i}} = 1$. Constraint (5) forces $v_{j,i,t} = 1$ and (9) forces $v_{j,i,t-1} = 0$. Consequently, when $w_{j,i,t} = 1$ also $v_{j,i,t} - v_{j,i,t-1} = 1$, which implies that period t is the first period of the change-over between product j and product i.
To proof the 'only if' part in case $a_{j,i} = 0$, let $w_{j,i,t} = 1$. Constraints (7) and (8) force $y_{j,t-1} = 1$ and $y_{i,t} = 1$. Thus, period t is the first production period for product i after the change-over from product j.

(C) The situation that $t_2 < t_1 + a_{j,i} + 1$ can not occur because of constraints (4). The situation that $t_2 > t_1 + a_{j,i} + 1$ can not occur, since, if product j and product i are produced in subsequent batches, periods in the interval $[t_1 + 1, t_2 - 1]$ exist for which (A) is violated. □

Remark 1. Note that in DLSPSD the initial machine states for product i can be indicated by the predetermined variables $y_{i,\tau}$ and $v_{j,i,\tau}$ for $\tau \leq 0$.

Remark 2. For ease of explanation we assume, without loss of generality, zero set-up time to switch the machine from the production status to the idleness status, i.e., $a_{j,0} = 0$ for all $j = 1, \ldots, N$.

DLSP has many important practical applications. An early example is the application of DLSP in an automated production scheduling system for a tire company (Lasdon and Terjung, 1971). Another example is reported by Van Wassenhove and Vanderhenst (1983), who describe the application of DLSP in a decision support system for production planning in a chemical plant. Fleischmann and Popp (1989) apply DLSP to model and solve a production planning problem in the food industry. Although the above problem settings include both sequence dependent set-up costs and sequence dependent set-up times, the authors choose to ignore them in their model formulations, due to the lack of computationally efficient solution procedures that deal with this type of problems (except Fleischmann and Popp, who were the first to consider sequence dependent set-up *costs*).

Recently, many papers have examined theoretical and computational aspects of DLSP. Salomon et al. (1991) show that the single machine multi-product case without set-up times is NP-Hard, and that the problem of finding a feasible solution in the presence of sequence independent set-up times is already NP-Complete. They also prove that finding a feasible solution to the parallel machine DLSP is NP-Complete when machines have non-identical production speeds. Van Hoesel et al. (1994) present efficient solution procedures for the single product case, based on dynamic programming and polyhedral techniques. Fleischmann (1990), and Magnanti and Vachani (1990) propose effective exact solution procedures for the multi-product case with sequence independent set-up costs and zero set-up times. Cattrysse et al. (1993) develop heuristic procedures for DLSP with sequence independent set-up costs and set-up times. Jordan and Drexl (1994) propose an exact enumeration based procedure for the latter problem, which can

also be applied to solve problems with sequence-dependent set-up times and set-up costs, as long as inventory holding costs are equal for all products. Fleischmann and Popp (1989) and Fleischmann (1994) consider the problem with sequence dependent set-up costs and zero set-up times. Fleischmann formulates it as a Travelling Salesman Problem with Time Windows (TSPTW). Problems of moderate size are solved using simple local improvement based heuristics. Lower bounds to evaluate the quality of the solutions obtained by the heuristics are generated by Lagrangean relaxation procedures. The computational study in Fleischmann shows that in some cases the gap between lower and upper bounds may be as large as 30%. Therefore, Fleischmann concludes his paper with the following statement: *'Important tasks of further research on the DLSP are the determination of exact solution procedures for sample problems, e.g. by means of branching, in order to give a more profound evaluation of the lower bounds, and the development of faster and better heuristics'.*

The contribution of this paper is twofold. First, we generalize Fleischmann's reformulation of DLSP as TSPTW to problems with sequence dependent set-up times. Second, using a specialized algorithm for TSPTW we are the first to solve DLSPSD instances of moderate size to proven optimality with a reasonable computational effort.

This paper is organized as follows. In Section 2 the reformulation of DLSPSD as TSPTW is discussed. Section 3 sketches the exact solution procedure for TSPTW that we apply to DLSPSD. In Section 4 the results of a computational study are discussed, and conclusions are presented in Section 5.

2 A reformulation of DLSPSD as TSPTW

Our reformulation of DLSPSD as TSPTW is inspired by Fleischmann (1994). However, as already mentioned, our reformulation allows for the non-trivial extension of sequence dependent set-up times in addition to sequence dependent set-up costs. A formal description of the reformulation is found in Section 2.1 below, whereas a numerical example is given in Section 2.2.

2.1 Formal description of the reformulation

The TSPTW graph $\mathcal{G}$ that we construct consists of the following attributes:

- the *nodes* represent the demand occurrences for each regular product ($i = 1, \ldots, N$) in periods ($t = 1, \ldots, T$), as well as the 'demand' for idle periods ($i = 0$),
- the *time-windows* on the nodes ensure that no demand backlog occurs,
- the *costs* related to the nodes represent the inventory holding costs,
- the *arcs* represent feasible transitions between demand and idleness periods. The *arc costs* represent the costs of the state transitions. The *arc travel times* represent production and set-up times related to the state transitions.

Without loss of generality we assume in the remainder of this paper that the initial machine status (in period 0) is *idle*. Under this assumption the precise construction of the graph $\mathcal{G}$ is as follows:

- *Nodes*: A node labelled START represents the (idle) machine status in period 0. The set of nodes $\mathcal{N}_1$ consists of all product-demand period combinations $(i, t_i^{(k)})$ $(i = 1, \ldots, N,$ $k = 1, \ldots, k_i^{tot})$. Here, $t_i^{(k)}$ denotes the k-th demand period of product i. Since $d_{i,t} \in \{0, 1\}$ it follows that $t_i^{(k)} = \min\{\tau | D_{i,\tau} = k\}$. Furthermore, k_i^{tot} is defined as the *total* number of demand periods for product i, i.e. $k_i^{tot} = D_{i,T}$. The set $\mathcal{N}_2 = \{O^{(1)}, \ldots, O^{(D_0)}\}$ consists of nodes representing *idle* periods (i.e. periods in which no set-up and no production is scheduled). Here, node $O^{(p)}$ corresponds to the p-th idle period, and D_0 is an upper bound on the number of idle periods. This upper bound is computed by subtracting the total demand $(\sum_{i=1}^{N} D_{i,T})$ and the total set-up time from the available capacity (T). However, since the total set-up time is not known in advance, we *under*-estimate the total set-up time by $\sum_{i=1}^{N} \Delta_{i,T}$, where $\Delta_{i,T}$ is defined as a lower bound on the cumulative set-up time for product i up to period T. A procedure to calculate the lower bound $\Delta_{i,T}$ will be outlined below. More formally, the upper bound D_0 on the number of idle periods is computed as,

$$D_0 = T - \sum_{i=1}^{N} (D_{i,T} + \Delta_{i,T}) \tag{13}$$

For ease of notation we further define $\mathcal{N} = \{START\} \bigcup \mathcal{N}_1 \bigcup \mathcal{N}_2$. Note that any feasible DLSP instance must satisfy $|\mathcal{N}| \leq T + 1$.

- *Time windows on nodes:* In the sequel we denote the time windows on the nodes corresponding to the k-th demand period of product i by $[lb_i^{(k)}, ub_i^{(k)}]$, where $lb_i^{(k)}$ is the lower bound on the arrival time at the node (i.e. the earliest period by which production of the k-th demand period may be finished), and $ub_i^{(k)}$ is the upper bound on the arrival time at the node (i.e. the latest period by which production of the k-th demand period must be finished). To calculate $lb_i^{(k)}$ we define for each (i, t) combination a constant $C_{i,t}$. Constants $C_{i,t}$ can be viewed as the *maximum* number of periods that remains for production of product i in periods $1, \ldots, t$, after accounting for demands $D_{j,t}$ of products $j \neq i$, as well as for the minimum required set-up time up to period t, denoted by $\sum_j \Delta_{j,t}$. Constants $C_{i,t}$ are obtained from the following backward recursion:

$$C_{i,T} = \begin{cases} D_0, & i = 0 \\ T - \sum_{\substack{j \neq i \\ j \neq 0}} D_{j,T} - \sum_j \Delta_{j,T} & i = 1, \ldots, N \end{cases} \tag{14}$$

and for $i = 0, \ldots, N$, $t = T - 1, \ldots, 1$:

$$C_{i,t} = \min \left(C_{i,t+1}, t - \sum_{\substack{j \neq i \\ j \neq 0}} D_{j,t} - \sum_j \Delta_{j,t} \right) \tag{15}$$

The minimum required set-up time $\Delta_{i,t}$ is calculated by the procedure presented below. In explaining the procedure, we use the following notation: $n_i^{(k)}$ is a variable which is set to one if an *additional* set-up is *necessary* to produce the k-th demand period of product

i, and $n_i^{(k)}$ is set to zero otherwise. Furthermore, a_i is defined as the minimum required set-up time for product i, i.e., $a_i = \min_{j \neq i}\{a_{j,i}\}$. Note that, if $a_i = 0$, then $\Delta_{i,t} = 0$ for all periods t.

Procedure to calculate $\Delta_{i,t}$ ($a_i > 0$):

Initialization: In the initialization step the lower bounds $lb_i^{(k)}$ and upper bounds $ub_i^{(k)}$ on all time windows are initialized, by planning for each product $i = 1, \ldots, N$ one single set-up to produce the demand of the *first* demand period, i.e,

$$n_i^{(k)} = \begin{cases} 1 & \text{if } k = 1 \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, given the set-up for the first demand period, the cumulative number of set-up periods for product $i = 1, \ldots, N$ up to period $t = 1, \ldots, T$ is initialized as,

$$\Delta_{i,t} = \begin{cases} 0 & t < t_i^{(1)} - a_i \\ \alpha + 1 & t = t_i^{(1)} - a_i + \alpha, \text{ and } \alpha = 0, \ldots, a_i - 1 \\ a_i & t \geq t_i^{(1)} \end{cases}$$

The upper bound $ub_i^{(k)}$ is set equal to the period in which the demand occurs, i.e., $ub_i^{(k)} = t_i^{(k)}$ for $i = 1, \ldots, N$ and $k = 1, \ldots, k_i^{tot}$. Goto Step 2 to update constants $C_{i,t}$.

Step 1: In this step *additional* set-ups may be scheduled, based on the argument that if two *subsequent* demand periods of the same product cannot be produced in the same production batch, then an additional production batch must be created, resulting in additional set-up time. Note that two subsequent demand periods of product i (say, $t_i^{(k)}$ and $t_i^{(k+1)}$) cannot be produced in the same production batch if the time windows of the two demand periods are *non-overlapping*. Here, time-windows related to the k-th and $k+1$-th demand period are defined to be non-overlapping if the condition,

$$ub_i^{(k)} < lb_i^{(k+1)} - 1 \tag{16}$$

is satisfied. If no set-up has been planned for the $k+1$-th demand period, i.e., when $n_i^{(k+1)} = 0$, an additional set-up needs to be scheduled for the $k+1$-th demand period. The latter is accomplished by putting $n_i^{(k+1)} := 1$, and by updating $\Delta_{i,t}$ as follows,

$$\Delta_{i,t} := \begin{cases} \Delta_{i,t} & t < t_i^{(k+1)} - a_i \\ \Delta_{i,t} + \alpha + 1 & t = t_i^{(k+1)} - a_i + \alpha, \text{ and } \alpha = 0, \ldots, a_i - 1 \\ \Delta_{i,t} + a_i & t \geq t_i^{(k+1)} \end{cases} . \tag{17}$$

To search for additional set-ups the following procedure has been implemented:

additional_set-ups_found := false;
For each product $i = 1, ..., N$ do
{
  $k := 1$;
  While $(k < k_i^{tot})$ do
  {
    If $(n_i^{(k+1)} = 0)$ and ((16) is satisfied) Then
    {
      additional_set-ups_found := true;
      $n_i^{(k+1)} = 1$;
      Update Δ using (17);
    }
    $k := k + 1$;
  }
}
If (additional_set-ups_found = false) Then STOP Else Goto Step 2,

Step 2: Apply (13), (14), and (15) to (re-)calculate $C_{i,t}$ for $i = 0, \ldots, N$, and $t = 1, \ldots, T$. Goto Step 3,

Step 3: (Re-)calculate the lower bounds $lb_i^{(k)} = \min\{\tau | C_{i,\tau} = D_{i,t_i^{(k)}}\}$ for $i = 1, \ldots, N$ and $k = 1, \ldots, k_i^{tot}$ (see Lemma 3). Goto Step 1.

Remark 3. Note that once the upper bounds on the time windows have been calculated in the initialization step, they remain unchanged during the iterative procedure.

Remark 4. Note that the lower bounds on the time-windows can even be tightened by taking into account the set-up time related to a change-over from the initial machine state, i.e., the product that has been produced before the first planning period ($t \leq 0$), to the first product that will be produced in periods $t \geq 1$.

Lemma 2 *If set-up times are present, then for each product $i = 1, \ldots, N$ the condition $D_{i,t} \leq C_{i,t}$, $t = 1, \ldots, T$ is necessary for problem feasibility. If set-up times are zero the condition is necessary and sufficient.*

Proof. If set-up times are zero, the proof of this Lemma is found in Fleischmann (1994). If set-up times are non-zero, the expression $\sum_{i=1}^{N} (D_{i,t} + \Delta_{i,t})$ is a *lower bound* on the cumulative required capacity up to period t, since $\Delta_{i,t}$ is a lower bound on the total required set-up time for product i up to period t. Thus, in case of non-zero set-up times, the condition $D_{i,t} \leq C_{i,t}$ for all t, is necessary for problem feasibility. Since the actual cumulative set-up time for product i up to period t can be larger than $\Delta_{i,t}$, the condition is *not* sufficient in this case. □

Lemma 3 *There exists no feasible solution in which* $lb_i^{(k)} < \min\{\tau | C_{i,\tau} = D_{i,t_i^{(k)}}\}$ *or* $ub_i^{(k)} > t_i^{(k)}$.

Proof. Note that $lb_i^{(k)}$ is the first period in which sufficient capacity may exist to produce demand for period $t_i^{(k)}$, i.e. $D_{i,t_i^{(k)}} \leq C_{i,\tau}$ for $\tau = lb_i^{(k)}, \ldots, T$. The availability of sufficient capacity is a necessary condition for problem feasibility in case of non-zero set-up times, and a necessary and sufficient condition for problem feasibility in case of zero set-up times (Lemma 2). Of course, since backlogging is not allowed, the latest production period for the k-th demand period is period $t_i^{(k)}$. □.

Remark 5. Note that the Lemma's 2–3 imply that when set-up times are zero, problem feasibility is guaranteed by the construction of the time windows. However, in case of nonzero set-up times the construction of the time windows cannot guarantee problem feasibility.

Based on Lemma 3 time windows on the nodes in $\mathcal{G}$ can be calculated according to Table 1.

Table 1. Time windows on nodes.

instance type	$\{START\}$	$(i, t_i^{(k)}) \in \mathcal{N}_1$	$O^{(p)} \in \mathcal{N}_2$
with set-up times	$[T+1, T+D_0+1]$	$[\min\{\tau \mid C_{i,\tau} = D_{i,t_i^{(k)}}\}, t_i^{(k)}]$	$[\min\{\tau \mid C_{0,\tau} = p\}, T+p]$
without set-up times	$[T+1, T+1]$	(see Lemma 3)	$[\min\{\tau \mid C_{0,\tau} = p\}, T+p-D_0]$

From Table 1 it can be observed that for problem instances without set-up times the START node, once it has been left in period $t = 0$, will be revisited in period $T+1$ in any TSPTW tour corresponding to a feasible DLSPSD solution. Furthermore, in any feasible solution to a DLSPSD instance with set-up times the START node must be revisited at or before period $T + D_0 + 1$. Note that the latter implies that feasible solutions may exist in which some of the nodes in $\mathcal{N}_2$ are visited *after* the end of the planning horizon, i.e., after period T.

- *Node costs*: In Table 2 we summarize the costs of arriving at the nodes 'too early'. These costs correspond to the inventory holding costs in the original problem. We assume that the arrival time at a given node is u, and that u lies in the time window.

Table 2. Costs at nodes.

	$\{START\}$	$(i, t_i^{(k)}) \in \mathcal{N}_1$	$O^{(p)} \in \mathcal{N}_2$
$u \leq T$	–	$h_i(t_i^{(k)} - u)$	0
$u > T$	0	–	0

Contrary to our approach, where inventory holding costs are directly taken into account, Fleischmann (1994) eliminates inventory holding costs from the model formulation by

expressing them in terms of time dependent production costs. This leads to a TSPTW with time dependent costs related to the arcs. Unfortunately, this type of TSPTW cannot be handled by our exact algorithm.

- *Arcs*: Feasible state transitions are represented by arcs. Machine set-up costs are represented by the costs of state transitions in $\mathcal{G}$. These costs are listed in Table 3 below (starting nodes correspond to column entries, ending nodes correspond to row entries). Infeasible state transitions are indicated by infinite costs.

Table 3. Arc costs.

	From START	From $(i, t_i^{(\ell)}) \in \mathcal{N}_1$	From $O^{(p)} \in \mathcal{N}_2$
To START	∞	0 if $\ell = k_i^{tot}$ ∞ otherwise	0 if $p = D_0$ ∞ otherwise
To $(j, t_j^{(m)}) \in \mathcal{N}_1$	$S_{0,j}$ if $m = 1$ ∞ otherwise	$S_{i,j}$ if $i \neq j$ 0 if $i = j$ and $m = \ell + 1$ ∞ otherwise	$S_{0,j}$
To $O^{(q)} \in \mathcal{N}_2$	0 if $q = 1$ ∞ otherwise	0	0 if $q = p + 1$ ∞ otherwise

Production and set-up times are represented by the travel times (lengths) of the arcs in $\mathcal{G}$. The travel times are listed in Table 4 below. Infinite state transitions are indicated by infinite travel times.

Table 4. Arc travel times.

	From $\{START\}$	From $(i, t_i^{(\ell)}) \in \mathcal{N}_1$	From $O^{(p)} \in \mathcal{N}_2$
To $\{START\}$	∞	1 if $\ell = k_i^{tot}$ ∞ otherwise	1 if $p = D_0$ ∞ otherwise
To $(j, t_j^{(m)}) \in \mathcal{N}_1$	$a_{0,j} + 1$ if $m = 1$ ∞ otherwise	$a_{i,j} + 1$ if $i \neq j$ 1 if $i = j$ and $m = \ell + 1$ ∞ otherwise	$a_{0,j} + 1$
To $O^{(q)} \in \mathcal{N}_2$	1 if $q = 1$ ∞ otherwise	1	1 if $q = p + 1$ ∞ otherwise

In the sequel we denote the set of arcs by $\mathcal{A}$. Arcs may depart from the $\{START\}$ node to at most $N + 1$ other nodes, representing the first demand occurrences $t_i^{(1)}$ $(i = 1, \ldots, N)$ or the first idleness occurrence $O^{(1)}$. Furthermore, each node representing a demand or idleness occurrence may have an arc to another node representing a demand or idleness occurrence. Finally, the nodes corresponding to the last demand occurrences $t_i^{(k_i^{tot})}$ and node $O^{(D_0)}$ are connected to the $\{START\}$ node, resulting in $N + 1$ arcs. Since the number of demand plus idleness occurrences is bounded by T, it follows that $|\mathcal{A}| \leq (N + 1) + T(T - 1) + (N + 1) = T^2 - T + 2(N + 1)$. However, since infeasible state transitions (i.e. state transitions with infinite costs, and state transitions that are eliminated in the pre-processing step of the algorithm presented in Section 3) need not to be represented by arcs, the actual number of arcs in $\mathcal{A}$ will in practice be much lower than this upper bound.

2.2 An example of the reformulation

Because of non-triviality of the reformulation, a small numerical example is presented here. We consider a two product ($N = 2$) instance of DLSPSD with ten time periods ($T = 10$), where the initial machine state is idle. The demand are specified in Table 5.

Table 5. Item-period demand $d_{i,t}$

	period									
product	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	1	1	0	0	1
2	0	0	0	0	0	0	1	0	0	0

From Table 5 we read that $t_1^{(1)} = 6$, $t_1^{(2)} = 7$, $t_1^{(3)} = 10$, and $t_2^{(1)} = 7$. Set-up times $a_{j,i}$ are listed in Table 6. Note that the set-up time to change-over from product 1 to product 2 and from product 2 to product 1 via an idle period do not satisfy the triangle inequality.

Table 6. Set-up times $a_{j,i}$.

	From product 0	From product 1	From product 2
To product 0	-	0	0
To product 1	1	-	2
To product 2	2	3	-

From the table above we obtain for instance that $a_{0,1} = 1$ and $a_{1,0} = 0$. Furthermore, $a_1 = 1$ and $a_2 = 2$. Applying the procedure to calculate $\Delta_{i,t}$, we obtain after *one* iteration the results specified in Table 7.

Table 7. Cumulative set-up time $\Delta_{i,t}$

	period									
product	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	1	1	1	1	1	1
2	0	0	0	0	1	2	2	2	2	2

The number of artificial nodes D_0 then equals $T - \sum_{i=1}^{N} (D_{i,T} + \Delta_{i,T}) = 10 - 3 - 1 - 1 - 2 = 3$. From this the graph $\mathcal{G}$ is now constructed. In the graph we have a $\{START\}$-node, while $\mathcal{N}_1 = \{(1,6),(1,7),(1,10),(2,7)\}$ and $\mathcal{N}_2 = \{O^{(1)}, O^{(2)}, O^{(2)}\}$. Table 8 shows the time windows for each of the nodes. Note that at calculation of the time-windows the initial machine state has been taken into account.

Table 8. Time windows.

node	$\{START\}$	(1,6)	(1,7)	(1,10)	(2,7)	$O^{(1)}$	$O^{(2)}$	$O^{(3)}$
time window	[11,14]	[2,6]	[3,7]	[4,10]	[3,7]	[1,11]	[8,12]	[9,13]

Furthermore, *'travel times'* between nodes are given by Table 9 (the columns correspond to starting nodes, and the rows correspond to ending nodes).

Remark 6. Entries labelled with '(*)' in Tables 9 and 11 will be eliminated from $\mathcal{A}$ in the pre-processing phase of the DP-algorithm (see Section 3).

Table 9. Arc travel times.

node	From $\{START\}$	From (1,6)	From (1,7)	From (1,10)	From (2,7)	From $O^{(1)}$	From $O^{(2)}$	From $O^{(3)}$
To (1,6)	2	-	-	-	3	2	2 (*)	2 (*)
To (1,7)	-	1	-	-	3	2	2 (*)	2 (*)
To (1,10)	-	-	1	-	3	2	2 (*)	2 (*)
To (2,7)	3	4	4	4	-	3	3 (*)	3 (*)
To $O^{(1)}$	1	1	1	1	1	-	-	-
To $O^{(2)}$	-	1 (*)	1	1	1	1	-	-
To $O^{(3)}$	-	1 (*)	1 (*)	1	1 (*)	-	1	-
To $\{START\}$	-	-	-	1	1	-	-	1

Inventory holding costs are $h_1 = 1$ and $h_2 = 2$ per unit end-of-period stock. Set-up costs are listed in Table 10.

Table 10. Set-up costs $S_{j,i}$.

	From 'product' 0	From product 1	From product 2
To 'product' 0	-	0	0
To product 1	10	-	15
To product 2	15	20	-

From the above table we obtain e.g., $S_{0,1} = 10$ and $S_{1,0} = 0$. The associated *'arc-travel costs'* are given in Table 11.

Table 11. Arc travel costs.

node	From $\{START\}$	From (1,6)	From (1,7)	From (1,10)	From (2,7)	From $O^{(1)}$	From $O^{(2)}$	From $O^{(3)}$
To (1,6)	10	-	-	-	15	10	10 (*)	10 (*)
To (1,7)	-	0	-	-	15	10	10 (*)	10 (*)
To (1,10)	-	-	0	-	15	10	10 (*)	10 (*)
To (2,7)	15	20	20	20	-	15	15 (*)	15 (*)
To $O^{(1)}$	0	0	0	0	0	-	-	-
To $O^{(2)}$	-	0 (*)	0	0	0	0	-	-
To $O^{(3)}$	-	0 (*)	0 (*)	0	0 (*)	0	-	-
To $\{START\}$	-	-	-	0	0	-	-	0

Table 12 shows three feasible production schedules in the *original* network. In Table 12 the following notation is used:

I = idle
Si = in set-up for item i
Pi = in production for item i

Table 12. Three feasible production schedules

	period														
	(0)	1	2	3	4	5	6	7	8	9	10	(11)	(12)	(13)	(14)
Schedule 1	START	S1	P1	P1	S2	S2	S2	P2	S1	S1	P1	I	I	I	START
Schedule 2	START	S2	S2	P2	S1	S1	P1	P1	P1	I	I	I	START	-	-
Schedule 3	START	S2	S2	P2	S1	S1	P1	P1	I	S1	P1	I	I	START	-

Schedule 1 in Table 12 corresponds to the following path through the original network: $\{START\}$, (1,6) . (1,7), (2,7), (1,10), $O^{(1)}$, $O^{(2)}$, $O^{(3)}$, $\{START\}$. The set-up costs are $S_{0,1} + S_{1,2} + S_{2,1} = 10 + 20 + 15 = 45$. Total holding costs for this plan are obtained from Table 13. Note that in Schedule 1,2 and 3 the $\{START\}$ node is reached in periods 14, 12, and 13 respectively. Note further that for each of the three schedules the triangle inequality with respect to set-up times is satisfied (i.e., no change-over from product 1 to product 2 or product 2 to product 1 via an idle period occurs). Hence, the schedules can be represented by the mathematical formulation of DLSPSD (See Section 1).

Table 13. Inventory holding costs in Schedule 1.

node	*arrival time*	*earliness*	$h_i \times$ *earliness*
$\{START\}$	0	0	0
(1,6)	2	$6 - 2 = 4$	$1 \times 4 = 4$
(1,7)	3	4	4
(2,7)	7	0	0
(1,10)	10	0	0
$O^{(1)}$	11	0	0
$O^{(2)}$	12	0	0
$O^{(3)}$	13	0	0
$\{START\}$	14	0	0

Inventory holding costs of Schedule 1 are $4 + 4 = 8$, and the total costs of Schedule 1 are $45 + 8 = 53$.

Table 14 shows the representation of Schedule 1 in terms of DLSPSD variables.

Table 14. Non-zero variables in the DLSPSD representation of Schedule 1.

$t = 0$	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$	$t = 6$	$t = 7$	$t = 8$	$t = 9$	$t = 10$
	$v_{0,1,1} = 1$			$v_{1,2,4} = 1$	$v_{1,2,5} = 1$	$v_{1,2,6} = 1$		$v_{2,1,8} = 1$	$v_{2,1,9} = 1$	
	$w_{0,1,1} = 1$			$w_{1,2,4} = 1$				$w_{2,1,8} = 1$		
$y_{0,0} = 1$		$y_{1,2} = 1$	$y_{1,3} = 1$				$y_{2,7} = 1$			$y_{1,10} = 1$

3 An effective algorithm for TSPTW

To solve DLSPSD as TSPTW we apply the algorithm of Dumas et al. (1995). This forward Dynamic Programming (DP) algorithm is the first algorithm reported in the literature that is able to solve medium size difficult TSPTW instances (with fairly wide and overlapping time windows) to optimality. The strength of the method stems from using the time windows and cost structure to significantly *reduce* the state space and the number of state transitions in the underlying DP network. These reductions are performed during the pre-processing phase and during the execution of the algorithm.

In the pre-processing phase the arc set $\mathcal{A}$ is reduced to $\mathcal{A}'$, applying the rules specific to the TSPTW described in Langevin et al. (1990). For example, consider two demand periods $t_i^{(k)}$ with time window $[lb_i^{(k)}, ub_i^{(k)}]$, and $t_j^{(\ell)}$ with time window $[lb_j^{(\ell)}, ub_j^{(\ell)}]$. The arc between the nodes $(i, t_i^{(k)})$ and $(j, t_j^{(\ell)})$ is eliminated from $\mathcal{A}$ in the pre-processing phase if production of demand period $t_j^{(\ell)}$ directly after demand period $t_i^{(k)}$ is infeasible due to the time window constraints. The latter will be the case if either of the conditions $ub_i^{(k)} + a_{i,j} + 1 < lb_j^{(\ell)}$ *or* $lb_i^{(k)} + a_{i,j} + 1 > ub_j^{(\ell)}$ is satisfied. In the example of Section 2.2 the arcs that can be eliminated in this way from $\mathcal{A}$ are labelled by '(*)' in Tables 9 and 11.

To provide a cursory description of the DP algorithm itself, we define the triple $(\mathcal{S}, n, t)$ as follows: $\mathcal{S} \subset \mathcal{N}$ is an unordered set of visited nodes, $n \in \mathcal{S}$ is the last visited node, and t is the arrival time at node n. Then, we define $F(\mathcal{S}, n, t)$ as the least cost of a path starting at node $\{START\}$, passing through every node of $\mathcal{S} \subset \mathcal{N} \setminus \{START\}$ exactly once, and arriving at node n at time t. The least costs $F(\mathcal{S}, n, t)$ are computed by solving the recurrence equations described in Dumas et al. (1995). These equations define a shortest path on a state graph whose nodes are the states $(\mathcal{S}, n, t)$ and whose arcs represent transitions from one state to another. At stage ℓ, $\ell = 1, \ldots, |\mathcal{N}| - 1$, the forward DP algorithm generates a least cost path of length ℓ $(= |\mathcal{S}|)$. Among the paths that visit every node in $\mathcal{S}$ and end at node n a *cost dominance* test is carried out which ensures that dominated elements are discarded from the state graph. Here, dominated is defined as follows: suppose we have the states $(\mathcal{S}, n, t')$ and $(\mathcal{S}, n, t'')$. If $t' \leq t''$ and $F(\mathcal{S}, n, t') \leq F(\mathcal{S}, n, t'')$, then the second state is dominated, and will not be included in the state-graph.

Unfortunately, the dominance test applies to *non*-decreasing cost functions over the time windows only (see Desrosiers et al. 1995). Yet, inventory holding costs are non-increasing functions. To still exploit the power of the dominance tests, we transform the original network into a *mirror* network. The mirror network is constructed by reversing the direction of all arcs in the original network and by transforming all time windows $[lb, ub]$ into $[T^{max} - ub, T^{max} - lb]$. Here, T^{max} is chosen such that in the reversed network all time windows are in the interval $[1, T^{max}]$. To do so, T^{max} is taken one period larger then the upper bound on the time window of the START node in the original network, i.e., $T^{max} = T + D_0 + 2$ if set-up times occur, and $T^{max} = T + 2$ if no set-up times occur. By doing so, arrival at node $(i, t_i^{(k)})$ in period $u \in [T^{max} - ub_i^{(k)}, T^{max} - lb_i^{(k)}]$ yields inventory holding costs $h_i(u - t_i^{(k)})$. Clearly, application of the transformation yields inventory holding costs that are non-decreasing in the arrival time u.

To illustrate the construction of the time windows in the mirror network numerically, we apply the transformation to the example of Section 2.2. Taking $T^{max} = 15$, the results of the transformation are as shown in Table 15.

Table 15. Time windows in original network and in mirror network.

node	$\{START\}$	(1,6)	(1,7)	(1,10)	(2,7)	$O^{(1)}$	$O^{(2)}$	$O^{(3)}$
time windows (original network)	[11,14]	[2,6]	[3,7]	[4,10]	[3,7]	[1,11]	[8,12]	[9,13]
time windows (mirror network)	[1,4]	[9,13]	[8,12]	[5,11]	[8,12]	[4,14]	[3,7]	[2,6]

Furthermore, to speed up the execution of the algorithm, an *elimination* test is carried out. The test is based on the observation that a partial path obtained at stage ℓ must necessarily be

'extendible' to a path which visits all nodes at stage $|\mathcal{N}|$. Such extensions may not be feasible for all partial paths, due to the existence of time window constraints. The test detects whether a node cannot extend a current partial path, thereby permitting the *elimination* of such paths in the state graph. For example, if in a partial path corresponding to the set S node $n = (i, t_i^{(k)})$ is visited at time t (with t in the interval $[lb_i^{(k)}, ub_i^{(k)}]$), then all paths to the nodes $(j, t_j^{(\ell)})$ for which $t + a_{i,j} + 1 < lb_j^{(\ell)}$ or $t + a_{i,j} + 1 > ub_j^{(\ell)}$ can be eliminated from the state graph. If a path to node (S, n, t) cannot be extended to any other node in the state graph, then this state is eliminated from the state graph.

Remark 7. Although the mathematical model formulation (DLSPSD) of Section 1 relies on the assumption that set-up times satisfy the triangle inequality, the solution procedure presented here is also applicable if set-up times do not satisfy this inequality.

4 Computational Experiments

The TSPTW algorithm was coded in C programming language and the computational experiments were conducted on a Hewlett-Packard workstation (HP9000/730, 76 mips, 22 M flops).

The algorithm was tested on problem instances that differ with respect to the following characteristics:

- *Problem dimension*: The problem dimension is represented by the number of products (N), and the number of periods (T),
- *Holding costs*: We have experimented with problem instances *with* and *without* inventory holding costs. In case of non-zero holding costs, for each product i the inventory holding costs have been generated randomly from a discrete uniform $DU(h^{min}, h^{max})$ distribution,
- *Set-up costs*: For each two products i and j sequence dependent set-up costs $S_{i,j}$ have been generated randomly from a discrete uniform $DU(S^{min}, S^{max})$ distribution (see Table 16),
- *Set-up times*: We have experimented with problem instances *with* sequence dependent set-up times, and problem instances *without* set-up times. Instances with sequence dependent set-up times have been generated as follows:

  Step 1: We construct a $N \times N$ matrix. Each matrix entry is generated randomly from a discrete uniform $DU(0, 1)$ distribution,

  Step 2: For each non-zero matrix entry (i, j) we generate the sequence dependent set-up time $a_{i,j}$ from a discrete uniform $DU(a^{min}, a^{max})$ distribution. For zero matrix entries we set the corresponding sequence dependent set-up time equal to zero.

- *Production capacity utilization*: We have experimented with problems with different production capacity utilizations. In our study we define production capacity utilization (ρ) as:

$$\rho = \frac{\sum_{i=1}^{N} D_{i,T}}{T}$$

Note that set-up times are *not* explicitly taken into account in the definition of *production* capacity utilization.

- *Demand pattern*: Demand has been generated according to the following procedure:

  Step 1: For period T we randomly select a product i^* from a discrete uniform $DU(1,N)$ distribution. For product i^* we set $d_{i^*,T} = 1$. By doing so, we ensure that the horizon over which non-zero demand occurs equals the length of the planning horizon T,

  Step 2: For all products i, except for product i^*, we randomly generate *one* period t_i from a discrete uniform $DU(1,T)$ distribution. For period t_i we set $d_{i,t_i} = 1$. By doing so, we ensure that each product has a non-zero cumulative demand, i.e. $D_{i,T} > 0$ for all i,

  Step 3: For each cell in an $N \times T$ matrix, except for the cells corresponding to the (i,t) combinations for which we set $d_{i,t} = 1$ in Steps 1 and 2, we randomly generate a number $\alpha_{i,t}$ from a discrete uniform DU(1,$N \times T$) distribution. Problem instances with prespecified production capacity utilization ρ are now generated by selecting the cells (i,t) containing the $\lceil \rho \times T - N \rceil$ smallest numbers $\alpha_{i,t}$. For those cells we set the corresponding demand $d_{i,t} = 1$. All other demands are set to zero.

  Step 4: If the *rough-cut* capacity check $D_{i,t} \leq C_{i,t}$ is violated for at least one product-period combination (i,t), repeat *Step 3*.

The influence of capacity utilization on algorithmic performance

Since we expected that production capacity utilization may have a major influence on algorithmic performance, we first investigated the relationship between ρ and average CPU-time. To do so, we generated instances with 40 planning periods ($T = 40$) and $N = 3$, $N = 5$, and $N = 10$ products, respectively. The production capacity utilization ρ was varied between 0.25 and 1.00, in steps of 0.05. Holding costs and set-up times were set to *zero*, and set-up costs were randomly generated from a discrete uniform distribution function with $S^{min} = 100$ and $S^{max} = 200$. For each (N,ρ) combination 5 problems were generated, resulting in $3 \times 16 \times 5 = 240$ generated instances. For the problem instances with $N = 3$ and $N = 5$ the relation between ρ and average CPU-time is plotted in Figure 1.[1]

From Figure 1 we conclude that for problem instances with either *low* or *high* production capacity utilization the DP algorithm requires far less CPU-time than for problem instances with *medium* capacity utilization. This is explained as follows: when ρ is low, only a few nodes in $\mathcal{N}_1$ exist. For example, if $T = 40$, and $\rho = 0.25$, there will be on average 10 nodes in $\mathcal{N}_1$ and 30 nodes in $\mathcal{N}_2$. However, since strict precedence relations exist among the nodes in $\mathcal{N}_2$, their impact is not significant. Hence, solving the original 40-period problem is from a computational point of view not much harder than solving a 10-period problem. Furthermore, when ρ is high, the time-windows of the nodes in $\mathcal{N}_1$ are relatively tight, and the number of precedence relations among the nodes in $\mathcal{N}_1$ is relatively large (since all products are produced with high frequency). Both effects significantly increase the effectiveness of the elimination test.

[1] For instances with $N = 10$ the relation between ρ and average CPU-time is not shown in Figure 1, since average CPU-times become too large given the scaling of the figure. However, for $N = 10$ exactly the same effects occur as for $N = 3$ and $N = 5$.

Insert Figure 1 about here

The influence of other problem characteristics on algorithmic performance

To investigate the influence of other problem characteristics than production capacity utilization we have carried out additional experiments involving difficult problem instances with *medium* capacity utilization. In these experiments four problem sets (Set I–IV) have been generated. In each problem set the dimension of the instances was varied over all elements in $\{(N,T)|N = 3,5,10 \text{ and } T = 20,40,60\}$. The production capacity utilization ρ was varied between 0.5 and 0.75, in steps of 0.05. For each (N,T,ρ) combination again 5 instances were generated, resulting in $3 \times 3 \times 6 \times 5 = 270$ problem instances per set. The other specific characteristics of the instances in each set are summarized in Table 16.

Table 16. Characteristics of generated problems.

Set	*Holding costs* $DU(h^{min}, h^{max})$	*Set-up costs* $DU(S^{min}, S^{max})$	*Set-up times* $DU(a^{min}, a^{max})$
Set I	no holding costs	$DU(100,200)$	no set-up times
Set II	no holding costs	$DU(100,200)$	$DU(1,2)$
Set III	$DU(5,10)$	$DU(100,200)$	no set-up times
Set IV	$DU(5,10)$	$DU(100,200)$	$DU(1,2)$

Table 17 shows the relation between problem dimension (N,T), and the number of problems within each set that could be solved to optimality, given a memory limit of 20 Mb and a CPU-time limit of 1200 seconds per instance. Here, all five instances with the same (N,T,ρ) combination have been aggregated over the six different production capacity utilizations ρ, resulting in 30 problem instances per cell.

Table 17. Number of problems that could be solved for each problem dimension.

	(3,20)	(5,20)	(10,20)	(3,40)	(5,40)	(10,40)	(3,60)	(5,60)	(10,60)
Set I	30	30	30	30	30	21	30	30	4
Set II	30	30	30	30	30	21	30	25	4
Set III	30	30	30	30	24	1	30	0	0
Set IV	30	30	30	30	30	1	30	0	0

Insert Figures 2–6 about here

Figures 2–5 show the influence of changes in *problem dimension* on average CPU-time for different values of the production capacity utilizations. We have chosen to compare for each set problem instances with dimension $N = 5$ and $T = 20$ to problem instances in which the number of products and the number of periods has been *doubled*, i.e., problem instances with dimension $(N,T) = (10,20)$, and $(N,T) = (5,40)$. Each point in Figures 2–5 corresponds to the average CPU-time over five problem instances[2].

From Table 17 and Figures 2–5 we observe the following effects,

[2]In Set III the six problem instances in the category $(N,T) = (5,40)$ that could not be solved to optimality given the CPU-time limit of 1200 seconds are not considered in Figures 4 and 6.

- doubling the number of *time periods* results in a much stronger increase in average CPU-time than doubling the number of *products*. This effect is because, for a fixed number of products, the total number of nodes in $\mathcal{G}$ increases on average linearly in T. The linear increase is approximately proportional to ρ for nodes in $\mathcal{N}_1$, and approximately proportional to $(1-\rho)$ for nodes in $\mathcal{N}_2$. However, for a fixed number of periods, the average total number of nodes in $\mathcal{G}$ is *not* influenced by an increase in the number of products. Nevertheless, as can be observed from Figure 1 the average CPU-times still increase, since the average number of *strict* precedence relations among the nodes in $\mathcal{N}_1$ decreases considerably. As a consequence, the time window reduction procedure and the elimination test become less effective. The latter explains the observed effect of increased CPU-times.

- in the interval $[0.5, 0.75]$ an increase in ρ results for some problem instances in an increase in average CPU-time over the entire interval, whereas for the other problem instances the average CPU-time behaves as a concave function in ρ. The latter behaviour was also observed in Figure 1. For those problem instances that do not show this concave behaviour, average CPU-times start to decrease for larger values of ρ ($\rho > 0.75$).

Next, we focus on the influence of *inventory holding costs* and *set-up times* on average CPU-time. Table 17 and Figure 6 show these influences for instances of Sets I–IV with dimension $(N,T) = (5,40)$ for different values of the production capacity utilization.

From this figure we observe the following effects,

- for a given production capacity utilization, problem instances with non-zero inventory holding costs (Sets III–IV) require on average considerably more CPU-time than problem instances with zero inventory holding costs (Sets I–II). This is because, when holding costs are present, the original network has to be transformed into the mirror network. In the original network the time-windows corresponding to early demand periods are on average smaller than time windows corresponding to late demand periods. In the mirror network the opposite occurs: early demand periods tend to have large time windows, and late demand periods have small time windows. In particular, the fact that early demand periods have relatively large time windows causes the elimination test in the forward DP-algorithm to become less effective. This results in an increase of average CPU-times and memory usage[3],

- for a given production capacity utilization, problem instances with non-zero set-up times (Set II and Set IV) require on average less CPU-time than problem instances with zero set-up times (Set I and Set III). There are two reasons why this effect occurs. First, the presence of set-up times reduces the number of idle periods, and thus the number of nodes in $\mathcal{N}_2$. Second, set-up times result in tighter time-windows. When time-windows are tight, the effectiveness of the elimination test increases, leading to a decrease in the number of states evaluated during the execution of the dynamic program.

[3] As can be concluded from the discussion, the increase in CPU-times is not caused by the introduction of inventory holding costs itself, but caused by the necessity to apply the DP-algorithm to the mirror network. If the mirror network is used to solve problems with *zero* inventory holding costs, average CPU-times become even larger than for problems without inventory holding costs, since the *presence* of inventory holding costs *increases* the effectiveness of the cost based dominance tests.

5 Conclusions

In this paper we have examined the Discrete Lotsizing and Scheduling Problem with sequence dependent set-up costs and set-up times. We have reformulated the problem as a Travelling Salesman Problem with time windows, and we have solved it to optimality using a dynamic programming algorithm due to Dumas et al. (1995). The approach presented here is the first one reported in the literature that is capable of solving medium sized lotsizing problems with sequence dependent set-up costs and sequence dependent set-up times to proven optimality. Our results may serve as benchmarks for future research on optimal methods or heuristics.

Our empirical study has shown that the performance of the suggested approach is sensitive to,

- *problem dimension.* The larger the dimension of the problem instances, the higher the CPU-times. However, an increase in the number of periods T leads to a much stronger increase in average CPU-times than an increase in the number of products N,
- *inventory holding costs.* As soon as inventory holding costs are involved, the DP-algorithm must be applied to the mirror network. Due to the structure of the time-windows in the mirror network, it turns out that CPU-times increase when inventory holding costs are present,
- *set-up times.* The presence of (sequence-dependent) set-up times decreases average CPU-times significantly, i.e., our DP-approach performs better on problems with set-up times than on problems without set-up times,
- *production capacity utilization.* Problem instances with either low or high production capacity utilization require less CPU-time than problem instances with medium capacity utilization.

An interesting subject for future research is to investigate whether other type of lotsizing problems with sequence dependent set-up costs and/or set-up times can also be solved to optimality using algorithms that have been developed for the Travelling Salesman Problem with Time Windows. Furthermore, a computational comparison between the performance of dynamic programming based approaches and alternative (polyhedral) approaches would be very useful.

Acknowledgements. The authors would like to thank Eric Gelinas from GERAD, Montreal, Canada for running the computational experiments and for his insightful comments. The research of the first author was partially supported by INSEAD (Grant #2062R). The research of the second author was partially supported by the Patrick F. and Helen C. Walsh Research Professorship and by the 'City of Lyon' Visiting Chair for Computer Integrated Manufacturing. Furthermore, the authors want to thank Erwin van der Laan, an anonymous referee (who pointed out an error in the original formulation of DLSPSD), and Andreas Drexl and Carsten Jordan from the University of Kiel (Germany) for their helpful comments on an earlier draft of this paper.

References

D. Cattrysse, M. Salomon, R. Kuik, and L.N. Van Wassenhove (1993). A dual ascent and column generation heuristic for the Discrete Lotsizing and Scheduling Problem with set-up times.

Management Science, 39(4):477–486.

J. Desrosiers, Y. Dumas, M. Solomon, and F. Soumis (1995). Time constrained routing and scheduling. Handbooks in Operations Research and Management Science, Volume on 'Networks', North-Holland publishers.

Y. Dumas, J. Desrosiers, E. Gelinas, and M.M. Solomon (1995). An optimal algorithm for the Travelling salesman problem with time windows. *Operations Research*, 43(2):367–371.

B. Fleischmann (1990). The discrete lotsizing and scheduling problem. *European Journal of Operational Research*, 44(3):337-348.

B. Fleischmann (1994). The discrete lot-sizing and scheduling problem with sequence-dependent set-up costs. *European Journal of Operational Research*, 75(2):395-404.

B. Fleischmann and Th. Popp (1989). Das dynamische Losgrössenproblem mit reihenfolgeabhängigen Rüstkosten. In: *Operations Research Proceedings 1988*, Springer-Verlag, Berlin-Heidelberg, 510–515. (in German).

S. van Hoesel, R. Kuik, M. Salomon, and L.N. Van Wassenhove (1994). The single-item discrete lotsizing and scheduling problem: Optimization by linear and dynamic programming. *Discrete Applied Mathematics*, 49:289–303.

C. Jordan, and A. Drexl (1994). Lotsizing and scheduling by batch sequencing. Working paper # 343. Christian-Albrecht-Universität zu Kiel, Kiel, Germany.

A. Langevin, M. Desrochers, J. Desrosiers, and F. Soumis (1990). A two-commodity flow formulation for the Travelling salesman problem with time windows. Working Paper G90-44, GERAD, Montreal, Canada.

T.L. Magnanti, and R. Vachani (1990). A strong cutting-plane algorithm for production scheduling with changeover costs. *Operations Research*, 38(3):456-473.

M. Salomon, L.G. Kroon, R. Kuik and L.N. Van Wassenhove (1991). Some extensions of the discrete lotsizing and scheduling problem. *Management Science*, 37(7):801–812.

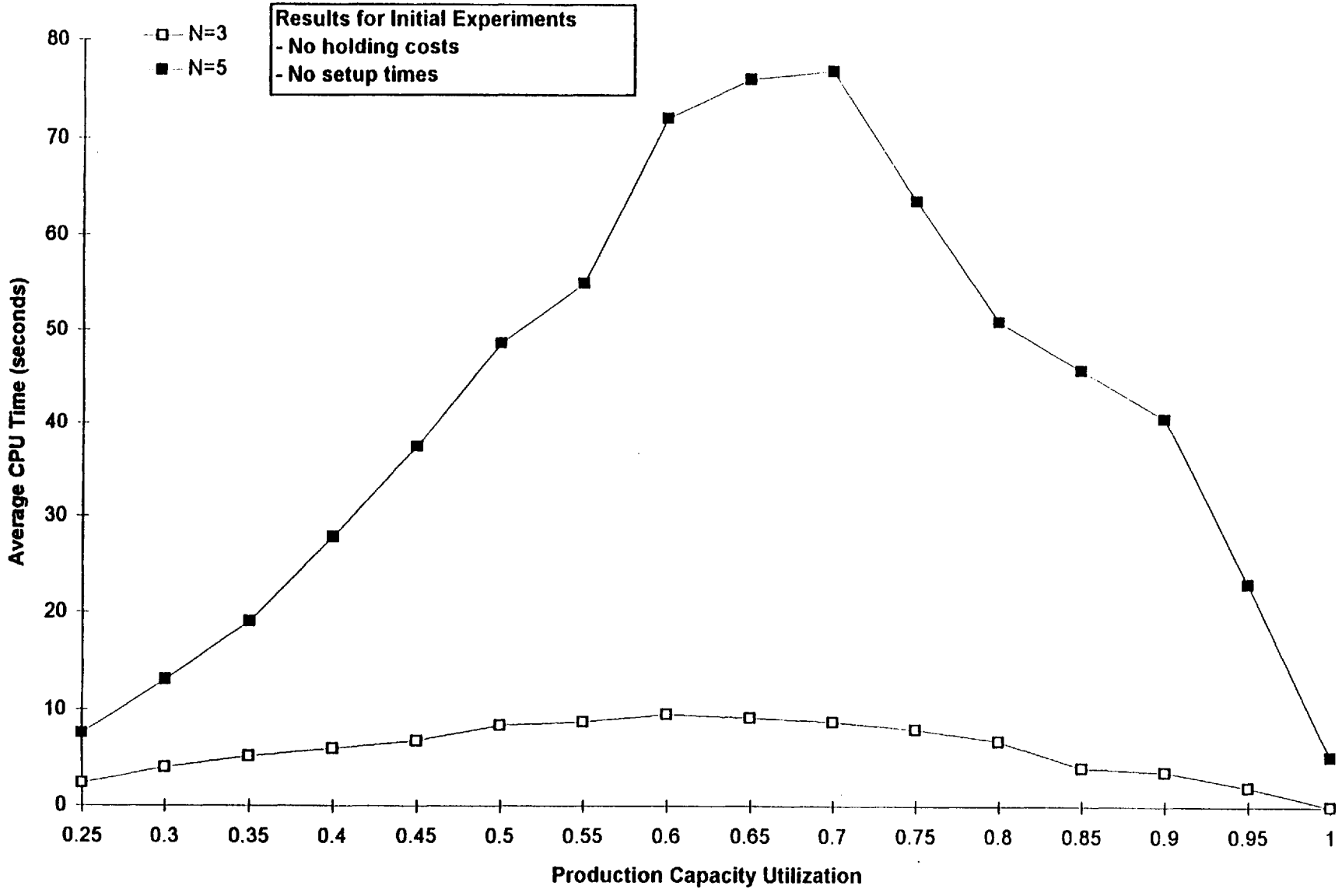
Results for Initial Experiments
- No holding costs
- No setup times
N=3
N=5
Average CPU Time (seconds)
0
10
20
30
40
50
60
70
80
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
Production Capacity Utilization

Figure 1

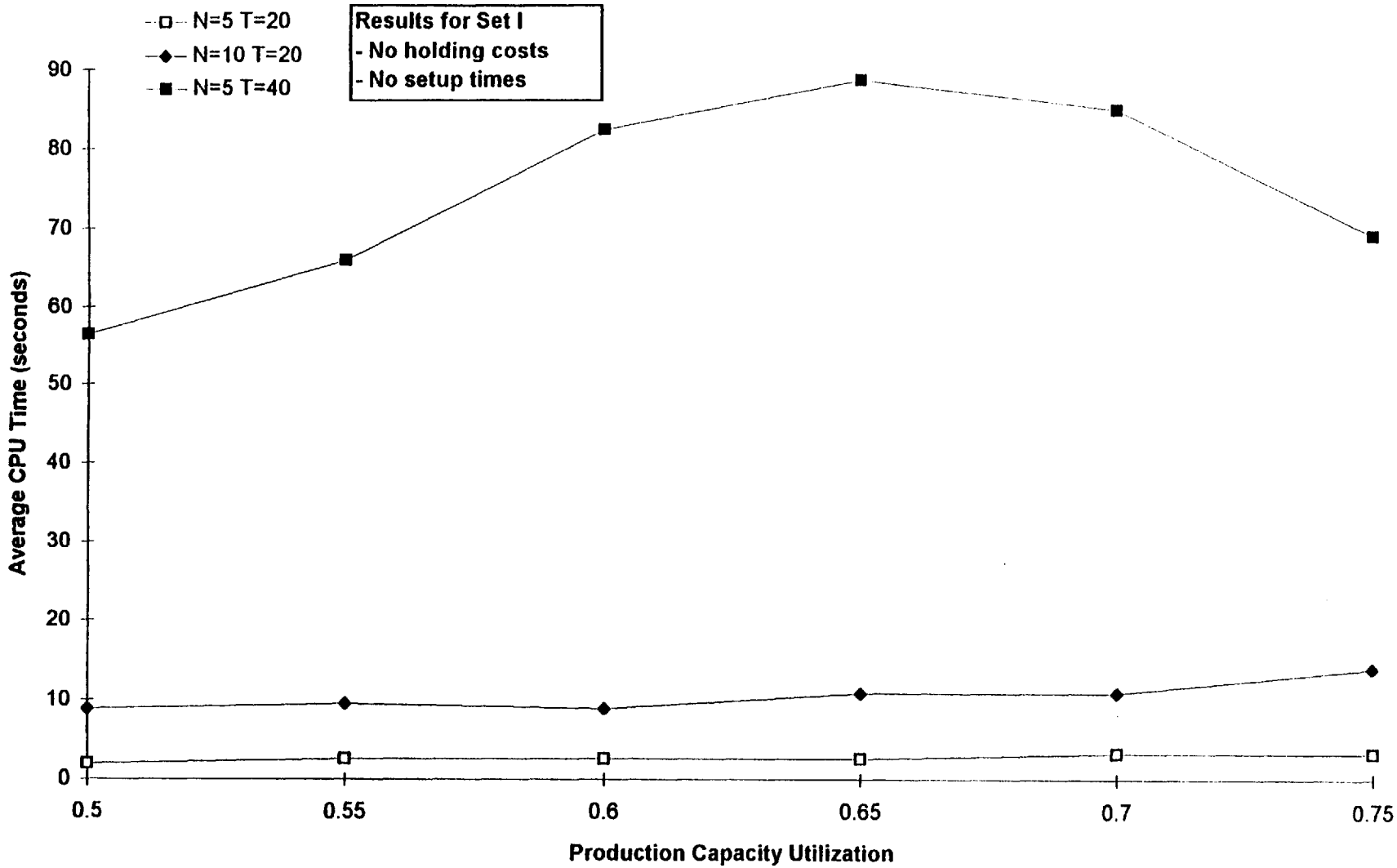
N=5 T=20
N=10 T=20
N=5 T=40
Results for Set I
- No holding costs
- No setup times
Average CPU Time (seconds)
90
80
70
60
50
40
30
20
10
0
0.5
0.55
0.6
0.65
0.7
0.75
Production Capacity Utilization

Figure 2

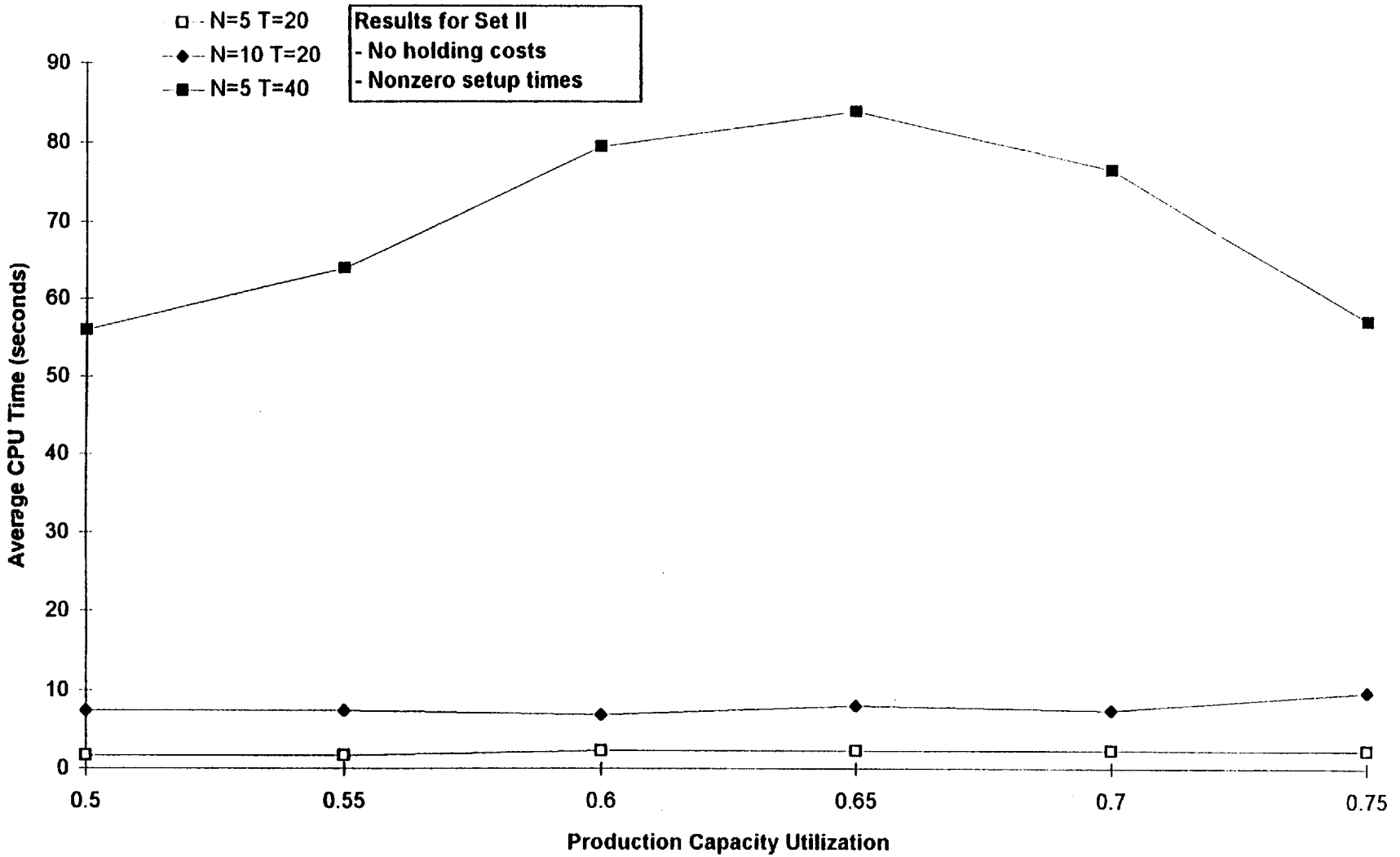
N=5 T=20
N=10 T=20
N=5 T=40
Results for Set II
- No holding costs
- Nonzero setup times
Average CPU Time (seconds)
90
80
70
60
50
40
30
20
10
0
0.5
0.55
0.6
0.65
0.7
0.75
Production Capacity Utilization

Figure 3

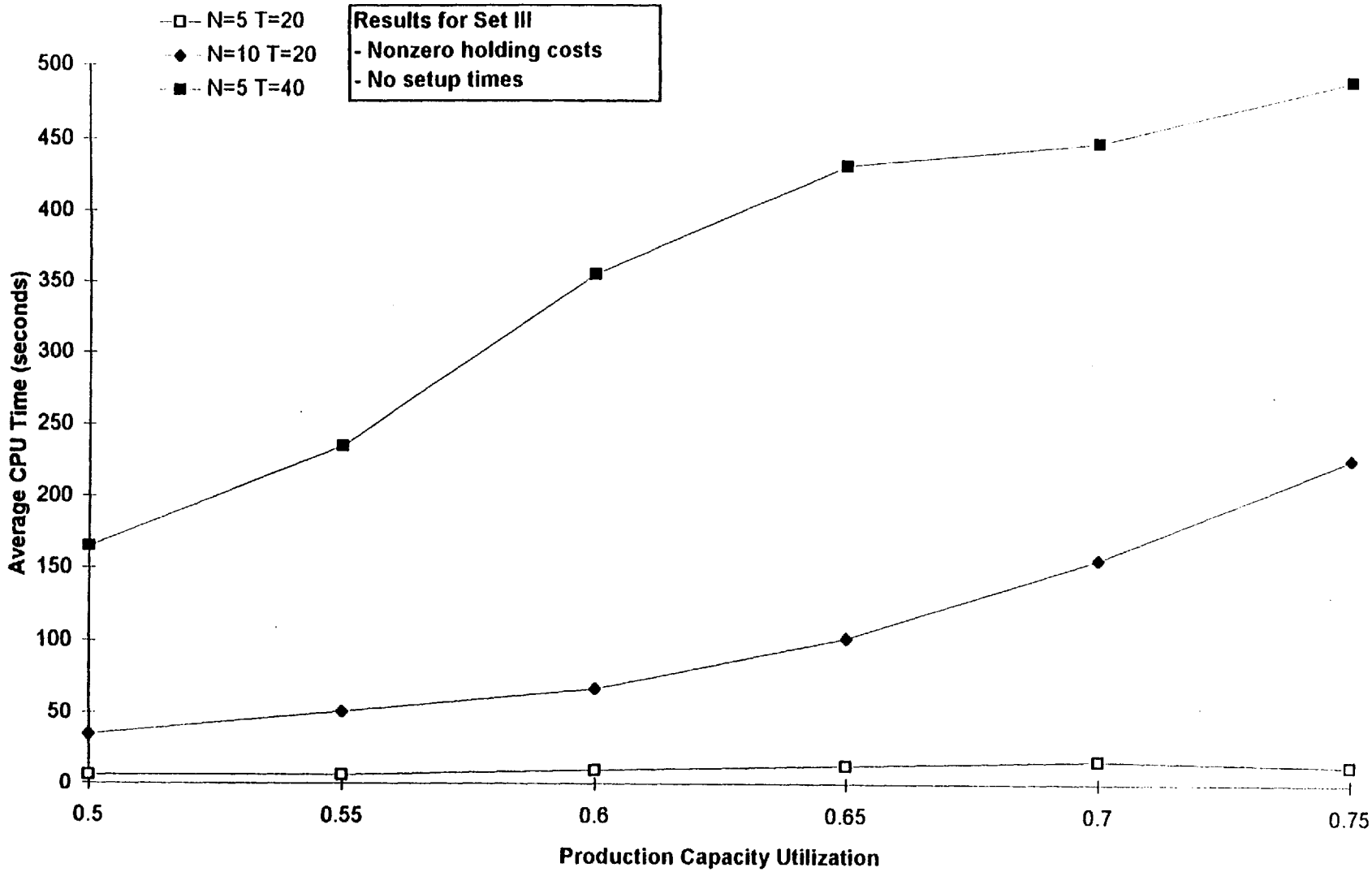
N=5 T=20
N=10 T=20
N=5 T=40
Results for Set III
- Nonzero holding costs
- No setup times
Average CPU Time (seconds)
500
450
400
350
300
250
200
150
100
50
0
0.5
0.55
0.6
0.65
0.7
0.75
Production Capacity Utilization

Figure 4

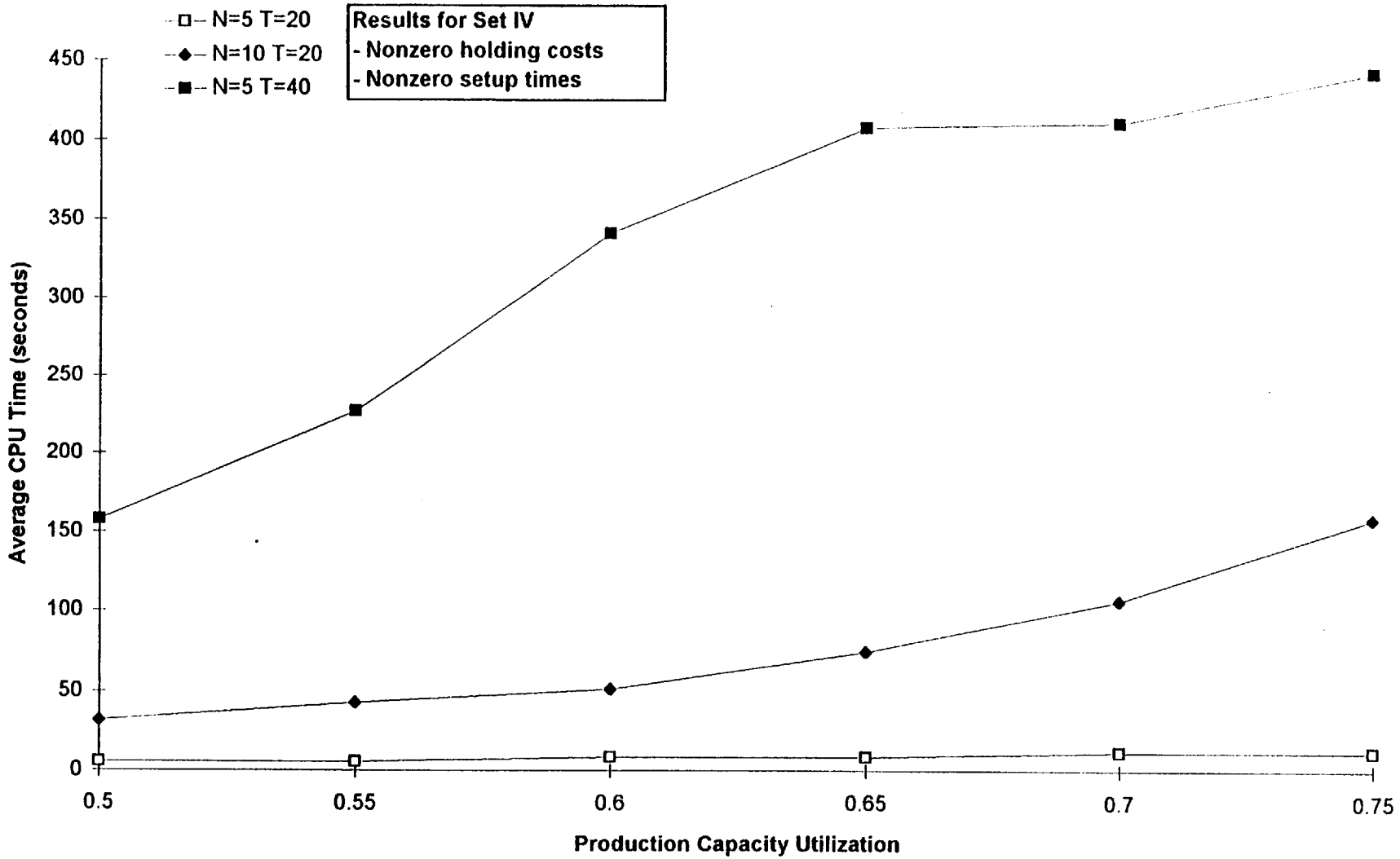
N=5 T=20
N=10 T=20
N=5 T=40
Results for Set IV
- Nonzero holding costs
- Nonzero setup times
Average CPU Time (seconds)
450
400
350
300
250
200
150
100
50
0
0.5
0.55
0.6
0.65
0.7
0.75
Production Capacity Utilization

Figure 5

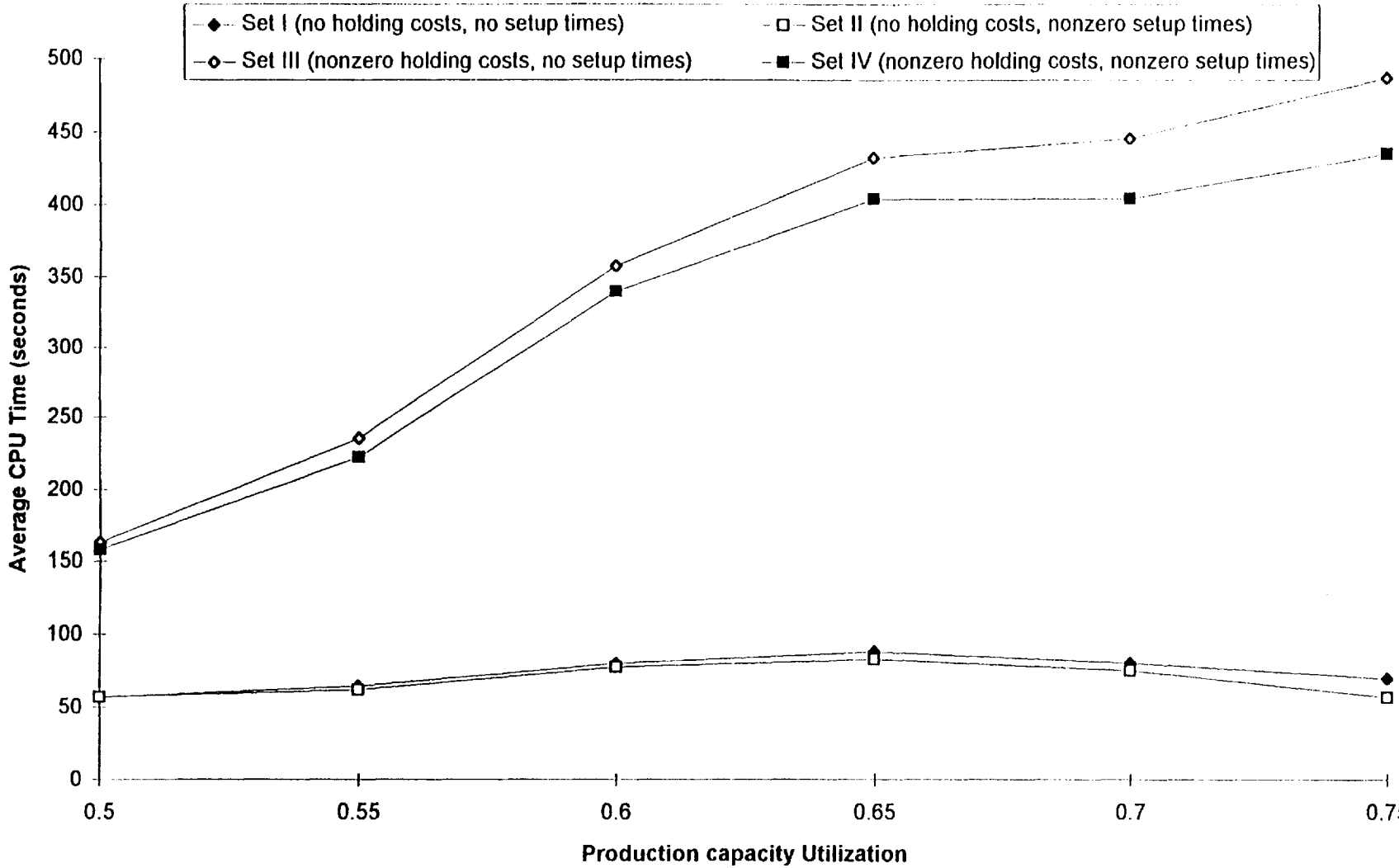
Set I (no holding costs, no setup times)
Set II (no holding costs, nonzero setup times)
Set III (nonzero holding costs, no setup times)
Set IV (nonzero holding costs, nonzero setup times)
Average CPU Time (seconds)
500
450
400
350
300
250
200
150
100
50
0
0.5
0.55
0.6
0.65
0.7
0.75
Production capacity Utilization

Figure 6