A Modification of Dynamic Programming Algorithms to Reduce the Running Time or/and Complexity

Evgeny R. Gafarov

Institute of Control Sciences of the Russian Academy of Sciences,
Profsoyuznaya st. 65, 117997 Moscow, Russia,
email: axel73@mail.ru

Alexander A. Lazarev

Institute of Control Sciences of the Russian Academy of Sciences,
Profsoyuznaya st. 65, 117997 Moscow, Russia,
Lomonosov Moscow State University,
Higher School of Economics – State University,
Moscow Institute of Physics and Technology – State University,
email: jobmath@mail.ru

Frank Werner

Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg,
PSF 4120, 39016 Magdeburg, Germany,
email: frank.werner@mathematik.uni-magdeburg.de

July 22, 2010

Abstract

In this paper, we present a modification of dynamic programming algorithms (*DPA*), which we denote as graphical algorithms (*GrA*). For the knapsack problem and some single machine scheduling problems, it is shown that the time complexity of the *GrA* is less than the time complexity of the standard *DPA*. Moreover, the average running time of the *GrA* is often essentially smaller. A *GrA* can also solve large-scale instances and instances, where the parameters are not integer. In addition, for some problems, *GrA* has a polynomial time complexity in contrast to a pseudo-polynomial complexity of *DPA*.

1 Introduction

Dynamic programming is a general optimization technique developed by Bellman [1]. It can be considered as a recursive optimization procedure which interprets the optimization problem as a multi-step decision process. This means that the problem is decomposed into a number of steps. In each step, a decision has to be made which has an impact on the decision to be made in later steps. By means of Bellman's optimization principle [1], a recursive equation is set up which describes the optimal criterion value in a given step in terms of the optimal criterion values of the previously considered step. Bellman's optimality principle can be briefly formulated as follows: Starting from any current step, an optimal policy for the subsequent steps is independent of the policy adopted in the previous steps. In the case of a combinatorial problem, in some step j sets of a particular size j are considered. To determine the optimal criterion value for a particular subset of size j, one has to know the optimal values for all necessary subsets of size $j-1$. If the problem includes n elements, the number of subsets to be considered is equal to $O(2^n)$. Therefore, dynamic programming usually results in an exponential complexity. However, if the problem considered is only NP-hard in the ordinary sense, it is possible to derive pseudo-polynomial algorithms.

In this paper, we give the basic idea of a graphical modification of dynamic programming algorithms (*DPA*), which we denote as *graphical algorithms* (*GrA*). This approach often reduces the number of states to be considered in each step of a *DPA*. Moreover, in contrast to a classical *DPA*, it can also treat problems with non-integer data without necessary transformations of the corresponding data. In addition, for some problems, a *GrA* essentially reduces the time complexity.

We note that for the knapsack problem and for the single machine weighted number of tardy jobs problem, dynamic programming algorithms with the same idea like that used in a *GrA* are known (see e.g. [2, 3]). In such a *DPA*, not all integer states $t \in [0, A]$ are considered but only such states which have different objective function values (here, A is the largest value of a state to be considered, e.g. the capacity of the knapsack in the knapsack problem). As a result, the time complexity of such a *DPA* is bounded by $O(nF_{opt})$, where F_{opt} is the optimal objective function value. However, these algorithms can be useful only for problems with $F_{opt} < A$, otherwise this procedure has no advantage over the classical *DPA*. We generalize the idea of such algorithms for an objective function, for which $F_{opt} \gg A$ may hold (e.g., for the single machine total weighted tardiness maximization problem).

Moreover, in contrast to algorithms from [2, 3], this modification is also

useful for some special scheduling problems, where the starting time is variable and one wishes to find all Pareto-optimal solutions (see Section 4.2).

This paper is organized as follows. In Section 2, we give the basic idea of the *GrA*. Specific graphical algorithms are presented for the binary knapsack problem in Section 3 and for some single machine scheduling problems in Section 4. In particular, for problem $1(nd)||\max\sum w_jT_j$ of maximizing total weighted tardiness on a single machine, whose complexity status was open up to now, we first give an NP-hardness proof and then a graphical algorithm. Here nd means that only non-delay schedules are considered as feasible solutions. For the single machine problem $1(nd)||\max\sum T_j$ of maximizing total tardiness, the resulting graphical algorithm improves the complexity from $O(n\sum p_j)$ to $O(n^2)$. Some concluding remarks are given in Section 5.

2 Basic Idea of the Graphical Algorithm

Usually in a dynamic programming algorithm, we have to compute the value $f_j(t)$ of a particular function for each possible state t at each stage (step) j of a decision process, where $t \in [0, A]$ and $t \in Z$. If this is done for any stage $j = 1, 2, \ldots, n$, where n is a size of the problem, the time complexity of such a *DPA* is typically $O(nA)$. However, often it is not necessary to store the result for any integer state since in the interval $[t_l, t_{l+1})$, we have a functional equation $f_j(t) = \varphi(t)$ for describing the best function value for a state t in step j (e.g. $f_j(t) = k_j \cdot t + b_j$, i.e., $f_j(t)$ is a continuous linear function when allowing also real values t).

Assume that we have the following functional equations in a *DPA* for a minimization problem, which correspond to Bellman's recursive equations:

$$f_j(t) = \min \begin{cases} \Phi^1(t) = \alpha_j(t) + f_{j-1}(t - a_j), & j = 1, 2, \ldots, n; \\ \Phi^2(t) = \beta_j(t) + f_{j-1}(t - b_j), & j = 1, 2, \ldots, n. \end{cases} \tag{1}$$

with the initial conditions

$$\begin{array}{ll} f_0(t) = 0, & \text{for } t \geq 0, \\ f_0(t) = +\infty, & \text{for } t < 0. \end{array} \tag{2}$$

In (1), function $\Phi^1(t)$ characterizes a setting $x_j = 1$ while $\Phi^2(t)$ characterizes a setting $x_j = 0$ representing a yes/no decision, e.g. for an item, a job, etc. In step j, $j = 1, 2, \ldots, n$, we compute and store the data

in the form given in Table 1.

Table 1: Computations in *DPA*

t	0	1	2	...	y	...	A
$f_j(t)$	$value_0$	$value_1$	$value_2$	...	$value_y$	...	$value_A$
optimal partial solution $X(t)$	$X(0)$	$X(1)$	$X(2)$	...	$X(y)$	...	$X(A)$

Here $X(y), y = 0, 1, \ldots, A$, is a vector which describes an optimal partial solution and which consists of j elements (values) $x_1, x_2, \ldots, x_j \in \{0, 1\}$.

However, this data can also be stored in a condense tabular form as given in Table 2.

Table 2: Computations in *GrA*

t	$[t_0, t_1)$	$[t_1, t_2)$	...	$[t_l, t_{l+1})$	...	$[t_{m_j-1}, t_{m_j}]$
$f_j(t)$	$\varphi_1(t)$	$\varphi_2(t)$	...	$\varphi_{l+1}(t)$	...	$\varphi_{m_j}(t)$
optimal partial solution $X(t)$	$X(t_0)$	$X(t_1)$	...	$X(t_l)$	...	$X(t_{m_j-1})$

Here, we have $0 = t_0 < t_1 < t_2 < \ldots < t_{m_j} = A$. To compute function $f_{j+1}(t)$, we compare two temporary functions $\Phi^1(t)$ and $\Phi^2(t)$ which are as follows.

The function $\Phi^1(t)$ is a combination of the terms $\alpha_{j+1}(t)$ and $f_j(t-a_{j+1})$. Function $f_j(t - a_{j+1})$ has the same structure as in Table 2, but all intervals $[t_l, t_{l+1})$ have been replaced by $[t_l - a_{j+1}, t_{l+1} - a_{j+1})$, i.e., we shift the graph of function $f_j(t)$ to the right by the value a_{j+1}. If we can present function $\alpha_{j+1}(t)$ in the same form as in Table 2 with μ_1 columns, we store function $\Phi^1(t)$ in the form of Table 2 with $m_j + \mu_1$ columns. In an analogous way, we store function $\Phi^2(t)$ in the form of Table 2 with $m_j + \mu_2$ columns.

Then we construct function

$$f_{j+1}(t) = \min\{\Phi^1(t), \Phi^2(t)\}.$$

For example, let the columns of Table $\Phi^1(t)$ contain the intervals

$$[t^1_0, t^1_1), [t^1_1, t^1_2), \ldots, [t^1_{(m_j+\mu_1)-1}, t^1_{(m_j+\mu_1)}]$$

and the columns of Table $\Phi^2(t)$ contain the intervals

$$[t^2_0, t^2_1), [t^2_1, t^2_2), \ldots, [t^2_{(m_j+\mu_2)-1}, t^2_{(m_j+\mu_2)}].$$

To construct function $f_{j+1}(t)$, we compare the two functions $\Phi^1(t)$ and $\Phi^2(t)$ on each interval, which is formed by means of the points

$$\{t_0^1, t_1^1, t_2^1, \ldots, t_{(m_j+\mu_1)-1}^1, t_{(m_j+\mu_1)}^1, t_0^2, t_1^2, t_2^2, \ldots, t_{(m_j+\mu_2)-1}^2, t_{(m_j+\mu_2)}^2\},$$

and we determine the intersection points $t_1^3, t_2^3, \ldots, t_{\mu_3}^3$. Thus, in the table of function $f_{j+1}(t)$, we have at most $2m_j + \mu_1 + \mu_2 + \mu_3 \leq A$ intervals.

In fact, in each step $j = 1, 2, \ldots, n$, we do not consider all points $t \in [0, A],\ t \in Z$, but only points from the interval in which the optimal partial solution changes or where the resulting functional equation of the objective function changes. For some objective functions, the number of such points M is small and the new algorithm based on this graphical approach has a time complexity of $O(n \min\{A, M\})$ instead of $O(nA)$ for the original dynamic programming algorithm.

Moreover, such an approach has some other advantages.

1. The *GrA* can solve instances, where (some of) the parameters $a_j,\ b_j,\ j = 1, 2, \ldots, n$ or/and A are not in Z.

2. The running time of the *GrA* for two instances with the parameters $\{a_j, b_j, A\}$ and $\{a_j \cdot 10^k \pm 1, b_j \cdot 10^k \pm 1, A \cdot 10^k \pm 1\}, k > 1$, is the same while the running time of the *DPA* will be 10^k times larger in the second case. Thus, using the *GrA*, one can usually solve considerably larger instances.

3. Properties of an optimal solution are taken into account (e.g. for the knapsack problem, it is possible that an item with the smallest value $\frac{c_j}{a_j}$ does not influence the running time, for the notation see Section 3).

4. As we will show below, for several problems, the *GrA* has even a polynomial time complexity or we can at least essentially reduce the complexity of the standard *DPA*.

3 Graphical Algorithm for the Knapsack Problem

In this section, we describe the application of this approach to the one-dimensional knapsack problem.

One-dimensional knapsack problem (KP): One wishes to fill a knapsack of capacity A with items having the largest possible total utility. If any item can be put at most once into the knapsack, we get the binary

or $0-1$ knapsack problem. This problem can be written as the following integer linear programming problem:

$$\begin{cases} f(x) = \sum\limits_{j=1}^{n} c_j x_j \to \max \\ \sum\limits_{j=1}^{n} a_j x_j \le A; \\ x_j \in \{0,1\},\ j = 1,2,\ldots,n. \end{cases} \tag{3}$$

Here, $c_j > 0$ gives the utility and $a_j > 0$ the required capacity of item $j,\ j = 1,2,\ldots,n$. The variable $x_j \in \{0,1\}$ characterizes whether item j is put into the knapsack or not.

The dynamic programming algorithm based on Bellman's optimality principle is one of the standard algorithms for KP. It is assumed that all parameters are integer: $A, a_j \in Z^+, j = 1,2,\ldots,n$. Note that there also exists a dynamic programming algorithm by Papadimitriou [2] with complexity $O(nF_{opt})$, where F_{opt} denotes the optimal objective function value. However, in the following, we describe the standard algorithm by Bellman since the graphical algorithm is derived from this variant.

For KP, Bellman's recursive equations are as follows:

$$f_j(t) = \max \begin{cases} \Phi^1(t) = c_j + f_{j-1}(t - a_j), & j = 1,2,\ldots,n; \\ \Phi^2(t) = f_{j-1}(t), & j = 1,2,\ldots,n. \end{cases} \tag{4}$$

where

$$\begin{aligned} f_0(t) &= 0, \quad t \ge 0, \\ f_0(t) &= +\infty,\ t < 0. \end{aligned}$$

$\Phi^1(t)$ represents the setting $x_j = 1$ (i.e., item j is put into the knapsack) while $\Phi^2(t)$ represents the setting $x_j = 0$ (i.e., item j is not put into the knapsack). In each step $j,\ j = 1,2,\ldots,n$, the function values $f_j(t)$ are calculated for each integer point (i.e., 'state') $0 \le t \le A$. For each point t, a corresponding best (partial) solution $X(t) = (x_1(t), x_2(t), \ldots, x_j(t))$ is stored.

The algorithm is illustrated by the following example:

$$\begin{cases} f(x) = 5x_1 + 7x_2 + 6x_3 + 3x_4 \to \max \\ 2x_1 + 3x_2 + 5x_3 + 7x_4 \le 9; \\ x_j \in \{0,1\},\ j = 1,\ldots,4. \end{cases} \tag{5}$$

Table 3: Application of the dynamic programming algorithm

t	$f_1(t)$	$X(t)$	$f_2(t)$	$X(t)$	$f_3(t)$	$X(t)$	$f_4(t)$	$X(t)$
0	0	(0,,,)	0	(0,0,,)	0	(0,0,0,)	0	(0,0,0,0)
1	0	(0,,,)	0	(0,0,,)	0	(0,0,0,)	0	(0,0,0,0)
2	5	(1,,,)	5	(1,0,,)	5	(1,0,0,)	5	(1,0,0,0)
3	5	(1,,,)	7	(0,1,,)	7	(0,1,0,)	7	(0,1,0,0)
4	5	(1,,,)	7	(0,1,,)	7	(0,1,0,)	7	(0,1,0,0)
5	5	(1,,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
6	5	(1,,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
7	5	(1,,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
8	5	(1,,,)	12	(1,1,,)	13	(0,1,1,)	13	(0,1,1,0)
9	5	(1,,,)	12	(1,1,,)	13	(0,1,1,)	13	(0,1,1,0)

The results with the dynamic programming algorithm are summarized in Table 3. Therefore, for $t = 9$, we get the optimal solution

$$X(13) = (x_1(13),\ x_2(13),\ x_3(13),\ x_4(13)) = (0, 1, 1, 0)$$

and the corresponding optimal objective function value $f_4(9) = 13$. The time complexity of this algorithm is $O(nA)$.

The idea of the *GrA* [13] for KP is as follows. In each step of *GrA*, we store function $f_j(t)$ in tabular form as given in Table 4, where $0 = t_1 < t_2 < \ldots < t_{m_j}$ and $W_1 < W_2 < \ldots < W_{m_j}$.

Table 4: Function $f_l(t)$

t	t_1	t_2	$\ldots$	t_{m_j}
$f_j(t)$	W_1	W_2	$\ldots$	W_{m_j}
optimal partial solution $X(t)$	X_1	X_2	$\ldots$	X_{m_j}

The above data means the following. For each value $t \in [t_l, t_{l+1}),\ 1 \leq l < m_j$, we have an optimal partial solution $X_l = (x_1, x_2, \ldots, x_j)$ and the objective function value $f_j(t) = \sum_{i=1}^{j} c_i \cdot x_i = W_l$. The points t_l are called the *break points* (or 'jumps' in this case), i.e., we have $f_j(t') < f_j(t'')$ for $t' < t_l \leq t''$.

In the next step $j + 1$, we transform function $f_j(t)$ into functions

$$\Phi^1(t) = c_{j+1} + f_j(t - a_{j+1}) \qquad \text{and} \qquad \Phi^2(t) = f_j(t)$$

in $O(m_j)$ operations, i.e., the graph of $\Phi^1(t)$ can be constructed from the graph of $f_j(t)$ by an 'upward' shift by c_{j+1} and a 'right' shift by a_{j+1}. In

each of the tables for $\Phi^1(t)$ and $\Phi^2(t)$, we have at most m_j break points. Then we compute a new table of function

$$f_{j+1}(t) = \max\{\Phi^1(t), \Phi^2(t)\}$$

in $O(m_j)$ operations. In the new table of function $f_{j+1}(t)$, there are at most $2m_j$ break points (usually, this number is smaller). In fact, we do not consider all points t from the interval $[0, A]$, but only points from this interval at which the objective function value changes.

If we have a situation illustrated in Table 5, then we cut the column which corresponds to point t_{l+1}, i.e., we can combine both intervals $[t_{l-1}, t_l)$ and $[t_l, t_{l+1})$, and we can take the same optimal partial solution: $X_{l+1} := X_l$.

Table 5: Reducing the number of intervals for function $f_{j+1}(t)$

t	...	t_l	t_{l+1}	...
$f_{j+1}(t)$	...	W_l	$W_{l+1} = W_l$	...
optimal partial schedule	...	X_l	X_{l+1}	...

Let us consider the computational steps of the *GrA* for the instance considered for illustration the *DPA*. The results of the calculations are presented in Tables 6-9.

Table 6: Step $j = 1$.

t	0	2
$f_1(t)$	0	5
$X(t)$	(0, , ,)	(1, , ,)

Due to $a_2 = 3$, one has to consider the intervals obtained from the boundary points $0, 2, 0 + 3, 2 + 3$ in order to construct function $f_2(t)$. As the result, we get Table 7.

Table 7: Step $j = 2$.

t	0	2	3	5
$f_2(t)$	0	5	7	12
$X(t)$	(0, 0, ,)	(1, 0, ,)	(0, 1, ,)	(1, 1, ,)

To construct function $f_3(t)$, one has to consider the intervals obtained from the boundary points $0, 2, 3, 5, 0 + 5, 2 + 5, 3 + 5$ due to $a_3 = 5$. The point $5 + 5 > 9$ need not to be considered. We obtain the results given in

Table 8.

Table 8: Step $j = 3$.

t	0	2	3	5	8
$f_3(t)$	0	5	7	12	13
$X(t)$	(0, 0, 0,)	(1, 0, 0,)	(0, 1, 0,)	(1, 1, 0,)	(0, 1, 1,)

In the last step, one has to consider the intervals resulting from the boundary points $0, 2, 3, 5, 8, 0+7, 2+7$ in order to construct function $f_4(t)$. The points $3+7, 5+7, 8+7$ need not to be considered since they are larger than $A = 9$. Therefore, it suffices to consider five points. As a result, we obtain Table 9.

Table 9: Step $j = 4$.

t	0	2	3	5	8
$f_4(t)$	0	5	7	12	13
$X(t)$	(0, 0, 0, 0)	(1, 0, 0, 0)	(0, 1, 0, 0)	(1, 1, 0, 0)	(0, 1, 1, 0)

It is obvious that the time complexity of *GrA* is $O(\min\{2^n, n \min\{A, F_{opt}\}\})$, where F_{opt} is the optimal objective function value.

For the numerical instance, the *GrA* stored 12 break points: 1 in the first step ($t = 2$), 3 in the second step ($t = 2, 3, 5$), 4 in the third step ($t = 2, 3, 5, 8$), and 4 in the last step ($t = 2, 3, 5, 8$), too. In contrast, the standard dynamic programming algorithm would consider $4 \times 9 = 36$ points. Consequently, the running time of the *DPA* can be essentially reduced.

It is known that for the following instance of KP [5], known Branch and Bounds algorithms with any method of choosing a subproblem or a variable for branching have an exponential time complexity [5, 6]:

$$\begin{cases} f(x) = \sum_{j=1}^{n} 2 \cdot x_j \to \max \\ \sum_{j=1}^{n} 2 \cdot x_j \le 2\lfloor \frac{n}{2} \rfloor + 1. \end{cases} \tag{6}$$

Let us now consider a generalized instance as follows:

$$\begin{cases} f(x) = \sum_{j=1}^{n} a \cdot 2 \cdot x_j \to \max \\ \sum_{j=1}^{n} a \cdot 2 \cdot x_j \le a \cdot \left(2\lfloor \frac{n}{2} \rfloor + 1\right), \end{cases} \tag{7}$$

where $a > 1$.

It is easy to show that for each a, the graphical algorithm has a running time of $O(n)$ since in each step of the *GrA*, only objective function values from the set $\{0, a, 2a, \ldots, n \cdot a\}$ have to be considered. For the first instance, the running time of the classical *DPA* requires $O(n \cdot 2\lfloor \frac{n}{2} \rfloor)$ operations and for the second one, it requires $O(n \cdot a \cdot 2\lfloor \frac{n}{2} \rfloor)$ operations.

In [6], a worse subcase (i.e., a subcase where the number of points in the search tree of a Branch and Bounds algorithm is greater than in [5]) has been presented. For this type of instances, we have a set of numbers $B = \{b_1, b_2, \ldots, b_n\}$ which are used by the following parameterized optimization instance:

$$\begin{cases} f(x) = \sum\limits_{j=1}^{m+n} c_j x_j \to \max \\ \sum\limits_{j=1}^{m+n} c_j x_j \leq ka + 1, \\ a \geq 2, \\ c_j = a, \ j = 1, 2, \ldots, m, \\ c_j = b_j \cdot a, \ j = m+1, 2, \ldots, m+n, \\ x_j \in \{0, 1\}, \ j = 1, 2, \ldots, m+n. \end{cases} \tag{8}$$

It is obvious that the time complexity of the *GrA* for this type of instances is independent of a and is equal to $O(m + n \min\{k, 2^n\})$, where $k \leq m + \sum_{i=1}^{n} b_i$. Clearly, this complexity can be essentially smaller than the running time of the Branch and Bound algorithm (which is already $O(\frac{2^n}{\sqrt{n}})$ for instance (6) and not smaller for instance (8)) and the running time of the standard *DPA*.

Thus, the use of a *GrA* can reduce both the time complexity and the running time of existing algorithms for KP. The application of the *GrA* to the partition problem is described in detail in [8], where also computational results are presented and compared with algorithms from [14].

4 Graphical Algorithm for Single Machine Scheduling Problems

In this section, we present graphical algorithms for several single machine scheduling problems, which can be formulated as follows.

We are given a set $N = \{1, 2, \ldots, n\}$ of n independent jobs that must be processed on a single machine. Preemptions of a job are not allowed and at any time, no more than one job can be processed. The processing of the jobs

starts at time 0. For each job $j \in N$, a processing time $p_j > 0$, a weight w_j and a due date d_j are given.

A schedule is uniquely determined by a permutation $\pi = (j_1, j_2, \ldots, j_n)$ of the jobs of set N. Let $C_{j_k}(\pi) = \sum_{l=1}^{k} p_{j_l}$ be the completion time of job j_k in schedule π. If $C_j(\pi) > d_j$, then job j is tardy and we have $U_j(\pi) = 1$, otherwise $U_j(\pi) = 0$. If $C_j(\pi) \leq d_j$, then job j is said to be on-time. Moreover, let $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$ be the tardiness of job j in schedule π. Additionally, let $GT_j(\pi) = \min\{\max\{0, C_j(\pi) - d_j\}, p_j\}$.

For the problem of minimizing the weighted number of tardy jobs $1||\sum w_j U_j$, the objective is to find an optimal schedule π^* that minimizes the value $\sum_{j=1}^{n} w_j U_j(\pi)$ and for the problem of maximizing total tardiness $1(nd)||\max\sum T_j$ [4, 9, 11], the objective is to find an optimal schedule π^* that maximizes the value $\sum_{j=1}^{n} T_j(\pi)$, where each feasible schedule starts at time 0 and does not have any idle time between the processing of jobs. Additionally, for the special case of the single machine generalized total tardiness minimization problem $1||\sum GT_j$, we wish to minimize the value $\sum_{j=1}^{n} GT_j(\pi)$ [10].

In [13], the authors propose an algorithm for the special NP-hard case of the single machine total tardiness minimization problem, where $p_1 \geq \ldots \geq p_n,\ d_1 \leq \ldots \leq d_n,\ d_n - d_1 \leq p_n$. This special case is called B-1.

In Table 10, we summarize the time complexity of existing and new graphical algorithms for particular scheduling problems.

Table 10: Time complexity of the *GrA*

Problem	Time complexity of *GrA*	Time complexity of classical *DPA*
$1\|\|\sum w_j U_j$	$O(\min\{2^n, n \cdot \min\{d_{max}, F_{opt}\}\})$ [10]	$O(nd_{max})$
$1\|d_j = d'_j + A\|\sum U_j$ [7]	$O(n^2)$	$O(n\sum p_j)$
$1\|\|\sum GT_j$	$O(\min\{2^n, n \cdot d_{max}\})$ [10]	$O(nd_{max})$
$1\|\|\sum T_j$ special case B-1	$O(\min\{2^n, n \cdot d_{max}\})$	$O(nd_{max})$
$1(nd)\|\|\max\sum w_j T_j$	$O(\min\{2^n, n \cdot \min\{d_{max}, \sum w_j\}\})$	$O(nd_{max})$
$1(nd)\|\|\max\sum T_j$	$O(n^2)$ [11]	$O(nd_{max})$

Usually, the running time of the *GrA* will be smaller while the running time of the *DPA* is equal to the time complexity.

4.1 Graphical Algorithm for the Minimization of the Weighted Number of Tardy Jobs on a Single Machine

Lemma 1 *For problem* $1||\sum w_jU_j$*, there exists an optimal schedule* $\pi = (G, H) = (EDD, LDD)$*, where all jobs* $j \in H$ *are tardy and all jobs* $i \in G$ *are on-time. All jobs from the set* G *are processed in EDD (early due date) order and all jobs from the set* H *are processed in LDD (last due date) order.*

Note that in an optimal schedule, the on-time jobs can be scheduled in EDD order while the tardy jobs can be scheduled in arbitrary order. Now we present a solution algorithm for the problem of maximizing the total weight of the on-time jobs which is identical to the problem under consideration. The following algorithm is based on Lemma 1.

Algorithm 1

1. Enumerate the jobs according to non-increasing due dates: $d_1 \geq d_2 \geq \ldots \geq d_n$.

2. $\pi_1(t) := (1)$. For each $t \in Z \bigcap [0, \sum_{i=2}^{n} p_i]$, compute:
   if $p_1 + t - d_1 \leq 0$, then $f_1(t) := w_1$ else $f_1(t) := 0$;

3. FOR $j := 2$ TO n DO

   FOR $t := 0$ TO $\sum_{i=j+1}^{n} p_i$ $(t \in Z)$ DO
   $\pi^1 := (j, \pi_{j-1}(t + p_j))$, $\pi^2 := (\pi_{j-1}(t), j)$;
   If $p_j + t - d_j \leq 0$, then $\Phi^1(t) := w_j + f_{j-1}(t + p_j)$
   else $\Phi^1(t) := f_{j-1}(t + p_j)$;
   If $\sum_{i=1}^{j} p_i + t - d_j \leq 0$, then $\Phi^2(t) := f_{j-1}(t) + w_j$
   else $\Phi^2(t) := f_{j-1}(t)$;
   If $\Phi^1(t) < \Phi^2(t)$, then $f_j(t) := \Phi^2(t)$ and $\pi_j(t) := \pi^2$,
   else $f_j(t) := \Phi^1(t)$ and $\pi_j(t) := \pi^1$;

4. $\pi_n(0)$ is an optimal schedule with the objective function value $f_n(0)$.

$\pi_j(t)$ represents the best partial schedule (sequence) of the jobs $1, 2, \ldots, j$ when the first job starts at time t, and $f_j(t) = \sum_{i=1}^{j} w_i[1 - U_i(\pi_j(t))]$ denotes the corresponding weighted number of on-time jobs.

Theorem 1 *[10] Algorithm 1 constructs an optimal schedule for problem* $1||\sum w_jU_j$ *in* $O(n\sum p_j)$ *time.*

In fact, for this dynamic programming algorithm, we have the following functional equations:

$$f_j(t) = \max \begin{cases} \Phi^1(t) = \alpha(t) + f_{j-1}(t+p_j), & j = 1, 2, \ldots, n; \\ \Phi^2(t) = \beta(t) + f_{j-1}(t), & j = 1, 2, \ldots, n. \end{cases} \tag{9}$$

where

$$f_0(t) = 0 \text{ for } t \geq 0.$$

$\Phi^1(t)$ represents the setting $x_j = 1$ (i.e., we add the job j to the beginning of the partial schedule) while $\Phi^2(t)$ represents the setting $x_j = 0$ (i.e., we add it to the end). If $p_j + t - d_j \leq 0$, then $\alpha(t) = w_j$ else $\alpha(t) = 0$. If $\sum_{i=1}^{j} p_i + t - d_j \leq 0$, then $\beta(t) = w_j$ else $\beta(t) = 0$.

Algorithm 1 can be modified by considering for each $j = 1, 2, \ldots, n$, only the interval $[0, d_j - p_j]$ instead of the interval $[0, \sum_{i=j+1}^{n} p_i]$ since for each $t > d_j - p_j$, job j is tardy in any partial schedule $\pi_j(t)$ and the partial schedule $\pi^2 := (\pi_{j-1}(t), j)$ is optimal. Thus, the time complexity of the modified Algorithm 1 is equal to $O(nd_{max})$. Here we note that one can assume that $d_{max} < \sum_{j=1}^{n} p_j$ since otherwise the job with maximal due date is always on-time and can be excluded from the consideration.

The idea of the graphical algorithm (GrA) for this problem is as follows. In each step of the GrA, we store function $f_j(t)$ in tabular form as given in Table 11, where $t_1 < t_2 < \ldots < t_{m_j}$ and $W_1 > W_2 > \ldots > W_{m_j}$.

Table 11: Function $f_j(t)$

t	t_1	t_2	$\ldots$	t_{m_j}
$f_j(t)$	W_1	W_2	$\ldots$	W_{m_j}
optimal partial schedule	π_1	π_2	$\ldots$	π_{m_j}

The above data means the following. For each value $t \in (t_l, t_{l+1}]$, $1 \leq l < m_j$, we have an optimal partial schedule $\pi_l = (G, H) = (EDD, LDD)$ and the objective function value $f_j(t) = W_l = \sum_{i \in G} w_i$. The points t_l are called the *break points*, i.e., we have $f_j(t') > f_j(t'')$ for $t' \leq t_l < t''$.

In the next step $j + 1$, we transform function $f_j(t)$ into functions $\Phi^1(t)$ and $\Phi^2(t)$ according to Step 3 of Algorithm 1 in $O(m_j)$ operations. In each

of the tables for $\Phi^1(t)$ and $\Phi^2(t)$, we have at most m_j+1 break points. Then we compute a new table of the function

$$f_{j+1}(t) = \max\{\Phi^1(t), \Phi^2(t)\}$$

in $O(m_j)$ operations. In the new table of function $f_{j+1}(t)$, there are at most $2m_j + 2$ break points (usually, this number is smaller). In fact, we do not consider all points t from the interval $[0, \min\{d_j - p_j,\ \sum_{i=j+1}^{n} p_i\}]$, but only points from this interval at which the objective function value changes.

If we have a situation described in Table 12, we cut the column corresponding to point t_{l+1}, combine both intervals, and we can use $\pi_l := \pi_{l+1}$.

Table 12: Reducing the number of intervals for function $f_{j+1}(t)$

t	...	t_l	t_{l+1}	...
$f_{j+1}(t)$	...	W_l	$W_{l+1} = W_l$	...
optimal partial schedule	...	π_l	π_{l+1}	...

It is obvious that we will have at most F_{opt} break points in the GrA, where $F_{opt} \leq \sum_{j=1}^{n} w_j$ is the optimal total weight of the on-time jobs. In each step $j = 1, 2, \ldots, n$ of the GrA, we have to consider at most $\min\{2^j,\ d_j - p_j,\ \sum_{i=j+1}^{n} p_i, \sum_{i=1}^{j} w_i, F_{opt}\}$ break points. Thus, the time complexity of the GrA is $O(\min\{2^n, n \cdot \min\{d_{max}, F_{opt}\}\})$.

4.2 Graphical Algorithm for the Single Machine of Minimizing the Number of Late Jobs when the Starting Time of the Machine is Variable

In the standard model of problem $1||\sum U_j$, it is generally assumed that the machine starts at time $t_0 = 0$. In this section, we abandon this assumption. We assume that it is possible to start the machine at any possible time $-G$, where $G \geq 0$. The quality of a feasible schedule is measured by two criteria. The first one is the number of late jobs (or, what is the same, the number of on-time jobs) and the second one is the cost of the starting time of the machine. We wish to find all $n + 1$ Pareto optimal schedules. Denote this problem as $1|d_j = d'_j + G|\sum U_j$.

In [7], an exact algorithm for the problem with time complexity $O(n^4)$ has been proposed. In this section, we present an algorithm with time complexity $O(n^2)$.

First, we construct a modification of Algorithm 1, where $w_j = 1$ for all $j = 1, 2, \ldots, n$, and in each step $j = 1, 2, \ldots, n$, we consider the interval

$[d_{min} - \sum_{i=1}^{n} p_i, \sum_{i=j+1}^{n} p_i]$. The time complexity of the resulting algorithm is $O(n \sum p_j)$. The graphical modification of this algorithm for problem $1|d_j = d'_j + G|\sum U_j$ has the time complexity $O(n \min\{d_{max}, 2^n, \sum_{i=1}^{n} w_i, F_{opt}\})$. Since $\sum_{i=1}^{n} w_i = n$, the time complexity of GrA for problem $1|d_j = d'_j + G|\sum U_j$ is $O(n^2)$.

We recall that a 'state' t in the GrA has the same meaning like $-G$, i.e., t denotes the starting time of a (partial) schedule. Assume that in the last step of the GrA, we have obtained the data given in Table 13.

Table 13: Function $f_n(t)$

t	t_1	t_2	...	t_{n+1}
$f_n(t)$	n	$n-1$	...	0
optimal partial schedule	π_1	π_2	...	π_{n+1}

This means that we have $n+1$ Pareto-optimal solutions described by the pairs (t_2, n), $(t_3, n-1), \ldots, (t_{n+1}+1, 0)$, where the first value is $-G$ and the second one is the number of on-time jobs.

Moreover, we can use the same idea of the GrA for the generalized problem $1|d_j = d'_j + G|\sum w_j U_j$ although for the generalized problem, the time complexity of the GrA will be pseudo-polynomial (see the previous section).

4.3 Graphical Algorithm for Maximizing Weighted Total Tardiness on a Single Machine

In contrast to the usual minimization problem, here we wish to maximize the value $\sum_{j=1}^{n} w_j T_j(\pi)$, where each feasible schedule starts at time 0 and does not have any idle time between the processing of jobs. The problem has both a theoretical importance and some practical interpretations [4, 9, 11, 12]. For this problem, in the next subsections, we present an NP-hardness proof, a pseudo-polynomial algorithm and the graphical modification (GrA) of this algorithm. For the special case of $w_j = 1$ for all $j = 1, 2, \ldots, n$, it follows that the GrA has a polynomial time complexity.

4.3.1 Proof of NP-Hardness for Problem $1(nd)||\max \sum w_j T_j$

In this section, we give a polynomial reduction from the partition problem to a special case of problem $1(nd)||\max \sum w_j T_j$.

Partition problem: Given is a set $N = \{b_1, b_2, \ldots, b_n\}$ of numbers $b_1 \geq b_2 \geq \ldots \geq b_n > 0$ with $b_i \in Z_+$, $i = 1, 2, \ldots, n$. Does there exist a subset

$N' \subset N$ such that

$$\sum_{i \in N'} b_i = A = \frac{1}{2} \sum_{i=1}^{n} b_i ?$$

Without loss of generality we assume that $n > 3$ and $\sum_{i=1}^{n} b_i > 10$.

Given an instance of the partition problem, we construct the following instance of problem $1(nd)||\max \sum w_j T_j$:

$$\begin{cases} w_{2i} = M^i, \ i = 1, 2, \ldots, n, & (10.1) \\ w_{2i-1} = w_{2i} + b_i, \ i = 1, 2, \ldots, n, & (10.2) \\ p_{2i} = \sum_{j=1}^{i-1} w_{2j} + \frac{1}{2} \sum_{j \in N \setminus \{i\}} b_j, \ i = 1, 2, \ldots, n, & (10.3) \\ p_{2i-1} = p_{2i} + b_i, \ i = 1, 2, \ldots, n, & (10.4) \\ d_{2i} = d_{2i-1} = P - \sum_{j=i}^{n} p_{2j}, \ i = 1, 2, \ldots, n, & (10.5) \end{cases} \tag{10}$$

where $M = (n \sum_{i=1}^{n} b_i)^{10}$ and $P = \sum_{j=1}^{2n} p_j$.

We renumber the jobs of set $\bar{N} = \{1, 2, \ldots, 2n\}$ as

$$V_1, V_2, V_3, V_4, \ldots, V_{2i-1}, V_{2i}, \ldots, V_{2n-1}, V_{2n}.$$

Let $(V_{n,1}, V_{n-1,1}, \ldots, V_{i,1}, \ldots, V_{1,1}, V_{1,2}, \ldots, V_{i,2}, \ldots, V_{n-1,2}, V_{n,2})$ be a *canonical schedule*, where $\{V_{i,1}, V_{i,2}\} = \{V_{2i-1}, V_{2i}\}, \ i = 1, 2, \ldots, n$.

Lemma 2 *For case (10), all optimal schedules are canonical schedules, or they can be reduced to a canonical schedule if the LPT (longest processing time order) rule is applied to the first n jobs.*

Proof.
1) First, we prove that one of the jobs V_{2n} and V_{2n-1} is tardy in any optimal schedule. Suppose that both jobs are on-time in an optimal schedule $\pi = (\pi_1, V_{2n}, \pi_2, V_{2n-1}, \pi_3, \pi_4)$, where only jobs from set π_4 are tardy. We note that we consider only optimal schedules of the type $\pi = (G, H)$, where all jobs $j \in H$ are tardy and all jobs $i \in G$ are on-time (see Lemma 3 in Section 4.3.2). For the schedule $\pi' = (\pi_1, \pi_2, V_{2n-1}, \pi_3, \pi_4, V_{2n})$, we have

$$\sum_{j=1}^{n} w_j T_j(\pi') - \sum_{j=1}^{n} w_j T_j(\pi) \geq w_{2n} T_{2n}(\pi') - p_{2n} \sum_{j \in \pi_4} w_j \geq$$

$$\geq p_{2n} M^n - p_{2n} \sum_{i=1}^{n-1} (2M^i + b_i) = p_{2n} M^n - p_{2n} \left(2M \frac{M^{n-1} - 1}{M - 1} + \sum_{i=1}^{n-1} b_i \right) > 0.$$

Thus, schedule π is not optimal, i.e., one of the jobs V_{2n} and V_{2n-1} is tardy in any optimal schedule.

2) Now we prove that the following inequalities hold:

$$\frac{w_2}{p_2} < \frac{w_4}{p_4} < \ldots < \frac{w_{2n}}{p_{2n}}.$$

We have to prove:

$$\frac{M^{i-1}}{\sum_{j=1}^{i-2} w_{2j} + \frac{1}{2}\sum_{j\in N\setminus\{i-1\}} b_j} < \frac{M^i}{\sum_{j=1}^{i-1} w_{2j} + \frac{1}{2}\sum_{j\in N\setminus\{i\}} b_j}\ .$$

Let $B_i = \frac{1}{2}\sum_{j\in N\setminus\{i\}} b_j,\ i = 1, 2, \ldots, n$. Then, by equivalent transformations, we obtain

$$\frac{M^{i-1}}{M\frac{M^{i-2}-1}{M-1} + B_{i-1}} < \frac{M^i}{M\frac{M^{i-1}-1}{M-1} + B_i}$$

$$\iff \quad \frac{1}{\frac{M(M^{i-2}-1)+B_{i-1}(M-1)}{M-1}} < \frac{M}{\frac{M(M^{i-1}-1)+B_i(M-1)}{M-1}}$$

$$\iff \quad M(M^{i-1}-1) + B_i(M-1) < M[M(M^{i-2}-1) + B_{i-1}(M-1)]$$

$$\iff \quad 0 < M^2(B_{i-1}-1) - M(B_{i-1}+B_i-1) + B_i.$$

The latter inequality is true since $M^2 > M\cdot 2\sum_{j=1}^{n} b_j$ and $(B_{i-1}-1) > 1$. Thus, the above inequalities hold. Analogously, we can prove that

$$\frac{w_{2(i-1)-1}}{p_{2(i-1)-1}} < \frac{w_{2i}}{p_{2i}}, \quad \frac{w_{2(i-1)}}{p_{2(i-1)}} < \frac{w_{2i-1}}{p_{2i-1}} \quad \text{and} \quad \frac{w_{2(i-1)-1}}{p_{2(i-1)-1}} < \frac{w_{2i-1}}{p_{2i-1}}$$

for each $i = 2, 3, \ldots, n$.

Then, one of the jobs V_{2n} and V_{2n-1} is the last tardy job in any optimal schedule (see Lemma 3 in Section 4.3.2). In the following, we consider only optimal schedules of the type $(V_{n,1}, \pi_\alpha, V_{n,2})$, where $\{V_{n,1}, V_{n,2}\} = \{V_{2n-1}, V_{2n}\}$.

3) Analogously to 1) and 2), we can prove that for each $i = n-1, n-2, \ldots, 1$, one of the jobs V_{2i} and V_{2i-1} is tardy in any optimal schedule.

Thus, the lemma is true.

□

Theorem 2 *Problem* $1(nd)||\max\sum w_jT_j$ *is NP-hard in the ordinary sense.*

Proof. For the schedule $\pi = (V_{2n-1}, V_{2(n-1)-1}, \ldots, V_3, V_1, V_2, V_4, \ldots, V_{2(n-1)}, V_{2n})$, we have the objective function value

$$F(\pi) = \sum_{j=1}^{n} w_{2j} T_{2j}(\pi) = \sum_{j=1}^{n} w_{2j} p_{2j}.$$

Now we consider a canonical schedule

$$\pi' = (V_{n,1}, V_{n-1,1}, \ldots, V_{i,1}, \ldots, V_{1,1}, V_{1,2}, \ldots, V_{i,2}, \ldots, V_{n-1,2}, V_{n,2}).$$

In addition, let us denote

$$x_i = \begin{cases} 1, & \text{if } V_{i,2} = V_{2i-1}, \\ 0, & \text{if } V_{i,2} = V_{2i}. \end{cases}$$

Then we have

$$\begin{aligned}
F(\pi') &= F(\pi) + \sum_{i=1}^{n} x_i \left[(w_{2i-1} - w_{2i}) \cdot T_{V_{2i}}(\pi) - (p_{2i-1} - p_{2i}) \left(\sum_{j=1}^{i-1} w_{V_{j,2}} \right) \right] \\
&= F(\pi) + \sum_{i=1}^{n} x_i \left[b_i \cdot p_{2i} - b_i \cdot \left(\sum_{j=1}^{i-1} w_{V_{j,2}} \right) \right] \\
&= F(\pi) + \sum_{i=1}^{n} x_i \left[b_i \cdot p_{2i} - b_i \cdot \left(\sum_{j=1}^{i-1} w_{2j} + \sum_{j=1}^{i-1} x_j b_j \right) \right] \\
&= F(\pi) + \sum_{i=1}^{n} x_i b_i \left[p_{2i} - \frac{1}{2} \sum_{j=i+1}^{n} x_j b_j - \sum_{j=1}^{i-1} w_{2j} - \frac{1}{2} \sum_{j=1}^{i-1} x_j b_j \right] \\
&= F(\pi) + \frac{1}{2} \sum_{i=1}^{n} x_i b_i \left(\sum_{j \in N \setminus \{i\}} b_j - \sum_{j \in N \setminus \{i\}} x_j b_j \right) \\
&= F(\pi) + \frac{1}{2} \sum_{i=1}^{n} \sum_{j \in N \setminus \{i\}} x_i \cdot b_i \cdot b_j \cdot (1 - x_j).
\end{aligned}$$

If and only if there exists a subset $N' \subset N$ for the instance of partition problem such that $\sum_{i \in N'} b_i = A$, then for an optimal canonical schedule π^*,

we have

$$F(\pi^*) = F(\pi) + \frac{1}{2}A^2,$$

where $x_i = 1$ if $i \in N'$ since

$$F(\pi^*) = F(\pi) + \frac{1}{2}\sum_{i\in N'}\sum_{j\in N\setminus N'} b_i \cdot b_j = F(\pi) + \frac{1}{2}A \cdot A.$$

If there is no solution for the instance of the partition problem, then

$$\begin{aligned} F(\pi^*) &= F(\pi) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j\in N\setminus\{i\}} x_i \cdot b_i \cdot b_j \cdot (1 - x_j) \\ &= F(\pi) + \frac{1}{2}(A - y)(A + y) \\ &= F(\pi) + \frac{1}{2}A^2 - \frac{1}{2}y^2, \end{aligned}$$

where $y > 0$.

□

In addition, in the following section, a pseudo-polynomial solution algorithm for problem $1(nd)||\max\sum w_jT_j$ is presented. Thus, problem $1(nd)||\max\sum w_jT_j$ is NP-hard in the ordinary but not in the strong sense.

As a consequence, we can prove that problem $1(sa)|r_j|\max\sum w_jT_j$, the complexity status of which was open [12], is NP-hard since the special case $1(sa)|r_j = 0|\max\sum w_jT_j$ is equivalent to $1(nd)||\max\sum w_jT_j$, where *sa* means that we consider only semi-active schedules as feasible solutions.

4.3.2 Solution Algorithms for Problem $1(nd)||\max\sum w_jT_j$

First, we present a property of an optimal schedule and an exact algorithm for the problem under consideration.

Lemma 3 *For problem $1(nd)||\max\sum w_jT_j$, there exists an optimal schedule $\pi = (G, H)$, where all jobs $j \in H$ are tardy and all jobs $i \in G$ are on-time. All jobs from the set G are processed in non-increasing order of values $\frac{w_j}{p_j}$ and all jobs from the set H are processed in non-decreasing order of values $\frac{w_j}{p_j}$.*

As a consequence, we obtain the following corollary.

Corollary 1 *For the problem* $1(nd)||\max\sum T_j$, *there exists an optimal schedule* $\pi = (G, H) = (SPT, LPT)$, *where all jobs* $j \in H$ *are tardy and all jobs* $i \in G$ *are on-time. All jobs from set* G *are processed in SPT (shortest processing time) order and all jobs from set* H *are processed in LPT (longest processing time) order.*

The proof of Corollary 1 has been presented in [11]. Lemma 3 can be proved analogously (or see [4]).

For problem $1(nd)||\max\sum w_jT_j$, we present the following pseudo-polynomial algorithm based on Lemma 3.

Algorithm 2

1. Number the jobs such that $\frac{w_1}{p_1} \leq \frac{w_2}{p_2} \leq \ldots \leq \frac{w_n}{p_n}$;
2. $\pi_1(t) := (1)$, $f_1(t) := w_1 \max\{0, p_1 + t - d_1\}$ for all $t \in Z$ with $t \in [0, \sum_{i=2}^{n} p_i]$;
3. FOR $j := 2$ TO n DO

   FOR $t := 0$ TO $\sum_{i=j+1}^{n} p_i$ $(t \in Z)$ DO
   $\pi^1 := (j, \pi_{j-1}(t + p_j))$, $\pi^2 := (\pi_{j-1}(t), j)$;
   $\Phi^1(t) := w_j \max\{0, p_j + t - d_j\} + f_{j-1}(t + p_j)$;
   $\Phi^2(t) := f_{j-1}(t) + w_j \max\left\{0, \sum_{i=1}^{j} p_i + t - d_j\right\}$;
   If $\Phi^1(t) > \Phi^2(t)$ then $f_j(t) := \Phi^1(t)$ and $\pi_j(t) := \pi^1$,
   else $f_j(t) := \Phi^2(t)$ and $\pi_j(t) := \pi^2$;
4. $\pi_n(0)$ is an optimal schedule with the objective function value $f_n(0)$.

$\pi_j(t)$ represents the best partial schedule of the jobs $1, 2, \ldots, j$ when the first job starts at time t, and $f_j(t)$ denotes the corresponding total weighted tardiness.

In fact, for this dynamic programming algorithm, we have the following functional equations:

$$f_j(t) = \max \begin{cases} \Phi^1(t) = w_j \max\{0, p_j + t - d_j\} + f_{j-1}(t + p_j), & j = 1, 2, \ldots, n; \\ \Phi^2(t) = w_j \max\left\{0, \sum_{i=1}^{j} p_i + t - d_j\right\} + f_{j-1}(t), & j = 1, 2, \ldots, n. \end{cases} \tag{11}$$

where

$$f_0(t) = 0 \quad \text{for all } t \geq 0.$$

Function $\Phi^1(t)$ represents the setting $x_j = 1$ (which means that we add job j as the first job of the corresponding partial schedule of the first $j-1$ jobs) while $\Phi^2(t)$ represents the setting $x_j = 0$ (which means that job j is added as the last job to the partial schedule of the first $j-1$ jobs).

Theorem 3 *Algorithm 2 constructs an optimal schedule in $O(n \sum p_j)$ time.*

The proof of a similar theorem for problem $1(nd)||\max \sum T_j$ has been presented in [11]. Moreover, for $t \geq d_{max}$, all jobs are tardy in each partial schedule which starts at time t. Thus, we can reduce the time complexity to $O(nd_{max})$. We note that a similar algorithm for this problem with time complexity $O(n \sum p_j)$ has been presented in [4]. However, from Algorithm 2, it is easy to construct the *GrA*.

In each step of the graphical algorithm, we store function $f_j(t)$ in tabular form as given in Table 14.

Table 14: Function $f_j(t)$

t	$(-\infty, t_1]$	$(t_1, t_2]$	...	$(t_{m_j}, +\infty)$
$f_j(t)$	$b_1 = 0$	b_2	...	b_{m_j+1}
total weight of tardy jobs	$u_1 = 0$	u_2	...	u_{m_j+1}
optimal partial schedule	π_1	π_2	...	π_{m_j+1}

Note that the notations π_j from Table 14 and $\pi_j(t)$ from Algorithm 2 have a different meaning. The above data means the following. For each value $t \in (t_l, t_{l+1}]$, we have an optimal partial schedule π_{l+1} with the total weight of tardy jobs u_{l+1} and the function value $f_j(t) = b_{l+1} + (t - t_l) \cdot u_{l+1}$. In [11], it has been shown for the corresponding problem with unit weights that this table represents a continuous, piecewise-linear and convex function $f_j(t)$ which is also true for the problem under consideration. The points $t_1, t_2, \ldots, t_{m_j}$ are called the *break points* since there is a change from value u_l to u_{l+1} (which means that the slope of the piecewise-linear function changes). For describing each linear segment, we store its slope u_{l+1} and its function value b_{l+1} at point $t = t_l$. In the following, we describe how function $f_{j+1}(t)$ is determined by means of function $f_j(t)$.

Function $\Phi^1(t)$ is obtained from function $f_j(t)$ by the following operations. We shift the graph of function $f_j(t)$ to the left by the value p_{j+1} and in the table for function $f_j(t)$, we add a column which results from the new

break point $t' = d_{j+1} - p_{j+1}$. If $t_l - p_{j+1} < t' < t_{l+1} - p_{j+1},\ l+1 \leq m_j$, then in the new table for $\Phi^1(t)$, we have two new intervals of t: $(t_l - p_{j+1}, t']$ and $(t', t_{l+1} - p_{j+1}]$. Moreover, we increase the values $u_{l+1}, u_{l+2}, \ldots, u_{m_j+1}$ by w_{j+1}, i.e., the total weight of the tardy jobs (and thus the slope of the corresponding function) increases. The corresponding partial sequences π^1 are obtained by adding job $j+1$ as the first job to each previous partial sequence.

Function $\Phi^2(t)$ is obtained from function $f_j(t)$ by the following operations. In the table for $f_j(t)$, we add a column which results from the new break point $t' = d_{j+1} - \sum_{i=1}^{j+1} p_i$. If $t_l < t' < t_{l+1},\ l+1 \leq m_j$, then in the new table, we have two new intervals of t: $(t_l, t']$ and $(t', t_{l+1}]$. Moreover, we increase the values $u_{l+1}, u_{l+2}, \ldots, u_{m_j+1}$ by w_{j+1}, i.e., the total weight of tardy jobs increases. The corresponding partial sequences π^2 are obtained by adding job $j+1$ at the end to each previous partial sequence.

Now we construct a table that corresponds to the function

$$f_{j+1}(t) = \max\{\Phi^1(t), \Phi^2(t)\}.$$

We compare the intervals from both tables and search for intersection points of the graphs of functions $\Phi^1(t)$ and $\Phi^2(t)$. This step requires $O(m_j)$ operations. If in the table for function $f_{j+1}(t)$, we have the situation displayed in Table 15, we cut the column, which corresponds to interval $(t_l, t_{l+1}]$, and combine both intervals, i.e., we set $t_l := t_{l+1}$.

Table 15: Deletion of a column

t	$\ldots$	$(t_{l-1}, t_l]$	$(t_l, t_{l+1}]$	$\ldots$
$f_{j+1}(t)$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
total weight of tardy jobs	$\ldots$	u_l	$u_{l+1} = u_l$	$\ldots$
optimal partial schedule $\pi_{j+1}(t)$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

A detailed description, complexity results and a numerical example of a similar algorithm for problem $1(nd)||\max\sum T_j$ have been presented in [11]. It is obvious that in each step j, $j = 1, 2, \ldots, n$, we have at most $\min\{\sum_{i=1}^{n} p_i, d_{max}, O(2^j), \sum_{i=1}^{j} w_i\} + 1$ columns in the corresponding table. Thus, the time complexity of the *GrA* for problem $1(nd)||\max\sum w_j T_j$ is $O(\min\{2^n, n \cdot \min\{d_{max}, \sum_{j=1}^{n} w_j\}\})$. Moreover, since we have $\sum_{j=1}^{n} w_j = n$ for problem $1(nd)||\max\sum T_j$, the time complexity of the *GrA* reduces to $O(n^2)$ [11] for the latter problem.

5 Concluding Remarks

The graphical approach can be applied to problems, where a pseudo-polynomial algorithm exists and Boolean variables are used in the sense that yes/no decisions have to made (e.g. in the scheduling problems under consideration, a job may be completed on-time or not or for a knapsack problem, an item can be put into the knapsack or not). For the knapsack problem, the graphical algorithm often reduces substantially the number of points to be considered but the time complexity of the algorithm remains pseudo-polynomial. However, for the single machine problem of maximizing total tardiness, the graphical algorithm improved the complexity from $O(n \sum p_j)$ to $O(n^2)$. Thus, the graphical approach has not only a practical but also a theoretical importance.

Acknowledgements

Partially supported by DAAD (Deutscher Akademischer Austauschdienst): A/08/80442/Ref. 325.

References

[1] *Bellman R.*, *Dynamic Programming*, Princeton: Princeton Univ. Press, 1957.

[2] *Papadimitriou Ch. and Steiglitz K.*, *Combinatorial Optimization: Algorithms and Complexity*, Englewood Cliffs: Prentice Hall, 1982.

[3] *Sahni S. K., Algorithms for Scheduling Independent Jobs*, J. Assoc. Comput. Mach., N 23, 1976, 116-127

[4] *Lawler E.L., Moore J.M., A Functional Equation and its Application to Resource Allocation and Sequencing Problems*, Management Science, Vol. 16, No. 1, 1969, 77 – 84

[5] *Finkelstein Yu. Yu.*, *Approximate Methods and Applied Problems of Discrete Optimization*, Nauka, Moscow, 1976 (in Russian).

[6] *Posypkin M.A., Sigal I. Kh., Speedup estimates for some variants of the parallel implementations of the branch-and-bound method*, Computational Mathematics and Mathematical Physics, Vol. 46, N 12, 2006, 2189 –2 202.

[7] *Hoogeveen H. and T'Kindt V., Minimizing the number of late jobs when the start time of the machine is variable*, Booklet of Abstracts, PMS 2010 Conference, 235 – 238.

[8] *Lazarev A.A., Werner F., A Graphical Realization of the Dynamic Programming Method for Solving NP-Hard Combinatorial Problems*, Computers and Mathematics with Applications, 2009, Vol. 58, No. 4, 619 – 631.

[9] *Gafarov E.R., Lazarev A.A., Werner F., Algorithms for Maximizing the Number of Tardy Jobs or Total Tardiness on a Single Machine*, to appear in Automation and Remote Control, Vol. 71, No. 10, 2010.

[10] *Gafarov E.R., Lazarev A.A., Werner F., Single Machine Scheduling with Generalized Total Tardiness Objective Function*, Preprint 10/10, FMA, OvGU Magdeburg, 2010.

[11] *Gafarov E.R., Lazarev A.A., Werner F., A Polynomial Time Graphical Algorithm for Maximizing Total Tardiness on a Single Machine*, Preprint 12/10, FMA, OvGU Magdeburg, 2010.

[12] *Aloulou M.A., Kovalyov M.Y., Portmann M.-C., Evaluation Flexible Solutions in Single Machine Scheduling via Objective Function Maximization: the Study of Computational Complexity*, RAIRO Oper. Res., Vol. 41, 2007, 1 – 18.

[13] *Lazarev A.A., Werner F., Algorithms for Special Cases of the Single Machine Total Tardiness Problem and an Application to the Even-Odd Partition Problem,* Mathematical and Computer Modelling, Vol. 49, No. 9-10, 2009, 2061 – 2072.

[14] *Kellerer, H., Pferschy, U., Pisinger, D., Knapsack Problems*, New York: Springer, 2004.