A Graphical Approach to Solve Combinatorial Problems: Algorithms and Some Computational Results ⋆

Evgeny R. Gafarov, * **Alexander A. Lazarev,** **
Frank Werner ***

* *Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997 Moscow, Russia, (e-mail: axel73@mail.ru.)*
** *Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997 Moscow, Russia, (e-mail: jobmath@mail.ru)*
*** *Fakultät für Mathematik, Otto-von-Guericke-Universität, PSF 4120, 39016 Magdeburg, Germany, (e-mail: frank.werner@ovgu.de)*

Abstract: In this paper, we present a modification of dynamic programming algorithms (*DPA*), which we denote as graphical algorithms (*GrA*). For 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. For some problems, *GrA* has a polynomial time complexity in contrast to a pseudo-polynomial complexity of a *DPA*.

Keywords: Dynamic programming, Optimization, Scheduling algorithms, Graphical algorithm, Single machine, Algorithms.

1. INTRODUCTION

Dynamic programming is a general optimization technique developed by Bellman (1957). It is a recursive optimization procedure which interprets an optimization problem as a multi-step decision process. 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, 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 optimization 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*. In contrast to a classical *DPA*, it can also treat problems with non-integer data without necessary transformations of the corresponding data. Sometimes, 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, *DPA* with the same idea like that used in a *GrA* are known (see e.g. Sahni (1976)). 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 Sahni (1976), this modification is also useful for some scheduling problems, where the starting time is variable and one wishes to find all Pareto-optimal solutions (see Section 3.2).

⋆ Partially supported by RFBR (Russian Foundation for Basic Research): 11-08-01321, 11-08-13121 and DAAD (Deutscher Akademischer Austauschdienst): A/08/80442/Ref. 325.

2. BASIC IDEA OF THE GRAPHICAL ALGORITHM

Usually in a *DPA*, 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 the 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 Bellman's recursive equations in a *DPA* for a minimization problem:

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

with the initial conditions

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

In (1), function $\Phi^1(t)$ characterizes the setting $x_j = 1$ while $\Phi^2(t)$ characterizes the 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
$f_j(t)$	val_0	val_1	val_2	...	val_y	...	val_A
OPS	$X(0)$	$X(1)$	$X(2)$	...	$X(y)$	...	$X(A)$

Here $X(t), t = 0, 1, \ldots, A$, is a vector which describes an optimal partial solution (OPS) and which consists of j numbers $x_1, x_2, \ldots, x_j \in \{0, 1\}$. This data can also be stored in a condense tabular form as given in Table 2.

Table 2: Computations in *GrA*

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

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

$$\{t^1_0, t^1_1, t^1_2, \ldots, t^1_{(m_j+\mu_1)-1}, t^1_{(m_j+\mu_1)},$$
$$t^2_0, t^2_1, t^2_2, \ldots, t^2_{(m_j+\mu_2)-1}, t^2_{(m_j+\mu_2)}\},$$

and we determine the intersection points $t^3_1, t^3_2, \ldots, t^3_{\mu_3}$. 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 *DPA*. Since we have to save in memory only one Table 2, which is changed after each step, the size of memory used does not exceed $O(n \min\{A, M\})$.

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, one can usually solve considerably larger instances with the *GrA*.
3. Properties of an optimal solution can be taken into account, and sometimes the *GrA* has even a polynomial time complexity, or we can at least essentially reduce the complexity of the standard *DPA* (see below).

3. GRAPHICAL ALGORITHMS FOR SINGLE MACHINE SCHEDULING PROBLEMS

In this section, we present *GrA* 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 sequence 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 sequence π and 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 sequence π^* that minimizes the value $\sum_{j=1}^{n} w_j U_j(\pi)$ and for

the problem of maximizing total tardiness $1(nd)||\max\sum T_j$ Lawler et al. (1969), Gafarov et al. (2010b), the objective is to find an optimal sequence 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. In the case of the single machine generalized total tardiness minimization problem $1||\sum GT_j$, we wish to minimize the value $\sum_{j=1}^{n} GT_j(\pi)$ (see Gafarov et al. (2010a)).

In Table 3, we summarize the time complexity of existing and new *GrA* for particular scheduling problems, where $d_{max} = \max_{j\in N}\{d_j\}$.

Table 3: Time complexity of *GrA* and *DPA*

Problem	*GrA*	*DPA*
$1\|\|\sum w_jU_j$	$O(\min\{2^n, n \cdot \min\{d_{max}, F_{opt}\}\})$ Gafarov et al. (2010a)	$O(nd_{max})$
$1\|d_j = d'_j + A\|\sum U_j$ Hoogeveen et al. (2010)	$O(n^2)$	$O(n\sum p_j)$
$1\|\|\sum GT_j$	$O(\min\{2^n, n \cdot d_{max}\})$ Gafarov et al. (2010a)	$O(nd_{max})$
$1\|\|\sum T_j$ case B-1	$O(\min\{2^n, n \cdot d_{max}\})$ Lazarev et al. (2009)	$O(nd_{max})$
$1(nd)\|\|\max\sum w_jT_j$	$O(\min\{2^n, n \cdot \min\{d_{max}, \sum w_j\}\})$	$O(nd_{max})$
$1(nd)\|\|\max\sum T_j$	$O(n^2)$ Gafarov et al. (2010b)	$O(nd_{max})$

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

3.1 A GrA 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 sequence $\pi = (G, H)$, 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 EDD (earliest 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 arbitrarily scheduled. Now we present a solution algorithm for the equivalent problem of maximizing the total weight of the on-time jobs which 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 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. (Gafarov et al. (2010a)) Algorithm 1 constructs an optimal schedule for problem $1||\sum w_jU_j$ in $O(n\sum p_j)$ time.

For this *GrA*, we have the following functional equations:

$$f_j(t) = \max\begin{cases} \Phi^1(t) = \alpha(t) + f_{j-1}(t + p_j), & j = \overline{1,n}; \\ \Phi^2(t) = \beta(t) + f_{j-1}(t), & j = \overline{1,n}. \end{cases} \quad (3)$$

where $f_0(t) = 0$ for $t \geq 0$.

$\Phi^1(t)$ represents the setting $x_j = 1$ (job j is added to the beginning of the partial sequence) while $\Phi^2(t)$ represents the setting $x_j = 0$ (job j is added 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 sequence $\pi_j(t)$ and the partial sequence $\pi^2 := (\pi_{j-1}(t), j)$ 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 *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 4, where $t_1 < t_2 < \ldots < t_{m_j}, W_1 > W_2 > \ldots > W_{m_j}$ and OPS denotes an optimal partial sequence.

Table 4: Function $f_j(t)$

t	t_1	t_2	$\ldots$	t_{m_j}
$f_j(t)$	W_1	W_2	$\ldots$	W_{m_j}
OPS	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 sequence $\pi_l = (G, H)$ 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 5, we cut the column corresponding to point t_{l+1}, combine both intervals, and we can use $\pi_l := \pi_{l+1}$.

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

t	...	t_l	t_{l+1}	...
$f_{j+1}(t)$	...	W_l	$W_{l+1} = W_l$	...
OPS	...	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 $O(\min\{2^n, n \cdot \min\{d_{max}, F_{opt}\}\})$.

3.2 A GrA 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 equivalent, 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 Hoogeveen et al. (2010), an exact algorithm for this problem with time complexity $O(n^4)$ has been proposed. Next, we present an algorithm with time complexity $O(n^2)$. First, we construct a modification of Algorithm 1, where $w_j = 1$ for $j = 1, 2, \ldots, n$, 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 the *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) sequence. Assume that in the last step of the *GrA*, we have obtained the data given in Table 6.

Table 6: Function $f_n(t)$

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

This means that we have $n+1$ Pareto-optimal solutions described by the following 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 $1|d_j = d'_j + G|\sum w_j U_j$ but the time complexity of this *GrA* is pseudo-polynomial.

3.3 A GrA for Maximizing Weighted Total Tardiness on a Single Machine

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 is NP-hard (see Gafarov et al. (2010b)). First, we present a property of an optimal schedule.

Lemma 2. For problem $1(nd)||\max\sum w_j T_j$, 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 $\pi = (G, H)$, 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 Gafarov et al. (2010b). Lemma 2 can be proved analogously (or see Lawler et al. (1969)). For problem $1(nd)||\max\sum w_j T_j$, we present the following pseudo-polynomial algorithm based on Lemma 2.

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
   $\pi^1 := (j, \pi_{j-1}(t + p_j))$, $\pi^2 := (\pi_{j-1}(t), j)$;
   $\Phi^1(t) := w_j \max\{0, p_j + t - d_j\} + f_{j-1}(t + p_j)$;
   $\Phi^2(t) := f_{j-1}(t) + w_j \max\{0, \sum_{i=1}^{j} p_i + t - d_j\}$;
   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 sequence with the objective function value $f_n(0)$.

$\pi_j(t)$ represents the best partial sequence 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. For this *DPA*, we have the following functional equations:

$$f_j(t) = \max \begin{cases} \Phi^1(t) = w_j \max\{0, p_j + t - d_j\}, \\ +f_{j-1}(t + p_j) \quad j = \overline{1,n}; \\ \Phi^2(t) = w_j \max\{0, \sum_{i=1}^{j} p_i \\ +t - d_j\} + f_{j-1}(t), \quad j = \overline{1,n}. \end{cases} \tag{4}$$

where $f_0(t) = 0 \quad$ for all $t \geq 0$.

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

Theorem 2. 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 Gafarov et al. (2010b). 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 $O(nd_{max})$. A similar algorithm for this problem with time complexity $O(n \sum p_j)$ has been presented in Lawler et al. (1969). However, from Algorithm 2, it is easy to construct the *GrA*. In each step of the *GrA*, we store function $f_j(t)$ in tabular form as given in Table 7, where TWTJ denotes the total weight of tardy jobs and $u_1 < u_2 < \ldots < u_{m_j+1}$.

Table 7: Function $f_j(t)$

t	$(-\infty, t_1]$	$(t_1, t_2]$	$\ldots$	$(t_{m_j}, +\infty)$
$f_j(t)$	$b_1 = 0$	b_2	$\ldots$	b_{m_j+1}
TWTJ	$u_1 = 0$	u_2	$\ldots$	u_{m_j+1}
OPS	π_1	π_2	$\ldots$	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 sequence 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 Gafarov et al. (2010b), 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 8, 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 8: Deletion of a column

t	$\ldots$	$(t_{l-1}, t_l]$	$(t_l, t_{l+1}]$	$\ldots$
$f_{j+1}(t)$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
TWTJ	$\ldots$	u_l	$u_{l+1} = u_l$	$\ldots$
OPS $\pi_{j+1}(t)$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

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 Gafarov et al. (2010b). 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 $O(n^2)$ (Gafarov et al. (2010b)) in this case.

4. COMPUTATIONAL RESULTS

For problem $1||\sum w_j U_j$, we have run two sets of instances for testing the graphical variant of Algorithm 1. The first set is as follows: The processing times are randomly generated from the interval $[p_{\min}, p_{\max}]$, the weights are randomly generated from $[1, w_{\max}]$, and the due dates are randomly generated from $[p_j, p_j + m_{\max}]$. The following values of the parameters were used:

$(p_{\min}, p_{\max})$:	(0,100), (25,75)
$w_{\max}$:	1, 10, 100
$m_{\max}$:	50, 200, 350, 500, 650

The second set is generated as follows. The processing times are randomly generated from the interval $[0, 100]$, the weights are randomly generated from $[1, w_{\max}]$, and the due dates are randomly generated from $[p_j, p_j + Kn]$. The following values of parameters were used:

$w_{\max}$:	10, 99
K:	1, 5, 10, 20

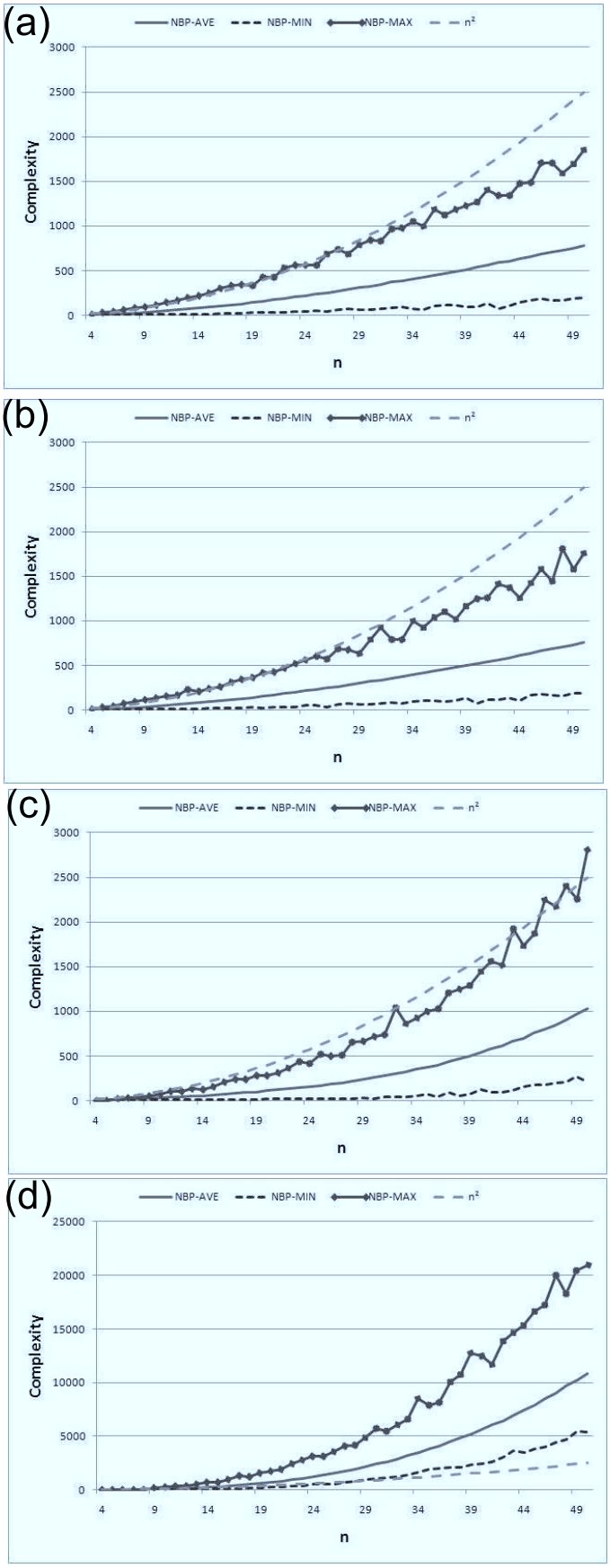

Fig. 1. Computational Results

In both cases, for each combination of the parameters and $n \in \{4, 5, \ldots, 50\}$, a series of 2500 test instances has been generated.

For each instance, we have computed the minimal (NBP-MIN), average (NBP-AVE) and maximal (NBP-MAX) number of break points. Fig. 1(a) and Fig. 1(b) present the results for two representative combinations of the first set, and Fig. 1(c) and Fig. 1(d) present the results for two representative combinations of the second set. For the instances of the first type, NBP only moderately increases, and NBP-AVE is lower than 1000 even for the large problems. Fig. 1(c) and Fig. 1(d) demonstrate the influence of the parameter K on NBP. For $K = 10$, NBP is roughly ten times as large as for $K = 1$.

As expected, the lengths of the intervals considered has the strongest influence on NBP. The largest NBP were observed for instances with $m_{max} = 650$ in the first set and for instances with $K = 20$ in the second set. We also have observed that in the first set of instances in the case of $w_{max} = 1$ ($w_j = 1$ for $j \in N$), NBP may substantially differ from those of the other test series. For instance, NBP-AVE in the series with $w_{max} = 1, m_{max} = 50$ is much smaller than NBP-AVE for the case $w_{max} = 10, m_{max} = 50$. When m_{max} is large, e.g. when $m_{max} = 500$ or $m_{max} = 650$, NBP in the series with $w_{max} = 1$ is considerably smaller than in the series with other values of w_{max}.

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. For the single machine problem of maximizing total tardiness, the *GrA* 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.

REFERENCES

R. Bellman. Dynamic Programming. Princeton: Princeton Univ. Press, 1957.

S.K. Sahni. Algorithms for scheduling independent jobs. J. Assoc. Comput. Mach., vol. 23, 116–127, 1976.

E.L. Lawler, J.M. Moore. A functional equation and its application to resource allocation and sequencing problems. Management Sci., vol. 16, No. 1, 77–84, 1969.

H. Hoogeveen, V. T'Kindt. Minimizing the number of late jobs when the start time of the machine is variable. Proceedings PMS 2010, 235–238, 2010.

E.R. Gafarov, A.A. Lazarev and F. Werner. A note on a single machine scheduling problem with generalized total tardiness objective function. Information Processing Letters, vol. 112, No. 3, 72 – 76, 2012.

E.R. Gafarov, A.A. Lazarev and F. Werner. Transforming a pseudo-polynomial algorithm for the single machine total tardiness maximization problem into a polynomial one. Annals of Operations Research, DOI 10.1007/s10479-011-1055-4, 2012, in print.

A.A. Lazarev, F. Werner. Algorithms for special cases of the single machine total tardiness problem and an application to the even-odd partition problem. Math. Comp. Modelling, vol. 49, No. 9-10, 2061–2072, 2009.