Algorithmica (2010) 57: 769–795
DOI 10.1007/s00453-008-9248-1

Fully Polynomial Approximation Schemes for a Symmetric Quadratic Knapsack Problem and its Scheduling Applications

Hans Kellerer · Vitaly A. Strusevich

Received: 27 October 2005 / Accepted: 22 October 2008 / Published online: 11 November 2008

Abstract We design a fully polynomial-time approximation scheme (FPTAS) for a knapsack problem to minimize a symmetric quadratic function. We demonstrate how the designed FPTAS can be adopted for several single machine scheduling problems to minimize the sum of the weighted completion times. The applications presented in this paper include problems with a single machine non-availability interval (for both the non-resumable and the resumable scenarios) and a problem of planning a single machine maintenance period; the latter problem is closely related to a single machine scheduling problem with two competing agents. The running time of each presented FPTAS is strongly polynomial.

Keywords Quadratic knapsack · Single machine scheduling · Total weighted completion time · Availability constraints · Scheduling agents · FPTAS

1 Introduction

In this paper, we study a Boolean quadratic programming problem of the following structure

H. Kellerer (✉)
Institut für Statistik und Operations Research, Universität Graz, Universitätsstraße 15, 8010 Graz, Austria
e-mail: hans.kellerer@uni-graz.at

V.A. Strusevich
School of Computing and Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, Greenwich, London SE10 9LS, UK
e-mail: V.Strusevich@greenwich.ac.uk

$$\begin{aligned}
\text{Minimize}\quad & Z=\sum_{1\le i<j\le n}\alpha_i\beta_j x_i x_j+\sum_{1\le i<j\le n}\alpha_i\beta_j(1-x_i)(1-x_j)\\
&\quad+\sum_{j=1}^{n}\mu_j x_j+\sum_{j=1}^{n}\nu_j(1-x_j)+\Gamma\\
\text{Subject to}\quad & \sum_{j=1}^{n}\alpha_j x_j\le A\\
& x_j\in\{0,1\},\ j=1,2,\dots,n.
\end{aligned}\tag{1}$$

We call this problem the *Symmetric Quadratic Knapsack Problem*, or problem (SQKP). Here all coefficients $\alpha_j, \beta_j, \mu_j, \nu_j$, $j=1,2,\dots,n$, and Γ are non-negative integers. We call the problem *symmetric* because both the quadratic and the linear parts of the objective function are separated into two terms, one depending on the variables x_j, and the other depending on the variables $(1-x_j)$. An important feature of our problem is that the coefficients α_j in the linear constraint are the same as in the quadratic terms of the objective function. We can view the value α_j as the weight of item j, $1\le j\le n$, i.e., $x_j=1$ means that item j is placed into the knapsack of a total weight at most A, while $x_j=0$ means that the corresponding item is not placed into the knapsack.

Notice that the coefficients of the quadratic terms of the objective function correspond to the upper triangle of the matrix with the elements $A_{ij}=\alpha_i\beta_j$.

In general, the quadratic knapsack problem (QKP) is NP-hard in the strong sense. See Chap. 12 of the book by Kellerer et al. [7] and a recent survey [14] by Pisinger for an overview of principal results on the QKP.

As applications of problem (SQKP), in this paper we consider single machine scheduling models with *machine availability constraints*. In all problems, the jobs of set $N=\{1,2,\dots,n\}$ have to be processed on a single machine. The processing of job $j\in N$ takes p_j time units. There is a positive weight w_j associated with job j, which indicates its relative importance. The completion time of job $j\in N$ in a feasible schedule S is denoted by $C_j(S)$ or just C_j. There is an additional restriction that the machine is not available for $\Delta>0$ time units.

We consider two types of machine non-availability intervals, fixed and floating. The model with *fixed* intervals is the most studied in this area of scheduling, see a recent survey by Lee [12] for a state-of-the-art review of deterministic scheduling under availability constraints. A fixed non-availability interval can be due to some scheduled activity other than processing (a scheduled maintenance period, a rest period, pre-scheduled jobs, etc.). In any case, a fixed non-availability interval occupies the time interval $[s,t]$, where $t=s+\Delta$, during which the machine cannot perform the processing of any job. The job that is affected by the non-availability interval is called the *crossover* job. There are several possible scenarios to handle the crossover job. Under the *non-resumable* scenario, the crossover job that cannot be completed by time s is restarted from scratch at time t. Under the *resumable* scenario the crossover job is interrupted at time s and resumed from the point of interruption at time t. The objective is to minimize the total weighted completion time.

Extending standard scheduling notation, we denote the resulting problem under the non-resumable scenario by $1|h(1), N-res|\sum w_jC_j$, and that under the resumable scenario by $1|h(1), Res|\sum w_jC_j$. Both problems are NP-hard in the ordinary sense, see [1] and [11]. See Sect. 3 for a detailed review of relevant results.

In another problem that we address in this paper, a *floating* machine non-availability interval can be interpreted as a maintenance period (MP) of length Δ, and the decision-maker may choose when to start the MP, provided that it is completed no later than a given deadline d. We denote this problem by $1|C_{MP} \le d|\sum w_jC_j$, where C_{MP} denotes the completion time of the maintenance period. This problem is closely related to a single machine problem with two competing agents, A and B. Agent A wants to minimize the sum of the completion times of its jobs, while Agent B will only accept schedules in which all its jobs are completed before a given deadline d. The latter problem is considered in [2] and proved *NP*-hard. See Sect. 4 for more details.

The fact that the problems under consideration are NP-hard calls for design and analysis of approximation algorithms. Recall that for a problem of minimizing a function $Z(x)$, where x is a collection of decision variables, a polynomial-time algorithm that finds a feasible solution x^H such that $Z(x^H)$ is at most $\rho \ge 1$ times the optimal value $Z(x^*)$ is called a ρ-*approximation* algorithm; the value of ρ is called a *worst-case ratio bound*. If a problem admits a ρ-approximation algorithm it is said to be *approximable* within a factor ρ. A family of ρ-approximation algorithms is called a *fully polynomial-time approximation scheme*, or an *FPTAS*, if $\rho = 1+\varepsilon$ for any $\varepsilon > 0$ and the running time is polynomial with respect to both the length of the problem input and $1/\varepsilon$.

Notice that in general the quadratic knapsack problem (QKP) is NP-hard in the strong sense, i.e., admits no FPTAS unless $\mathcal{P} = \mathcal{NP}$.

In this paper, we show that problem (SQKP) is solvable by a pseudopolynomial-time dynamic programming (DP) algorithm, and based on that fact design an FPTAS. One of the conditions of the existence of an FPTAS for problem (SQKP) is a polynomial-time algorithm which either behaves as a constant-ratio approximation algorithm or delivers a worst-case bound that depends polynomially on the length of the input of the problem. To the best of our knowledge, such an approximation algorithm is not known for the general problem (SQKP). Still, the developed FPTAS can be applied to the scheduling problems under consideration due to their special structure.

For problem $1|h(1), Res|\sum w_jC_j$, Wang et al. present a 2-approximation algorithm that requires $O(n^2)$ time, see [17], while neither for problem $1|h(1), N-res|\sum w_jC_j$, nor for problem $1|C_{MP} \le d|\sum w_jC_j$ a constant-ratio approximation algorithm has been reported so far.

The remainder of this paper is organized as follows. Section 2 is devoted to designing an FPTAS for problem (SQKP). In Sect. 3 we address the scheduling problems with a fixed non-availability interval. We obtain an FPTAS for problem $1|h(1), N-res|\sum w_jC_j$ by formulating it as an instance of problem (SQKP) and by giving a very simple 4-approximation algorithm. For its resumable counterpart, problem $1|h(1), Res|\sum w_jC_j$, we show how to modify the FPTAS for problem (SQKP) to be able to handle the crossover job that is split by the non-availability interval.

Finally, we show how an FPTAS for problem $1|h(1), Res|\sum w_j C_j$ can be adopted to handle problem $1|C_{MP} \le d|\sum w_j C_j$. It should be stressed that the running time of each presented FPTAS is strongly polynomial, i.e., polynomially depends on n and $1/\varepsilon$ only and does not depend on the size of the coefficients. Section 5 contains concluding remarks.

2 Symmetric Quadratic Knapsack Problem

The goal of this section is to demonstrate that the symmetric quadratic knapsack problem under certain conditions admits an FPTAS. We describe dynamic programming algorithms for solving problem (SQKP) and show how to convert these algorithms into an FPTAS.

It should be noted that if the knapsack constraint in (1) is removed, the problem becomes close to a Boolean programming problem known as the *half-product*, independently introduced by Kubiak [9] and by Badics and Boros [3]. The latter problem admits an FPTAS and has a number of scheduling applications. Especially relevant to our problem (SQKP) are the positive half-product and the symmetric ordered half-product recently studied in [6] and [10], respectively.

2.1 Dynamic Programming

Consider problem (SQKP) written in the form (1). We develop dynamic programming (DP) algorithms for its solution.

We give two versions of the DP algorithm. The decision variables are scanned in the order of their numbering and are given either the value of 1 (an item is put into the knapsack) or 0 (an item is not put into the knapsack).

Define

$$A_k = \sum_{j=1}^{k} \alpha_j, \quad k = 1, 2, \dots, n$$

and suppose that the values $x_1, x_2, \dots, x_k$ have been assigned. One version of our DP algorithm deals with partial solutions associated with states of the form

$$(k, Z_k, y_k),$$

where

k is the number of the assigned variables;
Z_k is the current value of the objective function (temporarily ignoring the constant Γ);
$y_k := \sum_{j=1}^{k} \alpha_j x_j$, the total weight of the items put into the knapsack.

Let us call the states of the form (k, Z_k, y_k) the *primal* states, and let the DP algorithm that manipulates with the primal sates be called the *primal* algorithm.

For our purposes, we also need another form of the DP algorithm that manipulates with the states of the dual form $(k, Z_k, \widetilde{y}_k)$, where k and Z_k have the same meaning as above, while $\widetilde{y}_k = A_k - y_k$. It is clear that $\widetilde{y}_k$ is the total weight of the considered items that have not been put into the knapsack.

We now give a formal statement and implementation details of the primal DP algorithm.

Algorithm PDP

1. Start with the initial state $(0, Z_0, y_0) = (0, \Gamma, 0)$. Compute the values $A_k = \sum_{j=1}^{k} \alpha_j, k = 1, 2, \dots, n$.
2. For all k from 0 to $n-1$ do
   (a) Make transitions from each stored state of the form
   $$(k, Z_k, y_k) \tag{2}$$
   into the states of the form
   $$(k+1, Z_{k+1}, y_{k+1}) \tag{3}$$
   by assigning the next variable x_{k+1}.
   (b) Define $x_{k+1} = 1$, provided that item $k+1$ fits into the knapsack, i.e., if the inequality $y_k + \alpha_{k+1} \le A$ holds. If feasible, the assignment $x_{k+1} = 1$ changes a state (2) to a state of the form (3), where
   $$Z_{k+1} = Z_k + \beta_{k+1} y_k + \mu_{k+1}, \qquad y_{k+1} = y_k + \alpha_{k+1}. \tag{4}$$
   (c) Define $x_{k+1} = 0$, which is always feasible. This assignment changes a state of the form (2) into the state of the form (3) such that
   $$Z_{k+1} = Z_k + \beta_{k+1}(A_k - y_k) + \nu_{k+1}; \qquad y_{k+1} = y_k. \tag{5}$$
3. Output the optimal value of the function that corresponds to the smallest value of Z_n among all found states of the form (n, Z_n, y_n).

If implemented in a straightforward way, so that all states created in an iteration are kept, Algorithm PDP requires $O(nZ^{UB}A)$ time, where Z^{UB} denotes an upper bound on the optimal value of the objective function.

However, this running time can be reduced if we do not store those states that may not lead to an optimal solution. Comparing two states (k, Z'_k, y_k) and (k, Z''_k, y_k) such that $Z'_k < Z''_k$, notice that both states created in iteration $k+1$ based on state (k, Z'_k, y_k) will have a smaller value of the objective function than those created based on state (k, Z''_k, y_k). Thus, we may say that state (k, Z'_k, y_k) *dominates* state (k, Z''_k, y_k). This essentially means that among all states created in an iteration k we only need to keep undominated states, no more than one state for each value of y_k, $0 \le y_k \le A$, i.e., only the state (k, Z_k, y_k) such that Z_k is the smallest value of the objective function for all states (k, Z, y_k) with the same value of y_k.

The removal of the dominated states can be implemented efficiently using a bucket sorting procedure. After iteration k, all created states are scanned in an arbitrary order and a state (k, Z, y) is placed into bucket $y, 0 \le y \le A$, either if the bucket is empty or if it replaces in the bucket the state (k, Z', y) with $Z' > Z$. Thus, after each iteration at most $A+1$ states are kept. In the next iteration at most $2A+2$ states will be created and it will take $O(A)$ time to remove the dominated states. The overall time complexity of the algorithm reduces to $O(nA)$.

The optimal values of the decision variables can be found by backtracking.

The corresponding dual DP algorithm, which we will refer to as Algorithm DDP, also starts with the state $(0, \Gamma, 0)$, but manipulates with the dual states. We skip its formal description, since is it very similar to that of Algorithm PDP. It suffices to say that in iteration k, given a state

$$(k, Z_k, \widetilde{y}_k),$$

Algorithm DDP transforms it into a state

$$(k+1, Z_{k+1}, \widetilde{y}_{k+1}),$$

where for $x_{k+1}=1$ we define

$$Z_{k+1} = Z_k + \beta_{k+1}(A_k - \widetilde{y}_k) + \mu_{k+1}, \qquad \widetilde{y}_{k+1} = \widetilde{y}_k, \tag{6}$$

provided that $A_k - \widetilde{y}_k \le A$, while for $x_{k+1}=0$ we define

$$Z_{k+1} = Z_k + \beta_{k+1}\widetilde{y}_k + \nu_{k+1}, \qquad \widetilde{y}_{k+1} = \widetilde{y}_k + \alpha_{k+1}. \tag{7}$$

Similarly, using the bucket sorting for storing the undominated states in each iteration, i.e., at most one state with the smallest Z-value per each $\widetilde{y}$-value, the dual Algorithm DDP can also be run in $O(nA)$ time.

To summarize, we formulate the following statement.

Theorem 1 *Problems (SQKP) is solvable by a dynamic programming algorithm in $O(nA)$ time.*

Algorithms PDP and DDP can be converted into an FPTAS for problem (SQKP) as described in Sect. 2.2. Both versions, primal and dual, of the dynamic programming algorithm are needed for the FPTAS. As will be seen from its description and analysis, our FPTAS manipulates with the states created by both algorithms, PDP and DDP.

Notice that it is straightforward to modify the described DP algorithms to handle a more general problem in which the term $\sum_{1\le i<j\le n} \alpha_i\beta_j(1-x_i)(1-x_j)$ is replaced by $\sum_{1\le i<j\le n} \alpha_i\gamma_j(1-x_i)(1-x_j)$, where γ_j may be different from β_j. For our purposes, however, it suffices to concentrate on the situation that $\gamma_j = \beta_j$.

2.2 FPTAS

Consider problem (SQKP) with the optimal value of the objective function equal to Z^*. The dynamic programming algorithms described in the previous subsection can be converted into an FPTAS for problem (SQKP), provided that an upper bound Z^{UB} is available such that $Z^{UB}/Z^* \le \rho$, where ρ is a positive constant.

Given Z^{UB}, we derive that

$$Z_{LB} = \frac{1}{\rho} Z^{UB}$$

is a lower bound on Z^*.

Our FPTAS is based on both versions of the dynamic programming algorithms from Sect. 2.1, the primal and the dual. To reduce the number of computed function values we round the computed values up to a multiple of a chosen small number. To reduce the number of states stored after each iteration we split the range of possible y-values (and $\widetilde{y}$-values) into subintervals of a variable length and for each of the resulting subintervals we keep at most two y-values (and at most two $\widetilde{y}$-values) related to the same value of the function.

Algorithm Eps

1. Given an instance of problem (SQKP), find an upper bound Z^{UB} on the optimal value Z^* of the objective function, such that $Z^{UB}/Z^* \le \rho$.
2. Given an arbitrary $\varepsilon > 0$, define $Z_{LB} = \frac{1}{\rho} Z^{UB}$ and

$$\delta = \frac{\varepsilon Z_{LB}}{2n}.$$

3. Let there be $h \le n$ distinct values among $\beta_j,\ j = 1, 2, \dots, n$. Sort these values in decreasing order, i.e., determine a permutation $\pi = (\pi(1), \pi(2), \dots, \pi(h))$ such that

$$\beta_{\pi(1)} > \beta_{\pi(2)} > \cdots > \beta_{\pi(h)}.$$

   Split the interval $[0, \frac{Z^{UB}}{\beta_{\pi(h)}}]$ into h intervals

$$I_1 = \left[0, \frac{Z^{UB}}{\beta_{\pi(1)}}\right], I_2 = \left[\frac{Z^{UB}}{\beta_{\pi(1)}}, \frac{Z^{UB}}{\beta_{\pi(2)}}\right], \dots, I_h = \left[\frac{Z^{UB}}{\beta_{\pi(h-1)}}, \frac{Z^{UB}}{\beta_{\pi(h)}}\right].$$

   Additionally, split each interval I_j into subintervals I_j^r of length $\delta/\beta_{\pi(j)}$ (it may turn out that the last of the subintervals of an interval I_j is strictly shorter than $\delta/\beta_{\pi(j)}$).
4. Store the initial state $(0, \Gamma, 0)$. For each k, $0 \le k \le n-1$, do the following:
   (a) According to Algorithm PDP described in Sect. 2.1, move from a stored primal state (k, Z_k, y_k) to at most two primal states of the form $(k + 1, Z_{k+1}, y_{k+1})$, where $Z_{k+1} \le Z^{UB}$, using the relations (4) and (5), each time rounding up the updated value of Z_{k+1} to the next multiple of δ. For each selection of states with the same value of Z_{k+1} and a subinterval I_j^r, determine the value $y_{k+1}^{\min}$ as the smallest value of y_{k+1} that belongs to I_j^r and the value $y_{k+1}^{\max}$ as the largest value of y_{k+1} that belongs to I_j^r. If these values exist and are distinct, then out of all states $(k + 1, Z_{k+1}, y_{k+1})$ with the same value of Z_{k+1} for $y_{k+1} \in [y_{k+1}^{\min}, y_{k+1}^{\max}]$ store only two states $(k + 1, Z_{k+1}, y_{k+1}^{\min})$ and $(k + 1, Z_{k+1}, y_{k+1}^{\max})$.
   (b) According to Algorithm DDP described in Sect. 2.1, move from a stored dual state $(k, Z_k, \widetilde{y}_k)$ to at most two dual states of the form $(k + 1, Z_{k+1}, \widetilde{y}_{k+1})$, where $Z_{k+1} \le Z^{UB}$, using the relations (6) and (7), each time rounding up the updated value of Z_{k+1} to the next multiple of δ. For each selection of states with the same value of Z_{k+1} and a subinterval I_j^r, determine the value $\widetilde{y}_{k+1}^{\min}$ as the smallest value of $\widetilde{y}_{k+1}$ that belongs to I_j^r and the value $\widetilde{y}_{k+1}^{\max}$ as the largest

value of $\widetilde{y}_{k+1}$ that belongs to I_j^r. If these values exist and are distinct then out of all states $(k+1, Z_{k+1}, \widetilde{y}_{k+1})$ with the same value of Z_{k+1} for $\widetilde{y}_{k+1} \in [\widetilde{y}_{k+1}^{\min}, \widetilde{y}_{k+1}^{\max}]$ store only two states $(k+1, Z_{k+1}, \widetilde{y}_{k+1}^{\min})$ and $(k+1, Z_{k+1}, \widetilde{y}_{k+1}^{\max})$.

(c) For each primal state $(k+1, Z_{k+1}, y_{k+1})$ stored in Step 4a of this iteration additionally store the dual state $(k+1, Z_{k+1}, \widetilde{y}_{k+1})$, where $\widetilde{y}_{k+1} = A_{k+1} - y_{k+1}$, unless it coincides with one of the states stored in Step 4b of this iteration.

(d) For each dual state $(k+1, Z_{k+1}, \widetilde{y}_{k+1})$ stored in Step 4b of this iteration additionally store the primal state $(k+1, Z_{k+1}, y_{k+1})$, where $y_{k+1} = A_{k+1} - \widetilde{y}_{k+1}$, unless it coincides with one of the states stored in Step 4a of this iteration.

5. Among all values Z_n found in Step 4 identify the smallest one associated with a feasible value $y_n \le A$. With this value of Z_n, perform the backtracking to find the corresponding decision variables x_j, $j = 1, \dots, n$. Compute the value of the objective function with the found x_j's, call this value Z^ε and accept it as an approximate value of the objective function.

We will show that Algorithm Eps is an FPTAS. We start with a statement that explains the role of the intervals created in Step 3 of Algorithm Eps.

For each k, $0 \le k \le n-1$, define

$$B(k) := \max\{\beta_{k+1}, \beta_{k+2}, \dots, \beta_n\}. \tag{8}$$

The following statement holds.

Lemma 1 *Assume that the primal dynamic programming Algorithm PDP from Sect. 2.1 is applied to problem (SQKP) and finds a chain of states*

$$(0, \Gamma, 0), (1, Z_1^*, y_1^*), \dots, (n, Z_n^*, y_n^*)$$

leading to the optimal value $Z^* = Z_n^*$. Then for each k, $0 \le k \le n-1$, for $B(k)$ defined by (8) either

$$B(k) y_k^* \le Z^*,$$

or

$$B(k) \widetilde{y}_k^* \le Z^*,$$

where $\widetilde{y}_k^* = A_k - y_k^*$.

Proof First, it can be seen that for each k, $0 \le k \le n-1$, either $\beta_{k+1} y_k^* \le Z^*$ holds or $\beta_{k+1} \widetilde{y}_k^* \le Z^*$ holds. If for some $k, 0 \le k \le n-1$, we have that $\beta_{k+1} y_k^* > Z^*$, then in Algorithm PDP the transition from the state (k, Z_k^*, y_k^*) to the state $(k+1, Z_{k+1}^*, y_{k+1}^*)$ has been made by formula (5), so that $\beta_{k+1}(A_k - y_k^*) = \beta_{k+1} \widetilde{y}_k^* \le Z^*$.

Recall that according to (4) and (5) for any $k, 0 \le k \le n-1$, we have either $y_{k+1}^* = y_k^* + \alpha_{k+1}$ or $y_{k+1}^* = y_k^*$, i.e., the sequence $y_0^* = 0, y_1^*, \dots, y_n^*$ is non-decreasing. In a similar way, the sequence $\widetilde{y}_0^* = 0, \widetilde{y}_1^*, \dots, \widetilde{y}_n^*$ can also be proved non-decreasing.

Now, let $B(k) = \beta_{k+v+1}$ for some $v, 0 \le v \le n-k-1$. Then, as proved above, we either have $\beta_{k+v+1} y_{k+v}^* \le Z^*$ or $\beta_{k+v+1} \widetilde{y}_{k+v}^* \le Z^*$, and the lemma follows immediately from the inequalities $y_{k+v}^* \ge y_k^*$ and $\widetilde{y}_{k+v}^* \ge \widetilde{y}_k^*$. $\square$

The following statement studies the behavior of Step 4 of Algorithm Eps.

Lemma 2 *Assume that the primal dynamic programming Algorithm PDP from Sect.* 2.1 *is applied to problem* (*SQKP*) *and finds a chain of primal states*

$$(0,\Gamma,0),(1,Z_1^*,y_1^*),\dots,(n,Z_n^*,y_n^*)$$

leading to the optimal value $Z^*=Z_n^*$. *Let*

$$(0,\Gamma,0),(1,Z_1^*,\widetilde{y}_1^*),\dots,(n,Z_n^*,\widetilde{y}_n^*)$$

be the corresponding chain of dual states. Then for each k, $1\le k\le n$, *Algorithm Eps finds*

(i) *a pair of primal states* (k,Z_k',y_k') *and* (k,Z_k'',y_k'') *such that*

$$Z_k'\le Z_k^*+2k\delta;\qquad Z_k''\le Z_k^*+2k\delta \tag{9}$$

and

$$y_k'\le y_k^*\le y_k'',\quad y_k''-y_k^*\le\frac{\delta}{B(k)},\quad y_k^*-y_k'\le\frac{\delta}{B(k)}; \tag{10}$$

(ii) *a pair of dual states* $(k,Z_k',\widetilde{y}_k')$ *and* $(k,Z_k'',\widetilde{y}_k'')$ *such that* (9) *holds and*

$$\widetilde{y}_k'\le\widetilde{y}_k^*\le\widetilde{y}_k'',\quad \widetilde{y}_k''-\widetilde{y}_k^*\le\frac{\delta}{B(k)},\quad \widetilde{y}_k^*-\widetilde{y}_k'\le\frac{\delta}{B(k)}, \tag{11}$$

where $B(k)$ *is defined by* (8).

Proof The proof is done by induction. The statement holds for $k=1$, since in the first iteration of Step 4 of Algorithm Eps only two states are created and at least one is kept. A possible difference between the values of Z_k' (or Z_k'') and Z_k^* does not exceed δ due to the rounding of the objective function.

Suppose that properties (i) and (ii) hold for $k=q$, where $1\le q\le n-1$. Recall that in the optimal chain of computation of the primal Algorithm PDP the transition from state (q,Z_q^*,y_q^*) to state $(q+1,Z_{q+1}^*,y_{q+1}^*)$ is done either by formula (4) or (5), which in general can be written as $Z_{q+1}^*=Z_q^*+\Delta(y_q^*)$, $y_{q+1}^*=\varphi(y_q^*)$, where either $\Delta(y_q^*)=\beta_{q+1}y_q^*+\mu_{q+1}$, $\varphi(y_q^*)=y_q^*+\alpha_{q+1}$ or $\Delta(y_q^*)=\beta_{q+1}(A_q-y_q^*)+\nu_{q+1}$, $\varphi(y_q^*)=y_q^*$. In iteration q of Step 4(a), Algorithm Eps computes two primal states $(q+1,Z_{q+1}',\varphi(y_q'))$ and $(q+1,Z_{q+1}'',\varphi(y_q''))$, where Z_{q+1}' and Z_{q+1}'' denote the rounded values $Z_q'+\Delta(y_q')$ and $Z_q''+\Delta(y_q'')$, respectively.

Due to property (i) for $k=q$, we have that $\varphi(y_q')\le y_{q+1}^*=\varphi(y_q^*)\le\varphi(y_q'')$, so that the differences $\varphi(y_q'')-y_{q+1}^*$ and $y_{q+1}^*-\varphi(y_q')$ are equal to $y_q''-y_q^*$ and $y_q^*-y_q'$, respectively, and each of them does not exceed $\frac{\delta}{B(q)}$.

From Lemma 1, we conclude that either $B(q+1)y_{q+1}^*\le Z^{UB}$ or $B(q+1)\tilde{y}_{q+1}^*\le Z^{UB}$. Notice that there is no guarantee that either $B(k+1)y_{k+1}^*\le Z^{UB}$ holds in

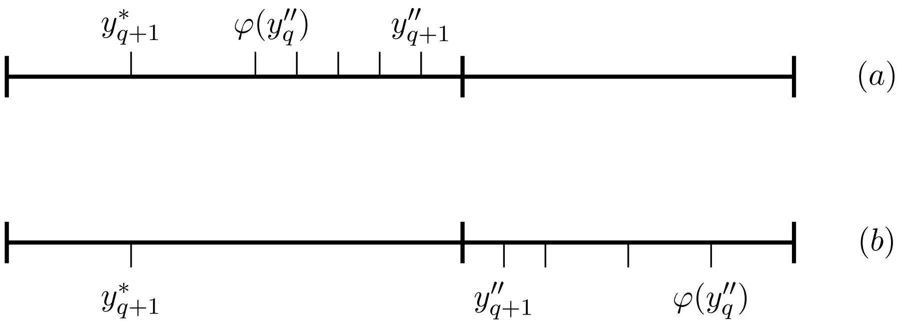

Fig. 1 Defining the value of y''_{q+1}

each iteration k for $k = 0, 1, \dots, n-1$, or $B(k+1)\tilde{y}^*_{k+1} \le Z^{UB}$ holds in each iteration k for $k = 0, 1, \dots, n-1$. This explains why we keep both primal and dual states in all iterations of Algorithm Eps.

The remainder of the proof is presented assuming that $B(q+1)y^*_{q+1} \le Z^{UB}$ holds. In our proof we work with the primal states and rely on property (i). If $B(q+1)\tilde{y}^*_{q+1} \le Z^{UB}$, the proof is symmetric: it operates with the dual states and relies on property (ii).

The assumption that $B(q+1)y^*_{q+1} \le Z^{UB}$ holds implies that y^*_{q+1} belongs to the interval $[0, \frac{Z^{UB}}{B(q+1)}]$ and falls into a subinterval I^r_j created in Step 3 of the algorithm of length at most $\frac{\delta}{B(q+1)}$, where $B(q+1) \le B(q)$. For the states of the form $(q+1, Z'_{q+1}, y)$ generated in iteration q of Step 4 of Algorithm Eps perform the following: if $\varphi(y'_q)$ and y^*_{q+1} belong to the same subinterval I^r_j, define $y'_{q+1} \le \varphi(y'_q)$ as the smallest y-value in the subinterval containing both y^*_{q+1} and $\varphi(y'_q)$; otherwise, define $y'_{q+1} \ge \varphi(y'_q)$ as the largest y-value in the subinterval containing $\varphi(y'_q)$. Similarly, for the states of the form $(q+1, Z''_{q+1}, y)$ generated in iteration $q+1$ of Step 4 of Algorithm Eps perform the following: if $\varphi(y''_q)$ and y^*_{q+1} belong to the same subinterval I^r_j, then define $y''_{q+1} \ge \varphi(y''_q)$ as the largest y-value in the subinterval containing both $\varphi(y''_q)$ and y^*_{q+1} (see Fig. 1(a)); otherwise, define $y''_{q+1} \le \varphi(y''_q)$ as the smallest y-value in the subinterval containing $\varphi(y''_q)$ (see Fig. 1(b)).

It is clear that the states $(q+1, Z'_{q+1}, y'_{q+1})$ and $(q+1, Z''_{q+1}, y''_{q+1})$ will be stored and $y'_{q+1} \le y^*_{q+1} \le y''_{q+1}$. Moreover, if y'_{q+1} and y^*_{q+1} belong to the same subinterval I^r_j then $y^*_{q+1} - y'_{q+1} \le \frac{\delta}{B(q+1)}$; otherwise, y^*_{q+1} is closer to y'_{q+1} than to $\varphi(y'_q)$, i.e., $y^*_{q+1} - y'_{q+1} \le y^*_{q+1} - \varphi(y'_q) \le \frac{\delta}{B(q)} \le \frac{\delta}{B(q+1)}$. Similarly, if y''_{q+1} and y^*_{q+1} belong to the same subinterval I^r_j then $y''_{q+1} - y^*_{q+1} \le \frac{\delta}{B(q+1)}$; otherwise, y^*_{q+1} is closer to y''_{q+1} than to $\varphi(y''_q)$, i.e., $y''_{q+1} - y^*_{q+1} \le \varphi(y''_q) - y^*_{q+1} \le \frac{\delta}{B(q)} \le \frac{\delta}{B(q+1)}$. Thus, (10) follows.

If $\Delta(y'_q) = \beta_{q+1}y'_q + \mu_{q+1}$, we use property (i) to derive

$$\begin{aligned}
Z'_{q+1} &\le Z'_q + \beta_{q+1} y'_q + \mu_{q+1} + \delta \\
&\le \left(Z^*_q + 2q\delta\right) + \beta_{q+1} y^*_q + \mu_{q+1} + \delta \\
&= \left(Z^*_q + \beta_{q+1} y^*_q + \mu_{q+1}\right) + (2q+1)\delta \\
&\le Z^*_{q+1} + (2q+2)\delta
\end{aligned}$$

and

$$\begin{aligned}
Z''_{q+1} &\le Z''_q + \beta_{q+1} y''_q + \mu_{q+1} + \delta \\
&\le \left(Z^*_q + 2q\delta\right) + \beta_{q+1}\left(y^*_q + \frac{\delta}{B(q)}\right) + \mu_{q+1} + \delta \\
&\le Z^*_q + \beta_{q+1}\left(y^*_q + \frac{\delta}{\beta_{q+1}}\right) + \mu_{q+1} + (2q+1)\delta \\
&= \left(Z^*_q + \beta_{q+1} y^*_q + \mu_{q+1}\right) + (2q+2)\delta \\
&= Z^*_{q+1} + (2q+2)\delta.
\end{aligned}$$

Otherwise, if $\Delta(y'_q) = \beta_{q+1}(A_q - y'_q) + \nu_{q+1}$, we obtain

$$\begin{aligned}
Z'_{q+1} &\le Z'_q + \beta_{q+1}(A_q - y'_q) + \nu_{q+1} + \delta \\
&\le \left(Z^*_q + 2q\delta\right) + \beta_{q+1}\left(A_q - \left(y^*_q - \frac{\delta}{B(q)}\right)\right) + \nu_{q+1} + \delta \\
&\le (Z^*_q + \beta_{q+1}(A_q - y^*_q) + \nu_{q+1}) + 2q\delta + \beta_{q+1}\left(\frac{\delta}{\beta_{q+1}}\right) + \delta \\
&= Z^*_{q+1} + (2q+2)\delta
\end{aligned}$$

and

$$\begin{aligned}
Z''_{q+1} &\le Z''_q + \beta_{q+1}(A_q - y''_q) + \nu_{q+1} + \delta \\
&\le \left(Z^*_q + 2q\delta\right) + \beta_{q+1}\left(A_q - y^*_q\right) + \nu_{q+1} + \delta \\
&= \left(Z^*_q + \beta_{q+1}\left(A_q - y^*_q\right) + \nu_{q+1}\right) + (2q+1)\delta \\
&\le Z^*_{q+1} + (2q+2)\delta.
\end{aligned}$$

In any case, the inequalities (9) hold, so that property (i) is proved for $k = q+1$.

Recall that for the kept primal states $(k+1, Z'_{k+1}, y'_{k+1})$ and $(k+1, Z''_{k+1}, y''_{k+1})$ Algorithm Eps in Step 4c also stores the corresponding dual states $(k+1, Z'_{k+1}, A_{k+1} - y'_{k+1})$ and $(k+1, Z''_{k+1}, A_{k+1} - y''_{k+1})$. Define $\widetilde{y}'_{k+1} = A_{k+1} - y''_{k+1}$ and $\widetilde{y}''_{k+1} = A_{k+1} - y'_{k+1}$. Since the differences $y''_{k+1} - y^*_{k+1}$ and $y^*_{k+1} - y'_{k+1}$ are equal to $\widetilde{y}^*_{k+1} - \widetilde{y}'_{k+1}$ and $\widetilde{y}''_{k+1} - \widetilde{y}^*_{k+1}$, respectively, we conclude that for $k = q+1$ properties (i) and (ii) hold simultaneously. □

Now we prove that Algorithm Eps is an FPTAS.

Theorem 2 *Let Z^* denote an optimal value of the objective function for problem (SQKP). Given a positive ε, Algorithm* Eps *outputs a value Z^ε such that*

$$Z^\varepsilon - Z^* \leq \varepsilon Z^*$$

and requires $O(T(n) + \frac{n^4}{\varepsilon^2})$ time, where $T(n)$ denotes the time needed for finding an upper bound Z^{UB} such that $Z^{UB}/Z^* \leq \rho$.

Proof Algorithm Eps creates a state (n, Z_n, y_n) that satisfies property (i) of Lemma 2 for $k = n$. Since $y_n \leq y_n^*$ due to (10), the knapsack constraint $y_n \leq A$ is satisfied, which means that Algorithm Eps finds a feasible solution to the original problem (SQKP). Since Z^ε is computed in Step 5 without any rounding, it follows $Z^\varepsilon \leq Z_n$ and we derive from (9) that

$$Z^\varepsilon - Z^* \leq Z_n - Z_n^* \leq 2n\delta.$$

By definition of δ in Step 2 of Algorithm Eps, we have that $2n\delta = \varepsilon Z_{LB}$, as required.

Let us estimate the running time of the algorithm. First, notice that the total number of the subintervals created in Step 3 does not exceed $n\frac{Z^{UB}}{\delta}$, because each interval I_u, $1 \leq u \leq h$, is split into subintervals of length $\frac{\delta}{\beta_{\pi(u)}}$, so that

$$\frac{Z^{UB}}{\beta_{\pi(1)}} \times \frac{\beta_{\pi(1)}}{\delta} + \sum_{u=2}^{h} \left[\frac{Z^{UB}}{\beta_{\pi(u)}} - \frac{Z^{UB}}{\beta_{\pi(u-1)}} \right] \frac{\beta_{\pi(u)}}{\delta}$$
$$\leq \sum_{u=1}^{h} \frac{Z^{UB}}{\beta_{\pi(u)}} \times \frac{\beta_{\pi(u)}}{\delta} \leq h\frac{Z^{UB}}{\delta} \leq n\frac{Z^{UB}}{\delta}.$$

In some iteration k, $0 \leq k \leq n-1$, of Step 4a the primal states with no more than Z^{UB}/δ distinct values of the objective function are originally created due to the rounding. For each of these values at most two states are kept in each of at most $n\frac{Z^{UB}}{\delta}$ subintervals. Thus, in each iteration of Step 4a at most $O(n(\frac{Z^{UB}}{\delta})^2)$ states are created and stored.

We outline a possible implementation of the storage procedure of the states in each iteration.

Suppose that all subintervals created in Step 3 are numbered by the integers $1, 2, \ldots$ in increasing order of their left endpoints, and for each subinterval u its left endpoint $y^{(u)}$ is kept. Notice that the length of these subintervals does not decrease as the their numbers increase, see Step 3. This takes $O(n\frac{Z^{UB}}{\delta})$ time.

Suppose that in some iteration k of Step 4a for each stored state (k, Z_k, y_k) we additionally keep a reference to the number of the subinterval that contains y_k. Before running the next iteration $k+1$ we perform additional *pre-processing* that for each potentially created state $(k+1, Z_{k+1}, y_{k+1})$ will predict at most two subintervals that may contain y_{k+1}. We only need to consider the case that $y_{k+1} = y_k + \alpha_{k+1}$, since in the other case $y_{k+1} = y_k$ and for the new created state $(k+1, Z_{k+1}, y_{k+1})$ the value y_{k+1} belongs to the same subinterval as y_k.

We start with the value $y^{(1)} + \alpha_{k+1}$. Applying binary search, find in no more than $O(\log \alpha_{k+1})$ time the subinterval that contains $y^{(1)} + \alpha_{k+1}$. The value $y^{(2)} + \alpha_{k+1}$ will either belong to the same subinterval as $y^{(1)} + \alpha_{k+1}$ or to the next subinterval, since the distance between $y^{(2)}$ and $y^{(1)}$ is equal to that between $y^{(2)} + \alpha_{k+1}$ and $y^{(1)} + \alpha_{k+1}$, while the lengths of the subintervals do not decrease. Similarly, the value $y^{(3)} + \alpha_{k+1}$ either belongs to the same subinterval as $y^{(2)} + \alpha_{k+1}$ or to the next interval, etc. Notice that binary search is required only for finding the subinterval that hosts the smallest value, $y^{(1)} + \alpha_{k+1}$. The time needed for this pre-processing is linear in the number of subintervals, i.e., $O(n\frac{Z^{UB}}{\varepsilon})$.

After this pre-processing is done, for each state (k, Z_k, y_k) we make predictions for possible states to be created regarding the subintervals that might contain the value $y_k + \alpha_{k+1}$. This is done by following the pointer to subinterval u that contains y_k, and then checking the predicted subintervals that contain $y^{(u)} + \alpha_{k+1}$ and $y^{(u+1)} + \alpha_{k+1}$ (either one subinterval or two consecutive subintervals). These predictions take constant time for each stored state, i.e., require $O(n(\frac{Z^{UB}}{\delta})^2)$ time. Then iteration $k+1$ is run as described in Step 4a, and for each found state $(k+1, Z_{k+1}, y_{k+1})$ the obtained predictions are used for finding the actual pointer to the subinterval that contains y_{k+1}.

All together, an iteration of Step 4a can be implemented in $O(n(\frac{Z^{UB}}{\delta})^2)$ time. An iteration of Step 4b has the same time complexity. Thus, the running time of Steps 3–5 of Algorithm Eps is at most $O(n^2(\frac{Z^{UB}}{\delta})^2)$. Step 1 requires $T(n)$ time.

By definition of δ and ρ we have that

$$\frac{Z^{UB}}{\delta} = \frac{\rho Z_{LB}}{\delta} = \frac{2\rho n}{\varepsilon} = O\left(\frac{n}{\varepsilon}\right),$$

so that the running time of Algorithm Eps is $O(T(n) + \frac{n^4}{\varepsilon^2})$. □

An important feature of the described approach is the existence of a ρ-approximation algorithm for problem (SQKP), i.e., a polynomial-time algorithm that finds a heuristic solution to the problem with the objective function value Z^{UB}, which does not exceed ρZ^*. The fact that Z^{UB} should be found in time that is polynomial in n is used to demonstrate that the overall running time of the approximation scheme also depends on n polynomially. Additionally, due to $Z_{LB} = Z^{UB}/\rho$, the required accuracy of the scheme is guaranteed.

Notice that Algorithm Eps remains an FPTAS if the ratio ρ in $Z^{UB}/Z^* \le \rho$ is not a constant but polynomially depends on n. More precisely, if $\rho = O(n^c)$ for a positive c, then the running time of Algorithm Eps is $O(T(n) + \frac{n^{2c+4}}{\varepsilon^2})$.

For scheduling applications a version of problem (SQKP) in which all $\beta_j = 1$ is of certain importance. In this case Algorithm Eps is less time-consuming. In Step 3 of the algorithm it is sufficient to split the interval $[0, Z^{UB}]$ into subintervals of equal length δ. Thus, in each iteration for at most Z^{UB}/δ values of the objective function and each of Z^{UB}/δ subintervals we store at most two states, i.e., the total number of states created and stored in each iteration is $O((\frac{Z^{UB}}{\delta})^2)$, i.e., factor n less than in the general case of arbitrary β_j. This reduces the overall running time of Algorithm Eps for the case that all $\beta_j = 1$ to $O(T(n) + \frac{n^3}{\varepsilon^2})$.

For problem (SQKP) in its general setting (1) no approximation algorithm is known that provides $Z^{UB}/Z^* \le \rho$, even for ρ being a polynomial of n. However, as demonstrated in the subsequent sections, several scheduling problems can be formulated as instances of (SQKP) and for these problems the required approximation algorithms exist, typically with ρ bounded by a constant and $T(n) = O(n^2)$.

Notice that Woeginger [19] claims that another DP algorithm with an extended number of states could be designed for problem (SQKP) and it would be possible to convert it into an FPTAS in accordance with the general scheme presented in [18]. However, even assuming that all necessary technical issues are successfully resolved, this approach would result into a much less efficient algorithm than Algorithm Eps above. Given that problem (SQKP) is of interest not only in its own right but as an underlying problem for various scheduling applications, our intention of setting up a fairly fast FPTAS is well justified. Additionally, our approach allows us to handle scheduling applications, in particular problems $1|h(1), Res|\sum w_jC_j$ and $1|C_{MP} \le d|\sum w_jC_j$, that are relevant to problem (SQKP), but do not reduce to it directly. Besides, the crucial technical feature of Algorithm Eps is the splitting of the range of the objective function values into subintervals of variable length and storing at most two states related to each small subinterval, and that may appear to be useful for other algorithms and schemes of a similar nature.

3 Scheduling with a Fixed Non-Availability Interval

In this section, we consider two versions of the problem of minimizing the sum of the weighted completion times on a single machine with a fixed machine non-availability interval and present an FPTAS for each of these problems.

Recall that in our scheduling applications, the jobs of set $N = \{1, 2, \dots, n\}$ have to be processed without preemption on a single machine. The processing of job $j \in N$ takes p_j time units. There is a weight w_j associated with job j, which indicates its relative importance. All values p_j and w_j are positive integers. The machine processes at most one job at a time. The completion time of job $j \in N$ in a feasible schedule S is denoted by $C_j(S)$, or shortly C_j if it is clear which schedule is referred to. In a specific problem, it is required to minimize the function $Z(S) = \sum_{j=1}^n w_jC_j(S)$. For all problems under consideration S^* denotes an optimal schedule, i.e., $Z(S^*) \le Z(S)$ for any feasible schedule S.

In what follows, we assume that in each scheduling problem the jobs are numbered in such a way that

$$\frac{p_1}{w_1} \le \frac{p_2}{w_2} \le \cdots \le \frac{p_n}{w_n}. \tag{12}$$

We call the sequence of jobs numbered in accordance with (12) a Smith sequence or a WSPT sequence (Weighted Shortest Processing Time). Recall that in an optimal schedule for the classical single machine problem of minimizing the sum of the weighted completion times, the jobs are processed according to the WSPT sequence, see [16].

In the scheduling model we consider in this section, we assume that the machine non-availability interval occupies the interval $I=[s,t]$. Under the non-resumable scenario, the crossover job $k \in N$ that cannot be completed by time s is restarted, i.e., is processed for p_k time units starting at time t. Under the resumable scenario, the crossover job $k \in N$ is interrupted at time s and resumed at time t; its total processing before and after interval I is equal to p_k. Recall that we denote the resulting problems of minimizing the total weighted completion time either by $1|h(1), N-res|\sum w_j C_j$ (if the non-resumable scenario applies) or by $1|h(1), Res|\sum w_j C_j$ (if the resumable scenario applies).

First, we give a brief review of the known results for these two problems with a fixed non-availability interval. Even if all weights are equal, problem $1|h(1), N-res|\sum w_j C_j$ is NP-hard in the ordinary sense, see [1]. Lee [11] proves that for problem $1|h(1), N-res|\sum w_j C_j$ there exists an optimal schedule in which the jobs sequenced before interval I and after interval I obey the WSPT rule. Furthermore, if the jobs are scheduled according to their numbering (12) and the crossover job restarts from scratch at time t, then for the resulting schedule S_{WSPT} the ratio $Z(S_{WSPT})/Z(S^*)$ can be arbitrarily large but for some instances is bounded by 3, see [5, 11]. No approximation algorithm with a fixed worst-case performance ratio has been reported so far. Notice also that a straightforward modification of the proof given by Breit et al. [4] shows that the problem with more than one non-availability interval is not approximable within a constant factor unless $\mathcal{P}=\mathcal{NP}$.

For the unweighted case, the numbering (12) is equivalent to the SPT (Shortest Processing Time) rule. Lee and Liman [13] by refining the analysis by Adiri et al. [1] demonstrate that for problem $1|h(1), N-res|\sum C_j$ the schedule found by the SPT rule guarantees that $Z(S_{SPT})/Z(S^*) \le \frac{9}{7}$, and this bound is tight. For problem $1|h(1), N-res|\sum C_j$, an algorithm with an improved worst-case performance ratio of $\frac{20}{17}$ is given by Sadfi et al. [15].

Problem $1|h(1), Res|\sum w_j C_j$ in which the crossover job is dealt with according to the resumable scenario is studied by Lee [11]. He shows that this problem is NP-hard, but its counterpart with equal weights, i.e., problem $1|h(1), Res|\sum C_j$, is solvable in $O(n \log n)$ time, since schedule S_{SPT} is optimal. Similarly to the non-resumable scenario, for problem $1|h(1), Res|\sum w_j C_j$ there exists an optimal schedule in which the jobs sequenced before and after the crossover job follow the WSPT rule. For problem $1|h(1), Res|\sum w_j C_j$, Wang et al. [17] have recently reported a 2-approximation algorithm that requires $O(n^2)$ time.

3.1 Non-Resumable Scenario

In this subsection, we show that problem $1|h(1), N-res|\sum w_j C_j$ can be formulated as problem (SQKP). We present a simple 4-approximation algorithm based on the algorithm by Wang et al. [17] for the resumable counterpart of this problem. These facts allow us a straightforward application of the FPTAS that has been designed for problem (SQKP) to problem $1|h(1), N-res|\sum w_j C_j$.

Given problem $1|h(1), N-res|\sum w_jC_j$, introduce a Boolean variable x_j in such a way that

$$x_j = \begin{cases} 1, & \text{if job } j \text{ completes before interval } I, \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

for each job $j,\ 1 \le j \le n$. If job j completes before the interval I, then

$$C_j = \sum_{i=1}^{j} p_i x_i,$$

while if it completes after the interval I then

$$C_j = t + \sum_{i=1}^{j} p_j(1 - x_j).$$

Thus, the sum of the weighted completion times can be written as

$$\begin{aligned} Z &= \sum_{j=1}^{n} w_j x_j \sum_{i=1}^{j} p_i x_i + \sum_{j=1}^{n} w_j(1-x_j)\left(t + \sum_{i=1}^{j} p_i(1-x_i)\right) \\ &= \sum_{1\le i\le j\le n} p_i w_j x_i x_j + \sum_{1\le i\le j\le n} p_i w_j (1-x_i)(1-x_j) + t\sum_{j=1}^{n} w_j(1-x_j). \end{aligned}$$

Since $x_j^2 = x_j$ for any j, we derive

$$\begin{aligned} &\sum_{1\le i\le j\le n} p_i w_j x_i x_j + \sum_{1\le i\le j\le n} p_i w_j (1-x_i)(1-x_j) + t\sum_{j=1}^{n} w_j(1-x_j) \\ &\quad = \sum_{1\le i<j\le n} p_i w_j x_i x_j + \sum_{1\le i<j\le n} p_i w_j (1-x_i)(1-x_j) \\ &\qquad + \sum_{j=1}^{n} p_j w_j x_j + \sum_{j=1}^{n} p_j w_j (1-x_j) + t\sum_{j=1}^{n} w_j(1-x_j), \end{aligned}$$

so that problem $1|h(1), N-res|\sum w_jC_j$ can be formulated as the following Boolean quadratic programming problem

$$\begin{aligned} \text{Minimize}\quad Z = &\sum_{1\le i<j\le n} p_i w_j x_i x_j + \sum_{1\le i<j\le n} p_i w_j (1-x_i)(1-x_j) \\ &+ t\sum_{j=1}^{n} w_j(1-x_j) + \sum_{j=1}^{n} p_j w_j \end{aligned} \tag{14}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} p_j x_j \le s$$

$$x_j \in \{0,1\}, \ j = 1, 2, \dots, n.$$

If in (1) we define

$$\alpha_j = p_j, \qquad \beta_j = w_j, \qquad \mu_j = 0, \qquad \nu_j = w_j t, \quad j = 1, 2, \dots, n;$$

$$A = s, \qquad \Gamma = \sum_{j=1}^{n} p_j w_j,$$

then (14) becomes an instance of (1).

Now we demonstrate how a heuristic algorithm by Wang et al. [17] developed for problem $1|h(1), Res|\sum w_j C_j$ can be used as a basis of a 4-approximation algorithm for problem $1|h(1), N-res|\sum w_j C_j$.

Given an instance of problem $1|h(1), N-res|\sum w_j C_j$, denote

- S_N^*—an optimal schedule for the non-resumable scenario;
- S_R^*—an optimal schedule for the resumable scenario;
- S_R^W—a schedule for the resumable scenario found by the algorithm by Wang et al. [17];
- S_N^W—a schedule that is obtained from S_R^W by processing the crossover job according to the non-resumable scenario, i.e., inserting it to restart from time t.

Lemma 3 *Schedule S_N^W is feasible for problem $1|h(1), N-res|\sum w_j C_j$ and the bound*

$$Z(S_N^W) \le 4Z(S_N^*)$$

holds. Finding schedule S_N^W takes $O(n^2)$ time.

Proof Take an instance of problem of $1|h(1), N-res|\sum w_j C_j$ and temporarily allow the resumable scenario of handling the crossover job. It is clear that the obtained problem $1|h(1), Res|\sum w_j C_j$ is a relaxation of the original problem $1|h(1), N-res|\sum w_j C_j$, so that $Z(S_R^*) \le Z(S_N^*)$.

As proved in [17], the inequality $Z(S_R^W) \le 2Z(S_R^*)$ is guaranteed and schedule S_R^W can be found in $O(n^2)$ time. Consider schedule S_R^W. Assume that in this schedule the jobs complete according to the sequence $\pi = (\pi(1), \dots, \pi(n))$, and job $\pi(k)$ is the crossover job, i.e., it is the job that is interrupted at time s and resumed at time t. For schedule S_R^W, denote the total processing time of job $\pi(k)$ before time s by y.

Convert schedule S_R^W into a new schedule S_N^W that is feasible for the non-resumable scenario by temporarily removing job $\pi(k)$ and then inserting it to start at time t. Formally,

$$C_{\pi(j)}(S_N^W) = C_{\pi(j)}(S_R^W), \quad j = 1, \dots, k-1;$$

$$C_{\pi(j)}(S_N^W) = C_{\pi(j)}(S_R^W) + y, \quad j = k, \dots, n.$$

It follows that

$$Z(S_N^W) = Z(S_R^W) + y \sum_{j=k}^{n} w_{\pi(j)}.$$

Since $y \le s \le t$ and in schedule S_R^W each job $\pi(j)$ for $k \le j \le n$ completes after time t, we deduce that $y \sum_{j=k}^{n} w_{\pi(j)} \le Z(S_R^W)$, i.e., $Z(S_N^W) \le 2Z(S_R^W)$. Now, since $Z(S_R^*) \le Z(S_N^*)$, we derive

$$Z(S_N^W) \le 2Z(S_R^W) \le 4Z(S_R^*) \le 4Z(S_N^*),$$

as required. Since transforming schedule S_R^W into schedule S_N^W takes at most $O(n)$ time, finding schedule S_N^W requires $O(n^2)$ time. $\square$

We are now ready to apply Algorithm Eps to obtain an FPTAS for problem $1|h(1), N-res|\sum w_j C_j$.

In Step 1 of Algorithm Eps, we take $Z(S_N^W)$ found by the 4-approximation algorithm described above as an upper bound Z^{UB}. The dynamic programming algorithms can be applied to problem (SQKP) of the form (14). Suppose that Algorithm Eps finishes, having found a solution vector $x^\varepsilon = (x_1^\varepsilon, \ldots, x_n^\varepsilon)$ where each $x_j^\varepsilon \in \{0, 1\}$. As an approximate solution of problem $1|h(1), N-res|\sum w_j C_j$ we take schedule S^ε in which the jobs of set $N_1 = \{j \in N | x_j^\varepsilon = 1\}$ are sequenced from time zero according to the WSPT rule and complete before time s, while the remaining jobs of set $N_2 = \{j \in N | x_j^\varepsilon = 0\}$ are sequenced from time t according to the WSPT rule. Theorem 2 and Lemma 3 guarantee that $Z(S^\varepsilon)/Z(S^*) \le 1+\varepsilon$ and that the running time of the resulting algorithm is $O(n^2 + n^4/\varepsilon^2) = O(n^4/\varepsilon^2)$.

Notice, that for problem $1|h(1), N-res|\sum C_j$ with equal weights of jobs, the running time of an FPTAS can be reduced. In this case we may take the value $Z(S_{SPT})$ as an upper bound Z^{UB}. Recall that Lee and Liman [13] demonstrate that $Z(S_{SPT}) \le \frac{9}{7} Z(S^*)$ and finding schedule S_{SPT} requires $O(n \log n)$ time. Besides, minimizing $\sum C_j$ is equivalent to solving problem (SQKP) with all $\beta_j = 1$. As noted in the end of Sect. 2.2, in this case Algorithm Eps requires only $O(T(n) + n^3/\varepsilon^2)$ time, so that with $T(n) = O(n \log n)$ the running time of an FPTAS for problem $1|h(1), N-res|\sum C_j$ is $O(n^3/\varepsilon^2)$.

3.2 Resumable Scenario

The purpose of this subsection is to design an FPTAS for problem $1|h(1), Res|\sum w_j C_j$. This will be done by reducing the original problem $1|h(1), Res|\sum w_j C_j$ to a sequence of n auxiliary problems similar to problem (SQKP), one problem for each job selected as crossover.

Suppose that a certain job is chosen as the crossover job. Denote the processing time and the weight of the chosen crossover job by p and w, respectively, and renumber the remaining jobs taken according to the WSPT rule by the integers $1, 2, \ldots, m$, where $m = n - 1$.

A feasible schedule for problem $1|h(1), Res|\sum w_j C_j$ with a fixed crossover job can be found by inserting the crossover job into a schedule for processing the jobs $1, 2, \ldots, m$ under the non-resumable scenario.

Let S^* denote the optimal schedule that delivers the smallest value $Z(S^*)$ of the objective function, while $S(p)$ denote a feasible schedule with a fixed crossover job with the processing time p and weight w. Denote the smallest value of the function among all schedules with the chosen crossover job by $Z^*(p)$.

Define the Boolean decision variables x_j such that (13) holds for each j, $1 \le j \le m$. It follows from (14) that for an arbitrary assignment of variables x_j the sum of the weighted completion times of the corresponding schedule S_m under the non-resumable scenario is given by

$$Z_m = \sum_{1 \le i < j \le m} p_i w_j x_i x_j + \sum_{1 \le i < j \le m} p_i w_j (1 - x_i)(1 - x_j) + t \sum_{j=1}^{m} w_j (1 - x_j) + \sum_{j=1}^{m} p_j w_j,$$

where

$$\sum_{j=1}^{m} p_j x_j \le s, \quad x_j \in \{0, 1\}, \ j = 1, 2, \ldots, m.$$

To convert a schedule S_m into a schedule $S(p)$ that is feasible for problem $1|h(1), Res|\sum w_j C_j$ with the chosen crossover job we process the crossover job for px time units before the non-availability interval starting at time $\sum_{j=1}^{m} p_j x_j$. Here either $x = 1$, if

$$p < s - \sum_{j=1}^{m} p_j x_j,$$

or

$$px = s - \sum_{j=1}^{m} p_j x_j,$$

otherwise. In the latter case, the crossover job is additionally processed for $p(1 - x)$ time units starting at time t, and this increases the starting (and the completion) time of each job with $x_j = 0$ by $p(1 - x)$.

The value of the objective function of the resulting schedule $S(p)$ can be written as

$$Z(p) = Z_m + f(y_m, W_m, x), \tag{15}$$

where

$$f(y_m, W_m, x) = \begin{cases} w\,(y_m + p), & \text{if } x = 1, \\ w\,(t + p(1 - x)) + W_m p(1 - x), & \text{otherwise,} \end{cases}$$

$$y_m = \sum_{j=1}^{m} p_j x_j,$$

$$W_m = \sum_{j=1}^{m} w_j (1 - x_j).$$

3.2.1 Dynamic Programming

The value of Z_m can be found either by a primal dynamic programming algorithm, similar to Algorithm PDP, or by its dual counterpart; see Sect. 2.1. The only difference is that now for computing the overall value of the objective function Z we need to know the value of W_m, or rather $W_m p$ that contributes into Z. This value can be found recursively during the iterations of the DP algorithm.

Recall that to be able to apply the dynamic programming algorithms and the subsequent FPTAS the existence of an upper bound on an optimal value of the function is important. As such an upper bound we can use the value $Z(S_W)$, where S_W is a schedule delivered by the algorithm given in [17]. Recall that $Z(S_W)/Z(S^*) \le 2$, and finding S_W takes $O(n^2)$ time.

The description of the resulting algorithm is similar to that of Algorithm PDP, provided that

$$\alpha_j = p_j, \qquad \beta_j = w_j, \qquad \mu_j = 0, \qquad \nu_j = t w_j, \quad j = 1, 2, \dots, m;$$

$$n = m, \qquad A = s, \qquad \Gamma = \sum_{j=1}^{m} p_j w_j.$$

Additionally, a typical primal state after the values $x_1, x_2, \dots, x_k$ have been assigned is represented by a state of the form

$$(k, Z_k, y_k, V_k),$$

where

k is the number of the assigned variables;
Z_k is the current value of the objective function;
$y_k := \sum_{j=1}^{k} p_j x_j$, the total processing time of the jobs processed before the non-availability interval;
$V_k := p \sum_{j=1}^{k} w_j (1 - x_j)$, the total weight of the jobs processed after the non-availability interval times the processing time of the crossover job.

Initially, the algorithm starts with the state $(0, Z_0, y_0, V_0) = (0, \Gamma, 0, 0)$. In iteration k of the primal algorithm a move from a state (k, Z_k, y_k, V_k) to a state $(k+1, Z_{k+1}, y_{k+1}, V_{k+1})$ is done in accordance with

$$Z_{k+1} = Z_k + w_{k+1} y_k, \qquad y_{k+1} = y_k + p_{k+1}, \qquad V_{k+1} = V_k, \tag{16}$$

provided that $y_{k+1} \leq s$, and with

$$Z_{k+1} = Z_k + w_{k+1}(A_k - y_k) + v_{k+1}; \qquad y_{k+1} = y_k, \qquad V_{k+1} = V_k + w_{k+1}p, \tag{17}$$

where $A_k = \sum_{j=1}^{k} p_j$.

For each state generated in the last iteration of the primal algorithm, we find x and compute the value of the function Z by formula (15). The smallest of the found Z-values corresponds to an optimal value $Z^*(p)$ of the total weighted completion time for problem $1|h(1), Res|\sum w_j C_j$ with a chosen crossover job.

If $Z^*(p)$ is computed from some state (m, Z_m^*, y_m^*, V_m^*) by (15), then we derive that $V_m^* \leq Z^*(p)$, since V_m^* represents a lower bound on the total weighted completion time of the jobs that are scheduled after the chosen crossover job among all schedules $S(p)$. This implies that all intermediate states of the DP algorithm with $V_k > Z^{UB}$ should be discarded.

If the value of $Z^*(p)$ is achieved for $x = 1$, i.e., the crossover job is not interrupted by the non-availability interval, then the corresponding schedule should be disregarded, since the overall optimal schedule belongs to a class of schedules with another crossover job.

The DP algorithm outlined above is used as a subroutine in the FPTAS presented in the next subsection. For this purpose, no removal of dominated states is allowed in the DP algorithm, i.e., all feasible combinations should be generated and stored, except those states (k, Z_k, y_k, V_k) for which either Z_k or V_k exceeds Z^{UB}. For a fixed crossover job such a DP algorithm requires $O(nA\,(Z^{UB})^2)$ time, i.e., an overall optimal solution will be found in $O(n^2 A(Z^{UB})^2)$ time.

If the DP algorithm is to be used as a pseudopolynomial-time algorithm for finding an optimal schedule, then among all states (k, Z_k, y_k, V_k) related to the same pair (y_k, V_k) it suffices to keep only one state, with the smallest value of Z_k. This reduces the number of states in each iteration and the overall running time becomes $O(n^2 A\, Z^{UB})$.

We skip the details of the dual version of our algorithm for finding the values of Z_m.

3.2.2 FPTAS

Given problem $1|h(1), Res|\sum w_j C_j$ with a chosen crossover job, we show how to convert the corresponding primal and dual dynamic programing algorithms into an FPTAS. It is clear that we only need approximation at the stage of computing all values of Z_m; for a found value of Z_m the insertion of the crossover job is done in a straightforward way.

The FPTAS for finding the values Z_m can be obtained using the same principles that we have employed in designing Algorithm Eps; see Sect. 2.2. First, this means that both primal and dual dynamic programming algorithms are run in each iteration. Second, to reduce the number of computed values of the function, rounding is performed. Additionally, for each value of the function we store at most two y-values (or $\tilde{y}$-values in the dual version) in a certain small subinterval. Lemma 2 demonstrates that for the stored states the Z-values and y-values are close enough to those on an optimal path.

In our case, however, we operate with extended states that also involve the V-values. Here there is no need in designing an additional sophisticated mechanism of reducing the number of stored states (k, Z_k, y_k, V_k). The reason is that the V-values do not contribute into the computed Z-values (until the crossover jobs is inserted) and do not depend on the computed y-values.

Define $Z^{UB} = Z(S_R^W)$ as an upper bound on $Z(S^*)$, where S_R^W is a schedule delivered by the algorithm by Wang et al. [17]. It follows that

$$Z_{LB} = \frac{1}{2} Z^{UB}$$

is a lower bound on $Z(S^*)$, and therefore on $Z^*(p)$.

In Step 2 of Algorithm Eps redefine

$$\delta = \frac{\varepsilon Z_{LB}}{4m}.$$

Change Step 4a of the algorithm in the following way. Move from a stored primal state (k, Z_k, y_k, V_k) to at most two primal states of the form $(k+1, Z_{k+1}, y_{k+1}, V_{k+1})$, where $Z_{k+1} \le Z^{UB}$ and $V_{k+1} \le Z^{UB}$, using the relations (16) and (17), each time rounding up each of the updated values of Z_{k+1} and V_{k+1} to the next multiple of δ. For each selection of states related to the same pair (Z_{k+1}, V_{k+1}) and a subinterval I_j^r, determine the value $y_{k+1}^{\min}$ as the smallest value of y_{k+1} that belongs to I_j^r and the value $y_{k+1}^{\max}$ as the largest value of y_{k+1} that belongs to I_j^r. If these values exist and are distinct, then out of all states $(k+1, Z_{k+1}, y_{k+1}, V_{k+1})$ with the same values of Z_{k+1}, V_{k+1} for $y_{k+1} \in [y_{k+1}^{\min}, y_{k+1}^{\max}]$ store only two states $(k+1, Z_{k+1}, y_{k+1}^{\min}, V_{k+1})$ and $(k+1, Z_{k+1}, y_{k+1}^{\max}, V_{k+1})$.

The description of Step 4b of Algorithm Eps is altered similarly; here we manipulate with the corresponding dual states.

If an optimal DP algorithm were applied, then $Z^*(p)$ would be computed from some state (m, Z_m^*, y_m^*, V_m^*) by inserting the crossover job, so that

$$\begin{aligned} Z^*(p) &= Z_m^* + wt + \left(wp + V_m^*\right)(1 - x^*) \\ &= Z_m^* + wt + wp + V_m^* - \left(wp + V_m^*\right) x^*, \end{aligned}$$

where

$$x^* = \frac{s - y_m^*}{p}$$

denotes the part of the crossover job processed before the non-availability interval.

To find an approximate value of the objective function problem $1|h(1), Res| \sum w_j C_j$ with a chosen crossover job, perform the following:

- for each stored state of the form (m, Z_m, y_m, V_m) by backtracking find the associated values of the decision variables and then compute the values Z'_m and V'_m of the objective function and the V-variable, respectively, without any rounding, thereby creating a triple (Z'_m, y_m, V'_m);

- for each of the obtained triples, compute the value of the overall objective function with the inserted crossover job.

Let $Z^{\varepsilon}(p)$ denote the smallest value of the objective function computed from the triples (Z'_m, y_m, V'_m). For the modified algorithm outlined above, a statement similar to Lemma 2 holds and guarantees that a triple $(Z^{\varepsilon}_m, y^{\varepsilon}_m, V^{\varepsilon}_m)$ will be found such that $Z^{\varepsilon}_m - Z^*_m \le 2m\delta$ and $y^{\varepsilon}_m \le y^*_m$, so that

$$Z^{\varepsilon}(p) \le Z^{\varepsilon}_m + wt + \big(wp + V^{\varepsilon}_m\big)\,(1 - x^{\varepsilon}),$$

where

$$x^{\varepsilon} = \frac{s - y^{\varepsilon}_m}{p}.$$

Suppose that the triple $(Z^{\varepsilon}_m, y^{\varepsilon}_m, V^{\varepsilon}_m)$ is obtained from some state (m, Z_m, y_m, V_m). Recall that in each iteration of the modified Algorithm Eps the V-values are related neither to the Z-values nor to the y-values computed, so that the computation path that leads to V_m contains at most m rounding operations, so that $V^{\varepsilon}_m \le V_m \le V^*_m + m\delta$. Also, since V^{ε}_m is computed without rounding we obtain that $V^{\varepsilon}_m \ge V_m - m\delta \ge V^*_m - m\delta$, so that

$$V^*_m - m\delta \le V^{\varepsilon}_m \le V^*_m + m\delta. \tag{18}$$

Due to $Z^{\varepsilon}_m - Z^*_m \le 2m\delta$, we have that

$$\begin{aligned} Z^{\varepsilon}(p) - Z^*(p) &\le \big(Z^{\varepsilon}_m - Z^*_m\big) + \big(V^{\varepsilon}_m - V^*_m\big) - \big(\big(wp + V^{\varepsilon}_m\big)\,x^{\varepsilon} - \big(wp + V^*_m\big)\,x^*\big) \\ &\le 2m\delta + \big(V^{\varepsilon}_m - V^*_m\big) - wp(x^{\varepsilon} - x^*) - V^{\varepsilon}_m x^{\varepsilon} + V^*_m x^*. \end{aligned}$$

Since $y^{\varepsilon}_m \le y^*_m$, it follows that $x^{\varepsilon} > x^*$, so that $wp(x^{\varepsilon} - x^*)$ is positive and $V^{\varepsilon}_m x^{\varepsilon} > V^{\varepsilon}_m x^*$, so that

$$Z^{\varepsilon}(p) - Z^*(p) \le 2m\delta + \big(V^{\varepsilon}_m - V^*_m\big) - V^{\varepsilon}_m x^* + V^*_m x^*.$$

Further, from (18) we deduce that

$$\begin{aligned} Z^{\varepsilon}(p) - Z^*(p) &\le 2m\delta + \big(V^{\varepsilon}_m - V^*_m\big) + (V^*_m - V^{\varepsilon}_m)x^* \\ &\le 2m\delta + m\delta + m\delta x^*. \end{aligned}$$

Taking into account that $0 < x^* < 1$, we finally deduce

$$Z^{\varepsilon}(p) - Z^*(p) \le 4m\delta = \varepsilon Z_{LB}$$

which guarantees the required accuracy.

For each choice of the crossover job, running a modified FPTAS that finds the values Z_m takes $O(n^5/\varepsilon^3)$ time. Compared with the running time of Algorithm Eps (see Sect. 2.2), the extra factor n/ε is due to the fact that now the FPTAS has to store the V-values, and they take at most $Z^{UB}/\delta = O(n/\varepsilon)$ distinct values. To find an approximate solution to the original problem $1|h(1), Res|\sum w_j C_j$, we need to run the FPTAS for each job selected as the crossover job, rejecting the solutions in which the selected job completes before the non-availability interval. Thus, the overall running time of the FPTAS is $O(n^6/\varepsilon^3)$.

Recall that the unweighted counterpart of problem $1|h(1), Res|\sum w_j C_j$ is polynomially solvable.

4 Scheduling of a Machine Maintenance Interval

In this section we address problem $1|C_{MP} \leq d|\sum w_j C_j$ with a *floating* machine non-availability interval of length Δ. We demonstrate that the FPTAS designed for problem $1|h(1), Res|\sum w_j C_j$ in Sect. 3.2 can be adopted to handle problem $1|C_{MP} \leq d|\sum w_j C_j$.

A possible meaningful interpretation of problem $1|C_{MP} \leq d|\sum w_j C_j$ is that the machine is subject to a compulsory maintenance during the planning period, the length of the maintenance is Δ time units, it must be completed by the deadline d, and the decision-maker has to decide when to start the maintenance period (MP). This problem is closely related to one of the single machine scheduling problems with two competing agents, introduced and studied by Agnetis et al. [2] together with other two-agent scheduling problems. Suppose that two agents intend to use a single machine, Agent A owns the A-jobs, while Agent B owns the B-jobs. Agent A wants to minimize the sum of the weighted completion times of the A-jobs, while Agent B wants to have all the B-jobs completed by a given deadline d. It is easily verified that in any feasible schedule the B-jobs can be processed like a block, without intermediate idle time, and this will not increase the objective function of Agent A. Thus, provided that the processing times and weights of the A-jobs are equal to p_j and w_j, respectively, and the total processing time of the B-jobs is equal to Δ, the two-agent problem is equivalent to problem $1|C_{MP} \leq d|\sum w_j C_j$. Notice that the two-agent problem is proved NP-hard in the ordinary sense [2], but nothing is known on the approximability status of this problem and the equivalent problem $1|C_{MP} \leq d|\sum w_j C_j$.

In the reminder of this section, we treat problem $1|C_{MP} \leq d|\sum w_j C_j$ as a problem of scheduling the MP, rather than a two-agent problem. As before, $Z(S)$ denotes the value of the objective function for a schedule S.

Given problem $1|C_{MP} \leq d|\sum w_j C_j$, introduce the *associated* problem $1|h(1), Res|\sum w_j C_j$, in which the fixed non-availability interval of length Δ is defined by $[s, t] = [d - \Delta, d]$, while the processing times of the jobs and their weights remain equal to p_j and w_j, respectively.

Lemma 4 *Problem* $1|C_{MP} \leq d|\sum w_j C_j$ *and the associated problem* $1|h(1), Res|\sum w_j C_j$ *are equivalent.*

Proof Consider a schedule S_R that is feasible for the associated problem $1|h(1), Res|\sum w_j C_j$, and let in this schedule the crossover job r be processed before the start of the non-availability period for y time units. Transform schedule S_R into a schedule S by swapping the part of the crossover job processed before the non-availability interval with that interval. In schedule S the non-availability period completes at time $d-y$, and is now followed by the whole job r, i.e., S is feasible for the original problem $1|C_{MP}\le d|\sum w_jC_j$. In schedule S, the completion times of all jobs remain as in schedule S_R, so that $Z(S)=Z(S_R)$.

Now, for problem $1|C_{MP}\le d|\sum w_jC_j$, consider a feasible schedule S in which the jobs are processed according to the sequence $\pi=(\pi(1),\dots,\pi(n))$. Assume that the MP of length Δ starts right after the first k jobs are completed, i.e., at time $\sum_{j=1}^{k}p_{\pi(j)}$. If job $\pi(k+1)$ that immediately follows the MP completes by the deadline, it can be swapped with the MP and this will result in a deadline-feasible schedule with a smaller value of the objective function $\sum w_jC_j$, because the completion time of job $\pi(k+1)$ will decrease, while the completion time of each of the remaining jobs stays the same. Thus, without loss of generality, in schedule S the job that immediately follows the MP completes after time d. The processing of job $\pi(k+1)$ can be seen as consisting of two parts: part 1, of y time units, that completes at time d, and part 2, that starts at time d. Interchange part 1 of the processing of job $\pi(k+1)$ with the MP. The obtained schedule S_R is feasible for the associated problem $1|h(1),Res|\sum w_jC_j$, with job $\pi(k+1)$ as the crossover job. In schedule S_R, the completion times of all jobs remain as in schedule S, so that $Z(S_R)=Z(S)$. $\square$

Lemma 4 implies that any schedule feasible for the associated problem $1|h(1),Res|\sum w_jC_j$ can be transformed into a schedule for the original problem $1|C_{MP}\le d|\sum w_jC_j$ and vise versa, without any change in the objective function value. It follows that $Z(S^*)=Z(S_R^*)$, where S^* and S_R^* are optimal schedules for problem $1|C_{MP}\le d|\sum w_jC_j$ and for the associated problem $1|h(1),Res|\sum w_jC_j$, respectively.

Now, let S_R^H be a schedule that is feasible for the associated problem $1|h(1),Res|\sum w_jC_j$ such that $Z(S_R^H)\le\rho Z(S_R^*)$. It immediately follows that schedule S_R^H can be transformed into a schedule S^H that is feasible for the original problem $1|C_{MP}\le d|\sum w_jC_j$, such that

$$Z(S^H)=Z(S_R^H)\le\rho Z(S_R^*)=\rho Z(S^*).$$

This implies that if for the associated problem $1|h(1),Res|\sum w_jC_j$ a ρ-approximation algorithm is available, then the original problem $1|C_{MP}\le d|\sum w_jC_j$ also admits a ρ-approximation algorithm. Thus, the FPTAS for problem $1|h(1),Res|\sum w_jC_j$ can be taken as the basis for the FPTAS for problem $1|C_{MP}\le d|\sum w_jC_j$ as outlined below:

1. Given an instance $1|C_{MP}\le d|\sum w_jC_j$, introduce an instance of the associated problem $1|h(1),Res|\sum w_jC_j$.
2. Run an FPTAS for the obtained problem, output a schedule S_R^{ε}.
3. If in schedule S_R^{ε} no job is interrupted by the non-availability interval, accept this schedule as an approximate solution S^{ε} for the original problem. Otherwise, swap the non-availability interval with the part of the crossover job that is processed before the non-availability interval and output the resulting schedule as schedule S^{ε}.

It is clear that $Z(S^{\varepsilon}) \leq (1+\varepsilon)Z(S^{*})$ and the time required for finding schedule S^{ε} is determined by the running time of the FPTAS for finding schedule S_R^{ε}, i.e., the outlined algorithm is an FPTAS for problem $1|C_{MP} \leq d|\sum w_j C_j$.

5 Conclusion

In this paper we address a special form of the quadratic knapsack problem that we call problem (SQKP) and present an approximation scheme for it solution. This scheme is a fully polynomial-time approximation scheme (FPTAS), provided that a polynomial time approximation algorithm with a bounded worst-case ratio is available. We show how such an FPTAS can be used as a subroutine for several single machine scheduling problems to minimize the sum of the weighted completion times. As a result, for a problem with a single fixed machine non-availability interval we obtain an FPTAS for the version with the non-resumable scenario as well as for the version with the resumable scenario. We also use the FPTAS for the latter problem as the basis for an FPTAS for the problem with a machine maintenance interval that has to completed before a given deadline. It is worth noticing that, the running time of each presented FPTAS in strongly polynomial, i.e., depends on n and $1/\varepsilon$ only. This is not the case for several relevant problems, e.g., those studied in [3] and [8].

We have chosen not to include into this paper our results on constant-ratio approximation algorithm for the convex (SQKP) and its applications for designing an FPTAS for a single machine problem to minimize the total weighted earliness and tardiness with respect to a small common due date; these issues will be topics of a separate paper.

Further research may include the search for the conditions under which the quadratic knapsack problem admits an FPTAS or a constant-ratio approximation algorithm. Finding additional possible applications of the corresponding knapsack problems is also of interest.

Acknowledgement We are grateful to the anonymous referees whose comments have considerably improved the clarity of presentation.

References

1. Adiri, I., Bruno, J., Frostig, E., Rinnooy Kan, A.H.G.: Single machine flow-time scheduling with a single breakdown. Acta Inf. **26**, 679–696 (1989)
2. Agnetis, A., Mirchandani, P.B., Pacciarelli, D., Pacifici, A.: Scheduling problems with two competing agents. Oper. Res. **52**, 229–242 (2004)
3. Badics, T., Boros, E.: Minimization of half-products. Math. Oper. Res. **33**, 649–660 (1998)
4. Breit, J., Schmidt, G., Strusevich, V.A.: Non-preemptive two-machine open shop scheduling with non-availability constraints. Math. Method. Oper. Res. **57**, 217–234 (2003)
5. Kacem, I., Chu, C.: Worst-case analysis of the WSPT and MWSPT rules for single machine scheduling with one planned setup period. Eur. J. Oper. Res. **187**, 1080–1089 (2008)
6. Janiak, A., Kovalyov, M.Y., Kubiak, W., Werner, F.: Positive half-products and scheduling with controllable processing times. Eur. J. Oper. Res. **165**, 416–422 (2005)
7. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2004)
8. Kovalyov, M.Y., Kubiak, W.: A fully polynomial approximation scheme for weighted earliness-tardiness problem. Oper. Res. **47**, 757–761 (1999)

9. Kubiak, W.: New results on the completion time variance minimization. Discrete Appl. Math. **58**, 157–168 (1995)
10. Kubiak, W.: Minimization of ordered, symmetric half-products. Discrete Appl. Math. **146**, 287–300 (2005)
11. Lee, C.-Y.: Machine scheduling with an availability constraint. J. Glob. Optim. **9**, 395–416 (1996)
12. Lee, C.-Y.: Machine scheduling with availability constraints. In: Leung, J.Y.-T. (ed.) Handbook of Scheduling: Algorithms, Models and Performance Analysis, pp. 22-1–22-13. Chapman & Hall/CRC, London (2004)
13. Lee, C.-Y., Liman, S.D.: Single machine flow time scheduling with scheduled maintenance. Acta Inf. **29**, 375–382 (1992)
14. Pisinger, D.: The quadratic knapsack problem—a survey. Discrete Appl. Math. **155**, 623–648 (2007)
15. Sadfi, C., Penz, B., Rapin, C., Błażewicz, J., Formanowicz, P.: An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. Eur. J. Oper. Res. **161**, 3–10 (2005)
16. Smith, W.E.: Various optimizers for single stage production. Nav. Res. Logist. Q **3**, 59–66 (1956)
17. Wang, G., Sun, H., Chu, C.: Preemptive scheduling with availability constraints to minimize total weighted completion times. Ann. Oper. Res. **133**, 183–192 (2005)
18. Woeginger, G.J.: When does a dynamic programming formulation guarantee the existence of an FPTAS? INFORMS J. Comput. **12**, 57–75 (2000)
19. Woeginger, G.J.: Private communication (2005)