Moderate Exponential-time Quantum Dynamic Programming Across the Subsets for Scheduling Problems

Camille Grange, Michael Poss, Eric Bourreau, Vincent t'Kindt, Olivier Ploton

Moderate Exponential-time Quantum Dynamic Programming Across the Subsets for Scheduling Problems*

Camille Grange[1,2], Michael Poss[1], Eric Bourreau[1], Vincent T'kindt[3], and Olivier Ploton[3]

[1]University of Montpellier, LIRMM, CNRS, 161 rue Ada, Montpellier, France

[2]SNCF, Technology, Innovation and Group Projects Department, 1 avenue François Mitterand, Saint-Denis, France

[3]University of Tours, LIFAT, 64 avenue Jean Portalis, Tours, France

Abstract

Grover Search is currently one of the main quantum algorithms leading to hybrid quantum-classical methods that reduce the worst-case time complexity for some combinatorial optimization problems. Specifically, the combination of Quantum Minimum Finding (obtained from Grover Search) with dynamic programming has proved particularly efficient in improving the complexity of NP-hard problems currently solved by classical dynamic programming. For these problems, the classical dynamic programming complexity in $\mathcal{O}^*(c^n)$, where $\mathcal{O}^*$ denotes that polynomial factors are ignored, can be reduced by a hybrid algorithm to $\mathcal{O}^*(c^n_{quant})$, with $c_{quant} < c$. In this paper, we provide a bounded-error hybrid algorithm that achieves such an improvement for a broad class of NP-hard single-machine scheduling problems for which we give a generic description. Moreover, we extend this algorithm to tackle the 3-machine flowshop problem. Our algorithm reduces the exponential-part complexity compared to the best-known classical algorithm, sometimes at the cost of an additional pseudo-polynomial factor.

keywords: Discrete Optimization, Quantum computing, Scheduling, Dynamic Programming

1 Introduction

The fields of quantum computing and combinatorial optimization are becoming every day more closely linked, thanks to the work of the quantum computing community, as well as the more re-

*This is an extension of the conference paper of Grange et al. (2023a).

cent interest of the operations research community that has been focusing on the new quantum paradigm. More precisely, there are two types of quantum algorithms for solving optimization problems. The first type encompasses heuristics, often designed today as hybrid quantum-classical algorithms, such as the class of Variational Quantum Algorithms described by Cerezo et al. (2021) or by Grange et al. (2023b) and, within it, the famous Quantum Approximate Optimization Algorithm (QAOA) of Farhi et al. (2014). Essentially, these algorithms require the optimization problem to be formulated as a QUBO (Quadratic Unconstrained Binary Optimization) and can be implemented on current noisy quantum computers because the quantum part can be made rather small. Among others, the problems of MAX-CUT (Farhi et al., 2014), Travelling Salesman Problem (Ruan et al., 2020), MAX-3-SAT (Nannicini, 2019), Graph Coloring (Tabi et al., 2020) and Job Shop Scheduling (Kurowski et al., 2023) are reformulated as QUBO and solved with hybrid heuristics on small instances. However, due to the small size of instances processed today and the nature of heuristics whose performances are evaluated empirically, no quantum advantage with these heuristics is emerging yet. This is where the second type of quantum algorithms differ: they are *exact* algorithms (i.e. that output the optimal solution with a high probability of success) that provide theoretical speed-ups for several types of problems and algorithms, as displayed by Nannicini (2022) and Sutter et al. (2020). Notice that with the current quantum hardware, it is impossible to implement them today because of the huge size of quantum resources they require.

Grover (1996) provides one key exact quantum algorithm, that achieves a quadratic speed-up when searching for a specific element in an unsorted table, where the complexity is computed as the number of queries of the table and is done by an oracle. Grover Search represents the routine of many exact quantum algorithms. For instance, Durr and Hoyer (1996) use Grover Search as a subroutine for a hybrid quantum-classical algorithm that finds the minimum of an unsorted table, resulting in the algorithm called Quantum Minimum Finding. Later, Ambainis et al. (2019) combine Quantum Minimum Finding with dynamic programming to address NP-hard vertex ordering problems, such as the Traveling Salesman Problem (TSP) or the Minimum Set Cover problem. The problems of interest must satisfy a specific property which implies that they can be solved by classical dynamic programming in $\mathcal{O}^*(c^n)$, where c is usually not smaller than 2. Henceforth, we use $\mathcal{O}^*$ which is the usual asymptotic notation that ignores the polynomial factors in the complexity. The hybrid algorithm of Ambainis et al. (2019) reduces the complexity to $\mathcal{O}^*(c_{\text{quant}}^n)$ for $c_{\text{quant}} < c$. As an example, Held and Karp (1970) dynamic programming solves the TSP in $\mathcal{O}^*(2^n)$ whereas the hybrid algorithm of Ambainis et al. (2019) achieves to solve it in $\mathcal{O}^*(1.728^n)$ by combining the dynamic programming recurrence of Held

and Karp with Quantum Minimum Finding. Following the work of Ambainis et al. (2019), other NP-hard problems have been tackled with the idea of combining Grover Search (or Quantum Minimum Finding) and classical dynamic programming. This has led to quantum speed-ups for the Steiner Tree problem (Miyamoto et al., 2020) and the graph coloring problem (Shimizu and Mori, 2022).

The purpose of this work is to provide a hybrid quantum-classical algorithm, adapting the seminal idea of Ambainis et al. (2019), that reduces the time complexity of solving NP-hard scheduling problems. For that, we propose an extended version of well-known Dynamic Programming Across the Subsets (DPAS) recurrences used to solve combinatorial optimization problems like scheduling problems (see e.g. T'kindt et al. (2022)). Notice that DPAS is a common technique for designing exact algorithms for NP-hard problems as described by Woeginger (2003).

Scheduling problems and DPAS. A scheduling problem lies in finding the optimal assignment of a set of jobs to machines over time. Each job j is defined by at least a processing time p_j and possibly additional data like a due date d_j, a deadline $\tilde{d}_j$, or even a weight w_j reflecting its priority. One or more machines can process the set of jobs, however, at any time point, a machine can only process one job at a time. The computation of a schedule is done to minimize a given objective function.

In Sections 2 and 3, we consider single-machine scheduling problems. Let $[n] := \{1, \ldots, n\}$ be the set of jobs to schedule on the machine. While a solution to a single-machine scheduling problem is described by a starting time for each job on the machine, it is standard to describe instead such a solution by a permutation $\pi \in S_{[n]}$ of the n jobs. Indeed, the starting times can be directly deduced from the order of jobs in the permutation and the potential constraints thanks to the following assumptions. First, we assume that only one job can be processed at any time on the machine. Second, we deal only with non-preemptive scheduling, meaning that a job must be run to completion when it started. Henceforth, we use the permutation representation for the solutions. In Section 5, we consider the 3-machine flowshop of n jobs. The definition of this problem, introduced in the above-mentioned section, makes also a solution entirely described by a permutation of $[n]$ even if there are 3 machines.

Throughout this paper, we use the usual notation $\alpha|\beta|\gamma$, introduced by Graham et al. (1979), to describe the scheduling problem consisting of α machines, with the constraints β and the criterion γ to be minimized. For instance, $1|\tilde{d}_j|\sum_j w_j C_j$ is the problem of minimizing the total weighted completion time with deadline constraints on a single machine. The reader interested in scheduling can refer to any textbook in scheduling, e.g. to Pinedo (2012).

The single-machine scheduling problems addressed in this work are those that satisfy the Dynamic Programming Across the Subsets (DPAS) property. It means that these problems can be solved by Dynamic Programming where the optimal solution for a set of jobs $J \subseteq [n]$ is computed as the best concatenation overall $j \in J$ of the optimal solution for $J \setminus \{j\}$ and the cost of setting j as the last processed jobs. Specifically, if we note $\mathrm{OPT}[J]$ the optimal value for processing the set J of jobs, the recursion is

$$\mathrm{OPT}[J] = \min_{j \in J} \mathrm{OPT}[J \setminus \{j\}] + \phi_j \left(\sum_{k \in J} p_k \right), \tag{1}$$

where ϕ_j is a function depending on job j. This generic recursion captures many single-machine scheduling problems as recalled in the survey of T'kindt et al. (2022), leading to the worst-case time complexity of $\mathcal{O}^*(2^n)$ to solve all these problems. This naturally raises the question of the existence of moderate exponential-time algorithms with a complexity $\mathcal{O}^*(c^n)$ where $c < 2$. The question has been answered positively for specific problems such as minimizing the total weighted completion time with precedence constraints in $\mathcal{O}^*((2-\epsilon)^n)$ for small $\epsilon > 0$ by Cygan et al. (2014). But, as far as we know, no generic method provides such an improvement for a broad class of scheduling problems. In this paper, we present a hybrid algorithm that solves the problems satisfying (1) in $\mathcal{O}^*(1.728^n)$, sometimes with an additional pseudo-polynomial factor in the complexity that comes from the generalization of the recurrence.

Our contributions. We extend existing recurrences for scheduling problems leading to a quantum speed-up for solving a general class of scheduling problems. The dynamic programming recurrences are adapted to solve scheduling problems with a proposed hybrid algorithm Q-DDPAS, which is an extension of the algorithm of Ambainis et al. (2019). In particular, it applies to problems with temporal constraints and non-linear objective functions. Specifically, we cover three types of problems that satisfy three different kinds of dynamic programming properties. For each of them, the best-known classical time complexity is in $\mathcal{O}^*(2^n)$ that is reduced in $\mathcal{O}^*(pseudop \cdot 1.728^n)$ by the hybrid algorithm of this paper, where $pseudop$ is a pseudo-polynomial factor. Not only it applies to problems for which the dynamic programming property is based on the *addition* of optimal values of the problem on sub-instances (as done by Grange et al. (2023a)) but it also relates to problems for which the dynamic programming naturally applies on the *composition* of optimal values of the problem on sub-instances. Furthermore, we address the 3-machine flowshop problem that differs from previous problems by the nature of the recurrence property and widens the range of problems solved by the hybrid algorithm. Last, we also propose an approximation scheme for the 3-machine flowshop problem based on the hybrid algorithm. We summarize in Table 1 the complexities of several NP-hard scheduling

problems through which we illustrate the recurrences in this paper.

Problem	Our hybrid algorithm	Best classical algorithm
$1\|\tilde{d}_j\|\sum w_jC_j$	$\mathcal{O}^*\left(\sum p_j \cdot 1.728^n\right)$	$\mathcal{O}^*(2^n)$ (T'kindt et al., 2022)
$1\|\|\sum w_jT_j$	$\mathcal{O}^*\left(\sum p_j \cdot 1.728^n\right)$	$\mathcal{O}^*(2^n)$ (T'kindt et al., 2022)
$1\|prec\|\sum w_jC_j$	$\mathcal{O}^*\left(1.728^n\right)$	$\mathcal{O}^*((2-\epsilon)^n)$, for small ϵ (Cygan et al., 2014)
$1\|r_j\|\sum w_jU_j$	$\mathcal{O}^*\left((\sum w_j)^3 \cdot \sum p_j \cdot 1.728^n\right)$	$\mathcal{O}^*(\sum w_j \cdot \sum p_j \cdot 2^n)$ (Ploton and T'kindt, 2022)
$1\|r_j\|\sum w_jC_j$	$\mathcal{O}^*\left((\sum w_j)^3 \cdot (\sum p_j)^4 \cdot 1.728^n\right)$	$\mathcal{O}^*(\sum w_j \cdot (\sum p_j)^2 \cdot 2^n)$ (Ploton and T'kindt, 2022)
$F3\|\|C_{\max}$	$\mathcal{O}^*\left((\sum p_{ij})^4 \cdot 1.728^n\right)$	$\mathcal{O}^*(3^n)$ (Shang et al., 2018; Ploton and T'kindt, 2023)

Table 1: Comparison of worst-case time complexities between our hybrid algorithm and the best-known classical algorithm

Structure of the paper. First, we present in Section 2 problems for which the dynamic programming property is based on the *addition* of optimal values of the problem on sub-instances (called Additive DPAS). We begin with the example of problem $1|\tilde{d}_j|\sum_j w_jC_j$ and then provide a generic description of the problems at stake. We describe the related hybrid algorithm (called Q-DDPAS) as it is usually done in the algorithmic quantum literature, namely with a high-level description where quantum *boxes* interact with the classical part. We provide a rigorous and detailed description of the circuit-based implementation for interested readers in our companion paper (Grange et al., 2024). Similarly, we tackle in Section 3 problems for which the dynamic programming property is based on the *addition* of optimal values of the problem on sub-instances (called Composed DPAS), beginning with the example of problem $1|r_j|\sum w_jU_j$. We provide in Section 4 some applications of Q-DDPAS to the scheduling literature. Lastly, in Section 5, we consider the 3-machine flowshop problem, for which the dynamic programming recurrence applies to its decision version. It results in a slightly different hybrid algorithm. Additionally, we provide an approximation scheme for this problem, based on the hybrid algorithm, that disposes of the pseudo-polynomial factor in the time complexity. We recall in Appendix A useful bounds to derive the complexities of the proposed algorithms.

2 Additive DPAS

In this section, we present problems for which the dynamic programming recursion is based on the *addition* of optimal values of problems for sub-instances. Next, we detail the hybrid

algorithm Q-DDPAS to solve these problems.

2.1 A scheduling example

The NP-hard single-machine scheduling problem at stake is the minimization of the total weighted completion time with deadline constraints, often referred to as $1|\tilde{d}_j|\sum_j w_j C_j$ in the scheduling literature. The input is given, for each job $j \in [n]$, by a weight w_j, a processing time p_j and a deadline $\tilde{d}_j$ before which the job must be completed. We define the completion time $C_j(\pi)$ of job j as the end time of the job on the machine for the permutation π. So, if j starts as time t for the permutation π, then $C_j(\pi) = t + p_j$. The problem aims at finding the feasible permutation for which the total weighted sum of completion times is minimal. A permutation π is feasible if $C_j(\pi) \leq \tilde{d}_j$ for all job j. Thus, the problem can be formulated as follows:

$$\min_{\pi \in \Pi} \sum_{j=1}^{n} w_j C_j(\pi)\,,$$

where the set of feasible permutations is $\Pi = \{\pi \in S_{[n]} \,|\, C_j(\pi) \leq \tilde{d}_j\,, \forall j \in [n]\}\,.$

This problem satisfies two recurrences. For deriving them, we need to introduce the set $T := \left[\!\left[0, \sum_{j=1}^{n} p_j\right]\!\right]$, where we use the notation $[\![a, b]\!] = \{a, a+1, \ldots, b\}$ for integers a and b. For $J \subseteq [n]$ and $t \in T$, we define $\mathrm{OPT}[J, t]$ as the optimal value of the problem in which only jobs in J are scheduled from time t. Thus, solving our nominal problem $1|\tilde{d}_j|\sum_j w_j C_j$ amounts to compute $\mathrm{OPT}[[n], 0]$.

The first recurrence comes from the standard Dynamic Programming Across the Subsets (DPAS) described in (1). However, compared to usual DPAS, we introduce an extra parameter t necessary for the solution with our hybrid algorithm as explained later. The idea of this recurrence is to get the optimal value of our problem for jobs in J and starting at time t by finding, over all jobs $j \in J$, the permutation that ends by j with the best cost value. It is possible to do so because no matter what the optimal permutation of the first $(|J| - 1)$ jobs is, the cost of setting job j at the end of the permutation is always known. Indeed, the time taken to process all jobs in $J \setminus \{j\}$ is always $\sum_{k \in J \setminus \{j\}} p_k$. Thus, the completion time of j is defined by $c_j = t + \sum_{k \in J} p_k$. It results that the cost of setting j at the end of the permutation is $w_j(t + \sum_{k \in J} p_k)$. It also implies that the deadline constraint for job j is satisfied if $t + \sum_{k \in J} p_k \leq \tilde{d}_j$. Specifically, for all $J \subseteq [n]$ and for all $t \in T$, we have

$$\mathrm{OPT}[J, t] = \min_{j \in J} \mathrm{OPT}[J \setminus \{j\}, t] + \begin{cases} w_j \left(t + \sum_{k \in J} p_k\right) & \text{if } t + \sum_{k \in J} p_k \leq \tilde{d}_j \\ +\infty & \text{otherwise} \end{cases}, \qquad (2)$$

initialized by $\mathrm{OPT}[\emptyset] = 0$.

The second recurrence generalizes the previous one. For this recurrence, the principle of computing $\mathrm{OPT}[J,t]$ is similar to (2) but instead of setting one job at the end of the permutation, we choose $|J|/2$ jobs and set them to be the half last jobs of the permutation. Specifically, for all $J \subseteq [n]$ of even cardinality and $t \in T$, we have

$$\mathrm{OPT}[J,t] = \min_{\substack{X \subseteq J \\ |X|=|J|/2}} \Big\{ \mathrm{OPT}[X,t] + \mathrm{OPT}[J \setminus X, t + \sum_{i \in X} p_i] \Big\}, \tag{3}$$

initialized by, $\forall j \in [n]$ and $t \in T$, $\mathrm{OPT}[\{j\},t] = \begin{cases} w_j(p_j+t) & \text{if } \tilde{d}_j \geq p_j + t \\ +\infty & \text{otherwise} \end{cases}$.

For a given $X \subseteq J$ of size $|J|/2$, recurrence (3) computes the best permutation of jobs in X starting at time t, and the best permutation of jobs in $J \setminus X$ starting at time $t + \sum_{k \in X} p_k$ as we know that, as before, no matter what is the optimal permutation for jobs in X, the time taken to process them all is exactly $\sum_{k \in X} p_k$.

The two above recurrences have been illustrated with problem $1|\tilde{d}_j|\sum_j w_j C_j$. In the next section, we propose a general formulation of these recurrences that will be used to elaborate our algorithm as general as possible to solve a broad class of scheduling problems.

2.2 General formulation of recurrences

Let us consider the following general scheduling problem:

$$\mathcal{P}: \qquad \min_{\pi \in \Pi} f(\pi),$$

where $\Pi \subseteq S_{[n]}$ is the set of feasible permutations of $[n] := \{1, \ldots, n\}$ according to given constraints and f is the objective function. We introduce a related problem P useful for deriving the dynamic programming recursion, for which we specify the instance: for $J \subseteq [n]$ and $t \in \mathbb{Z}$,

$$P(J,t): \qquad \min_{\pi \in \Pi(J,t)} f(\pi, J, t) \tag{4}$$

as the nominal scheduling problem $\mathcal{P}$ that schedules only jobs in J and starts the schedule at time t. Let us note $\mathrm{OPT}[J,t]$ the optimal value of $P(J,t)$. It results that solving $\mathcal{P}$ amounts to solving $P([n],0)$, and it can be performed by Q-DDPAS if the related problem P satisfies the two recurrences (Add-DPAS) and (Add-D-DPAS) below. Henceforth, we denote by $2^{[n]}$ the set of all subsets of $[n]$, and by $[\![a,b]\!]$ the set $\{a, a+1, \ldots, b\}$. Let us introduce the first recurrence.

Property 2.1 (Additive DPAS)**.** *There exists a function $g : 2^{[n]} \times [n] \times T \to \mathbb{R}$, computable in polynomial time, such that, for all $J \subseteq [n]$ and for all $t_0 \in T$,*

$$\mathrm{OPT}[J,t_0] = \min_{j \in J} \Big\{ \mathrm{OPT}[J \setminus \{j\}, t_0] + g(J,j,t_0) \Big\} \tag{Add-DPAS}$$

initialized by $\mathrm{OPT}[\emptyset, t_0] = 0$.

Lemma 2.2. *Dynamic programming* (Add-DPAS) *solves* $\mathcal{P}$ *in* $\mathcal{O}^*(2^n)$.

Proof. We solve Equation (Add-DPAS) for all J such that $|J| = k$, and for $t_0 = 0$, starting from $k = 1$ to $k = n$. For a given J, the values $\{\mathrm{OPT}[J \setminus \{j\}, 0] : j \in J\}$ are known, so $\mathrm{OPT}[J, 0]$ is computed in time $poly(n) \cdot k$ according to Equation (Add-DPAS) (the computation of g is polynomial). Eventually, the total complexity of computing $\mathrm{OPT}[[n], 0]$ is $\sum_{k=1}^{n} poly(n) k \binom{n}{k} = poly(n) \cdot n \cdot 2^{n-1} = \mathcal{O}^*(2^n)$. □

Throughout, we commit a slight abuse of language by letting (Add-DPAS) both refer to the property satisfied by a given optimization problem and to the resulting dynamic programming algorithm. Notice the presence of the additional parameter t_0 in the above definition, which is typically absent in the scheduling literature. In particular, t_0 is a constant throughout the whole recursion (Add-DPAS) and does not impact the resulting computational complexity. The use of that extra parameter in T shall be necessary later when applying our hybrid algorithm.

Property 2.1 expresses that finding the optimal value of P for jobs in J and starting at time t is done by finding over all jobs $j \in J$ the permutation that ends by j with the best cost value. Function g represents the cost of j being the last job of the permutation. Notice that isolating the last job of the permutation is a usual technique in scheduling as displayed in (1). In the second recurrence below, we provide a similar scheme, where instead of one job, we *isolate* half of the jobs in J, turning the computation of g to the solution of another problem on a sub-instance with $|J|/2$ jobs.

Property 2.3 (Additive Dichotomic DPAS)**.** *There exist two functions* $\tau_{shift} : 2^{[n]} \times 2^{[n]} \times T \to T$ *and* $h : 2^{[n]} \times 2^{[n]} \times T \to \mathbb{R}$*, computable in polynomial time, such that, for all* $J \subseteq [n]$ *of even cardinality, and for all* $t \in T$*,*

$$\mathrm{OPT}[J, t] = \min_{\substack{X \subseteq J \\ |X| = |J|/2}} \Big\{ \mathrm{OPT}[X, t] + h(J, X, t) + \mathrm{OPT}[J \setminus X, \tau_{shift}(J, X, t)] \Big\} \qquad \text{(Add-D-DPAS)}$$

initialized by the values $\mathrm{OPT}[\{j\}, t]$ *for each* $j \in [n]$ *and* $t \in T$*.*

For a given $X \subseteq J$, the above recursion computes the best permutation of jobs in X starting at time t, and the best permutation of jobs in $J \setminus X$ starting at time τ_{shift}, adding the function h that represents the cost of the concatenation between these two permutations.

Remark 2.4. *We observe that problem* (4) *satisfies recurrence* (Add-DPAS) *if and only if it satisfies* (Add-D-DPAS)*. This can be seen by developing recursively both recurrences, which essentially leads to optimization problems over* $\pi \in S_{[n]}$*, whose objective functions respectively*

involve g in the first case and h and τ_{shift} in the second case. Here, one readily verifies that g can then be defined from h and τ_{shift} and reciprocally.

Despite the previous remark, the two recurrences differ on the size of the subsets considered along the recursions, leading to different formulations and therefore require more or less subproblems to be solved optimally in the dynamic programming process. This is formalized in the following proposition. Note that we use the notation $f_1(n) = \omega(f_2(n))$ if f_1 dominates asymptotically f_2.

Lemma 2.5. *Dynamic programming* (Add-D-DPAS) *solves $\mathcal{P}$ in $\omega(|T| \cdot 2^n)$.*

The proof is given in the Supplementary Materials. Notice that if n is not a power of 2, we can still add fake jobs without changing the following conclusion: solving $\mathcal{P}$ with (Add-DPAS) is faster than with (Add-D-DPAS). However, in the next section, we describe a hybrid algorithm Q-DDPAS that improves the complexity of solving $\mathcal{P}$ by combining recurrences (Add-DPAS) and (Add-D-DPAS) with a quantum subroutine.

2.3 Hybrid algorithm for Additive DPAS

In this subsection, we describe our hybrid algorithm Q-DDPAS adapted from the work of Ambainis et al. (2019). Notice that it assumes to have a quantum random access memory (QRAM) (Giovannetti et al., 2008), namely, to have a classical data structure that stores classical information but can answer queries in quantum superposition. We underline that this latter assumption is strong because QRAM is not yet available on current universal quantum hardware. First, let us introduce the Quantum Minimum Finding algorithm of Durr and Hoyer (1996), which constitutes a fundamental subroutine in our algorithm. This algorithm essentially applies several times Grover Search (Grover, 1996) and provides a quadratic speedup for the search of a minimum element in an unsorted table.

Definition 2.6 (Quantum Minimum Finding (Durr and Hoyer, 1996))**.** *Let $f : [n] \to \mathbb{Z}$ be a function. Quantum Minimum Finding computes the minimum value of f and the corresponding minimizer* $\arg\min_{i\in[n]}\{f(i)\}$*. The complexity of Quantum Minimum Finding is $\mathcal{O}\left(\sqrt{n} \cdot C_f(n)\right)$, where $\mathcal{O}(C_f(n))$ is the complexity of computing a value of f.*

Remark 2.7 (Success probability and bounded-error algorithm (Bernstein and Vazirani, 1993))**.** *Durr and Hoyer (1996) prove that Quantum Minimum Finding computes the minimum value with a probability of success strictly larger than $\frac{1}{2}$, independent of n. Thus, for $\epsilon > 0$, finding the minimum value with probability $(1 - \epsilon)$ is achieved by repeating $\mathcal{O}(\log \frac{1}{\epsilon})$ times Quantum*

Minimum Finding. Henceforth, we refer to this statement when we write that Quantum Minimum Finding finds the minimum value with high probability*. Equivalently, we say that this is a* bounded-error *algorithm. More generally, in the rest of the paper, we call a* bounded-error algorithm *an algorithm that provides the optimal solution with a probability as close to 1 as we want by repeating it a number of times independent of the instance size.*

Next, we describe the algorithm of Ambainis et al. (2019) adapted for our Additive DPAS recurrences which implies extra parameters in T. We call it Q-DDPAS and it consists essentially of calling recursively twice Quantum Minimum Finding and computing classically the left terms. Without loss of generality, we assume that 4 divides n. This can be achieved by adding at most three fake jobs and, therefore, does not change the algorithm complexity. Q-DDPAS consists of two steps. First, we compute classically by (Add-DPAS) the optimal values of P on sub-instances of $n/4$ jobs and for all starting times $t \in T$. Second, we call recursively two times Quantum Minimum Finding with (Add-D-DPAS) to find optimal values of P on sub-instances of $n/2$ jobs starting at any time $t \in T$, and eventually of n jobs starting at $t = 0$ (corresponding to the optimal value of the nominal problem $\mathcal{P}$). Specifically, we describe Q-DDPAS in Algorithm 1.

Algorithm 1: Q-DDPAS for Additive DPAS

Input: Problem P satisfying (Add-DPAS) and (Add-D-DPAS)

Output: OPT$[[n], 0]$ with high probability

begin classical part

    for $X \subseteq [n]$ *such that* $|X| = n/4$*, and* $t \in T$ **do**

1   Compute OPT$[X, t]$ with (Add-DPAS) and store the results in the QRAM;

begin quantum part

2   Apply Quantum Minimum Finding with (Add-D-DPAS) to find OPT$[[n], 0]$;

3   To get values for the Quantum Minimum Finding above (the values OPT$[J, t]$ for $J \subseteq [n]$ of size $n/2$ and $t \in T$), apply Quantum Minimum Finding with (Add-D-DPAS);

4   To get values for the Quantum Minimum Finding above (the values OPT$[X, t']$ for $X \subseteq [n]$ of size $n/4$ and $t' \in T$), get them on the QRAM

Theorem 2.8. *The bounded-error algorithm Q-DDPAS (Algorithm 1) solves* $\mathcal{P}$ *in* $\mathcal{O}^*(|T| \cdot 1.754^n)$.

The detailed proof of the correctness of the algorithm, involving the description of the gate

implementation, is detailed in the companion paper Grange et al. (2024), with all the low-level details for implementing the algorithm.

Proof. Hereafter, we provide a high-level proof. The upper bounds to simplify the complexities are detailed in the Appendix A.

- Classical part: computing all $\mathrm{OPT}[X,t]$ for all X of size $n/4$ and for all $t \in T$ (Step 1) is done by (Add-D-DPAS) in time $\mathcal{O}^*\left(|T| \cdot \sum_{k=1}^{n/4} k\binom{n}{k}\right) = \mathcal{O}^*(|T| \cdot 2^{0.811n})$.
- Quantum part: according to Quantum Minimum Finding complexity (Definition 2.6), computing $\mathrm{OPT}[[n],0]$ with Quantum Minimum Finding (Step 2) is done in $\mathcal{O}\left(\sqrt{\binom{n}{n/2}} \cdot C_1(n)\right)$, where $C_1(n)$ is the complexity of computing $\mathrm{OPT}[J,t]$ for a J of size $n/2$ and $t \in T$. *The essence of the quantum advantage here is that we do not need to enumerate all sets J and all time t but we apply the Quantum Minimum Finding in parallel to all at once.* Notice that $\binom{n}{n/2}$ is the number of balanced bi-partitions of $[n]$, namely the number of elements we search over to find the minimum of Equation (Add-D-DPAS) when computing $\mathrm{OPT}[[n],0]$. Thus, $C_1(n)$ is exactly the complexity of Quantum Minimum Finding applied on Step 3, namely $C_1(n) = \mathcal{O}\left(\sqrt{\binom{n/2}{n/4}} \cdot C_2(n)\right)$ where $C_2(n)$ is the complexity of computing $\mathrm{OPT}[X,t']$ for X of size $n/4$ and $t' \in T$. Those values are already computed and stored in the QRAM (Step 1), namely, $C_2(n) = \mathcal{O}^*(1)$. Thus, the quantum part complexity is $\mathcal{O}^*\left(\sqrt{\binom{n}{n/2}\binom{n/2}{n/4}}\right) = \mathcal{O}^*(2^{0.75n})$.

Eventually, Q-DDPAS complexity is the maximum of the classical and the quantum part complexity. Specifically, the total complexity is $\mathcal{O}^*(|T| \cdot 2^{0.811n}) = \mathcal{O}^*(|T| \cdot 1.754^n)$. □

We observe that the complexity of Q-DDPAS can be further reduced by performing a third call to Equation (Add-D-DPAS) as suggested by Ambainis et al. (2019).

Observation 2.9. *A slight modification of Q-DDPAS reduces the complexity to $\mathcal{O}^*(|T| \cdot 1.728^n)$.*

Proof. The slight modification of Q-DDPAS amounts to adding a level of recurrence in the quantum part so that the complexity of the classical part reduces whereas the complexity of the quantum part increases so that both are equal and thus minimize the total complexity. The third call searches for the best concatenation among all the bi-partitions of size $(0.945 \cdot \frac{n}{4}, 0.055 \cdot \frac{n}{4})$ (that are integers asymptotically), i.e. solving

$$\mathrm{OPT}[J,t] = \min_{\substack{X \subseteq J \\ |X| = 0.945|J|}} \left\{ \mathrm{OPT}[X,t] + h(J,X,t) + \mathrm{OPT}[J \setminus X, \tau_{\mathrm{shift}}(J,X,t)] \right\}.$$

The classical part computes all $\mathrm{OPT}[X,t]$ for X of size $0.945 \cdot \frac{n}{4}$ and $0.055 \cdot \frac{n}{4}$, in $\mathcal{O}^*(1.728^n)$. The quantum part applies three levels of recurrence of Quantum Minimum Finding, computing

the minimum over functions with a domain of size $\binom{n}{n/2}$, $\binom{n/2}{n/4}$ and $\binom{n/4}{0.945\cdot n/4}$ respectively. Its complexity is then $\mathcal{O}^*\left(\sqrt{\binom{n}{n/2}\binom{n/2}{n/4}\binom{n/4}{0.945\cdot n/4}}\right) = \mathcal{O}^*(1.728^n)$ (see Appendix A). □

Notice that the classical part of Q-DDPAS can be replaced by any classical algorithm $\mathcal{A}$, if $\mathcal{A}$ computes in $\mathcal{O}^*(|T| \cdot 1.728^n)$ all $\mathrm{OPT}[X,t]$ for $X \subseteq [n]$ of size $n/4$ and $t \in T$. Moreover, if $\mathcal{A}$ happens to reduce the classical part complexity $\mathcal{O}^*(|T| \cdot c^n)$ for $c < 1.728$, the complexity of Q-DDPAS can also be reduced in the same spirit as the slight modification of Observation 2.9.

The application of Q-DDPAS for Additive DPAS to the specific problem $1|\tilde{d}_j|\sum_j w_j C_j$ introduced in Section 2.1 is given in Section 4.1, together with other scheduling examples. Before introducing other types of problems tackled by Q-DDPAS in the next section, we provide some insights to underline why the use of the quantum subroutine Quantum Minimum Finding in Q-DDPAS must be carefully combined with classical computation to achieve a quantum speedup.

Remark 2.10. *Solving $\mathcal{P}$ with* (Add-DPAS) *and replacing each classical computation of the minimum by the quantum subroutine Quantum Minimum Finding would not improve the best classical complexity. Indeed, the complexity would be* $\sum_{k=1}^{n} poly(n)\sqrt{k}\binom{n}{k} = \mathcal{O}^*(2^n)$.

Remark 2.11. *Solving $\mathcal{P}$ exclusively by recursive calls to Quantum Minimum Finding (thus avoiding the classical computations for sets of size $n/4$) would not improve the classical complexity. Using recurrence* (Add-D-DPAS)*, which is the quantum part of Algorithm 1 with roughly* $\log_2(n)$ *recursive calls, would give a complexity in* $\mathcal{O}\left(\sqrt{\binom{n}{n/2}\binom{n/2}{n/4}\dots\binom{2}{1}}\right)$ *that is worse than* $\mathcal{O}^*(2^n)$*. Using recurrence* (Add-DPAS) *would be even worse because it would require n recursive calls leading to the complexity* $\mathcal{O}(\sqrt{n(n-1)\dots 1})$.

3 Composed DPAS

In this section, we study scheduling problems whose constraints enable only the *composition* of problems on sub-instances. We describe the adaptation of Q-DDPAS for these problems.

3.1 A scheduling example

We begin with the specific problem of minimizing the total weighted number of late jobs with release date constraints, often referred to as $1|r_j|\sum w_j U_j$ in the literature. The input is given by, for each job $j \in [n]$, a weight w_j, a processing time p_j, a release date r_j that is the time from which the job can be scheduled (and not before), and a due date d_j that indicates the time after which the job is late. Thus, a job j is late in permutation π if its completion time is larger than d_j, namely if $C_j(\pi) > d_j$. We name $U_j(\pi) = \mathbb{1}_{C_j(\pi)>d_j}$ its indicator function of lateness. This problem aims at finding the feasible permutation, namely where each job starts

after its release date, for which the total weighted number of late jobs is minimal. Thus, the problem can be formulated as follows:

$$\min_{\pi\in\Pi} \sum_{j=1}^{n} w_j U_j(\pi)\,,$$

where the set of feasible solutions is $\Pi = \{\pi \in S_{[n]} \,|\, C_j(\pi) \geq r_j + p_j\}$.

This problem does not satisfy the recurrences (Add-DPAS) and (Add-D-DPAS) because the release date constraints do not allow the addition of sub-instances. Let us take the example of (Add-D-DPAS). The starting time of the second half of jobs $J \setminus X$ in (Add-D-DPAS) can be known only if we know the optimal permutation of the first half job, which is in opposition with the dynamic programming principle. Indeed, the release dates enable empty slots in the scheduling on the first half job such that the time to process all these jobs is not always equal to $\sum_{k\in X} p_k$ and can be larger.

This observation leads to different recurrences, where the time to process the jobs would be known by dynamic programming. For that, we define an auxiliary problem on which the recurrences apply and we introduce a new set of parameters $E := \left[\!\left[0, \sum_{j=1}^{n} w_j\right]\!\right]$. For $J \subseteq [n]$, $t \in T := \left[\!\left[0, \sum_{j=1}^{n} p_j\right]\!\right] \cup \{+\infty\}$ and $\epsilon \in E$, we note $\mathrm{OPT}[J, t, \epsilon]$ the minimum makespan, i.e. the completion time of the last job, for jobs in J beginning at time t where the weighted number of late jobs is exactly ϵ. Notice that by convention, $\mathrm{OPT}[J, t, \epsilon] = +\infty$ if there is no feasible solution, i.e. if $\left\{\pi \in S_J : C_j(\pi) \geq \max(t, r_j) + p_j\,, \forall j \in J \text{ and } \sum_{j\in J} w_j U_j(\pi) = \epsilon\right\} = \emptyset$. Thus, our initial problem $1|r_j|\sum w_j U_j$ is

$$\min_{\epsilon\in E}\{\epsilon : \mathrm{OPT}[[n], 0, \epsilon] < +\infty\}\,.$$

The following recurrence that satisfies the auxiliary problem is inspired by the work of Lawler (1990) for the problem of minimizing the total weighted number of late jobs on a single machine under preemption and release date constraints ($1|r_j, pmtn|\sum w_j U_j$). For $J \subseteq [n]$, $t \in T$, $\epsilon \in E$,

$$\mathrm{OPT}[J, t, \epsilon] = \min_{j\in J}\Big\{\underbrace{\mathrm{OPT}\Big[\{j\}, \mathrm{OPT}[J \setminus \{j\}, t, \epsilon], 0\Big]}_{\text{job } j \text{ is not late}}, \underbrace{\mathrm{OPT}\Big[\{j\}, \mathrm{OPT}[J \setminus \{j\}, t, \epsilon - w_j], w_j\Big]}_{\text{job } j \text{ is late}}\Big\}\,.$$

In this recurrence, for each $j \in J$, we impose j as the last job of the permutation and distinguish two cases, whether it is late or not. Notice that the starting time of j is known and equal to $\mathrm{OPT}[J\setminus\{j\}, t, .]$ which represents the value for the time parameter. The recurrence is initialized

by, for $j \in [n]$, $t \in T$ and $\epsilon \in E$,

$$\mathrm{OPT}[\{j\}, t, \epsilon] = \begin{cases} C_j := \max(t, r_j) + p_j, & \text{if } C_j \leq d_j \text{ and } \epsilon = 0 \\ +\infty, & \text{if } C_j > d_j \text{ and } \epsilon = 0\,, \text{ or if } C_j \leq d_j \text{ and } \epsilon = w_j \\ C_j, & \text{if } C_j > d_j \text{ and } \epsilon = w_j \\ +\infty, & \text{if } \epsilon \in [\![1, w_j - 1]\!] \cup [\![w_j + 1, \sum_{k=1}^{n} w_k]\!] \end{cases}$$

This recurrence generalizes into the following *dichotomic* version for which, instead of setting the last job of the permutation, we set the half-last jobs. For all $J \subseteq [n]$ of even cardinality, $t \in T$ and $\epsilon \in E$,

$$\mathrm{OPT}[J, t, \epsilon] = \min_{\substack{\epsilon' \in E \\ X \in J : |X| = |J|/2}} \Big\{ \mathrm{OPT}\Big[X, \mathrm{OPT}[J \setminus X, t, \epsilon - \epsilon'], \epsilon'\Big] \Big\}\,,$$

initialized by the same values of $\mathrm{OPT}[\{j\}, t, \epsilon]$ for $j \in [n]$, $t \in T$ and $\epsilon \in E$. Next, we provide generic recurrences to consider problems for which the composition of sub-instances is possible.

3.2 General formulation of recurrence

Let us consider a scheduling problem with n jobs

$$\mathcal{P} : \qquad \min_{\pi \in \Pi} f(\pi)\,,$$

where $\Pi \subseteq S_{[n]}$ is the set of feasible permutations of $[n] := \{1, \ldots, n\}$ according to given constraints and f is the objective function. Following the example detailed in the previous subsection, we consider an auxiliary problem $\mathcal{P}'$ useful for deriving the dynamic programming recursion, for which we specify the instance: for $J \subseteq [n]$ the jobs to be scheduled, $t \in \mathbb{Z}$ the starting time of the schedule and $\epsilon \in \mathbb{Z}$, we define

$$P'(J, t, \epsilon) : \qquad \min_{\pi \in \Pi'(J,t,\epsilon)} f'(\pi, J, t, \epsilon)\,, \tag{5}$$

where f', respectively Π', is the objective function, respectively the feasible set, are different from those of $\mathcal{P}$. We assume that solving $\mathcal{P}$ amounts to finding the smallest $\epsilon \in \mathbb{Z}$ such that the auxiliary problem $\mathcal{P}'$ is bounded. Specifically,

$$\mathcal{P} : \quad \min_{\epsilon \in \mathbb{Z}} \Big\{ \epsilon : \mathrm{OPT}[[n], 0, \epsilon] < +\infty \Big\}\,. \tag{6}$$

To solve the nominal problem $\mathcal{P}$ by classical dynamic programming, problem $\mathcal{P}'$ must satisfy recurrence (Comp-DPAS) or recurrence (Comp-D-DPAS) below (as in Remark 2.4, we can state that a problem satisfies one if and only if it satisfies the other one). As we explain later, solving $\mathcal{P}$ with our hybrid algorithm requires problem P' to satisfy the two recurrences.

Property 3.1 (Composed DPAS)**.** *For all $J \subseteq [n]$, $t \in T$ and $\epsilon \in E$,*

$$\mathrm{OPT}[J,t,\epsilon] = \min_{\substack{\epsilon' \in E \\ j \in J}} \Big\{ \mathrm{OPT}\Big[\{j\}, \mathrm{OPT}[J \setminus \{j\}, t, \epsilon - \epsilon'], \epsilon'\Big] \Big\}, \qquad \text{(Comp-DPAS)}$$

initialized by the values of $\mathrm{OPT}[\{j\},t,\epsilon]$ *for all $j \in [n]$, $\epsilon \in E$ and $t \in T$. Notice that for $J \subseteq [n]$, $t \in T$ and $\epsilon \in E$, we adopt the convention* $\mathrm{OPT}[J,t,\epsilon] = +\infty$ *for $\epsilon \notin E$.*

Recurrence (Comp-DPAS) differs from recurrence (Add-DPAS) in two aspects. First, the optimal values of the problem on sub-instances are composed, and not added, because of the nature of the constraints. Second, the search for the minimum value is done not only over all jobs in J, but also over all values in E. More precisely, for a given $\epsilon_0 \in E$, the optimal value of $P'(J,t,\epsilon_0)$ is the minimum value of all possible composition of optimal values of the problem on sub-instances with parameters ϵ_1 and ϵ_2 such that $\epsilon_1 + \epsilon_2 = \epsilon_0$. We have the following result.

Lemma 3.2. (Comp-DPAS) *solves $\mathcal{P}$ in $\mathcal{O}^*(|E|^3 \cdot |T| \cdot 2^n)$.*

Proof. Let $\epsilon_0 \in E$. Similarly to the proof of Lemma 2.2, we show that (Comp-DPAS) solves $P'([n],0,\epsilon_0)$ in $\mathcal{O}^*(|E|^2 \cdot |T| \cdot 2^n)$. Indeed, to compute $\mathrm{OPT}[[n],0,\epsilon_0]$, we need to solve Equation (Comp-DPAS) for all J such that $|J| = k$ starting from $k = 1$ to $k = n$, and for all $t \in T$ and $\epsilon \in E$. For a given J, $t \in T$ and $\epsilon \in E$, the values $\{\mathrm{OPT}[J \setminus \{j\}, t', \epsilon'] : j \in J, t' \in T, \epsilon' \in E\}$ and $\{\mathrm{OPT}[\{j\}, t', \epsilon'] : j \in J, t' \in T, \epsilon' \in E\}$ are known, so $\mathrm{OPT}[J,t,\epsilon]$ is computed in time $|E| \cdot k$ according to Equation (Comp-DPAS). Eventually, the total complexity of computing $\mathrm{OPT}[[n],0,\epsilon_0]$ is $\sum_{k=1}^{n} |T| \cdot |E|^2 \cdot k\binom{n}{k} = \mathcal{O}^*(|T| \cdot |E|^2 \cdot 2^n)$. Moreover, solving $\mathcal{P}$ amounts to solving $P'([n],0,\epsilon)$,for all $\epsilon \in E$, according to (6). The complexity results directly from the above complexity of computing $\mathrm{OPT}[[n],0,\epsilon_0]$, for $\epsilon_0 \in E$. □

The auxiliary problem P' must satisfy the following recurrence (Comp-D-DPAS) in addition to recurrence (Comp-DPAS).

Property 3.3 (Composed Dichotomic DPAS)**.** *For all $J \subseteq [n]$ of even cardinality, $t \in T$ and $\epsilon \in E$,*

$$\mathrm{OPT}[J,t,\epsilon] = \min_{\substack{\epsilon' \in E \\ X \in J : |X| = |J|/2}} \Big\{ \mathrm{OPT}\Big[X, \mathrm{OPT}[J \setminus X, t, \epsilon - \epsilon'], \epsilon'\Big] \Big\}, \qquad \text{(Comp-D-DPAS)}$$

initialized by the values of $\mathrm{OPT}[\{j\},t,\epsilon]$ *for all $j \in [n]$, $t \in T$ and $\epsilon \in E$.*

Lemma 3.4. (Comp-D-DPAS) *solves $\mathcal{P}$ in $\omega(|E|^3 \cdot |T| \cdot 2^n)$.*

Proof. This proof is essentially the same as the one of Lemma 2.5 with the same modifications that for the proof of Lemma 3.2. □

As for the Additive DPAS, we notice that, with a classical dynamic programming algorithm, the time complexity to solve $\mathcal{P}$ with recurrence (Comp-DPAS) is better than with recurrence (Comp-D-DPAS). Next, we show that the hybrid algorithm applied to problems satisfying Additive DPAS recurrences can be easily adapted to tackle problems satisfying Composed DPAS recurrences.

3.3 Hybrid algorithm for Composed DPAS

The hybrid algorithm for Composed DPAS derives naturally from Algorithm 1. It amounts to replacing the recurrence (Add-DPAS), respectivelly (Add-D-DPAS), by (Comp-DPAS), respectivelly (Comp-D-DPAS), resulting in Algorithm 2. Eventually, we use Algorithm 2 as a subroutine to solve Equation (6), i.e. to solve the nominal problem $\mathcal{P}$.

Algorithm 2: Q-DDPAS for Composed DPAS

Input: $\epsilon_0 \in E$, auxiliary problem P' satisfying (Comp-DPAS) and (Comp-D-DPAS)

Output: $\mathrm{OPT}[[n], 0, \epsilon_0]$ with high probability

begin classical part

  for $X \subseteq [n]$ *such that* $|X| = n/4$, *and* $t \in T$ **do**

    Compute $\mathrm{OPT}[X, t, \epsilon_0]$ with (Comp-DPAS) and store the results in the QRAM;

begin quantum part

  Apply Quantum Minimum Finding with (Comp-D-DPAS) to find $\mathrm{OPT}[[n], 0, \epsilon_0]$;

  To get values for the Quantum Minimum Finding above (the values $\mathrm{OPT}[J, t, \epsilon]$ for $J \subseteq [n]$ of size $n/2$, $t \in T$ and $\epsilon \in E$), apply Quantum Minimum Finding with (Comp-D-DPAS);

  To get values for the Quantum Minimum Finding above (the values $\mathrm{OPT}[X, t', \epsilon']$ for $X \subseteq [n]$ of size $n/4$, $t' \in T$ and $\epsilon' \in E$), get them on the QRAM

Lemma 3.5. *Let $\epsilon_0 \in E$. The bounded-error algorithm Q-DDPAS (Algorithm 2) solves $P'([n], 0, \epsilon_0)$ in $\mathcal{O}^*(|E|^2 \cdot |T| \cdot 1.754^n)$.*

Notice that the implementation of this algorithm is slightly different from the one of Algorithm 1, mainly due to the operation of composition. The details are given in the companion paper (Grange et al., 2024).

Theorem 3.6. *The bounded-error Algorithm 3, with Q-DDPAS as a subroutine, solves $\mathcal{P}$ in $\mathcal{O}^*(|E|^3 \cdot |T| \cdot 1.754^n)$.*

Algorithm 3: Meta-algorithm with subroutine Q-DDPAS for Composed DPAS

Input: Auxiliary problem P' satisfying (Comp-DPAS) and (Comp-D-DPAS)

Output: $\min_{\epsilon \in E}\left\{\epsilon : \mathrm{OPT}[[n], 0, \epsilon] < +\infty\right\}$ with high probability

```
ε* ← +∞;
for ε ∈ E do
    Solve P([n], 0, ε) with Algorithm 2;
    if OPT[[n], 0, ε] < +∞ and ε < ε* then
        ε* ← ε;
Return ε*
```

As for the case of Q-DDPAS for Additive DPAS, we can reduce the exponential part of Q-DDPAS complexity for Composed DPAS, by the modification indicated in Observation 2.9, thus leading to the following observation.

Observation 3.7. *A slight modification of the Q-DDPAS algorithm can reduce the complexity of Algorithm 3 to* $\mathcal{O}^*(|E|^3 \cdot |T| \cdot 1.728^n)$.

We illustrate in Section 4.2 the application of Q-DDPAS for Composed DPAS to the problem $1|r_j|\sum w_j U_j$ described in Section 3.1, together with another similar scheduling problem.

4 Application to the scheduling literature

In Section 2.2 and Section 3.2, we provided general formulations of problems satisfying Additive and Composed DPAS recurrences. Next, we illustrate these recurrences with several NP-hard single-machine scheduling problems enabling their resolution with our hybrid algorithm Q-DDPAS. The list of problems is non-exhaustive but highlights the structures' specificity of scheduling problems that enable such recurrences. Eventually, for each problem, we compare in Table 1 the worst-case time complexity of Q-DDPAS with the complexity of the best-known moderate exponential-time exact algorithm. Q-DDPAS improves the exponential-part complexity, sometimes at the cost of an additional pseudo-polynomial factor.

4.1 Scheduling with deadlines and precedence constraints

Single-machine scheduling problems with no constraints, deadline constraints or precedence constraints satisfy the *addition* of optimal values of the problem on sub-instances. We provide next several examples of problems that satisfy Additive DPAS and thus can be solved by Q-

DDPAS (Algorithm 1).

In Subsection 2.1, we have presented the problem of minimizing the total weighted completion time with deadline constraints ($1|\tilde{d}_j|\sum_j w_jC_j$). The formulation needed the set T to be equal to $[\![0, \sum_{j=1}^{n} p_j]\!]$, hence its resolution with Q-DDPAS is in $\mathcal{O}^*(\sum p_j \cdot 1.728^n)$ according to Observation 2.9. Next, we give two more examples, beginning with the strongly NP-hard scheduling problem with minimization of the total weighted tardiness. Henceforth, we note $p(J) = \sum_{j\in J} p_j$ the sum of processing times of the jobs in $J \subseteq [n]$.

Example 1 (Minimizing the total weighted tardiness, $1||\sum_j w_jT_j$)**.** *For each job $j \in [n]$, we are given a weight w_j, a processing time p_j, and a due date d_j that indicates the time after which the job is late. Thus, a job j is late in permutation π if its completion time is larger than d_j, and we define as $T_j(\pi) = \max(0, C_j(\pi) - d_j)$ its tardiness. Our problem aims at finding the permutation that minimizes the total weighted tardiness, referred to as $1||\sum_j w_jT_j$ in the scheduling literature. Let $T = [\![0, \sum_{j=1}^{n} p_j]\!]$ be the set of all possible starting times. We define the related problem P of Equation* (4) *as follows: for $J \subseteq [n]$ and $t \in T$, $\Pi(J,t) = S_J$, and for $\pi \in \Pi(J,t)$:*

$$f(\pi, J, t) = \sum_{j\in J} w_j \max(0, C_j(\pi) - d_j + t)\,,$$

where $\max(0, C_j - d_j + t)$ *represents the tardiness of job j for the* effective due date $(d_j - t)$*. Problem $1||\sum_j w_jT_j$ satisfies both Additive DPAS recurrences. Indeed, Equation* (Add-DPAS) *is valid with: $\forall J \subseteq [n], \forall j \in J, \forall t \in T$,*

$$g(J, j, t) = w_j \max(0, p(J) - d_j + t)\,,$$

where the computation of g is polynomial (linear). Moreover, Equation (Add-D-DPAS) *is valid for the following functions: $\forall X \subseteq J \subseteq [n]$ s.t. $|X| = |J|/2, \forall t \in T$,*

$$\tau_{shift}(J, X, t) = t + p(X) \qquad and \qquad h(J, X, t) = 0 \tag{7}$$

initialized by, for $j \in [n]$ and $t \in T$, $\mathrm{OPT}[\{j\}, t] = w_j \max(0, p_j - d_j + t)\,.$

We consider the scheduling problem with precedence constraints and minimization of the total weighted completion time that is also NP-hard. Conversely to the two previous examples, the set T is reduced to $\{0\}$, and function h translates the potential infeasibility of the concatenation of problem P on two sub-instances.

Example 2 (Minimizing the total weighted completion time with precedence constraints, $1|prec|\sum_j w_jC_j$)**.** *We are given, for each job $j \in [n]$, a processing time p_j, a weight w_j, and a set of precedence constraints $K = \{(i,j) : i \prec j\}$. A pair of jobs (i,j) in K implies that i must*

precede j in the permutation, namely that i must be processed before j. Our problem, denoted by $1|prec|\sum_j w_jC_j$, aims at finding the feasible permutation, i.e. that respects the precedence constraints, that minimizes the total weighted completion time. Let be $T = \{0\}$. Here, an instance of the problem P of Equation (4) under consideration is only indexed by the chosen subset of $[n]$. Thus, we consider the problem P as follows: for $J \subseteq [n]$, $\Pi(J,0) = \{\pi \in S_J \,|\, \pi \text{ respects } K\}$, and for $\pi \in \Pi(J,0)$: $f(\pi, J, 0) = \sum_{j\in J} w_jC_j(\pi)$. Our problem $1|prec|\sum_j w_jC_j$ satisfies both Additive DPAS recurrences. Indeed, Equation (Add-DPAS) *is valid for:*

$$\forall J \subseteq [n], \forall j \in J, \quad g(J,j,0) = \begin{cases} +\infty & \text{if } \exists (j,k) \in E | k \in J \\ w_j p(J) & \text{otherwise} \end{cases},$$

where the computation of g is polynomial (quadratic). This problem P also satisfies (Add-D-DPAS). *Indeed, Equation* (Add-D-DPAS) *is valid for the following functions: $\forall X \subseteq J \subseteq [n]$ such that $|X| = |J|/2$,*

$$\tau_{shift}(J,X,0) = 0 \qquad \text{and} \qquad h(J,X,0) = \begin{cases} +\infty & \text{if } \exists (j,k) \in E | j \in J \setminus X \text{ and } k \in X \\ p(X) \cdot \sum\limits_{j \in J\setminus X} w_j & \text{otherwise} \end{cases}$$

where the computation of h is polynomial (quadratic). The initialization is, for $j \in [n]$, $\mathrm{OPT}[\{j\}, 0] = w_j p_j$.

The three NP-hard scheduling problems examples described above can be solved with Q-DDPAS for Additive DPAS (Algorithm 1). We illustrate in Table 2 the worst-case time complexities of solving them with Q-DDPAS and compare them with the complexities of the best-known exact classical algorithms. Q-DDPAS improves the complexity of the exponent but sometimes at the cost of a pseudo-polynomial factor.

Problem	Q-DDPAS for Additive DPAS	Best classical algorithm
$1\|\tilde{d}_j\|\sum w_jC_j$	$\mathcal{O}^*\left(\sum p_j \cdot 1.728^n\right)$	$\mathcal{O}^*(2^n)$ (T'kindt et al., 2022)
$1\|\|\sum w_jT_j$	$\mathcal{O}^*\left(\sum p_j \cdot 1.728^n\right)$	$\mathcal{O}^*(2^n)$ (T'kindt et al., 2022)
$1\|prec\|\sum w_jC_j$	$\mathcal{O}^*\left(1.728^n\right)$	$\mathcal{O}^*((2-\epsilon)^n)$, for small ϵ (Cygan et al., 2014)

Table 2: Comparison of complexities between Q-DDPAS and the best-known classical algorithm for some scheduling problems satisfying (Add-DPAS) and (Add-D-DPAS)

4.2 Scheduling with release date constraints

Single-machine scheduling problems with release date constraints do not satisfy the *addition* of optimal values of the problem on sub-instances but enable the *composition* of them. We illustrate this notion with two examples of problems that satisfy Composed DPAS and thus can be solved by Q-DDPAS (Algorithm 2).

We have presented in Subsection 3.1 an example that is the problem of minimizing the weighted number of late jobs with release date constraints ($1|r_j|\sum w_jU_j$). We have shown that the two sets to define the auxiliary problem are $E = [\![0, \sum_{j=1}^{n} w_j]\!]$ and $T = [\![0, \sum_{j=1}^{n} p_j]\!] \cup \{+\infty\}$. Thus, Q-DDPAS solves this problem in $\mathcal{O}^*\left((\sum w_j)^3 \cdot \sum p_j \cdot 1.728^n\right)$ according to Observation 3.7. Next, we present another example which is the strongly NP-hard problem of minimizing the total weighted completion time with release date constraints.

Example 3 (Minimizing the total weighted completion time with release date constraints, $1|r_j|\sum w_jC_j$)**.** *Each job $j \in [n]$ has a weight w_j, a processing time p_j, and a release date r_j. This problem aims at finding the feasible permutation, namely where each job starts after its release date, for which the total weighted completion time is minimal. Let $T = [\![0, \sum_{j=1}^{n} p_j]\!] \cup \{+\infty\}$ and $E = [\![0, \sum_{j=1}^{n} w_j \cdot \sum_{j=1}^{n} p_j]\!]$. For a given $\epsilon \in E$, we consider the problem P' of Equation* (5) *as follows: $\forall J \subseteq [n], t \in T$,*

$$P'(J,t,\epsilon): \quad \min_{\pi \in \Pi'(J,t,\epsilon)} C_{max}(\pi)\,,$$

where C_{max} is the makespan, and

$$\Pi'(J,t,\epsilon) = \{\pi \in S_J : C_j(\pi) \geq \max(t, r_j) + p_j \text{ and } \sum_{j \in J} w_jC_j(\pi) = \epsilon\}\,,$$

where C_j is the completion time of job j. Problem P' satisfies the two Composed DPAS recurrences (Comp-DPAS) *and* (Comp-D-DPAS)*. The initialization of the recurrences is, for $j \in [n]$, $t \in T$ and $\epsilon \in E$,*

$$\mathrm{OPT}[\{j\}, t, \epsilon] = \begin{cases} C_j := \max(t, r_j) + p_j, & \text{if } \epsilon = w_jC_j \\ +\infty, & \text{otherwise} \end{cases}$$

We synthesize in Table 3 the worst-case time complexities achieved by Q-DDPAS on the examples of scheduling problems satisfying the Composed DPAS recurrences. We compare them with the best-known classical complexities for exact algorithms. The latter comes from the algorithm of Inclusion-Exclusion designed by Ploton and T'kindt (2022), which provides a generic method to solve such problems. We observe that Q-DDPAS improves the exponential part of the complexity, at a cost of a higher degree for the pseudo-polynomial factor.

Problem	Q-DDPAS for Composed DPAS	Best classical algorithm
$1\|r_j\|\sum w_jU_j$	$\mathcal{O}^*\left((\sum w_j)^3 \cdot \sum p_j \cdot 1.728^n\right)$	$\mathcal{O}^*(\sum w_j \cdot \sum p_j \cdot 2^n)$, (Ploton and T'kindt, 2022)
$1\|r_j\|\sum w_jC_j$	$\mathcal{O}^*\left((\sum w_j)^3 \cdot (\sum p_j)^4 \cdot 1.728^n\right)$	$\mathcal{O}^*(\sum w_j \cdot (\sum p_j)^2 \cdot 2^n)$, (Ploton and T'kindt, 2022)

Table 3: Comparison of complexities between Q-DDPAS and the best-known classical algorithm for some scheduling problems satisfying (Comp-DPAS) and (Comp-D-DPAS)

5 Decision-based DPAS

We saw in the previous section that the recurrence to solve $\mathcal{P}$ can be applied to a minimization problem, possibly involving an auxiliary problem. Sometimes, the recurrence does not apply directly to a minimization problem but to a *decision* problem. This is the case of the 3-machine flowshop problem. In this section, we adapt the hybrid algorithm Q-DDPAS to solve this problem. Notice that it easily generalizes to the m-flowshop problem, for $m \geq 4$.

5.1 3-machine flowshop and dynamic programming

We consider the permutation flowshop problem on 3 machines for n jobs with minimizing the makespan as the objective function. This strongly NP-hard problem is often referred to as $F3||C_{\max}$ in the literature, as mentioned by Shang et al. (2018). Each job $j \in [n]$ consists of 3 operations O_{ij} for $i \in [3]$, each operation being processed on the i-th machine. We note p_{ij} the processing time of operation O_{ij}. Each machine performs at most one operation at a time. For each job j, operations must be processed in the specific order O_{1j}, O_{2j}, O_{3j}: the first operation gets processed on the first machine, then the second operation gets processed on second machine (as soon as the first operation is finished and the second machine is available), and eventually the third operation gets processed on the third machine (as soon as the second operation is finished and the third machine is available). Thus, only the processing order of the jobs has to be decided, implying that a solution is entirely described by the permutation of jobs on the first machine. Thus, the problem can be formulated as

$$\min_{\pi \in S_{[n]}} C_{\max}(\pi)\,, \tag{8}$$

where $C_{\max}$ is the maximum completion time, namely the completion time of the last job processed on the last machine (third machine). Because the two techniques presented so far do not apply to (8), we present an alternative approach involving the decision counterpart of the above optimization problem.

We introduce below a decision problem for deriving the recurrences. For that, we define the

bounded set

$$T = \left[\!\left[0, \sum_{j\in[n], i\in[3]} p_{ij} \right]\!\right] \subseteq \mathbb{Z}\,.$$

Definition 5.1 (Decision problem)**.** *For $J \subseteq [n]$, $\vec{\beta} = (\beta_2, \beta_3) \in T^2$ and $\vec{\epsilon} = (\epsilon_2, \epsilon_3) \in T^2$, we define the decision problem $D(J, \vec{\beta}, \vec{\epsilon})$ on a sub-instance associated with jobs in J as the following question: "Does there exist a permutation $\pi \in S_J$ such that, for $i \in \{2, 3\}$, $b_i(\pi) \geq \beta_i$, and $e_i(\pi) \leq \epsilon_i$?", where $b_i(\pi)$, respectively $e_i(\pi)$, denotes the time at which the first operation begins, respectively the last operation ends, on the i-th machine.*

In other words, problem $D(J, \vec{\beta}, \vec{\epsilon})$ asks whether or not there exists a feasible permutation with jobs in J such that it *holds* between the two *temporal fronts* $\vec{\beta}$ and $\vec{\epsilon}$. Notice that it is not necessary to impose any beginning and ending time for the first machine ($i = 1$). Indeed, the problem is time-invariant, thus we can always consider that the scheduling problem starts at time 0, and that the total completion time on the first machine is known and equal to the sum of processing times of the scheduled jobs. Notice that the number of parameters is four for the 3-machine flowshop, but generalizes to $2(m-1)$ parameters for the m-machine flowshop.

With these notations, $\mathcal{P}$ can be cast as follows:

$$\mathcal{P}: \qquad \min_{c\in T}\Big\{ c : D[[n], (0,0), (c,c)] = \text{True} \Big\}\,. \tag{9}$$

The decision problem D satisfies both recurrences (Dec-DPAS) and (Dec-D-DPAS) below.

Property 5.2 (Decision DPAS)**.** *For all $J \subseteq [n]$ of even cardinality, $\vec{\beta} \in T^2$ and $\vec{\epsilon} \in T^2$,*

$$D[J, \vec{\beta}, \vec{\epsilon}] = \bigvee_{\substack{X\subseteq J : |X| = |J|/2, \\ \vec{t}\in[\vec{\beta},\vec{\epsilon}]}} \Big(D[\{j\}, \vec{\beta}, \vec{t}] \wedge D[J \setminus \{j\}, \vec{t} \ominus p_{1j}, \vec{\epsilon} \ominus p_{1j}] \Big)\,, \tag{Dec-DPAS}$$

where $\vec{t} \in [\vec{\beta}, \vec{\epsilon}]$ means that the i-th coordinate of $\vec{t}$ is between the i-th coordinates of $\vec{\beta}$ and $\vec{\epsilon}$, and where operation $\vec{v} \ominus c$, for a vector $\vec{v}$ and a constant c, subtracts c to each coordinate of $\vec{v}$.

This latter recurrence enables $\mathcal{P}$ to be solved by a classical dynamic programming algorithm.

Lemma 5.3. (Dec-DPAS) *solves $\mathcal{P}$ in $\mathcal{O}^*(|T|^4 \cdot 2^n)$.*

Proof. First, we can show that, for a given $\vec{\beta}_0, \vec{\epsilon}_0 \in T^2$, (Dec-DPAS) solves $D([n], \vec{\beta}_0, \vec{\epsilon}_0)$ in $\mathcal{O}^*(|T|^4 \cdot 2^n)$. This is essentially the same lines of the proof as in Lemma 2.2. Second, to solve $\mathcal{P}$, we make a dichotomic search over T to find the minimum $c \in T$ such that $D([n], (0, c), (0, c))$ is True according to Equation (9). Thus, (Dec-DPAS) is called $\log_2(|T|)$ times. Because $|T| = \sum p_{ij}$ is a pseudo-polynomial term of the instance, the total complexity is $\mathcal{O}^*(\log_2(|T|) \cdot |T|^4 \cdot 2^n) = \mathcal{O}^*(|T|^4 \cdot 2^n)$. □

Property 5.4 (Decision Dichotomic DPAS)**.** *For all $J \subseteq [n]$ of even cardinality, $\vec{\beta} \in T^2$ and $\vec{\epsilon} \in T^2$,*

$$D[J, \vec{\beta}, \vec{\epsilon}] = \bigvee_{\substack{X \subseteq J : |X| = |J|/2, \\ \vec{t} \in [\vec{\beta}, \vec{\epsilon}]}} \left(D[X, \vec{\beta}, \vec{t}] \wedge D[J \setminus X, \vec{t} \ominus \sum_{j \in X} p_{1j}, \vec{\epsilon} \ominus \sum_{j \in X} p_{1j}] \right) . \quad \text{(Dec-D-DPAS)}$$

Lemma 5.5. (Dec-D-DPAS) *solves $\mathcal{P}$ in $\omega(|T|^4 \cdot 2^n)$.*

Proof. This proof is similar to the proof of Lemma 2.5, with the argument that a dichotomic search is polynomial in the size of the instance as in the proof of Lemma 5.3. □

Once again, we observe that recurrence (Dec-DPAS) outperforms recurrence (Dec-D-DPAS) to solve by classical dynamic programming our problem $\mathcal{P}$. In the next section, we describe how we adapt Q-DDPAS to take advantage of those two recurrences to solve the 3-machine flowshop problem.

5.2 Hybrid algorithm for Decision-based DPAS

We call Q-Dec-DDPAS the adapted decision version of Q-DDPAS. The main difference is that instead of searching for a minimum value in a set in recurrence (Add-D-DPAS) or (Comp-D-DPAS), we search for a True value in a set in recurrence (Dec-D-DPAS). Thus, it essentially amounts to replacing Quantum Minimum Finding with the algorithm of Boyer et al. (1998) specified below, which extends Grover Search (Grover, 1996).

Definition 5.6 (Grover Search Extension (Boyer et al., 1998))**.** *Let $f : [n] \to \{0, 1\}$ be a function. Grover Search Extension computes with high probability the logical OR of all the f values and the corresponding antecedent(s) $x \in [n]$ such that $f(x) = 1$. The complexity of Grover Search Extension is $\mathcal{O}\left(\sqrt{n} \cdot C_f(n)\right)$, where $\mathcal{O}(C_f(n))$ is the complexity of computing a value of f.*

Notice that we cannot use Grover Search because we do not know the number of x such that $f(x) = 1$. The generalization of Boyer et al. (1998) enables us to deal with an unknown number of solutions while keeping the same complexity of Grover Search. Moreover, if there are $t \in \mathbb{N}^*$ solutions, the complexity is $\mathcal{O}\left(\sqrt{n/t} \cdot C_f(n)\right)$ but, having no bounds on t whenever we call Grover Search Extension, we omit it in the complexity.

Lemma 5.7. *Let $\vec{\beta}_0, \vec{\epsilon}_0 \in T^2$. The bounded-error algorithm Q-Dec-DDPAS (Algorithm 4) solves $D([n], \vec{\beta}_0, \vec{\epsilon}_0)$ in $\mathcal{O}^*((\sum p_{ij})^4 \cdot 1.754^n)$.*

Algorithm 4: Q-Dec-DDPAS for 3-machine flowshop

Input: $\vec{\beta}_0, \vec{\epsilon}_0 \in T^2$, decision problem D satisfying (Dec-DPAS) and (Dec-D-DPAS)

Output: $D[[n], \vec{\beta}_0, \vec{\epsilon}_0]$ with high probability

begin classical part

  for $X \subseteq [n] : |X| = n/4$ *and* $\vec{\beta}, \vec{\epsilon} \in T^2$ **do**

    Compute $D[X, \vec{\beta}, \vec{\epsilon}]$ with (Dec-DPAS) and store the results in the QRAM;

begin quantum part

  Apply Grover Search Extension with (Dec-D-DPAS) to find $D[[n], \vec{\beta}_0, \vec{\epsilon}_0]$;

  To get values for the Grover Search Extension above (the values $D[J, \vec{\beta}, \vec{\epsilon}]$ for $J \subseteq [n]$ of size $n/2$ and $\vec{\beta}, \vec{\epsilon} \in T$), apply Grover Search Extension with (Dec-D-DPAS);

  To get values for Grover Search Extension above (the values $D[X, \vec{\beta}', \vec{\epsilon}']$ for $X \subseteq [n]$ of size $n/4$ and $\vec{\beta}', \vec{\epsilon}' \in T$), get them on the QRAM;

The proof is essentially the same as for Theorem 2.8, detailed in Supplementary Materials. All the details of correctness and low-level implementation are given in our companion paper (Grange et al., 2024).

Algorithm 5: Meta-algorithm with subroutine Q-Dec-DDPAS for the 3-machine flowshop

Input: 3-machine flowshop

Output: Minimum makespan with high probability

1 $c^* \leftarrow +\infty$;

2 **for** $c \in T$ **do**

3 Solve $D([n], (0,0), (c,c))$ with Algorithm 4;

4 **if** $D[[n], (0,0), (c,c)]$ *and* $c < c^*$ **then**

5 $c^* \leftarrow c$;

6 Return c^*

Theorem 5.8. *The bounded-error Algorithm 5 solves the 3-machine flowshop in* $\mathcal{O}^*((\sum p_{ij})^4 \cdot 1.754^n)$ *with high probability.*

Once again, as mentioned in Observation 2.9, the complexity can be reduced thanks to a slight modification on the Q-Dec-DDPAS that constitutes the subroutine, thus leading to the following observation.

Observation 5.9. *A slight modification of Algorithm 5 reduces the complexity of solving the 3-machine flowshop in* $\mathcal{O}^*((\sum p_{ij})^4 \cdot 1.728^n)$ *with high probability.*

This new method improves the best-known classical algorithm that is in $\mathcal{O}^*(3^n)$ or in $\mathcal{O}^*(M \cdot 2^n)$ if there exists a constant M such that $p_{ij} \leq M$, for all $i \in [3], j \in [n]$, presented by Shang et al. (2018) and Ploton and T'kindt (2023). Hybrid quantum-classical bounded-error Algorithm 5 reduces the exponential part of the time complexity at the cost of a pseudo-polynomial factor. For most cases, this factor is negligible because the numerical values of 3-machine flowshop instances are small compared to the exponential part value. However, we present in the next subsection a way to dispose of this factor with an approximation scheme.

It is worth noting that the previous algorithm easily generalizes to the m-machine flowshop problem. Indeed, the only difference is the description of the *temporal front* that necessitates $2(m-1)$ parameters.

Observation 5.10. *The bounded-error Algorithm 5 generalizes to solve the* m*-machine flowshop in* $\mathcal{O}^*((\sum p_{ij})^{2(m-1)} \cdot 1.728^n)$ *with high probability.*

Notice that Ploton and T'kindt (2023) present a classical resolution for the m-machine flowshop by Inclusion-Exclusion in $\mathcal{O}^*((\sum p_{ij})^m \cdot 2^n)$.

5.3 Approximation scheme for the 3-machine flowshop

We present an approximation scheme for the 3-machine flowshop problem that trades the pseudo-polynomial factor in the complexity of Q-Dec-DDPAS and the optimality of the algorithm for a polynomial factor in $\frac{1}{\epsilon}$ and an approximation factor of $(1+\epsilon)$. In other words, we provide Algorithm 6 that finds a solution in time $\mathcal{O}^*\left(\frac{1}{\epsilon^3} \cdot 1.728^n\right)$ for which the makespan is not greater than $(1+\epsilon)$ times the optimal makespan. The latter point denotes that this is an ϵ-approximation scheme. Our algorithm belongs to the class of moderate exponential-time approximation algorithms. Notice that the 3-machine flowshop problem does not admit an FPTAS (fully polynomial-time approximation scheme) because it is strongly NP-hard, meaning that no ϵ-approximation algorithm exists to solve the 3-machine flowshop in time $\mathcal{O}\left(poly(n, \frac{1}{\epsilon})\right)$ unless P = NP (Vazirani, 2001). In comparison, Hall (1998) provides for the m-machine flowshop problem an FPT-AS (fixed-parameter tractable approximation scheme), namely an ϵ-approximation algorithm that runs in time $\mathcal{O}(f(\epsilon, \kappa) \cdot poly(n))$ for κ a fixed parameter of the instance and f a computable function. Hall (1998) choose κ to be the number of machines of the flowshop, leading to an FPT-AS that runs in time $\mathcal{O}\left(n^{3.5} \cdot (\frac{m}{\epsilon})^{\frac{m^4}{\epsilon^2}}\right)$. In our case, we consider the case $m = 3$.

Algorithm 6: Hybrid approximation scheme for the 3-machine flowshop

Input: $\epsilon > 0$, 3-machine flowshop on n jobs with processing times $\{p_{ij} : i \in [3], j \in [n]\}$

Output: solution at most $1 + \epsilon$ times the optimal solution

1 $P = \max_{i \in [3], j \in [n]} \{p_{ij}\}$;

2 $K = \frac{\epsilon P}{n+2}$;

3 **for** $i \in [3], j \in [n]$ **do**

4 $\quad p'_{ij} = \lceil \frac{p_{ij}}{K} \rceil$;

5 Solve the 3-machine flowshop on n jobs with new processing times $\{p'_{ij} : i \in [3], j \in [n]\}$ with Algorithm 5 that outputs permutation π';

6 Return π'

Lemma 5.11. *Let π^* be an optimal solution of the 3-machine flowshop problem, for the processing times $\{p_{ij} : i \in [3], j \in [n]\}$. Let π' be the output of Algorithm 6. We have*

$$C_{max}(\pi') \leq (1 + \epsilon) \cdot C_{max}(\pi^*) \,.$$

Next, we introduce two observations necessary to prove Lemma 5.11. The proofs are omitted because of their simplicity.

Observation 5.12. *Let π be a permutation and let be $\alpha \in \mathbb{R}^*_+$. We note $C_{max}(\pi)$ the makespan of π of the 3-machine flowshop for processing times $\{p_{ij} : i \in [3], j \in [n]\}$. We note $C'_{max}(\pi)$ the makespan of π of the 3-machine flowshop for processing times $\{p'_{ij} : i \in [3], j \in [n]\}$ such that $p'_{ij} := \alpha p_{ij}$ for all i, j. Then, $C'_{max}(\pi) = \alpha C_{max}(\pi)$. Notice that for $p'_{ij} \leq \alpha p_{ij}$, we have $C'_{max}(\pi) \leq \alpha C_{max}(\pi)$ even if the critical path in π may differ to obtain C_{max} and C'_{max}.*

Observation 5.13. *Let π be a permutation and let $\beta \in \mathbb{R}$ such that $\beta \geq -\min_{i \in [3], j \in [n]} \{p_{ij}\}$. We note $C_{max}(\pi)$ the makespan of π of the 3-machine flowshop for processing times $\{p_{ij} : i \in [3], j \in [n]\}$. We note $C''_{max}(\pi)$ the makespan of π of the 3-machine flowshop for processing times $\{p''_{ij} : i \in [3], j \in [n]\}$ such that $p''_{ij} := p_{ij} + \beta$ for all $i \in [3], j \in [n]$. Then, $C''_{max}(\pi) \leq C_{max}(\pi) + \beta(n + 2)$. Notice that for $p''_{ij} \leq p_{ij} + \beta$, we still have $C''_{max}(\pi) \leq C_{max}(\pi) + \beta(n + 2)$ even if the critical path in π may differ to obtain C_{max} and C''_{max}.*

Proof of Lemma 5.11. Let be $\epsilon > 0$. The new processing times considered $p'_{ij} := \lceil \frac{p_{ij}}{K} \rceil$ imply that $\frac{p_{ij}}{K} \leq p'_{ij} < \frac{p_{ij}}{K} + 1$. We note $C'_{\max}$ the makespan of the new problem, i.e. the 3-machine flowshop problem with processing times $\{p'_{ij} : i \in [3], j \in [n]\}$.

On the one hand, we have $p'_{ij} < \frac{p_{ij}}{K} + 1$, for all $i \in [3], j \in [n]$. Thus, according to Observations 5.12 and 5.13 considering the optimal permutation π^*, $C'_{\max}(\pi^*) \leq \frac{C_{\max}(\pi^*)}{K} + n + 2$,

namely, because $K > 0$,

$$KC'_{\max}(\pi^*) \leq C_{\max}(\pi^*) + K(n+2)\,. \tag{10}$$

On the other hand, we have $\frac{p_{ij}}{K} \leq p'_{ij}$. Thus, according to Observation 5.12 considering the output permutation π' of Algorithm 6, $\frac{C_{\max}(\pi')}{K} \leq C'_{\max}(\pi')$, namely, because $K > 0$,

$$C_{\max}(\pi') \leq KC'_{\max}(\pi') \leq KC'_{\max}(\pi^*) \tag{11}$$

$$\leq C_{\max}(\pi^*) + K(n+2) = C_{\max}(\pi^*) + \epsilon P \tag{12}$$

$$\leq C_{\max}(\pi^*) + \epsilon C_{\max}(\pi^*) = (1+\epsilon)C_{\max}(\pi^*)\,, \tag{13}$$

where (11) comes from the fact that π' is the optimal solution for makespan $C'_{\max}$, (12) results from Equation (10), and (13) is true because the makespan is always larger than $P = \max_{i\in[3],j\in[n]}\{p_{ij}\}$. □

Theorem 5.14. *Algorithm 6 is an approximation scheme for the 3-machine flowshop problem and outputs a solution whose makespan it at most* $(1+\epsilon)$ *times the optimal value in time* $\mathcal{O}^*\left(\frac{1}{\epsilon^3}\cdot 1.728^n\right)$.

Proof. First, according to Lemma 5.11, Algorithm 6 outputs a solution whose makespan it at most $(1+\epsilon)$ times the optimal value. Second, Algorithm 5 solves the new problem in time $\mathcal{O}^*((\sum p'_{ij})^4 \cdot 1.728^n) = \mathcal{O}^*(\frac{1}{\epsilon^4}\cdot 1.728^n)$. Indeed,

$$\sum p'_{ij} \leq \sum\left(\frac{p_{ij}}{K}+1\right) = \frac{1}{K}\sum p_{ij} + 3n \leq \frac{1}{K}\cdot 3nP + 3n = \frac{3n(n+2)}{\epsilon} + 3n\,.$$

Thus, $\sum p'_{ij} \leq poly(n, \frac{1}{\epsilon})$. □

6 Conclusion

In this work, we propose generalized dynamic programming recurrences for NP-hard scheduling problems to solve a broad class of such problems with our hybrid algorithm Q-DDPAS which is an adapted version of the quantum-classical algorithm of Ambainis et al. (2019). Q-DDPAS provides a quantum speed-up to their exact resolution. Specifically, our hybrid algorithm reduces the best-known classical time complexity, often equal to $\mathcal{O}^*(2^n)$ for single-machine problems and $\mathcal{O}^*(3^n)$ for the 3-machine flowshop, to $\mathcal{O}^*(1.728^n)$, sometimes at the cost of an additional pseudo-polynomial factor as summarized in Table 1. Future work should be dedicated to finding a quantum brick, e.g. (Grover, 1996), Quantum Fourier Transform (Kitaev, 1995) or Quantum Walks (Aharonov et al., 2001), that could speedup exponential algorithms such as Sort-and-Search (Lenté et al., 2013), Inclusion-Exclusion (Ploton, 2023) or Branch-and-Reduce (T'kindt et al., 2022).

Acknowledgements. This work has been partially financed by the ANRT through the PhD number 2021/0281 with CIFRE funds.

Bibliography

Aharonov, D., Ambainis, A., Kempe, J., and Vazirani, U. (2001). Quantum walks on graphs. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 50–59.

Ambainis, A., Balodis, K., Iraids, J., Kokainis, M., Prūsis, K., and Vihrovs, J. (2019). Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1783–1793. SIAM.

Bernstein, E. and Vazirani, U. (1993). Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20.

Boyer, M., Brassard, G., Høyer, P., and Tapp, A. (1998). Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505.

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L., et al. (2021). Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644.

Cygan, M., Pilipczuk, M., Pilipczuk, M., and Wojtaszczyk, J. O. (2014). Scheduling partially ordered jobs faster than 2^n. *Algorithmica*, 68:692–714.

Durr, C. and Hoyer, P. (1996). A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*.

Farhi, E., Goldstone, J., and Gutmann, S. (2014). A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*.

Giovannetti, V., Lloyd, S., and Maccone, L. (2008). Quantum random access memory. *Physical review letters*, 100(16):160501.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier.

Grange, C., Bourreau, E., Poss, M., and t'Kindt, V. (2023a). Quantum speed-ups for single-machine scheduling problems. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 2224–2231.

Grange, C., Poss, M., and Bourreau, E. (2023b). An introduction to variational quantum algorithms for combinatorial optimization problems. *4OR*, pages 1–41.

Grange, C., Poss, M., Bourreau, E., t'Kindt, V., and Ploton, O. (2024). Companion Paper: Moderate Exponential-time Quantum Dynamic Programming Across the Subsets for Scheduling Problems. HAL preprint, https://hal.science/hal-04296238.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219.

Hall, L. A. (1998). Approximability of flow shop scheduling. *Mathematical Programming*, 82(1-2):175–190.

Held, M. and Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162.

Kitaev, A. Y. (1995). Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*.

Kurowski, K., Pecyna, T., Slysz, M., Rozycki, R., Waligora, G., and Weglarz, J. (2023). Application of quantum approximate optimization algorithm to job shop scheduling problem. *European Journal of Operational Research*, 310(2):518–528.

Lawler, E. L. (1990). A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133.

Lenté, C., Liedloff, M., Soukhal, A., and T'kindt, V. (2013). On an extension of the sort & search method with application to scheduling theory. *Theoretical Computer Science*, 511:13–22.

Miyamoto, M., Iwamura, M., Kise, K., and Gall, F. L. (2020). Quantum speedup for the minimum steiner tree problem. In *COCOON 2020, Atlanta, GA, USA, August 29–31, 2020, Proceedings*, pages 234–245. Springer.

Nannicini, G. (2019). Performance of hybrid quantum-classical variational heuristics for combinatorial optimization. *Physical Review E*, 99(1):013304.

Nannicini, G. (2022). Fast quantum subroutines for the simplex method. *Operations Research*.

Pinedo, M. L. (2012). *Scheduling*, volume 29. Springer.

Ploton, O. (2023). Contributions of inclusion-exclusion to exact or approximate solution of scheduling problems.

Ploton, O. and T'kindt, V. (2022). Exponential-time algorithms for parallel machine scheduling problems. *Journal of Combinatorial Optimization*, 44(5):3405–3418.

Ploton, O. and T'kindt, V. (2023). Moderate worst-case complexity bounds for the permutation flowshop scheduling problem using inclusion–exclusion. *Journal of Scheduling*, 26(2):137–145.

Ruan, Y., Marsh, S., Xue, X., Liu, Z., Wang, J., et al. (2020). The quantum approximate algorithm for solving traveling salesman problem. *CMC*, 63(3):1237–1247.

Shang, L., Lenté, C., Liedloff, M., and T'Kindt, V. (2018). Exact exponential algorithms for 3-machine flowshop scheduling problems. *Journal of Scheduling*, 21:227–233.

Shimizu, K. and Mori, R. (2022). Exponential-time quantum algorithms for graph coloring problems. *Algorithmica*, pages 1–19.

Sutter, D., Nannicini, G., Sutter, T., and Woerner, S. (2020). Quantum speedups for convex dynamic programming. *arXiv preprint arXiv:2011.11654*.

Tabi, Z., El-Safty, K. H., Kallus, Z., Hága, P., Kozsik, T., Glos, A., and Zimborás, Z. (2020). Quantum optimization for the graph coloring problem with space-efficient embedding. In *2020 IEEE (QCE)*, pages 56–62. IEEE.

T'kindt, V., Della Croce, F., and Liedloff, M. (2022). Moderate exponential-time algorithms for scheduling problems. *4OR*, pages 1–34.

Vazirani, V. V. (2001). *Approximation algorithms*, volume 1. Springer.

Woeginger, G. J. (2003). Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization—Eureka, You Shrink! Papers*, pages 185–207. Springer.

A Useful upper bounds

We define the binary entropy of $\epsilon \in]0,1[$ by $H(\epsilon) = -(\epsilon \log_2(\epsilon) + (1-\epsilon)\log_2(1-\epsilon))$. We remind some useful upper bounds of binomial coefficients Ambainis et al. (2019):

$$\binom{n}{k} \leq 2^{H(\frac{k}{n})\cdot n}, \ \forall k \in [\![1,n]\!] \quad \text{and} \quad \sum_{i=1}^{k}\binom{n}{i} \leq 2^{H(\frac{k}{n})\cdot n}, \forall k \in \left[\!\left[1,\frac{n}{2}\right]\!\right].$$

Thus, it leads to the following upper bounds to compute the complexities of interest:

$$\sum_{i=k}^{n/4}\binom{n}{k} \leq 2^{0.811n}, \sum_{k=1}^{0.945\cdot n/4}\binom{n}{k} \leq 2^{0.789n}, \sqrt{\binom{n}{n/2}\binom{n/2}{n/4}} \leq 2^{0.75n}, \sqrt{\binom{n}{n/2}\binom{n/2}{n/4}\binom{n/4}{0.945\cdot n/4}} \leq 2^{0.789n}$$