FGCS

Future Generation Computer Systems 00 (2024) 1–10

Quantum annealing solution for the unrelated parallel machine scheduling with priorities and delay of task switching on machines

F. Orts[a], A.M. Puertas[b], G. Ortega[c], E.M. Garzón[c]

[a]*Institute of Data Science and Digital Technologies, University of Vilnius, Lithuania*
[b]*Department of Chemistry and Physics, University of Almería, Spain*
[c]*Informatics Department, University of Almería, ceiA3, Spain*

Abstract

Quantum computing has emerged in recent years as an alternative to classical computing, which could improve the latter in solving some types of problems. One of the quantum programming models, Adiabatic Quantum Computing, has been successfully used to solve problems such as graph partitioning, traffic routing, and task scheduling. In this paper, the focus is on the scheduling of the problem of unrelated parallel machines, where the processing time of tasks on any of the available processing elements is known. Moreover, the proposed model is extended in two relevant aspects for this kind of problem: the existence of some degree of priority of tasks, and the introduction of a delay or penalty every time a processing unit or machine changes the type of task that executes. In all cases, the problem is expressed as Quadratic Unconstrained Binary Optimisation, which can be subsequently solved using quantum annealers. The quantum nonlinear programming framework discussed in this work consists of three steps: quadratic approximation of cost function, a binary representation of parameter space, and solving the resulting Quadratic Unconstrained Binary Optimisation on the quantum annealer platform D-Wave. One of the novelties in tackling this problem is the compaction of the model bearing in mind the repetitions of each task, to allow solving larger scheduling problems with the quantum resources available in the experimentation platform. An estimation of the number of qubits required in relation to the scheduling parameters is analysed. The models have been implemented on the D-Wave platform and validated with respect to other traditional methods. Furthermore, the proposed extensions to consider priorities and to switch the delay of tasks have been analysed using a case study.

Keywords: Quantum Computing, Adiabatic Quantum Computing, Quantum annealing, Quadratic Unconstrained Binary Optimisation, Combinatorial Optimization, Scheduling on unrelated parallel machines problem,

1. Introduction

Quantum computing takes advantage of the Quantum Mechanics effects to process information. Quantum hardware implements such principles to solve general computational problems. There are two ways of performing computational operations on a quantum computer. The most well-known is the approach based on a quantum circuit model of computation. This approach provides both a framework for formulating quantum algorithms and an architecture for the physical construction of quantum computers. This model of computation might provide a complete design of quantum computing in the long run, but nowadays is severely limited by the small number of qubits that make up real quantum platforms and by the errors introduced by each quantum gate.

The other alternative to quantum processing is Adiabatic Quantum Computing (AQC), and currently, it can provide enough resources to solve particular applications of practical interest. It relies on the adiabatic theorem and is focused on the solution of combinatorial optimization problems. Although the goal of AQC is particular, it is of great interest since many of these problems are NP-complete and they are a challenge for conventional computation when the input problem grows. Moreover, these problems are involved in a wide set of applications as illustrated in [1]. AQC can solve such problems efficiently because their solution can be expressed as the ground state of an Ising Hamiltonian, which evolves in polynomial time [2]. The Ising Hamiltonian is used to model quadratic unconstrained binary optimization (QUBO). So, the QUBO problem is formulated to find the minimum of a quadratic polynomial with unitary variables. Strictly, the physical realization of AQC is unreachable since the non-ideals conditions avoid the adiabatic evolution of the quantum hardware. Thus, Quantum Annealing (QA) is based on the AQC principles but in a flexible sense [3].

Currently, the Ising solvers are realized with quantum annealer, such as the D-Wave platform [4]. Thus, the translation of combinatorial optimization problems in QUBO models is the key to their QA solution.

Scheduling is one of the active areas of discrete optimization that plays a crucial role in the manufacturing and service industries. Scheduling theory has been a focus of interest by researchers in management science, industrial engineering, and operations research. Many classical approaches to solve the different types of scheduling problems can be found in the literature, just to name a few [5, 6, 7]. In recent years, quantum computing has been postulated as an alternative and several authors have studied the behavior of these problems on quantum platforms. In [8] authors formulate the uncapacitated task allocation problem as a QUBO model. The work by Carugno et al. [9] studies the application of quantum annealing to solve the job-shop scheduling problem and compares the solution quality with various classical solvers. In the present work, we focused our attention on the unrelated parallel machine scheduling problem (UPM).

The aim of this work is to show how to solve scheduling problems with additional constraints on current quantum annealers as an illustration of the potential of QA to solve large combinatorial optimization problems of practical interest. The contribution is twofold, on one hand, the definition of the specific QUBO model on a practical context and, on the other hand, the implementation/evaluation of the real and accessible annealer using the Ocean library and platforms of D-Wave. The UPM problem consists on finding the optimal scheduling of a set of tasks among the available (heterogeneous) processing elements if the processing time of any procedure executed on any of the available processing elements is a priori known. The goal is to minimize the maximum completion time of tasks without additional constraints. In our previous work [10], this problem was expressed with a QUBO formulation, which allowed the use of the D-Wave quantum annealer to solve it. This transformation was done by the quadratic approximation of the cost function, binary encoding of the integer variables, and solving the problem using a quantum annealer.

In this work, the objective is to solve the UPM with two additional constraints on the real annealers. Such constraints of the problem focus on: the existence of some degree of priority in the type of task, and the incorporation of a delay/penalty every time a processing unit/machine changes the type of task that is executed. Both extensions are of great practical interest in the application fields of these scheduling problems. This way, the main contributions of this work are the following:

1. The UPM scheduling problem with the two mentioned constraints in terms of task priorities and the switching delays have been defined and expressed with a QUBO model.
2. The reduction of the number of required qubits on quantum annealers, if repetitions of tasks can be considered in the UPM scheduling, i.e. the tasks can be grouped by similar or same runtimes on the same processing element. The number of qubits for the UPM scheduling problem with and without task repetition has been analyzed to prove the large resource savings that can be obtained by exploiting repetitions in the model expression. This approach allows us to solve very large scheduling problems with the resources supplied by the current D-Wave annealers.
3. The UPM scheduling problem with both constraints has been developed with the programming tools supplied with the D-Wave quantum annealer, the Ocean library. To show the correct impact of the constraints in the solution computed by the proposed model on the annealer, the results computed for the UPM scheduling with constraints have been analysed in comparison with the obtained ones without constraints for a representative example of the problem.

These contributions illustrate how to express the constraints in the methodology to solve combinatorial optimization problems on innovative quantum annealers and their potential for solving problems of large dimensions.

The paper has been organized as follows. Section 2 is devoted to describing the quantum annealing computational model. Section 3 introduces the scheduling problems addressed in this work with their QUBO formulations. First, the UPM scheduling and next the constraints focused on the priority of tasks, Subsection 3.3, and the delays due to the switches between kinds of tasks on machines, Subsection 3.4. Section 4 shows how the QUBO formulations for the UPM scheduling problem and the extensions are solved on the D-wave annealer. In Section 5 the number of qubits required by the proposed models is quantified in relation to the parameters of scheduling. On the D-Wave platform, the proposed models are validated and the extensions have been analysed by a case study. Finally, in Section 6 the main conclusions are drawn.

2. Quantum annealing computing

The adiabatic theorem assures that if we start at a state of minimum energy of a simple Hamiltonian and it evolves slowly, it will always remain in the state of lowest energy, the ground state. So, the idea of quantum adiabatic computing is to select a ground state of a simple Hamiltonian, H_0, and make the system evolve during a time T to the state of minimum energy of the Hamiltonian of the problem H_p. So, we can define a Hamiltonian as a function of time to model its temporal evolution:

$$H(t) = (1 - \frac{t}{T})H_0 + \frac{t}{T}H_p \quad (1)$$

In practice, it is difficult to guarantee the adiabatic conditions, and quantum annealing is used as a heuristic approach which combines the adiabatic theorem and the Ising model [11] to build solvers of combinatorial optimization problems. It consists of:

1. $H_0 = -\sum_{i=1}^{n} \sigma_i^x$ is defined as initial Hamiltonian where n is the number of qubits and σ_i^x is the Pauli x operator on the ith qubit.

2. H_p is defined as the target Ising Hamiltonian,
3. The system evolves from H_0 to H_p without adiabatic conditions being fully guaranteed,
4. The final state is measured to compute a possible minimum,
5. The process is repeated several times to compute various approximations of minima.

We recall that the Ising Hamiltonian defines a model of ferromagnetism in statistical mechanics with unitary discrete variables since they represent magnetic spins that can be in one of two states. Therefore, QA is useful to solve combinatorial optimization problems with unitary variables. Moreover, the Ising model can be translated to the QUBO model, which unifies a rich variety of combinatorial optimization problems [12].

Currently, D-Wave Systems Inc. has developed quantum hardware based on QA with a large number of qubits. This technology still suffers from limitations such as resource scarcity and control errors, among others. However, a wide set of practical optimization applications are being currently adapted to this technology since potentially it offers a huge computational power for solving large combinatorial optimization problems which are NP-complete.

The next sections are focused on the application of the QA methodology for solving UPM scheduling. Therefore, we develop the steps related to the previous methodology: (1) quadratic approximation of cost function of a compact model, (2) binary representation of the discrete variables, and (3) solving and testing the resulting QUBO model on the D-wave annealer.

3. Unrelated parallel machine scheduling problem using quantum annealing

3.1. Definition of the unrelated parallel machine scheduling problem

The problem of distributing N tasks of J different types in M processing units has been already described in the literature [13]; thus here we only give a short account. Briefly, the optimal distribution that minimizes the total time to execute all tasks is sought, namely:

$$\begin{array}{rcl} \text{Find:} & n_{j,\alpha} & \\ \text{to minimize} & max\{T_\alpha\} & \\ \text{with} & T_\alpha = \sum_j n_{j,\alpha} t_{j,\alpha} & \alpha = 1,\ldots,M \\ \text{subject to} & \sum_\alpha n_{j,\alpha} = x_j & j = 1,\ldots,J \end{array} \tag{2}$$

Here $\{n_{j,\alpha}\}$ is the number of tasks of type j assigned to the Processing Unit (PU) named α, T_α is the time required by PU α to complete all its tasks and $\{t_{j,\alpha}\}$ is the runtime matrix of a task of type j in every PU α. The J restrictions (last line) indicate that all jobs of type j, x_j, must be assigned. Hereinafter, they are referred to as the main constraint of UPM scheduling.

The inputs to the problem are the number of PUs, M, the number of tasks of every type that must be assigned, $\{x_j\}$, and the runtime matrix, $\{t_{j,\alpha}\}$. Different strategies have been proposed to solve this problem, and commercial software is available, such as AMPL [14] and CPLEX [15].

3.2. From Binary Integer Programming (BIP) to QUBO

In order to solve this problem with quantum annealing, it has been reformulated as a quadratic unconstrained binary optimization (QUBO) problem following [12]. For this purpose, let us define the function:

$$O_0 = \sum_\alpha T_\alpha^2 \tag{3}$$

Because the ratio $T_\alpha / max\{T_\alpha\}$ is smaller than one for all PUs, except for the PU with the largest runtime, $max\{T_\alpha\}$, the summation in O_0 is dominated by this largest term, i.e. by $max\{T_\alpha\}$, and therefore, minimizing $max\{T_\alpha\}$ or make-span, as required by the scheduling problem, is equivalent to minimizing O_0. Note that a higher exponent would make this equivalence clearer, but only with powers 1 and 2 can it be formulated as a QUBO problem.

The first, brute force, approach to the problem is to consider that all tasks are different: $n_{j,\alpha}$ becomes then a binary variable, $n_{j,\alpha} = 1$ if task j is assigned to PU α and 0 otherwise, and the restriction now reads $\sum_\alpha n_{j,\alpha} = 1$ for all j, ensuring that all tasks are run once. This restriction must be incorporated in the function to be minimized:

$$\begin{aligned} O &= \sum_\alpha T_\alpha^2 + \sum_j P_j \left(1 - \sum_\alpha n_{j,\alpha}\right)^2 = \\ &= \sum_\alpha \left[\sum_j n_{j,\alpha}^2 t_{j,\alpha}^2 + 2 \sum_{k,l;k\neq l} n_{k,\alpha} n_{l,\alpha} t_{k,\alpha} t_{l,\alpha}\right] + \\ &+ \sum_j P_j \left(1 - \sum_\alpha n_{j,\alpha}\right)^2 \end{aligned} \tag{4}$$

where P_j are "large" constants [12], and the expression of T_α has been substituted in the second line to get an explicit expression [2]; if P_j is smaller, or comparable to T_α^2, the constraint can be violated to minimize O, but if P_j is too large, the first term in O becomes irrelevant. This formulation corresponds to a QUBO problem, as expected, and can be implemented in a quantum annealer.

This approach, however, requires as many unitary variables as elements in the matrix $n_{j,\alpha}$, i.e. tasks to be assigned times the number of PUs, what restricts importantly the size of the problem that can be studied. To overcome this issue, we make use of the repetition of jobs for the same task. In this case, $n_{j,\alpha}$ is no longer a unitary variable, and therefore it is subject to the last condition in the problem definition, $\sum_\alpha n_{j,\alpha} = x_j$. To continue within the QUBO formulation it can be, nevertheless, expressed using unitary variables for the digits in its binary representation [2]:

$$n_{j,\alpha} = \sum_{k=0}^{B} n_{j,\alpha,k} 2^k \tag{5}$$

where $B = int\left[log_2(R+1)+1\right]$, with $R = max\left\{x_j\right\}$; variables $n_{j,\alpha,k}$ are unitary. Introducing this representation of $n_{j,\alpha}$ in the expression of O_0 and the restriction yields finally for O:

$$\begin{aligned} O &= \sum_\alpha T_\alpha^2 + \sum_j P_j\left(x_j - \sum_\alpha n_{j,\alpha}\right)^2 = \\ &= \sum_\alpha \left[\sum_j t_{j,\alpha} \sum_{k=0}^{B} n_{j,\alpha,k} 2^k\right]^2 + \\ &+ \sum_j P_j\left(x_j - \sum_\alpha \sum_{k=0}^{B} n_{j,\alpha,k} 2^k\right)^2 \end{aligned} \tag{6}$$

It can be easily confirmed that this expression corresponds to a QUBO problem in the variables $n_{j,\alpha,k}$, and allows finding the distribution of $J \times R$ tasks in M processing units using $J \times B \times M$ unitary variables. Since $B \sim \log_2 R$, this implies an important reduction in computing resources with respect to the initial formulation, given by Eq. 4.

In the next subsections, this problem is extended by introducing two aspects typically found in scheduling problems, namely, the existence of some degree of priority in the type of tasks, and introducing a delay or penalty every time the type of task needs to be changed in a PU. Both problems are tackled by modifying the objective function O.

3.3. Priority tasks

Consider that in the scheduling problem defined by (3), one type of tasks must be finished as soon as possible, although other tasks do not depend on the outcomes of these priority ones. In a first approach, this can be considered as two different scheduling problems to be run sequentially: a first one for the priority tasks and the second problem for the non-priority ones. However, since the outcome of the priority tasks is not necessary for the non-priority ones, this is inefficient, as some PU are idle while all priority tasks are finished, waiting for the second problem to start. This is particularly important when the number of priority tasks is low, or even smaller than M, the number of PUs.

The definition of this new problem requires a slight modification with respect to the previous one. Let $J_p < J$ be the number of types of priority of tasks, while J is the total number of tasks, including priority and non-priority ones. The problem of the optimal distribution of tasks therefore is:

$$\begin{aligned} \text{Find:} \quad & n_{j,\alpha} \\ \text{to minimize} \quad & max\left\{T_{p,\alpha}\right\} \text{ and } max\left\{T_\alpha\right\} \\ \text{with} \quad & T_{p,\alpha} = \textstyle\sum_{j_p=1}^{J_p} n_{j_p,\alpha} t_{j_p,\alpha} \qquad \alpha = 1,\ldots,M \\ \text{and} \quad & T_\alpha = \textstyle\sum_{j=1}^{J} n_{j,\alpha} t_{j,\alpha} \qquad \alpha = 1,\ldots,M \\ \text{subject to} \quad & \textstyle\sum_\alpha n_{j,\alpha} = x_j \qquad j = 1,\ldots,J \end{aligned} \tag{7}$$

where $T_{p,\alpha}$ is the time spent by PU α to perform only the priority tasks, whose maximum needs to be minimized. Note that for clarity, we have used sub-index p for the summation of $T_{p,\alpha}$, which implies only the priority tasks, i.e. j_p runs from 1 to J_P. The restriction in the last line applies for both priority and non-priority tasks, and therefore it is not modified by the new considerations.

Following the same strategy as in the previous section, we assume that the maximum $T_{p\alpha}$ dominates the summation $O_p = \sum T_{p,\alpha}^2$. Thus, to minimize both O_p and O_o, as required by the problem definition above, with the restrictions, the following function is defined:

$$\begin{aligned} O' &= A\sum_\alpha T_{p,\alpha}^2 + \sum_\alpha T_\alpha^2 + \sum_j P_j\left(x_j - \sum_\alpha n_{j,\alpha}\right)^2 = \\ &= A\sum_\alpha \left(\sum_{j_p} n_{j_p,\alpha} t_{j_p,\alpha}\right)^2 + \sum_\alpha \left(\sum_j n_{j,\alpha} t_{j,\alpha}\right)^2 + \\ &+ \sum_j P_j\left(x_j - \sum_\alpha n_{j,\alpha}\right)^2 \end{aligned} \tag{8}$$

where the first summation corresponds to priority tasks and the second one to all tasks. The constant A is introduced as a measure of the priority of the tasks and should be compared to the ratio of the maximum execution times of all tasks to all priority tasks. If A is much smaller than this ratio, tasks are not prioritised, and the algorithm searches for a distribution that minimizes the total execution time. In the opposite case, if A is much larger than the ratio of maximum runtimes, the solution to the problem guarantees that all priority tasks are finished as soon as possible, even if this implies a larger total execution time. In any case, to guarantee that all jobs of type j are assigned, $A \max\left\{T_\alpha^2\right\}$ must be smaller than P_j.

In equation (8), $n_{j,\alpha}$ must be substituted by its binary representation as given by Eqn. (5) to make a QUBO problem suitable for implementation.

Finally, let us mention that although only two levels of prioritisation are considered in the definition of the problem and in the objective function O', the problem can be readily extended to incorporate many more levels. For this purpose, other terms should be added to O' with different prefactors – the tasks considered in the summation with higher constants would have higher priority.

3.4. Delay due to task switching on processing units

The second modification to the original scheduling problem is the consideration of a delay time whenever the type of task performed on a PU is changed. This is motivated by the existence of a switching time due to the preparation of the infrastructure for the new task. This can have several forms, such as physical modifications in production chains or assembly lines, or the upload of configuration files and memory allocation in the computation of complex calculations, to give just two examples.

Ideally, the total delay in the execution of PU α can be calculated exactly if the number of types of tasks is known. However, this requires calculating how many $n_{j,\alpha}$ are greater than

zero, but this cannot be formulated as a QUBO problem. Alternatively, since the number of switches is minimized when the number of tasks of the same type, $n_{j,\alpha}$ is maximized for every PU, we define the problem of seeking the optimal distribution of tasks with delay due to switching as follows:

$$\begin{array}{rcl} \text{Find:} & n_{j,\alpha} & \\ \text{to minimize} & max\{T_\alpha\} & \\ \text{and maximmize} & n_{j,\alpha} & \forall\alpha, j \\ \text{with} & T_\alpha = \sum_j n_{j,\alpha} t_{j,\alpha} & \alpha = 1,\ldots,M \\ \text{subject to} & \sum_\alpha n_{j,\alpha} = x_j & j = 1,\ldots,J \end{array} \tag{9}$$

The new condition of maximizing $n_{j,\alpha}$ for all j and α is cast into a QUBO problem following the same strategy as in the original problem, i.e. maximizing the summation of $n_{j,\alpha}^2$. Thus, the following function is defined:

$$\begin{aligned} O'' &= \sum_\alpha T_\alpha^2 - \Delta^2 \sum_\alpha \sum_j n_{j,\alpha}^2 + \\ &+ \sum_j P_j \left(x_j - \sum_\alpha n_{j,\alpha} \right)^2 = \\ &= \sum_\alpha \left(\sum_j n_{j,\alpha} t_{j,\alpha} \right)^2 - \Delta^2 \sum_{j,\alpha} n_{j,\alpha}^2 + \\ &+ \sum_j P_j \left(x_j - \sum_\alpha n_{j,\alpha} \right)^2 \end{aligned} \tag{10}$$

where Δ stands for the delay time in every switching (which has been taken identically for all cases). Note that the second term in the r.h.s. is negative, because it should be maximized and not minimized. For every value of α, the summation in j is dominated by the maximum $n_{j,\alpha}$, i.e. by the higher number of tasks of the same type, and the summation in α is dominated by the PU that repeats more tasks. Therefore, by minimizing this term, PUs are forced to perform as many tasks as possible of the same type, reducing the number of changes.

It is also interesting to note that the prefactor of this term is given by the delay, Δ. Thus, if $\Delta \ll t_{j,\alpha}$ for all j and α, this term is negligible compared with the first one, and task switching does not contribute to the final optimal distribution. On the other hand, if $\Delta \gg t_{j,\alpha}$, this contribution is dominant, and the optimal distribution will sacrifice the minimisation of the global runtime of tasks to avoid task switching. Different from P_j or A, in previous equations, Δ is not a parameter that has to be set to guarantee the performance of the QUBO approximation to the problem.

Finally, let us note that $n_{j,\alpha}$ should be coded in its binary representation to convert the minimisation of O'' in a QUBO problem. Also, it is straightforward to include a dependency of the delay times on the type of tasks, Δ_j, introducing it inside the summation.

4. D-Wave implementation

The described problem has been formulated for execution on current quantum annealers through D-Wave Leap [16, 17, 18]. The code has been written in Python. Since the entire problem and its different variants has been formulated in QUBO format, a direct implementation using BQM is trivial to perform. However, since we want to study how resources are consumed as a function of the size and other parameters of each problem instance, for the sake of clarity we have made an implementation using CQM to measure more conveniently as a function of the inputs.

The natural inputs to the problem are the number of PUs to be handled (**M**), the number of different types of tasks to be executed (**J**), the number of repetitions of each type of task (**R**), the priority of each type of task ($p_1, p_2, ..., p_J$) and the time needed to switch task types on a machine, which we will simply call delay (Δ). It is also necessary to specify the time that each type of task takes to execute on each machine, information that can be expressed as a matrix, where each row is related to a type of task, and it is composed of the time that the PU needs to execute a task of that type ($\mathbf{t}_{j,\alpha}$). The program accepts this information through a text file. Figure 1 shows an example of what a program input file should look like. The time units used are not specified, but it is mandatory that all times indicated use the same unit. Based on Section 3, it is assumed that there are no dependencies between tasks and that each PU can only execute a single task at a time.

```
#Num of different jobs: 3
#Num of repetitions: 5
#Num of machines: 2
#Delay: 0

  job id    machine 0    machine 1     priority
--------    ---------    ---------     ---------
       0            1            2            1
       1            2            3            1
       2            3            1            1
```

Figure 1: Example of an input file for a problem with 2 PUs, 3 different type of tasks, and 5 repetitions. In this example it is stated that there is no delay caused by switching between task types, and that all tasks have the same priority. Since tasks of the same type share execution times, the time is only shown once for each type of task.

The program also accepts as input arguments, this time via console, the name of the mentioned text file, and the time limit imposed to finish the executions (make-span). Following the nomenclature used in the previous section, the following parameters have been considered to represent the problem in the code:

- **J**: is the number of different types of tasks.
- **j**: is a specific type of tasks (1, 2, ..., J).
- **R**: is the repetitions of the all types of tasks, $R = x_j \; \forall j$.
- **p**: is the priority of each type of task ($p_1, p_2, ..., p_J$).

- **A**: is a constant to measure the task priority.
- **M**: is the number of PUs.
- α: is a specific PU $(1, 2, ..., M)$.
- $\mathbf{t}_{\mathbf{j},\alpha}$: is the processing duration that PU α needs for one task of type j.
- Δ: is the delay when switching task types on PUs.
- **V**: estimation of maximum possible completion time (make-span).

For simplicity, it is assumed that all tasks have the same number of repetitions R, and that the delay Δ is always the same for all tasks and PUs. However, it would be straightforward to modify the code to be able to indicate the number of repetitions or the delay for each type of task individually (assuming, for the delay, that its duration is associated with the type of task). These variables have also been used to work with the outputs of the program:

- **O**: is a positive integer variable that defines the completion time (make-span).
- **B**: is the number of necessary binary digits to represent the number of repetitions (Eq. 5).
- $\left\{\mathbf{n}_{\mathbf{j},\alpha,\mathbf{k}}\right\}$: is the matrix of the distribution of tasks, where $n_{j,\alpha,k}$ represents the k-th digit in the binary representation of $n_{j,\alpha}$, which stands for the number of tasks of type j that are assigned to PU α.

According to Eq. 4 and Eq. 6, the goal is still to minimise the time in which the problem is solved, O. Although we have used two variables for the make-span, O and V, the difference between the two is that O will contain the best result proposed by the quantum annealers, while V is an input parameter indicating the desired maximum limit for the make-span. Of course, it is possible that the problem cannot be solved in the proposed time. If O is greater than V, the software will display a message warning that it has not been possible to solve the problem in the requested time.

According to the model described in the previous section, a constraint needs to be established to ensure that each task is executed only R times:

$$\sum_{\alpha} n_{j,\alpha} = R \tag{11}$$

where R will be 1 if we use the model set out in Eq. 4, or any other natural number if we work with the model in Eq. 6.

The solution given by the quantum annealer is not limited to the time O, but also includes the complete scheme by which this time has been achieved. This scheme is returned via the array $n = \left\{\mathbf{n}_{\mathbf{j},\alpha,\mathbf{k}}\right\}$, described above. Although the array n contains all the necessary information to properly distribute the tasks in the machines, its reading is neither comfortable nor easy to interpret for the human eye (with the aggravating factor of logarithmic notation). For this reason, the output of the software

```
#Number of jobs: 15
#Number of machines: 2
#Completion time: 11.0
```

	machine 0			machine 1		
job id	type	start	dur	type	start	dur
0	0	0	1	0	0	0
1	0	1	1	0	0	0
2	0	2	1	0	0	0
3	0	3	1	0	0	0
4	0	4	1	0	0	0
5	1	5	2	0	0	0
6	1	7	2	0	0	0
7	1	9	2	0	0	0
8	0	0	0	1	0	3
9	0	0	0	1	3	3
10	0	0	0	2	6	1
11	0	0	0	2	7	1
12	0	0	0	2	8	1
13	0	0	0	2	9	1
14	0	0	0	2	10	1

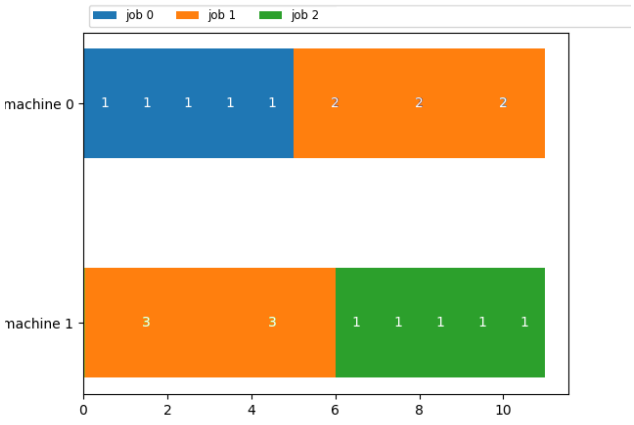

Figure 2: Example of an output file for the problem shown in Figure 1. At the top, each row of the table corresponds to an executed task. The first column assigns a unique id to each task for the sake of clarity. The following groups of three columns correspond to each PU. For each task/PU, the type of task, the start time, and its duration are indicated. If the duration is 0, it is understood that the task has not been executed on that PU. At the bottom of the figure, the output graph for the problem is shown. Each colour corresponds to a type of executed task, every bar is related to every PU or machine and the horizontal axis represents the time units for the execution.

does not show this information directly but converts it into a more visual format that is easier to interpret. For this purpose, we have relied on some works available in the literature to represent the results. In particular, we have relied on the work of Ku et al. [19], and on the implementations available in the D-Wave repository (especially those focused on scheduling problems) [20]. Such implementations include useful routines to transform the output of the problem into a more user-friendly format. The adaptation of these data processing routines to the UPM scheduling problem is simply a matter of Python programming skills. Examples of the output in text and graphical format can be seen in Fig 2.

Scheduled tasks, $N = J$	**Number of PUs (**M**)**
4	1249
8	624
16	312
32	156
64	78
128	39
256	19
512	9
1024	4
2048	2

Table 1: Maximum number of different tasks and PUs using 5000 qubits and the model defined in Eq. 4.

5. Evaluation

The evaluation of the software has been split up into several parts. First, the possible problem sizes to be addressed are studied using first the model described by Eq. 4 and second by Eq. 6. The size of each problem depends on the number of tasks and the number of PUs for the case of Eq. 4, and on the number of different tasks, the maximum number of repetitions, and the number of PUs for the case of Eq. 6. Second, the accuracy of the computed solution on the quantum annealer for UPM scheduling has been analysed. Finally, the impact of the extensions introduced in the model is studied by the analysis of a case study.

5.1. Resource assessment

The D-Wave device on which the software has been tested has 5000 qubits. According to the CQM model, when all tasks are different, $N = J$, then $J \times M$ binary variables and an integer variable to the make-span are involved in the model described in Eq. 4. Ideally, the variable-qubit correspondence is direct. This approach allows us to solve any problem with J different tasks without repetition and M PUs as long as $J \times M < 5000$. That is, if we set a number of PUs M, the number of possible tasks will be a maximum of $4999/M$, and if we set a number of different tasks, J, the maximum number of PUs we can include in the planning will be $4999/J$. However, in practice, some extra qubits are needed for topology reasons, so it is not possible to use the 5000 qubits in the way described. Table 1 shows the maximum possible values of each variable as a function of the value of the other.

For the case where tasks can be grouped by type, an integer variable is still dedicated to the make-span, but in this case, the problem needs $J \times R \times M$ binary variables. However, according to Eq. 6, the number of repetitions is represented as $B = int[log2(R + 1) + 1]$, so the actual number of variables will be $J{\times}B{\times}M$ (again, a certain number of qubits must be dedicated to allowing correct transpilation to the topology of the quantum computer). Again, $J \times B \times M < 5000$ must be satisfied, so any combination of J, R and M values that satisfies this expression is feasible to be solved by the proposed software. In this case, the introduction of B allows the number of executed tasks to be greatly increased if they can be grouped into types. Since the representation of the number of tasks is the one that allows expressing larger numbers occupying fewer qubits, R (B) is the variable that can grow the most so that if the problem contains few PUs and types of tasks, it can be solved involving millions of tasks, since in this case $N = J \times R$. This is in contrast to the data shown in Table 1. Table 2 shows an example with 16 PUs ($M = 16$) and 7 types of tasks ($J = 7$) for several values of R. It can be seen how more than $4.8E+12$ repetitions can be allowed for each task type. That is, more than $N = 3.3E + 13$ tasks can be scheduled in total. This is much higher than the 300 tasks we could solve with 7 PUs using the former formulation.

5.2. Validation of results

To test the accuracy of the software, it has been used to solve more than 50 scheduling random problems by varying J, M, and R and keeping the number of executed tasks as small as possible. To perform this, a Python script was developed to generate input files with the corresponding configurations quickly and easily. This script accepts as input per command J, M, R, the maximum time that a task can last (we will denote t), and optionally the name of the output file. The name of the output file is, by default, instance_J_R_M.txt. The file will be in the format specified by Figure 1, but naturally adapted to the specific parameters. The time for each type of task on each PU will be a random value between 1 and t, both values included.

Once the test files have been obtained, the optimal time for the planning of the problems they represent has been calculated using AMPL [14] and CPLEX [15]. The optimal value of each problem has been established as the maximum make-span for the execution of that problem with the proposed software with the aim of verifying whether it is capable of finding a schedule in that time. A maximum execution time in D-Wave of 10 seconds has been set. In all tested cases, the software was able to find a valid schedule at the optimal time. As a simple test, times shorter than the optimal time have also been tested. In such cases, the software has correctly indicated that it is not possible to find a task/PU configuration that solves the problem in the given time.

Additionally, a case study is analysed to illustrate the impact of the priorities and the switching delay in the computed scheduling. So, the scheduling of seven types of tasks, $J = 7$, with two hundred and fifty repetitions, $R = 250$, on fifteen PUs,

R	B	Variables (qubits)	Scheduled tasks, N
4	3	336	28
8	4	448	56
16	5	560	112
32	6	672	224
64	7	784	448
128	8	896	896
256	9	1008	1792
512	10	1120	3584
5.72E+5	20	2240	4.00E+6
5.86E+8	30	3360	4.10E+9
6.00E+11	40	4480	4.20E+12
1.20E+12	41	4592	8.40E+12
2.40E+12	42	4704	1.68E+13
4.80E+12	43	4816	3.36E+13

Table 2: Number of variables (qubits) used varying the repetitions R for an example with $M = 16$ and $J = 7$. In the quantum device, R is expressed as $B = int[log_2(R + 1) + 1]$ so, for clarity of display, B is also shown. Furthermore, column 'Scheduled tasks' identifies the total number of executed tasks, calculated by multiplying the number of task types by the number of repetitions ($J \times R$).

$M = 15$, is considered. So the number of total tasks to be scheduled is $N = 1750$. Therefore, the proposed QUBO models require at least 850 qubits of the D-Wave platform. The execution times of every kind of task on every PU, are defined in the following matrix $\{t_{\alpha,j}\}$

$$\begin{pmatrix}
800 & 10359 & 27900 & 15689 & 23711 & 107272 & 38424 \\
5296 & 27628 & 5800 & 15005 & 18294 & 152253 & 43881 \\
5098 & 7200 & 27082 & 28413 & 22429 & 126957 & 35278 \\
4800 & 22090 & 21537 & 17275 & 24599 & 112874 & 35849 \\
5942 & 26295 & 12315 & 21808 & 19492 & 102791 & 38899 \\
6000 & 12171 & 23337 & 20775 & 23087 & 110362 & 15600 \\
2630 & 13222 & 14550 & 10400 & 19931 & 172546 & 38835 \\
3101 & 28517 & 24094 & 10885 & 24600 & 171713 & 44200 \\
2391 & 16068 & 17410 & 16494 & 24460 & 118324 & 41683 \\
3762 & 18997 & 10141 & 23692 & 20731 & 110300 & 37766 \\
2016 & 19772 & 23306 & 15768 & 18917 & 111734 & 15500 \\
5847 & 13227 & 5800 & 21090 & 23924 & 112981 & 15400 \\
4356 & 22400 & 25877 & 15833 & 24683 & 100522 & 41829 \\
4640 & 13805 & 2240 & 2930 & 4450 & 12050 & 7650 \\
4243 & 12184 & 2240 & 2930 & 4450 & 12050 & 7650
\end{pmatrix}$$

where every row α is related to every PU with $1 \leq \alpha \leq 15$ and every column to every kind of task $1 \leq j \leq 7$ and the values represent the runtimes of every kind of task on every PU in a unit of time. Notice that this matrix is the transpose of the matrix used in the models above defined.

It is relevant to underline that in this case the execution times have been defined according to a pattern. The values of each column, j, of the matrix $\{t_{\alpha,j}\}$ are of similar order of magnitude, i.e. every kind of task is characterized by its workload and its execution time is of the same order of magnitude for most PUs. Furthermore, the wide range of values of $t_{\alpha,j}$ shows the heterogeneity of the types of tasks and the capacity of the PUs to complete the workload, which is an additional challenge to find optimal scheduling.

To study the impact of the priority of tasks on the computed solution, a set of types of tasks with maximal priority is defined, $j_p = 1, 2, 3, 4$ and the results plotted in Fig. 3 are analysed. Every bar of Fig. 3 represents the execution time on every machine or PU to complete its scheduled tasks. Two colours define the bars to show the time spent on prioritized tasks, in red, and not prioritized, in green. Fig. 3 on the left shows the computed scheduling without a definition of priorities, that is, when the model of Eq. 8 is used with $A = 0$. As shown in the figure, the prioritized tasks are not considered and the main goal is to minimize the global make-span, almost 1500000 units of time in this case. However, the prioritized tasks finish at the same time that the rest of the tasks, since the priority is not considered in the model. When the model of Eq. 8 is used and the prioritized tasks are set $j_p = 1, 2, 3, 4$ with $A = 4$ then the scheduling is different, as shown in Fig. 3 (b). The value selected for A has been chosen to be large enough to give weight to the priorities. It may be tempting in the first instance to choose a tremendously large value to ensure the condition, but this may cause the model to not work properly [3]. Instead, values that are large enough to ensure the condition but keep the results in the same order of values should be used. In this particular case, the value chosen is close to the total time achieved in case (a) after being multiplied by the corresponding time value. Although each case should be evaluated independently, in our experiments it has worked well for us to use a value A of the same order as the time achieved without applying the priorities. Fig. 3 illustrates how the prioritized tasks are planned first and all of them finish after 900000 units of time, as a counterpart the global make-span increases, since it is not the main goal of our problem. However, the exploitation of all PUs is balanced since all of them process during a similar time, even when the priority is considered. So, the application of the proposed model reduces the time for the execution of the tasks with priority and distributes the available PUs in a balanced way.

To illustrate how the switching delays affect the computed scheduling, the results of Fig. 4 are analysed. On the top, the optimal solution of the case study when the model of Eq. 10

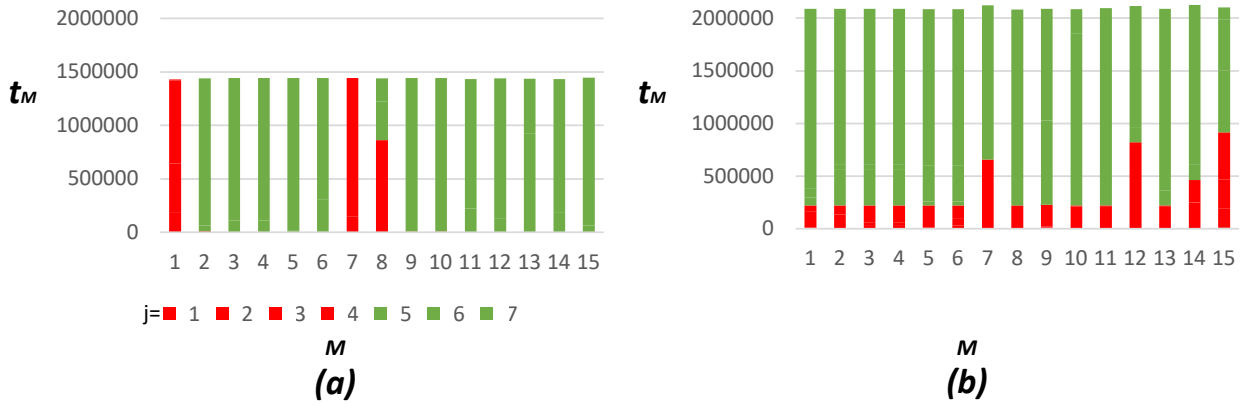

Figure 3: Scheduling of tasks with/without priority, in red/green colours, when the model of Eq. 8 executed on the D-Wave platform has the value $A = 0$ (Fig. (a)); and $A = 4$ (Fig. (b)).

without any switching delay on PUs is shown, that is $\Delta = 0$. Every PU attends different kinds of tasks represented in different colours, and the make-span is optimized since it corresponds to the validated cases described at the beginning of this subsection. However, when the switching delay is considered this solution is not optimal, as illustrated in the pairs of figures 4 (b)-(c) and 4 (d)-(e). These figures represent the times spent by every PU on every task and also the switching time is plotted in brown colour. Every switching introduces a delay in the activity of the PUs, even to start the computation of the first attended task. So, all PUs include a delay even without switching of tasks, and in general as more switching more delay. In this regard, it is necessary to emphasize that not all task-type switches can be appreciated in the graphs of Fig. 4. Since the time for a new task type may be so short that it falls below the accuracy of the graph scale. Fig. 4 (b) and (d) plot the task execution time and the switching delay for the optimal scheduling of Fig. 4 (a) obtained with the simplest model without switching delays. In Fig. 4 (b) and (d) every task switch introduces a delay of 30% and 40% of the runtime without delay, respectively. The time to complete the whole set of tasks is more than 2600000 and 3000000 units of time respectively for mentioned delays. Fig. 4 (c) and (e) show the computed task scheduling with the model of Eq. 10 for switching delays of 30% and 40% respectively. In this case, as expected, the number of switching of scheduled tasks decreases as the switching delay increases. Moreover, the time to complete the whole set of tasks is reduced to 2200000 and 2400000 units of time respectively. So, the use of the proposed model that considers the switching delay allows reducing the make-span in percentages of 18% and 25% respectively. Therefore, the model of Eq. 10 allows the makespan to be reduced when task-switching delays are appreciable.

6. Conclusions

The scheduling of heterogeneous tasks on unrelated parallel machines has been solved using quantum annealing. The proposed solution has been centred on the scheduling of unrelated parallel machines, where a classification of tasks is defined according to their processing time on the available processing elements. These processing times are different and known in advance. Two relevant aspects of the scheduling have been introduced: the existence of some degree of priority in the type of task, and the introduction of a delay every time a processing unit changes the type of task that executes. The three versions of the problem have been defined as QUBO models and developed with the programming tools supplied with the D-Wave quantum annealer. To reduce the number of required qubits, the number of repetitions of every type of task is coded with binary variables. This way, large-scale scheduling problems can be addressed with real quantum annealers. The results obtained have been compared with classical methods such as CPLEX and AMPL to show the effectivity of the quantum algorithm. A case study has been analysed to illustrate how the scheduling solutions with priorities and switching delays computed on quantum annealer achieve the goals of the problem in each case. It should be noted that the methodology used in this work can be applied to other combinatorial optimization problems and the results obtained show the great potential of quantum annealers to solve this kind of optimization problem.

Acknowledgements

This work has been supported by the projects: PID2021-123278OB-I00 and PID2021-127836NB-I00 (funded by MCIN /AEI/10.13039/501 100011033/FEDER “A way to make Europe”); P20_00748 and UAL2020-TIC-A2101 (funded by Junta de Andalucía and the European Regional Development Fund, ERDF).

Authors would also like to thank Professor Dr. Elías F. Combarro, from the Informatics Department of the University of Oviedo, Spain, because this work has been possible thanks to the contents of his interesting lectures about Quantum Computing at Almería University.

References

[1] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, Y. Wang, The unconstrained binary quadratic programming problem: A survey, J. Comb. Optim. 28 (1) (2014) 58–81.
[2] A. Lucas, Ising formulations of many np problems, Frontiers in physics (2014) 5.
[3] E. Combaro, S. González-Castillo, A Practical Guide to Quantum Machine Learning and Quantum Optimization: Hands-on Approach to Modern Quantum Algorithms, Packt Publishing, 2023.
[4] E. K. Grant, T. S. Humble, Adiabatic quantum computing and quantum annealing (07 2020).
[5] J. Gehrke, K. Jansen, S. Kraft, J. Schikowski, A PTAS for scheduling unrelated machines of few different types, in: SOFSEM 2016: Theory and Practice of Computer Science, Springer, 2016, pp. 45–55.
[6] V. Sels, J. Coelho, A. Dias, M. Vanhoucke, Hybrid tabu search and a truncated branch-and-bound for the unrelated parallel machine scheduling problem, Comput Oper Res 53 (2015) 107–117.
[7] T. Wang, Z. Liu, Y. Chen, Y. Xu, X. Dai, Load balancing task scheduling based on genetic algorithm in cloud computing, in: Proceedings of the 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing. DASC '14, IEEE Computer Society, 2014, pp. 146–152.
[8] M. Lewis, B. Alidaee, G. Kochenberger, Using XQx to model and solve the uncapacitated task allocation problem, Oper. Res. Lett. 33 (2) (2005) 176–182.
[9] C. Carugno, M. Ferrari Dacrema, P. Cremonesi, Evaluating the job shop scheduling problem on a D-wave quantum annealer, scientific reports (2022) 1–11.
[10] F. Orts, A. Puertas, G. Ortega, E. Garzón, Quantum annealing to solve the unrelated parallel machine scheduling problem, in: PPAM 2022. Lecture Notes in Computer Science, Springer, 2023.
[11] P. L. M. Naeimeh Mohseni, T. Byrnes, Ising machines as hardware solvers of combinatorial optimization problems, Nature Reviews Physics.
[12] F. Glover, G. Kochenberger, R. Hennig, Y. Du, Quantum bridge analytics i: a tutorial on formulating and using qubo models, Annals of Operations Research (2022) 1–43.
[13] F. Orts, G. Ortega, A. M. Puertas, E. M. Garzón, I. García, On solving the unrelated parallel machine scheduling problem: active microrheology as a case study, The Journal of Supercomputing 76 (2020) 8494–8509.
[14] R. Fourer, D. M. Gay, B. W. Kernighan, AMPL. A modeling language for mathematical programming, Thomson, 2003.
[15] C. Bliek1ú, P. Bonami, A. Lodi, Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report, in: Proceedings of the twenty-sixth RAMP symposium, 2014, pp. 16–17.
[16] A. S. Koshikawa, M. Ohzeki, T. Kadowaki, K. Tanaka, Benchmark test of black-box optimization using D-Wave quantum annealer, Journal of the Physical Society of Japan 90 (6) (2021) 064001.
[17] F. Phillipson, H. S. Bhatia, Portfolio optimisation using the D-Wave quantum annealer, in: International Conference on Computational Science, Springer, 2021, pp. 45–59.
[18] D. Willsch, M. Willsch, C. D. Gonzalez Calaza, F. Jin, H. De Raedt, M. Svensson, K. Michielsen, Benchmarking advantage and D-Wave 2000Q quantum annealers with exact cover problems, Quantum Information Processing 21 (4) (2022) 1–22.
[19] W.-Y. Ku, J. C. Beck, Mixed integer programming models for job shop scheduling: A computational analysis, Computers & Operations Research 73 (2016) 165–173.
[20] Ocean SDK demos.
URL `https://github.com/dwavesystems/demos`

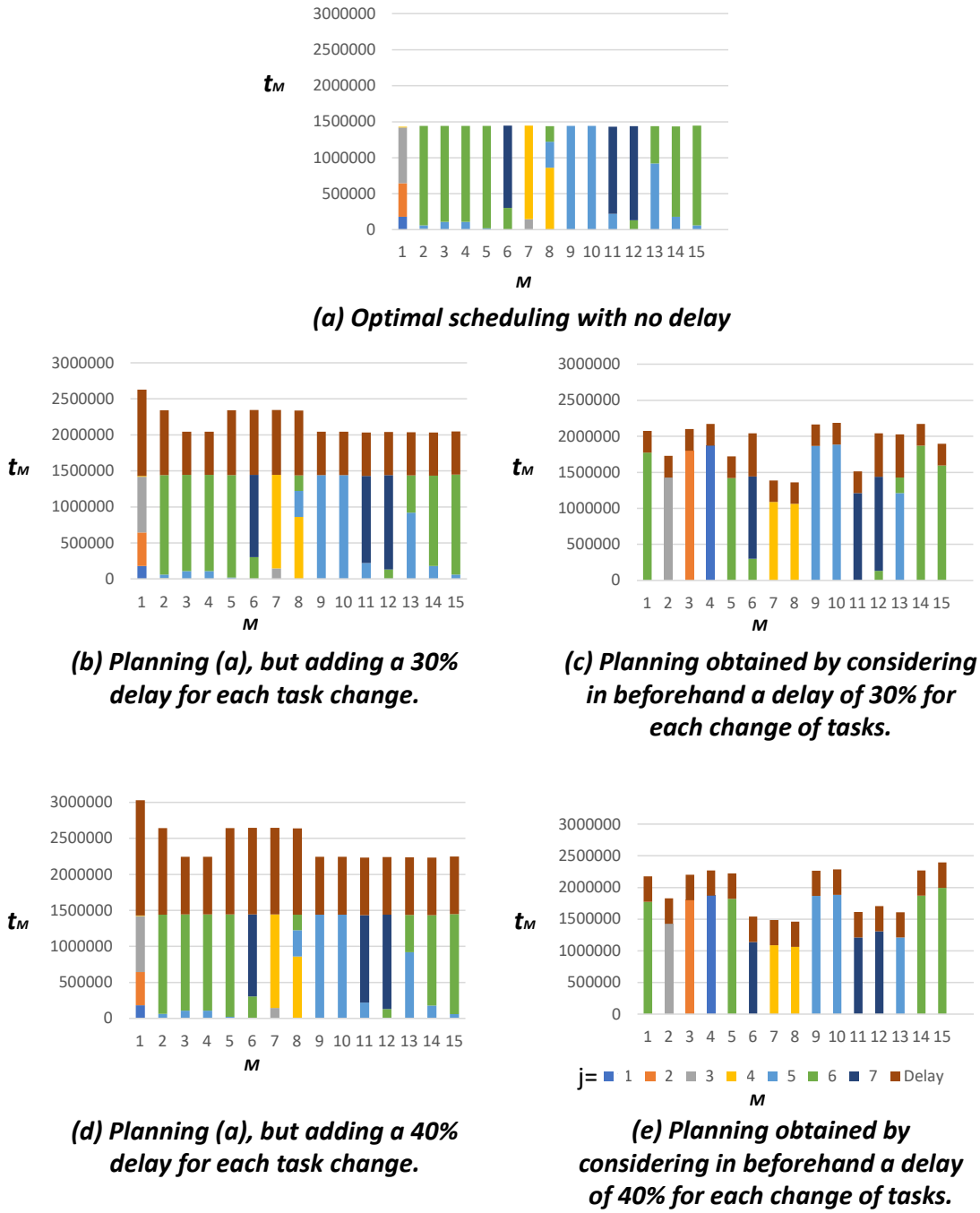

Figure 4: (a) Computed Scheduling with the model of Eq. 10 with switching delay null; (b) and (d) plot the optimal scheduling of (a) including the switching delays of tasks of 30% and 40% of runtime for every switching respectively; (c) and (e) Computed Scheduling with the model of Eq. 10 with switching delay of 30% and 40% of runtime for every switching respectively