

# An $R || C_{max}$ Quantum Scheduling Algorithm

Feng Lu and Dan C. Marinescu  
School of Computer Science

University of Central Florida, P. O. Box 162362  
Orlando, FL 32816-2362

Email: (lvfeng, dcm)@cs.ucf.edu

September 12, 2018

## Abstract

Grover's search algorithm can be applied to a wide range of problems; even problems not generally regarded as searching problems, can be reformulated to take advantage of quantum parallelism and entanglement, and lead to algorithms which show a square root speedup over their classical counterparts.

In this paper, we discuss a systematic way to formulate such problems and give as an example a quantum scheduling algorithm for an  $R||C_{max}$  problem.  $R||C_{max}$  is representative for a class of scheduling problems whose goal is to find a schedule with the shortest completion time in an unrelated parallel machine environment.

Given a deadline, or a range of deadlines, the algorithm presented in this paper allows us to determine if a solution to an  $R||C_{max}$  problem with  $N$  jobs and  $M$  machines exists, and if so, it provides the schedule. The time complexity of the quantum scheduling algorithm is  $\mathcal{O}(\sqrt{M^N})$  while the complexity of its classical counterpart is  $\mathcal{O}(M^N)$ .

## 1 Introduction

In recent years we have witnessed significant theoretical and some encouraging experimental results in the area of quantum computing. In 1994, Peter Shor found a polynomial time algorithm for the factorization of  $n$ -bit numbers on quantum computers [20]. His discovery generated a wave of enthusiasm for quantum computing, for two major reasons: the intrinsic intellectual beauty of the algorithm and the fact that efficient integer factorization is a very important practical problem. The security of widely used cryptographic protocols is based upon the conjectured difficulty of the factorization of large integers.

Shor's algorithm reduces the factorization problem to the problem of finding the period of a function, but uses quantum parallelism to find a superposition of all values of the function in one step. Then the algorithm calculates the Quantum Fourier Transform of the function, which sets the amplitudes into multiples of the fundamental frequency, the reciprocal of the period. To factor an integer, Shor's algorithm measures the period of the function<sup>1</sup>.

In 1996, Grover described a quantum algorithm for searching an unsorted database containing  $N$  items in a time of order  $\sqrt{N}$  while on a classical computer the search requires a time of order  $N$  [9]. The critical aspect of a quantum algorithm is to create a superposition of all possible states and amplify the "solution". The speedup of Grover's algorithm is achieved by exploiting both quantum parallelism and the fact that in quantum theory a probability is the square of an amplitude. Bennett and his co-workers [1] and Zalka [21] showed that Grover's algorithm is optimal. No classical or quantum algorithm can solve this problem faster than time of order  $\sqrt{N}$ .

<sup>1</sup>A powerful version of the technique used by Shor is the *phase-estimation algorithm* of Kitaev [12]

Grover’s search can be applied to a large number of unstructured problems and lead to a square root speedup over the corresponding classical algorithms. For well structured problems classical algorithms that produce an approximate solution and perform faster exit.

Recently, Furrow discussed applications based upon Grover-type algorithms [6]. He considered three classes of applications: a) Graph algorithms, e.g. Breadth-First Search (BFS), Depth-First Search (DFS), bipartite matching. b) Computational Geometry algorithms, e.g. Maximum points on a line. c) Dynamic programming algorithms, such as coin changer. The author reports the quantum versus classical complexity for BFS and DFS,  $\mathcal{O}(\sqrt{VE} \lg V)$ , versus  $\mathcal{O}(E)$ , with  $V$  the number of vertices and  $E$  the number of edges of the graph; for bipartite matching,  $\mathcal{O}(V \sqrt{(E+V) \lg V})$ , versus  $\mathcal{O}((E+V)\sqrt{V})$ ; for maximum points that lie on a line in  $\mathbb{R}^2$ , out of  $N$  points,  $\mathcal{O}(N^{3/2} \lg N)$  versus  $\mathcal{O}(N^2 \lg N)$ , and so on.

Most of the problems discussed in [6] are intrinsically search problems and the idea of applying Grover’s search comes naturally to mind. There is an even larger class of problems which, at the first sight, do not seem directly related to Grover’s search. Applications such as scheduling and resource allocation are not naturally search problems; nevertheless they share some common properties, can be reformulated to take advantage of quantum parallelism and entanglement, and lead to algorithms which show polynomial speedups over their classical counterparts.

A scheduling problem is characterized by a tuple  $(\alpha \mid \beta \mid \gamma)$  where  $\alpha$  denotes the machine environment,  $\beta$  summarizes the set of constraints, and  $\gamma$  denotes the optimality criterion. The makespan of a schedule,  $C_{max}$  is the maximum completion time of any job in the schedule. For example,  $P||C_{max}$  and  $R||C_{max}$  require the shortest makespan and apply to identical machine environment and, respectively, a non-homogeneous one.

When we turn our attention to problems when a deadline is imposed, or when we wish to find a schedule with a given range of possible average completion time we discover that a full range of scheduling problems have a quantum counterpart which can take advantage of Grover’s search.

We illustrate these ideas with an example: given a deadline, or a range of deadlines, the algorithm presented in this paper allows us to determine if a solution to an  $R||C_{max}$  problem with  $N$  jobs and  $M$  machines exists, and if so, it provides the schedule. The time complexity of the quantum scheduling algorithm is  $\mathcal{O}(\sqrt{M^N})$  while the complexity of its classical counterpart is  $\mathcal{O}(M^N)$ .

Real-time systems are subject to deadlines and Quality of Service (QoS) constraints imposed to many systems require a given range of average completion times. Thus, the classes of scheduling algorithms we discuss in this paper are of significant practical importance. Such algorithms have a quantum counterpart that enjoy a square root speedup.

## 2 Scheduling Algorithms

Scheduling is the problem of assigning tasks to a set of resources subject to a set of constraints, over time, in an “optimal” manner.

We are given a set of  $N = 2^n$  jobs,  $\mathcal{J} = \{\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_N\}$  and a set of  $M = 2^m$  machines,  $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_M\}$ . A *schedule*  $\mathcal{S}$  for the sets  $(\mathcal{J}, \mathcal{M})$  specifies which  $T_{ij}$  units of time machine  $\mathcal{M}_j$  uses to process job  $\mathcal{J}_i$ . We call  $C_i^{\mathcal{S}}$  the *completion time* of job  $\mathcal{J}_i$  under the schedule  $\mathcal{S}$ . The *makespan* of the schedule  $\mathcal{S}$  is the maximum completion time of any job in schedule  $\mathcal{S}$ :

$$C_{max}^{\mathcal{S}} = \max_i C_i^{\mathcal{S}}.$$

We are often interested in scheduling problems involving *multiple* machines. We distinguish three cases in the *parallel machine environment*:

1. *Identical parallel environment.* All the machines are identical and job  $\mathcal{J}_i$  requires  $T_i$  units of time on any machine,
2. *Uniformly related parallel environment.* Each machine  $\mathcal{M}_j$  has a speed  $s_j > 0$  and job  $\mathcal{J}_i$  if processed entirely on machine  $\mathcal{M}_j$  would take  $T_i/s_j$  units of time, and

3. *Unrelated parallel machine environment.* Different machines have different capabilities and the speed of machine  $\mathcal{M}_j$  on job  $\mathcal{J}_i$  is  $s_{ij}$ . Then the processing time of job  $\mathcal{J}_i$  on machine  $\mathcal{M}_j$  is  $T_{ij} = T_i/s_{ij}$ ,  $0 \leq T_{ij} \leq Q$  and  $Q = 2^q$ .

These machine environment are denoted by  $P, Q$ , and  $R$ , respectively.

Examples of scheduling constraints include deadlines (e.g., job  $i$  must be completed by time  $t$ ), resource capacities (e.g., there are only 5 machines), precedence constraints on the order of tasks (e.g., one must be done before another), and priorities on tasks (e.g., finish job  $j$  as soon as possible while meeting the other deadlines). A *priority rule* assigns to job  $\mathcal{J}_i$  a priority  $\pi_i$ . A *busy schedule* defines the situation where when one machine becomes available it starts processing the job with the highest priority.

Each scheduling problem could have problem specific optimization criteria. The one which requires the minimization of the makespan is referred to in scheduling theory as  $C_{max}$ . Other optimization criteria can be considered. For example, we may wish to optimize the average completion time of all jobs:

$$\frac{1}{N} \sum_{i=1}^N C_i^S$$

an optimization criterion denoted as  $\sum_{i=1}^N C_i^S$ .

Many scheduling problems are  $\mathcal{NP}$ -hard on classical computers [3] and have proved to be very difficult even for a relatively small instance. For example, a 10-job-10-machine job-shop scheduling problem posed in 1963 remained unsolved until 1989 [4]. Most classical scheduling algorithms gain speedup only on some special cases with relatively small instance or they try to find good schedules instead of the optimal one. Furrow[6] also gave a special case of the  $P||C_{max}$  problem that can be solved by dynamic programming, which requires that the number of different jobs processing times be bounded by a constant. It turns out that there are polynomial approximations for some  $P||C_{max}$  problems.

We consider an  $R||C_{max}$  scheduling problem in which all machines are unrelated. All jobs are available at the beginning time and there are no precedence constraints. As no preemption is allowed, once job  $\mathcal{J}_i$  started processing on machine  $\mathcal{M}_j$  it must complete its execution before another job,  $\mathcal{J}_k$  can be processed on  $\mathcal{M}_j$ .

Given a set of  $N = 2^n$  jobs and  $M = 2^m$  machines we can construct  $2^{m2^n}$  different schedules. This  $R||C_{max}$  scheduling problem is  $\mathcal{NP}$ -hard and is difficult for classical algorithms even with small instance. Up to now, most classical algorithms use linear programming based rounding techniques(LP-rounding) to search an approximate schedule instead of the optimal one [8, 19]. If the number of machines  $m$  is part of the input, the best approximation algorithm to date is a 2-approximation by Lenstra, Shmoys and Tardos [13] which can find a schedule with  $C_{max} < 2 * C_{max}^{opt}$ . Moreover, the problem cannot be approximated within a factor strictly smaller than 3/2, unless  $P=NP$  [13]. No proper classical algorithm addresses a general case of searching the optimal schedule except the exhausting search which has time complexity  $\mathcal{O}(M^N)$  on a classical computer. The paper suggests a reformulation of such problems to take advantage of Grover-type search and gain square root speedup on searching the optimal schedules over their classical counterparts.

### 3 Information Encoding

Consider a set of  $N = 2^n$  jobs running on  $M = 2^m$  unrelated machines. We assume that the jobs could have different processing times on different machines and that the processing times are integers in the range  $0 \leq T_{ij} < Q = 2^q$ ,  $1 \leq i \leq 2^n$ ,  $1 \leq j \leq 2^m$ . We also assume that we have a quantum system with  $r = m + q$  qubits for each job.

Given job  $\mathcal{J}_i$  running on machine  $\mathcal{M}_j$ , we encode the *job-machine* information as a vector  $|e_i^j\rangle$  obtained as the tensor product of the machine index,  $j$ , and the processing time,  $T_{ij}$ :

$$|e_i^j\rangle = |j\rangle \otimes |T_{ij}\rangle.$$

Then we define *job state vectors* for job  $i$ ,  $|J_i\rangle$  as any superposition of its job-machine vectors. First, we consider an equal superposition of job-machine vectors as:

$$|J_i\rangle = \frac{1}{2^{m/2}} \sum_{j=1}^{2^m} |e_i^j\rangle.$$

A *schedule* is a tensor product of job-machine vectors:

$$|\mathcal{S}_k\rangle = \otimes_{i=1}^{2^n} |e_i^{j_i}\rangle, \quad 1 \leq j_i \leq 2^m,$$

which includes one job-machine vector for each job. A specific machine may be present in multiple job-machine vectors, when the schedule requires multiple jobs to be executed on the same machine, or may not appear in any job machine vectors of the schedule  $|\mathcal{S}_k\rangle$  if none of the jobs is scheduled on that machine.

The equal superposition of all schedules is:

$$|\mathcal{S}\rangle = \frac{1}{\sqrt{\sigma}} \sum_{k=1}^{\sigma} |\mathcal{S}_k\rangle, \quad \sigma = 2^{m2^n}$$

Let  $\Omega$  be an operator which given a schedule constructs the running time on each machine for that schedule. When applied to the equal superposition of all schedules, it produces a superposition of the running time  $|\mathcal{T}\rangle$  on each machine for all schedules:

$$|\widehat{\mathcal{ST}}\rangle = \Omega(|\mathcal{S}\rangle | 0\rangle).$$

where,  $|\widehat{\mathcal{ST}}\rangle$  denotes the entangled state of  $\mathcal{S}$  and  $\mathcal{T}$ , while the tensor product of  $\mathcal{S}$  and  $\mathcal{T}$  is  $|\mathcal{ST}\rangle$ .

Let  $\Delta$  be an operator which computes a superposition of the makespan of all schedules:

$$|\widehat{\mathcal{STC}_{max}}\rangle = \Delta(|\widehat{\mathcal{ST}}\rangle | 0\rangle) = \Delta \Omega(|\mathcal{S}\rangle | 0\rangle | 0\rangle).$$

Figure 1 outlines our procedure to produce an equal superposition of the makespans of all schedules. We now turn our attention to the quantum circuits to carry out the transformations discussed in this section and to the question how to obtain from the superposition of all makespans the optimal one.

First, we need to prepare the job vector  $|J_i\rangle$  in an equal superposition state which includes the processing times of job  $\mathcal{J}_i$  on all machines, as shown in Figure 2. We use  $m$  index qubits to control the job-machine information encoding. As each index qubit is prepared in state  $1/\sqrt{2}(|0\rangle + |1\rangle)$ , the target qubits will be prepared in superpositions of all possible job-machine states,  $e_i^j$ ,  $1 \leq i \leq n$ ,  $1 \leq j \leq m$ .

**Example:** Table 1 summarizes the processing time of 8 jobs on 4 machines, where  $0 < T_{ij} < 2^4 = 16$ . Thus  $n = 3$ ,  $m = 2$ , and  $q = 4$ . The running time of  $\mathcal{J}_1$  on machines  $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$  and  $\mathcal{M}_4$  are respectively, 1, 3, 7, and 15 units of time.

The four vectors used to encode the processing time of job  $\mathcal{J}_1$  on machines  $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$  and  $\mathcal{M}_4$  are respectively:

$$|e_1^1\rangle = |000001\rangle, \quad |e_1^2\rangle = |010011\rangle, \quad |e_1^3\rangle = |100111\rangle, \quad \text{and} \quad |e_1^4\rangle = |111111\rangle.$$

For  $\mathcal{J}_2$  the basis vectors are  $|000010\rangle, |010001\rangle, |101001\rangle, |110011\rangle$ , and so on.

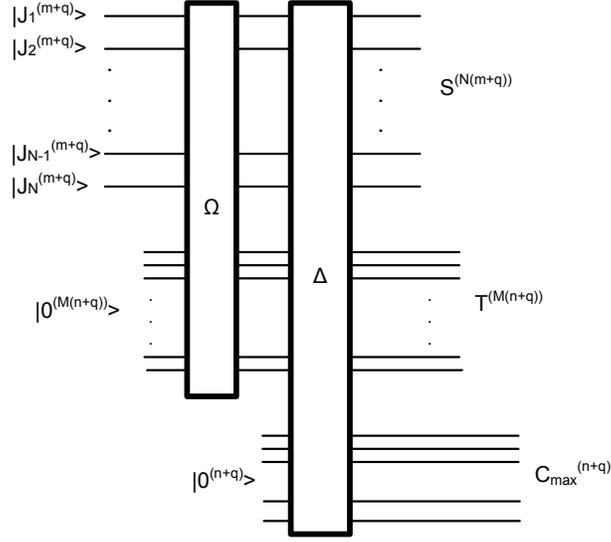


Figure 1: Outline of our algorithm to prepare the makespan vector. First we prepare job vectors in an equal superposition of all possible schedules  $|\mathcal{S}^{(N(m+q))}\rangle$ . Using a quantum circuit  $\Omega$  with inputs  $|\mathcal{S}^{(N(m+q))}\rangle$  and a register of  $M(n+q)$  qubits in state  $|0\rangle$ , we construct the superposition of the running time on each machine for every schedule. Then we construct the makespan of each schedule using the operation  $\Delta$ . Superscripts indicate the number of qubits for vectors. Note that the number of jobs is  $N = 2^n$ , the number of machines is  $M = 2^m$ , and the maximum execution time of a job on any machine is  $Q = 2^q$ .

Table 1: The processing time of 8 jobs on 4 machines

Job/Machine	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	Job/Machine	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$
$\mathcal{J}_1$	1	3	7	15	$\mathcal{J}_5$	15	12	3	10
$\mathcal{J}_2$	2	1	9	3	$\mathcal{J}_6$	10	7	8	14
$\mathcal{J}_3$	6	2	5	8	$\mathcal{J}_7$	5	2	3	9
$\mathcal{J}_4$	11	13	7	4	$\mathcal{J}_8$	1	10	11	13

Figure 3 shows the circuit used to prepare the job vector  $|J_1\rangle$  for our example. The job state  $|J_1\rangle$  is prepared in an equal superposition of basis states:

$$|J_1\rangle = \frac{1}{2}(|000001\rangle + |010011\rangle + |100111\rangle + |111111\rangle).$$

We can prepare other job vectors in the same way, for example:

$$|J_2\rangle = \frac{1}{2}(|000010\rangle + |010001\rangle + |101001\rangle + |110011\rangle).$$

A schedule vector  $|\mathcal{S}\rangle$  is the tensor product of all job vectors. As each job vector is prepared as an equal superposition, the schedule vector is in the equal superposition of all possible schedules.

We now provide two examples of schedule vectors:

(i) Schedule  $\mathcal{S}_1$ :

$$[\mathcal{J}_1 \mapsto \mathcal{M}_1, \mathcal{J}_2 \mapsto \mathcal{M}_2, \mathcal{J}_3 \mapsto \mathcal{M}_1, \mathcal{J}_4 \mapsto \mathcal{M}_4, \mathcal{J}_5 \mapsto \mathcal{M}_3, \mathcal{J}_6 \mapsto \mathcal{M}_2, \mathcal{J}_7 \mapsto \mathcal{M}_3, \mathcal{J}_8 \mapsto \mathcal{M}_1]$$

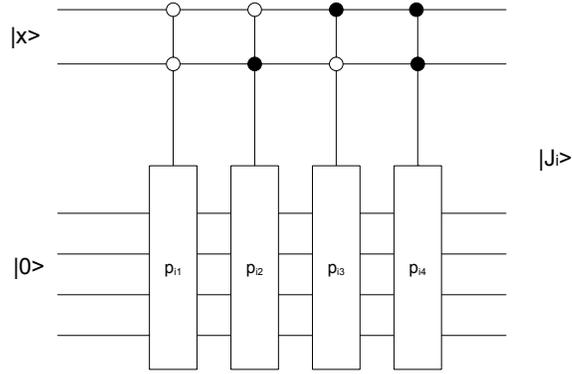


Figure 2: A quantum circuit to prepare the job state vectors with  $m = 4$ . In this case  $|x\rangle$  is a set of two control qubits. Each control qubit is set to  $1/\sqrt{2}(|0\rangle + |1\rangle)$ . The target qubits will be prepared in a superposition of all possible job-machine states.

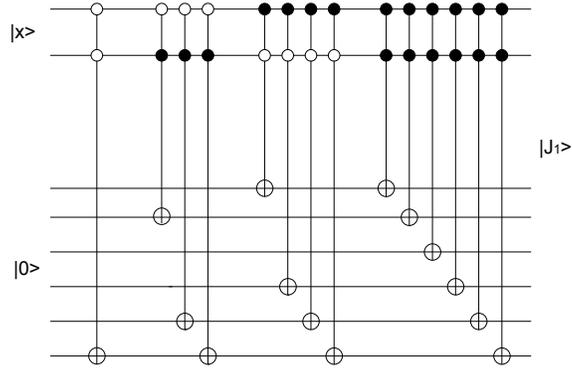


Figure 3: A quantum circuit to prepare  $|J_1\rangle$ . The execution times of  $J_1$  are 1, 3, 7, and 15 units of time respectively.  $|x\rangle$  are control qubits which control the selection of the four job-machine states,  $e_1^j$ ,  $1 \leq j \leq 4$ . The first two qubits of  $|J_1\rangle$  record the machine index, and the remaining four qubits record the time if  $J_1$  is assigned to the corresponding machine.

Schedule  $\mathcal{S}_1$  corresponds to the following job state vectors:

$$\begin{aligned}
 |J_1\rangle &= |\underline{000001}\rangle & |J_2\rangle &= |\underline{010001}\rangle & |J_3\rangle &= |\underline{000110}\rangle & |J_4\rangle &= |\underline{110100}\rangle \\
 |J_5\rangle &= |\underline{100011}\rangle & |J_6\rangle &= |\underline{010111}\rangle & |J_7\rangle &= |\underline{100011}\rangle & |J_8\rangle &= |\underline{000001}\rangle
 \end{aligned}$$

The schedule vector for schedule  $\mathcal{S}_1$  is:

$$\begin{aligned}
 |\mathcal{S}_1\rangle &= |\underline{000001}\rangle \otimes |\underline{010001}\rangle \otimes |\underline{000110}\rangle \otimes |\underline{110100}\rangle \\
 &\quad \otimes |\underline{100011}\rangle \otimes |\underline{010111}\rangle \otimes |\underline{100011}\rangle \otimes |\underline{000001}\rangle
 \end{aligned}$$

The completion times on all machines are:

$$C^{A_1}(\mathcal{M}_1) = 1 + 6 + 1 = 8, \quad C^{A_1}(\mathcal{M}_2) = 1 + 7 = 8, \quad C^{A_1}(\mathcal{M}_3) = 3 + 3 = 6, \quad C^{A_1}(\mathcal{M}_4) = 4.$$

The makespan of schedule  $\mathcal{S}_1$  is equal to the largest completion time over all machines:

$$C_{max}^{A_1} = 8.$$

(ii) Schedule  $\mathcal{S}_2$ :

$$[\mathcal{J}_1 \mapsto \mathcal{M}_2, \mathcal{J}_2 \mapsto \mathcal{M}_1, \mathcal{J}_3 \mapsto \mathcal{M}_3, \mathcal{J}_4 \mapsto \mathcal{M}_4, \mathcal{J}_5 \mapsto \mathcal{M}_3, \mathcal{J}_6 \mapsto \mathcal{M}_2, \mathcal{J}_7 \mapsto \mathcal{M}_1, \mathcal{J}_8 \mapsto \mathcal{M}_2]$$

The schedule vector is:

$$|\mathcal{S}_2\rangle = |\underline{010011}\rangle \otimes |\underline{000010}\rangle \otimes |\underline{100101}\rangle \otimes |\underline{110100}\rangle \\ \otimes |\underline{100011}\rangle \otimes |\underline{010111}\rangle \otimes |\underline{000101}\rangle \otimes |\underline{011010}\rangle$$

The schedule vector can also be in a superposition of some basic states, for example, the schedule vector could be in an equal superposition of the two schedules,  $\mathcal{S}_1$  and  $\mathcal{S}_2$ :

$$|\mathcal{S}\rangle = 1/\sqrt{2}(|\mathcal{S}_1\rangle + |\mathcal{S}_2\rangle) \\ = 1/\sqrt{2}(|\underline{000001}\rangle \otimes |\underline{010001}\rangle \otimes |\underline{000110}\rangle \otimes |\underline{110100}\rangle \\ \otimes |\underline{100011}\rangle \otimes |\underline{010111}\rangle \otimes |\underline{100011}\rangle \otimes |\underline{010001}\rangle \\ + |\underline{010011}\rangle \otimes |\underline{000010}\rangle \otimes |\underline{100101}\rangle \otimes |\underline{110100}\rangle \\ \otimes |\underline{100011}\rangle \otimes |\underline{010111}\rangle \otimes |\underline{000101}\rangle \otimes |\underline{011010}\rangle).$$

## 4 The Running Time of Machine $\mathcal{M}_j$ under Schedule $\mathcal{S}_k$

Computing the total running time of machine  $\mathcal{M}_j$  under schedule  $\mathcal{S}_k$  requires summation of the running times of all jobs assigned to  $\mathcal{M}_j$ . A quantum adder similar to a classic adder, e.g.  $|5\rangle \hat{+} |6\rangle = |11\rangle$ , has:

- $n$  inputs  $a_1, a_2, \dots, a_n$ , each one is a register of  $q$  qubits,
- one carry in  $c$ , and
- one output  $S = \sum_{i=1}^n a_i + c$ , a register of  $q + n$  qubits.

To obtain the total running time of machine  $\mathcal{M}_j$  under schedule  $\mathcal{S}_k$  we add the execution times, (in our examples those qubits of a job-machine vector which are not underlined) for all jobs assigned to  $\mathcal{M}_j$  and create a running time vector for machine  $\mathcal{M}_j$ ,  $|T_j\rangle$ . The *running time vector for schedule  $\mathcal{S}_k$*  is the tensor product of the execution time on individual machines under schedule  $\mathcal{S}_k$ :

$$|\mathcal{T}^{\mathcal{S}_k}\rangle = |T_1^{\mathcal{S}_k}\rangle \otimes |T_2^{\mathcal{S}_k}\rangle \dots \otimes |T_M^{\mathcal{S}_k}\rangle.$$

For example, for machine  $\mathcal{M}_1$  under schedule  $\mathcal{S}_1$  (see Section 3):

$$|T_1^{\mathcal{S}_1}\rangle : \quad |0001\rangle \hat{+} |0000\rangle \hat{+} |0110\rangle \hat{+} |0000\rangle \hat{+} |0000\rangle \hat{+} |0000\rangle \hat{+} |0000\rangle \hat{+} |0001\rangle,$$

or

$$|T_1^{\mathcal{S}_1}\rangle : \quad |0001000\rangle.$$

We want to construct a quantum circuit to sum the execution time of jobs assigned to each machine  $\mathcal{M}_j$ . We use the index qubits as control qubits to control the summation of each index, or each machine; the index qubits are entangled with the target qubits which give the running time on that machine. As the input qubits  $|\mathcal{S}\rangle$  are prepared in the equal superposition of all possible schedules, this operation will prepare the running time of machines under all possible schedules.

In Figure 4,  $T_1, T_2, \dots, T_M$  represent the execution time on machines  $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_M$  respectively. When the index qubits are not “active”, the respective input for the summation circuit will be zero.

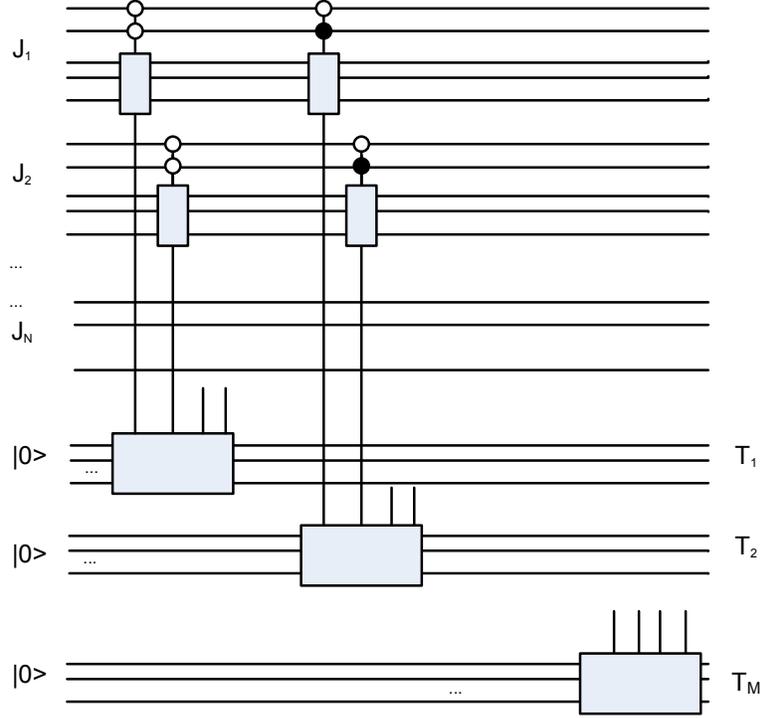


Figure 4: A circuit to compute the sum of the execution time of jobs assigned to each machine.  $J_1, J_2 \dots J_N$  are the job vectors prepared according to the scheme discussed in Section 3, and  $T_1, T_2, \dots T_M$  represent the execution time on machines  $\mathcal{M}_1, \mathcal{M}_2, \dots \mathcal{M}_M$  respectively.

For example, if the index of  $e_2^2 = |010001\rangle$  is 01, rather than 00, then the index qubits of  $e_2^2$  for  $|T_1\rangle$  are not active.

How to implement arithmetic on a quantum computer has been address by many authors. Many detailed quantum circuits and discussions can be found in [7, 5, 14]. For the example discussed in Section 3, after the summation operation, the system will be in the state:

$$|\widehat{ST}\rangle = 1/\sqrt{2}(|S_1T^1\rangle + |S_2T^2\rangle)$$

or

$$\begin{aligned} |\widehat{ST}\rangle = & 1/\sqrt{2}(|000001\rangle \otimes |010001\rangle \otimes |000110\rangle \otimes |110100\rangle \otimes |100011\rangle \otimes |010111\rangle \\ & \otimes |100011\rangle \otimes |010001\rangle \otimes |0001000\rangle \otimes |0001000\rangle \otimes |0000110\rangle \otimes |0000100\rangle \\ & + |010011\rangle \otimes |000010\rangle \otimes |100101\rangle \otimes |110100\rangle \otimes |100011\rangle \otimes |010111\rangle \\ & \otimes |000101\rangle \otimes |011010\rangle \otimes |0000111\rangle \otimes |0010100\rangle \otimes |0001000\rangle \otimes |0000100\rangle) \end{aligned}$$

## 5 Determination of the Makespan

Now we have the system in state  $|\widehat{ST}\rangle$ , of which the last  $M(n+q)$  qubits provide the running time of all  $M$  machines under all schedules. The makespan of a schedule is equal to the maximum running time among the  $M$  machines under that schedule. We want to construct a Max circuit, as shown in Figure 5, to compute the makespan of each schedule. The quantum circuit computes the maximum over an input set, e.g.,  $\text{Max}(|5\rangle, |7\rangle, |4\rangle) = |7\rangle$ . The input to this circuit is a set of  $n+q$

qubits. The output  $|\text{Max}\rangle$ , has also  $n + q$  qubits. Implementing such arithmetic on a quantum computer has been addressed by many researchers[7, 5, 14] and we omit the detail circuit here.

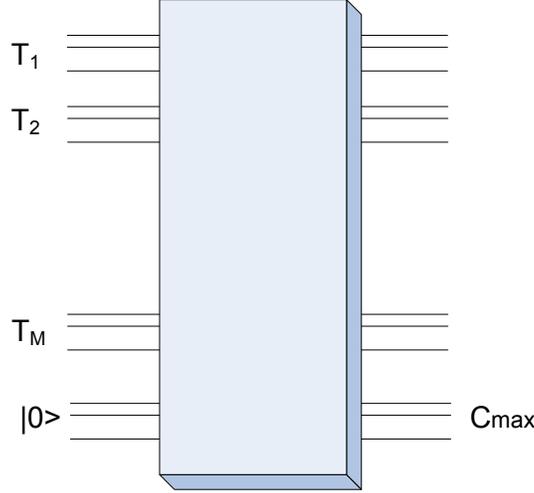


Figure 5: A quantum circuit to compute the makespan, the maximum running time among all machines. The output of this circuit is an entangled state.

The output of a quantum circuit as in Figure 5 is an entangled state, rather than the tensor product  $|T_1\rangle \otimes |T_2\rangle \otimes \dots \otimes |T_M\rangle \otimes |C_{max}\rangle$ . Recall that the  $|\mathcal{T}\rangle$  was prepared in an equal superposition of running times of the machines under all possible schedules. Thus, such a **Max** operation prepares the  $C_{max}$  register in an equal superposition of the makespans of all possible schedules.

Up to now, we discussed the implementation of the quantum circuits presented in Figure 1. During these operations, we successfully entangle the job vectors with the **Sum** and the **Makespan** vectors. These vectors are prepared in the equal superposition of all possible schedules, which can be written as:

$$|\widehat{\mathcal{STC}_{max}}\rangle = \sum_1^{2^{m2^n}} \mathcal{S}_i \mathcal{T}_i C_{maxi},$$

where,  $1 \leq i \leq 2^{m2^n}$  indexes different schedules, and we ignore the coefficients.

For the simple example in Section 3, the system will be in the state :

$$\begin{aligned} |\widehat{\mathcal{STC}_{max}}\rangle &= 1/\sqrt{2}(|\mathcal{S}_1\rangle |T_1\rangle |C_{max1}\rangle + |\mathcal{S}_2\rangle |T_2\rangle |C_{max2}\rangle) \\ &= 1/\sqrt{2}(|\underline{000001}\rangle \otimes |\underline{010001}\rangle \otimes |\underline{000110}\rangle \otimes |\underline{110100}\rangle \otimes |\underline{100011}\rangle \otimes |\underline{010111}\rangle \otimes |\underline{100011}\rangle \\ &\quad \otimes |\underline{010001}\rangle \otimes |\underline{0001000}\rangle \otimes |\underline{0001000}\rangle \otimes |\underline{0000110}\rangle \otimes |\underline{0000100}\rangle \otimes |\underline{0001000}\rangle \\ &\quad + |\underline{010011}\rangle \otimes |\underline{000010}\rangle \otimes |\underline{100101}\rangle \otimes |\underline{110100}\rangle \otimes |\underline{100011}\rangle \otimes |\underline{010111}\rangle \otimes |\underline{000101}\rangle \\ &\quad \otimes |\underline{011010}\rangle \otimes |\underline{0000111}\rangle \otimes |\underline{0010100}\rangle \otimes |\underline{0001000}\rangle \otimes |\underline{0000100}\rangle \otimes |\underline{0010100}\rangle) \end{aligned}$$

## 6 Searching for a Schedule with a Given Makespan

In our approach, a generalized version of Grover's search algorithm allows us to find a schedule with a given makespan. The basic ideas of Grover's quantum search algorithm are discussed next.

Consider a search space  $\mathcal{T}_{search} = \{E_x\}$  consisting of  $N = 2^n$  elements. Each element  $E_x, 1 \leq x \leq 2^n$ , is uniquely identified by a binary  $n$ -tuple  $x$ , called *the index* of the element. We assume that  $M \leq N$  elements satisfy the requirements of a query and we wish to identify one of them.

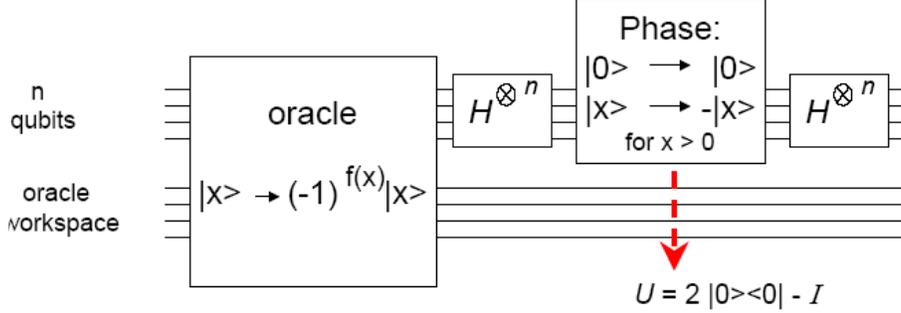


Figure 6: A quantum circuit for the Grover's iteration

The classic approach is to repeatedly select an element  $E_j$ , decide if the element is a solution to the query, and if so, terminate the search. If there is a single solution ( $M = 1$ ) then a classical exhaustive search algorithm requires  $\mathcal{O}(2^n)$  iterations.

For the Grover quantum searching algorithm, we apply a Walsh-Hadamard transform to create an equal superposition state which includes all elements of the search space:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle.$$

Then we perform Grover's iterations. The circuit for this algorithm is presented in Figure 6

An oracle examines an index/label,  $x$ , and decides if it matches the search argument or not. To abstract this process we consider a function  $f(x)$  with  $0 \leq x \leq 2^n - 1$  such that

$$f(x) = \begin{cases} 0 & \text{if } x \text{ is not a solution} \\ 1 & \text{if } x \text{ is a solution.} \end{cases}$$

An *oracle qubit*,  $|q\rangle$ , initially set to  $|0\rangle$  is reset to  $|1\rangle$  when the oracle recognizes a solution to the search problem we pose. The black box oracle  $O$  performs the following transformation

$$O |x\rangle |q\rangle = |x\rangle |q \oplus f(x)\rangle.$$

The oracle qubit can be initially in the state  $|q\rangle = (1/\sqrt{2})(|0\rangle - |1\rangle)$ .

Thus this transformation can be rewritten as

$$O |x\rangle (|0\rangle - |1\rangle)/\sqrt{2} = (-1)^{f(x)} |x\rangle (|0\rangle - |1\rangle)/\sqrt{2}.$$

The state of the oracle qubit does not change and can be omitted from the description of the quantum search algorithm

$$|x\rangle \mapsto (-1)^{f(x)} |x\rangle.$$

Let  $U$  be the following transformation:

$$U = 2|0\rangle\langle 0| - I.$$

Then a conditional phase shift in Figure 6 applied to the system is:

$$S_p = H^{\otimes n} U H^{\otimes n} = H^{\otimes n} (2|0\rangle\langle 0| - I) H^{\otimes n} = 2|\psi\rangle\langle\psi| - I.$$

A Grover's iteration consists of  $O$ , the transformation performed by the oracle followed by a conditional phase shift:

$$G = S_p O = (2 |\psi\rangle\langle\psi| - I) O$$

Thus, the quantum search algorithm could be written as:

$$G^R \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |q\rangle = [(2 |\psi\rangle\langle\psi| - I) O]^R \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle (|0\rangle - |1\rangle) / \sqrt{2} \approx |x_0\rangle (|0\rangle - |1\rangle) / \sqrt{2}$$

When after  $R = \mathcal{O}(\sqrt{\frac{N}{M}})$  iterations, we measure the first  $n$  qubits and obtain  $x_0$ , the solution to the search problem.

*Amplitude Amplification* represents a generalization of the Grover's quantum search idea [2, 11]. Let  $\mathcal{A}$  be a unitary operator in a Hilbert space,  $\mathcal{H}_N$  with an orthonormal basis  $|0\rangle, |1\rangle, \dots, |N-1\rangle$ ; the only condition imposed on  $\mathcal{A}$  is to be invertible, thus  $\mathcal{A}$  must not involve any measurements.

If  $\chi: \{0, 1, \dots, N-1\} \mapsto \{0, 1\}$  is a Boolean function we say that the basis state  $|x\rangle$  is a "Good" state if  $\chi(x) = 1$  and  $|x\rangle$  is a "Bad" state" if  $\chi(x) = 0$ . The central piece of the amplitude amplification is an operator  $\mathcal{Q}$  defined as:

$$\mathcal{Q} = \mathcal{Q}(\mathcal{A}, \chi, \phi, \varphi) = -\mathcal{A} S_0(\phi) \mathcal{A}^{-1} S_\chi(\varphi).$$

with  $\phi$  and  $\varphi$  two angles such that  $0 \leq \phi, \varphi \leq \pi$  and  $S_\chi$  an operator which conditionally changes the amplitudes of "Good" states:

$$|x\rangle \mapsto \begin{cases} e^{i\varphi} |x\rangle & \text{if } \chi(x) = 1 \\ |x\rangle & \text{if } \chi(x) = 0. \end{cases}$$

Similarly,  $S_0$  amplifies the amplitude by a factor  $e^{i\phi}$  if the state is not  $|0\rangle$ .

Let  $a$  denote the probability of finding a "Good" element  $x$ ; amplitude amplification allows to find a "Good"  $x$  after an expected number of applications of  $\mathcal{A}$  and of the inverse of  $\mathcal{A}$ ; the number of iterations is proportional to  $1/\sqrt{a}$ . We also define the angle  $\theta$  such that:

$$\sin^2(\theta) = a.$$

Grover's algorithm is a particular instance of amplitude amplification when the oracle implements the Boolean function  $f = \chi$ , and the transformation  $\mathcal{A}$  is the Walsh-Hadamard transform  $W = H^{\otimes n}$  on  $n$  qubits.

This iteration carried out by transformation  $\mathcal{Q}$  can be regarded as a *rotation* in the two-dimensional space spanned by the the state of a uniform superposition of non-solutions and the state consisting of a uniform superposition of solutions to the search problem. The initial state may be expressed as:

$$|\psi_0\rangle = \sqrt{a} |Good\rangle + \sqrt{1-a} |Bad\rangle$$

Figure 7 presents the effect of the transformation  $\mathcal{Q} = -\mathcal{A} S_0 \mathcal{A}^{-1} S_\chi$  as:

- the oracle operation  $S_\chi$  performs a reflection about the vector  $|Good\rangle$ .

$$S_\chi |x\rangle = |x\rangle \quad (\chi(x) = 1) \qquad S_\chi |x\rangle = -|x\rangle \quad (\chi(x) = 0)$$

- $\mathcal{A} S_0 \mathcal{A}^{-1}$  performs a reflection about the initial state  $|\psi_0\rangle$

$$S_0 |0\rangle = |0\rangle \qquad S_0 |x\rangle = -|x\rangle \quad (x \neq 0)$$

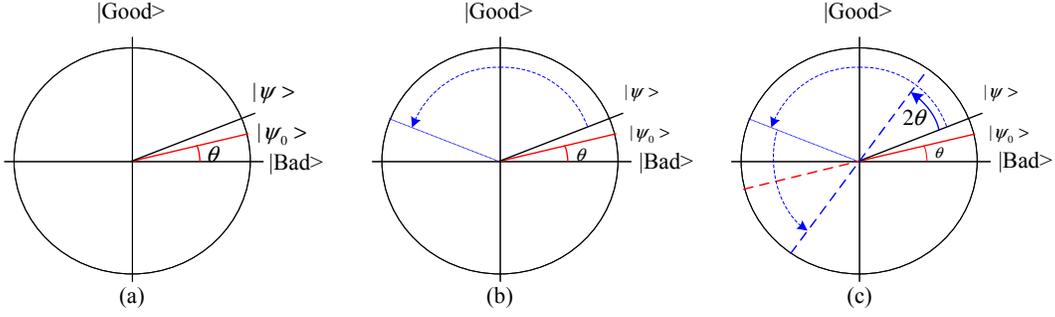


Figure 7: The search operator  $Q$  performs a rotation toward  $|Good\rangle$  states by  $2\theta$  radians.  $\theta$  is the angle between the initial state  $|\psi_0\rangle$  and the  $|Bad\rangle$  in the two-dimensional space spanned by the  $|Good\rangle$  and  $|Bad\rangle$  states and  $\sin^2(\theta) = a$  with  $a$  the probability of finding a “Good” element  $x$ . (a) The current state  $|\psi\rangle$  and the initial state  $|\psi_0\rangle$ . (b) The oracle operation  $S_\chi$  performs a reflection of the current state  $|\psi\rangle$  about the vector  $|Good\rangle$ . (c)  $\mathcal{A}S_0\mathcal{A}^{-1}$  performs a reflection about the initial state  $|\psi_0\rangle$

- $Q = -\mathcal{A}S_0\mathcal{A}^{-1}S_\chi$  performs a rotation toward  $|Good\rangle$  vector by  $2\theta$  radians, where  $\sin^2\theta = a$

Each iteration  $Q$  will rotate the system state by  $2\theta$  radians toward the solutions of the searching problem. Thus after  $m$  iterations, the measurement on the final state  $Q^m |\psi_0\rangle$  will produce a “Good” state with probability equal to  $\sin^2((2m+1)\theta)$ . The amplitude amplification algorithm could find a “Good” solution in  $\mathcal{O}(\frac{1}{\sqrt{a}})$  [2, 11]. Each iteration involves the application of  $\mathcal{A}$  and  $\mathcal{A}^{-1}$ .

Now let us return to our scheduling problem and recall that:

- We prepare each job vector  $|J_i\rangle$  in a superposition state which includes the running times on all machines. The first  $m$  qubits of a job vector are used for the index of the machine and remaining  $q$  qubits are used for the running time of that job on the machine.
- We summarize the execution time of all jobs according to their machine indexes, which produces all schedules and gives the running time of each machine  $T_j$  under these schedules. The system is prepared in an entangled state:

$$|\widehat{ST}\rangle = \frac{1}{\sqrt{\sigma}} \sum_{\text{each schedules } k} \left( \bigotimes_{i=1}^N |J_{ik}\rangle \bigotimes_{j=1}^M |T_{jk}\rangle \right) \quad \sigma = 2^{m2^n},$$

a superposition of job vectors and running time vectors of all possible schedules.

- We obtain the maximum running time among all machines using the **Max** quantum circuit and prepare the system in state:

$$|\widehat{STC_{max}}\rangle = \frac{1}{\sqrt{\sigma}} \sum_{\text{each schedules } k} \left( \bigotimes_{i=1}^N |J_{ik}\rangle \bigotimes_{j=1}^M |T_{jk}\rangle |C_{max} k\rangle \right) \quad \sigma = 2^{m2^n}$$

As we can see, our algorithm successfully prepares the system in an equal superposition of all  $2^{m2^n}$  possible schedules. We define this whole preparation process as  $\mathcal{Z}$ . This  $\mathcal{Z}$  transformation does not carry out a measurement of the system at any time. Therefore, there exists an inverse transformation operation  $\mathcal{Z}^{-1}$ . We can use the amplitude amplification algorithm to search the schedule with the makespan  $D_k = \mu$ . If we find such a makespan, the job vectors will be projected

as well. These projections will give us the actual mapping of jobs to machines corresponding to the schedule with the given makespan.

The searching process consists of the following steps:

- Apply the  $\mathcal{Z}$  transformation on  $|0\rangle$  to prepare the system in an equal superposition of all  $2^{m2^n}$  possible schedules,  $|\psi\rangle$ .

- Repeat the following steps  $\mathcal{O}(\sqrt{\sigma})$  times:

$$\text{Apply } Q \text{ on } |\psi\rangle, \text{ in which } Q = -\mathcal{Z}S_0\mathcal{Z}^{-1}S_x$$

- Measure the resulting state.
- Return the result; the job vectors give the detailed schedule.

The oracle in our searching algorithm exhibits some difference with the oracle in Grover's algorithm which checks all qubits to reverse the solution(s) of the searching problem. In our case, the oracle only checks a subset,  $C_{max}$  qubits. It is easy to implement such an oracle using an idea similar to the one in [17].

The algorithm presented in this paper can be optimized; to avoid over rotation we could use the fix-point quantum search [10].

## 7 Scheduling Problems with a Quantum Counterpart

Many other scheduling problems can be reformulated to take advantage of quantum search and exhibit a square root speedup versus their classical counterparts. Such problems require that a synthetic measure of performance,  $\mu$  be within a given range,  $\mu_{min} \leq \mu \leq \mu_{max}$ .

The generic quantum algorithm proceeds as follows:

1. Devise an encoding scheme for the information required to compute  $\mu_{S_i}$  for a given schedule  $S_i$ .
2. Design an algorithm to compute the synthetic measure of performance,  $\mu_{S_i}$ .
3. Construct  $|\mathcal{Q}\rangle$  with  $\mathcal{Q} = (\mu_{S_i}, S_i)$  in a superposition state for all possible schedules.
4. Design the quantum circuits for the specific encoding scheme and for the algorithms. Given a set of values  $\{q_1, q_2, \dots, q_n\}$  the circuit should be able to compute a function  $\mu_{S_i} = f(q_1, q_2, \dots, q_n)$ . For example, we may wish to compute the maximum, the minimum, or an average value.
5. Design an oracle to identify a specific value of the function  $\mu_{S_i}$ .
6. Use the quantum search to find if there is a value  $\mu_{S_i} = \mu_{min}$ . If so, determine the corresponding schedule  $S_i$ . Continue this process until  $\mu_{S_i} = \mu_{max}$ .
7. If no schedule can be found then report failure, otherwise provide the list of all schedules  $S_i$  and the corresponding measures of performance,  $\mu_{S_i}$ .

Consider for example the  $R|\sum_{i=1}^N C_i^S$  scheduling problem when the goal is to optimize the average completion time in unrelated parallel machine environment. It has the similar encoding process as the  $R|C_{max}$  problem. During the process we construct all schedules, we also summarize the completion time of different jobs using some simple arithmetic circuits, followed by Grover-type search. Other scheduling problems such as minimizing the average waiting time, could also take advantage of quantum search.

Oftentimes, we have to search for schedules that optimize the largest subset of a set of synthetic measures of performance,  $\mu, \nu, \pi, \rho, \theta \dots$ . For example, we could have multiple synthetic performance

indicators related to: timing, resource utilization, cost, and quality of service. In this case we would run repeatedly the scheduling algorithm for each performance measure and search for a solution in the given range for each measure. Once we have conducted all individual searches we determine the intersection of all schedules that satisfy all conditions; if the set is empty we remove individual conditions one by one until we find the a non-empty set.

Scheduling is also intimately related to planning when we have a complex goal and the sequence of actions to reach each goal have to be determined. The scenario described above is repeated for each plan thus the square root speedup of algorithms based upon quantum search becomes even more important.

## 8 Summary

When a deadline is imposed, or when we wish to find a schedule with a given range of possible average completion time we discover that a full range of scheduling problems have a quantum counterpart which can take advantage of Grover's search.

Many scheduling problems, resource allocations, and path-finding problems, share some common properties with the  $R||C_{max}$  problem discussed in this paper: a well-defined initial state, a well-defined desired state or a range of desired states, many paths to reach the desired state, and well-defined attributes of an optimal path.

The quantum algorithmic solution to such problems requires the following steps:

- Prepare the initial state in an equal superposition of all possible choices.
- Use some reversible quantum arithmetic to compute the specialized property (makespan in our case) needed.
- Construct the necessary oracle circuit.
- Use Grover-type algorithms to search for the desired solution.

The solution we propose based upon Grover's algorithm is not universally applicable. Problem requiring specific optimization criteria may require quantum circuits that cannot be easily implemented with reversible quantum arithmetic. Moreover, Grover-type algorithms lead to a square-root improvement over the exhausting search, while many good classical algorithms may have better performance for some special problems.

## 9 Acknowledgments

The research reported in this paper was partially supported by National Science Foundation grant CCF 0523603. The authors express their thanks to Lov Grover, Pawel Wocjan, and an anonymous reviewer for their comments and suggestions.

## References

- [1] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and Weaknesses of Quantum Computation*. SIAM Journal on Computing, 26: 1510–1523, 1997
- [2] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, *Quantum Amplitude Amplification and Estimation*. <http://www.arxiv.org/quant-ph/0005055>, 2000
- [3] Peter Brucker, Sigrid Knust *Complexity Results for Scheduling Problems*. <http://www.mathematik.uni-osnabrueck.de/research/OR/class/>, 2006

- [4] J. Carlier and E. Pinson, *An Algorithm for Solving the Job-shop Problem* Management Sci., 35: 164–176, 1989
- [5] T. G. Draper, S. A. Kutin, E. M. Rains and K. M. Svore, *A Logarithmic-depth Quantum Carry-lookahead Adder*. <http://www.arXiv.org/quant-ph/0406142> 2004.
- [6] Bartholomew Furrow, *A Panoply of Quantum Algorithms* <http://www.arxiv.org/abs/quant-ph/0606127>, 2006
- [7] P. Gossett, *Quantum Carry-save Arithmetic*. <http://www.arXiv.org/quant-ph/9808061>, 1998
- [8] A. Grigoriev, M. Sviridenko and M. Uetz *Unrelated Parallel Machine Scheduling with Resource Dependent Processing Times*, In proceedings of the 11th Conference on Integer Programming and Combinatorial Optimization IPCO 2005, LNCS 3509: 182–195, 2005
- [9] L. K. Grover, *A Fast Quantum Algorithm for Database Search*. Proc. ACM Symp. on Theory of Computing, ACM Press, 212–219, 1996.
- [10] L. K. Grover, *Fixed-Point Quantum Search*. Physical Review Letters, 95(15): 150501, 2005
- [11] P. Hoyer, *On Arbitrary Phases in Quantum Amplitude Amplification* Physical Review A, 62:052304, 2000
- [12] A. Yu. Kitaev, *Quantum Measurements and the Abelian Stabilizer Problem*. <http://www.arxiv.org/abs/quant-ph/9511026> v1, 1995.
- [13] J. K. Lenstra, D. B. Shmoys and E. Tardos, *Approximation Algorithms for Scheduling Unrelated Parallel Machines*, Mathematical Programming, Series A 46: 259–271, 1990
- [14] R. Van Meter and K. M. Itoh, *Fast Quantum Modular Exponentiation*. Physical Review A, 71: 052320, 2005
- [15] D. C. Marinescu and G. M. Marinescu, *Approaching Quantum Computing*. Prentice Hall, Upper Saddle River, NJ, ISBN 0-13-145224-X, 2004.
- [16] M. Mosca and A. Ekert, *The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer*. <http://www.arxiv.org/abs/quant-ph/9903071> v1, May 1999.
- [17] M. A. Nielsen and I. L. Chuang, *Quantum Computing and Quantum Information*. Cambridge University Press, ISBN 0-521-63245-8, 2000.
- [18] C. M. Papadimitriou, *Computational Complexity*. Addison-Wesley, Reading, Mass., ISBN 0-2015-3082-1, 1994.
- [19] A. S. Schulz, M. Skutella, *Scheduling Unrelated Machines by Randomized Rounding*, SIAM Journal on Discrete Mathematics 15: 450–469, 2002
- [20] P. W. Shor, *Algorithms for Quantum Computation: Discrete Log and Factoring*. Proc. 35 Annual Symp. on Foundations of Computer Science, Piscataway, New Jersey, IEEE Press, 124–134, 1994.
- [21] C. Zalka, *Grover's Quantum Searching Algorithm is Optimal*. Phys. Rev. A 60 2746-2751. <http://www.arxiv.org/abs/quant-ph/9711070> v2, Dec 1999.