A Novel Ant Colony Optimization Strategy for the Quantum Circuit Compilation Problem

Marco Baioletti, Riccardo Rasconi and Angelo Oddi

April 18, 2021

A Novel Ant Colony Optimization Strategy for the Quantum Circuit Compilation Problem

Marco Baioletti[1][0000−0001−5630−7173], Riccardo Rasconi[2][0000−0003−2420−4713], and Angelo Oddi[2][0000−0003−4370−7156]

[1] University of Perugia, Perugia, Italy
`marco.baioletti@unipg.it`
[2] Institute of Cognitive Sciences and Technologies (ISTC-CNR), Rome, Italy
{*name.surname*}`@istc.cnr.it`

Abstract. Quantum Computing represents the most promising technology towards speed boost in computation, opening the possibility of major breakthroughs in several disciplines including Artificial Intelligence. This paper investigates the performance of a novel Ant Colony Optimization (ACO) algorithm for the realization (compilation) of nearest-neighbor compliant quantum circuits of minimum duration. In fact, current technological limitations (e.g., decoherence effect) impose that the overall duration (makespan) of the quantum circuit realization be minimized, and therefore the production of minimum-makespan compiled circuits for present and future quantum machines is of paramount importance. In our ACO algorithm (QCC-ACO), we introduce a novel pheromone model, and we leverage a heuristic-based Priority Rule to control the iterative selection of the quantum gates to be inserted in the solution. The proposed QCC-ACO algorithm has been tested on a set of quantum circuit benchmark instances of increasing sizes available from the recent literature. We demonstrate that the QCC-ACO obtains results that outperform the current best solutions in the literature against the same benchmark, succeeding in significantly improving the makespan values for a great number of instances and demonstrating the scalability of the approach.

Keywords: Ant Colony Optimization · Quantum Circuit Compilation · Planning · Scheduling.

1 Introduction

Quantum Computing explores the implications of using quantum mechanics to model information and its processing. The impact of quantum computing technology on theoretical/applicative aspects of computation as well as on the society in the next decades is considered to be immensely beneficial [16]. While classical computing revolves around the execution of logical gates based on two-valued *bits*, quantum computing uses *quantum gates* that manipulate multi-valued bits (*qubits*) that can represent as many logical states (*qstates*) as are the obtainable linear combinations of a set of basis states (state *superpositions*). A quantum circuit is composed of a number of qubits and by a series of quantum gates that operate on those qubits, and whose execution realizes a specific quantum algorithm.

Executing a quantum circuit entails the chronological evaluation of each gate and the modification of the involved qstates according to the gate logic. Current quantum computing technologies like ion-traps, quantum dots, super-conducting qubits, etc. limit the qubit interaction distance to the extent of allowing the execution of gates between adjacent (i.e., *nearest-neighbor*) qubits only [6, 13, 23]. This has opened the way to the exploration of possible techniques and/or heuristics aimed at guaranteeing nearest-neighbor (NN) compliance in any quantum circuit through the addition of a number of so-called *swap* gates between adjacent qubits. The effect of a swap gate is to mutually exchange the qstates of the involved qubits, thus allowing the execution of the gates that require those qstates to rest on adjacent qubits. However, adding swap gates also introduces a time overhead in the circuit execution [4] that generally depends on the quantum hardware's topology; on the other hand, it is highly desirable to minimize the circuit's execution time (i.e., *makespan*), in order to mitigate the negative effects of *decoherence* and guarantee more stability to the quantum computation. The Quantum Circuit Compilation Problem (QCCP) can be described as the synthesis of a real quantum circuit to be executed on a specific quantum hardware.

The QCCP benchmarks used in this work[3] have been initially introduced and solved in [22] as a temporal planning problem. Subsequently, the same benchmark was tackled in [3, 17], respectively through a hybrid approach that integrates Temporal Planning with Constraint Programming, and a heuristically-based Greedy Randomized Search (GRS) technique [12, 19]. The results obtained in [17] have been further improved in [18] by means of a genetic algorithm. More recently, a similar version of the priority rule introduced in [17] has been used in [5] within a *rollout* procedure to further improve the best results, thus representing the benchmark's current bests.

In this work, we use a slightly modified version of the priority rule proposed in [5] within an Ant Colony Optimization (ACO) algorithm [9], and compare our results with those obtained in the same paper, significantly improving the makespan values for a considerable number of instances, and demonstrating the scalability of the approach.

ACO is a powerful metaheuristic, inspired by the foraging behaviour of colonies of ants that has been applied to many combinatorial optimization problems [9], and particularly to scheduling and permutation problems [14, 2]; indeed, the QCC problem tackled here has in fact a strong scheduling component. Among the evolutionary and swarm intelligence based algorithms, ACO seems to be well suited for the QCC problem because of its constructive nature. In fact, feasible

[3] A set of benchmark instances of different size belonging to the *Quantum Approximate Optimization Algorithm* (QAOA) class [10, 11] tailored for the MaxCut problem and devised to be executed on top of a hardware architecture proposed by Rigetti Computing Inc. [20].

solutions in ACO are built by means of an iterative process that starts from an empty solution and adds a component at a time consistently with the problem constraints, hence maintaining the feasibility of the solution at all times during the building process.

The paper is organized as follows. The next section proposes a formal statement of the tackled problem, whereas the subsequent section describes the novel ant colony optimization strategy proposed for its resolution. Finally, an empirical evaluation based on the benchmark proposed in [22] is performed, and some conclusions end the paper.

2 The QCC Problem

Formally, the Quantum Circuit Compilation Problem (QCCP) [17] is a tuple $P = \langle C_0, L_0, QM \rangle$, where: (i) C_0 is the input quantum circuit representing the execution of the algorithm of interest, (ii) L_0 is the initial assignment of qubits to qstates, and (iii) QM is a representation of the quantum hardware.

- C_0 is the input quantum circuit expressed as a tuple $\langle Q, P\text{-}S, MIX, P, TC_0 \rangle$, where: (i) $Q = \{q_1, q_2, \ldots, q_N\}$ is the set of qstates which, from a planning & scheduling perspective represent the *resources* necessary for each gate's execution (see for example [15], Chapter 15); (ii) $P\text{-}S$ and MIX respectively represent the set of *p-s* and *mix* gates that have to be scheduled, such that every *p-s*(q_i, q_j) gate requires two qstates for execution, and every *mix*(q_i) gate requires one qstate only; (iii) $P = \{1, ..., p\}$ is the number of times (i.e., *passes*) the gates in the $P\text{-}S$ and MIX must be executed; (iv) TC_0 is a set of simple precedence constraints imposed on the $P\text{-}S$ and MIX such that: (1) all the *p-s* gates belonging to the k-th pass ($P\text{-}S_k$) that involve a specific qstate q_i must be executed before all the *mix* gates belonging to the same pass (MIX_k) that involve the same qstate q_i, for $k = 1, 2, \ldots, p$, and (2) all the *mix* gates belonging to the k-th pass (MIX_k) that involve a specific qstate q_i must be executed before all the *p-s* gates belonging to the $(k+1)$-th pass ($P\text{-}S_{k+1}$) that involve the same qstate q_i, for $k = 1, 2, \ldots, (p-1)$. Lastly, the execution of every quantum gate requires the uninterrupted use of the involved qstates during its processing time, and each qstate q_i can process at most one quantum gate at a time.
- L_0 is the initial assignment at the time origin $t = 0$ of qstates q_i to qubits n_i. In this work, the i-th qstate q_i is assigned to the i-th qubit n_i.
- QM is a representation of the quantum hardware as an undirected multigraph $QM = \langle V_N, E_{p\text{-}s}, E_{swap}, \tau_{mix}, \tau_{p\text{-}s}, \tau_{swap} \rangle$, where $V_N = \{n_1, n_2, \ldots, n_N\}$ is the set of qubits (nodes), $E_{p\text{-}s}$ (E_{swap}) is a set of undirected edges (n_i, n_j) representing the set of *adjacent* locations the qstates q_i and q_j of the gates *p-s*(q_i, q_j) (*swap*(q_i, q_j)) can potentially be allocated to. In addition, the *labelling* functions $\tau_{p\text{-}s} : E_{p\text{-}s} \rightarrow \mathbb{Z}^+$ and $\tau_{swap} : E_{swap} \rightarrow \mathbb{Z}^+$ respectively represent the durations of the gate operations *p-s*(q_i, q_j) and *swap*(q_i, q_j) when the qstates q_i and q_j are assigned to the corresponding adjacent locations. Similarly, the *labelling* function $\tau_{mix} : V \rightarrow \mathbb{Z}^+$ represents the durations of

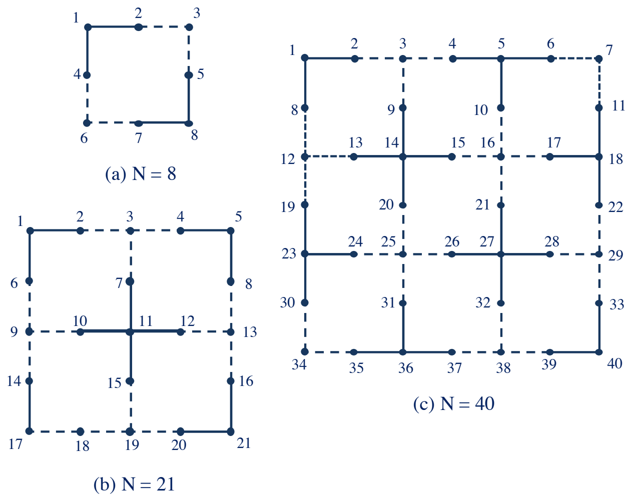

Fig. 1. Three quantum chip designs characterized by an increasing number of qubits ($N = 8, 21, 40$) inspired by Rigetti Computing Inc. Every qubit is located at a different location (node), and the integers at each node represent the qubit's identifier. Two qubits connected by an edge are adjacent, and each edge represents a 2-qubit gate (*p-s* or *swap*) that can be executed between those qubits. *p-s* gates executed on continuous edges have duration $\tau_{p\text{-}s} = 3$, while *p-s* gates executed on dashed edges have duration $\tau_{p\text{-}s} = 4$. Swap gates have duration $\tau_{swap} = 2$.

the *mix* gate (which can be executed at any node n_i). Figure 1 shows an example of quantum hardware designs, with gate durations.

A feasible solution is a tuple $S = \langle SWAP, TC \rangle$, which extends the initial circuit C_0 to a circuit $C_S = \langle Q, P\text{-}S, MIX, SWAP, P, TC_S \rangle$, such that $TC_S = TC_0 \cup TC$, where $SWAP$ is a set of additional *swap*(q_i, q_j) gates added to guarantee the adjacency constraints for the set of $P\text{-}S$ gates, and TC is a set of additional simple precedence constraints such that: (i) for each qstate q_i, a total order $\preceq_i$ is imposed among the set Q_i of operations requiring q_i, with $Q_i = \{op \in P\text{-}S \cup MIX \cup SWAP : op\ requires\, q_i\}$; (ii) all the *p-s*$(q_i, q_j)$ and *swap*(q_i, q_j) gate operations are allocated on adjacent qubits in QM, and (iii) the graph $\langle \{P\text{-}S \cup MIX \cup SWAP\}, TC_S \rangle$ does not contain cycles.

Given a solution S, the makespan $mk(S)$ corresponds to the maximum completion time of the gate operations in S. An optimal solution S^* is a feasible solution characterized by the minimum makespan.

3 A Novel Ant Colony Optimization Strategy

The solution pursued in this work to solve the QCCP exploits the Ant Colony Optimization (ACO) paradigm [9]. The proposed algorithm is called QCC-ACO

Algorithm 1 QCC-ACO

```
Require: A problem $P = \langle C_0, L_0, QM \rangle$
  InitializePheromoneValues()
  while not termination criterion do
    for $i \leftarrow 1$ to $n_a$ do
      BuildSolution(P)
    end for
    UpdatePheromoneValues()
  end while
  return BestSolution
```

Algorithm 2 BuildSolution

```
Require: A problem $P = \langle C_0, L_0, QM \rangle$
  $S \leftarrow$ InitSolution($P$)
  $t \leftarrow 0$
  while not all the $P$-$S$ and $MIX$ operations are inserted in $S$ do
    $op \leftarrow$ SelectExecutableOperation($P$, $S$, $t$)
    if $op \neq NULL$ then
      $S \leftarrow$ InsertOperation($op$, $S$, $t$)
    else
      $t \leftarrow t + 1$
    end if
  end while
  return $S$
```

and its main schema is depicted in Algorithm 1. The algorithm handles a colony of n_a artificial ants, which indirectly communicate through the *pheromone* model and create solutions of the QCC problem. The main loop is repeated for a certain number of times, according to a given stop criterion, for instance, a computation time budget. At each iteration, every artificial ant builds a solution by means of a constructive procedure (BuildSolution()); then, pheromone values are updated (UpdatePheromoneValues()) in order to select the best solutions.

In the next two subsections some details about the solution constructive procedure and the pheromone update strategy will be respectively provided.

3.1 Solution Construction Algorithm

The solution construction procedure used in QCC-ACO is described in Algorithm 2. The algorithm produces a complete solution S for the given QCC problem P by starting from an empty solution and by iteratively selecting (SelectExecutableOperation) and adding (InsertOperation) one operation (i.e., a quantum gate) at a time among those that can scheduled at the current instant t.

An operation *op* can be scheduled at t if the qstates required by *op* are not used in t. If no operation can be scheduled at t, the time is increased to $t + 1$.

Clearly, one essential step of the BuildSolution() procedure is the SelectExecutableOperation() method (shown in Algorithm 3) through which a new operation is selected for insertion in the partial solution, at each iteration. This method is based on a heuristic-based priority rule originally introduced in [17] and slightly modified in [5], whose technical details will be described in the following sections.

3.2 Gate Selection Procedure based on Priority Rules

The priority rule used in our QCC-ACO exploits the distance between qstate pairs measured on the quantum hardware; in the following, we describe in detail the criteria upon which such distance is assessed.

Let $op \in Q_i$ be a general gate operation that involves qstate q_i, we define a *chain* $ch_i = \{op \in Q_i : op \in S\}$ as the set of gates involving q_i and currently present in the partial solution S, among which a total order is imposed (see Figure 2 for a graphical representation of a complete solution composed of a set of chains, one for each qstate q_i).

Let us also define $last(ch_i)$ as the last operation in the chain ch_i according to the imposed total order and $n(last(ch_i))$ as the QM node at which the last operation in the chain ch_i terminates its execution. Given a partial solution S, the *state* L_S is the tuple $L_S = \langle n(last(ch_1)), n(last(ch_2)), \ldots, n(last(ch_N))\rangle$ of QM locations (nodes) where each last chain operation $last(ch_i)$ terminates its execution.

Given the multi-graph QM introduced in the QCC Problem section, we consider the distance graph $G_d(V, E_{p\text{-}s})$, so as to contain an undirected edge $(n_i, n_j) \in E_{p\text{-}s}$ when QM can execute a p-s gate on the pair (n_i, n_j). In the graph G_d, an undirected path p_{ij} between a node n_i and a node n_j is the list of edges $p_{ij} = ((n_i, n_{j1}), (n_{j1}, n_{j2}), \ldots, (n_{jk}, n_j))$ connecting the two nodes n_i and n_j and its length l_{ij} is the number of edges in the path p_{ij}. Let d_{ij} represent the minimal length among the set of all the paths between n_i and n_j. The distance d_{ij} between all nodes is computed only once at the beginning, by means of all-pairs shortest path algorithm, whose complexity is $O(|V|^3)$ in the worst case [7]. The distance d^{L_S} associated to a given p-$s(q_i, q_j)$ gate that requires two qstates q_i and q_j w.r.t. the state L_S of the partial solution S is defined as:

$$d^{L_S}(p\text{-}s(q_i, q_j)) = d(nlast(ch_i), nlast(ch_j)) \tag{1}$$

Two qstates q_i and q_j are in adjacent locations in the state L^S if $d^{L_S}(p\text{-}s(q_i, q_j)) = 1$. Intuitively, given a p-$s(q_i, q_j)$ gate and a partial solution S, the value $d^{L_S}(p\text{-}s(q_i, q_j))$ yields the minimal number of swaps for moving the two qstates q_i and q_j to adjacent locations on the machine QM.

The concept of distance defined on a single gate operation p-$s(q_i, q_j)$ can be extended to a set of gate operations, as follows. Let S be a partial solution, $\overline{P\text{-}S}^S$ is the set of p-$s(q_i, q_j)$ gates that are not yet scheduled in S and such that all predecessors gates according to the temporal order imposed by the set TC_0 (the set of simple precedences in the input circuit C_0) have already been scheduled in

S. The authors in [17] proposed two different functions to measure the distance separating the set $\overline{P\text{-}S}^S$ from the *adjacent state*. The first sums the set of the distances $d^{L_S}(p\text{-}s(q_i, q_j))$:

$$D^S_{sum}(\overline{P\text{-}S}^S) = \sum_{p\text{-}s \in \overline{P\text{-}S}^S} d^{L_S}(p\text{-}s(q_i, q_j)) \tag{2}$$

The second returns the minimal value of the distance $d^{L_S}(p\text{-}s(q_i, q_j))$ in the set $\overline{P\text{-}S}^S$:

$$D^S_{min}(\overline{P\text{-}S}^S) = MIN_{p\text{-}s \in \overline{P\text{-}S}^S} d^{L_S}(p\text{-}s(q_i, q_j)) \tag{3}$$

Given the functions (2) and (3), it is now possible to assess the priority of each gate operation *op* to possibly insert in the partial solution as follows:

$$f(S, op, \overline{P\text{-}S}^S) = \begin{cases} (D^{S'}_{sum}(\overline{P\text{-}S}^S \setminus \{op\}), 1) & \text{(p-s)} \\ (D^{S'}_{sum}(\overline{P\text{-}S}^{S'}), 1) & \text{(mix)} \\ (D^{S'}_{sum}(\overline{P\text{-}S}^{S'}), D^{S'}_{min}(\overline{P\text{-}S}^{S'})) & \text{(swap)} \end{cases} \tag{4}$$

where S' is the new partial solution after the addition of the selected gate operation *op*. We are now in the position to describe in detail the SELECTEXECUTABLEOPERATION() procedure, which operates over three phases (see Algorithm 3).

In the first phase (*Phase1*), a set Ω' of operations (PS, MIX or SWAP) that can be time and resource feasibly scheduled at the current instant t is selected through the ELIGIBLESET() procedure (*eligible* operations at time t). Subsequently, the values of D^S_{sum} and D^S_{min} of the solution S (baseline values), are computed using formulas 2 and 3, respectively. From this point, a new set Ω of operations is built by further restricting Ω', in a fashion inspired to the Priority Rule described in [5]. Namely, Ω is built by immediately inserting all the eligible P-S and MIX operations previously stored in Ω' (if any), while the SWAP operations will be considered only if their execution produces a partial solution which is better than the current solution with respect to the D_{sum} heuristic value or, being equal with respect to D_{sum}, it is better with respect to D_{min}.

The second phase (*Phase2*) is executed only if the first phase returns an empty Ω set, in which case the algorithm collects all the SWAP operations previously contained in Ω' whose execution produces a partial solution which is better than the current solution with respect to D_{min} only.

The third phase (*Phase3*) chooses the operation to be returned by the procedure. If Ω is still empty, SELECTEXECUTABLEOPERATION returns NULL and the solution construction scheme continues by increasing the current value of t. Otherwise, a selection probability value $prob(op)$ is firstly computed for each operation contained in Ω, according to the following formula:

$$prob(op) = \frac{\tau(op)^\alpha \eta(op)^\beta}{\sum_{op' \in \Omega} \tau(op')^\alpha \eta(op')^\beta} \tag{5}$$

Algorithm 3 SELECTEXECUTABLEOPERATION

```
Require: a problem $P$, a partial solution $S$, a time $t$
  //Phase1:
  $\Omega' \leftarrow$ ELIGIBLESET$(S, t)$
  $\Omega \leftarrow \emptyset$
  Init $D^S_{sum}$ and $D^S_{min}$) resp. according to (2) and (3)
  for all $op \in \Omega'$ do
    $(op.D^S_{sum}, op.D^S_{min}) \leftarrow f(S, op, \overline{P\text{-}S}^S)$
    if ($op$ is a MIX or a PS) or ($op.D^S_{sum} < D^S_{sum}$) or ($op.D^S_{sum} = D^S_{sum}$ and
    $op.D^S_{min} < D^S_{min}$) then
      $\Omega \leftarrow \Omega \cup \{op\}$
    end if
  end for
  //Phase2:
  if $\Omega = \emptyset$ then
    for all $op \in \Omega'$ do
      if ($op$ is a SWAP) and ($op.D^S_{min} < D^S_{min}$) then
        $\Omega \leftarrow \Omega \cup \{op\}$
      end if
    end for
  end if
  //Phase3: op selection
  if $\Omega = \emptyset$ then
    return NULL
  end if
  Evaluate $prob(op)$ for all $op \in \Omega$ according to (5)
  return $op$ chosen at random with probability $prob(op)$
```

where $\tau(op)$ is the pheromone value associated to the choice of op, $\eta(op)$ is the *desirability* value of op, and the parameters α and β regulate the contribution of the pheromone and the desirability values to the probability of component selection.

The pheromone values will be described in the Pheromone Models section, while the desirability value for an operation op is computed as:

$$\eta(op) = 1 - \frac{W \times op.\hat{D}^S{}_{sum} + op.\hat{D}^S{}_{min}}{W + 1}$$

where $op.\hat{D}^S{}_{sum}$ and $op.\hat{D}^S{}_{min}$ are the normalized values of $op.D^S_{sum}$ of $op.D^S_{min}$, respectively, and W is a constant which enhances the contribution of D^S_{sum} with respect to D^S_{min}.

Finally, the SELECTEXECUTABLEOPERATION() procedure selects an operation according to the probability distribution $prob(op)$.

3.3 Pheromone Models

Pheromone values could be associated directly to the operations, however this organization does not work well because P-S and MIX operations are present in all the feasible solutions (hence their pheromone value would be not significant), while SWAP operations can be used more times in the same solution. Therefore, it is necessary to associate a pheromone value to contextualized operations, i.e. to pairs (op, c), where c is a piece of information, denoting the context where op is executed.

Using a technique similar to what is done in ACO approaches to scheduling [14] and to planning [1], pheromone values can be associated to the pairs (op, t), where op is the operation and t is the start time of op. Pheromone values are organized as a matrix and the corresponding model is called Time Operation (TO) model. Hence, in this model the (possible) start time of the operations is taken into account both in the operation selection phase (equation 5), where $\tau(op)$ is replaced with $\tau(op, t)$.

The pheromone values are updated with the following two steps procedure.

Firstly, an evaporation phase is performed, which lowers the value associated to each operation op and each time step t with the formula

$$\tau(op, t) \leftarrow (1 - \rho)\tau(op, t) \tag{6}$$

where $\rho \in (0, 1]$ is a parameter called *evaporation rate*.

Secondly, the values associated to the best solutions are increased. In QCC-ACO we decided to reward the best solution found so far (S_{bs}) or the best solution found in the current iteration (S_{ib}); the choice of which solution should be rewarded is made at random, the probability of rewarding S_{ib} is 0.8, while S_{bs} is rewarded with probability 0.2.

Let S^* the solution to reward, the pheromone value of all the operations $op \in S^*$ is increased with the formula

$$\tau(op, t) \leftarrow \tau(op, t) + \frac{L}{mk(S^*)} \tag{7}$$

where t is the time where op is executed in S^* and L is constant (in the experiments its value is fixed to 10).

Another pheromone model, based on the TO model, is the Fuzzy Time Operation (FTO) model. In this new model the value of pheromone of a given operation op is spread around the time steps in the operation selection phase by fuzzyfing the time step t when the pheromone model is queried. Internally, the pheromone values are stored and updated in a matrix $\tau(op, t)$ as in the TO model. However, in the the operation selection phase, instead of employing $\tau(op, t)$, the value $\tau_w(op, t)$ is used in (5). This value is computed as the weighted average of the pheromone values of op at times close to t: $t-w, t-w+1, \ldots, t+w-1, t+w$.

More in detail, $\tau_w(op, t)$ is computed with the following formula:

$$\tau_w(op, t) = \sum_{h=-w}^{w} \sigma(h)\tau(op, t + h)$$

where w is the width of the window and

$$\sigma(h) = \frac{\exp(-\frac{h^2}{2})}{\sum_{k=-w}^{w} \exp(-\frac{k^2}{2})}$$

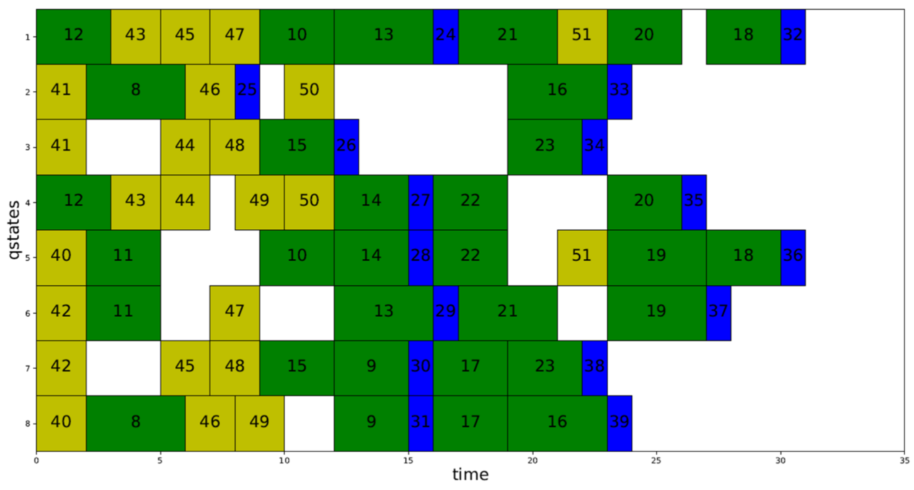

Fig. 2. Solution of the instance $n.8$ of the $N = 8$ benchmark set, with makespan $mk = 31$. In the plot, PS gates are depicted in green, SWAP gates in yellow, and MIX gates in blue.

is the weight for the displacement $h = -w, \ldots, w$. The width w is a parameter of the model FTO, hence we will denote by FTO_w the FTO model where the width is w.

The difference between TO and FTO can be summarised as follows. Suppose that the operation op is rewarded as a component of a good solution S^* and that op was executed at time t_0 in S^*. In the TO model the pheromone value is only affected at t_0, while in FTO also the time steps near t_0 can exploit of the value. The strength of influence of the pheromone value at time t depends on the difference $|t - t_0|$: the smaller the distance, the higher the influence. A similar way of using pheromone has already been introduced in [1] (called Fuzzy Level Action model).

4 Empirical Evaluation

The QCC-ACO algorithm has been implemented in Java standard (OpenJDK, v.11.0.9.1), without using any external library. In particular, we have used the Java built-in pseudo random generator. All the experiments were run on an Intel Xeon E312 machine equipped with 16 GB of RAM.

4.1 Tuning

QCC-ACO has many parameters to be set: α, β, ρ, the model $\mathcal{M}$ of the pheromone (the choice is among TO and FTO_w, for some reasonable values of w), the number of ants n_a, W, and L. Some of the parameters, namely n_a, W, and

L were chosen after some preliminary runs, resulting that QCC-ACO is not affected in a sensitive way by the values of those parameters. The choice is $n_a = 20, W = 10, L = 10$.

Hence, we have decided to perform an extensive tuning procedure to find the best configuration for QCC-ACO in terms of α, β, ρ and $\mathcal{M}$. We have created 5 new instances of the QCC problem for each value of $N = 8, 21, 40$ only for the tuning phase, in order to avoid to use the same instances for the tuning and for the test phases. Both the parameters α and β have been varied in the set $\{0, 1, 2, 3, 4\}$, while the possible values of ρ were $0.1, 0.2, 0.3$. Finally, TO competed against $FTO_1, \ldots, FTO_5$.

For each parameter configuration, QCC-ACO was run 10 times on the 15 tuning instances, using as termination criterion the time budget of 60 seconds for the instances with $N = 8$ and 300 seconds for the other instances.

The results of the tuning phase clearly indicates that the combinations ($\alpha = 1, \beta = 0, \rho = 0.3, \mathcal{M} = FTO3$) outperformed the others in terms of the average relative percentage difference. In line with the results obtained in [21], the best value for β resulted to be 0, i.e. QCC-ACO works better without using the heuristic function η in the probabilistic operation selection (formula 5). However, the heuristic function still plays an important role because it is used to prune the operations to select (see Algorithm 3) and therefore to reduce the branching factor.

Some preliminary experiments showed that the pruning stage is important because the performances of QCC-ACO greatly deteriorate if the operation selection works with $\Omega = \Omega'$, i.e. if all the eligible operations at the given time t can be selected.

A second, smaller set of tuning experiments have been conducted in order to see if some small value of β can improve the QCC-ACO performances. The parameter β have been varied in the set $\{0, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}\}$, while the other parameters have been fixed to the value of the best configuration. The second experiment confirmed that $\beta = 0$ is the best choice. Among the pheromone values, all the fuzzified versions of TO outperformed the crisp model TO, confirming that the fuzzification of TO works better. In particular, the best value for the width w is 3, which is an intermediate value among all the possible value for that parameter. Hence, we decided to adopt in the test phase the configuration ($\alpha = 1, \beta = 0, \rho = 0.3, \mathcal{M} = FTO3$).

4.2 Results

QCC-ACO has been tested on the 150 instances of the QCC problem (50 instances for each size $N = 8, 21, 40$) with $u = 1.0$ and $P = 2$. The termination criterion used in our experiments is the same that has been used in [5]: all runs for the $N = 8$ instances were limited to 60 seconds, while for the $N = 21$ and $N = 40$ instances all runs were limited to 300 seconds. For each instance QCC-ACO has been executed 10 times.

The results are depicted in Table 1, where the average value of the makespan, the standard deviation and the best value obtained by QCC-ACO are listed. The

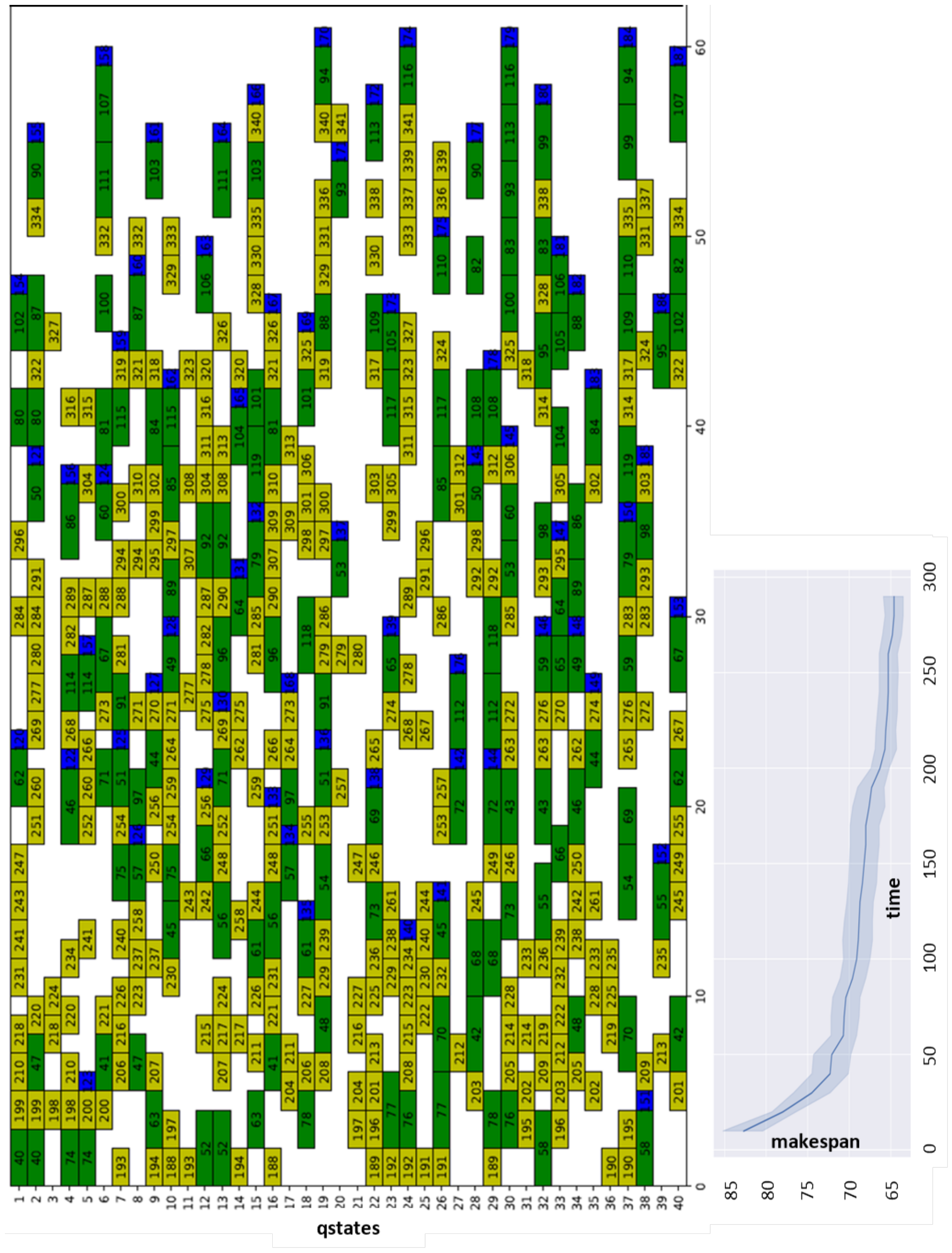

Fig. 3. Left: solution of the instance $n.24$ of the $N = 40$ benchmark set, with makespan $mk = 61$. Right: average plot of convergence times on 10 runs, the shaded area shows 95% confidence interval.

Table 1. Experimental results for all benchmarks

	$N = 8$					$N = 21$					$N = 40$				
Inst	Avg. ACO	SD ACO	Best ACO	Best RH	Δ	Avg. ACO	SD ACO	Best ACO	Best RH	Δ	Avg. ACO	SD ACO	Best ACO	Best RH	Δ
1	35.0	0.0	35	35	0	48.2	0.4	**48**	49	**-1**	61.6	1.9	**58**	65	**-7**
2	34.0	0.0	**34**	36	**-2**	50.3	0.5	50	50	0	65.8	2.3	**62**	74	**-12**
3	32.0	0.0	32	31	1	44.1	1.7	**41**	42	**-1**	63.6	1.6	**61**	71	**-10**
4	33.0	0.0	33	32	1	43.2	0.6	**42**	44	**-2**	69.1	3.2	**65**	74	**-9**
5	27.0	0.0	27	27	0	47.4	0.7	**46**	52	**-6**	71.4	2.3	**68**	78	**-10**
6	34.0	0.0	**34**	35	**-1**	48.9	0.3	**48**	50	**-2**	72.4	1.9	**69**	81	**-12**
7	32.0	0.0	32	31	1	53.9	0.9	**52**	55	**-3**	73.2	3.4	**66**	79	**-13**
8	32.6	0.7	**31**	34	**-3**	48.5	0.8	**47**	49	**-2**	65.6	1.0	**64**	68	**-4**
9	35.0	0.0	35	35	0	50.4	0.8	**49**	54	**-5**	67.0	1.2	**65**	66	**-1**
10	38.0	0.0	38	38	0	53.5	1.4	**50**	54	**-4**	70.9	1.4	**69**	80	**-11**
11	38.0	0.0	38	38	0	44.6	0.8	**44**	47	**-3**	63.0	2.1	**61**	68	**-7**
12	35.0	0.0	35	33	2	53.3	0.5	**53**	56	**-3**	69.5	2.9	**66**	74	**-8**
13	32.0	0.0	32	32	0	44.0	0.0	44	43	1	60.2	3.2	**56**	62	**-6**
14	32.0	0.0	32	32	0	46.2	0.6	46	46	0	69.8	3.0	**66**	74	**-8**
15	34.0	0.0	**34**	35	**-1**	43.5	1.3	**43**	46	**-3**	68.7	2.8	**64**	78	**-14**
16	32.0	0.0	32	32	0	57.6	0.5	57	57	0	68.2	1.8	**66**	77	**-11**
17	37.0	0.0	37	36	1	51.6	0.5	51	50	1	71.7	1.9	**68**	78	**-10**
18	30.0	0.0	30	29	1	54.5	1.4	**52**	54	**-2**	75.9	3.1	**71**	79	**-8**
19	32.0	0.0	32	32	0	52.6	1.1	**51**	56	**-5**	62.8	2.1	**60**	70	**-10**
20	31.2	0.4	31	31	0	50.3	0.7	**49**	50	**-1**	73.1	1.4	**71**	78	**-7**
21	28.1	0.3	28	27	1	50.3	0.5	**50**	51	**-1**	66.3	1.1	**64**	77	**-13**
22	40.0	0.0	40	39	1	51.0	0.0	**51**	54	**-3**	65.1	1.8	**62**	76	**-14**
23	36.0	0.0	36	35	1	49.0	0.0	49	48	1	58.9	1.5	**57**	63	**-6**
24	33.0	0.0	33	32	1	49.3	1.1	**48**	50	**-2**	65.4	2.5	**61**	80	**-19**
25	37.2	0.4	**37**	38	**-1**	50.9	0.3	50	50	0	66.7	1.7	**65**	71	**-6**
26	29.0	0.0	29	29	0	44.0	0.0	**44**	46	**-2**	67.2	1.6	**63**	81	**-18**
27	34.0	0.0	34	34	0	59.7	1.3	**58**	61	**-3**	68.6	3.1	**63**	81	**-18**
28	32.0	0.0	32	32	0	45.8	0.8	**45**	47	**-2**	72.2	1.5	**69**	88	**-19**
29	36.0	0.0	36	35	1	46.5	0.5	**46**	47	**-1**	63.7	1.4	**62**	77	**-15**
30	31.0	0.0	31	31	0	52.9	1.0	**51**	53	**-2**	64.1	1.9	**62**	72	**-10**
31	33.0	1.1	32	32	0	51.0	0.7	**50**	52	**-2**	66.7	2.3	**64**	69	**-5**
32	36.0	0.0	36	35	1	46.3	0.5	**46**	52	**-6**	56.4	1.5	**53**	62	**-9**
33	40.0	0.0	**40**	42	**-2**	50.0	0.0	**50**	52	**-2**	64.6	1.3	**63**	73	**-10**
34	33.0	0.0	**33**	35	**-2**	53.1	1.3	51	51	0	62.2	1.6	**59**	68	**-9**
35	35.0	0.0	**35**	38	**-3**	46.6	0.8	45	45	0	64.1	2.4	**59**	70	**-11**
36	29.0	0.0	29	28	1	51.1	0.9	50	49	1	72.0	2.8	**68**	80	**-12**
37	36.0	0.0	36	35	1	51.2	0.4	51	51	0	65.2	2.4	**62**	73	**-11**
38	30.0	0.0	30	29	1	52.4	0.5	**52**	53	**-1**	60.7	1.5	**58**	72	**-14**
39	29.0	0.0	**29**	30	**-1**	48.8	1.0	**47**	50	**-3**	72.3	2.5	**69**	82	**-13**
40	38.0	0.0	38	37	1	51.3	0.5	51	48	3	65.7	2.0	**64**	69	**-5**
41	34.0	0.0	**34**	35	**-1**	49.0	1.6	**47**	49	**-2**	70.8	1.8	**68**	76	**-8**
42	33.0	0.0	33	33	0	49.9	0.7	**49**	50	**-1**	62.1	1.4	**60**	65	**-5**
43	32.0	0.0	32	32	0	45.2	1.0	**44**	47	**-3**	63.4	1.5	**61**	72	**-11**
44	39.0	0.0	39	39	0	48.2	0.9	47	47	0	65.6	2.8	**62**	68	**-6**
45	36.9	0.3	**36**	38	**-2**	45.6	0.7	44	40	4	68.4	2.5	**64**	69	**-5**
46	33.0	0.0	**33**	34	**-1**	41.4	0.7	**40**	42	**-2**	63.5	2.3	**59**	78	**-19**
47	36.0	0.0	**36**	38	**-2**	47.7	0.5	**47**	52	**-5**	72.1	3.3	**66**	78	**-12**
48	32.0	0.0	**32**	33	**-1**	45.0	0.0	**45**	43	2	66.4	2.4	**63**	75	**-12**
49	38.0	0.0	38	36	2	55.0	0.9	**53**	54	**-1**	69.3	2.3	**67**	73	**-6**
50	31.0	0.0	31	30	1	50.7	0.7	**49**	53	**-4**	69.6	1.4	**67**	74	**-7**

columns labelled *Best RH* contain the best values obtained in [5] through their Rollout heuristic. The columns labelled Δ show the difference between our results and those of our competitor. The best results obtained from either procedure are shown in bold.

For $N = 8$, QCC-ACO outperformed the Rollout heuristic (RH) in 14/50 instances, obtained the same result in 18/50 instances, and was beaten in 18/50 instances. The Wilcoxon signed rank test on QCC-ACO results against the RH results does not show any significant difference, because its p-value is large (0.2031). Despite in the $N = 8$ benchmark there is no clear winner, it is interesting to note that the average improvement obtained by ACO over RH on the improved solutions (Δ column) is equal to 1.64, versus an average worsening of 1.11. It is also interesting to see that, in 44 instances, QCC-ACO produced the same or equivalent solutions in all the 10 executions, as the standard deviation is 0.

For $N = 21$, QCC-ACO outperformed the Rollout heuristic in 35 instances, in only 7 instances QCC-ACO was outperformed by its competitor, while in the remaining 8 instances there was a tie. The average improvement obtained by ACO over RH on the improved solutions is equal to 2.60, versus an average worsening of 1.85. In this case, the Wilcoxon signed rank test has a very small p-value ($3.887 \cdot 10^{-9}$), hence the results of QCC-ACO are significantly better than those of RH.

The most impressive results were obtained for $N = 40$: in all 50 instances QCC-ACO found better solutions than the Rollout heuristic. In particular, the largest value of Δ was -19, obtained in three instances; remarkably, the average improvement obtained by ACO over RH is equal to 12.12. It is worth to notice that in all the instances except the instance #9, also the average makespans obtained by QCC-ACO are better than the best values obtained by the Rollout heuristic. As expected, also the p-value of Wilcoxon signed rank test is even smaller than the case of $N = 21$, i.e. $8.277 \cdot 10^{-10}$.

Figure 2 shows the plot of the solution of problem instance $n.8$ belonging to the $N = 8$ benchmark set, while Figure 3 shows the plot of the solution of problem instance $n.24$ belonging to the $N = 40$ benchmark set (left), together with a graph of the average convergence times obtained on the 10 runs (right). Despite the descending trend shows the first signs of a plateau, there is still some room for a further solution improvement should the time limit of 300 seconds be increased.

5 Concluding Remarks and Future Work

In this work, we propose a novel Ant Colony Optimization (ACO) algorithm for the realization (compilation) of nearest-neighbor compliant quantum circuits. In particular, our ACO algorithm (QCC-ACO) introduces a novel pheromone model and leverages a heuristic-based priority rule inspired by the priority rules proposed by [17, 5] in the recent literature to control the iterative selection of quantum gates to be inserted in the solution.

Table 1 reports the overall and direct comparison of our ACO approach with the state-of-the-art represented, to the best of our knowledge, by the experimental results proposed in [5]. According to our experimental results, QCC-ACO scales quite well on the size of the QCC problem ($N = 8, 21, 40$): overall, QCC-ACO was able to produce a total of 99 better solutions out of the considered 150 QCC instances; in particular, for $N = 40$, QCC-ACO improved over all the given 50 instances.

The priority rule used in [5] extends the one proposed in [17], using a gate selection strategy targeted at minimizing the insertion of SWAP gates, based on (i) the D^S_{sum} and D^S_{min} values, and on (ii) an increasing value of start time t used as a scheduling criterion for iteratively inserting the operations in the solution. It is therefore reasonable to conjecture that the reasons of the better performance of [5] with respect to those obtained in [17] may reside on both previous components. In the future, it would be very interesting to perform an analysis to determine which component plays the leading role in such improvement.

Three further possible directions of future work are also worth being pursued: the first one is to apply our ACO algorithm to the case of QCC problems with cross-talk constraints [3]; the second one is to explore the feasibility of our evolutionary approach to the case of compilation of a quantum algorithms for graph coloring [8]; finally, the current ACO version may be further improved with local search procedures, though an efficient local search implementation for the QCCP seems to require a careful and not straightforward design.

References

1. Baioletti, M., Milani, A., Poggioni, V., Rossi, F.: Experimental evaluation of pheromone models in acoplan. Annals of Mathematics and Artificial Intelligence **62**(3), 187–217 (2011). https://doi.org/10.1007/s10472-011-9265-7, https://doi.org/10.1007/s10472-011-9265-7
2. Baioletti, M., Milani, A., Santucci, V.: A new precedence-based ant colony optimization for permutation problems. In: Simulated Evolution and Learning - 11th International Conference, SEAL 2017, Shenzhen, China, November 10-13, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10593, pp. 960–971. Springer (2017). https://doi.org/10.1007/978-3-319-68759-9_79, https://doi.org/10.1007/978-3-319-68759-9_79
3. Booth, K.E.C., Do, M., Beck, C., Rieffel, E., Venturelli, D., Frank, J.: Comparing and Integrating Constraint Programming and Temporal Planning for Quantum Circuit Compilation. In: Proceedings of the 28^{th} International Conference on Automated Planning & Scheduling, ICAPS-18. pp. 366–374 (2018)
4. Brierley, S.: Efficient implementation of quantum circuits with limited qubit interactions. Quantum Info. Comput. **17**(13-14), 1096–1104 (Nov 2017), http://dl.acm.org/citation.cfm?id=3179575.3179577
5. Chand, S., Singh, H.K., Ray, T., Ryan, M.: Rollout based heuristics for the quantum circuit compilation problem. In: 2019 IEEE Congress on Evolutionary Computation (CEC). pp. 974–981 (2019)
6. Cirac, J.I., Zoller, P.: Quantum computations with cold trapped ions. Phys. Rev. Lett. **74**, 4091–4094 (May 1995). https://doi.org/10.1103/PhysRevLett.74.4091, https://link.aps.org/doi/10.1103/PhysRevLett.74.4091
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, second edn. (2001)

8. Do, M., Wang, Z., O'Gorman, B., Venturelli, D., Rieffel, E., Frank, J.: Planning for compilation of a quantum algorithm for graph coloring. ArXiv **abs/2002.10917** (2020)
9. Dorigo, M., Stützle, T.: Ant Colony Optimization. Bradford Company, USA (2004)
10. Farhi, E., Goldstone, J., Gutmann, S.: A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028 (November 2014)
11. Guerreschi, G.G., Park, J.: Gate scheduling for quantum algorithms. arXiv preprint arXiv:1708.00023 (July 2017)
12. Hart, J., Shogan, A.: Semi-greedy heuristics: An empirical study. Operations Research Letters **6**, 107–114 (1987)
13. Herrera-Martí, D.A., Fowler, A.G., Jennings, D., Rudolph, T.: Photonic implementation for the topological cluster-state quantum computer. Phys. Rev. A **82**, 032332 (Sep 2010). https://doi.org/10.1103/PhysRevA.82.032332, https://link.aps.org/doi/10.1103/PhysRevA.82.032332
14. Merkle, D., Merkle, M., Schmeck, H.: Ant colony optimization for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation **6**(4), 333–346 (2002)
15. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
16. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, New York, NY, USA, 10th edn. (2011)
17. Oddi, A., Rasconi, R.: Greedy randomized search for scalable compilation of quantum circuits. In: van Hoeve, W.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 446–461. Springer International Publishing, Cham (2018)
18. Rasconi, R., Oddi, A.: An innovative genetic algorithm for the quantum circuit compilation problem. In: Proceeding of the Thirty-Third Conference on Artificial Intelligence AAAI-2019. pp. 7707–7714. AAAI Press (2019)
19. Resende, M.G., Werneck, R.F.: A hybrid heuristic for the p-median problem. Journal of Heuristics **10**(1), 59–88 (Jan 2004). https://doi.org/10.1023/B:HEUR.0000019986.96257.50, https://doi.org/10.1023/B:HEUR.0000019986.96257.50
20. Sete, E.A., Zeng, W.J., Rigetti, C.T.: A functional architecture for scalable quantum computing. In: 2016 IEEE International Conference on Rebooting Computing (ICRC). pp. 1–6 (Oct 2016). https://doi.org/10.1109/ICRC.2016.7738703
21. Stützle, T.: An ant approach to the flow shop problem. In: Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing, EUFIT-98, Aachen, Germany. pp. 1560—-1564 (1998)
22. Venturelli, D., Do, M., Rieffel, E., Frank, J.: Temporal planning for compilation of quantum approximate optimization circuits. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. pp. 4440–4446 (2017). https://doi.org/10.24963/ijcai.2017/620, https://doi.org/10.24963/ijcai.2017/620
23. Yao, N.Y., Gong, Z.X., Laumann, C.R., Bennett, S.D., Duan, L.M., Lukin, M.D., Jiang, L., Gorshkov, A.V.: Quantum logic between remote quantum registers. Phys. Rev. A **87**, 022306 (Feb 2013). https://doi.org/10.1103/PhysRevA.87.022306, https://link.aps.org/doi/10.1103/PhysRevA.87.022306