An Improved Hirata Algorithm for Quantum Circuit LNN Conversion

Angel Amarilla, Joaquín Lima, Benjamín Barán
Facultad Politécnica
Universidad Nacional de Asunción
San Lorenzo, Paraguay
e-mail: fangelith@gmail.com, joaquin.lima@pol.una.py, bbaran@pol.una.py

***Abstract*—Hirata et al. proposed an efficient technique for converting general quantum circuits to Linear Nearest Neighbor architecture where an optimal transformed circuit is calculated from the original circuit. However, for some circuits, this algorithm still requires a considerable amount of running time for the conversion. Therefore, in this paper, additional techniques based on *Dynamic Programming* and *Branch & Bound* are proposed in order to improve the running time. Several test circuits from the state of the art have been tested. Experimental results demonstrate the effectiveness of the proposed improvements to reduce the original running time without any loss in solution quality measured as the number of SWAP gates that have been added.**

Keywords*—Quantum Computation; Quantum Circuit Conversion; Linear Nearest Neighbor; Dynamic Programming; Branch & Bound.

I. Introduction

A *quantum computer* is a device whose operation is based on the principles of quantum mechanics. Essentially, these devices allow the implementation of State Models based on performing *gates* operations over qubits, allowing algorithms that may potentially be exponentially faster than their classic counterparts [1]. Currently, these algorithms are implemented in *quantum circuits*. The theoretical design of quantum circuits generally assume the possible interaction of any pair of qubits. However, today quantum computers can only make operations between physically adjacent qubits [2][3]. The architecture of quantum circuits that only consider the interaction of adjacent qubits is called Linear Nearest Neighbor (LNN) [4].

Hirata et al. [5] present a traditional computing algorithm for the conversion of arbitrary quantum circuits to the LNN architecture. This technique is based on inserting *SWAP gates* within the original circuit in order to leave all operations between adjacent qubits. The SWAP gates are inserted taking into account the conversion efficiency of up to w subsequent gates, being w a *depth value* that defines the local search intensity. An appropiate depth value represents a trade-off between the conversion time and the quality of the solutions. Thus, a lower w value might result in a smaller solution quality while a higher w would result in a large calculation time.

This paper proposes two improvements that reduces the running time of Hirata algorithm for the same w values without loss of solution quality. The first approach is based on *Dynamic Programming* [6], in which the solution of circuit patterns are stored and reapplied when these patterns appear again. The second improvement is based on *Branch & Bound* [7], and reduces the time of exploration of the search tree when the next gates are considered for the conversion, avoiding path exploration that will not provide an improvement.

The rest of this work is organized as follows. The next section introduces quantum circuits. Section III presents the LNN Architecture. Section IV discusses the conversion of quantum circuits to an LNN Architecture. Later, in Section V the proposed methods are exposed. Finally, Section VI presents experimental results and the conclusions of this work.

II. Quantum Circuits

A quantum bit or *qubit* [4] is modeled as a mathematical entity that can hold two basic states $|0\rangle$ and $|1\rangle$, which are analogous to states 0 and 1 of classics bits, and also can hold a lineal combination of the basic states: $\alpha|0\rangle + \beta|1\rangle$, called superposition of states [8].

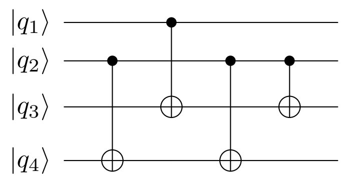


Figure 1. Example of a general quantum circuit with 4 qubits and 4 gates.

On other hand, *quantum gates* are capable of taking as input the states of a given number of qubits and to change the states of these qubits in a desired manner. In practice, a number of consecutive quantum gates conforms a *quantum circuit*. Figure 1 illustrates a quantum circuit, where the horizontal lines represent qubits being operated by quantum gates, which are shown transversely, taking as input one or more qubits. Thus, Figure 1 shows a circuit of 4 qubits and 4 gates.

III. Lineal Nearest Neighbor Architecture

In designing a quantum circuits, it is generally supposed that the quantum gates can operate any pair of qubits for example in [9] and [10]. However, quantum computers that allow a real implementation of quantum circuits with current technology, may not support the interaction of arbitrary further qubits. Indeed, some quantum architectures require circuits in where its quantum gates can only operate qubits that are physically adjacent [2][3]. The architecture of quantum circuits in which only adjacent qubits interact through a gate is called LNN [4].

A quantum gate of special attention is the *SWAP gate*, that is capable of taking two input qubits and interchange their states. The operation of a SWAP gate is given by $swap(|a,b\rangle) =$

$|b, a\rangle$. The representation of a SWAP gate in quantum circuits is given by the symbol $\times$ on two adjacent qubits as shown in Figure 2.

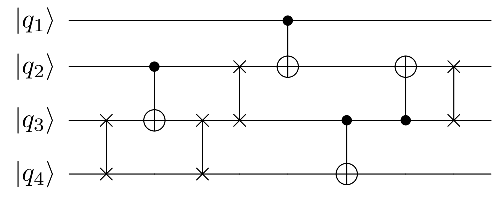


Figure 2. LNN circuit that is obtained adding four SWAP gates to circuit of Figure 1.

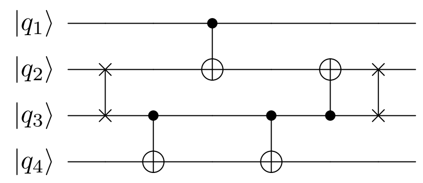


Figure 3. LNN circuit that is obtained adding two SWAP gates to circuit of Figure 1.

In general, quantum circuits can be converted to an LNN architecture by inserting additional SWAP gates within them. Figures 2 and 3 show two LNN circuits that are obtained by inserting additional SWAP gates in the general quantum circuit of Figure 1. It is interesting to note that in Figure 2 four new SWAP gates have been added to the original circuit, while in Figure 3 only two new SWAP gates are needed for the same task. In those cases, the insertion of a smaller number of new SWAP gates is preferred.

IV. LNN CONVERTION OF QUANTUM CIRCUITS

The conversion of a general quantum circuit to LNN architecture is defined by Hirata [5] as:

- **Input**: A general quantum circuit.
- **Output**: An equivalent LNN quantum circuit.
- **Objective**: Minimize the number of added SWAP gates.
- **Restriction:** the equivalent circuit output should have all qubits in the same original order.

$$\begin{array}{cccc} |q_1\rangle & |q_4\rangle & |q_2\rangle & |q_3\rangle \\ |q_2\rangle & |q_1\rangle & |q_4\rangle & |q_3\rangle \\ |q_2\rangle & |q_3\rangle & |q_1\rangle & |q_4\rangle \end{array}$$

Figure 4. Permutations considered for the Hirata algorithm.

One possible strategy to solve this problem is to convert each original gate inserting the smaller possible number of new SWAP gates. After the conversion of an original gate it must be proceed to converting the next original gate, taking into account the new order of the qubits according to the SWAP gates inserted before. Finally, after the conversion of the last original gate a small number of new SWAP gates it must be inserted in order to get the order of qubits given by the original circuit. This algorithm is known as *Greedy Strategy* [5], which is always able to find reasonable (sub-optimal) solutions.

V. HIRATA ALGORITHM

Hirata et al. [5] proposed an efficient algorithm to convert general quantum circuits to an LNN architecture.

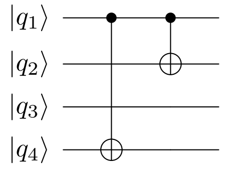


Figure 5. Quantum circuit of example.

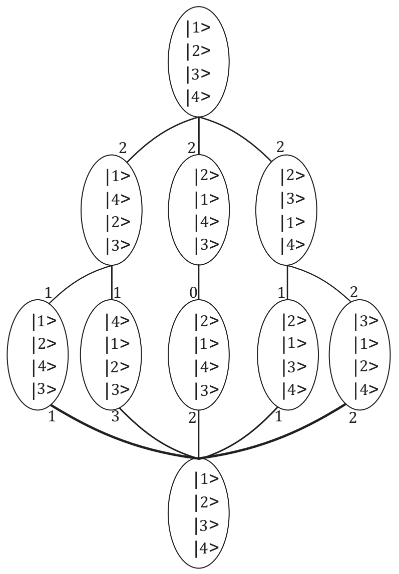


Figure 6. Whole search Tree for the quantum circuit of Figure 5.

For each pair of non-adjacent operated qubits that have m intermediate qubits, Hirata algorithm take into account $(m+1)$ possible permutations. For example, for the case of converting the first gate of the circuit of Figure 5, the qubits to be operated are $|q_1\rangle$ and $|q_4\rangle$, the value of m is 2 and the permutations considered by the algorithm are listed in the Figure 4. Note that in all cases, $|q_1\rangle$ and $|q_4\rangle$ are become neighbors, preserving the original order between them.

In Hirata algorithm [5], each considered permutation is called a *candidate*. Each candidate represents a possible re-order of the qubits to convert the current gate and it is a part of one or more solutions. In Figure 6, the conversion of the first gate of the quantum circuit from Figure 5 have 3 candidates. The numerical value over each candidate represents its required number of SWAP gates. Thus, the cost for each complete solution is given by the sum of these numerical values. The desired solution is the one with the lowest cost, that is illustrated in Figure 6 by the central path, with a final cost of $2 + 0 + 2 = 4$ additional swap gates.

```
swaps  = 0
l_order = {1,2,...,n}
c_order = l_order
w = local search deep
K = total of original gates
procedure HirataAlgorithm() {
  for i=1 to K do
    val_func_min = $\infty$
    S = $\emptyset$
    for j = 1 to candidates(i) do
      val_func=local_search(n_$order_{ij}$,w)
           +calc_swap(c_order,n_$order_{ij}$)
           +$\frac{c_k}{K+1-i}$calc_swap(n_$order_{i,j}$,$l\_order$)
      if (val_func < val_func_min) then
        val_func_min = val_func
        S = {n_$order_{i,j}$}
      else if (val_func = val_func_min) then
        S = S $\cup$ n_$order_{i,j}$
      end if
    end for
    $S_p$ = random(1,$|S|$)
    swap = swap + calc_swap(c_order, $S_p$)
    c   _order = $S_p$
  end for
  swap = swap + calc_swap(c_order,l_order)
  return swap
end procedure HirataAlgorithm
```

Figure 7. Hirata algorithm from LNN convertion.

Hirata algorithm [5], presented in Figure 7, performs the selection of a candidate for each gate considering:

- a Local Search procedure that evaluates the quality of the candidates orders (indicated by n_order) considering the following w gates;
- the amount of SWAP gates to be added to obtain the candidate order (n_order) from the current order (c_order);
- and, the cost of converting the candidate order (n_order) to the original order (l_order) of the quantum circuit, weighted by a value that takes precedence towards the end of the circuit.

VI. PROPOSED IMPROVEMENTS

This section presents two improvements for the algorithm of Hirata et al. [5]: one based on Dynamic Programming [6] [11] and the other based on Branch & Bound [7].

A. Branch & Bound improvement

This improvement is applied in the local search procedure to prevent the exploration of branches of the search tree that will not lead to a better solution than the one in search [7].

Figure 8 illustrates a complete *local search tree* corresponding to candidate $|q_1q_2q_4q_3\rangle$. Initially there is no *selected solution*. The calculation begins considering the first candidate that is considered as the selected one. For example, consider as the *first selected solution* the candidate most in the left of

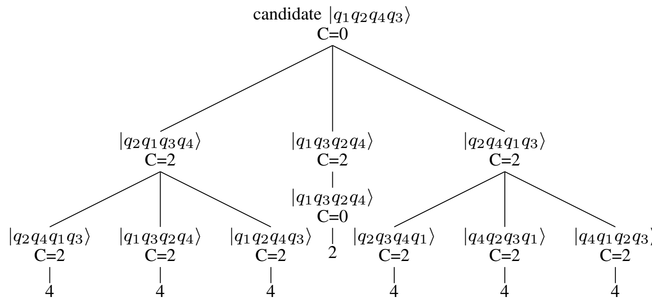


Figure 8. Local search tree corresponding only to candidate $|q_1q_2q_4q_3\rangle$ of the first gate of Figure 1 circuit.

the tree (see Figure 8) whose evaluation equals to 4 at the end of the local search tree. Note also that the height of local search tree is defined by $w = 2$.

Afterward, the algorithm proceed in evaluating the *next candidate*, whose evaluation culminate only if it is not worse than the current selected solution. This is determined taking into account the amount of SWAP gates accumulated during the search tree exploration. If the amount of SWAP gates corresponding to the current candidate is larger than the amount of the current selected solution, then the exploration is stopped and the candidate is discarded.

When the evaluation of a candidate ends with a lower amount of SWAP gates than the corresponding amount of the current selected solution, the candidate is chosen as the *new selected solution*. Consider again the search tree of Figure 8. Initially, the selected solution is the candidate most in the left, that is subsequently replaced by the candidate of the center.

With this improvement, the local search procedure can find the same solution as the original local search algorithm, but with the advantage of not fully evaluating the search tree, reducing this way its processing time. Figure 9 illustrates the search tree that is explored when the Branch & Bound improvement is applied. Note that it is not necessary to fully evaluate the final candidate

The local search procedure with the improved proposal of Branch & Bound is presented in Figure 10.

B. Dynamic Programming improvement

Dynamic Programming [6] is a technique that solves a problem P based on recursively solving all sub-problems S_i therein. Each sub-problem S_i is solved only once and its solution is saved in a table $T[S_i]$. When the sub-problem S_i is found once again, the solution saved in $T[S_i]$ is reapplied; therefore, the time and effort spent in solving S_i is saved.

The proposed improvement apply Dynamic Programming [6] in the Hirata algorithm considering as a sub-problem S_i each current order of qubits c_order and its w corresponding successive pairs of qubits.

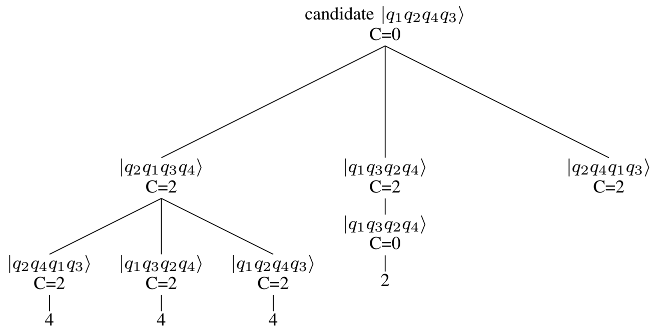


Figure 9. Local search tree explored for candidate $q_1q_2q_4q_3$ of the first gate of circuit of Figure 1 when considering the Branch & Bound improvement.

```
K = total of original gates
c_swap = SWAP gates in current solution
min_swap = SWAP gates in candidate solution
k = current gate in convertion
i = current gate in local search
w = deep value

procedure **local_search**(c_swap, i, c_order, k)
  **if** (i > K) or (i > k + w) then
    **if** c_swap < min_swap then
      min_swap = c_swap
    end **if**
  **else**
    **for** j = 1 to (m+1) candidates **do**
      n_swap=c_swap + **calc_swap**(c_order, n_$order_{i,j}$)
      **if** n_swap < min_swap then // Branch&Bound
        min_swap =
          **local_search**(n_swap, i+1, n_$order_{i,j}$, k)
      end **if**
    end **for**
  end **if**
  **return** min_swap
end procedure **local_search**
```

Figure 10. Local Search procedure with Branch & Bound.

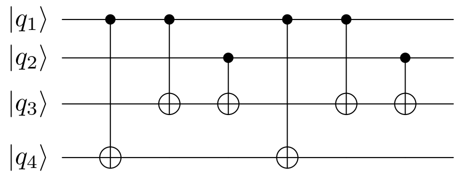


Figure 11. Example circuit for illustrate the Dynamic Programming improvement

For example, consider the circuit of Figure 11 and $w = 2$. The sub-problem in the first gate is as follows:

- c_order_1 = $q_1q_2q_3q_4$
- w next pairs of qubits = $[(q_1, q_3), (q_2, q_3)]$

Considering that the calculated solution for this sub-problem adds the SWAP gates between (q_1, q_2) and between (q_3, q_4) before the first gate, as shown in Figure 12; then, the corresponding input in the table T of patterns is as follows:

$$T[q_1q_2q_3q_4, (q_1, q_3), (q_2, q_3)] = (q_1, q_2), (q_3, q_4) \qquad (1)$$

Later, this sub-problem could need to be considered again. For example, when the conversion reaches up to the fourth gate, as shown in Figure 12. Here, the solution of equation (1) saved in the Table T is reapplied without recalculating the same pattern, as shown in Figure 13.

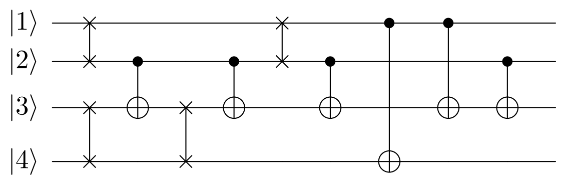


Figure 12. LNN conversion of the circuit 11 for the first 3 gates

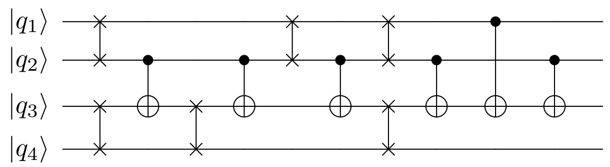


Figure 13. LNN conversion of the circuit 11 for the gate $k = 4$

VII. EXPERIMENTAL RESULTS

A. Combination of the proposed improvements

This section presents the experimental results obtained by the implementation of both proposed improvements in the original Hirata algorithm [5]. Nine test circuits were considered: three Shor circuits, three Modmulti circuits and three random circuits, also used in Hirata et al. [5].

For the performed experiments, a computer with 3 GHz Dual Core i5 processor, 8 GB RAM and the implementation of the algorithms in Java 7 were set. The values of $w \in \{8, 9, 10\}$ have been considered. Given the randomness of the algorithm in case of ties, each test circuit has been converted 10 times using each algorithm:

- H: the original Hirata algorithm,
- H-BB-PD: Hirata algorithm with both proposed improvements (Dynamic Programming and Branch & Bound);
- H-PD: Hirata algorithm with Dynamic Programming; and
- H-BB: Hirata algorithm with Branch & Bound.

Thus, a total of 360 runs were considered, obtaining in each test the following metrics:

- the average running time ($\overline{T}$) and its standard deviation ($\sigma(T)$); and
- the average number of SWAP gates of the calculated solution ($\overline{S}$) and its standard deviation ($\sigma(S)$).

A comparison of results of the original Hirata algorithm (H) versus the same algorithm with both proposed improvements (H-BB-PD) are shown in table I. In general, it can be observed that proposed improvements can achieve a significant reduction in running time without an appreciable difference with respect to the number of calculated SWAP gates by the original Hirata Algorithm.

On the other hand, another considered experiment consists in taking into account both the elapsed running time and the total number of SWAP gates after each 50 processed original gates. Thus, Figure 14 presents a comparison of the

TABLE I. RESULTS OF THE ORIGINAL HIRATA ALGORITHM COMPARED TO THE PROPOSED IMPROVED ALGORITHM.

		W=8		W=9		W=10	
Circuit	Algorithm	Time (ms.) $\overline{T}/\sigma(T)$	SWAPs $\overline{S}/\sigma(S)$	Time (ms.) $\overline{T}/\sigma(T)$	SWAPs $\overline{S}/\sigma(S)$	Time (ms.) $\overline{T}/\sigma(T)$	SWAPs $\overline{S}/\sigma(S)$
Shor 3	H	721 / 125.79	1876 / 34.78	1255 / 138.00	1870 / 25.64	2674 / 235.20	1880 / 0.0
	H-BB-PD	92 / 49.78	1876 / 34.78	106 / 12.59	1870 / 25.64	146 / 10.62	1880 / 0.0
Shor 5	H	21780 / 1375.69	11346 / 110.69	65072 / 3770.91	10959 / 0.0	194128 / 14956.65	11968 / 233.01
	H-BB-PD	765 / 111.05	11346 / 110.69	1484 / 324.06	10959 / 0.0	2380 / 301.39	11968 / 233.01
Shor 6	H	75134 / 3033.71	21000 / 520.83	244525 / 8562	21184 / 306	769350 / 47910	21198 / 253
	H-BB-PD	2636 / 491.29	21000 / 520.83	3752 / 799	21184 / 306	7004 / 1711	21198 / 253
Random 500	H	2004 / 153.46	748 / 1.15	5686 / 159.28	742 / 0.0	18419 / 482.62	752 / 3.02
	H-BB-PD	270 / 31.47	748 / 1.15	480 / 44.69	742 / 0.0	915 / 90.36	752 / 3.02
Random 1000	H	4076 / 202.39	1492 / 12.08	11914 / 394.75	1464 / 1.05	38652 / 271.91	1462 / 1.70
	H-BB-PD	563 / 55.94	1492 / 12.08	1000 / 103.17	1464 / 1.05	1866 / 147.84	1462 / 1.70
Random 2000	H	8920 / 126.29	3084 / 4.94	26650 / 659.38	3000 / 1.63	86859 / 868.22	3004 / 6.60
	H-BB-PD	1211 / 86.96	3084 / 4.94	2149 / 156.11	3000 / 1.63	4111 / 159.87	3004 / 6.60
Modmulti 3	H	143 / 12.56	322 / 7.73	307 / 31.82	320 / 6.82	740 / 78.67	328 / 1.15
	H-BB-PD	32 / 5.07	322 / 7.73	67 / 7.54	320 / 6.82	90 / 12.38	328 / 1.15
Modmulti 4	H	663 / 64.13	648 / 2.00	1714 / 113.42	654 / 3.89	5014 / 224.46	654 / 7.73
	H-BB-PD	142 / 19.94	648 / 2.00	256 / 16.39	654 / 3.89	425 / 35.33	654 / 7.73
Modmulti 5	H	2444 / 308.53	1138 / 10.80	6195 / 152.92	1124 / 1.94	19662 / 1022.59	1180 / 2.83
	H-BB-PD	535 / 54.45	1138 / 10.80	803 / 76.47	1124 / 1.94	1323 / 182.54	1180 / 2.83

performance of both algorithms for a value of $w = 8$ in test circuit Shor 6. It can be observed that the proposed improvements allow an equivalent solution in a shorter running time.

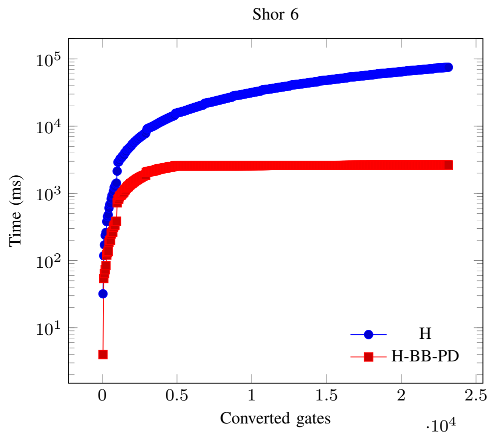


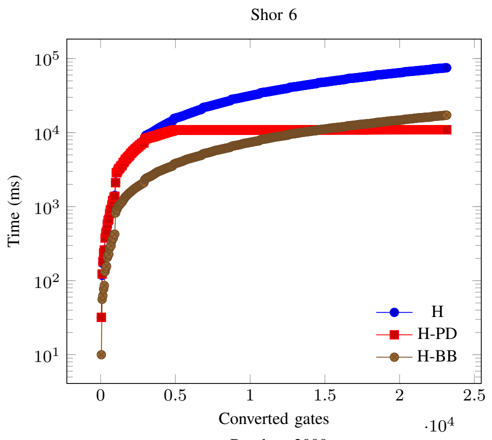
Figure 14. Evolution of conversion versus the conversion time between the original algorithm and both improvements.

B. Comparison between both methods

In order to determinate the individual advantage of using both improvement proposals, each of these two improvements were separately applied and then compared to the original Hirata algorithm [5]. Table II shows the normalized results of this experiment, using the notation:

$$\overline{T}_{norm} = \frac{\overline{T}_{alg}}{\overline{T}_H} \quad (2)$$

where $\overline{T}_{norm}$ is the normalized running time, $\overline{T}_{alg}$ is the value of the average running time of the considered algorithm, $\overline{T}_H$ is the average running time for the original Hirata algorithm. Thus, lower values of $\overline{T}_{norm}$ are preferred.



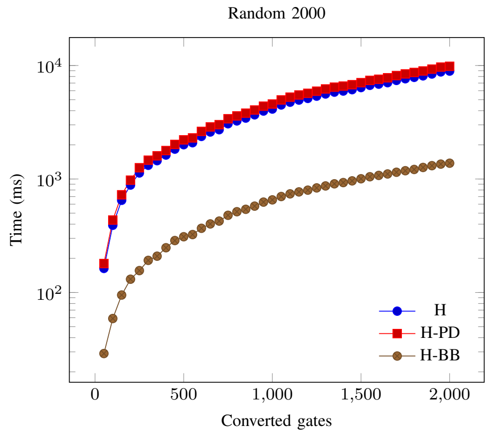


Figure 15. Evolution of conversion versus the conversion time for each improvement.

TABLE II. NORMALIZED COMPARISON BETWEEN IMPROVEMENTS.

Benchmark	w	H-BB $\overline{T}_{norm}$	H-DP $\overline{T}_{norm}$	H $\overline{T}_{norm}$
Shor 3	8	0,2927	0,3536	1
	9	0,2153	0,3593	1
	10	0,1622	0,3144	1
Shor 5	8	0,2012	0,1505	1
	9	0,1287	0,1509	1
	10	0,0730	0,1426	1
Shor 6	8	0,2069	0,1314	1
	9	0,1006	0,1177	1
	10	0,0565	0,1303	1
Random 500	8	0,1342	0,9017	1
	9	0,0818	0,8994	1
	10	0,0482	0,8912	1
Random 1000	8	0,1418	0,8929	1
	9	0,0833	0,8989	1
	10	0,0489	0,8880	1
Random 2000	8	0,1381	0,9863	1
	9	0,0819	1,0288	1
	10	0,0494	1,0386	1
Modmulti 3	8	0,2917	0,9375	1
	9	0,2016	0,8638	1
	10	0,1233	0,8657	1
Modmulti 4	8	0,2280	0,8297	1
	9	0,1578	0,9185	1
	10	0,0871	0,8961	1
Modmulti 5	8	0,2270	0,8996	1
	9	0,1241	0,8661	1
	10	0,0658	0,8738	1

Considering the improvement based on the Dynamic Programming technique, it can be seen that the test circuits Shor 3, Shor 5 and Shor 6 were converted in a shorter running time because several subcircuits are repeated during calculation. However, when there are few or no subcircuits that repeat within the original circuit to convert, such as in Random and Modmulti circuits, this improvement can not reach a better running time compared to the original Hirata algorithm. Thus, this method is only effective in circuits with repetitive patterns.

On the other hand, considering the Branch & Bound improvement, the results in all test circuits show a significant decrease in running time respect to the original Hirata algorithm. Therefore, this improvement is of general application in contrast to the Dynamic Programming approach which is more selective in its applicability.

There also have been taken samples to show the evolution of the resolution time versus the original number of converted gates. Figure 15 confirms that Dynamic Programming improvement only contribute to a reduction of the running time in circuits with repetitive patterns, such as in Shor circuits. On the other hand, all the plots in Figure 15 show a shorter running time when the improvement based on the Branch & Bound technique is applied. This result confirms the general application of the Branch & Bound improvement and the advantage of using it in almost any case.

VIII. CONCLUSIONS

In this work, it has been presented two proposals that improve the running time of the original Hirata algorithm [5] for the conversion of arbitrary quantum circuits to LNN architecture without any loss of quality in the calculated solutions.

The first improvement based on Dynamic Programming [6], proved to be effective converting circuits with repetitive constructions as Shor circuits, avoiding the recalculation of patterns that repeats; however, the improvement is only effective in such circumstances, and not in every studied circuit.

On the other hand, the second improvement based on Branch & Bound [7], has proved to be an improvement of general application with a positive effect over the original algorithm in all studied cases.

In general, the experimental results demonstrates that applying the two proposed improvements can achieve the same quality results that the original Hirata algorithm with a smaller running time without any loss of solution quality. Therefore, both proposals can be considered in order to implement an efficient version of the Hirata algorithm.

ACKNOWLEDGMENT

Special thanks to the Nara Institute of Science and Technology (NAIST) in the person of Yasuhiko Nakashima, Hiroyuki Seki, Masaki Nakanishi, Shigeru Yamashita and Yuichi Hirata, for all help and for providing the test circuits.

REFERENCES

[1] E. Strubell, "An introduction to quantum algorithms," COS498 Chawathe Spring, 2011.

[2] H. Häffner et al., "Scalable multiparticle entanglement of trapped ions," Nature, vol. 438, no. 7068, 2005, pp. 643–646.

[3] M. Laforest, D. Simon, J.-C. Boileau, J. Baugh, M. J. Ditty, and R. Laflamme, "Using error correction to determine the noise model," Physical Review A, vol. 75, no. 1, 2007, p. 012331.

[4] M. A. Nielsen and I. L. Chuang, Quantum computation and quantum information. Cambridge university press, 2010.

[5] Y. Hirata, M. Nakanishi, S. Yamashita, and Y. Nakashima, "An efficient conversion of quantum circuits to a linear nearest neighbor architecture," Quantum Info. Comput., vol. 11, no. 1, Jan. 2011, pp. 142–166.

[6] S. Dreyfus, "Richard bellman on the birth of dynamic programming," Operations Research, vol. 50, no. 1, 2002, pp. 48–51.

[7] J. Clausen, "Branch and bound algorithms-principles and examples," Department of Computer Science, University of Copenhagen, 1999, pp. 1–30.

[8] R. P. Feynman, "Quantum mechanical computers," Foundations of physics, vol. 16, no. 6, 1986, pp. 507–531.

[9] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on. IEEE, 1994, pp. 124–134.

[10] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," arXiv preprint quant-ph/0410184, 2004.

[11] R. E. Bellman and S. E. Dreyfus, Applied dynamic programming. Rand Corporation, 1962.