JPlace: A Clock-Aware Length-Matching Placement for Rapid Single-Flux-Quantum Circuits

Siyan Chen[1,2†], Rongliang Fu[3†], Junying Huang[4*], Zhimin Zhang[4], Xiaochun Ye[4], Tsung-Yi Ho[3], Dongrui Fan[5]
[1]*ShanghaiTech University*, Shanghai, China
[2]*Shanghai Innovation Center for Processor Technologies*, Shanghai, China
[3]*Department of Computer Science and Engineering, The Chinese University of Hong Kong*, Hong Kong, China
[4]*SKLP, Institute of Computing Technology, CAS*, Beijing, China
[5]*University of Chinese Academy of Sciences*, Beijing, China

***Abstract*—Superconducting rapid single-flux-quantum (RSFQ) logic has emerged as a promising candidate for future computing technology, owing to its low power consumption and high frequency characteristics. Given its ultra-high frequency operation, achieving precise timing alignment is crucial for RSFQ circuit physical design. To address the timing issue, this paper introduces JPlace, a clock-aware length-matching placement framework for RSFQ circuits. JPlace simultaneously addresses data and clock signal length matching, effectively ensuring accurate timing alignment and mitigating timing alignment challenges during the routing phase. We propose a heuristic method for constructing the clock distribution and a dynamic programming-based approach for minimizing the total vertical wirelength while maintaining fixed placement orders. Additionally, we introduce a barycenter-based reordering method to further explore the solution space and reduce wirelength. Experimental results on the RSFQ benchmark demonstrate the effectiveness and efficiency of JPlace.**

***Index Terms*—superconducting logic, RSFQ, placement**

I. Introduction

The rapid single-flux-quantum (RSFQ) [1] circuit is a type of Josephson Junction (JJ)-based superconducting integrated circuit that holds great promise for future high-performance computing systems. The benefits of RSFQ circuits, including their remarkable ultra-fast switching speed and ultra-low switching energy, have attracted significant attention. It has been demonstrated that several RSFQ microprocessors have achieved high clock frequencies at 25-50 GHz [2]–[4]. Nowadays, high-performance large-scale RSFQ circuits, such as hardware accelerators for machine learning [5]–[7], can also operate at frequencies in the tens of GHz. Although superconducting RSFQ circuits share similarities in design, manufacturing, and testing techniques with contemporary complementary metal oxide semiconductor (CMOS) integrated circuits, there are notable differences between the two that render existing CMOS electronic design automation (EDA) tools unsuitable for direct application in the design of RSFQ circuits.

From the perspective of physical design automation, some distinctive characteristics of RSFQ circuits stem from their pulse-driven nature, with the primary differences from CMOS circuits being as follows: 1) Clock-driven gate-level pipelining. In RSFQ circuits, almost all logic gates operate in synchronization with the clock. This means that nearly every RSFQ logic gate possesses a latching function, enabling gate-level pipelining. This structural feature of RSFQ circuits not only facilitates high clock frequencies but also imposes additional constraints on placement and routing methods, such as the requirement to arrange logic gates in columns according to their logical stage. 2) Pulse-driven timing scheme. Unlike traditional CMOS technology, the frequency of RSFQ circuits is determined by the time difference between the arrival of data and clock pulses. The frequency will decrease significantly if there is a significant difference in the arrival times of data and clock pulses at RSFQ logic gates. Therefore, matching the arrival times of data and clock pulses is crucial for maximizing the frequency of RSFQ circuits. Consequently, timing alignment is a critical consideration in the physical design of RSFQ circuits.

Research in the field of RSFQ placement has been relatively limited, with notable works including [8]–[10]. Kito *et al.* [8] introduced a simulated annealing (SA)-based RSFQ placement algorithm. However, the SA-based approach requires a long runtime, particularly for large-scale circuits, due to the computational nature of SA. Moreover, this method did not consider the issue of RSFQ timing alignment. Yan [9] proposed a fixed-order placement technique for RSFQ circuits that is faster than SA. Nevertheless, this approach enforces a static order for logic gates, meaning the relative positions of the gates remain unalterable during placement, solely focusing on optimizing their absolute positions. This constraint limits the exploration of potential solution spaces. Furthermore, this method focuses on matching the lengths of data signals without addressing the length matching of clock signals. This significantly escalates the complexity of timing alignment during the routing phase, particularly for complex and large-scale RSFQ circuits. Kitamura *et al.* [10] introduced a length-matching-aware RSFQ placement approach, aiming to minimize the sum of maximum vertical distances between connected gates to mitigate wirelength extension in wirelength matching. Unfortunately, this method did not address clock line length matching either.

In this paper, we propose a clock-aware length-matching RSFQ placement algorithm named JPlace, to tackle the timing alignment issue in RSFQ circuits. This innovative approach simultaneously considers both data and clock signal length matching, thereby achieving genuine timing alignment and alleviating the timing alignment challenges during the routing

† Equal contribution. * Corresponding author: huangjunying@ict.ac.cn.

979-8-3503-4859-0/DATE24/© 2024 EDAA

phase, especially for complex and large-scale RSFQ circuits.

Specifically, we make the following contributions:

- We propose JPlace, a clock-aware length-matching placement approach that considers the timing constraints of RSFQ circuits. This method employs a heuristic strategy to efficiently generate the clock distribution within a multi-stage pipelined architecture.
- We present a dynamic programming-based placement technique that identifies the optimal placement for each gate within its respective column for the given gate order.
- Moreover, we propose a barycenter-like reordering method to fine-tune the relative positions of gates within the same column, resulting in a further reduction in wirelength.
- Experimental results on the ISCAS85 [11] and EPFL [12] benchmarks show the effectiveness and efficiency of JPlace, an average reduction of 49.00% and 48.15% in the total vertical wirelength, respectively, compared to the baseline.

II. Preliminary

A. RSFQ Logic

RSFQ logic gates usually consist of inductors, resistors, and JJs. These JJs enable the data transmission as voltage pulses and the storage of magnetic flux quanta within the RSFQ gates. In contrast to CMOS circuits, which have higher fanout capabilities, RSFQ circuits require a splitter (SPL) when the fanout of a gate exceeds 1. The SPL divides one pulse into two or three, depending on its specific design. Most RSFQ gates are synchronized with a clock pulse to transfer the stored data to the adjacent gates. In other words, nearly all RSFQ gates have the latch functionality, allowing them to be pipelined without additional delay-flip-flops (DFFs). With this property, it is natural to adopt RSFQ pipelined placement based on the logical stage of each logic gate. The logical stage of one gate represents the maximum number of clocked gates in any path from any primary input (PI) to it. All the paths share the same number of clocked gates due to path balance during logic synthesis [13]. In the pipelined RSFQ layout, the logical stage for each logic gate is calculated first. Then, the logic gates are placed in columns according to their logical stages. Logic gates with the same logical stage are placed within the same column. These columns are ordered from left to right in ascending order of the logical stage, resulting in a multi-stage pipelined layout.

B. Timing Constraints

RSFQ logic offers superior performance compared to CMOS logic due to its utilization of quantized voltage pulses for rapid switching in digital data generation. RSFQ circuits employ concurrent-flow clocking as the representative clocking scheme [15], which effectively conceals data propagation delays by synchronizing the clock pulse with the data. The timing constraint for concurrent-flow clocking is depicted in Fig. 1(a). To ensure correct operation, the arrival time of the clock and data pulses must satisfy:

$$t_c + t_{hold} < t_{data} < t_c + t_T - t_{setup}, \tag{1}$$

which t_c and t_{data} are the arrival time of the clock pulse and the data pulse at an RSFQ gate, respectively. Besides, three crucial

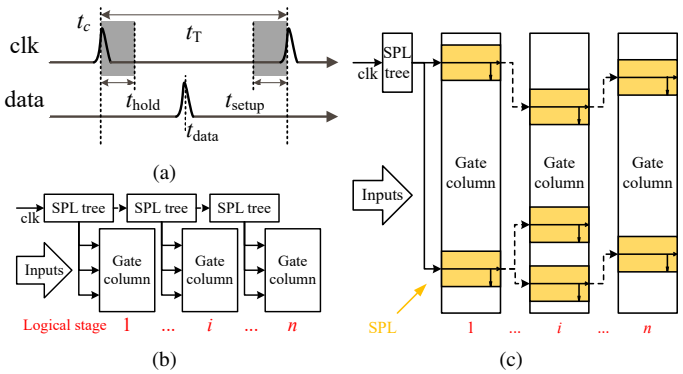


Fig. 1. (a) RSFQ logic timing constraints. (b) and (c) are the tree-based clock distributions for RSFQ circuits by Kito *et al.* [8] and Yan [14], respectively.

timing parameters must be considered: t_{setup}, t_{hold}, and t_T. t_{setup} denotes the minimum time that the input data pulse must be stable and available before the clock pulse. t_{hold} denotes the minimum time that the input data pulse must remain stable and unchanged after the clock pulse. t_T denotes the clock cycle.

In RSFQ circuits, the clock pulse should be delivered to nearly all logic cells in the design. Therefore, clock distribution is essential in the physical design of RSFQ circuits, as it requires significant routing resources and consumes considerable power. Previous studies have proposed various tree-based clock distribution methods for RSFQ circuits. Kito *et al.* [8] introduces a multi-level distribution tree for clock distribution. As shown in Fig. 1(b), the clock pulse from the PI enters an SPL tree and gets divided into multiple clock pulses that propagate to the gates in the current column and the SPL tree in the next column. However, since the clock pulses of all gates in one column source from the same SPL tree, the nets between the SPL tree's output and the gates' clock pin require lengthy wires, resulting in increased routing costs. Yan [14] proposes an alternative clock distribution method, as shown in Fig. 1(c), which reduces the wirelength of the clock nets. The clock pulse from the PI first flows into an SPL tree, similar to Kito's clock tree, generating clock pulses for the gates in the first column. For the remaining columns, the clock pulse passes through the preceding column using SPLs and propagates to the adjacent succeeding column. This paper adopts this clock distribution framework and proposes a heuristic approach to generate the clock distribution.

C. Length-Matching RSFQ Placement

The primary objective of RSFQ placement is to produce a layout that meets timing constraints while minimizing wirelength. The timing constraint ensures the circuit is functionally correct, and the minimal wirelength corresponds to a reduced area of routing regions, ultimately leading to a smaller layout area. In RSFQ circuits, the timing constraint can be met by extending the length of passive transmission lines (PTLs). PTL, a passive routing cell, is commonly used as the interconnect in RSFQ circuits due to its ability to enhance the operational margin. The delay of a PTL can be roughly proportional to its length. Consequently, the need for timing adjustment can be translated into length-matching requirements in the

placement process. To meet the timing requirements between the arrival of the clock and data pulses, the logic gates need to satisfy Equation (1). Considering that the delay of a PTL is approximately proportional to its length, timing constraints can be transformed into a length-matching problem:

$$l^c + l^{hold} < l^{data} < l^c + l^T - l^{setup}, \tag{2}$$

where l^c and l^{data} refer to the PTL lengths of the clock connection and data connection, respectively, with corresponding delays t_c and t_{data}. Besides, l^{hold}, l^T, and l^{setup} represent the PTL lengths associated with t_{hold}, t_T, and t_{setup}, respectively.

To offer better tolerance to process variations, we aim for the data pulse to arrive at the midpoint of the range, which means

$$l^{data} = \frac{1}{2}[(l^c + l^{hold}) + (l^c + l^T - l^{setup})] = l^c + \Delta l, \tag{3}$$

where $\Delta l = \frac{l^T - l^{setup} + l^{hold}}{2}$ represents the extra PTL length of the data connection compared to the clock connection.

Prior research has introduced the length-matching placement method for RSFQ circuits [8], [9], aiming to optimize the sum of the vertical minimum length and vertical matching length of data nets. The vertical minimum length pertains to the overall vertical Manhattan distance between logic gates, while the vertical matching length signifies the total length of the detour implemented to satisfy timing constraints. The connections between adjacent columns have the same horizontal length. Therefore, when considering minimum length optimization, previous approaches focus on the vertical length, referred to as the vertical minimum length, for each connection. Besides, these placement methods all assume that detour occurs exclusively in the vertical direction, hence they utilize the vertical matching length to represent the matching length. This paper also follows the same assumption. In their research, the matching length is used to fine-tune the arrival time of data inputs at a gate. Following length-matching, the objective is to achieve simultaneous arrival of all data inputs at an individual gate. However, their placement approaches do not take clock nets into account. In reality, a placement result that exclusively prioritizes data nets may not be optimal for clock nets, as the timing constraint, as outlined by Equation (3), is jointly defined by both the clock net and the data net. Given this context, we aim to propose a clock-aware length-matching RSFQ placement method that integrates the influence of the timing constraint during the placement process.

III. Problem Formulation and Terminology

An RSFQ circuit can be represented by a directed graph $G(V, E)$. E is the set of all nets, including data nets E_d and clock nets E_c. The node set V includes logic gates, PIs, and primary outputs (POs). The logic gate node corresponds to a logic gate component, which consists of a logic gate and an SPL, as illustrated in Fig. 2. All nodes are grouped into n columns in terms of their logical stages. The logical stages of PIs and POs are 0 and $n+1$, respectively. For node v, s_v denotes its logical stage. The nodes with a logical stage of i are contained in the set $V_i = \{v | s_v = i, v \in V\}$. Besides, a directed edge $(u, v) \in E$ indicates the flow of data from

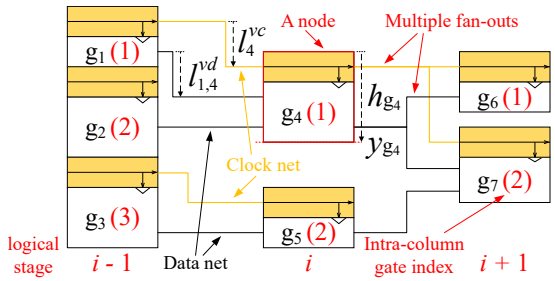


Fig. 2. Partial schematic of circuit placement. g_4 has a height of h_{g_4} and locates at y_{g_4}. Its data inputs come from g_1 and g_2, and its output flows to g_6 and g_7, i.e., $FI(g_4) = \{g_1, g_2\}$ and $FO(g_4) = \{g_6, g_7\}$. The vertical minimum length of data connection from g_1 to g_4 is $l_{1,4}^{vd}$. The vertical minimum length of clock connection of g_4 is l_4^{vc}.

node u to node v. For node v, $E_i(v)$ represents the set of its input edges, while $E_o(v)$ represents the set of its output edges. The set $FI(v) = \{u | (u, v) \in E_i(v)\}$ contains nodes that serve as data inputs to node v, and $FO(v) = \{u | (v, u) \in E_o(v)\}$ contains nodes that are data outputs of node v. The location of node v within a column, marked as y_v, represents the location of the node's lower boundary. Besides, we use h_v to represent the height of the node v.

As discussed in Section II-C, the optimization goal of the length-matching placement includes two parts: minimum length and matching length. So, the clock-aware length-matching placement of RSFQ circuits can be formulated as follows:

- Input:
  1) A path-balanced RSFQ circuit $G(V, E)$.
  2) An RSFQ cell library including physical and timing parameters of the logic gates, SPLs, and PTLs.
- Output:
  1) A legal y_v for $\forall v \in V_i, i \in [0, n]$.
  2) The clock distribution of the circuit.
- Constraints:
  1) Timing constraints defined by Equation (3).
  2) Overlap constraints: For two nodes u and v in the same column, $y_u + h_u \leq y_v$ if $y_u \leq y_v$.
  3) For every node $v \in V$, $y_v \geq 0$ and $y_v + h_v \leq H$, where H is the maximum height among all columns.
- Goal:
  Our goal is to minimize the total vertical wirelength (*TVWL*) of the placement, which is the sum of the vertical minimum length and the vertical matching length, formulated as:

$$TVWL = \sum_{\substack{v \in V_i \\ i \in [0,n]}} \left[l_v^{vc} + l_v^{mc} + \sum_{u \in FI(v)} \left(l_{u,v}^{vd} + l_{u,v}^{md} \right) \right], \tag{4}$$

where l_v^{vc} and l_v^{mc} are the vertical minimum length and vertical matching length of node v's clock connection, and $l_{u,v}^{vd}$ and $l_{u,v}^{md}$ are the vertical minimum length and vertical matching length of the data connection between u and v.

IV. JPlace

To solve the above RSFQ placement problem, we propose a clock-aware length-matching RSFQ placement framework with four steps: 1) initial placement; 2) clock distribution

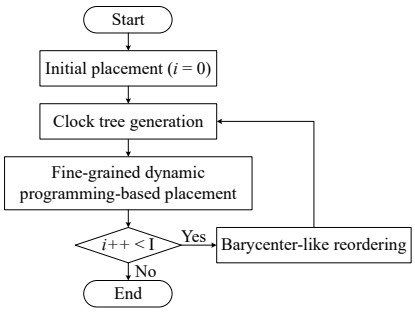


Fig. 3. The flow of JPlace, where I is the number of iterations.

generation; 3) fine-grained dynamic programming (DP)-based placement; and 4) barycenter-like reordering. The overall flow of our algorithm is summarized in Fig. 3. During the initial placement phase, we group all gates into columns according to their logical stage and assign their initial locations randomly. Then, a heuristic approach is employed to generate the clock distribution. After that, the DP-based placement minimizes the *TVWL* by considering both data and clock nets, with a focus on optimizing vertical minimum length and vertical matching length. Finally, the barycenter-like reordering stage further refines the placement result. Detailed explanations of these steps are provided in the following subsections.

A. Clock Distribution Generation

In our algorithm, a heuristic approach is proposed to generate the clock distribution. For each node v within a column, we first assign an index to it by sorting all nodes in that column based on their y_v values. Then, we keep track of the indices of its data inputs and utilize them as factors to determine the source of the clock pulse. For a target node v, we track all indices of $u \in FI(v)$. The source of the clock pulse for the target node v is determined as the node whose index is the average of indices in $FI(v)$. For instance, in Fig. 2, the data inputs of node g_4 are nodes g_1 and g_2 ($FI(g_4) = \{g_1, g_2\}$). The index of g_1 is 1, and the index of g_2 is 2. Consequently, the clock pulse of g_4 is derived from the node with an index of 1 (computed as $\lfloor (1+2)/2 \rfloor = 1$) in the former stage, which is g_1. As for g_5, it has only one data input, resulting in the clock pulse being sourced from the node with an index of $\lfloor 3/1 \rfloor = 3$, i.e., g_3.

B. Fine-grained DP-based Placement

After generating the clock distribution, we want to find a location for each node with minimum *TVWL*. However, finding the locations of nodes individually, without any constraint, can be a complicated problem. For a node in column V_i, there are at least $|V_i|!$ possible arrangements by just adjusting their orders. In a circuit with n logical stages, this becomes $\prod_{i}^{n} |V_i|!$, which is infeasible for large-scale circuits. Let

$$l_v^{sum} = l_v^{vc} + l_v^{mc} + \sum_{u \in FI(v)} (l_{u,v}^{vd} + l_{u,v}^{md}). \tag{5}$$

Then *TVWL* can be written as the sum of l_v^{sum} according to Equation (5). We observe that nodes within a single column contribute to *TVWL* independently, implying that the location of one node does not affect the others within the same column.

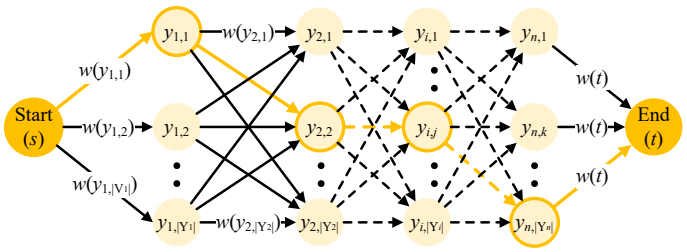


Fig. 4. Trellis diagram for the column-wise placement.

Based on this observation, we can transform the placement problem into a shortest path problem by introducing a fixed order assumption. This approach minimizes the sum of l_v^{sum} by identifying the path with the lowest cost on a graph.

Consequently, we propose a DP-based algorithm that enables effective and computationally efficient solution generation for the RSFQ placement problem. The DP-based placement algorithm assumes that nodes within a column are ordered in ascending order based on their y_v values. The algorithm performs DP column by column, starting from the bottom and moving upwards within each column. When updating the node locations within a column, it assumes that the locations of nodes in other columns remain fixed. We will illustrate the execution of the DP algorithm on the problem.

After sorting the nodes in V_n, they can be represented as $V_n' = \{v_1, v_2 \dots v_p \dots v_{|V_n|}\}$. When performing DP on the nodes in column n ($v_p \in V_n'$), each node maintains an exploration range of $[y_{v_p} - r, y_{v_p} + r]$, where r is a predefined hyperparameter. Besides, since the order of nodes is fixed, a node can only be placed in locations that can preserve enough space for other nodes within the range $[0, H]$. This can be expressed as the following constraint for all $v_p \in V_n'$:

$$\sum_{i=1}^{p-1} h_{v_i} \le y_{v_p} \le H - \sum_{i=p}^{|V_n|} h_{v_i}. \tag{6}$$

Then, the set Y_p of candidate locations for node v_p is given by:

$$Y_p = \{y_{p,m_p} | y_{p,m_p} \in [y_{v_p} - r, y_{v_p} + r] \cap [\sum_{i=1}^{p-1} h_{v_i}, H - \sum_{i=p}^{|V_n|} h_{v_i}]\}, \tag{7}$$

where $m_p = 1, 2, \dots, |Y_p|$. Assign y_{p,m_p} to gate v_p can ensure that there is always a feasible space for other gates in the same column to be placed, with the fixed-order constraint.

The DP-based algorithm generates a weighted directed graph, denoted as G_k, for each V_n', effectively converting the RSFQ placement problem into a shortest path problem. Shown in Fig. 4, the graph contains vertices $\{s, t\} \cap Y_1 \cap \dots \cap Y_{|V_n|}$. The vertex s connects to each vertex in Y_1, while all vertices in $Y_{|V_n|}$ connect to t. Besides, a vertex in Y_p is connected to a vertex in Y_{p+1} if and only if $y_{p,m_p} + h_p \le y_{p+1,m_{p+1}}$ which keeps the fixed order assumption. Each edge connected to y_{p,m_p} has a weight, denoted as $w(y_{p,m_p})$, representing the cost of moving to it. The value of $w(y_{p,m_p})$ depends on y_{p,m_p}. For columns with logical stages greater than 0, $w(y_{p,m_p}) = l_p^{sum}(y_{p,m_p})$, where $l_p^{sum}(y_{p,m_p})$ corresponds to l_v^{sum} when the location of v_p is y_{p,m_p}. For the column with a logical stage of 0, which

only contains PIs, $w(y_{p,m_p})$ is calculated as $\sum_{u \in FO(v)} l^d_{v,u}$, the sum of all vertical minimum length of its outputs. The edges connected to t have a weight of $w(t) = 0$. A path from s to t with a minimum cost, i.e., the sum of $w(y_{p,m_p})$, guides us to find the minimum *TVWL* for column n. The vertex along the path represents the location of each node v_p to get the minimum *TVWL*. Based on the above analysis, the state transition model of our DP-based algorithm can be formulated as follows:

$$dp[p][y_{p,m_p}] = \begin{cases} \min\limits_{y} \left(dp[p-1][y] + w(y_{p,m_p}), dp[p][y_{p,m_p}]\right), & p > 1, y + h_{p-1} < y_{p,m_p} \\ w(y_{p,m_p}), & p = 1 \end{cases}, \tag{8}$$

where dp array is used to record the partial sums of the cost. When $p = 1$, $dp[p][y']$ stores the value of $w(y_{p,m_p})$. For $p > 1$, $dp[p][y']$ stores the minimum subsum of $w(y_{p,m_p})$ considering the nodes from v_1 to v_p. So the minimum *TVWL* of this column corresponds to the minimal values in $dp[|V_k|][*]$. By backtracking the dp array, we can identify the placement associated with the minimum cost.

C. Barycenter-like Reordering

In the DP-based placement, the relative locations of nodes remain unchanged. However, it is important to acknowledge that this fixed order significantly limits the solution space. To address this challenge, we aim to design a method to find a new node order across all columns based on the obtained placement result. The new order can assist in achieving a placement result with reduced *TVWL* by DP. Evaluating the quality of a reordering outcome, specifically concerning *TVWL*, requires the completion of the reordering process and subsequent DP calculations, which can be time-consuming. So, it is more appropriate to establish a new target that is related to *TVWL* and can guide the reordering to minimize the *TVWL* wisely. According to Equations (4)–(5), *TVWL* is constructed with the minimum vertical minimum length and the vertical matching length, both of which are determined by the positions of nodes. This suggests that reordering nodes based on their length and positions could offer a promising approach. Therefore, we introduce a barycenter-like reordering algorithm, which enables the reordering of nodes within columns. The new location y_v for node v can be calculated as: $y_v = \frac{\sum w_{u,v} * y_u}{\sum w_{u,v}}$, where $w_{u,v}$ represents a weight related to the connection from node u to node v. A higher weight indicates that the location y_u of node u has a greater influence on the placement of node v, resulting in a tendency for node v to be placed closer to y_u in the subsequent update.

A straightforward method to select weight is to use $l^{vd}_{u,v} + l^{md}_{u,v}$ as $w_{u,v}$ for $u \in FI(V)$. In this situation, it updates the location of a gate based on all of its input connections, with the connection having the longest length exerting the most significant influence on the new position determination. However, this approach only captures a portion of the delay influence for a gate. Practically, it is the delay from the PI to a gate's input pins, rather than the delay from the preceding stage, that accurately defines a gate's delay. Given this understanding, we introduce a refined weighting strategy: for any $u \in FI(v)$, we employ the value representing the delay from the PI, passing through u and terminating at v, as the weight $w(u, v)$. This weighting strategy aids in identifying the input that serves as the critical path, allowing for the *TVWL* minimization from a broader, cross-column perspective. Consequently, gate v tends to be positioned closer to the gate with the greatest delay. By reducing this delay, other connections would require shorter vertical matching length to meet the timing constraints.

TABLE I
COMPARISON RESULTS BETWEEN JPLACE AND THE BASELINE.

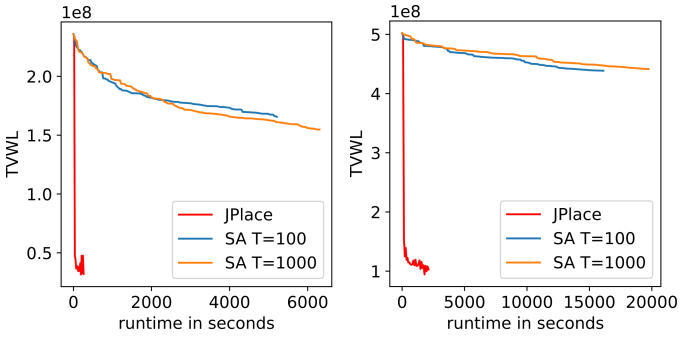
Circuits	Stages	Gates	JPlace / SA			
			T=100		T=1000	
			runtime	TVWL	runtime	TVWL
c432	39	1015	37.57%	71.36%	24.89%	72.02%
c499	14	604	21.75%	75.11%	18.07%	73.07%
c880	30	1376	42.93%	62.18%	33.11%	64.77%
c1355	14	635	33.23%	72.96%	21.27%	76.42%
c1908	24	1149	61.64%	62.96%	46.44%	63.09%
c3540	35	2279	24.23%	33.53%	18.75%	34.51%
c5315	31	5503	24.98%	35.21%	18.56%	34.84%
c6288	78	4765	22.39%	82.22%	17.01%	87.46%
c7552	50	7571	21.63%	27.62%	18.61%	28.75%
int2float	18	494	8.09%	64.56%	6.86%	63.31%
sin	171	16516	5.00%	19.36%	4.13%	20.70%
priority	217	17778	16.01%	60.24%	15.56%	60.19%
adder	257	49278	18.29%	28.04%	17.96%	28.34%
multiplier	259	77538	5.28%	46.24%	5.81%	47.07%
max	209	83933	13.13%	23.41%	10.72%	23.26%
Ave. Imp.	/	/	76.26%	49.00%	81.48%	48.15%

D. Time Complexity Analysis

Our algorithm's time complexity can be explained as follows. In the initial placement step, each node is assigned a location randomly, which takes $O(|V|)$ time. During clock distribution generation, each node records its index on its $FO(v)$ with an extra $O(|V|)$ space during sorting. Thus, finding the clock source can be done in $O(|V| \log |V| + |V|)$. Next, in the fine-grained DP-based placement, we introduce a hyperparameter r. According to Equation (8), the time complexity of this step can be estimated as $O(r^2|V|)$. Notably, setting a large value for r is unnecessary for achieving good performance. Following that, the barycenter-like reordering updates node locations based on their inputs. Since all nodes have at most three inputs, updating the location of a single node has a linear time complexity. Additionally, the sorting part of this phase requires $O(|V| \log |V|)$ time. Combining these factors, the time complexity for one iteration of the algorithm is $O(|V| + r^2|V| + |V| \log |V|)$. Since r is a small constant (set to 50 in our experiment), the overall time complexity can be simplified to $O(|V| \log |V|)$. Considering that the algorithm runs for I iterations (set to 100 in our experiment), the total time complexity of our algorithm is $O(I \cdot |V| \log |V|)$.

V. EXPERIMENTS

JPlace was implemented in Python programming language. All the experiments were performed on a machine equipped with an Intel(R) Core(TM) i7-11700 processor running on the Ubuntu 20.04 LTS in Window Subsystem for Linux (WSL), with 16 GB of RAM. In our evaluation, we utilized the circuits from the ISCAS85 and EPFL combinational benchmarks as our test cases. These circuits have varying numbers of logical



(a) sin (b) max

Fig. 5. Temporal evolution of *TVWL* for (a) sin and (b) max circuits using JPlace and the SA-based method with different initial annealing temperatures.

stages and gates, with the largest circuit having up to 209 logical stages and 83933 gates, while previous research finished placement on circuits that contain hundreds of gates. The sizes and timing parameters of the gates in the circuits were sourced from the ColdFlux logic cell library [16].

To evaluate the effectiveness of JPlace, we compared it against a modified version of Kito's SA-based method. To ensure a fair comparison, we made two primary modifications to Kito's SA-based method. First, we adjusted the objective function of Kito's original method to use *TVWL* as the metric. Secondly, we treated two gates within the same column as a pair and introduced changes to multiple pairs in a single placement perturbation to reduce the runtime of SA. For the remaining SA settings, we set the number of iterations to 10, using a cooling ratio of 0.95, and terminating annealing when the temperature drops below 0.01. We executed JPlace with $I = 100$ and $r = 50$. The algorithm recorded the best placement result, and if there was no improvement for five consecutive iterations, JPlace would terminate automatically.

Fig. 5 illustrates how the *TVWL* changes for the sin and max circuits during the execution of JPlace and the SA-based method. The initial annealing temperatures of the SA-based methods are set to 100 and 1000, respectively. It is evident that JPlace demonstrates rapid convergence towards superior results. Table I presents a comparison between JPlace and the SA-based method in terms of placement quality and runtime across various benchmarks. In the table, "Stages" and "Gates" denote the respective counts of logical stages and gates in each circuit. The final row presents the average improvement ratio between JPlace and SA results. On average, JPlace outperforms SA by 49.00% and 48.15% in terms of the *TVWL* and by 76.26% and 81.48% in the runtime, respectively.

VI. Conclusion

This paper introduced JPlace, a clock-aware length-matching method for RSFQ placement that considers both clock and data nets during the placement process to minimize the total vertical wirelength. JPlace comprises two key components: fine-grained dynamic programming-based placement and barycenter-like reordering. The fine-grained dynamic programming-based placement operates under an order fixed placement, which can minimize the goal of total vertical wirelength while maintaining acceptable time complexity. The barycenter-like reordering adjusts the order and positions of gates, exploring the search space for fine-grained dynamic programming-based placement in a heuristic manner. We evaluated the performance of JPlace on multiple circuits, using a SA-based method as the baseline. The experimental results demonstrated that JPlace outperformed SA on all benchmark circuits, with an average reduction of 49.00% and 48.15% in total vertical wirelength, while the runtime was improved by 76.26% and 81.48%, respectively.

Acknowledgments

This work was supported by National Key Research and Development Program (Grant No. 2022YFB4501404), the National Natural Science Foundation of China (Grant No. 62302477), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDA18000000), and the CAS Project for Youth Innovation Promotion Association.

References

[1] K. Likharev and V. Semenov, "RSFQ logic/memory family: a new josephson-junction technology for sub-terahertz-clock-frequency digital systems," *IEEE Trans. Appl. Supercond.*, vol. 1, no. 1, pp. 3–28, 1991.

[2] Y. Yamanashi, M. Tanaka, A. Akimoto, H. Park, Y. Kamiya, N. Irie, N. Yoshikawa, A. Fujimaki, H. Terai, and Y. Hashimoto, "Design and implementation of a pipelined bit-serial SFQ microprocessor, core1β," *IEEE Trans. Appl. Supercond.*, vol. 17, no. 2, pp. 474–477, 2007.

[3] Y. Ando, R. Sato, M. Tanaka, K. Takagi, N. Takagi, and A. Fujimaki, "Design and demonstration of an 8-bit bit-serial RSFQ microprocessor: Core e4," *IEEE Trans. Appl. Supercond.*, vol. 26, no. 5, pp. 1–5, 2016.

[4] R. Sato, Y. Hatanaka, Y. Ando, M. Tanaka, A. Fujimaki, K. Takagi, and N. Takagi, "High-speed operation of random-access-memory-embedded microprocessor with minimal instruction set architecture based on rapid single-flux-quantum logic," *IEEE Trans. Appl. Supercond.*, vol. 27, no. 4, pp. 1–5, 2017.

[5] K. Ishida, I. Byun, I. Nagaoka, K. Fukumitsu, M. Tanaka, S. Kawakami, T. Tanimoto, T. Ono, J. Kim, and K. Inoue, "SuperNPU: An extremely fast neural processing unit using superconducting logic devices," in *MICRO*, pp. 58–72, 2020.

[6] P. Gonzalez-Guerrero, M. G. Bautista, D. Lyles, and G. Michelogiannakis, "Temporal and SFQ pulse-streams encoding for area-efficient superconducting accelerators," in *ASPLOS*, p. 963–976, 2022.

[7] R. Fu, J. Huang, H. Wu, X. Ye, D. Fan, and T.-Y. Ho, "JBNN: A hardware design for binarized neural networks using single-flux-quantum circuits," *IEEE TC*, vol. 71, no. 12, pp. 3203–3214, 2022.

[8] N. Kito, K. Takagi, and N. Takagi, "A fast wire-routing method and an automatic layout tool for RSFQ digital circuits considering wire-length matching," *IEEE Trans. Appl. Supercond.*, vol. 28, no. 4, pp. 1–5, 2018.

[9] J.-T. Yan, "Fixed-order placement of pipelined architecture in rapid single-flux-quantum circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 10, pp. 1519–1532, 2022.

[10] K. Kitamura, T. Kawaguchi, and N. Takagi, "Wire length-matching aware placement method for rapid single flux quantum logic circuits," *IEEE Transactions on Applied Superconductivity*, vol. 33, no. 5, pp. 1–5, 2023.

[11] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," *IEEE Des. Test*, vol. 16, pp. 72–80, jul 1999.

[12] L. Amarù, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IWLS*, 2015.

[13] R. Fu, Z.-M. Zhang, G.-M. Tang, J. Huang, X.-C. Ye, D.-R. Fan, and N.-H. Sun, "Design automation methodology from RTL to gate-level netlist and schematic for RSFQ logic circuits," in *GLSVLSI*, p. 145–150, 2020.

[14] J.-T. Yan, "Tree-based clock distribution of multiple-stage pipelined architecture in rapid single-flux-quantum circuits," *IEEE TCAD*, vol. 41, no. 4, pp. 1090–1102, 2022.

[15] K. Gaj, E. G. Friedman, and M. J. Feldman, "Timing of multi-gigahertz rapid single flux quantum digital circuits," *J. VLSI Signal Process. Syst.*, vol. 16, no. 2–3, pp. 247–276, 1997.

[16] L. Schindler and T. Hall, "RSFQ cell library." https://github.com/sunmagnetics/RSFQlib. Version: 3.0, Release date: 21 March 2023.