양자 프로그래밍 툴 Depth 최적화

장경배

https://youtu.be/htIvll4htek

A Framework for Depth-Efficient Quantum Implementations of Linear Layers

Kyungbae Jang[1], Anubhab Baksi[2], and Hwajeong Seo[1]

[1] Hansung University, Seoul, South Korea
[2] Lund University, Sweden
starj1023@gmail.com, anubhab.baksi@eit.lth.se, hwajeong84@gmail.com

Abstract. Quantum depth plays a critical role in determining the performance of quantum implementations, yet quantum programming tools often fail to produce depth-optimal compilations of linear layers. In this work, we present a systematic and automated framework that reorders quantum gate sequences of linear layers to obtain depth-efficient quantum implementations. Our method consistently produces circuits with lower depth compared to prior implementations.
We apply the framework to a range of cryptographic operations, including the AES MixColumn, internal layers of the AES S-box, binary field squaring, and modular reduction in binary field multiplication. In all these cases, our method achieves meaningful reductions in quantum depth—for example, lowering the depth of the AES MixColumn and S-box circuits.
This work explores optimal quantum circuit designs for quantum programming tools, improves the accuracy of quantum resource estimation for cryptanalysis, and supports more realistic evaluations of post-quantum security.

Case Study: AES MixColumn

- 다음 두 논문에선 동일한 AES MixColumn (32×32)을 구현, 하지만 **다른 Depth를 보고**
  - PQCrypto'16 (M. Grassl et al, 전 NIST 표준)
    - 32 Qubits
    - 277 CNOTs
    - **39 Depth**
  - EUROCRYT'20 (S. Jaques et al. 현 NIST 표준) →
    - 32 Qubits
    - 277 CNOTs
    - **111 Depth**

```
def Euro_plu_mix(eng, word):
    CNOT | (word[7], word[0]);
    CNOT | (word[8], word[0]);
    CNOT | (word[9], word[0]);
    CNOT | (word[15], word[0]);
    CNOT | (word[17], word[0]);
    CNOT | (word[25], word[0]);
    CNOT | (word[9], word[1]);
    CNOT | (word[10], word[1]);
    CNOT | (word[18], word[1]);
    CNOT | (word[26], word[1]);
```

> *Note that [GLRS16] describes the same technique, while achieving a significantly smaller design than the one we obtain.*

Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Stein- wandt. Applying Grover's algorithm to AES: Quantum resource estimates. In Tsuyoshi Takagi, editor, Post-Quantum Cryptography, pages 29–43, Cham, 2016.
Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. Cryptology ePrint Archive, Report 2019/1146, 2019.

양자 프로그래밍 툴 Depth 최적화

- **하나의 큐빗의 반복적인 사용**은 다른 연산과의 병렬화를 방해
  ⇒ 양자 프로그래밍 툴이 최적의 Depth를 찾을 수 없음 (Qcrypton, ProjectQ, Q#, Qiksit, Cirq)

```
circuit = QuantumCircuit(9, 9)

# Depth Test
circuit.cx(0, 8)
circuit.cx(0, 4)
circuit.cx(0, 5)
circuit.cx(0, 6)
circuit.cx(0, 7)

circuit.cx(1, 4)
circuit.cx(1, 5)
circuit.cx(1, 6)
circuit.cx(1, 7)

circuit.cx(2, 4)
circuit.cx(2, 5)
circuit.cx(2, 6)
circuit.cx(2, 7)
circuit.cx(2, 8)

circuit.x(2)
circuit.x(4)
circuit.x(6)
circuit.x(8)
```

Qiskit Information	
Qubits	9
X	4
CX	14
Circuit Depth	9
CX Depth	8

양자 프로그래밍 툴 Depth 최적화

- 하나의 큐빗의 **반복적인 사용을 회피: Compiler-friendly**
  ⇒ 양자 프로그래밍 툴이 더 낮은 Depth를 찾음 (9 → 6)
  (암호 안전성 분석 측면에서, 올바른 복잡도를 도출하는 것 또한 유의미함)

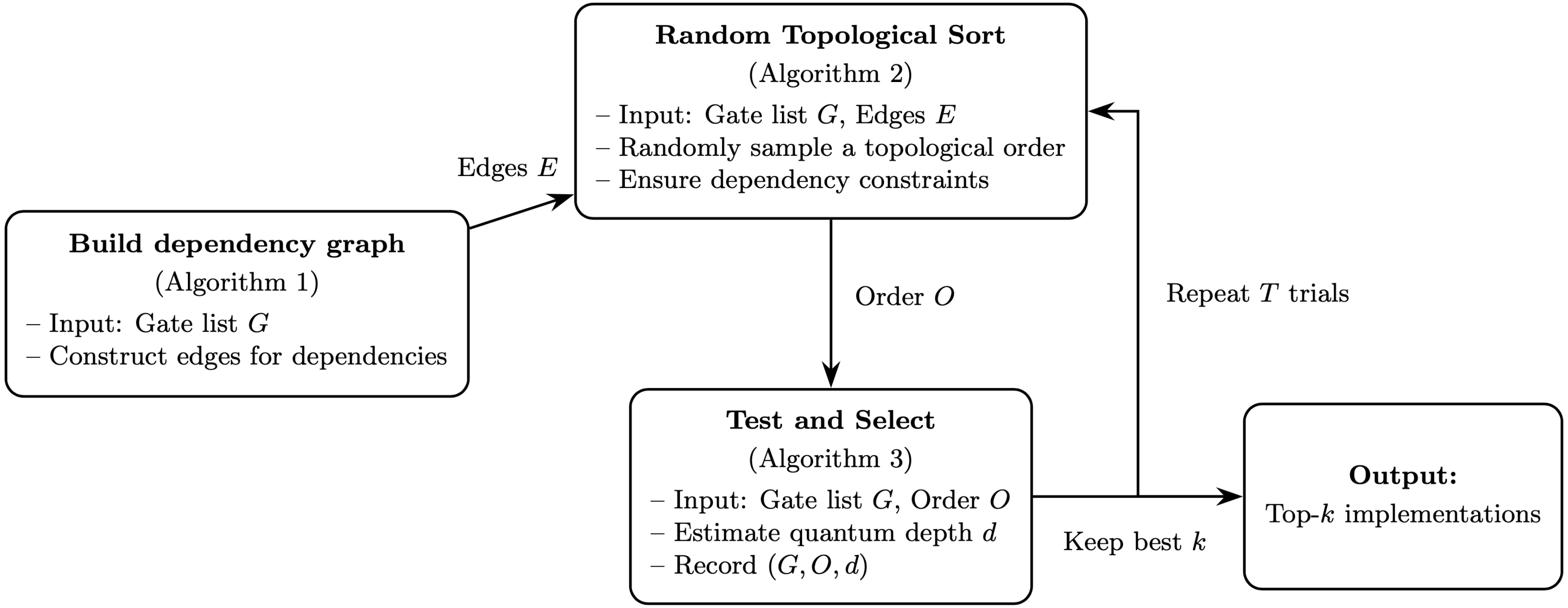
Reordered

```
# Depth Test (Reordered)
circuit.cx(0, 5)
circuit.cx(1, 4)
circuit.x(6)
circuit.cx(2, 7)
circuit.cx(0, 7)
circuit.cx(2, 6)
circuit.cx(1, 7)
circuit.cx(2, 5)
circuit.x(8)
circuit.cx(0, 4)
circuit.cx(2, 4)
circuit.cx(2, 8)
circuit.cx(0, 6)
circuit.x(4)
circuit.cx(0, 8)
circuit.cx(1, 6)
circuit.x(2)
circuit.cx(1, 5)
```

Qiskit Information	
Qubits	9
X	4
CX	14
Circuit Depth	9 → 6
CX Depth	8 → 6

Compiler-friendly 최적화 프레임워크

- 양자 프로그래밍 툴 용 Compiler-friendly 최적화 프레임워크 개발

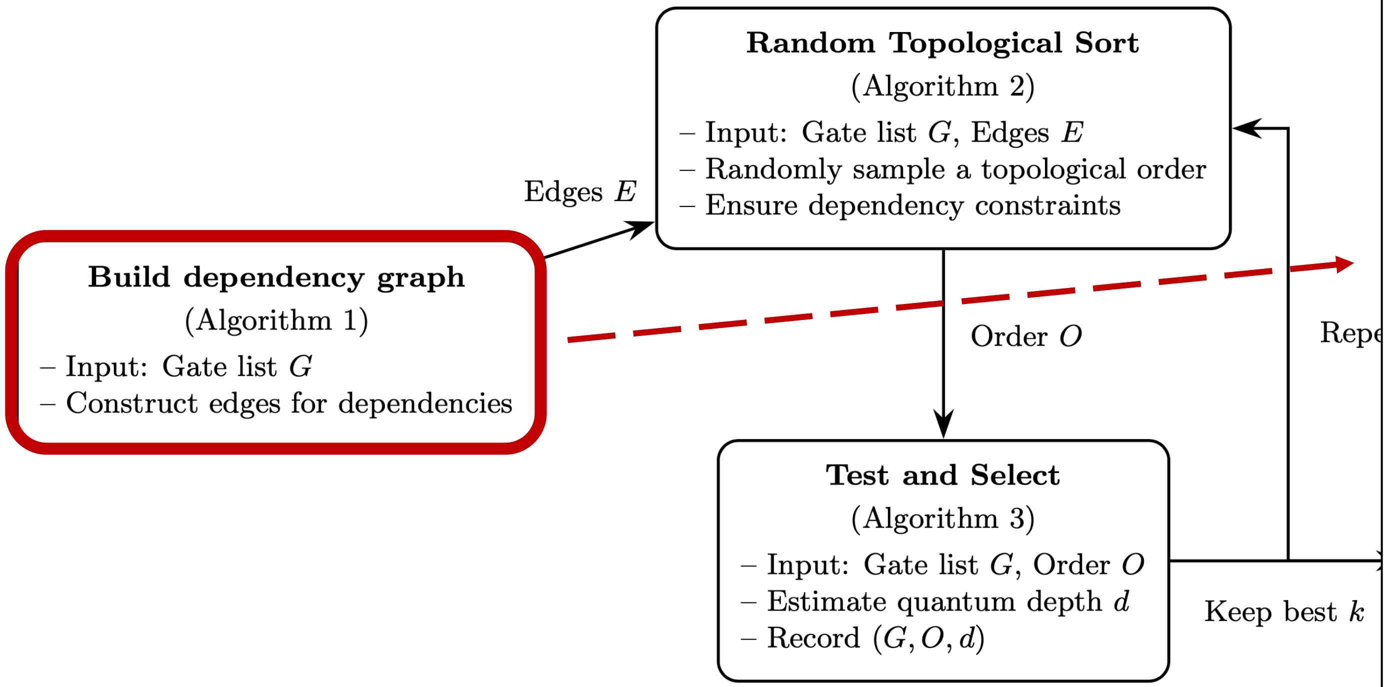


< 프레임워크 Overview >

Kyungbae Jang, Anubhab Baksi, and Hwajeong Seo “A Framework for Depth-Efficient Quantum Implementations of Linear Layers”, ePrint, archive 2025.

Compiler-friendly 최적화 프레임워크

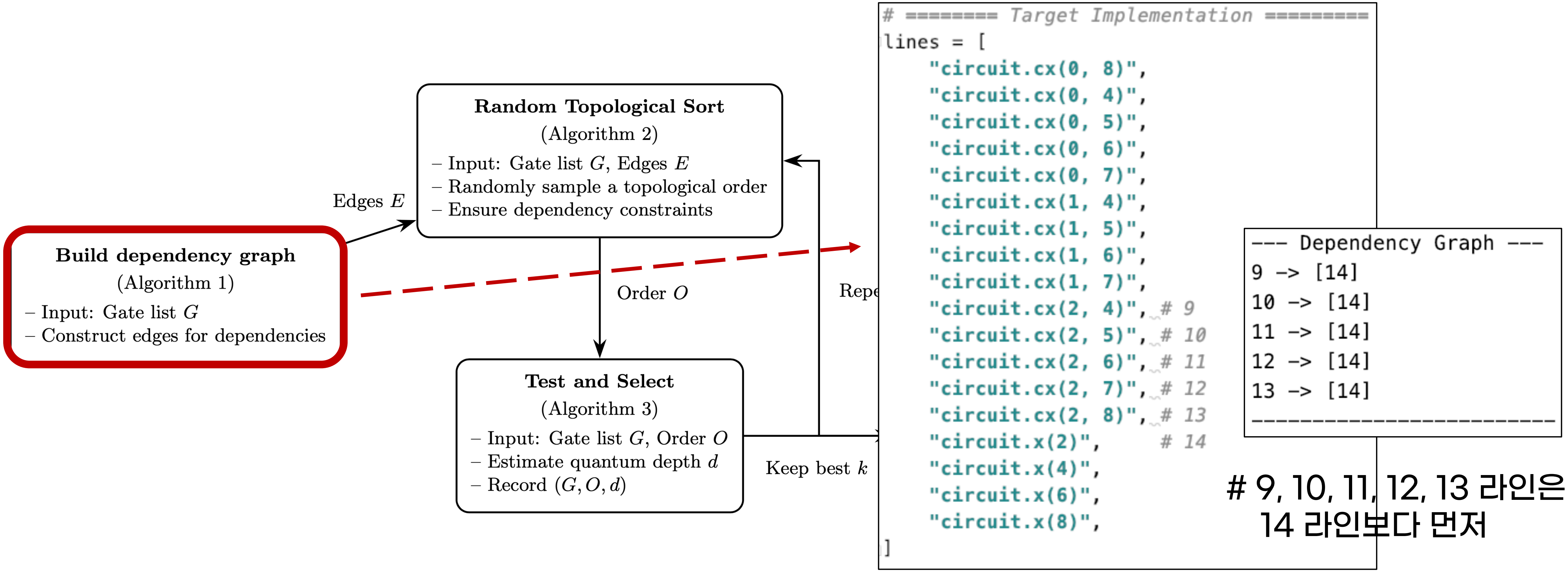
- 양자 프로그래밍 툴 용 Compiler-friendly 최적화 프레임워크 개발
  - 최적화하고 싶은 **양자 구현 입력**




```
# ======== Target Implementation =========
lines = [
    "circuit.cx(0, 8)",
    "circuit.cx(0, 4)",
    "circuit.cx(0, 5)",
    "circuit.cx(0, 6)",
    "circuit.cx(0, 7)",
    "circuit.cx(1, 4)",
    "circuit.cx(1, 5)",
    "circuit.cx(1, 6)",
    "circuit.cx(1, 7)",
    "circuit.cx(2, 4)",
    "circuit.cx(2, 5)",
    "circuit.cx(2, 6)",
    "circuit.cx(2, 7)",
    "circuit.cx(2, 8)",
    "circuit.x(2)",
    "circuit.x(4)",
    "circuit.x(6)",
    "circuit.x(8)",
]
```

Compiler-friendly 최적화 프레임워크

- 양자 프로그래밍 툴 용 Compiler-friendly 최적화 프레임워크 개발
  - 해당 양자 구현의 연산들 간의 **의존 그래프** 생성



9, 10, 11, 12, 13 라인은
14 라인보다 먼저

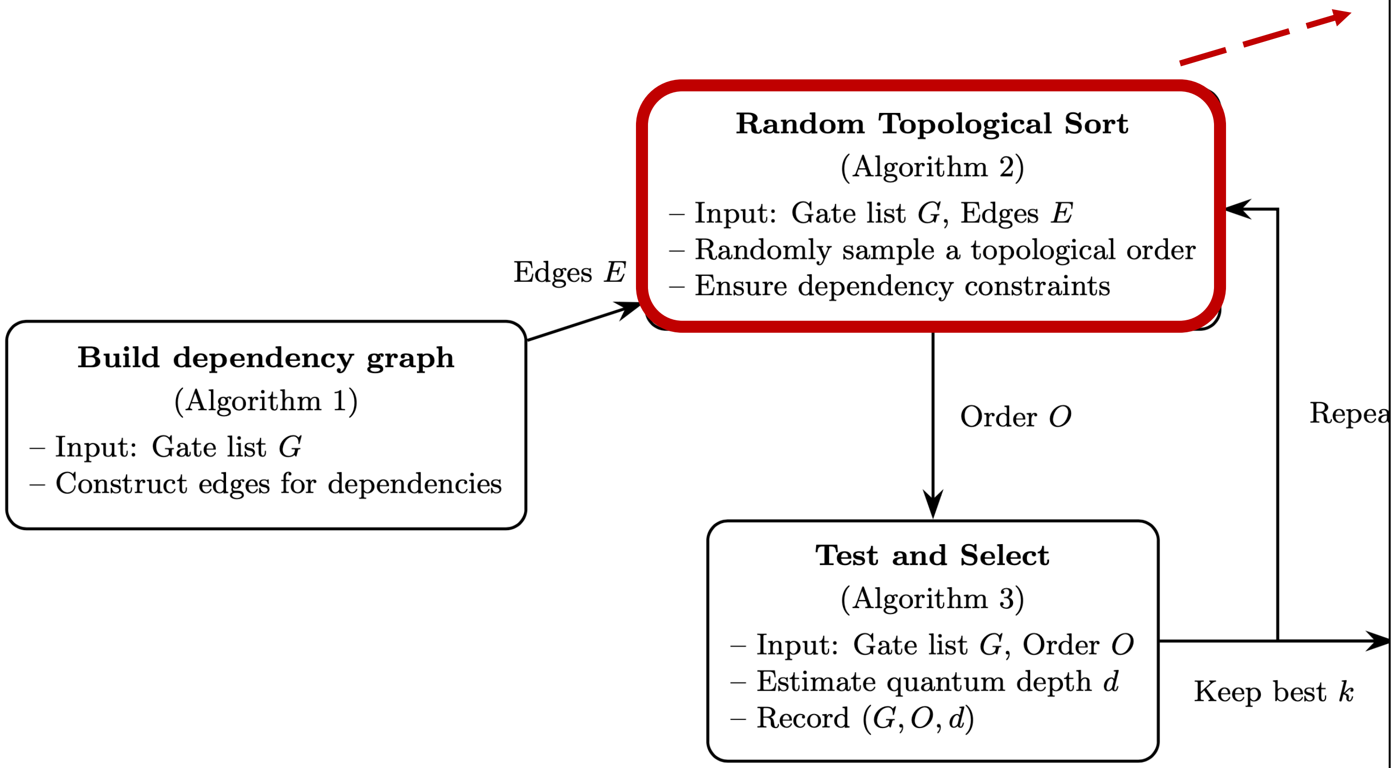
Algorithm 1: Build dependency graph

Input: Gate list $G = \{g_1, \ldots, g_n\}$ with $g = \texttt{CNOT}(ctrl, tgt)$ or $\texttt{X}(tgt)$
Output: Directed edge set $E \subseteq \{(i \to j) \mid 1 \le i < j \le n\}$

$E \leftarrow \emptyset$
for each qubit label q: $Writes[q] \leftarrow [\,]$, $Reads[q] \leftarrow [\,]$
for $i \leftarrow 1$ to n **do** ▷ collect per-qubit reads/writes
  if g_i is $\texttt{CNOT}$ **then**
    $Reads[ctrl(g_i)]$.append(i)
    $Writes[tgt(g_i)]$.append(i)
  else if g_i is $\texttt{X}$ **then**
    $Writes[tgt(g_i)]$.append(i)
  end if
end for
for $j \leftarrow 1$ to n **do** ▷ RAW: all prior writes → current read
  if g_j is $\texttt{CNOT}$ **then**
    $c \leftarrow ctrl(g_j)$
    $P \leftarrow \{\, i \in Writes[c] \mid i < j \,\}$
    if $P \neq \emptyset$ **then**
      $E \leftarrow E \cup \{\, (i \to j) \mid i \in P \,\}$
    end if
  end if
end for
for $j \leftarrow 1$ to n **do** ▷ WAR: all prior reads → current write
  if g_j is $\texttt{CNOT}$ **or** $\texttt{X}$ **then**
    $q \leftarrow tgt(g_j)$
    $R \leftarrow \{\, i \in Reads[q] \mid i < j \,\}$
    if $R \neq \emptyset$ **then**
      $E \leftarrow E \cup \{\, (i \to j) \mid i \in R \,\}$
    end if
  end if
end for
return E

Compiler-friendly 최적화 프레임워크

- 양자 프로그래밍 툴 용 Compiler-friendly 최적화 프레임워크 개발
  - 생성된 의존 그래프 기반 **랜덤 Reordering**



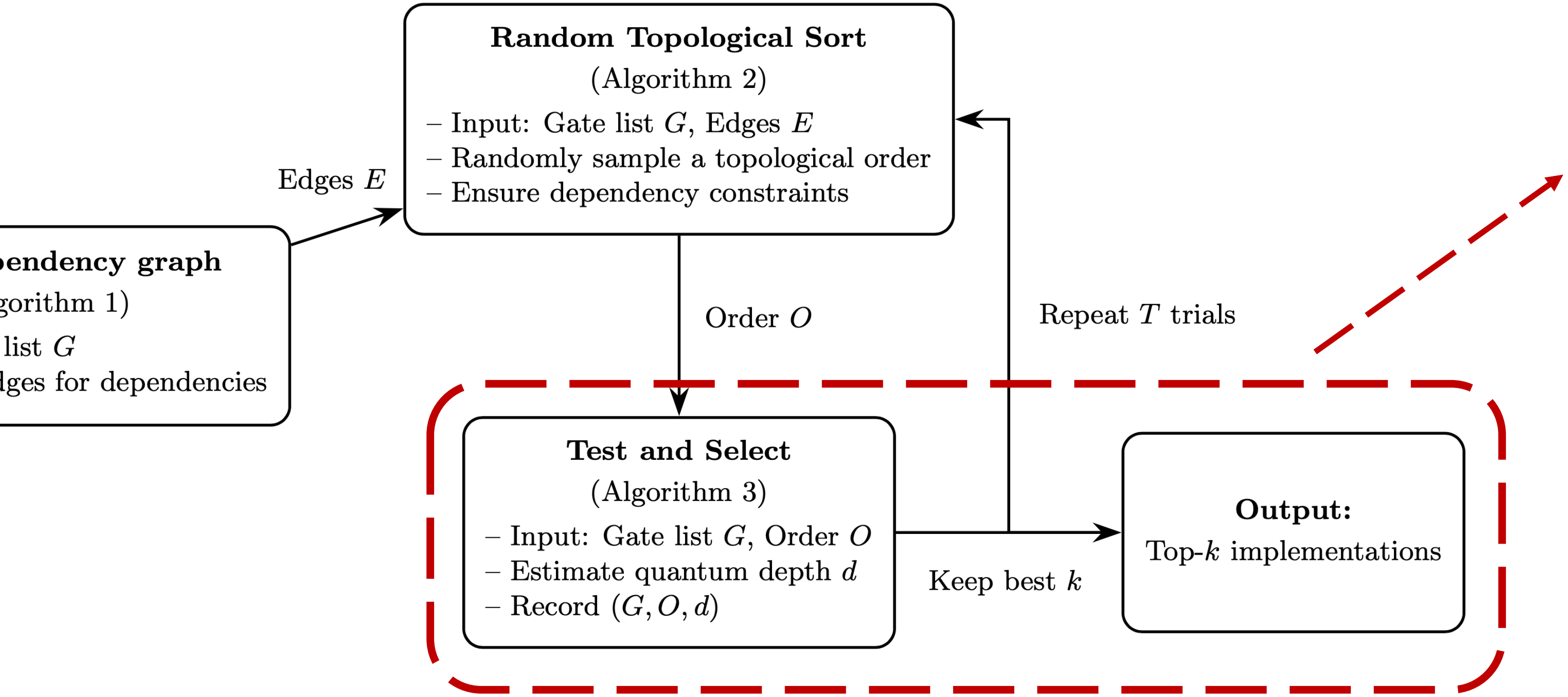
Algorithm 2: Randomized topological sort

Input: Gate list $G = \{g_1, \ldots, g_n\}$, dependency edges E
Output: A randomized topological order *Order* or `None`

```
Compute in-degrees deg[j] for all 1 ≤ j ≤ n
Pool ← { j | deg[j] = 0 },    Order ← [ ]
while Pool ≠ ∅ do
    Randomly shuffle Pool
    u ← pop one from Pool
    append u to Order
    for each (u → w) ∈ E do
        deg[w] ← deg[w] − 1
        if deg[w] = 0 then
            add w to Pool
        end if
    end for
end while
if |Order| = n then
    return Order
else
    return None
end if
```

Compiler-friendly 최적화 프레임워크

- 양자 프로그래밍 툴 용 Compiler-friendly 최적화 프레임워크 개발
  - Reorder된 구현의 **Depth 추정** 및 **Top-k 분류** (1000번 시도, $k = 3$)




```
Collected 1000 valid orders (trials=1000).

=== Top Reordered Schedules ===

[1] Depth = 6          [2] Depth = 6          [3] Depth = 6
circuit.cx(2, 4)       circuit.cx(2, 7)       circuit.cx(0, 4)
circuit.cx(2, 8)       circuit.cx(2, 8)       circuit.cx(2, 5)
circuit.cx(0, 7)       circuit.cx(0, 4)       circuit.cx(1, 5)
circuit.x(8)           circuit.cx(1, 5)       circuit.cx(2, 6)
circuit.cx(0, 5)       circuit.cx(1, 6)       circuit.x(4)
circuit.x(6)           circuit.cx(0, 5)       circuit.cx(0, 7)
circuit.cx(2, 7)       circuit.cx(2, 6)       circuit.cx(2, 8)
circuit.cx(1, 6)       circuit.cx(0, 7)       circuit.cx(1, 4)
circuit.cx(0, 4)       circuit.cx(1, 4)       circuit.cx(1, 7)
circuit.cx(1, 5)       circuit.cx(0, 6)       circuit.cx(0, 5)
circuit.cx(2, 6)       circuit.x(6)           circuit.x(6)
circuit.cx(0, 6)       circuit.cx(2, 4)       circuit.cx(2, 4)
circuit.cx(1, 4)       circuit.cx(1, 7)       circuit.cx(2, 7)
circuit.cx(1, 7)       circuit.x(4)           circuit.cx(0, 6)
circuit.cx(0, 8)       circuit.cx(2, 5)       circuit.x(8)
circuit.x(4)           circuit.x(2)           circuit.cx(1, 6)
circuit.cx(2, 5)       circuit.x(8)           circuit.cx(0, 8)
circuit.x(2)           circuit.cx(0, 8)       circuit.x(2)
```

Algorithm 3: Test and select

Input: Gate list G, sample budget T, top-k
Output: Top-k schedules with minimum depth

$E \leftarrow$ **Build dependency graph**(G)
$Seen \leftarrow \emptyset$ ▷ set of visited orders
$Cand \leftarrow \emptyset$ ▷ set of candidate schedules
for $t \leftarrow 1$ to T **do**
    $O \leftarrow$ **Randomized topological sort**(E)
    if $O =$ `None` **or** $O \in Seen$ **then**
        continue
    end if
    $Seen \leftarrow Seen \cup \{O\}$
    $d \leftarrow$ **Estimate depth**(G, O) ▷ e.g., `depth_of_dag`
    $Cand \leftarrow Cand \cup \{(O, d)\}$
end for
Sort $Cand$ by depth in ascending order
return first k elements of $Cand$

Real-World Examples

- **실제 양자 회로 구현**들을 대상으로 테스트
  - 선형 연산이 아니더라도, **내부 작은 선형 연산들**을 대상으로 적용하여 최적화 가능
    ⇒ AES S-box, Multiplication (모듈러 reduction)

Linear layer	Size (matrix)	Source	#Qubit	Full depth
AES MixColumn	32×32	Jaques et al. [9]	32*	111
		This paper		**81**
		Liu et al. [12]	135	13
		This paper		**9**
		Shi et al. [14]	131	14
		This paper		**9**
		Sun et al. [17]	122	18
		This paper		**11**
		Li et al. [10]	137	11
		This paper		**8**
		Lin et al. [11]	123	16
		This paper		**11**
AES S-box (internal linear layers)	-	Jang et al. [6]	76	67
		This paper		**64**
Squaring ($\mathbb{F}_{2^{16}}$)	16×16	J^+ [8] (Naïve)	32	22
		J^+ [8] (Compiler-friendly)		14
		This paper		**11**
Squaring ($\mathbb{F}_{2^{127}}$)	127×127	J^+ [8] (Naïve)	254	175
		J^+ [8] (Compiler-friendly)		125
		This paper		**125**
Multiplication ($\mathbb{F}_{2^{16}}$)	-	J^+ [7] (adopted in [8])	243	43
		This paper		**41**
Multiplication ($\mathbb{F}_{2^{163}}$)	-	J^+ [7] (adopted in [8])	13161	66
		This paper		**65**

Best AES S-box

감사합니다