A Structured Method for the Compilation of QAOA Circuits in Quantum Computing

June 29, ND-MIT QuCS Lecture Series

Zheng (Eddy) Zhang
Rutgers University

Brief Self Introduction

Zheng (Eddy) Zhang

- Associate Professor at Rutgers University, since 2018.
- Assistant Professor at Rutgers University, 2012 - 2018.
- PhD from the College of William and Mary, 2012
- Worked at and visited different places: Microsoft Research, PNNL, CMU at Silicon Valley, CWI at Amsterdam, USTC China, and etc.
- eddy.zhengzhang@gmail.com (Best way to reach me)
- people.cs.rutgers.edu/~zz124/

Research (and teaching):

- ***Central tenet: compiler techniques for emerging computing architectures***
- Quantum computing: hardware mapping and scheduling, fault tolerance, error mitigation, quantum optimal control and pulse generation.
- GPU accelerators: locality-aware optimizations and program transformations, binary instrumentation, information flow tracking, and etc.
- Shared memory multi-core processors: cache hierarchies, NUMA, CMP, and etc.
- Graph-theoretic models: graph partition, path finding, A* models, and etc.
- Markov chains, performance evaluation, and computer system characterization

Outline

- **Introduction**
- **The QAOA Mapper**
- **Experiment Results**
- **The Generic Mapper if time permitted**

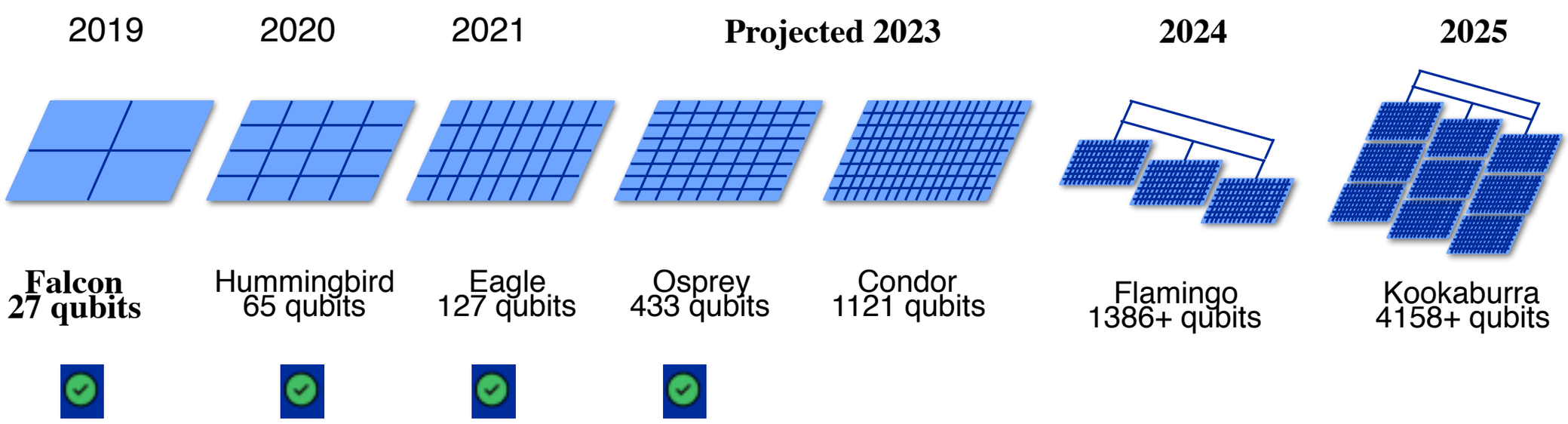
This talk is mainly based on our recent paper: "Exploiting the Regular Structure of Modern Quantum Architectures for Compiling and Optimizing Programs with Permutable Operators" that is accepted to ASPLOS'24.

The arXiv version is here: https://arxiv.org/abs/2112.06143

Quantum Computing is Getting Real

"Now is a privileged time in the history of science and technology, as we are witnessing the opening of the NISQ era (where NISQ = noisy intermediate scale quantum)."

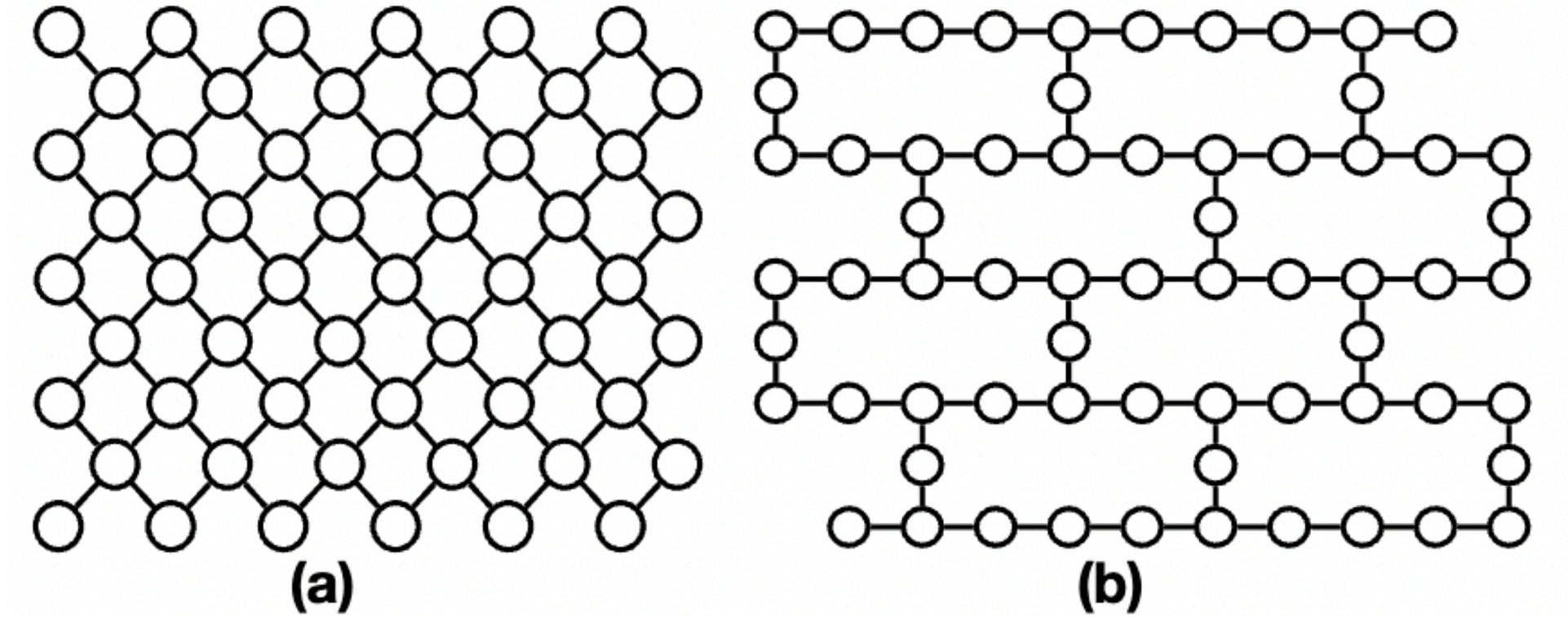
— John Preskill, Caltech



The evolution of IBM's quantum processors.

Regularity in Qubit Connectivity

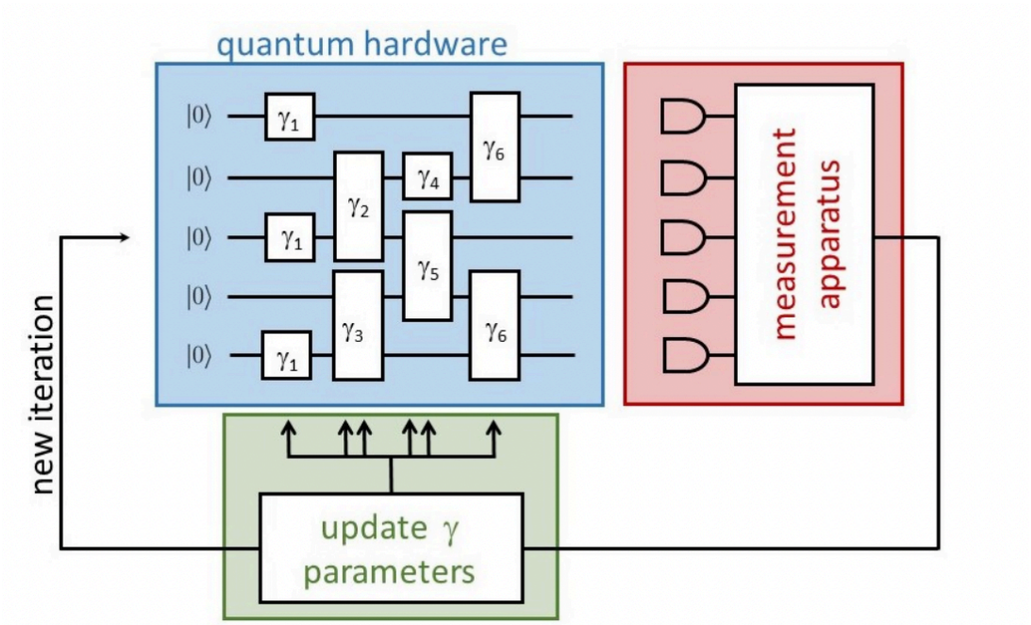
- **Google's rotated lattice and IBM's heavy-hex architecture**



Google **IBM**

The QAOA Mapping Problem

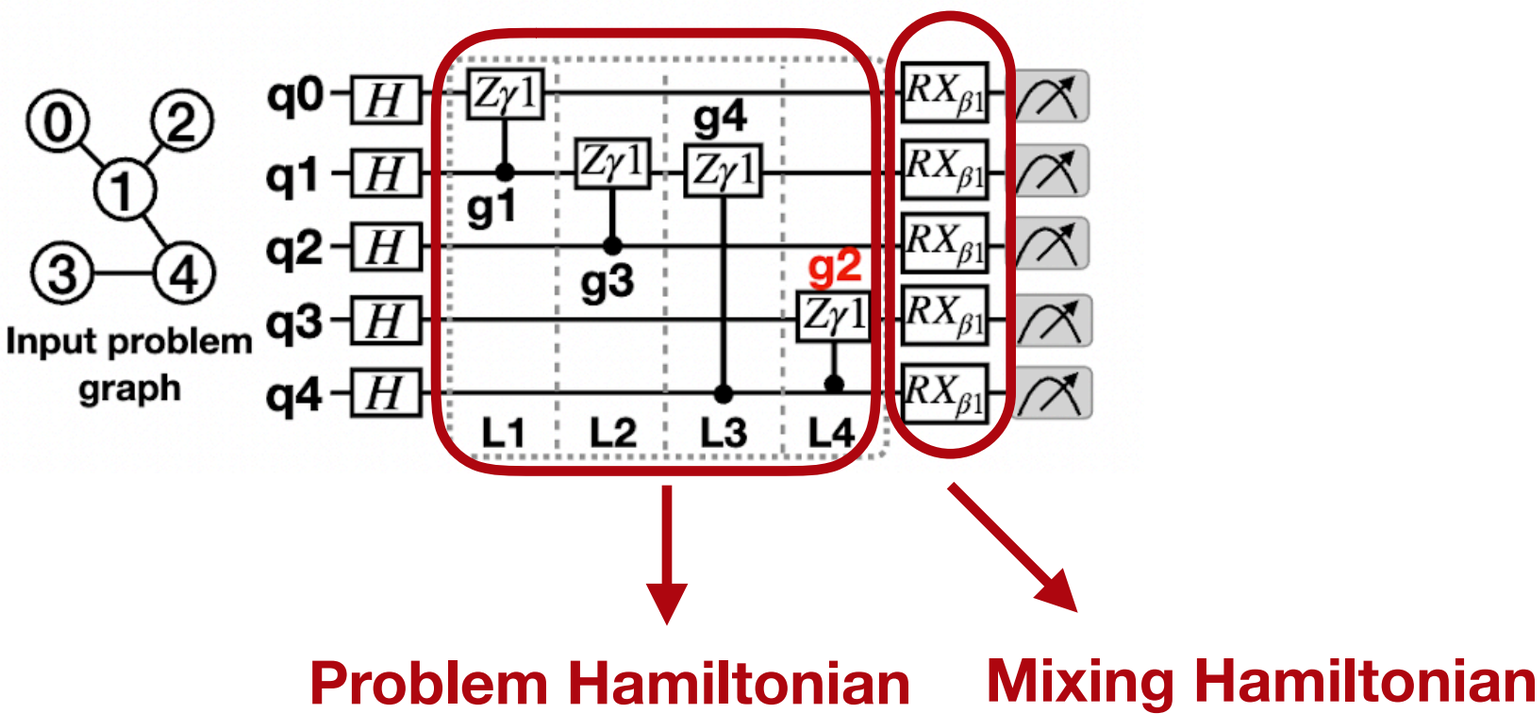
- **QAOA is a hybrid quantum-classical parameterized algorithms that exploit the strengths of quantum computers to solve constraint satisfaction problems (CSPs)**
  - Knapsack
  - Traveling salesman
  - Graph coloring
  - MAX-SAT
  - **MAX-CUT**



Guerreschi and Smelyanskiy, “Practical optimization for hybrid quantum-classical algorithms”, 2017

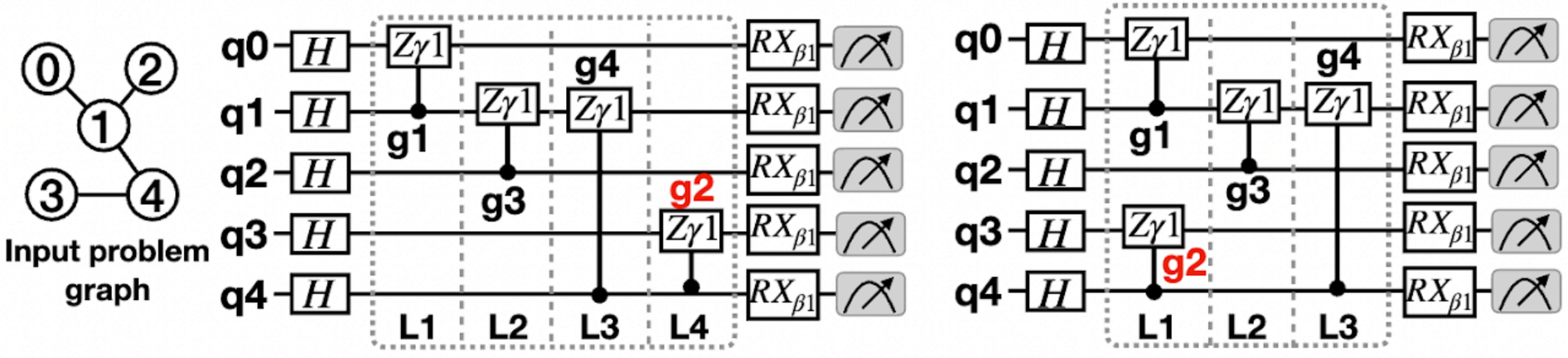
QAOA Optimization Opportunities

- **CPHASE gates are diagonal unitary gates. They commute.**
- **This also holds for Hamiltonian simulation in the 2-local case.**



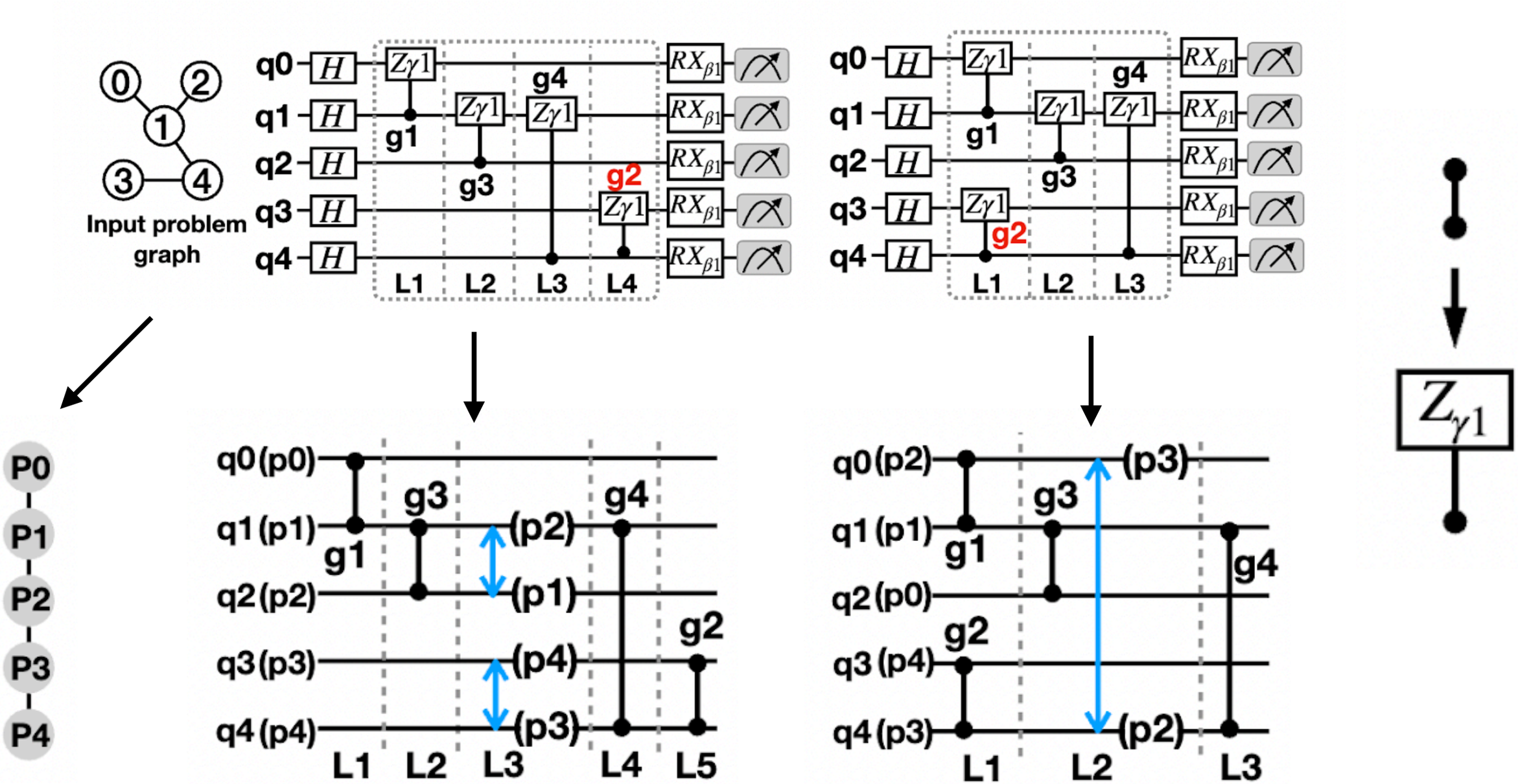
QAOA Optimization Opportunities

- **CPHASE gates are diagonal unitary gates. They commute.**
- **This also holds for Hamiltonian simulation in the 2-local case.**



QAOA Optimization Opportunities

- **CPHASE gates are diagonal unitary gates. They commute.**
- **This also holds for Hamiltonian simulation in the 2-local case.**

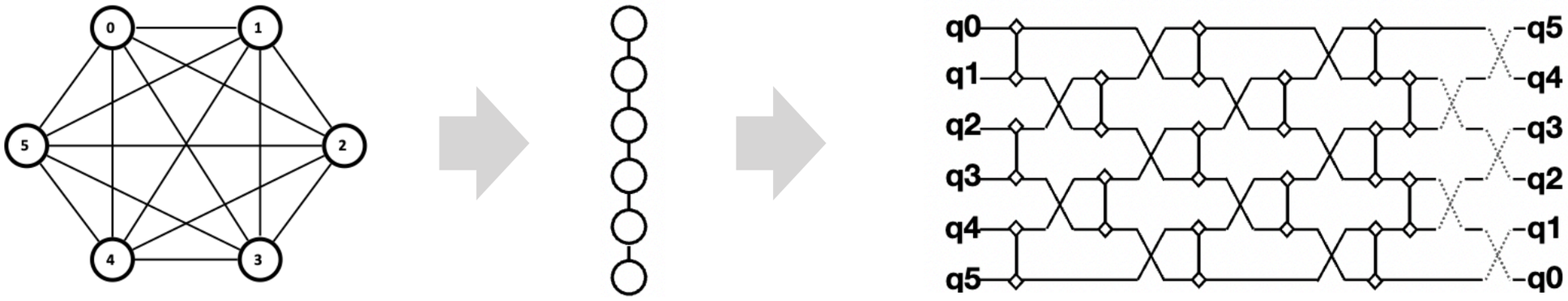


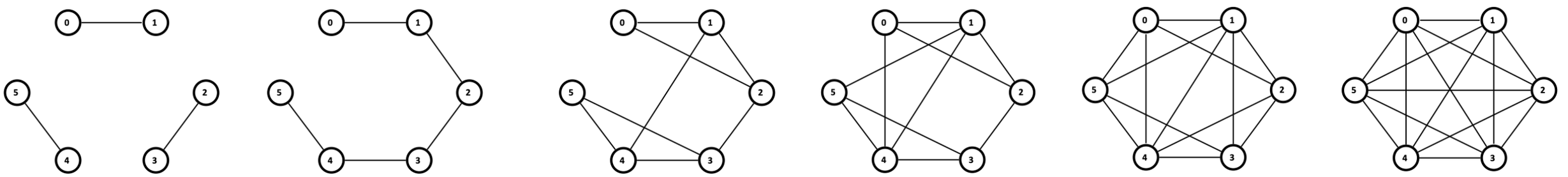
Main Inspirations That Lead to Our QAOA Mapping Approach

1. If we can solve a **n-clique QAOA circuit** efficiently, we can solve **any sub-circuit** of the n-clique circuit efficiently.
2. There is a famous **permutation network** result by Abraham Waksman that shows using nearest-neighbor binary cell alone can help achieve any permutation in linear time. We suspect that there is a similar result for QAOA's mapping on a quantum architecture.
3. A structured solution for **a small clique circuit** might be generalizable. But the condition is that the clique should be large enough for such a structured pattern to emerge. So we designed **an optimal solver** that can find solutions for reasonably sized circuits.

Main Result I — Solution for The Linear Architecture

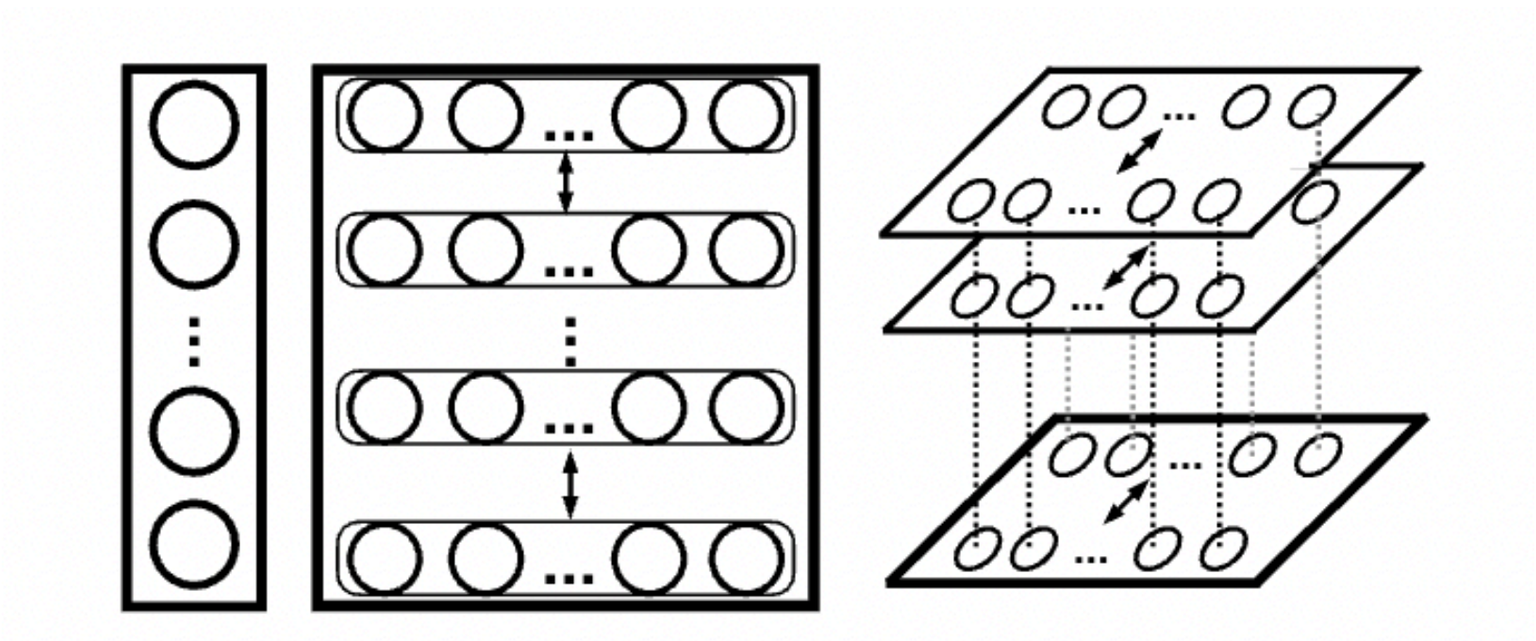
- **The input is a clique graph**





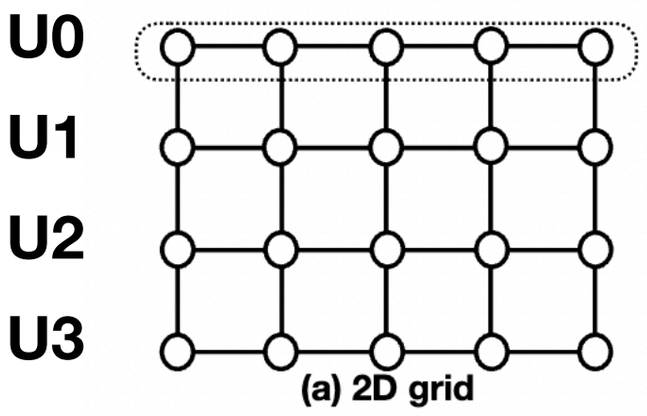
Main Result II — The Regular 2D Grid

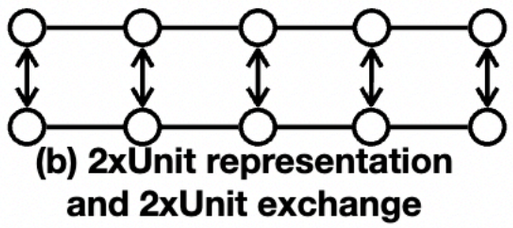
- **Our Divide and Conquer Approach**
  - We divide qubits into units.
  - Then we ensure all-to-all interaction within each unit,
  - and the bipartite all-to-all interactions between every two units.
  - This will eventually guarantee all-to-all interaction between all qubits.



Main Result II — The Regular 2D Grid

- **Units and Unit-exchange**

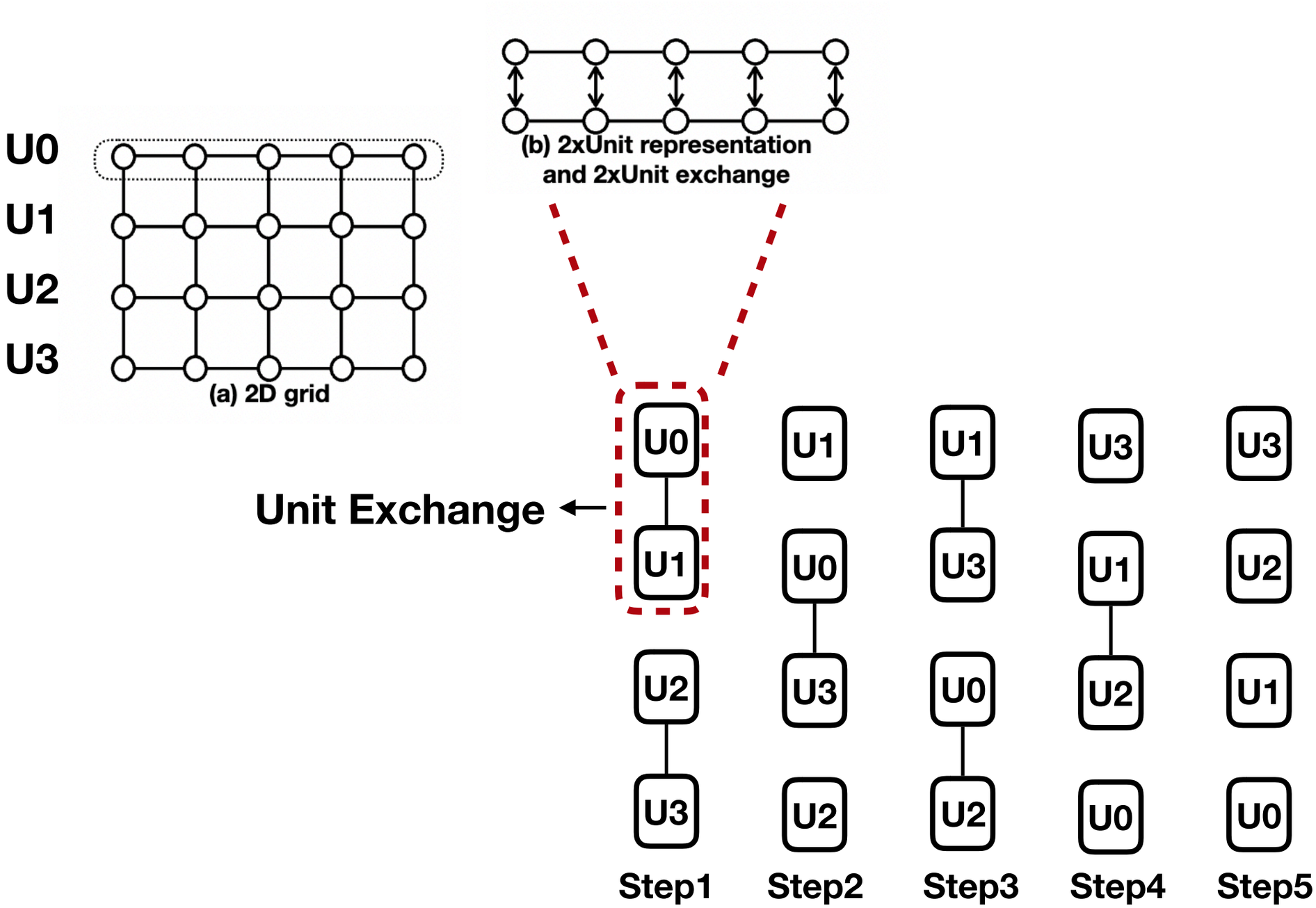




Mimicking the behavior of the linear architecture as if Ui is a single node.

Main Result II — The Regular 2D Grid

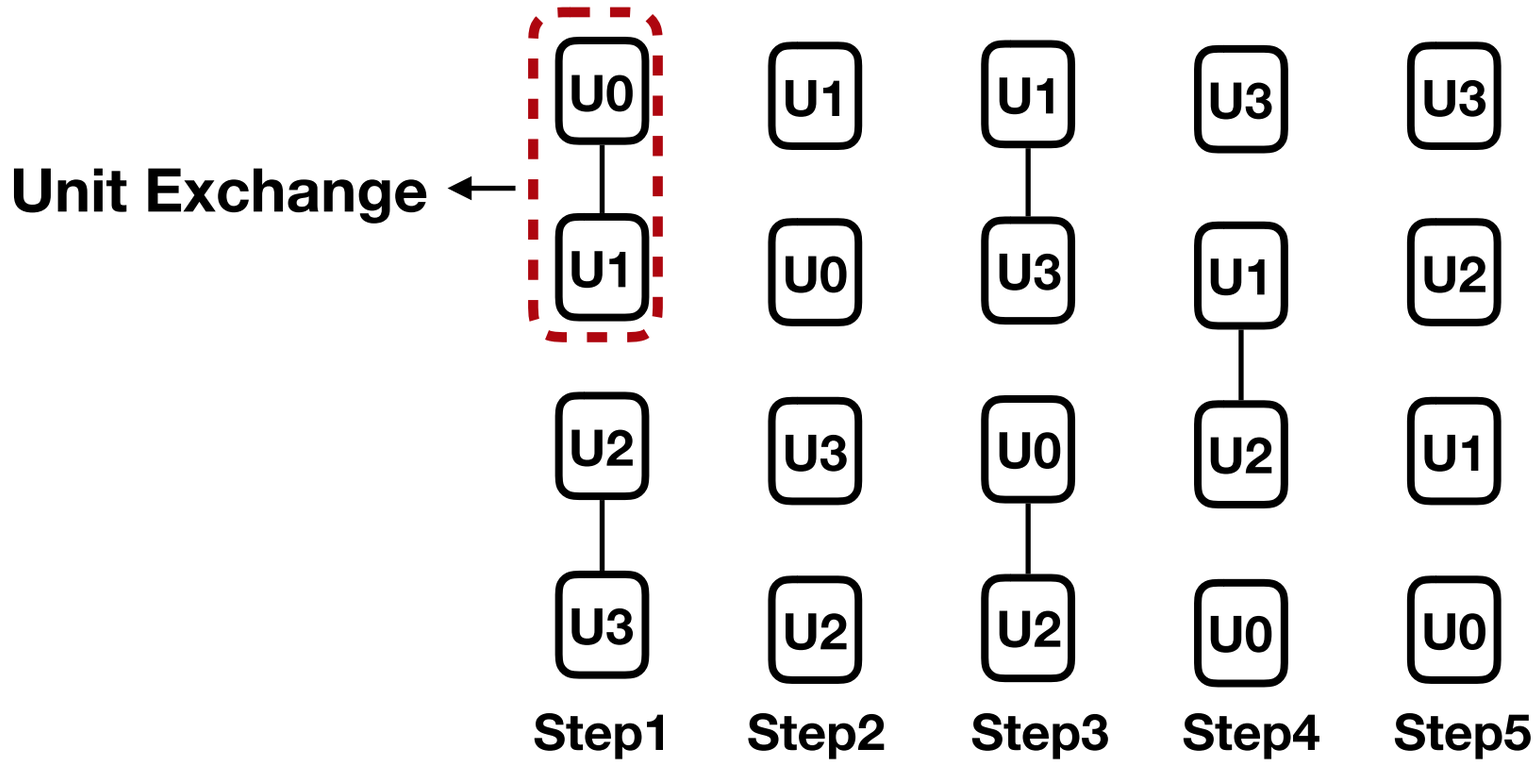
- **Units and Unit-exchange**



Mimicking the behavior of the linear architecture as if Ui is a single node.

Main Result II — The Regular 2D Grid

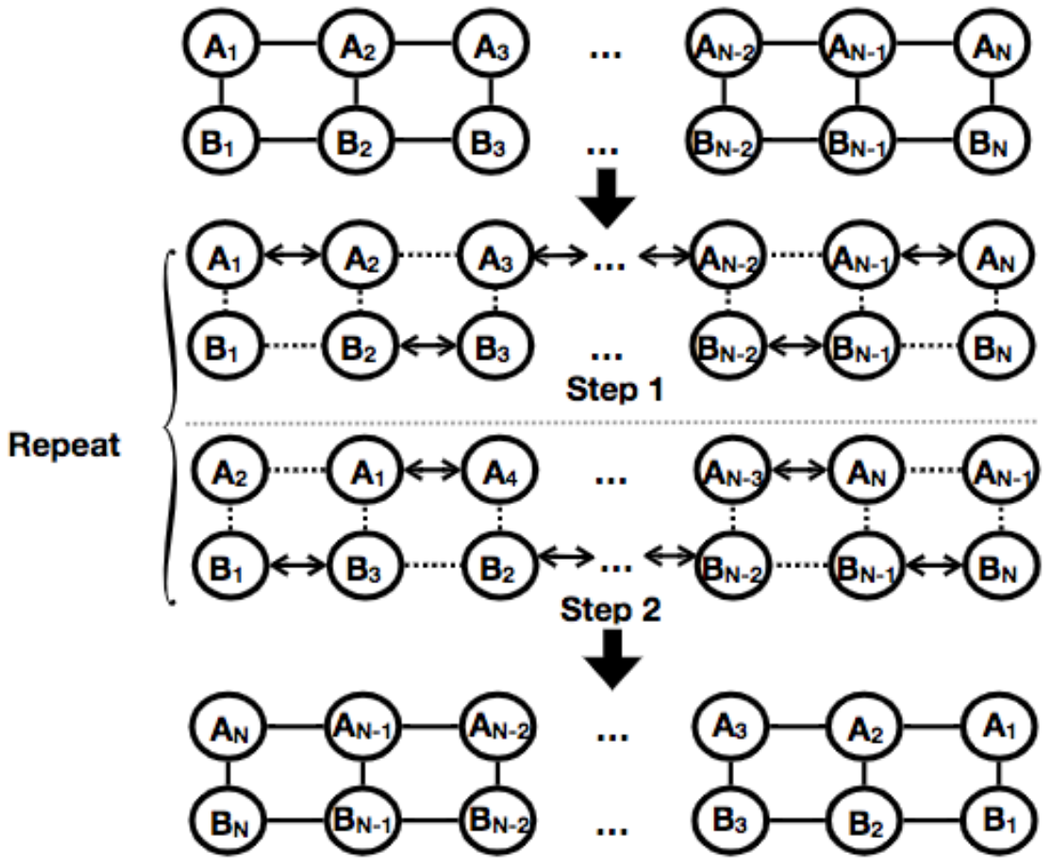
- Units and Unit-exchange
- **Each units gets to be the nearest neighbor of every other unit once**



Mimicking the behavior of the linear architecture as if Ui is a single node.

Main Result II — The Regular 2D Grid

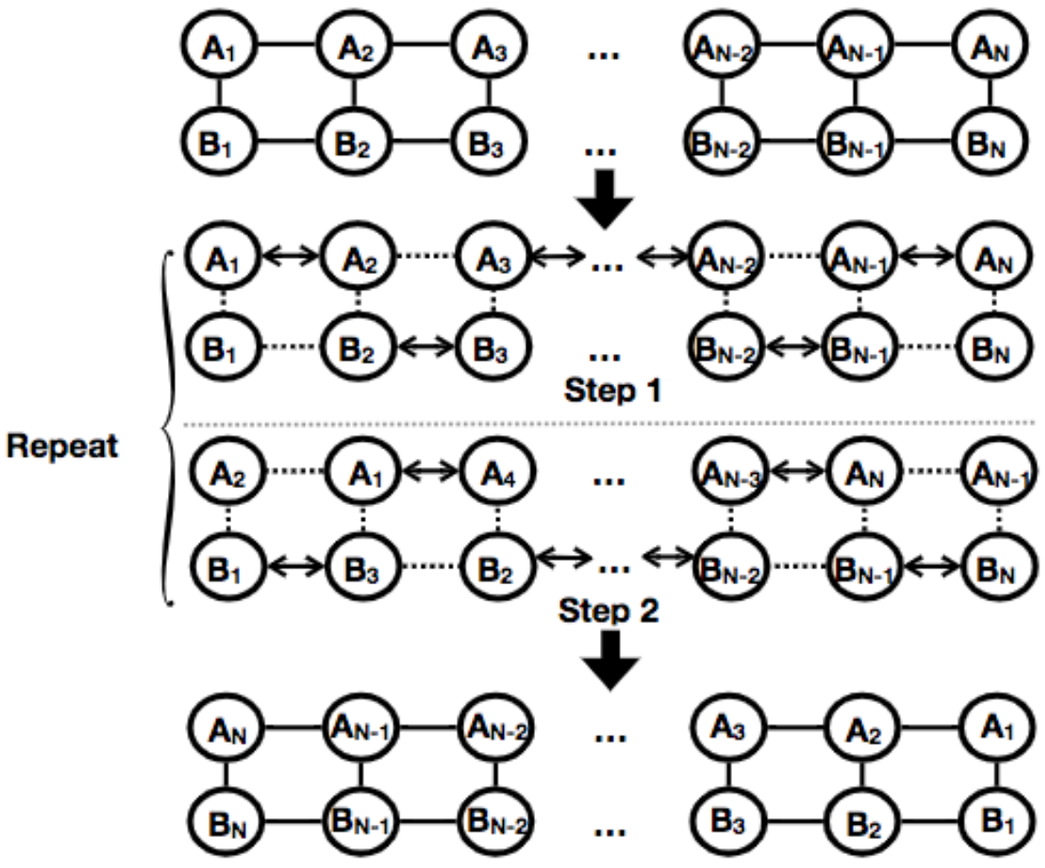
- Units and Unit-exchange
- Each units gets to be the nearest neighbor of every other unit once
- **Inter-unit bipartite all to all interaction**

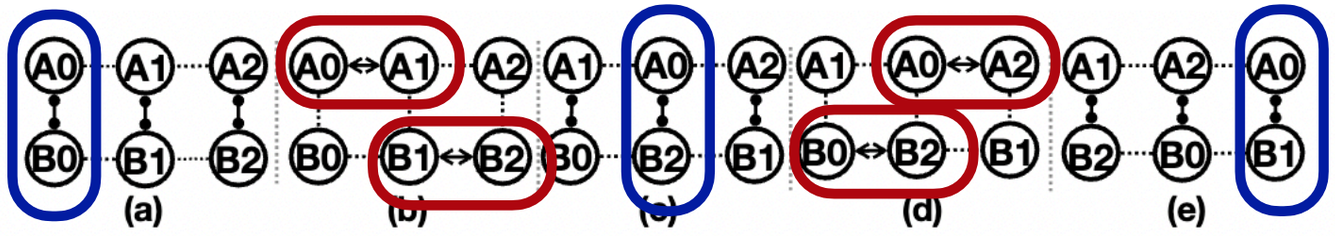


Result obtained by our depth-optimal A* solver.

Main Result II — The Regular 2D Grid

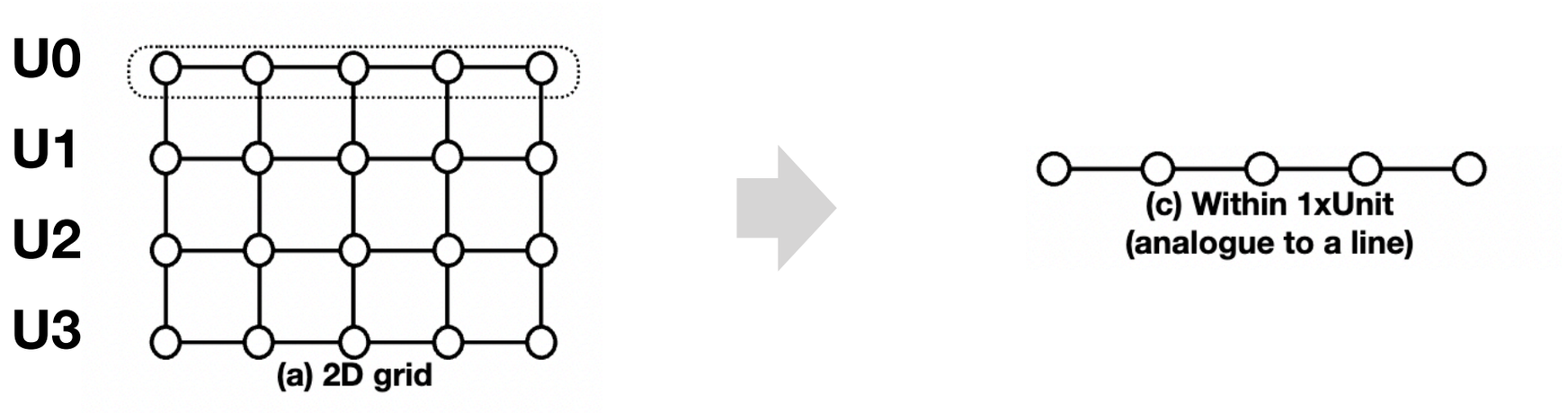
- Units and Unit-exchange
- Each units gets to be the nearest neighbor of every other unit once
- **Inter-unit bipartite all to all interaction**





Main Result II — The Regular 2D Grid

- Units and Unit-exchange
- Each units gets to be the nearest neighbor of every other unit once
- Inter-unit bipartite all to all interaction
- **Intra-unit all to all interaction**



Main Result II — The Regular 2D Grid

- **A few more other manual optimizations**
  - Reduced unit exchange layers
  - Interleaving intra-unit operations with inter-unit operations
  - See our paper for more details.
- **The complexity numbers**
  - 3N/2 + O(sqrt(N)) for all the operations including SWAP and computation
  - sqrt(N)/2 unit exchanges
  - N is the total number of nodes, assuming a square grid.

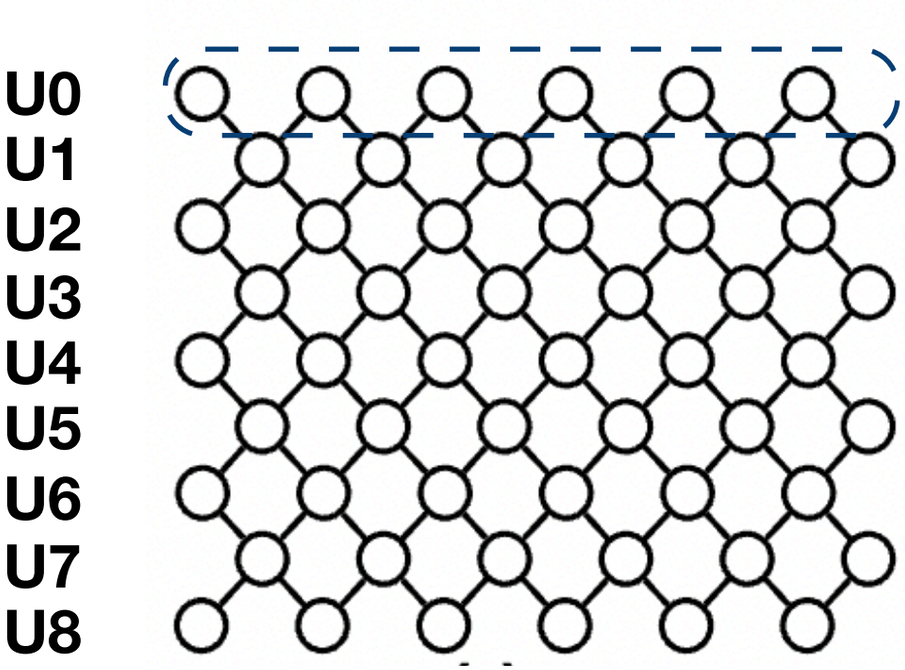
Key Takeaway Message

- Units and Unit-exchange
- Each units gets to be the nearest neighbor of every other unit once
- **Inter-unit bipartite all to all interaction**
- **Intra-unit all to all interaction**

Usually more challenging! We used an A* solver to solve it.

Main Result III — Google Sycamore

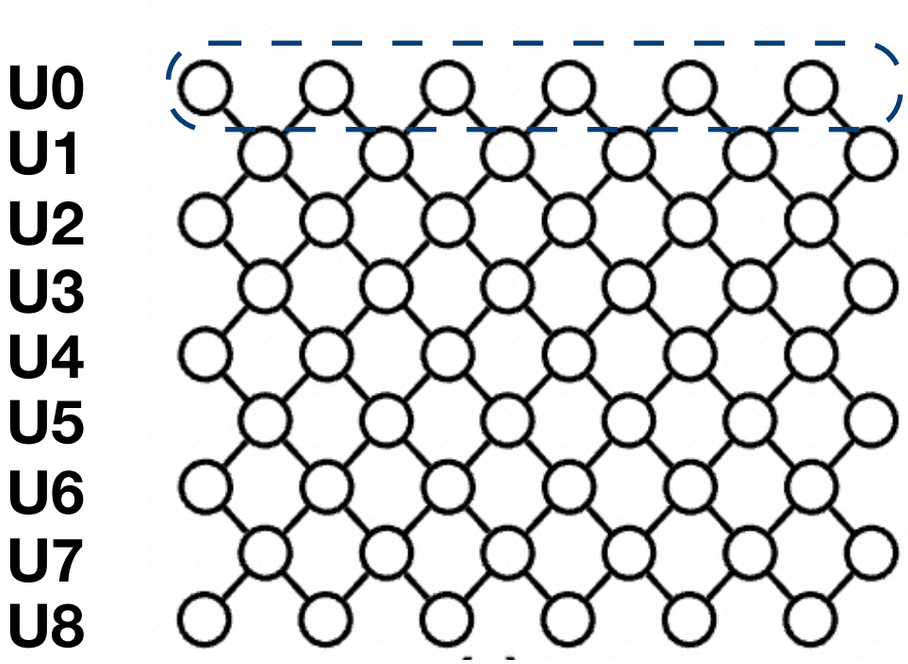
- Unit and Unit-exchanges

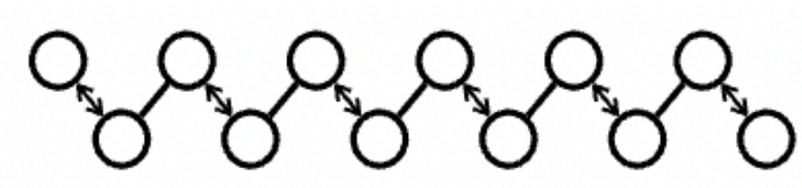


Unit Exchange

Main Result III — Google Sycamore

- Units and Unit-exchange
- Each unit gets to be the nearest neighbor of every other unit once

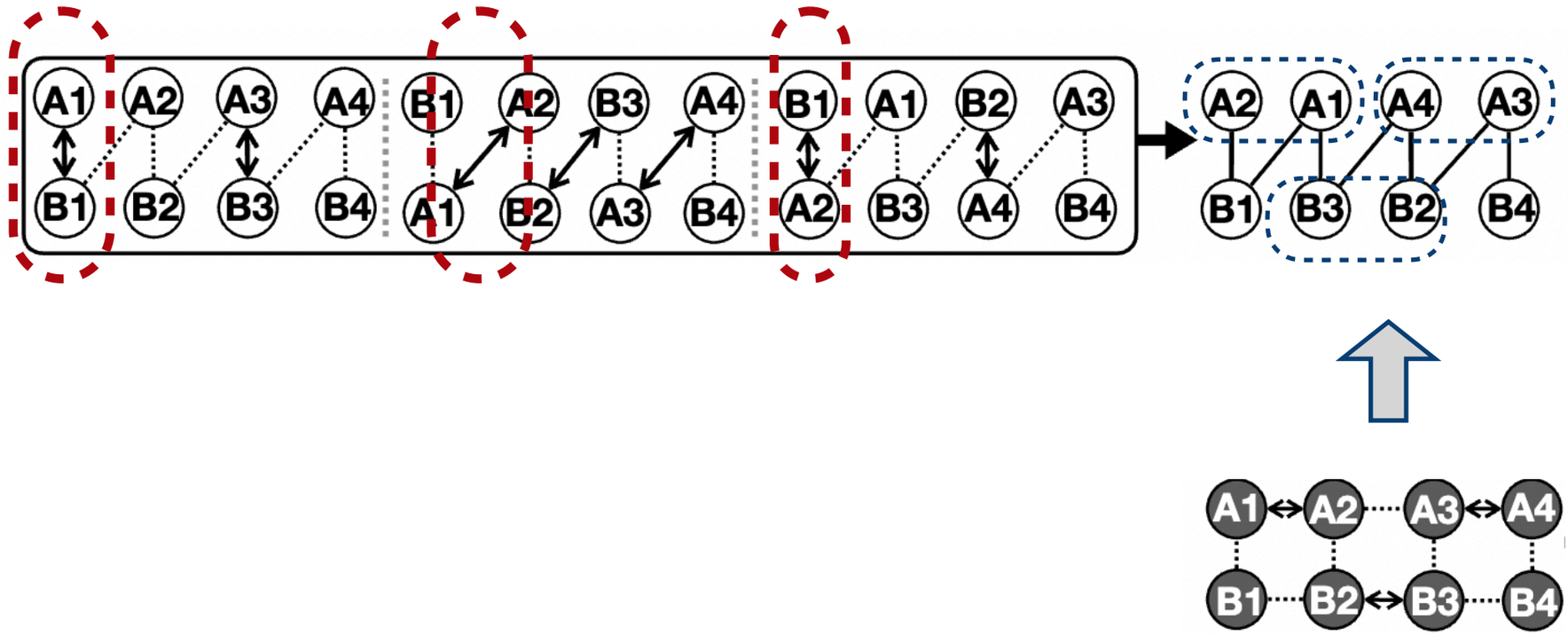




Unit Exchange

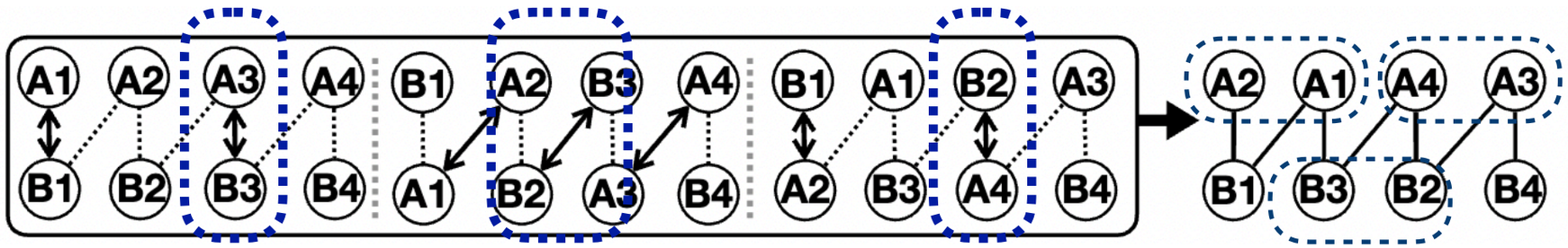
Main Result III — Google Sycamore

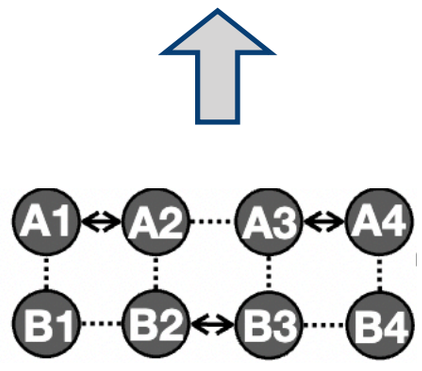
- Units and Unit-exchange
- Each unit gets to be the nearest neighbor of every other unit once
- **Inter-unit bipartite all to all interaction**



Main Result III — Google Sycamore

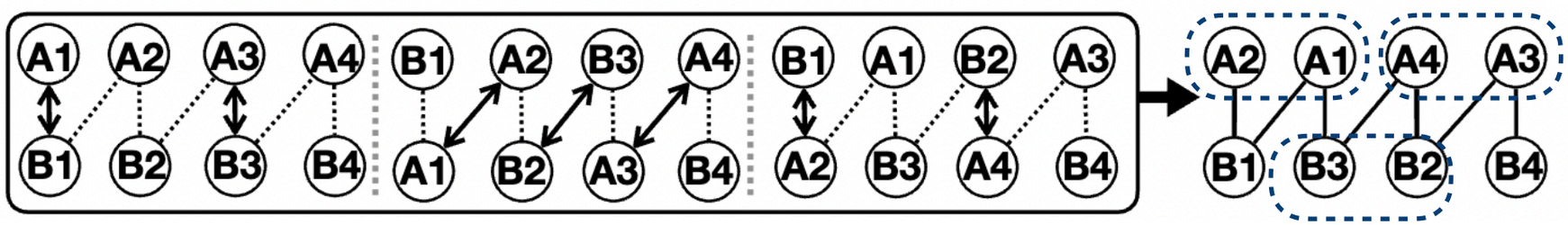
- Units and Unit-exchange
- Each unit gets to be the nearest neighbor of every other unit once
- **Inter-unit bipartite all to all interaction**

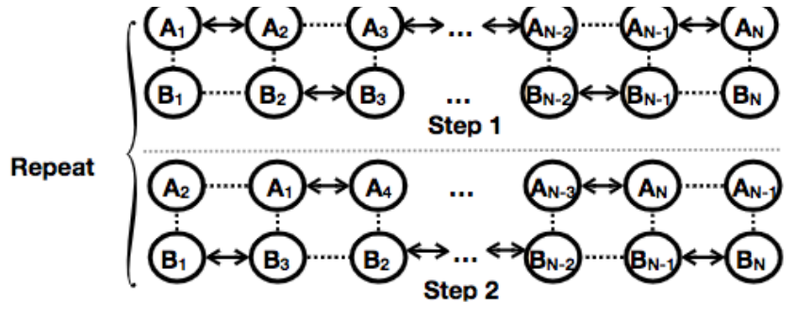


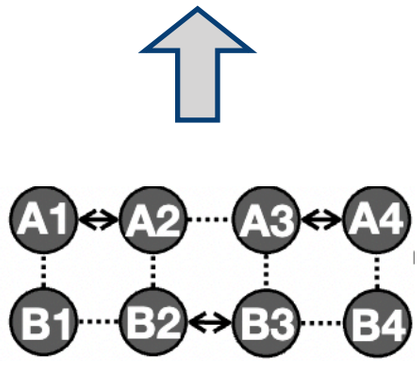


Main Result III — Google Sycamore

- Units and Unit-exchange
- Each unit gets to be the nearest neighbor of every other unit once
- **Inter-unit bipartite all to all interaction**





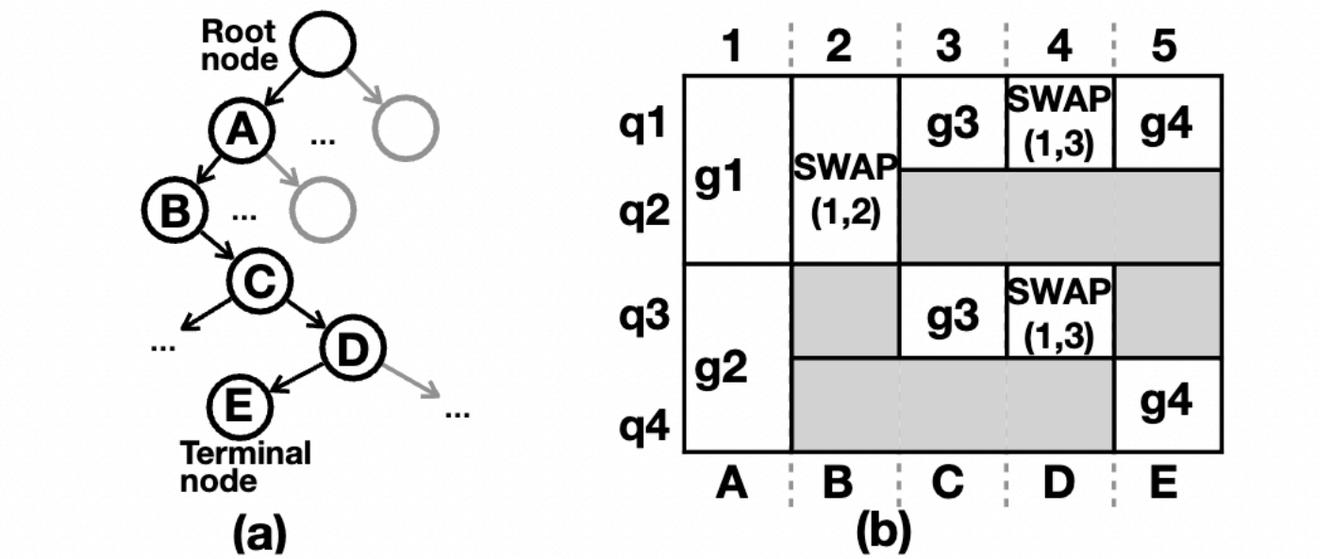


Main Result III — Google Sycamore

- **A few manual optimizations**
  - Mixing computation and SWAP operations into the same layers
  - Odd-number row optimization
  - See our paper for more details (arXiv: https://arxiv.org/abs/2112.06143)
- **Complexity**
  - Total operations: 5N/2 + O(sqrt(N))
  - N is the number of nodes, assuming a square grid.

The Optimal Solver Behind the Scenes

- **Search tree**
  - **Root node**: All gates not yet processed
  - **Edge**: One possible combination of gates execution. Cost = 1 cycle
  - **A state node**: status of circuit including remaining gates and current qubit mapping.
  - **Terminal node**: No gate remain unscheduled
- **The cost function (on the next page)**

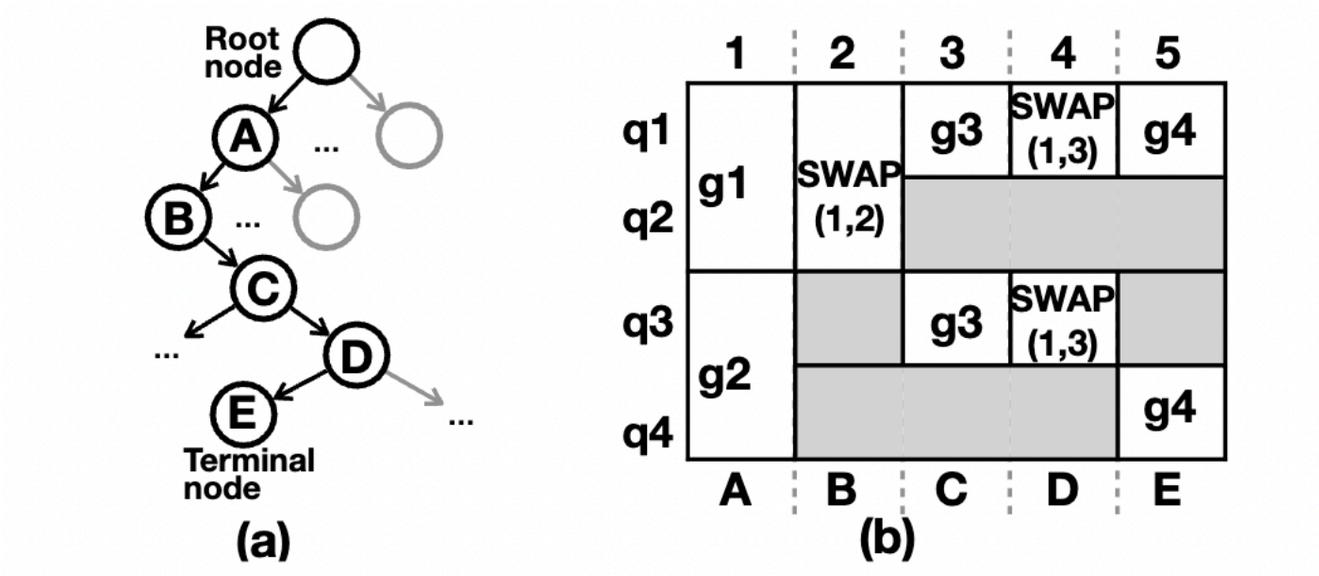


Our solver is open-sourced: https://github.com/ata-pattern/ata-pattern

The Cost Function in the A* Solver

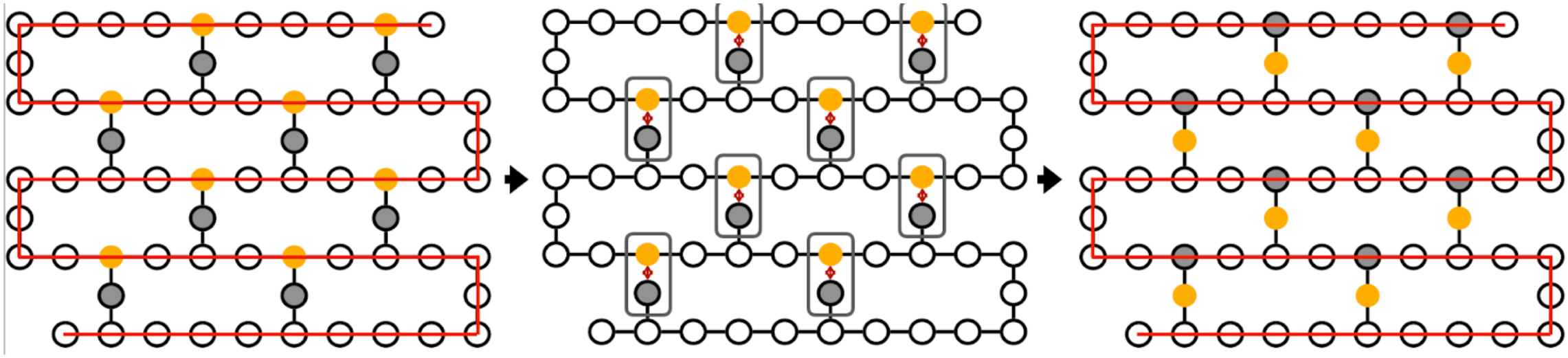
Cost function f (v) = c(v) + h(v)

- **c(v) is the cost of reaching the node v from the root**
- **ideal(g) = max(deg(g.q_i), deg(g.q_j)), estimating the minimum circuit depth with ideal architecture, assuming qubits are all-to-all connected.**
- **If the distance between q_i and q_j is d, at least d - 1 SWAP gates are needed in total.**
- **real(g) = $\min_{x=0..d-1}$ [max(deg(g.q_i) + x, deg(g.q_j) + d-1 -x)]**
- **Hence, we define h(v) = $\max_{g \in Erem(v)}$ real(g)**



IBM Heavy-hex (Adapting the linear solution)

- **Three passes**
  - All-to-all on-the-path (+ on-path with off-path whenever possible)
  - Swap between on-the-path and off-the-path nodes
  - All-to-all on-the path (+ on-path with off-path whenever possible)

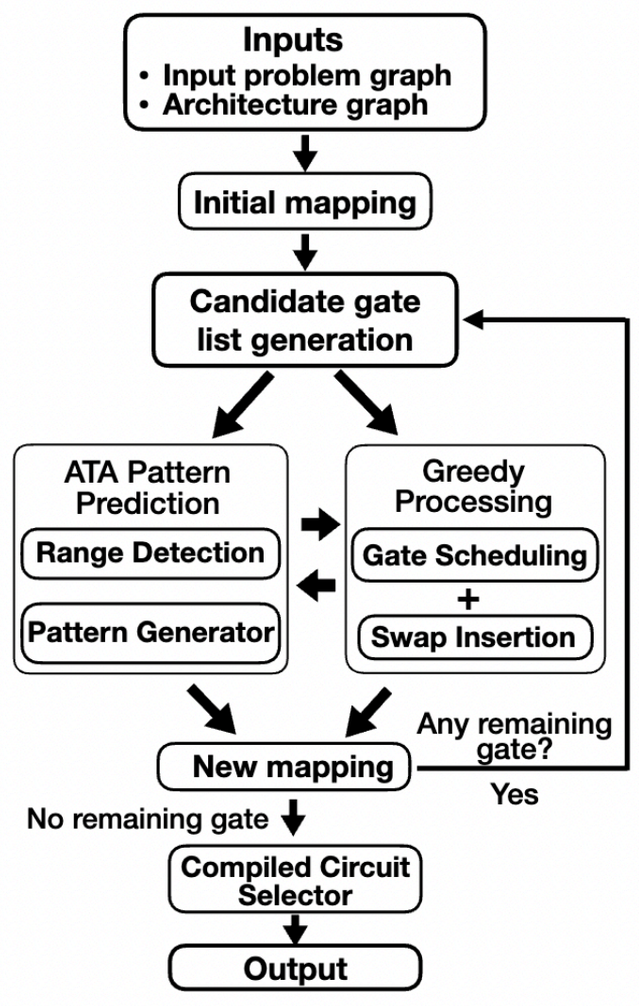


See more details in our paper:

"Exploiting the Regular Structure of Modern Quantum Architectures for Compiling and Optimizing Programs with Permutable Operators", accepted to ASPLOS'24.

Adapting ATA Solutions to Non-clique Circuits

- **Combining the greedy solution with the ATA pattern solution**
  - At each point, keep an additional compiled version that uses the circuit processed so far + using ATA for remaining gates.
  - Pick the best among all, after the complete greedy scheduling is done (we also considered fidelity when comparing circuits).
  - *Taking the best of both worlds*.

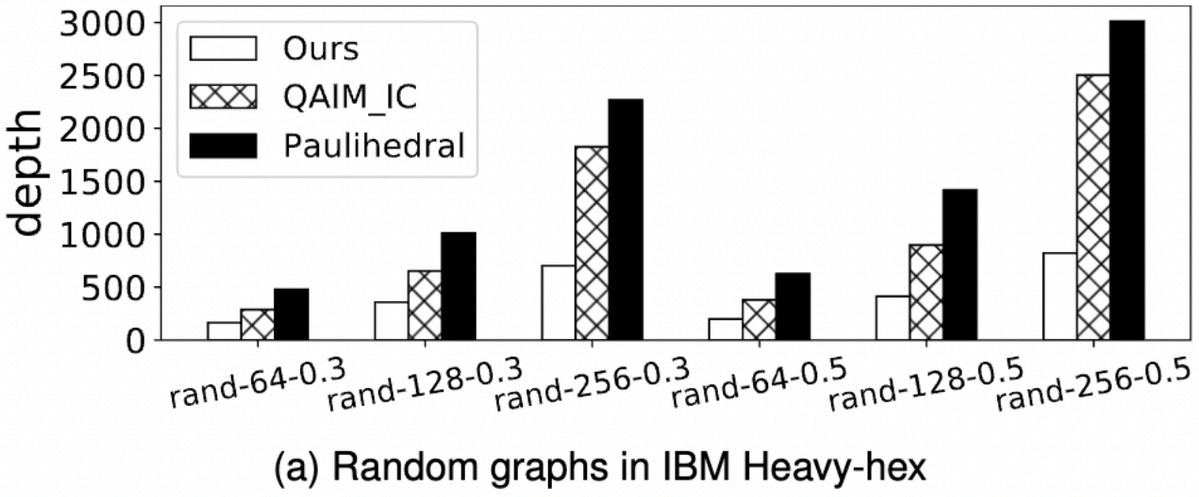


Experiments

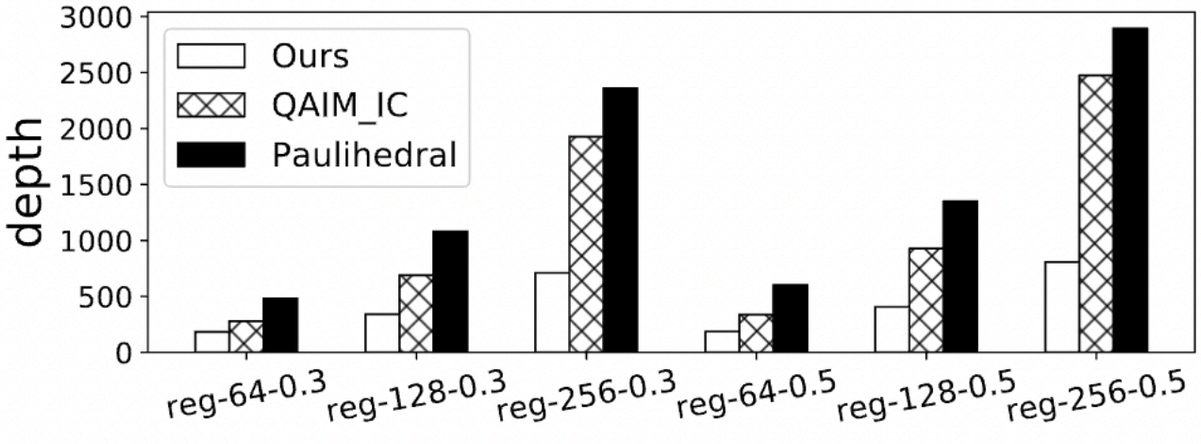
- **Architectures: IBM heavy-hex and Google Sycamore hardware**
- **Metrics: Circuit depth, two-qubit gate count, and total variant distance (TVD)**
- **Baselines: Paulihedral, QAIM, and 2QAN**
- **Benchmarks:**
  - Random graph and regular graph generated by python library NetworkX. Density of graph as 0.3 and 0.5.
  - The number of vertices of two types of benchmarks vary from 64 to 1024.

IBM Heavy-hex

- **Depth and Gate-count**

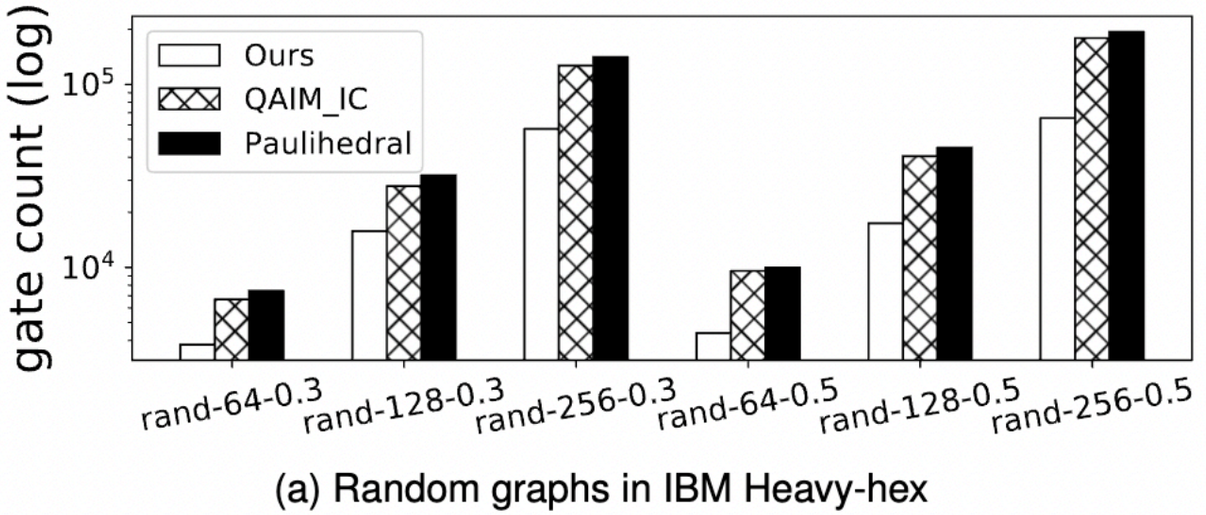


(a) Random graphs in IBM Heavy-hex

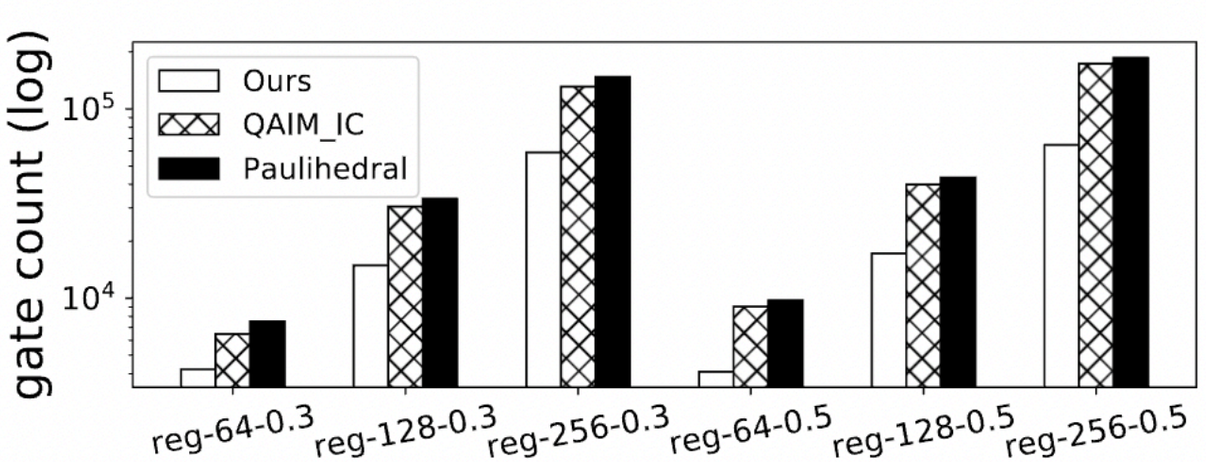


(b) Regular graphs in IBM Heavy-hex

Depth



(a) Random graphs in IBM Heavy-hex

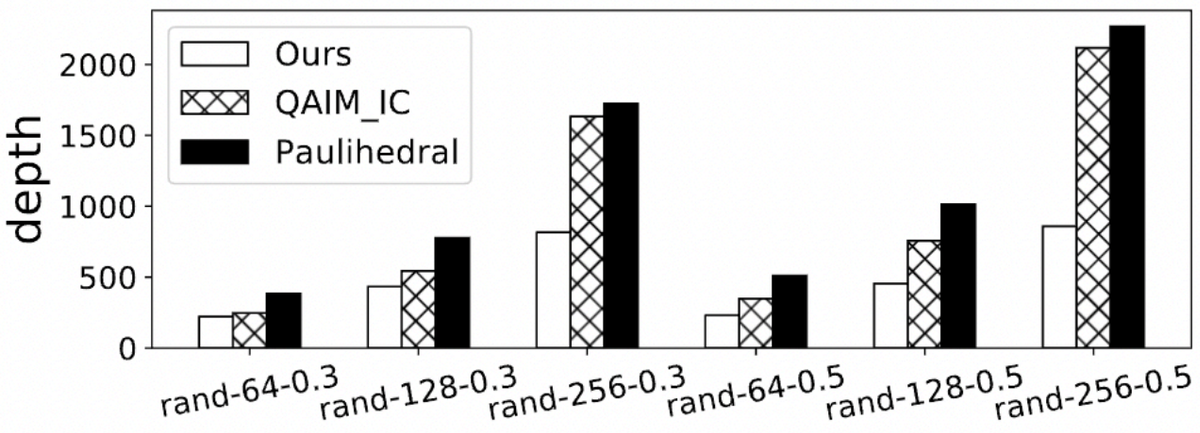


(b) Regular graphs in IBM Heavy-hex

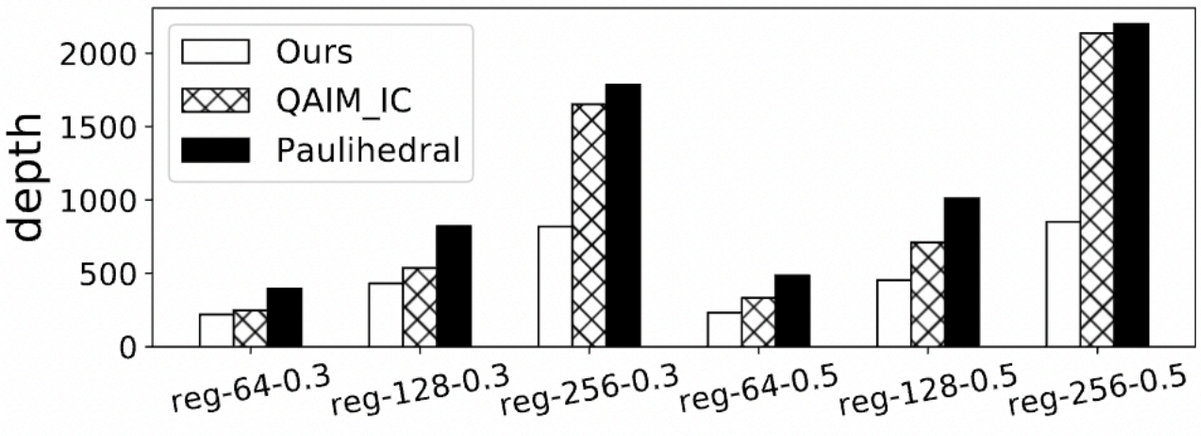
Gate-count

Google Sycamore

- **Depth and Gate-count**

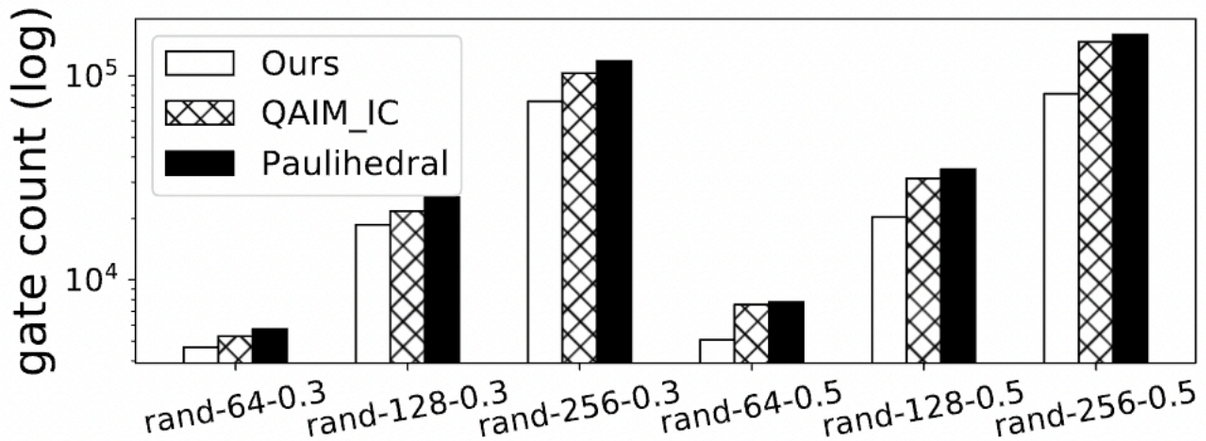


(a) Random graphs in Google Sycamore

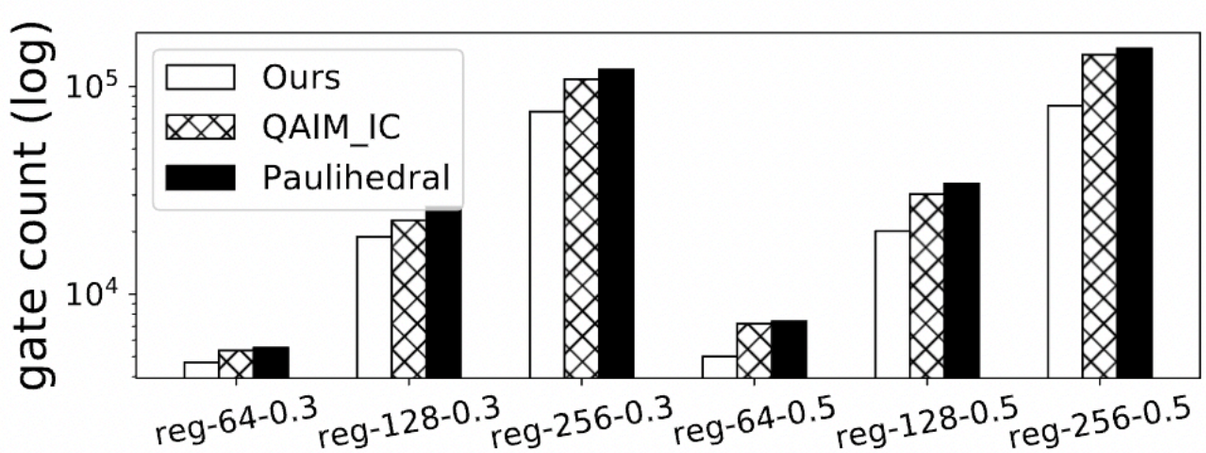


(b) Regular graphs in Google Sycamore

Depth



(a) Random graphs in Google Sycamore



(b) Regular graphs in Google Sycamore

Gate-count

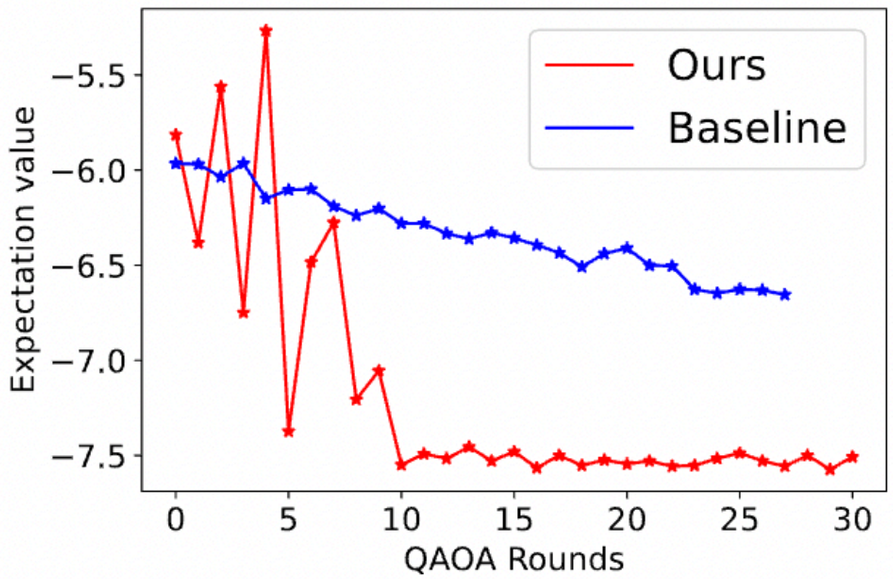
Comparison with 2QAN

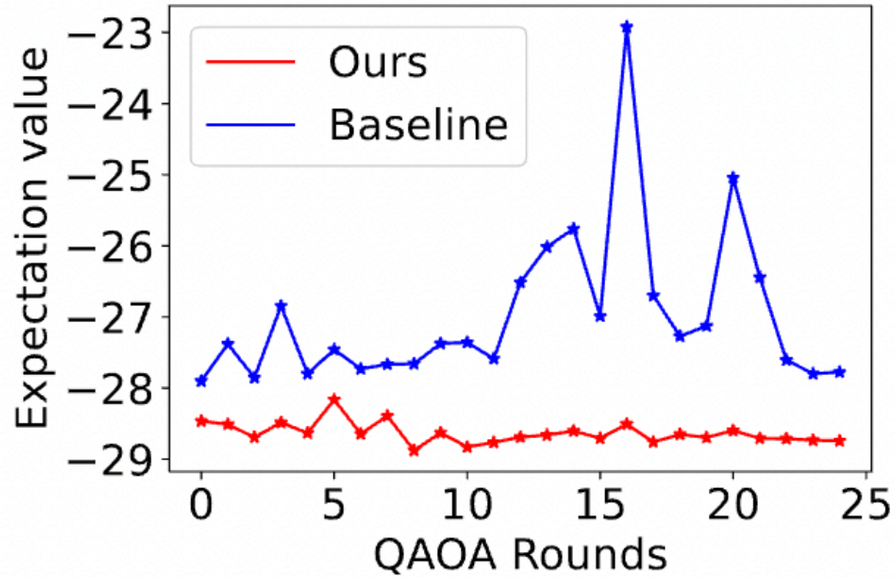
- **Depth and CNOT count**

		Depth			Gate Count (CNOT)		
Arch.	Graphs	Ours	2QAN	QAIM	Ours	2QAN	QAIM
Heavy-hex	64-0.3	164	241	288	3789	4902	6695
	128-0.3	357	781	652	15785	19563	27799
	256-0.3	703	-	1825	57157	-	126653
	64-0.5	198	339	380	4385	6816	9580
	128-0.5	413	939	898	17458	29088	40487
	256-0.5	822	-	2503	65541	-	179085
Sycamore	64-0.3	220	235	247	4662	4785	5294
	128-0.3	433	-	543	18593	-	21691
	256-0.3	817	-	1634	74959	-	103010
	64-0.5	233	265	348	5041	5352	7564
	128-0.5	454	-	756	20266	-	31367
	256-0.5	858	-	2118	81567	-	146646

Real Machine Experiments

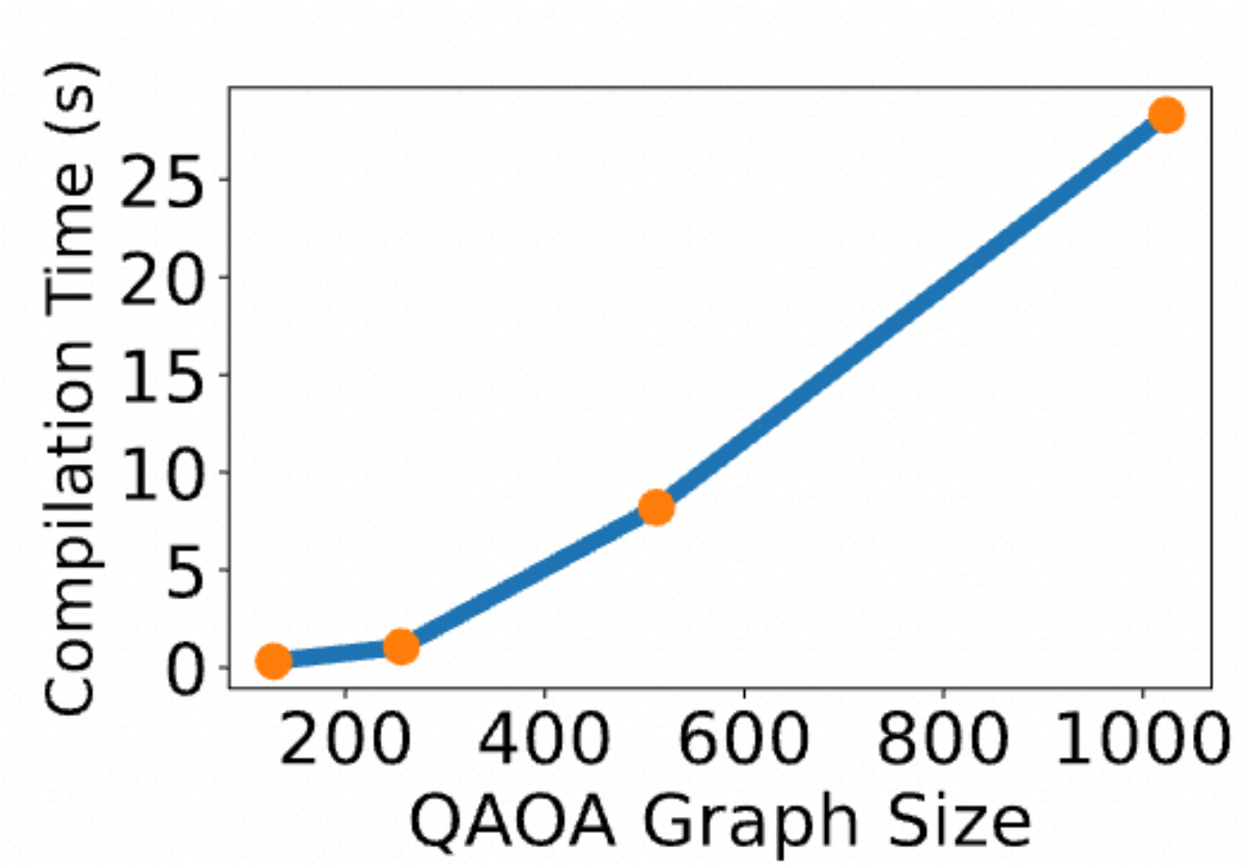
- **IBM Mumbai, graph density = 30%**





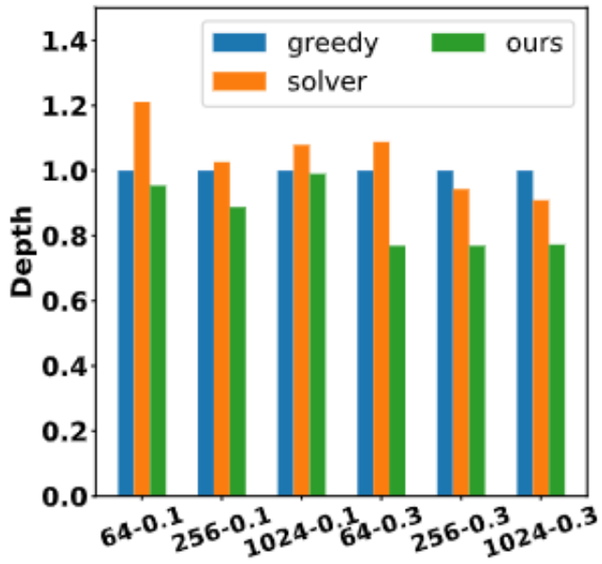
Compilation Overhead

- **Near linear when graph size is larger.**

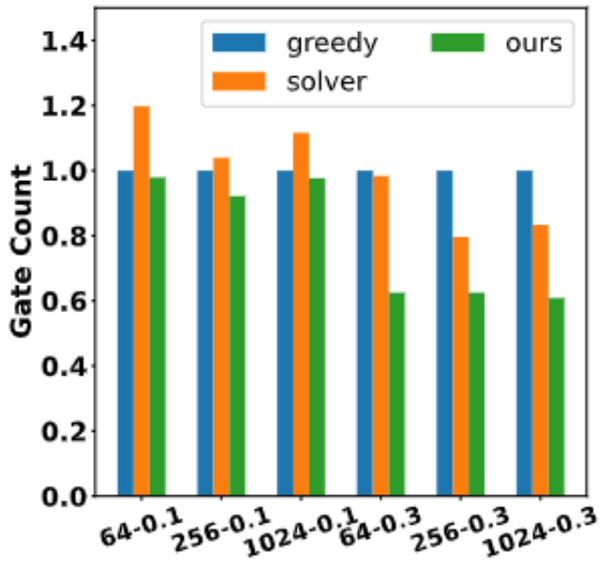


Taking the Best of Both Worlds

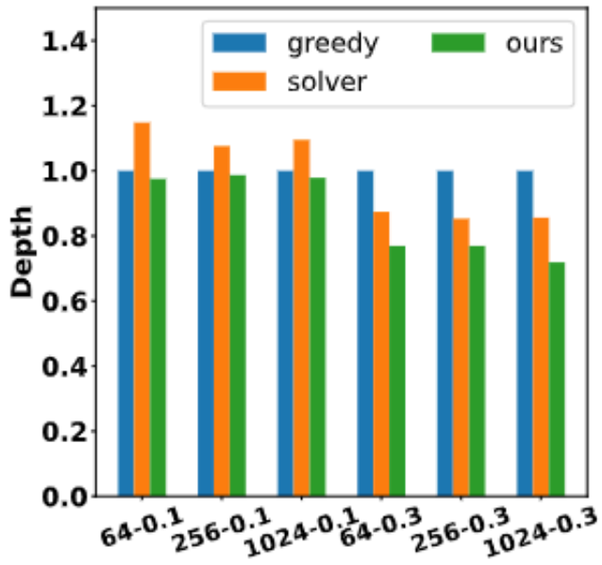
- **Pure greedy v.s. pure ATA solver guided v.s. our mixed solution**



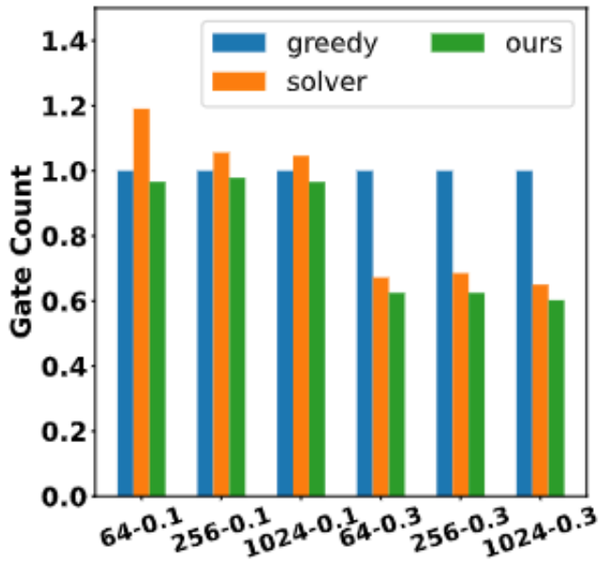
(a) Depth in Heavy-hex



(b) Gate Count in Heavyhex



(c) Depth in Sycamore



(d) Gate Count in Sycamore

Conclusion

1. We solved a **n-clique QAOA circuit** efficiently, hence we can solve **any sub-circuit** of the n-clique circuit efficiently.
2. We revealed **a linear-depth solution for QAOA mapper,** similar to that linear time solution in the famous **permutation network** result by Abraham Waksman.
3. We designed **an optimal solver** that helped us find the solutions for reasonably sized circuits, in a divide and conquer manner.

Questions?