Quantum Scheduler

Real-time and parallel task scheduling for Quantum Computing

Marcin Praski
Advisor: Emmanouil Giortamis
Chair of Computer Systems
https://dse.in.tum.de/

15.06.2024 – 15.12.2024

Background

Quantum Processing Units (QPUs):

- execute quantum jobs (= collections of circuits)
- are scarce
- exhibit heterogeneity (spatial, temporal) in their parameters

Users:

- want predictable ▲performance & ▲fidelity of their quantum jobs

Cloud providers:

- want to optimize QPU ▼allocation and maximize ▲throughput

Quantum Job Scheduler

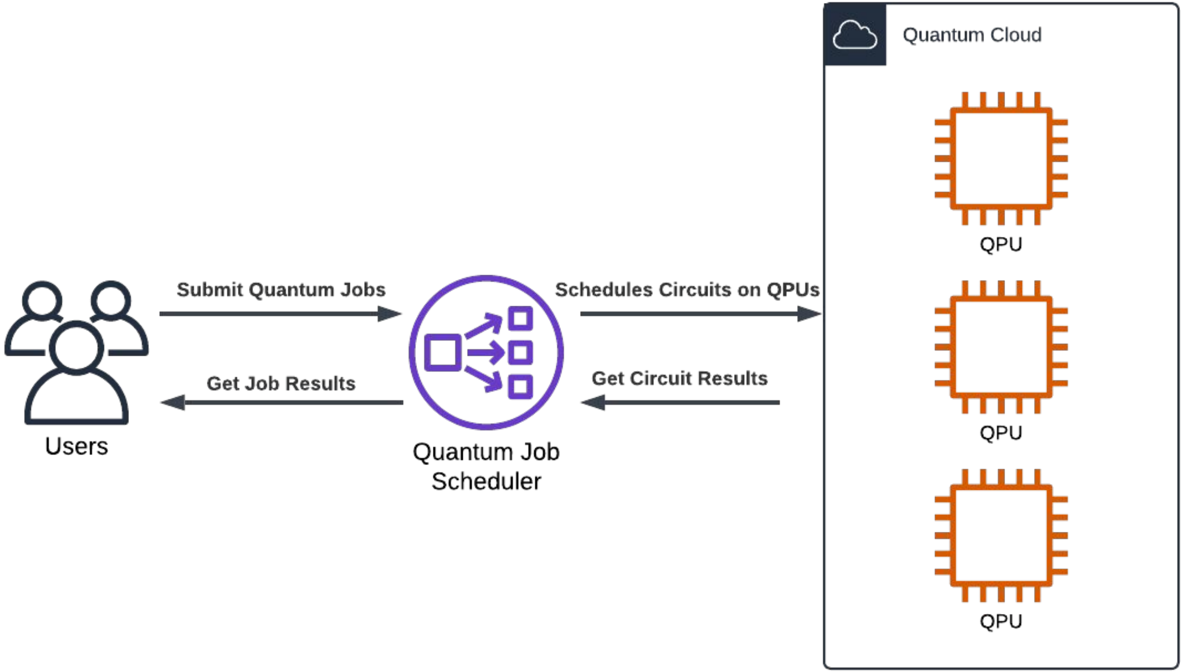
Scheduler needs to balance multiple conflicting objectives

System design goals:

- Multi-Objective Optimization
- Batch processing of incoming jobs
- Adaptable to problem size
- Scalable

Quantum Job Scheduler: Multi-Objective Optimizer

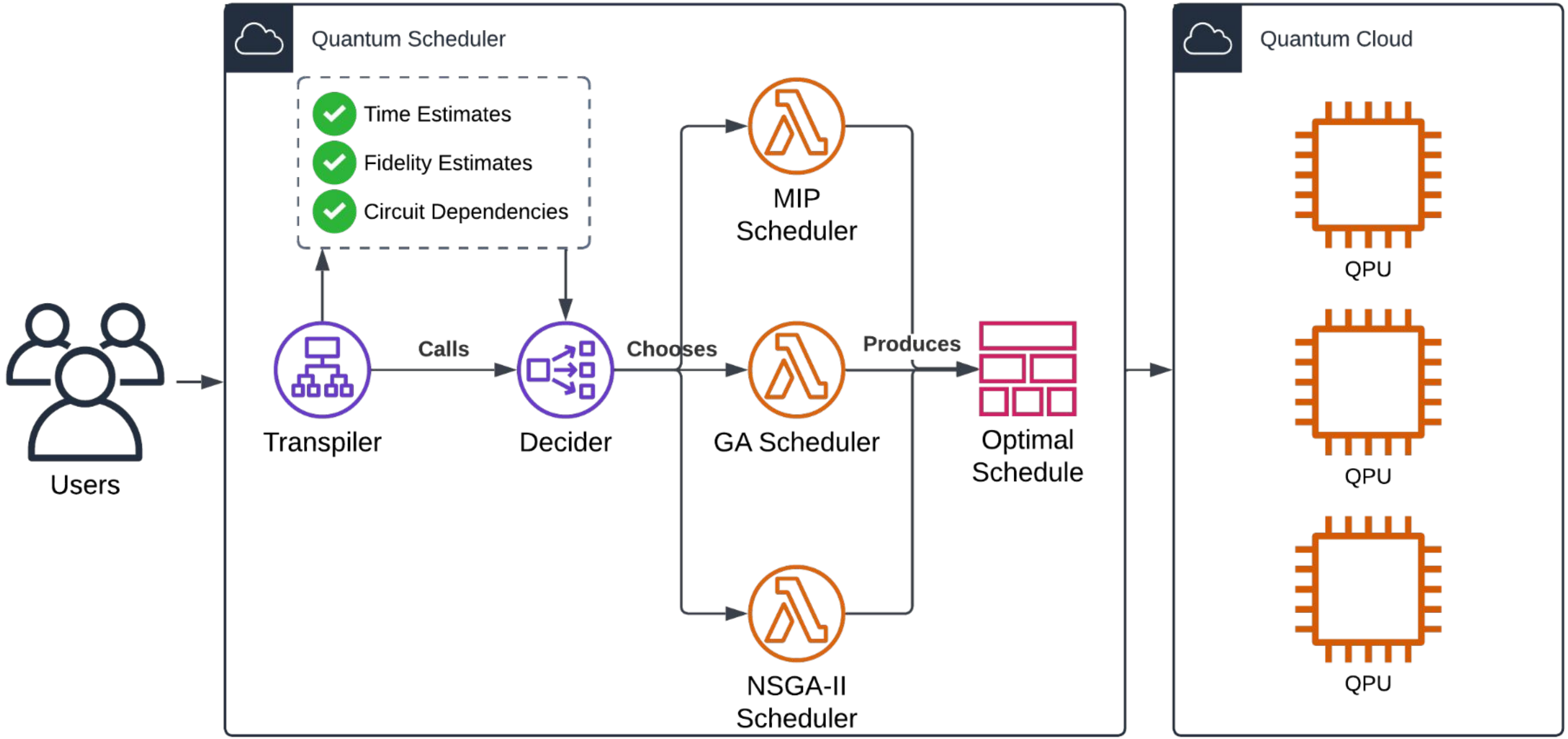
Quantum Scheduler for multi-objective schedule generation



Outline

- ~~Motivation~~
- Design overview
  - Mixed Integer Programming
  - Genetic Algorithm (GA + NSGA-II)
- Implementation
- Evaluation

Design overview



Mixed Integer Programming

Transform optimization problem into Linear Programming (LP) constraints and objectives. Iteratively **relax, branch-and-bound** to solve.

$$\min \begin{pmatrix} M_{\max} \\ E_{\text{avg}} \end{pmatrix} \quad \text{where}$$

$$M_{\max} = \max_{j=1,\ldots,Q} \left(\sum_{i=1}^{N} S_{ij} + t_{ij} \right)$$

$$E_{\text{avg}} = \frac{1}{N} \sum_{i=1}^{N} (1 - x_{ij} f_{ij})$$

s.t.

$$\sum_{j=1}^{Q} x_{ij} = 1 \quad \forall i = 1, \ldots, N \quad \text{(each task assigned to exactly one CPU)}$$

$$x_{ij} \in \{0,1\} \quad \forall i = 1, \ldots, N; \quad \forall j = 1, \ldots, Q$$

$$S_{ij} \geq S_{kj} + t_{kj} \quad \text{if job } k \text{ precedes job } i \text{ on QPU } j$$

$$S_{ij} \geq 0 \quad \forall i = 1, \ldots, N; \quad \forall j = 1, \ldots, Q \quad \text{(each job starts at some time)}$$

$$\min \begin{pmatrix} M \\ E \end{pmatrix} \qquad \text{(A1)}$$

s.t.

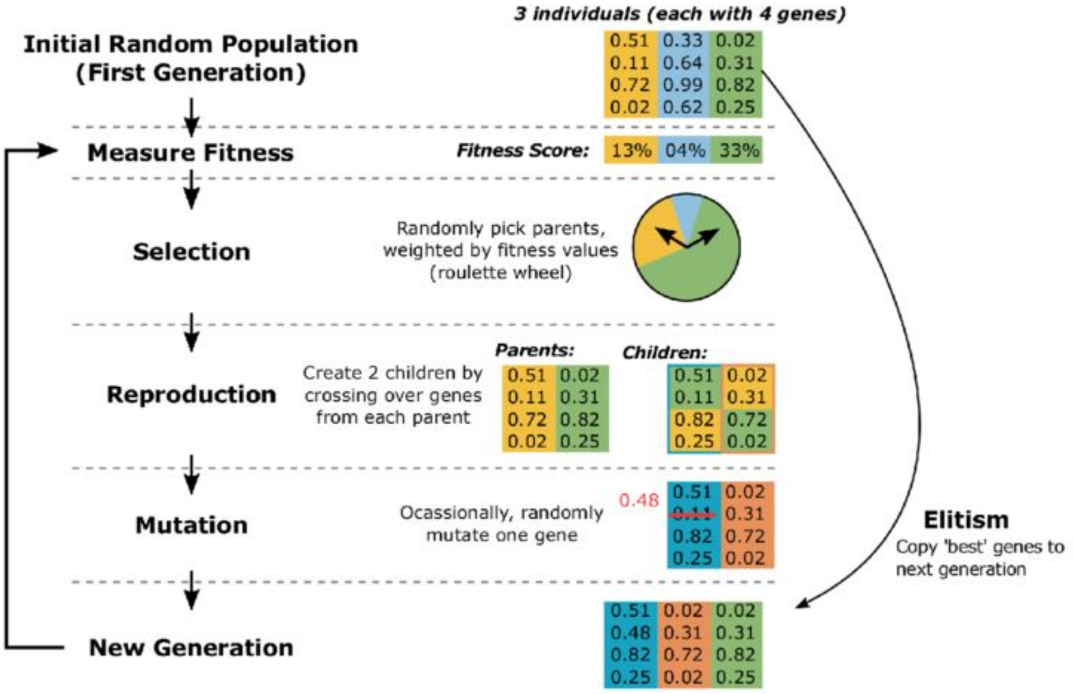
$$t_i + d_i \leq M \qquad \forall i \in C \qquad \text{(A2a)}$$
$$E = \sum_{i \in C} \sum_{b \in B} (h_{ib} - 1) mean\_error_{ib} \qquad \text{(A2b)}$$
$$d_i \geq \sum_{b \in B} (h_{ib} - 1) runtime_{ib} \qquad \forall i \in C \qquad \text{(A3)}$$
$$|B| h1_{ib} \geq r_i - b + \Delta \qquad \forall i \in C, \forall b \in B \qquad \text{(A4)}$$
$$|B|(1 - h1_{ib}) \geq b - r_i - \Delta \qquad \forall i \in C, \forall b \in B \qquad \text{(A5)}$$
$$|B| h2_{ib} \geq b - r_i + \Delta \qquad \forall i \in C, \forall b \in B \qquad \text{(A6)}$$
$$|B|(1 - h2_{ib}) \geq r_i - b - \Delta \qquad \forall i \in C, \forall b \in B \qquad \text{(A7)}$$
$$h_{ib} \geq h1_{ib} + h2_{ib} \qquad \forall i \in C, \forall b \in B \qquad \text{(A8)}$$
$$\sigma_{ij} + \sigma_{ji} \leq 1 \qquad \forall i \neq j \in C \qquad \text{(A9)}$$
$$\epsilon_{ij} + \epsilon_{ji} \leq 1 \qquad \forall i \neq j \in C \qquad \text{(A10)}$$
$$\sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} \geq 1 \qquad \forall i \neq j \in C \qquad \text{(A11)}$$
$$r_j + r_i - 1 - (\epsilon_{ij} - 1)|B| \geq 0 \qquad \forall i \neq j \in C \qquad \text{(A12)}$$
$$r_j + r_i - \epsilon_{ij}|B| \leq 0 \qquad \forall i \neq j \in C \qquad \text{(A13)}$$
$$t_j + d_i - (\sigma_{ij} - 1) M_{\max} \leq t_j \qquad \forall i \neq j \in C \qquad \text{(A14)}$$
$$t_i + d_i \leq t_j \qquad \forall (i,j) \in \gamma \qquad \text{(A15)}$$
$$\sigma_{ij} \in \{0,1\} \qquad \forall i \neq j \in C \qquad \text{(A16)}$$
$$\epsilon_{ij} \in \{0,1\} \qquad \forall i \neq j \in C \qquad \text{(A17)}$$
$$h_{ij}, h1_{ij}, h2_{ij} \in \{0,1\} \qquad \forall i \neq j \in C \qquad \text{(A18)}$$
$$r_{ij} \in \{1, ..., |B|\} \qquad \forall i \in C \qquad \text{(A19)}$$
$$t_i \geq 0 \qquad \forall i \in C \qquad \text{(A20)}$$
$$d_i \geq 1 \qquad \forall i \in C \qquad \text{(A21)}$$

Mixed Integer Programming

The

Genetic Algorithm

Define the chromosome and genetic operators to simulate natural evolution. Repeat **N generations** until sufficient fitness achieved.



Genetic Algorithm

Chromosome represents a single solution. They pair, mate, mutate to propagate their "good genes" to the next generation.

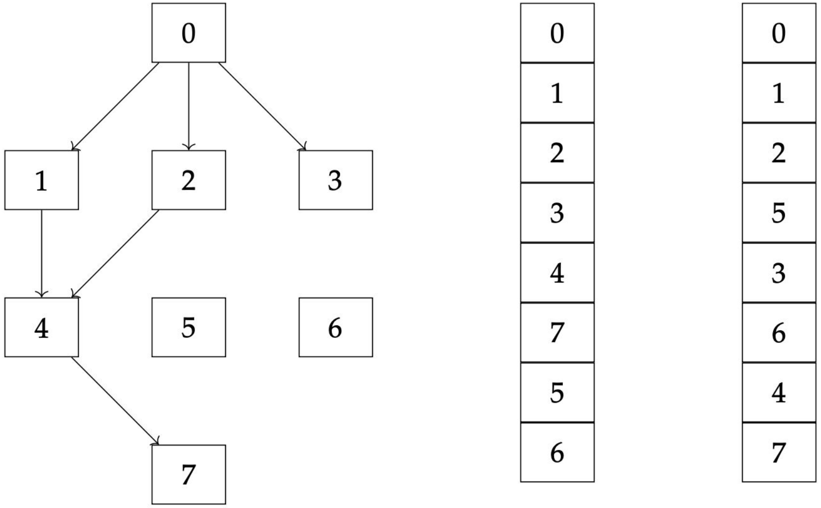
We define the chromosome as a **bijection** between sets of jobs and backends.

i_0	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	
(J_1, B_1)	(J_2, B_2)	(J_3, B_3)	(J_4, B_4)	(J_5, B_5)	(J_6, B_6)	(J_7, B_7)	(J_8, B_8)	(J_9, B_9)	$\cdots$

Figure 4.1: Chromosome representation

Genetic Algorithm

We sort the jobs in a **topological order** defined by the “starts-before” dependency relation. This ensures fast array operations and guarantees that all predecessor jobs have already been visited during genetic operators.



Dependency partial order

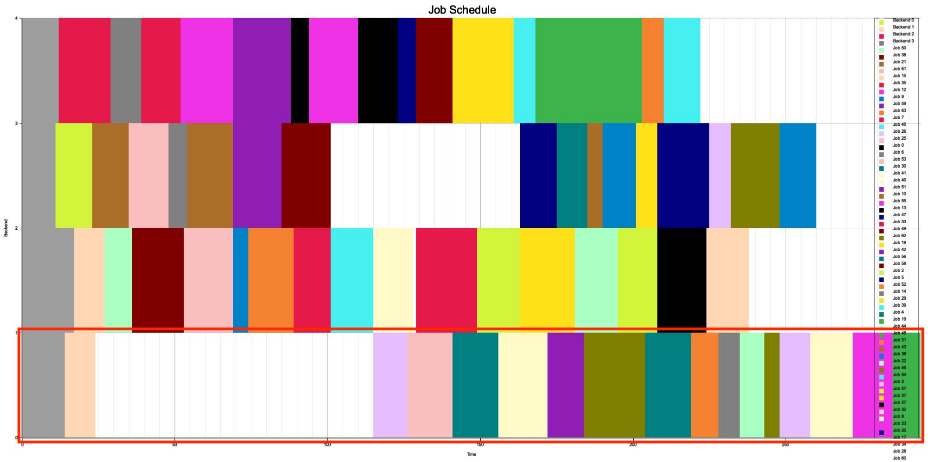
Linearization 1

Linearization 2

Genetic Algorithm

The **fitness function** of our chromosome is a weighted sum:

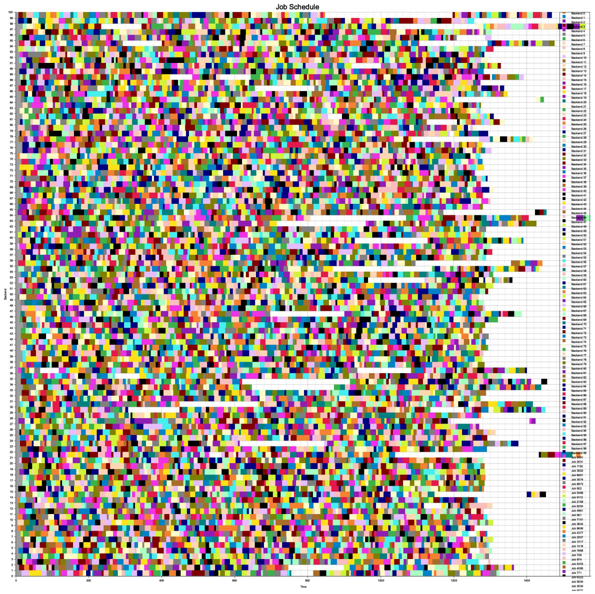
$$\lambda \frac{1}{max\_makespan} + (1-\lambda) \frac{total\_fidelity}{\text{LEN}(chromosome)}$$



Where makespan is the length of the longest schedule on any QPU, and total fidelity the sum of fidelities of all scheduled jobs

Genetic Algorithm

Additionally, we condense the schedules during mutation to boost the probability of finding a schedule with smaller makespan. This may negatively affect the fidelity component of the fitness function.



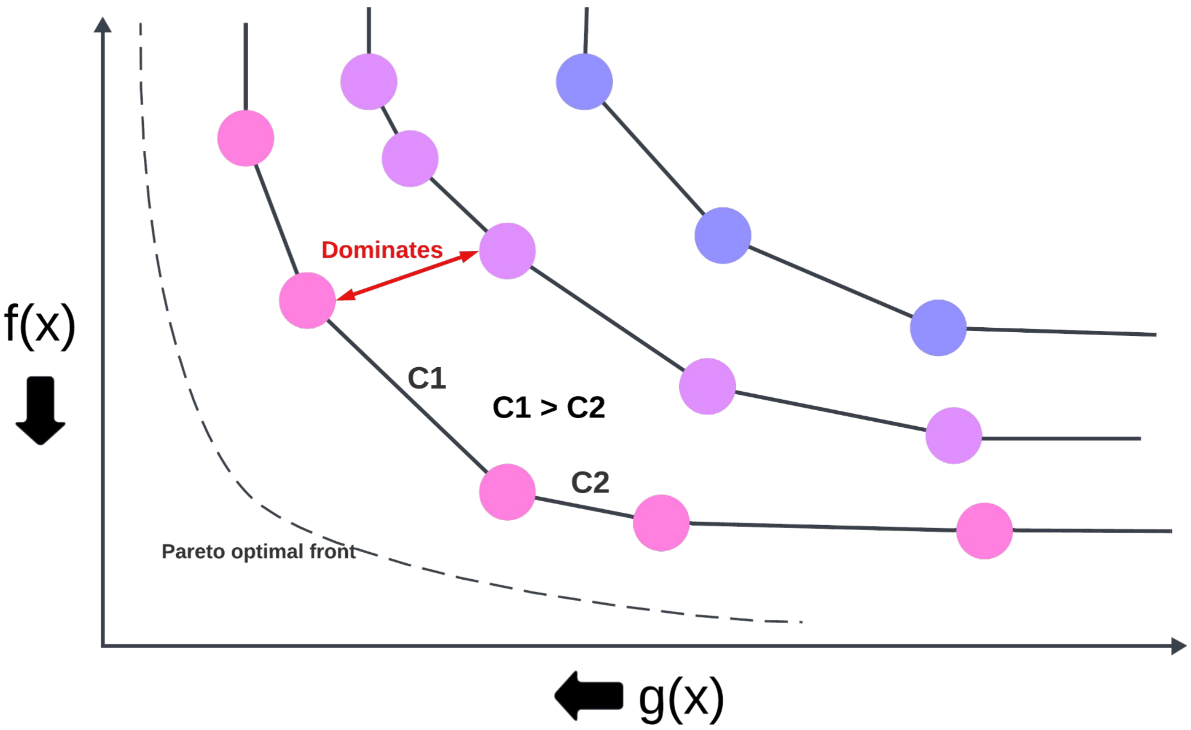
Non-dominated Sorting Genetic Algorithm (NSGA-II)

Key Novelties:

- Pareto Dominance
- Crowding Distance

Benefits:

- No more **scalarization and weighing** of multi-objectives
- Higher solution quality



Non-dominated Sorting Genetic Algorithm (NSGA-II)

Dominance Order between multi-objective solutions allows for categorization into Pareto fronts.

```
fn dominance_ord(&self, a: &Self::T, b: &Self::T) -> Ordering {
    let mut less_cnt = 0;
    let mut greater_cnt = 0;

    for objective in self.objectives.iter() {
        match objective.total_order(a, b) {
            Ordering::Less => { less_cnt += 1; }
            Ordering::Greater => { greater_cnt += 1; }
            Ordering::Equal => {}
        }
    }

    if less_cnt > 0 && greater_cnt == 0 {
        Ordering::Less
    } else if greater_cnt > 0 && less_cnt == 0 {
        Ordering::Greater
    } else {
        Ordering::Equal
    }
}
```

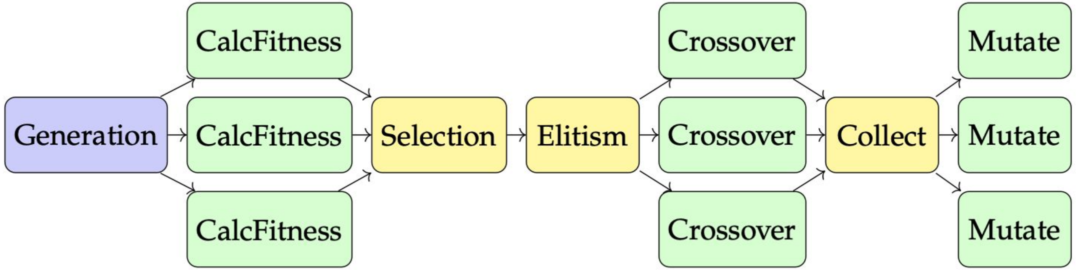
Outline

- ~~Motivation~~
- ~~Design overview~~
- Implementation
- Evaluation

Implementation

MIP Optimizer is built with Python/PuLP[1], GA + NSGA-II with Rust/Rayon[2]

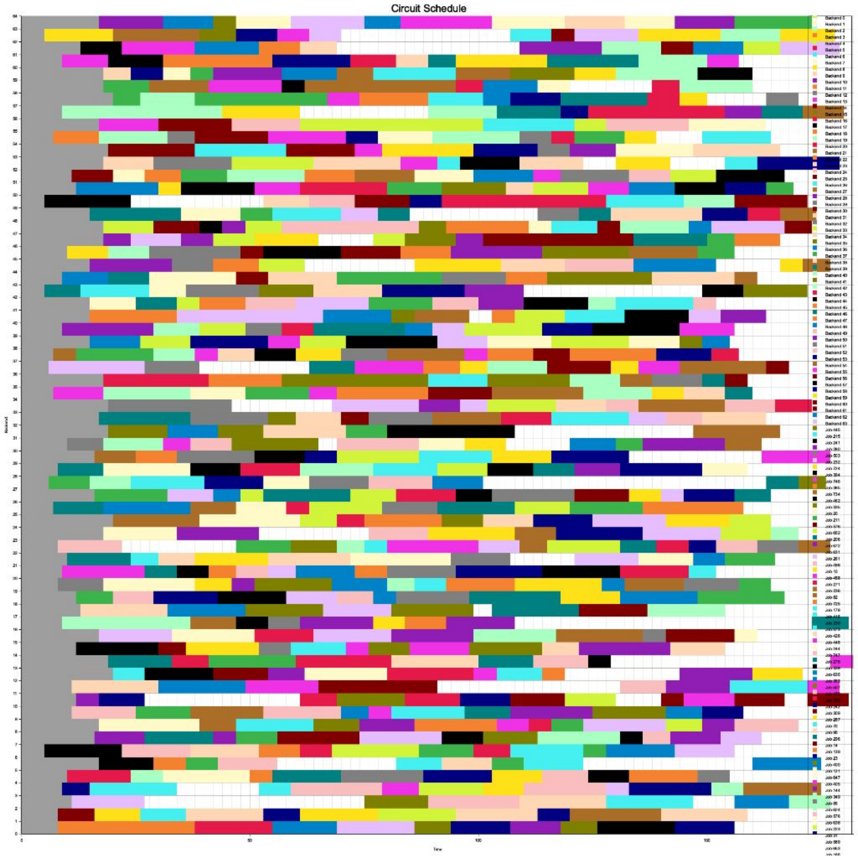
- High level interface for MIP solvers: PuLP
- Rust provides low-level performance with high-level abstractions
- Great for data-level parallelism in GA/NSGA-II → Rayon
- Embarrassingly parallel problem → fitness function, mutation etc.

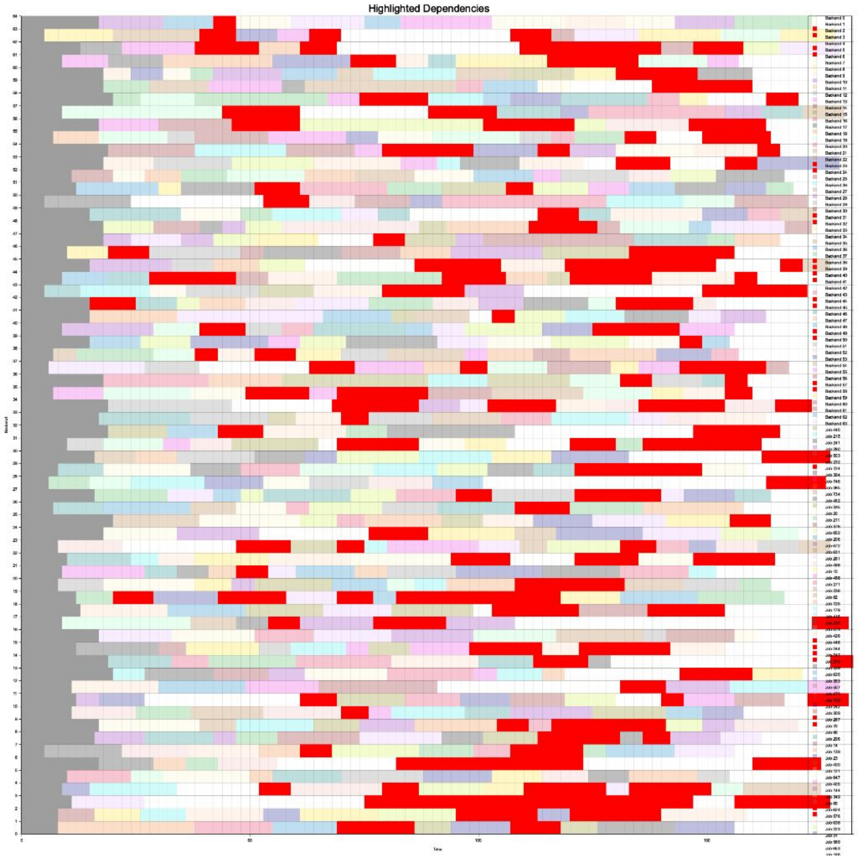


[1]Optimizing linear and mixed integer models with PuLP: https://coin-or.github.io/pulp

[2]Date-parallelism library for Rust: https://github.com/rayon-rs/rayon

Example Output





Outline

- ~~Motivation~~
- ~~Design overview~~
- ~~Implementation~~
- Evaluation

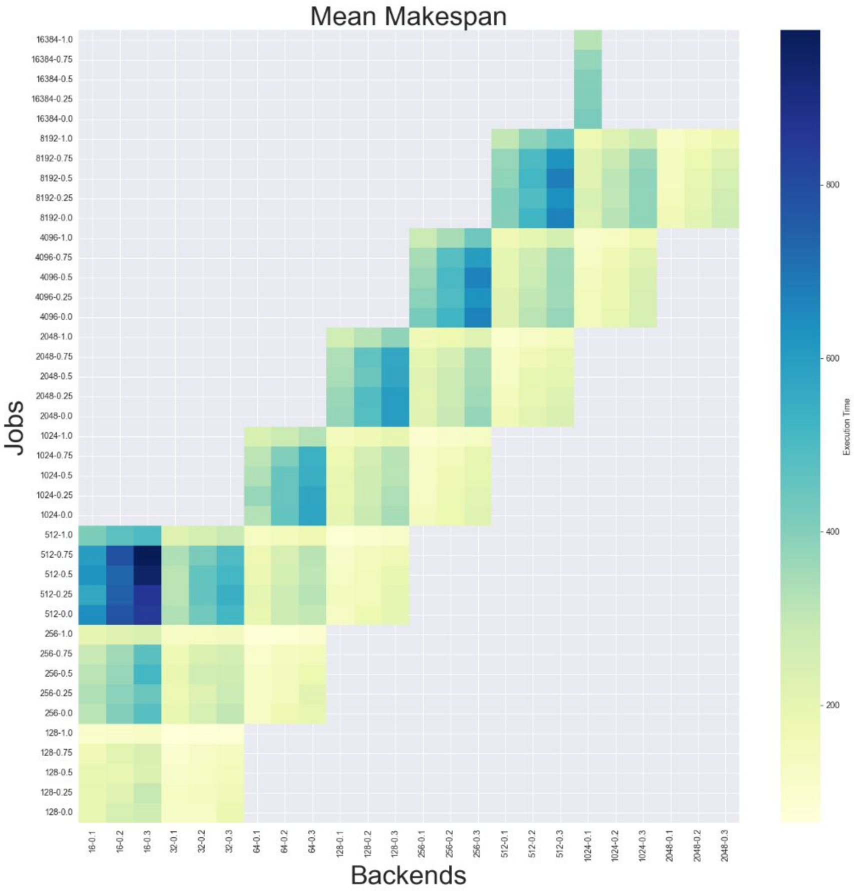
Evaluation

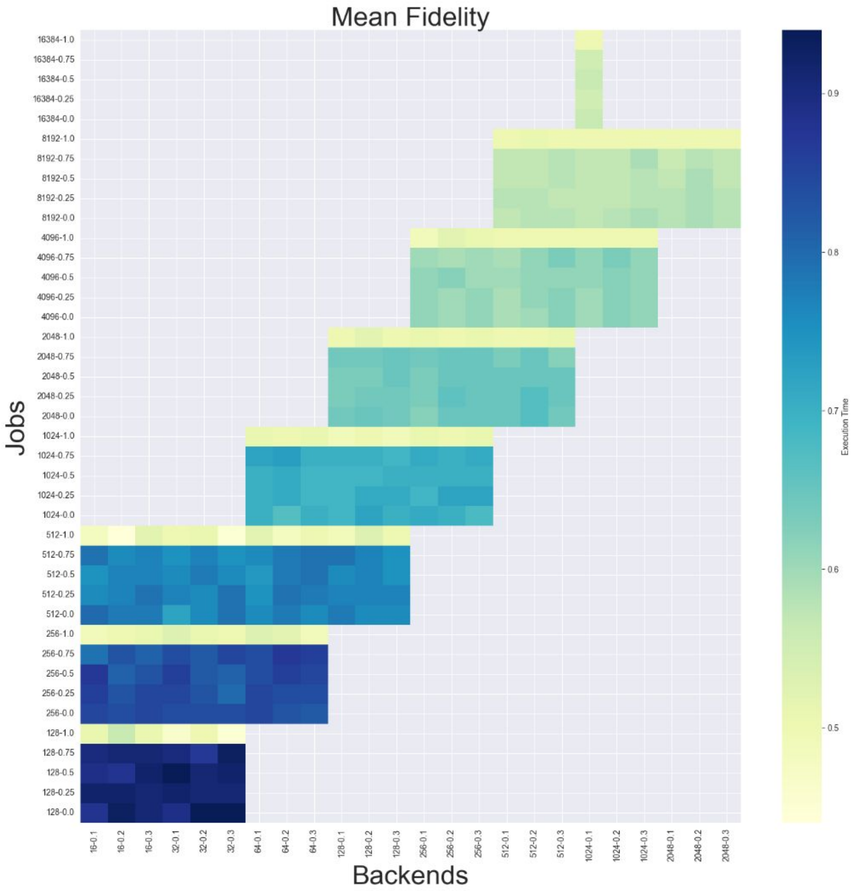
- What is the performance of every optimizer?
    - system time, CPU ops/ns, parallelizability
- How good are the generated schedules?
    - Fitness assessment of max. makespan, avg. fidelity
- How scalable is the optimizer?
    - Benchmarking on multi-core machine, combining the above metrics

Evaluation

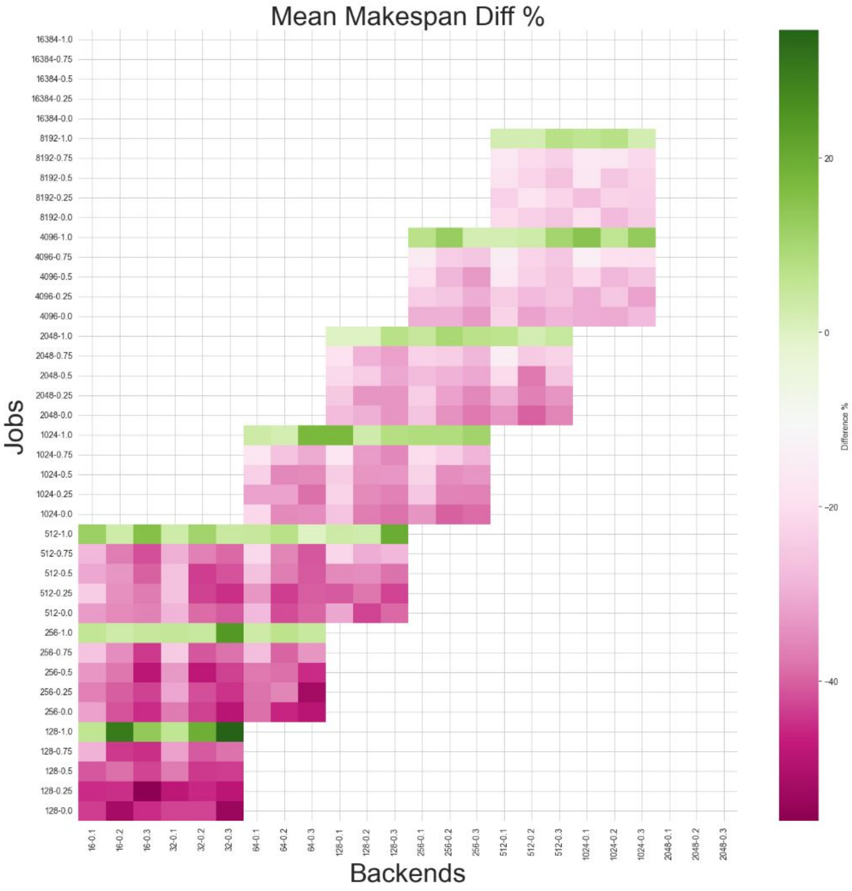
- Experimental setup:
  - Apple M3 Pro CPU (2.75-4.06 GHz, 11 cores)
  - 18 GB DRAM
  - 192+128 KiB L1 Cache, 16 MiB L2 Cache
- Synthetic data:
  - QPU initial waiting times
  - Job-on-QPU runtimes, fidelities

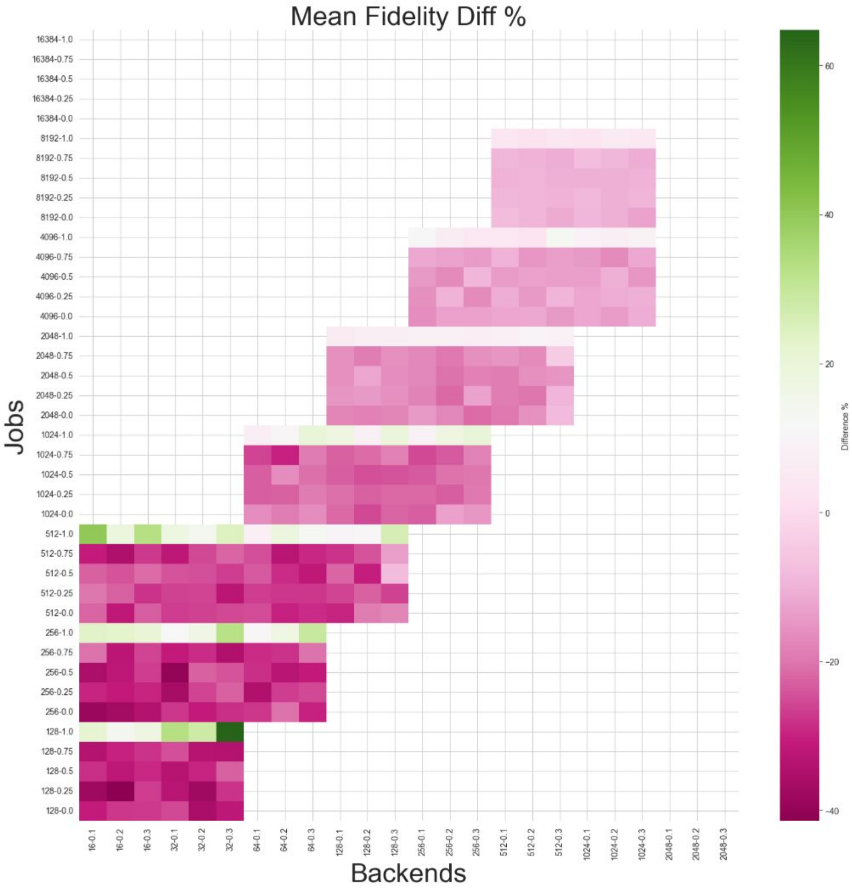
GA performance





NSGA-II performance





Summary

Variety of valid approaches for quantum schedule optimization

- MIP optimizer too slow for practical use
- GA optimizer promising for real life applications
- NSGA-II offers transparent tradeoff handling, yet requires fine-tuning

Future work:

- Fine-tuning the NSGA-II
- Parallelization models for GA (Island, Grid, …)
- Further algorithms (bin-packing like in Kubernetes)

Try it out!

https://github.com/mpraski/genetic-quantum-scheduler