Quantastica algorithm generator and transpiler

Quick Start Guide

Quantastica's "Quantum Algorithm Generator" and "Quantum Circuit Transpiler" are software tools under development, currently in the "beta" stage and available via "Early Access Program" for interested parties who are willing to test the products and provide feedback.

Both tools are available as standalone separate products, but are also integrated with the "Quantum Programming Studio" (QPS) for easy access, available via QPS web UI or via QPS Python API.

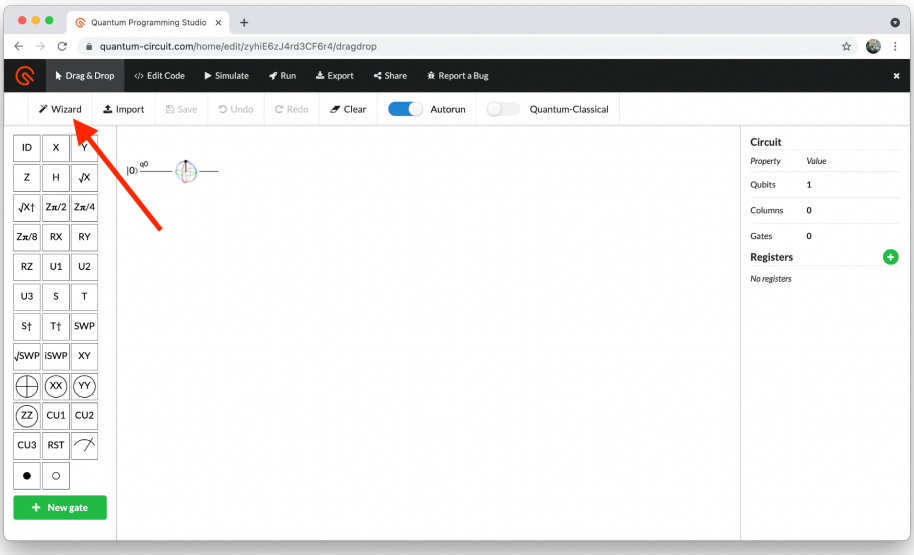
For more information on how generator works, see https://quantastica.com/generator/index.html

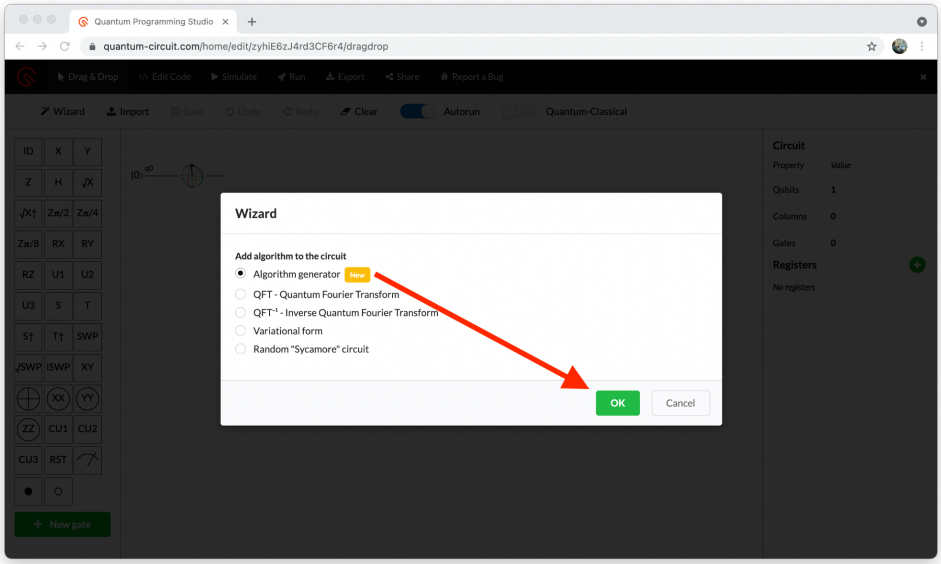
Access via QPS web UI

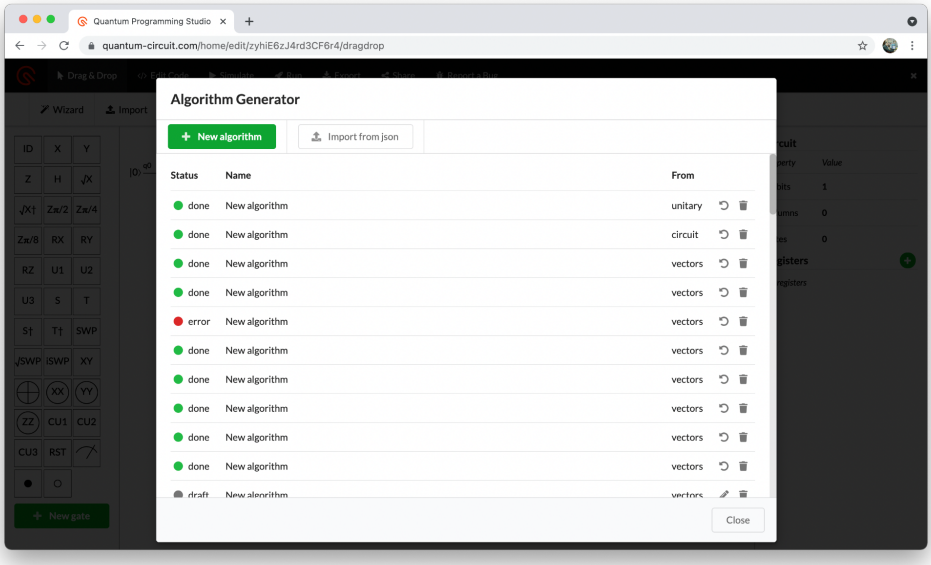
Users with granted access can use "Quantum Algorithm Generator" and "Quantum Circuit Transpiler" via "Quantum Programming Studio" web UI, available at:

https://quantum-circuit.com

Assuming that you already have registered your QPS username and password, please login and create a new project. Generator UI can be accessed via "Wizard" button as shown in screenshots:







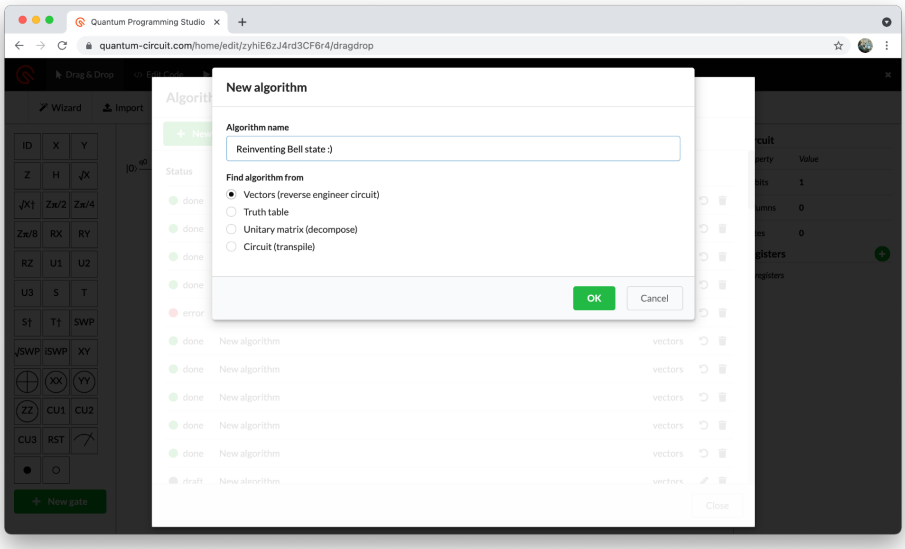
Now, the screen contains a list of your previous jobs (which is empty when you access it for the first time).

Using the generator

Algorithm generator is the tool which reverse-engineers quantum algorithms from classical data (example inputs and outputs - training set) encoded into state vectors (wavefunctions).

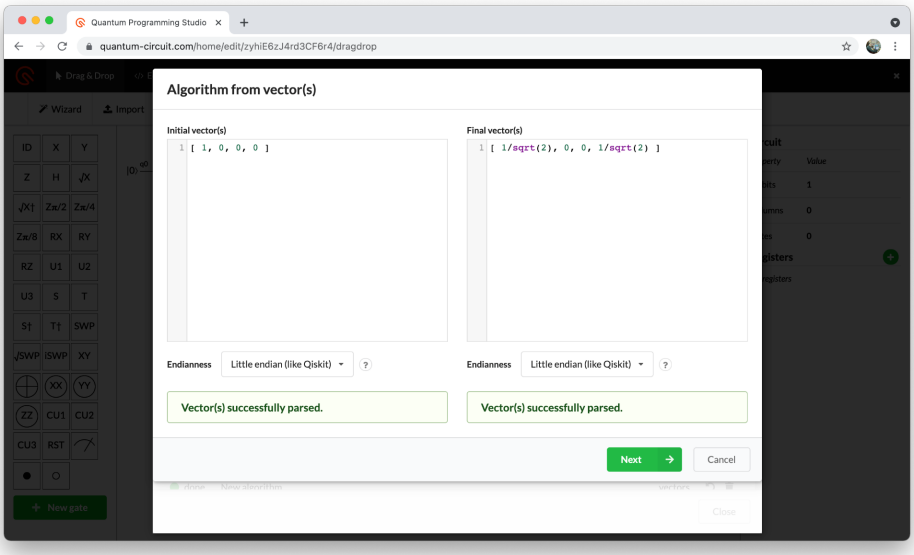
In order to access the generator, click the “New algorithm” button, then in the dialog give it a name and choose “Vectors (reverse engineer circuit)”.

See screenshot:



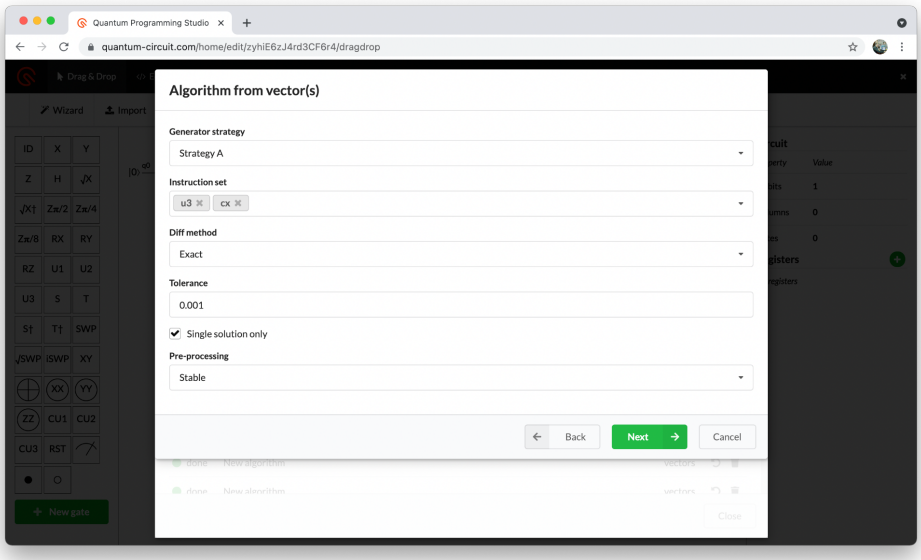
In the next screen, you can enter state vectors. Input vector (initial state) goes into the left editor, and output vector (final state) goes into right editor. Note that you need to enter each vector as a single line of text. You can enter multiple pairs of vectors (multiple input and output vectors), one vector per line.

Vectors can contain complex numbers and simple math expressions. Form appears pre-filled with 2-qubit zero state as initial vector and bell state as final vector:



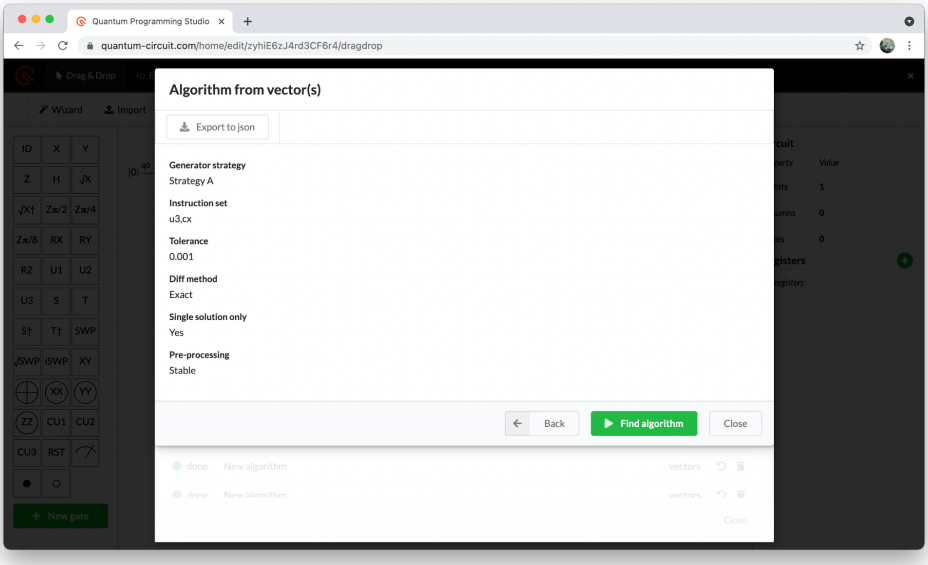
Click the “Next” button.

In the next screen, you can choose various parameters: strategy, target instruction set (gates), diff method (exact, ignore phase etc.), tolerance, pre-processing method, etc. More about parameters later.



For now, let's keep default values and click the "Next" button.

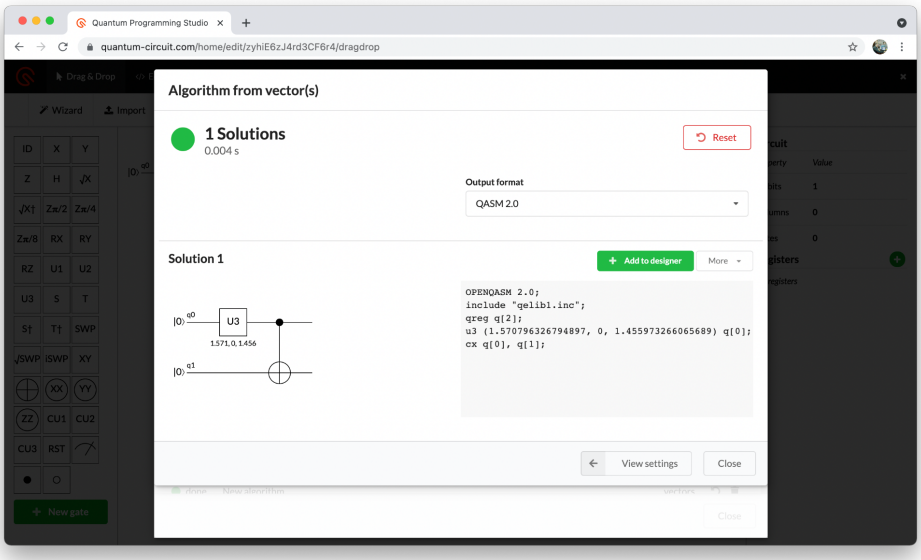
In next screen you will see the summary as a last step before executing a generator:



Before you start the generator, you can use the "Back" button to go back and change settings if needed.

To start the generator click the "Find algorithm" button, and the screen will be in status "running" with spinning hourglass.

When generator finds the solution, the resulting quantum circuit(s) will appear:



Long running jobs

Note that, for more complex problems it can take a long time to find a solution. From seconds or minutes, sometimes hours. It depends on the chosen method and parameters, and also on the complexity of the problem.

If you provide a combination of vectors which is impossible to solve, then note that some strategies are not able to detect if there is no solution and your job will run forever.

You can of course stop the job by clicking on the “Stop” button.

If it takes a long time, you don’t need to keep this screen open: if you close the window, even if you logout or completely close the browser, your job will continue to run on our servers. You will be able to see status and results next time when you come back to this screen.

While a job is running, you cannot start another job concurrently - it will be added to the queue and will be executed automatically after the previous job completes, fails or if you stop it manually.

It is good to know that even if you use the generator via Python API, your jobs will automatically appear in this UI where you can control/stop them.

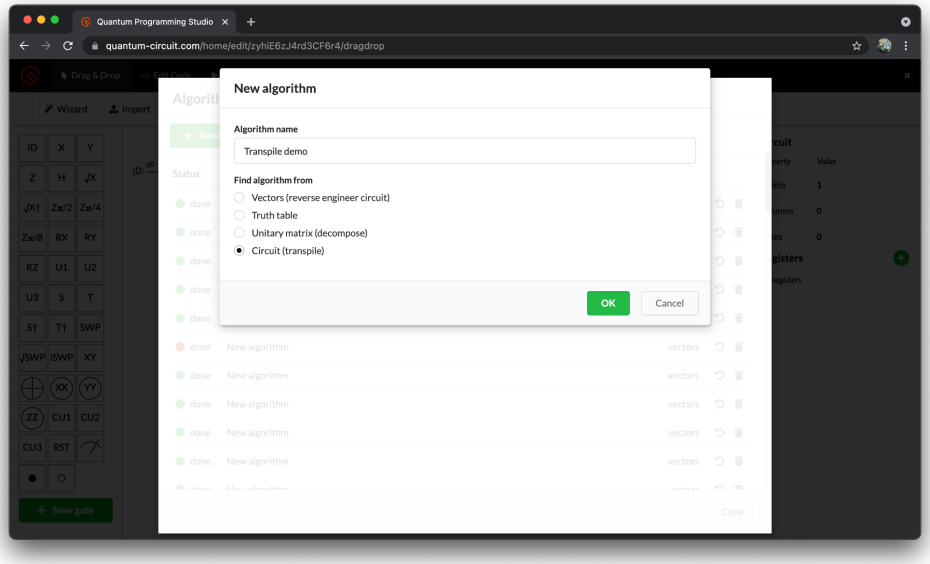
Using the transpiler

“Quantum Circuit Transpiler” is the tool which takes a quantum circuit at input and returns an equivalent quantum circuit with a different instruction set (set of quantum gates), potentially shallower than original circuit and hopefully with less 2-qubit gates.

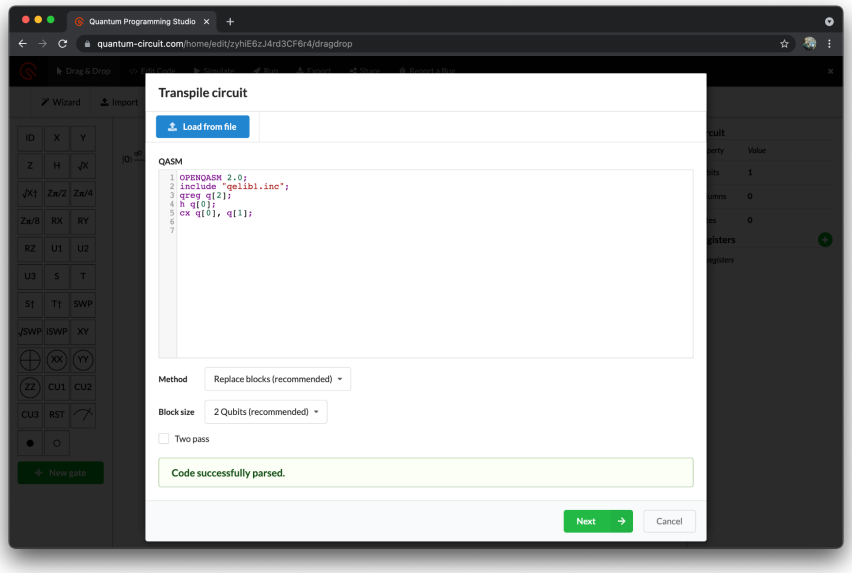
Transpiler can also take a unitary matrix at input which will be decomposed to an instruction set of your choice and returned as a quantum circuit.

Transpiling a circuit

In “New algorithm” screen, choose “Circuit (transpile)” and click the “OK” button:



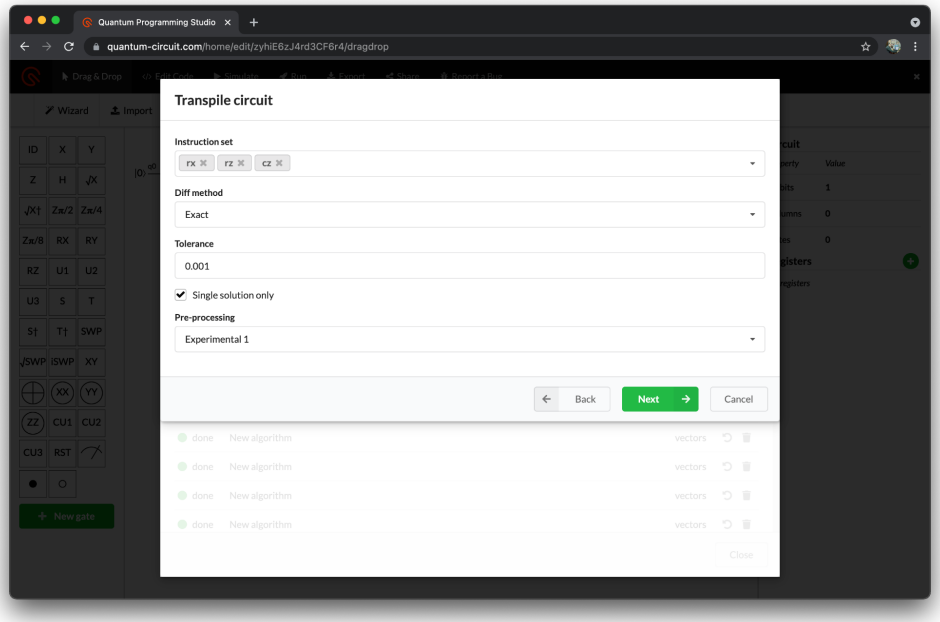
In the next screen, you need to provide a circuit in OpenQASM 2.0 format. You can enter it manually, paste it from clipboard or use the “Load from file” button to upload a qasm file from your computer:



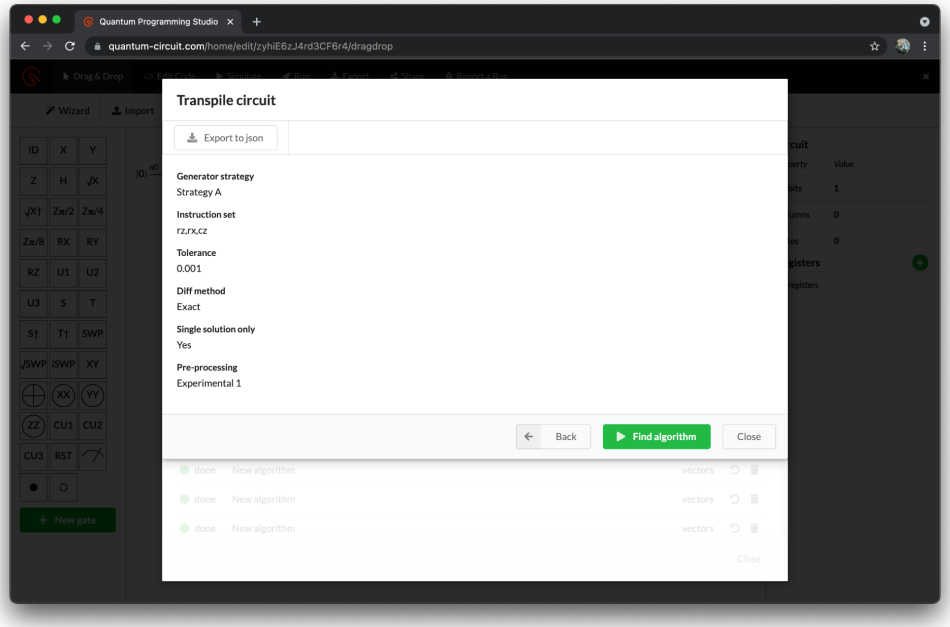
Note that only standard gates defined by QASM standard are accepted, plus extra gates in use by Quantum Programming Studio.

Below the code input box, you can choose the transpiling method and additional parameters. For now, you are good to go with default settings.

Hit the “Next” button and in the settings screen you can choose various parameters: target instruction set (gates), diff method (“exact”, “ignore phase” etc.), tolerance, pre-processing method, etc. More about parameters later.

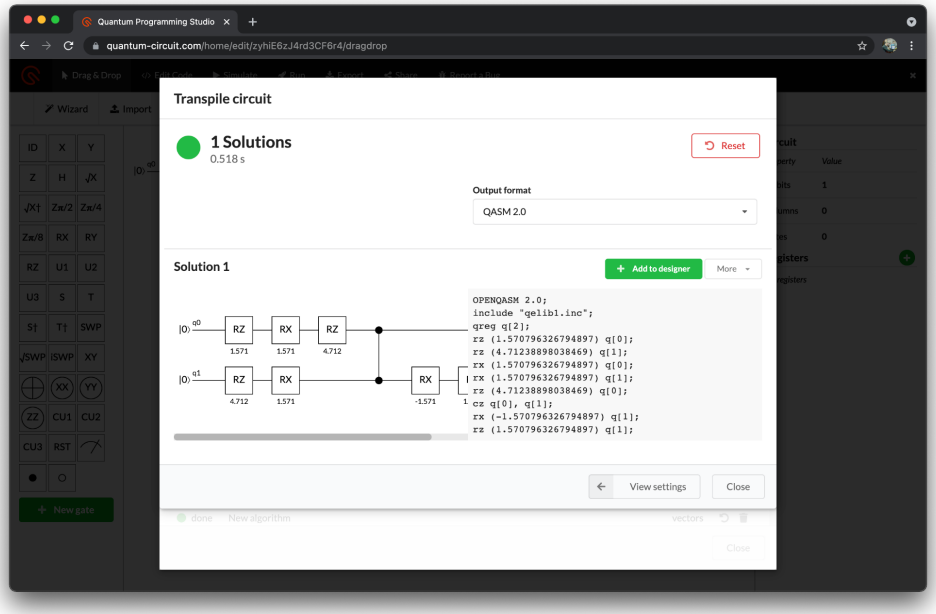


In next screen you will see the summary as a last step before executing a transpiler:



To start the transpiler click the “Find algorithm” button, and the screen will be in status “running” with spinning hourglass.

After transpiler finds the solution, the resulting quantum circuit will appear:

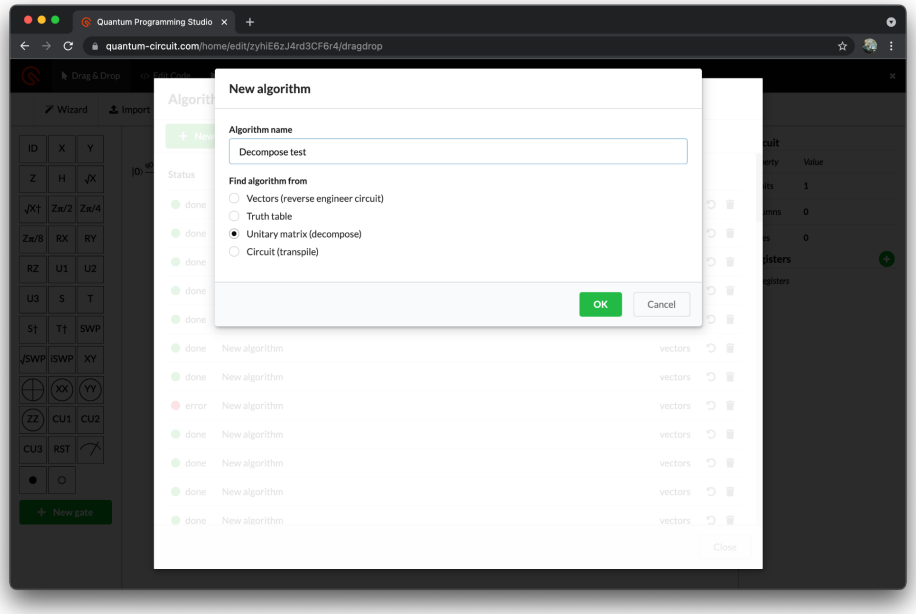


You can export the resulting circuit to multiple different programming languages, or add the circuit to the currently active QPS project by clicking on the “Add to designer” button, where it can be edited and executed.

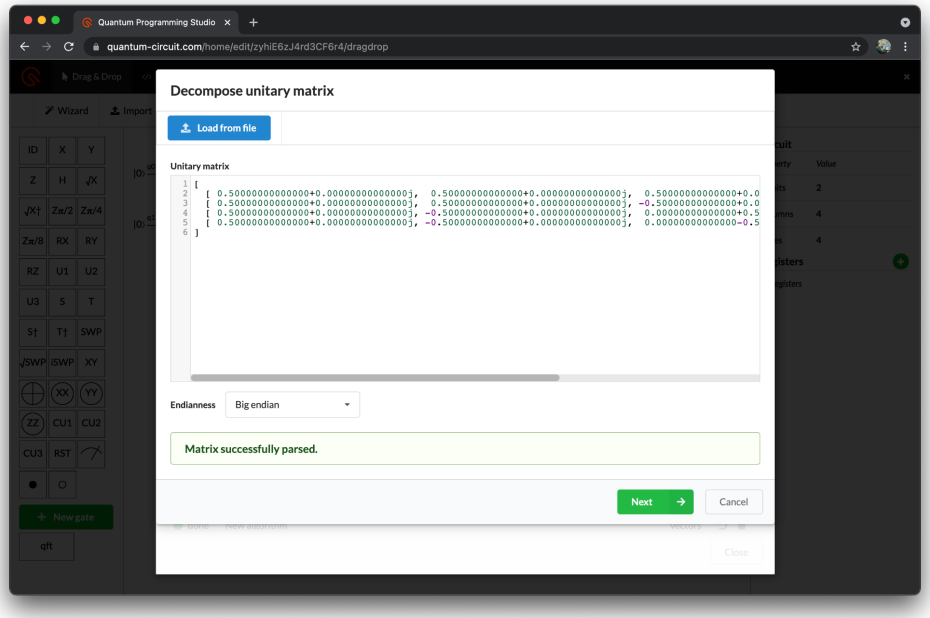
Decomposing an unitary matrix

Instead of providing the input circuit, you can also provide a unitary matrix. In that case the transpiler will act as a “decomposer” - it will return an equivalent quantum circuit.

In “New algorithm” screen, choose “Unitary matrix (decompose)” and click the “OK” button:



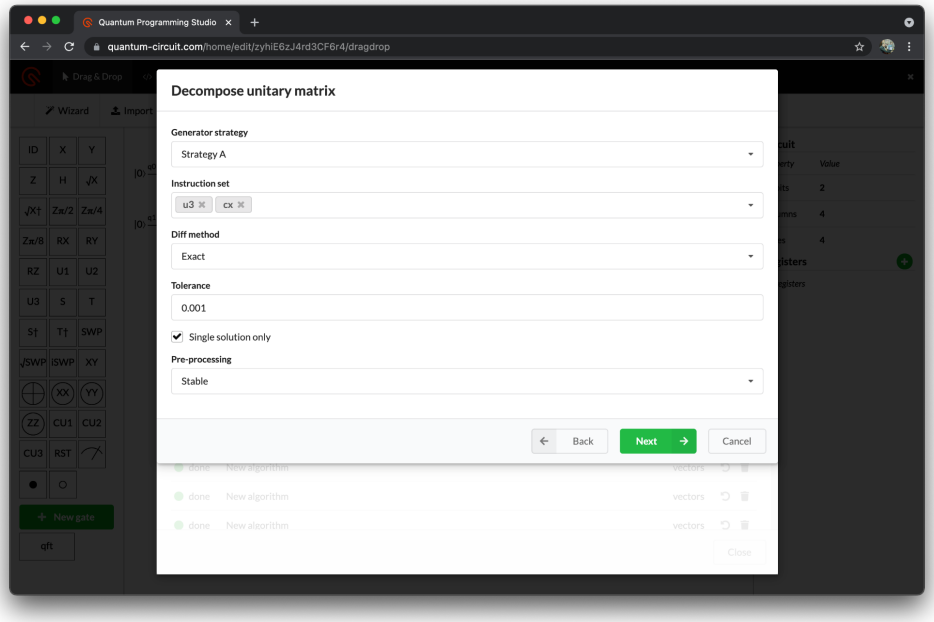
Hit the “OK” button to enter the matrix input screen.



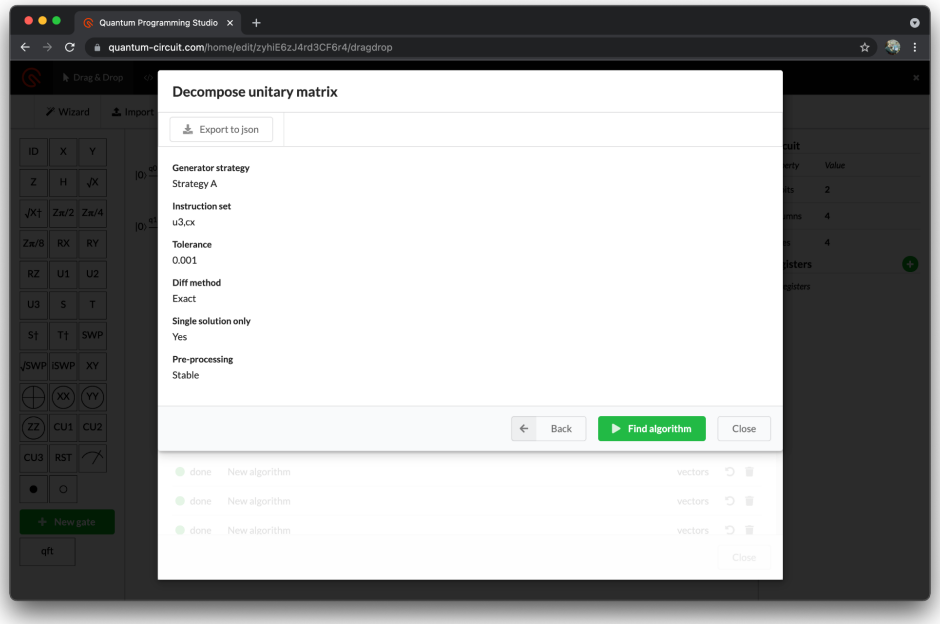
You can manually enter/paste the matrix or import it from a text or csv file. Complex numbers can be suffixed by “i” or by “j”, so 1+0i and 1+0j are both valid. Brackets are not mandatory, but

in that case you need to enter each row in a separate line of text. If you have brackets then the matrix can be entered both as a single line or as a multi-line string.

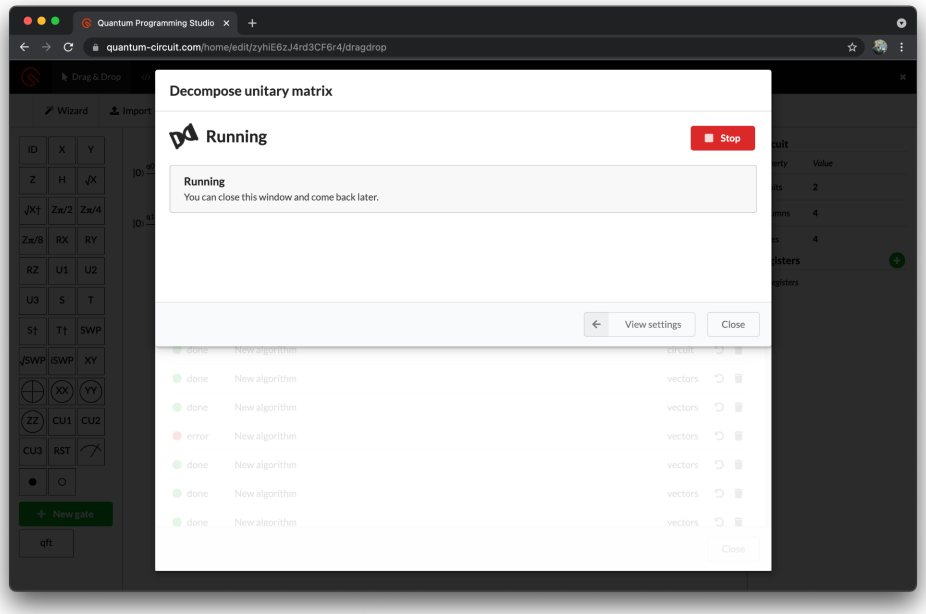
When the matrix is entered, click the “Next” button to enter settings screen where you can choose various parameters: strategy, target instruction set (gates), diff method (“exact”, “ignore phase” etc.), tolerance, pre-processing method, etc. More about parameters later.



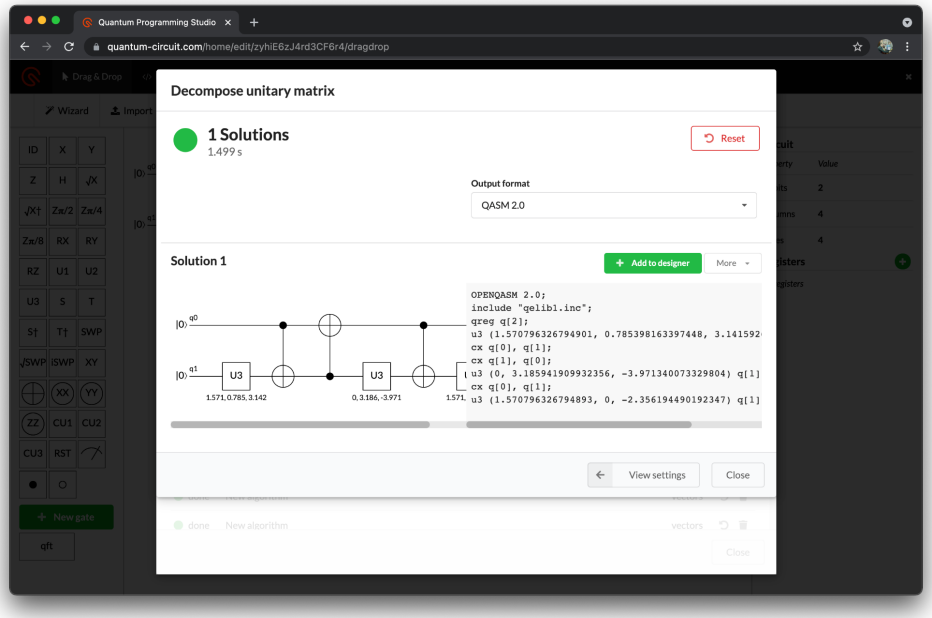
In next screen you will see the summary as a last step before executing a decomposer:



To start the decomposer click the “Find algorithm” button, and the screen will be in status “running” with spinning hourglass:



After transpiler finds the solution, the resulting quantum circuit will appear:



You can export the resulting circuit to multiple different programming languages, or add the circuit to the currently active QPS project by clicking on the “Add to designer” button, where it can be edited and executed.

Long running jobs

Transpiler UI appears very similar to generator from previous section, so repeating the same instructions here:

For deeper circuits with more qubits and larger matrices it can take a long time to find a solution. From seconds or minutes, sometimes hours. It depends on the chosen method and parameters, and also on the complexity of the problem.

You can of course stop the job at any time by clicking on the “Stop” button.

If it takes a long time, you don’t need to keep this screen open: if you close the window, even if you logout or completely close the browser, your job will continue to run on our servers. You will be able to see status and results next time when you come back to this screen.

While a job is running, you cannot start another job concurrently - it will be added to the queue and will be executed automatically after the previous job completes, fails or if you stop it manually.

It is good to know that even if you use the transpiler via Python API, your jobs will automatically appear in this UI where you can control/stop them.

Access via Python API

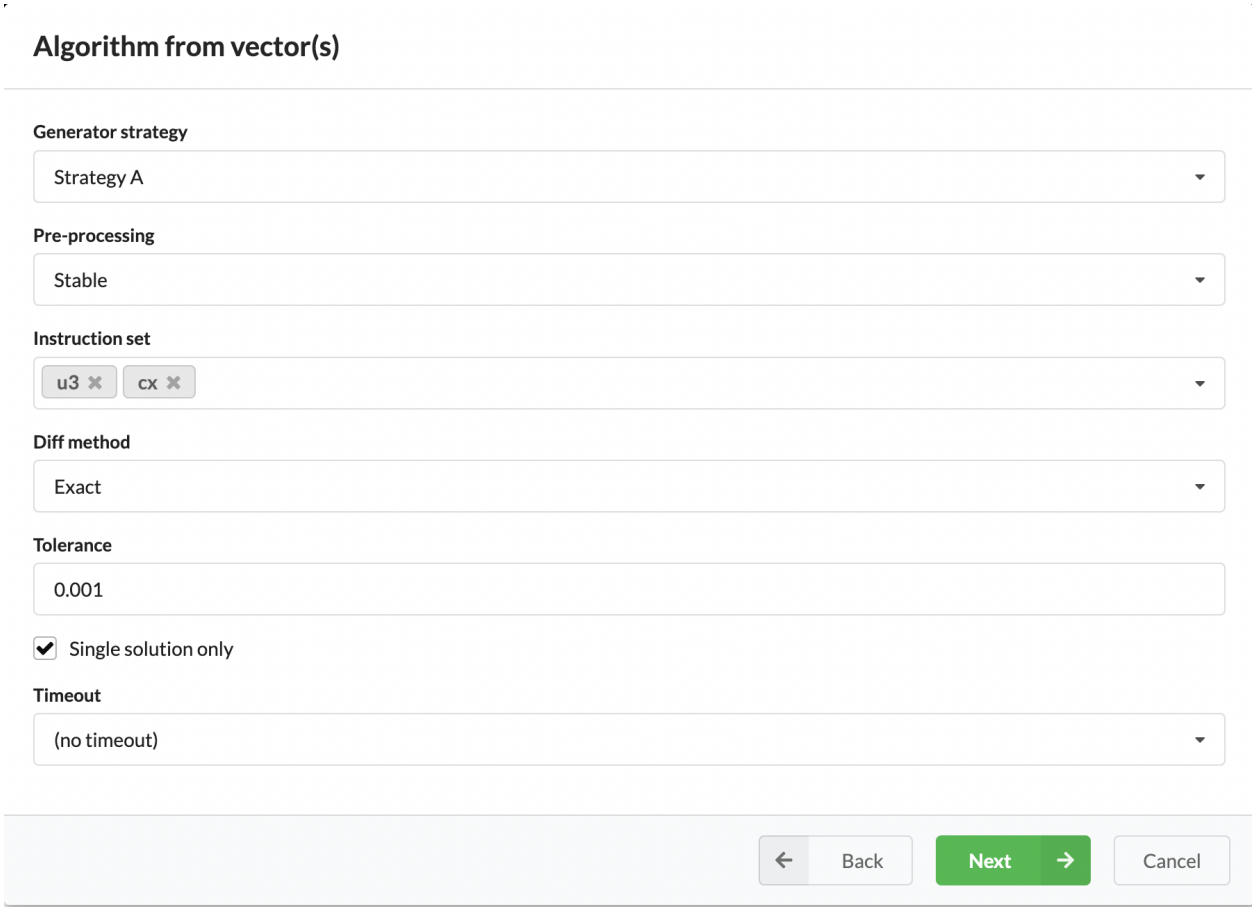
Both generator and transpiler can be used directly from python code without the QPS UI. For that purpose we have built a Python wrapper for the QPS REST API.

For installation instructions and documentation, please refer to:

https://pypi.org/project/quantastica-qps-api/

Choosing the best strategy, method and settings

We have implemented multiple methods for solving problems, and which one is best for your particular case depends on size and complexity of the input. It is best if you start with default values, and if the generator takes too long time to execute or if results are not optimal, you can try to change the strategy and parameters.



Generator strategy

- “Strategy A” is brute force, heuristics or heuristics supported by ML (depending on “pre-processing” parameter)

- “Strategy B” is using math+ML to find a solution

Both strategies have advantages and disadvantages, so it is best if you experiment and see what gives you the best results.

Pre-Processing

- “Stable” is a brute force algorithm. Gives optimal results but can be used only with small problems (only if resulting circuit will have small `num_qubits*num_gates`).

- “Experimental 1”, “Experimental 2”, “Experimental 3” and “Experimental 5” are different heuristic methods supported b machine learning and can reach bigger output circuits.

NOTE: Generator is compiled for maximum problem size of 6 Qubits. We can eventually provide the tool with support for more qubits on your request.

Instruction set

Set of gates to use for assembling the resulting circuits. Available only for “Strategy A”. Default is “u3, cx” but you can choose any set of gates. It is best if you choose a small set of gates (recommended 2-4 gates max).

Diff method

Method to use for evaluating cost (distance between target and candidate input vectors or matrix).

- “Exact” - searching for a solution with an exact global phase.
- “Ignore global phase” - searching for a solution with global phase ignored.
- “Ignore global phase HS” - (for matrices only) global phase is ignored and matrices are compared using Hilbert-Schmidt method (this is recommended diff method for decomposing matrices).
- “Absolute value” - both local and global phases are ignored (elements of a target and resulting vector will equal only by absolute values).

Tolerance

Generator will stop and return solution when difference between target and resulting vector or matrix reaches tolerance value.

Single solution only

Only for “Strategy A” with the “Stable” method: when checked (recommended!), the generator will return a single circuit. When unchecked, the generator will search all possible arrangements of gates and will return all valid circuits (this can take a very long time).

Timeout

If no solution is found, the generator will stop after specified time with a “timeout” error message. If the timeout is 0, the generator will run until a solution is found (or forever if there is no solution).