scientific reports

OPEN

GASP: a genetic algorithm for state preparation on quantum computers

Floyd M. Creevey[1], Charles D. Hill[1,2] & Lloyd C. L. Hollenberg[1]✉

The efficient preparation of quantum states is an important step in the execution of many quantum algorithms. In the noisy intermediate-scale quantum (NISQ) computing era, this is a significant challenge given quantum resources are scarce and typically only low-depth quantum circuits can be implemented on physical devices. We present a genetic algorithm for state preparation (GASP) which generates relatively low-depth quantum circuits for initialising a quantum computer in a specified quantum state. The method uses a basis set of R_x, R_y, R_z, and CNOT gates and a genetic algorithm to systematically generate circuits to synthesize the target state to the required fidelity. GASP can produce more efficient circuits of a given accuracy with lower depth and gate counts than other methods. This variability of the required accuracy facilitates overall higher accuracy on implementation, as error accumulation in high-depth circuits can be avoided. We directly compare the method to the state initialisation technique based on an exact synthesis technique by implemented in IBM Qiskit simulated with noise and implemented on physical IBM Quantum devices. Results achieved by GASP outperform Qiskit's exact general circuit synthesis method on a variety of states such as Gaussian states and W-states, and consistently show the method reduces the number of gates required for the quantum circuits to generate these quantum states to the required accuracy.

Quantum state preparation is key in many applications that require data input, such as finance[1–3], chemistry[4–6], bioinformatics[7], machine learning[8,9], and optimisation[10–12]. The ability to generate arbitrary quantum states efficiently and effectively, with high accuracy and low depth and gate count, will therefore be important in the future applications of quantum computing. Existing state preparation techniques[13–17] typically produce lengthy circuits with many qubit operations, or quantum gates, compromising their implementation on near-term hardware. In this paper, we present a genetic algorithm for state preparation (GASP) which creates circuits for state preparation to a specified accuracy and depth in an evolutionary framework. This work builds on a relatively old idea on the application of genetic algorithms to quantum circuit evolution[18]. For benchmarking purposes, of the methods that produce circuits for exact state preparation[13,19,20], we focus here on the method by Shende, Bullock, and Markov (SBM)[14], which has been encoded in the Qiskit library[21]. The SBM approach produces an n-qubit arbitrary state using a circuit containing no more than $2^{n+1} - 2n$ CNOT gates. We benchmark the GASP method against Qiskit[21] for Gaussian states and W-states in the context of simulation with gate noise and implementation on physical devices. Because the GASP circuits have much lower depth by design, the prospects for implementation on physical Noisy Intermediate Quantum (NISQ) devices are better, as we will show by explicit examples.

A genetic algorithm is a classical optimisation technique that aims to mimic the process of biological evolution[22]. The basic structure of a genetic algorithm is to first establish a population of individuals to evolve. This population can be many individuals or only a single individual. Generic genetic algorithms usually have many individuals and use crossover, however, if the population has only a single individual it is asexual, and only uses mutation. Each individual is an attempted solution to the given problem and is defined by a 'chromosome' where each parameter in the chromosome represents a 'gene'. This population is then subject to an iterative process where the individuals are selected by some selection criteria (rank selection, roulette wheel selection, etc.), and the selected individuals are bred together, and/or mutated. Each iteration of the genetic algorithm is referred to as a 'generation'. For each generation, the fitness of each individual in the population is evaluated, by some objective fitness function, and the fittest individuals have the highest probability of being bred together, and/or mutated[23]. These individuals then form the next generation of the algorithm, in an effort to iteratively increase the maximum fitness. Generally, the algorithm is completed when a specific fitness is achieved, or when a given number of iterations are run without achieving a specific fitness. This method of optimisation allows solutions to be pulled out of local optima, and move towards global optima. The important aspects of the genetic algorithm approach are as follows (see section "Methods" for further details):

[1]School of Physics, University of Melbourne, Melbourne 3010, Australia. [2]School of Mathematics and Statistics, University of Melbourne, Melbourne 3010, Australia. ✉email: lloydch@unimelb.edu.au

Initial population/individual.   In a sexual genetic algorithm the initial population is randomly generated with a specified population size parameter. Ideally, there is a considerable amount of divergence between different individuals, to allow for a broad area of the problem search space to be covered. In an asexual genetic algorithm, the initial population is only a single individual. Each individual in the population is instantiated with a chromosome defining a set of genes, the representation of these is subject to the problem being solved. Examples of possible chromosome representations include binary representations and list representations.

Crossover.   Crossover is the process where two parents are bred together, producing a child that contains one-half of its genomic information from one parent, and one-half from the other. There are various methods of crossover in genetic algorithms, the main two being single-point crossover, and k-point crossover. The crossover point is where in the chromosome 'list' the chromosome is split. Single-point crossover is where the crossover point is chosen and the child's genomic information is taken from one parent before the crossover point, and one parent after the crossover point. k-point crossover is a slightly more generalised version of this process, where there are k crossover points and the genomic information of the child ends up being $k + 1$ sections alternating between parents[24].

Mutation.   Mutation is the process where individual genes in an individual are randomly mutated, by some probability p, to increase genetic diversity in the population. In sexual genetic algorithms, the mutation rate is set low, as high mutation rates tend towards a primitive random search. In asexual genetic algorithms, the mutation rate is generally higher as it is the only means by which to introduce genetic diversity in the individual. The mutation operation allows the genetic algorithm to have increased genetic diversity, increasing the search space and allowing the algorithm to potentially escape local minima.

Selection.   Selection is the process by which the fittest individuals are chosen for crossover. The procedure generally involves assigning a probability of selection to each individual based on their fitness. There are many selection methods; common methods include roulette wheel selection, rank selection, and tournament selection. In *roulette wheel selection* each individual is given a probability of being selected dependent on their fitness. This allows the fittest individuals to have the highest probability of passing genes on to the next generation. It also allows lucky unfit individuals to pass their genes on to the next generation, increasing genetic diversity. The advantage of this method is no genetic material is conserved. *Rank selection* sorts the individuals by fitness and chooses the fittest individuals for crossover. The advantage of this method is you can more quickly converge to an optimal solution by virtue of only taking the fittest individuals. *Tournament selection* randomly pairs individuals and selects the individual with the higher fitness of the two for crossover. This is an intermediary between the roulette wheel and rank selection, allowing potential faster convergence while maintaining a potentially higher level of genetic diversity.

In the quantum circuit space, the introduction of genetic algorithms has focused largely on producing circuits to generate specific unitaries[18,24–28], and has been applied to quantum machine learning[29]. The flowchart describing GASP is depicted in Fig. 1. In this work, the genetic algorithm is designed to evolve quantum circuits for the task of state preparation. In this context, an individual is a quantum circuit, and a gene is a single quantum logic operation (gate) in the given quantum circuit.

The structure of the paper is as follows. Section "Results" will present the results, section "Methods" will describe the method presented in detail, and section "Discussion" our conclusions and potential future work.

Results
Gaussian states.   Gaussian states have relatively low entanglement and are defined as,

$$|\psi_G\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} g(x)|x\rangle, \tag{1}$$

where $g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$, μ is the mean, and σ is the standard deviation of the desired Gaussian. For the purposes of this paper, we let $\mu = \frac{2^n}{2}$, and $\sigma = \frac{2^n}{8}$. In Fig. 2a we show an example GASP circuit, comparing the depth and gate count with that produced by Qiskit's initialise function. A comparison of the states produced by Qiskit's initialise function, and GASP, in the absence of noise, is shown in Fig. 2b for a 6 qubit Gaussian state. In the zero-noise regime, the data shows that the Qiskit method produces the target exactly as expected, and the GASP is within the specified fidelity tolerance (99% in this case). For the same 6 qubit Gaussian, a comparison of the states produced by Qiskit's initialise function, and GASP simulated, in the presence of noise (modelled from *ibmq_guadalupe*) is shown in Fig. 2c, 16,384 shots were used. It can be seen that GASP performs much better in reproducing the desired target state vector in the presence of noise, though not perfectly. This shows that a slight reduction in the accuracy of the circuit, for a large reduction in circuit depth improves the desired outcome state in the presence of noise. GASP producing circuits shorter by orders of magnitude allow more realistic circuits to be implemented on NISQ-era hardware. Figure 2d shows the comparison between GASP and Qiskit for Gaussian states, in terms of gate scaling vs. the number of qubits. Figure 2e shows the improved performance of the circuits generated by GASP relative to Qiskit when simulated with noise, and run on IBM's *ibmq_guadalupe* machine, averaged over 10 tests as the number of qubits varied from 2 to 10 (noting that this measure does not include the phase information).
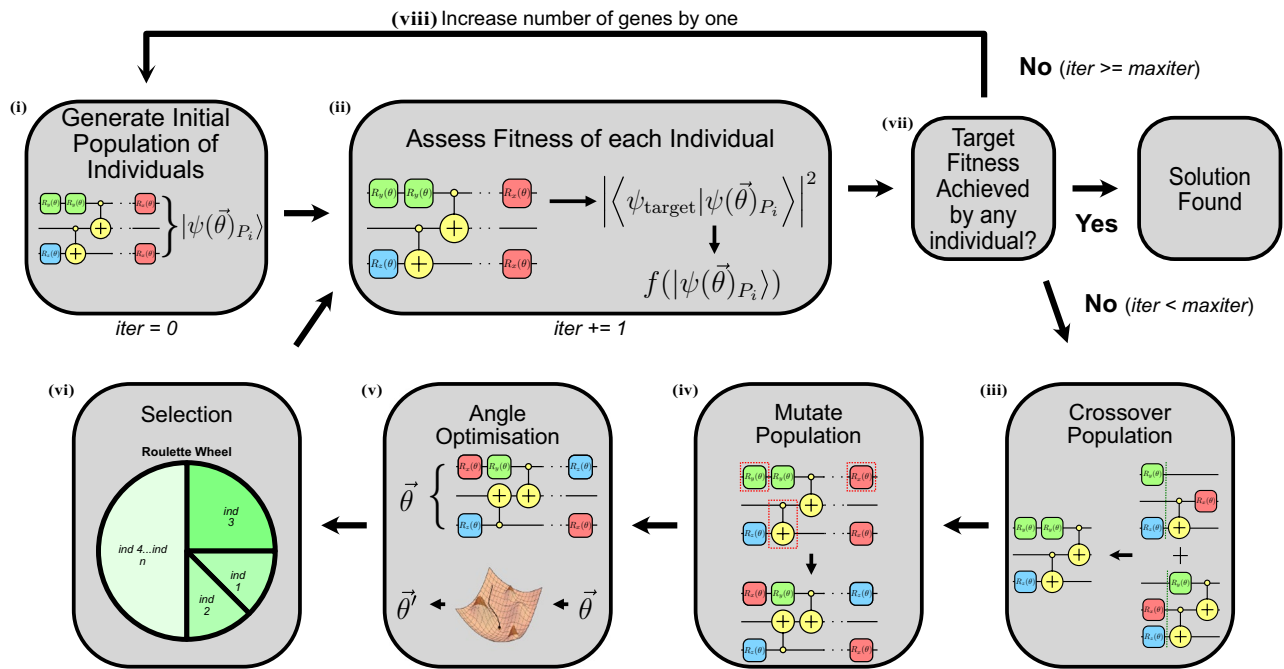
Figure 1. Overview of the GASP approach, given a desired target state vector $|\psi_{\text{target}}\rangle$. (**i**) Then, create initial population $\{P\}$, of individuals, P_i, which are each a quantum circuit, that generates the population states $|\psi(\vec{\theta})_P\rangle$, with the appropriate number of qubits and number of genes for the given state vector. (**ii**) Assess the fitness of the state vectors determined by each individual in the population: $f(|\psi(\vec{\theta})_{P_i}\rangle) = |\langle\psi_{\text{target}}|\psi(\vec{\theta})_{P_i}\rangle|^2$. (**iii**) Apply crossover to the population, producing $|\psi(\vec{\theta})_{\text{new}}\rangle$. (**iv**) Mutate the entire population with probability $p = 5\%$. (**v**) Run classical optimisation on each mutated individual to obtain the optimal θ values between 0 and 2π, to achieve the highest fitness for their generated circuit. (**vi**) Apply roulette wheel selection to the population, to select the individuals for the next generation based on their assessed fitness. (**vii**) Repeat until the desired fitness is achieved or *maxiter* iterations since the last increase in fitness was achieved. (**viii**) If *maxiter* iterations since the last increase in fitness, increase the number of genes by 1 and return to (**i**).

W-states.

W-states generally have higher entanglement and are defined as,

$$|\psi_W\rangle = \frac{1}{\sqrt{n}}(|100\ldots0\rangle + |010\ldots0\rangle + \cdots + |000\ldots1\rangle), \tag{2}$$

where n is the number of qubits. In Fig. 3a we show an example GASP circuit, comparing the depth and gate count with that produced by Qiskit's initialise function. A comparison of the produced states by Qiskit's initialise function, and GASP, with no noise, is shown in Fig. 3b. Figure 3c shows the comparison between Qiskit and GASP (99% fidelity) resultant distributions in the presence of noise for a 6 qubit W state. The noise model is that of IBM's *ibmq_guadalupe* machine. 16,384 shots were used. The grey dashed line is the exact distribution without noise, the orange bars are GASP, and the blue bars are Qiskit. Figure 3d shows the comparison between the Qiskit and GASP (99% fidelity) for W-states as the number of qubits varied from 2 to 10. Figure 3e shows the performance of the circuits generated by GASP and Qiskit when simulated with noise, averaged over 10 tests as the number of qubits varied from 2 to 10 noting that this measure does not include the phase information.

As can be seen in the results shown here, GASP consistently outperforms Qiskit in the number of total gates and the number of CNOT gates required; by more than two orders of magnitude in the higher qubit tests. It seems that GASP is a lower polynomial complexity in the number of gates required compared to Qiskit's initialisation method. It should be noted that the number of gates required for the Gaussian state generation is fewer than that of the W-states. This is likely due to Gaussian states being less entangled than W-states. It can be seen that the circuits produced by GASP have higher noise robustness than circuits produced by Qiskit's initialisation. However, the length of circuits produced by both techniques at high numbers of qubits have so many gates that the noise overwhelms the ability of the circuit to produce the desired state. It should also be noted that at useful fidelities (those above 50%), GASP outperforms Qiskit's initialisation.

Methods
In GASP, a gene is represented,

$$\text{gene} = [q_t^i, G^i, q_c^i, \theta^i], \tag{3}$$

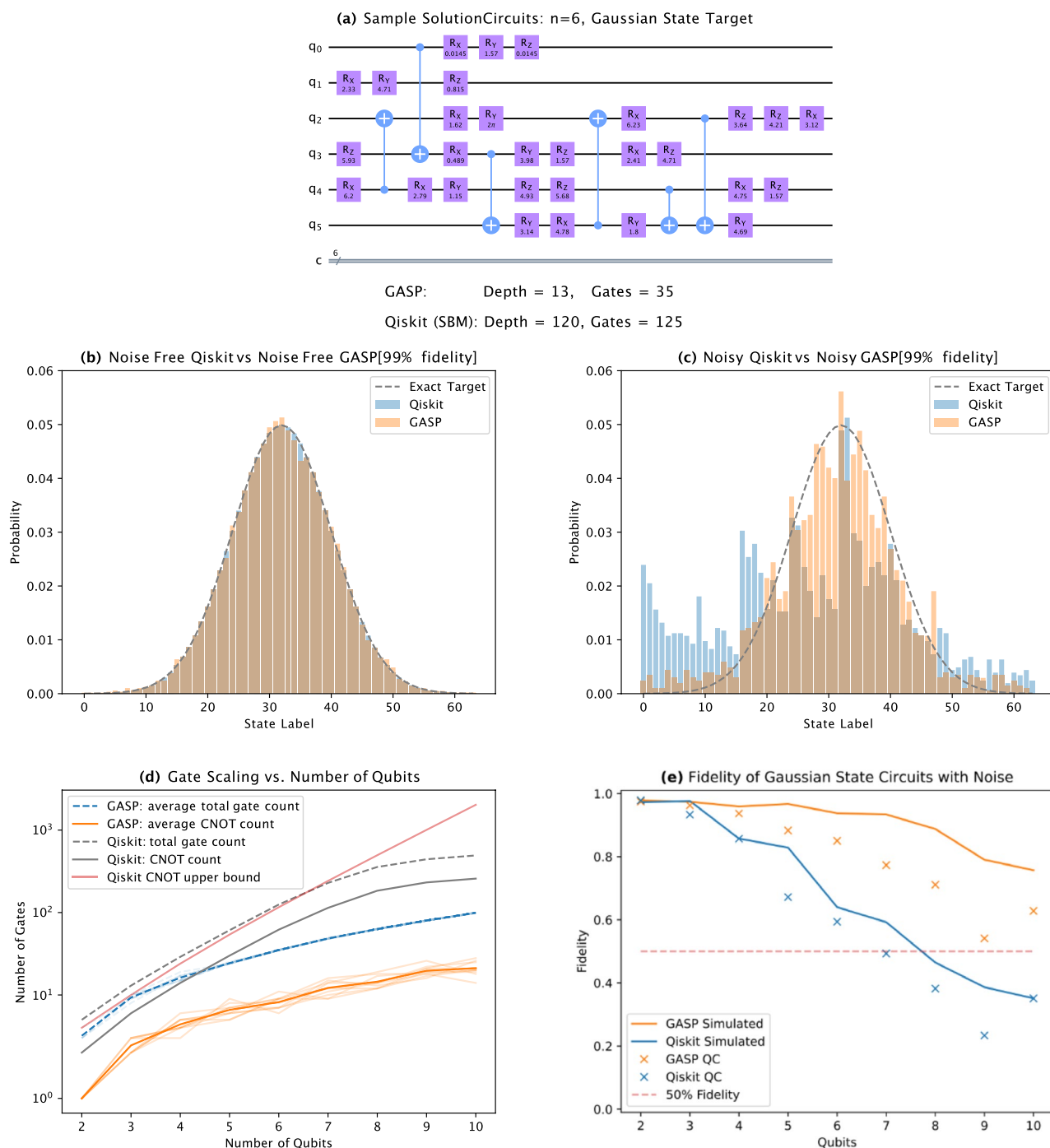and an individual is represented,

Figure 2. (**a**) Comparison between sample solution circuits generated by GASP and Qiskit for a 6 qubit Gaussian. GASP produced a circuit with a depth of 13 and 35 gates, Qiskit produced a circuit with a depth of 120 and 125 gates. Note: these are circuits before compilation on real hardware. (**b**) Comparison between Qiskit and GASP (99% fidelity) resultant distributions with no noise for a 6 qubit Gaussian state. The grey dashed line is the exact distribution, the orange bars are GASP, and the blue bars are Qiskit. (**c**) Comparison between Qiskit and GASP (99% fidelity) resultant distributions in the presence of noise for a 6 qubit Gaussian state. The noise model is that of IBM's *ibmq_guadalupe* machine. 16,384 shots were used. The grey dashed line is the exact distribution without noise, the orange bars are GASP, and the blue bars are Qiskit. (**d**) Gate comparison between the Qiskit and GASP (99% fidelity) as the number of qubits varied from 2 to 10. (**e**) Comparison between GASP (99% fidelity) and Qiskit for Gaussian states as the number of qubits varied from 2 to 10. Lines represent simulations with noise, crosses represent results from IBM's *ibmq_guadalupe* machine. 16,384 shots were used. The noise model is that of IBM's *ibmq_guadalupe* machine.
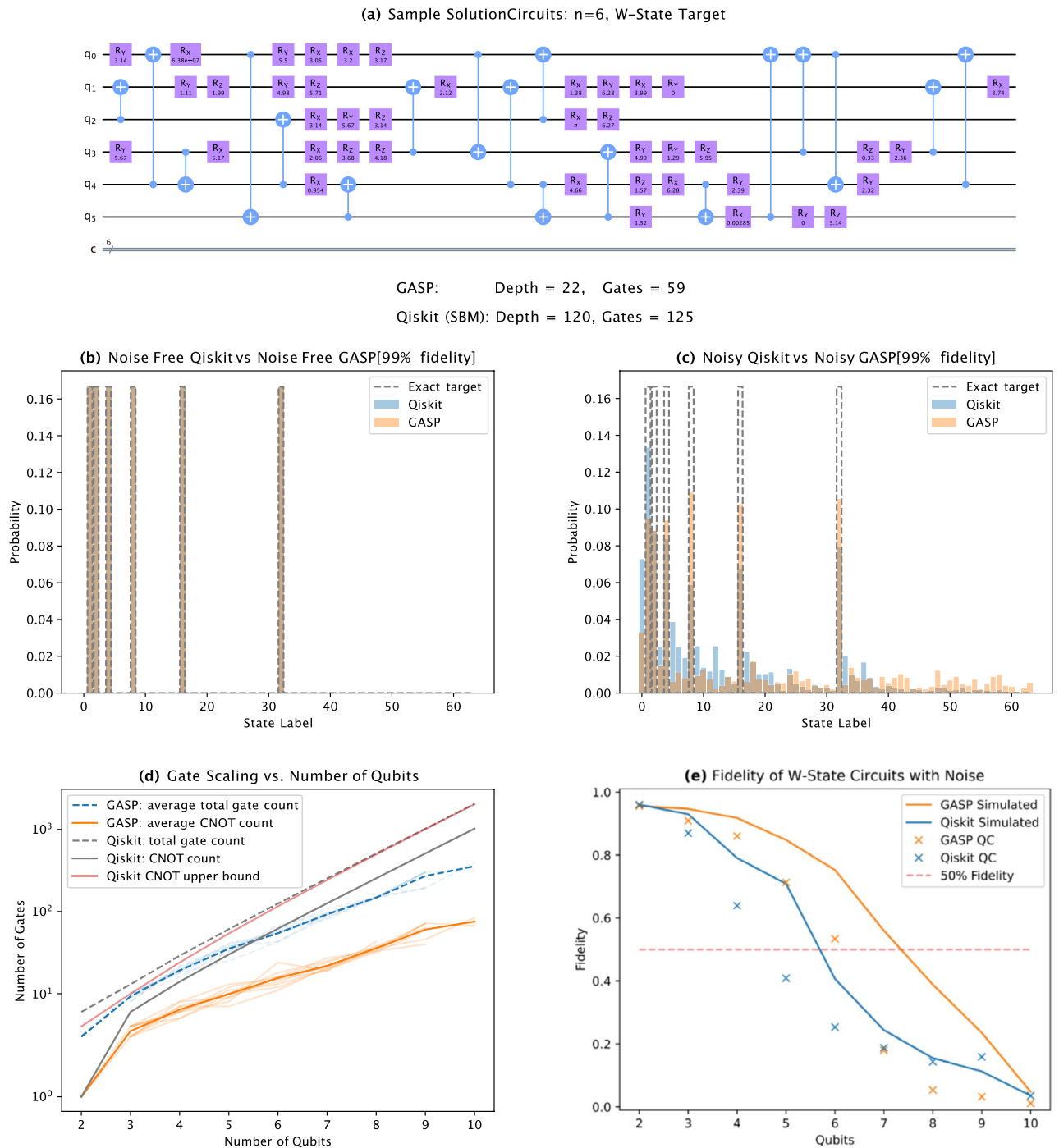
Figure 3. (**a**) Comparison between sample solution circuits generated by GASP and Qiskit for a 6 qubit W state. GASP produced a circuit with a depth of 22 and 59 gates, and Qiskit produced a circuit with a depth of 120 and 125 gates. Note: these are circuits before compilation on real hardware. (**b**) Comparison between Qiskit and GASP (99% fidelity) resultant distributions with no noise for a 6 qubit W state. The grey dashed line is the exact distribution, the orange bars are GASP, and the blue bars are Qiskit. (**c**) Comparison between Qiskit and GASP (99% fidelity) resultant distributions in the presence of noise for a 6 qubit W state. The noise model is that of IBM's *ibmq_guadalupe* machine. 16,384 shots were used. The grey dashed line is the exact distribution without noise, the orange bars are GASP, and the blue bars are Qiskit. (**d**) Comparison between the Qiskit and GASP (99% fidelity) for W-states as the number of qubits varied from 2 to 10. (**e**) Comparison between GASP (99% fidelity) and Qiskit for W-states as the number of qubits varied from 2 to 10. Lines represent simulations with noise, crosses represent results from IBM's *ibmq_guadalupe* machine. 16,384 shots were used. The noise model is that of IBM's *ibmq_guadalupe* machine.

$$P_i = [\text{gene}_1, \text{gene}_2, \ldots, \text{gene}_n], \tag{4}$$

where each gene ($i = 0 \ldots n$) represents a gate application, q_t is the target qubit, G is the chosen gate type from $\{R_x, R_y, R_z, \text{CNOT}\}$, q_c is the control qubit if any, and θ is the rotation angle in the chosen gate. For a single qubit gate, q_c is set to None. For a two-qubit gate q_c is set to another qubit in the circuit, and θ is set to None, allowing the classical optimisation to only optimisation single-qubit rotation angles. The number of genes in an individual is dependent on how many gates the circuit contains. As such, a mutation in the genetic algorithm would be the changing of certain gates with a given probability.

Crossover.   The crossover method used in GASP is a simple 1-point crossover with one half from each parent, applied to every generation (i.e. crossover point is 50%). In the quantum circuit context, this results in a new circuit containing half the gates from the first individual, and half the gates from the second individual.

Mutation.   Mutation in GASP is applied every generation. The default probability for mutation is 5%, however, this parameter can be varied dependent on the problem to be solved. The basic mutation of a quantum circuit changes certain genes in the individual to other genes, resulting in different gates for the quantum circuit, changing the resultant state vector.

Fitness.   Given the target state vector, $|\psi_{\text{target}}\rangle$, and the resultant state vector of the current individual, $|\psi(\vec{\theta})_{P_i}\rangle$, GASP searches for the individual whose circuit produces the highest fitness. The fitness is calculated by the cost function,

$$f(|\psi(\vec{\theta})_{P_i}\rangle) = |\langle\psi_{\text{target}}|\psi(\vec{\theta})_{P_i}\rangle|^2, \tag{5}$$

the norm squared of the inner product between the target state vector and the individual's state vector, which is the similarity between the target state vector and the individual's state vector. The fitness of the individual will always be a value between 0 and 1.

Selection.   The method of selection used in GASP is roulette wheel selection. This allows genetic diversity to be maintained through generations of the algorithm, while also increasing fitness. This is done by initially summing the total fitness of the entire population of individuals, f_T, then giving each individual a normalised fitness relative to the fitness of the entire population, $p(|\psi(\vec{\theta})_P\rangle)$, i.e,

$$f_T = \sum_i f(|\psi(\vec{\theta})_{P_i}\rangle), \tag{6}$$

$$p(|\psi(\vec{\theta})_{P_i}\rangle) = \frac{f(|\psi(\vec{\theta})_{P_i}\rangle)}{f_T}. \tag{7}$$

The individuals in the next generation are then selected based on their respective fitnesses.

Algorithm.   For a chosen $|\psi_{\text{target}}\rangle$, a population of individuals is produced each with a certain number of 'genes' based on the entanglement of the target state. The individual is the quantum circuit. Each 'gene' is one of the four gates identified in the universal set listed in section "Methods"; $\{R_x, R_y, R_z, \text{CNOT}\}$. The fitness of each individual is then assessed with the fitness function. Crossover is then applied to the population, to produce new individuals, doubling the population. Each individual in the population is then 'mutated' at a probability of 5%. SLSQP optimisation[30] is then run on each individual in the population to find the optimal θ's for each given individual. The population is then subject to roulette wheel selection, to select the individuals for the next generation, halving the population back down to its original size. This process is then repeated, iteratively increasing the best fitness of the population. If the desired fitness is not achieved within *maxiter* (1000 in the presented results) iterations of the last increase in fitness, the number of genes for each individual is increased by one, and the process restarts. Once a desired target state vector $|\psi_{\text{target}}\rangle$ (which also determines the number of qubits) is selected GASP can be broken down into the following steps:

a.   The initial population $\{P\}$, of individuals, P_i, is created, that dictates the trial state vectors $|\psi(\vec{\theta})_{P_i}\rangle$, with the appropriate number of qubits and number of genes for the given state vector
b.   The fitness of the state vectors determined by each individual in the population: $f(|\psi(\vec{\theta})_{P_i}\rangle) = |\langle\psi_{\text{target}}|\psi(\vec{\theta})_{P_i}\rangle|^2$ is assessed
c.   Crossover is applied to the population, generating $|\psi(\vec{\theta})_{\text{new}}\rangle$
d.   The entire population is mutated with probability $p = 5\%$.
e.   Classical optimisation is run on each mutated individual to obtain the optimal $\vec{\theta}$ values between 0 and 2π, which achieve the highest fitness for their generated circuit
f.   Roulette wheel selection is applied to the population, to select the individuals for the next generation based on their assessed fitness.
g.   Steps ii–vi are repeated until the desired fitness is achieved or *maxiter* (where *maxiter* is a parameter set prior to the start of the algorithm) iterations since the last increase in fitness was achieved

h. If *maxiter* iterations since the last increase in fitness are achieved, increase the number of genes by 1 and return to step i

In GASP, the genetic algorithm is being utilised to determine the course-grained optimisation of quantum circuit structure, while a much better fine-grained algorithm, SLSQP, was used for determining the optimal angles for each circuit structure generated. This allows the optimal, or at least close to optimal, fitness for each generated circuit structure to be achieved.

Discussion

In this paper, we have proposed and demonstrated a state preparation method based on a genetic evolutionary approach. Benchmarking GASP against Qiskit's initialisation method, the results show that in the noisy regime relevant to implementation on actual hardware, GASP significantly outperforms the exact approach through superior circuit compression, by more than an order of magnitude. As GASP is a stochastic algorithm, there is an increase in run time over the deterministic algorithms and an introduced uncertainty in the ability to produce a solution. However, the significant reduction in both the total gate count and the number of required CNOT gates may outweigh these for the application of GASP to the initialisation of quantum states in circuit lengths feasible on NISQ-era hardware. As GASP is a general technique that works for any given state, further work beyond this paper will include using GASP to generate more general quantum state preparation problems, such as amplitude-encoded image data, particle physics data, and biological data. Additionally, as this work is all classical simulation of quantum states, it would be interesting future work to develop a cost function so that GASP could be run on an actual quantum computer.

Note: During the preparation of this work, a recent paper by Rindell et al.[31] studying state preparation using a genetic algorithm was posted on the arXiv. Their method is similar to the GASP approach presented here, though differs in that they used a Fast Non-dominated Sorting Genetic Algorithm implemented in the DEAP Python package[32], and a different gate set of $\{R_z(\theta), X, \sqrt{X}, \text{CNOT}\}$. They also do not use classical optimisation, instead opting to adjust θ by adding a value from a selected Gaussian distribution. Their method was applied to the production of Haar random states up to 5 qubits, demonstrating similar fidelity improvements in the presence of noise.

Data availability

The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

References

1. Egger, D. J., García Gutiérrez, R., Mestre, J. C. & Woerner, S. Credit risk analysis using quantum computers. *IEEE Trans. Comput.* **70**, 2136–2145. https://doi.org/10.1109/TC.2020.3038063 (2021).
2. Egger, D. J. *et al.* Quantum computing for finance: State-of-the-art and future prospects. *IEEE Trans. Quantum Eng.* **1**, 1–24. https://doi.org/10.1109/TQE.2020.3030314 (2020).
3. Nakaji, K. *et al.* Approximate amplitude encoding in shallow parameterized quantum circuits and its application to financial market indicators. *Phy. Rev. Res.* **4**, 023136. https://doi.org/10.1103/PhysRevResearch.4.023136 (2022).
4. Bauer, B., Bravyi, S., Motta, M. & Chan, G.K.-L. Quantum algorithms for quantum chemistry and quantum materials science. *Chem. Rev.* **120**, 12685–12717. https://doi.org/10.1021/acs.chemrev.9b00829 (2020).
5. Cao, Y. *et al.* Quantum chemistry in the age of quantum computing. *Chem. Rev.* **119**, 10856–10915. https://doi.org/10.1021/acs.chemrev.8b00803 (2019).
6. Jones, M. A., Vallury, H. J., Hill, C. D. & Hollenberg, L. C. L. Chemistry beyond the Hartree–Fock energy via quantum computed moments. *Sci. Rep.* **12**, 8985. https://doi.org/10.1038/s41598-022-12324-z (2022).
7. Hollenberg, L. C. L. Fast quantum search algorithms in protein sequence comparison—Quantum biocomputing. *Phys. Rev. E* **62**, 7532–7535. https://doi.org/10.1103/PhysRevE.62.7532. arXiv:quant-ph/0002076 (2000).
8. Abbas, A. *et al.* The power of quantum neural networks. *Nat. Comput. Sci.* **1**, 403–409. https://doi.org/10.1038/s43588-021-00084-1. arXiv:2011.00027 [quant-ph] (2021).
9. Heredge, J., Hill, C., Hollenberg, L. & Sevior, M. Quantum support vector machines for continuum suppression in B meson decays. *Comput. Softw. Big Sci.* **5**, 27. https://doi.org/10.1007/s41781-021-00075-x (2021).
10. Hassija, V. *et al.* Present landscape of quantum computing. *IET Quantum Commun.* **1**, 42–48. https://doi.org/10.1049/iet-qtc.2020.0027 (2020).
11. Vallury, H. J., Jones, M. A., Hill, C. D. & Hollenberg, L. C. L. Quantum computed moments correction to variational estimates. *Quantum* **4**, 373. https://doi.org/10.22331/q-2020-12-15-373. arXiv:2009.13140 [quant-ph] (2020).
12. Vallury, H. J. *et al.* Noise-robust ground state energy estimates from deep quantum circuits. arXiv:2211.08780 [quant-ph] (2022).
13. Niemann, P., Datta, R. & Wille, R. Logic synthesis for quantum state generation. In *2016 IEEE 46th International Symposium on Multiple-Valued Logic (ISMVL)*, 247–252. https://doi.org/10.1109/ISMVL.2016.30. ISSN: 2378-2226 (2016).
14. Shende, V. V., Bullock, S. S. & Markov, I. L. Synthesis of quantum logic circuits. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **25**, 1000–1010. https://doi.org/10.1109/TCAD.2005.855930. arXiv: quant-ph/0406176 (2006).
15. Plesch, M. & Brukner, C. Quantum-state preparation with universal gate decompositions. *Phys. Rev. A* **83**, 032302. https://doi.org/10.1103/PhysRevA.83.032302 (2011).
16. Schlimgen, A. W., Head-Marsden, K., Sager, L. M., Narang, P. & Mazziotti, D. A. Quantum simulation of open quantum systems using a unitary decomposition of operators. *Phys. Rev. Lett.* **127**, 270503. https://doi.org/10.1103/PhysRevLett.127.270503 (2021).
17. Zhang, X.-M., Li, T. & Yuan, X. Quantum state preparation with optimal circuit depth: implementations and applications. *Phys. Rev. Lett.* **129**, 230504. https://doi.org/10.1103/PhysRevLett.129.230504 (2022).
18. Rubinstein, B. Evolving quantum circuits using genetic programming. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, vol. 1, 144–151. https://doi.org/10.1109/CEC.2001.934383 (2001).

19. Abdollahi, A. & Pedram, M. Analysis and synthesis of quantum circuits by using quantum decision diagrams. In *Proceedings of the Design Automation Test in Europe Conference*, vol. 1, 1–6. https://doi.org/10.1109/DATE.2006.244176. ISSN: 1558-1101 (2006).
20. Daskin, A. & Kais, S. Decomposition of unitary matrices for finding quantum circuits: application to molecular hamiltonians. https://doi.org/10.1063/1.3575402. arXiv:1009.5625 (2013).
21. Treinish, M. *et al.* Qiskit/qiskit: Qiskit 0.37.1. https://doi.org/10.5281/ZENODO.2573505 (2022).
22. Nielsen, M. A. & Chuang, I. L. *Quantum Computation and Quantum Information, 10th anniversary edn.* (Cambridge University Press, 2010).
23. Whitley, D. A genetic algorithm tutorial. *Stat. Comput.*https://doi.org/10.1007/BF00175354 *(1994)*.
24. Lahoz-Beltra, R. Quantum genetic algorithms for computer scientists. *Computers* **5**, 24. https://doi.org/10.3390/computers5040024 (2016).
25. Lukac, M. & Perkowski, M. Evolving quantum circuits using genetic algorithm. In *Proceedings 2002 NASA/DoD Conference on Evolvable Hardware*, 177–185. https://doi.org/10.1109/EH.2002.1029883 (2002).
26. Yabuki, T. & Iba, H. Genetic Algorithms for Quantum Circuit Design—Evolving a Simpler Teleportation Circuit (2000).
27. Williams, C. P. (ed.) *Quantum Computing and Quantum Communications: First NASA International Conference, QCQC '98, Palm Springs, California, USA, February 17–20, 1998: selected papers*. No. 1509 in Lecture notes in computer science (Springer, B1999).
28. Miranda, F. T., Balbi, P. P. & Costa, P. C. S. Synthesis of quantum circuits with an island genetic algorithm. arXiv:2106.03115 [physics, physics:quant-ph] (2021).
29. Lu, Z., Shen, P.-X. & Deng, D.-L. Markovian quantum neuroevolution for machine learning. *Phys. Rev. Appl.* **16**, 044039. https://doi.org/10.1103/PhysRevApplied.16.044039. arXiv:2012.15131 [cond-mat, physics:quant-ph] (2021).
30. Kraft, D. *A Software Package for Sequential Quadratic Programming* (Wiss. Berichtswesen d. DFVLR, 1988). Google-Books-ID: 4rKaGwAACAAJ.
31. Rindell, T. *et al.* Generating approximate state preparation circuits for NISQ computers with a genetic algorithm. https://doi.org/10.48550/arXiv.2210.06411 (2022).
32. Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A.G., Parizeau, M. & Gagné, C. DEAP: Evolutionary algorithms made easy. *J. Mach. Learn. Res.* **13**, 2171–2175 (2012).

Acknowledgements

Author contributions

L.C.L.H. and F.M.C. conceived the project. F.M.C. created the computational framework and performed the experimental calculations, with input from all authors. All authors had input in writing the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to L.C.L.H.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.