UDC 004.413 DOI https://doi.org/10.35546/kntu2078-4481.2024.2.18

M. O. VERNIK
Master in Software Engineering
at the Department of Computer Systems Software
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"
ORCID: 0009-0008-6156-1051

COMPARISON OF CLASSICAL AND QUANTUM COMPUTING FOR PARTICLE SWARM OPTIMIZATION

The article explored and delved into the advanced computational strategies of Particle Swarm Optimization (PSO) by contrasting classical and quantum computing paradigms. The advantages of quantum computing lie in its potential to solve computationally complex problems exponentially faster than classical computers. One of the advantages of Particle Swarm Optimization is its ability to find optimal solutions in complex search spaces. The research centers around the performance of PSO algorithms, as a part of the biological swarm optimization algorithms, when applied to a set of single-objective optimization functions, namely the Sphere, Rosenbrock, Booth, and Himmelblau functions. Utilizing a controlled setup of 100 particles, iterating 100 times across various dimensions tailored to each function, our study reveals that quantum Particle Swarm Optimization, implemented via Q# programming language and tested in Azure Quantum Workspace, consistently surpasses classical PSO in precision and convergence to global minima, despite the increased computational demands and error sensitivity inherent to quantum computations. The classical approach facilitated through Python programming language and leveraging deterministic pseudorandom number generators demonstrates robustness and lower computational costs but does not achieve the quantum's level of accuracy. The paper highlights the potential of quantum PSO to achieve superior optimization results in scenarios with smaller datasets and less complex problem spaces, paving the way for future applications where quantum advantages can be fully realized. The analysis goes further to discuss the implications of these findings for the future of optimization in various industries, including logistics, engineering, and finance, where optimization plays a critical role. The potential of quantum Particle Swarm Optimization to achieve superior optimization results in scenarios with smaller datasets and less complex problem spaces is particularly notable. It suggests that quantum computing could soon transform the landscape of computational optimization, providing solutions that are not only quicker but also more accurate.

Key words: *quantum computing, PSO, Q#, optimization, metaheuristic, Python programming language.*

М. О. ВЕРНІК
магістр з інженерії програмного забезпечення
кафедри програмного забезпечення комп'ютерних систем
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
ORCID: 0009-0008-6156-1051

ПОРІВНЯННЯ КЛАСИЧНОГО ТА КВАНТОВОГО ОБЧИСЛЕННЯ ДЛЯ PSO ОПТИМІЗАЦІЇ

У статті досліджено та детально описано передові обчислювальні стратегії оптимізації рою частинок (Particle Swarm Optimization, PSO), порівнюючи їх з класичними та квантовими парадигмами обчислень. Однією з переваг оптимізації рою частинок є її здатність до знаходження оптимальних рішень в складних просторах пошуку. Дослідження зосереджене на перевірці ефективності алгоритмів PSO, як частини біологічних алгоритмів оптимізації рою, застосованих до набору одноцільових функцій оптимізації, а саме функцій Сфера, Розенброка, Бута та Хіммельблау. Використовуючи контрольоване налаштування з 100 частинками, що ітеруються 100 разів по різних вимірах, адаптованих до кожної функції, дослідження показує, що квантовий метод PSO, реалізований за допомогою мови програмування Q# та перевірений в Azure Quantum Workspace, постійно перевершує класичний PSO за точністю та збіжністю до глобальних мінімумів, незважаючи на збільшені обчислювальні витрати та чутливість до помилок, які притаманні квантовим обчисленням. Класичний підхід, реалізований за допомогою мови програмування Python та використання визначених детерміністичних псевдовипадкових генераторів чисел, демонструє стійкість та менші обчислювальні витрати, але не досягає рівня точності квантового підходу. У статті зазначається потенціал квантового методу PSO для досягнення вищих результатів оптимізації в сценаріях з меншими наборами даних та менш складними просторами проблем, відкриваючи шлях для майбутніх застосувань, де переваги квантових обчислень можуть бути повністю реалізовані. Аналіз також розглядає наслідки цих висновків для майбутньої оптимізації у різних галузях, включаючи логістику, інженерію та фінанси, де оптимізація відіграє важливу роль. Особливо важливим є потенціал квантового методу PSO для досягнення вищих результатів оптимізації в сценаріях з меншими наборами даних та менш складними просторами проблем. Це свідчить про те, що квантові обчислення незабаром можуть трансформувати ландшафт обчислювальної оптимізації, надаючи рішення, які не лише швидкі, але й більш точні.

Ключові слова: *квантові обчислення, PSO, Q#, оптимізація, метаевристика, мова програмування Python.*

Introduction. Problem statement

Nowadays the art of development is constantly searching for new computational capabilities since regular silicon processors are becoming limited to complex computational problems, and for that humanity has created supercomputers [1], which will be working for some time until the rising era of quantum computing, or non-deterministic approaches. Classical metaheuristic algorithms are widely used for solving and improving optimization problems [2] where the deterministic optimization methods are taking a significant amount of time due to the "local traps" and their ability to extend the search space. Quantum data processing is based on the rules of quantum mechanics such as superpositioning and entanglement using a basic unit of information called a qubit, which could be 0 or 1, or both due to the superposition and the scalability by adding more qubits increasing exponentially, but the error in quantum computing diverges significantly due to the fundamental differences in how these systems process and store information.

Classical computers might have:

• hardware failures such as overheating, physical damage, or electronic failures can cause bit flips or crashes;

• software errors, and bugs, in programming can lead to crashes, incorrect outputs, or security vulnerabilities;

• external electromagnetic interference or power surges can disrupt operations or damage components;

• data transmission errors during data transfer, bits might be flipped due to noise or signal degradation, typically addressed using error-detection and error-correction codes like parity bits, checksums, or more complex algorithms like CRC (Cyclic Redundancy Check);

Classical errors can often be deterministically detected and corrected using well-established techniques like redundancy, error correction codes, and rigorous testing frameworks. On the other hand errors in quantum computers include:

• decoherence is the loss of quantum coherence wherein the system's quantum states due to unavoidable interactions with the environment (like thermal noise) lose their quantum behavior, essentially becoming classical. This is the primary source of error in quantum systems and severely limits the time over which quantum information is being processed;

• measurement errors in quantum computing can be error-prone, partly because they can be non-deterministic. A qubit in a superposition of states does not always yield the same measurement result;

• qubits are extremely sensitive to quantum noise and interference. This includes electromagnetic waves, temperature fluctuations, and material imperfections, all of which alter the state of a qubit unpredictably.

This article aims to analyze and compare a basic metaheuristic algorithm called PSO in classical and quantum representation for the test functions for single-objective optimization.

Analysis of recent research and related works

Algorithms were developed using Python and Q# technologies utilizing classical and quantum representations of the PSO. Explanation of the particle swarm optimization algorithm (Fig. 1).

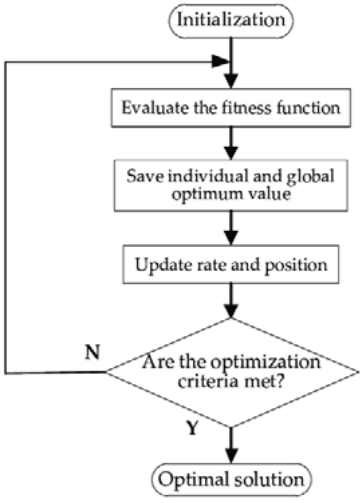

Fig. 1. PSO algorithm [3]

The Particle Swarm Optimization (PSO) algorithm, introduced by Kennedy and Eberhart [4], is a method for finding optimal values of continuous nonlinear functions and is an evolutionary computation technique inspired by natural social behaviors observed in swarms, bird flocking, and fish schooling. In PSO, each solution is conceptualized as a "particle" moving through the search space, where each particle has both a velocity and a position.

This algorithm updates the position and velocity of each particle based on its own experience as well as the experience of its neighbors. Each particle tracks the best position it has found, known as the personal best (p_{best}), and the best position found by the entire swarm is recorded as the global best (g_{best}). Utilizing these values, pbest and gbest, PSO iteratively adjusts the velocities and positions of each particle to search for optimal solutions.

The steps of the algorithm are outlined in detail below.

1. Initialize the swarm

Each particle i is initialized with a random position x_i and a random velocity v_i within the search space. The initial positions and velocities can be randomly generated as follows:

$$xi \sim \text{Uniform}(x\text{min}, x\text{max}), vi \sim \text{Uniform}(v\text{min}, v\text{max}) \quad (1)$$

2. Evaluate fitness

The fitness of each position $\vec{x_i}$ is evaluated based on the objective function $f(\vec{x_i})$. The objective function is problem-specific and determines the quality of each position.

3. Update p_{best}

For each particle, if the fitness of the current position is better than the fitness of its personal best position $\overrightarrow{p_{best,i}}$, then update p_{best} :

$$\overrightarrow{p_{best,i}} = \begin{cases} \vec{x_i} & if\ f(\vec{x_i}) < f(\overrightarrow{p_{best}}) \\ \overrightarrow{p_{best}} & otherwise \end{cases} \quad (2)$$

4. Update g_{best}

If the fitness of the current position of any particle is better than the fitness of the global best position $\overrightarrow{g_{best,i}}$, update g_{best}:

$$\overrightarrow{g_{best,i}} = \begin{cases} \vec{x_i} & if\ f(\vec{x_i}) < f(\overrightarrow{g_{best}}) \\ \overrightarrow{g_{best}} & otherwise \end{cases} \quad (3)$$

5. Adjust velocity and position for each particle

$$\overrightarrow{v_i^{(t+1)}} = w\overrightarrow{v_i^t} + c_1 r_1 \left(\overrightarrow{p_{best,i}} - \overrightarrow{x_i^t}\right) + c_2 r_2 \left(\overrightarrow{g_{best,i}} - \overrightarrow{x_i^t}\right) \quad (4)$$

$$\overrightarrow{x_i^{(t+1)}} = \overrightarrow{x_i^t} + \overrightarrow{v_i^{(t+1)}} \quad (5)$$

where:

w is the inertia weight that controls the influence of the previous velocity;

c_1 | and c_2 | are the cognitive and social coefficients, respectively;

r_1 | and r_2 | are random numbers uniformly distributed in [0,1], providing stochasticity.

6. Repeat

Steps 2–5 are repeated for a predetermined number of iterations or until a convergence criterion is met.

This structured approach enables PSOs to efficiently explore and exploit the search space, leading to the discovery of optimal or near-optimal solutions.

Research data

The selected 4 general testing functions will be executed on the cloud environments using tools Azure Quantum Workspace for the Q# and Jupyter Notebook for the Python implementation.

The main difference in the algorithm implementation will be a randomization function, since in Python will be used a NumPy library random function which works as module which contains two interfaces of pseudorandom number generators (PRNGs):

1. random uses the Mersenne Twister PRNG [5], which is not cryptographically secure;

2. SystemRandom uses either the /dev/urandom file on POSIX systems [6] or the CryptGenRandom() function on Windows NT systems, both are Cryptographically secure PRNGs [7].

On the other hand the Q# random implementation requires the use of qubits with applied Hadamard gate (H) to transform the basis states |0⟩ and |1⟩ into superpositions and forcing a qubit to “choose” one of its basis states.

```
operation QuantumRandomDouble(min : Double, max : Double, nQBits : Int) : Double {
use qubits = Qubit[nQBits];
ApplyToEach(H, qubits);
let results = MResetEachZ(qubits);
let power = IntAsDouble(1 <<< nQBits);
let decimal = IntAsDouble(ResultArrayAsInt(results)) / power;
return min + (max - min) * decimal;
}
```

Steps in depth:

1. Qubit Allocation

```
use qubits = Qubit[nBits];
```

Quantum Principle: in quantum computing, data is represented by qubits instead of classical bits. A qubit is a quantum system that can exist in a superposition of 0 and 1 states, unlike classical bits, which are definitively 0 or 1.

Operation: this line allocates nQBits qubits. Initially, each qubit is in the state |0⟩.

2. Applying the Hadamard Gate

ApplyToEach(H, qubits);

Quantum Principle: the Hadamard gate (H) is a fundamental quantum gate that transforms the basis states |0⟩ and |1⟩ into superpositions. Specifically, it maps |0⟩ to $\frac{(|0\rangle + |1\rangle)}{\sqrt{2}}$ and |1⟩ to $\frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$.

Operation: this line applies the Hadamard gate to each qubit. The result is that each qubit is put into a superposition of |0⟩ and |1⟩, effectively randomizing its state. This means each qubit now has a 50% probability of being measured as 0 and a 50% probability of being measured as 1.

3. Measurement and Reset

let results = MResetEachZ(qubits);

Quantum Principle: measurement in quantum mechanics forces a qubit to "choose" one of its basis states. According to the Copenhagen interpretation [8], the act of measuring a quantum state causes its wave function to collapse to one of the own states of the observable being measured.

Operation: this function measures each qubit on the computational (Z) basis, where the superposition state collapses to either |0⟩ or |1⟩ based on the probability amplitude of each state. The *MResetEachZ* operation also resets the qubits to |0⟩ after measurement, making them ready for reuse. The results are an array of classical bits reflecting the outcome of each quantum measurement.

Research results

In this study, the computational experiments were designed to evaluate the performance of both classical and quantum Particle Swarm Optimization (PSO) algorithms across a range of benchmark functions. The experiments utilized a consistent setup involving 100 particles, which iterated 100 times to optimize the given objective functions. The dimensionality of the search space varied depending on the function being tested: for the Sphere and Rosenbrock functions, the algorithms operated within a five-dimensional space. This higher dimensionality allowed for a comprehensive assessment of the algorithms' capabilities in handling complex, multi-dimensional landscapes, which are typical in many real-world optimization problems. Conversely, for the Booth and Himmelblau functions, the experiments were conducted in a two-dimensional space. This setup was chosen because these functions are naturally defined in two dimensions and are commonly used to benchmark optimization algorithms' performance in a more visually interpretable manner. This dimension-specific approach ensured that each function was tested under conditions that best reflected its typical use cases, thereby providing insights that are both relevant and applicable to typical scenarios encountered in optimization tasks.

Table 1

Test functions and results

Name	Function	Classical	Quantum	Global minimum
Sphere function	$f(x) = \sum_{i=1}^{n} x^2$	$1.012 * 10^{-16}$	$1.552 * 10^{-17}$	$f(0,\dots,0) = 0$
Rosenbrock function	$f(x) - \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$	$2.274 * 10^{-1}$	$4.781 * 10^{-2}$	$f(1,\dots,1) = 0$
Booth function	$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$	$1.077 * 10^{-20}$	$2.893 * 10^{-24}$	$f(1,3) = 0$
Himmelblau`s function	$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$	$2.551 * 10^{-20}$	$3.076 * 10^{-24}$	$f(3,2) = 0 f(-2.805118, 3.131312) = 0$ $f(-3.779310, -3.283186) = 0$ $f(3.584428, -1.848126) = 0$

The comparative analysis of classical and quantum PSO implementations on single-objective functions reveals distinct performance characteristics, particularly when evaluated with a small sample size. In scenarios involving 100 particles, 100 iterations, and 5 dimensions, the quantum approach consistently outperforms its classical counterpart across various benchmark functions.

Performance Summary:

Sphere Function: Quantum PSO achieved a significantly closer approximation to the global minimum, registering a result of $1.552 * 10^{-17}$, compared to $1.012 * 10^{-16}$ by the classical PSO.

Rosenbrock Function: Here too, quantum PSO showed superior performance with $4.781 * 10^{-2}$, versus $2.274 * 10^{-1}$ for the classical method.

Booth Function: Quantum optimization delivered an exceptionally precise result of $2.893 * 10^{-24}$, substantially better than the $1.077* 10^{-20}$ by classical PSO.

Himmelblau's Function: Quantum PSO achieved $3.076 * 10^{-24}$, improving upon the classical PSO's $2.551 * 10^{-20}$.

Conclusions and further work

While the quantum approach demonstrates enhanced efficacy in locating optimal or near-optimal solutions for small sample sizes, it faces scalability challenges. With larger samples, quantum PSO requires exponentially more time and incurs higher computational costs. In contrast, the classical PSO, though sometimes less accurate, maintains a lower computational cost and exhibits a reduced likelihood of error, making it more scalable and reliable for larger or more complex datasets.

These findings suggest that quantum PSO is particularly advantageous for small-scale or precision-critical optimization tasks where the quality of the solution is paramount and resources are sufficient to support higher computational demands. For broader or more resource-constrained applications, classical PSO remains a viable and effective option.

References

1. Ingrid Y. Bucher (1983). The computational speed of supercomputers. In Proceedings of the 1983 ACM SIGMETRICS conference on Measurement and modeling of computer systems (SIGMETRICS '83). Association for Computing Machinery, New York, NY, USA, 151–165. https://doi.org/10.1145/800040.801403.

2. Torres-Jimenez, Jose & Pavón, Juan (2014). Applications of metaheuristics in real-life problems. Progress in Artificial Intelligence. 2. 175-176. 10.1007/s13748-014-0051-8.

3. Xiao, Yunqi & Wang, Yi & Sun, Yanping (2018). Reactive Power Optimal Control of a Wind Farm for Minimizing Collector System Losses. Energies. 11. 3177. 10.3390/en11113177.

4. Kennedy J., Eberhart R. (1995). Particle Swarm Optimization. Proceedings of IEEE International Conference on Neural Networks. Vol. IV. pp. 1942–1948. doi:10.1109/ICNN.1995.488968.

5. Matsumoto M., Nishimura T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation. 8 (1): 3–30. CiteSeerX 10.1.1.215.1141. doi:10.1145/272991.272995. S2CID 3332028.

6. P1003.1 - Standard for Information Technology Portable Operating System Interface (POSIX(TM) Base Specifications, Issue 8. IEEE Standards Association. https://standards.ieee.org/ieee/1003.1/7700/

7. Huang Andrew (2003). Hacking the Xbox: An Introduction to Reverse Engineering. No Starch Press Series. No Starch Press. p. 111. ISBN 9781593270292.

8. Pearle P., Valentini A. (2006). Quantum Mechanics: Generalizations, Editor(s): Jean-Pierre Françoise, Gregory L. Naber, Tsou Sheung Tsun, Encyclopedia of Mathematical Physics, Academic Press. Pages 265-276, ISBN 9780125126663, https://doi.org/10.1016/B0-12-512666-2/00415-6.