Quantum Strategy of Population Initialization in Genetic Algorithm

Jun Suk Kim
School of Electrical Engineering and Computer Science
Gwangju Institute of Science and Technology
Gwangju, Republic of Korea
junsuk89@gmail.com

Chang Wook Ahn
AI Graduate School
Gwangju Institute of Science and Technology
Gwangju, Republic of Korea
cwan@gist.ac.kr

ABSTRACT

Quantum Genetic Algorithm is a relatively new field of study to enhance the computational efficiency of the Darwinian optimization process in genetic algorithms with quantum speedup techniques. This paper introduces an application strategy of the quantum counting algorithm to genetic algorithms, particularly aimed to enhance the initial population setup at the beginning of optimization. More specifically, our goal is to exploit a quantum algorithm to count the number of marked items from an unstructured list quadratically faster than classical algorithms in order to detect the presence and amount of unsuitable individuals in a stochastically generated initial population, thereby starting optimization with a mark of potential to improve the performance in the later stage. The advantage of our method is examined via a conventional genetic algorithm to solve the 0-1 Knapsack problem with varying cases of the constraints, and a comparative analysis on the optimizing performance is made accordingly.

CCS CONCEPTS

• **Theory of computation → Discrete optimization**; **Evolutionary algorithms**.

KEYWORDS

quantum computing, genetic algorithm, discrete optimization

ACM Reference Format:
Jun Suk Kim and Chang Wook Ahn. 2022. Quantum Strategy of Population Initialization in Genetic Algorithm. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3520304.3529010

1 INTRODUCTION

Quantum computing's main features include quantum superposition and entanglement, which enable parallel computation and thus the anticipated speedup over *classical* computers. [8] It is therefore not surprising to see that quantum optimization is currently one of the most highlighted fields in quantum computing [9]. Similarly, efforts to solve quantum optimization problems with evolutionary processes via Quantum Genetic Algorithm (QGA) have also been made for nearly two decades.

The main motivation of studies on QGA is to fully exploit the features of quantum computation to boost the effectiveness of the GA's heuristic optimization process. So far, they have mostly focused on utilizing Grover's search algorithm [3], which has been both theoretically and experimentally proven to accomplish a given task quadratically faster than its classical counterparts [3]. The QGA approach we introduce in this paper instead uses a quantum counting algorithm (QCA) to exploit its relatively fast item-counting capability to start optimization with more prospective individuals. Through it, the initial population could benefit from possessing higher fitness values that would have lasting effects for the rest of the process.

2 QUANTUM POPULATION INITIALIZATION

One of the crucial topics in GA is the initial population setup, i.e. properly constituting a population set to begin optimization with [7]. Despite the stochastic nature of the population setup in GA, apparently little differences in the initial population could often turn to notable distinctions in optimizing performance at the end of the task [6]. For the case of exploiting a GA to solve a discrete constrained optimization problem, we can safely assume that raising the bar of the constraint would rapidly reduce the number of individuals capable of surviving the first trial of selection. Optimization problems with higher constraints are thus more punishing with regards to generating a desirable initial population.

We now define p, the number of individuals in the population to satisfy the constraint. The amount of p in the initial population has a lasting effect on the optimization convergence in the later stages of GA applied to a constrained optimization problem. If a desired degree of p can be secured in the initialization step, we can technically skip the first few generations that would have otherwise been necessitated to achieve the same degree of p. Furthermore, since genetic algorithms are indeterministic [2] and thus do not guarantee a exact moment of achieving a certain p, starting the first generation with a desired degree of p can provide with considerable probability a better trace of fitness convergence. Fitness evaluation is a main factor that increases the overall computational cost of GA, and performing the above process classically could turn the whole algorithm even more ineffective, since the number of fitness evaluations skipped by securing a desired degree of p would be compensated by the number of fitness evaluations.

The main idea that our method proposes is to exploit QCA to count the number of the satisfactory individuals p in the initial population at the beginning of GA, thereby securing the p individuals

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9268-6/22/07.
https://doi.org/10.1145/3520304.3529010

within a computational complexity limit that would not harm the overall performance of the algorithm. After generating the quantum population, we proceed to evaluate the fitness of individuals for a given constraint optimization problem, or more specifically a 0-1 Knapsack problem in our context,

$$\begin{aligned} \text{maximize } f(\mathbf{x}) &= \sum_{i=1}^{n} x_i w_i \\ \text{subject to } &\sum_{i=1}^{n} x_i w_i \leq C \end{aligned} \tag{1}$$

where x_i and w_i are the i^{th} binary gene of the individual $\mathbf{x}$ and its corresponding item weight, with a capacity constraint C. The evaluation on each individual is done based on the metric

$$f(\mathbf{x}) = \begin{cases} \sum_{i=1}^{n} x_i w_i, & \text{if } \sum_{i=1}^{n} x_i w_i \leq C, \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

so that every individual that does not satisfy the constraint condition is guaranteed to be abandoned during the succeeding evolution. (2) implies that it is feasible to distinguish suitable and unsuitable individuals in a Boolean manner. This fact is especially important with regards to QCA, since marking the desired states in QCA is led by effectively mapping them in the quantum oracle, which can be structurally simpler and thus more cost-effective if the input question calls for Boolean answers [1].

The target problem function for QCA is encoded in its Grover's oracle, where it is reconstructed with quantum gates to return the same evaluation output as the classical circuit. We can setup the oracle O as follows to determine whether each individual satisfies the constraint condition or not,

$$O|\mathbf{x}\rangle = \begin{cases} -|\mathbf{x}\rangle, & \text{if } \sum_{i=1}^{n} x_i w_i \leq C, \\ +|\mathbf{x}\rangle, & \text{otherwise} \end{cases} \tag{3}$$

for each quantum individual state $|\mathbf{x}\rangle$. The oracle can return the result for all the superposed quantum states simultaneously, regardless of their number. Single QCA implementation requires $\Theta(\sqrt{N})$ oracle calls to properly count the number of the desired items, marked with negative signs in the oracle, out of N items [5]. This complexity indicates that the counting procedure with QCA can be done quadratically faster than the classical setup.

We can set p as a hyperparameter to determine whether, once the counting is finished, the certain number of individuals in the initial population satisfies the constraint condition to proceed beyond the initialization step. If not, then the generated population is disposed, and the initializing and counting procedure is performed repetitively until the preset number of p is met. Undesirably low p would fail to distinguish the advantage of our method from the conventional process in terms of performance, while undesirably high p would require an excessive number of QCA iterations especially when the constraint is severe.

In Algorithm 1, the procedure of our proposed method to exploit QCA in GA is written as a pseudocode. Under a GA designed to contain a population of n individuals and to optimize its given task via g generations, the total number of fitness evaluations done throughout the whole procedure, counting the evaluation over m genes in each individual as single run, can be represented as $O(ng)$. Our goal then is to design a quantum population initializer that would keep the said computational complexity polynomially equal, i.e. $O(tng)$ for a fixed integer $t \geq 0$. Suppose that we want a strict upper bound for the complexity, $O(2ng)$, allowing us to spend $O(ng)$ for the population initialization alone. We assume an ideal quantum scenario, where our quantum initializer would return exact results without errors caused by the physical deficiencies of a noisy quantum machine, such as quantum decoherence [8].

Algorithm 1 Quantum Population Initialization

Input
  f target problem function
  n size of population
  m size of individual
  p threshold of satisfactory individuals
  g number of total generations
  ϵ hyperparameter to control degree of error
  k hyperparameter to control counting iteration
Output
  pop population w/ p or more satisfactory individuals
$iter \leftarrow kg\sqrt{n}$
Construct quantum circuit QCA w/ embedded f and chosen ϵ
for $h = 1, 2, \ldots, iter$ **do**
  $n_{sat} \leftarrow 0$ ▷ number of satisfactory individuals
  $pop \leftarrow O_{n \times m}$ ▷ classical binary population
  Randomly generate individuals to fill pop
  Encode pop into superposed quantum states $|pop\rangle$
  Load $|pop\rangle$ onto QCA to implement quantum counting
  Measure qubits on QCA to count n_{sat}
  if $n_{sat} \geq p$ **then**
    break
  end if
end for
return pop

A fully functional quantum counting algorithm will return the desired result within $O(\sqrt{n})$ queries from the initial population of n individuals, leaving us with up to $O(\sqrt{ng})$ iterations available to meet the upper bound standard. In other words, if we want a strict upper bound, i.e. a choice that will spoil the algorithm's overall performance the least, we will iterate the quantum initialization procedure up to $\sqrt{ng}$ times and stop, even if the desired p hasn't been achieved. If one wants a more lenient upper bound, it is allowable to set the repetition limit to $k\sqrt{ng}$, where k is a constant integer to be manually chosen. The accuracy of the counting result depends on the number of qubits used to construct the QCA circuit. More precisely, the degree of error $|\Delta M|$ in the final estimate of QCA is

$$|\Delta M| < (\sqrt{2MN} + \frac{N}{2^{t+1}})2^{-t} \tag{4}$$

where N is the number of items in total, M is the number of the desired items, and $t = m - \lceil \log(2 + \frac{\epsilon}{2}) \rceil$ with ϵ as our choice [5]. In other words, one would like to add more qubits to the QCA circuit if higher accuracy of the result is desired.

3 EXPERIMENT

The objective of the experiment was to compare the optimization performances of a plain, original GA and a GA loaded with the quantum population initializer (QPI). We choose a 0-1 Knapsack problem with varying degrees of the capacity constraint to observe the changes in the proposed method's performance with respect to different constraining conditions. The 200-sized dataset from Kaggle knapsack 2020 competition [4] was loaded to set up the problem for the experiment. Within it, 200 items have the total value of 142318 and the total weight of 99618. We set five scenarios for different capacity limits, which are $C_1 = 35000, C_2 = 38000, C_3 = 39000, C_4 = 39500$, and $C_5 = 40000$. Each comparison run was iterated 10 times to evaluate the average of the difference, or the gap, in their performance and the number of the Grover oracle iterations called for QPI. A simple genetic algorithm with tournament selection was used as the basis with its hyperparameters and the corresponding values listed in Table 1. We set a strict upper bound for the number of QPI iterations, which is $1100 \approx 1\sqrt{ng} = 1131.37$. Under this setting, we can safely claim that the total time complexity of fitness evaluations would not exceed $O(2ng) = O(ng)$.

Table 1: Hyperparameter values for GA

Hyperparameter	Value
size of population	128
size of individual	200
number of generations	100
number of iterations	10
crossover probability	0.9
mutation probability	0.005

We manually set comparatively low values of p, i.e. $p = 1, 2$, and 4 to observe how the performance changes accordingly. Overall, we prepared 15 scenarios with different constraint capacities C in 35000, 38000, 39000, 39500, and 40000, and the satisfaction thresholds p in 1, 2, and 4. Observing the differences in the convergence tendency among these scenarios would allow to acknowledge and trace the impact of the corresponding constraints and satisfaction conditions upon the performance of the proposed method.

Figure 1 lists the plots of the optimization results for the 0-1 Knapsack problem, in the 15 scenarios with varied combinations of C and p. They graphically show the average fitness values from the satisfactory individuals, excluding the unsatisfactory ones, over 100 generations, and each plot is an average result of 10 iterations under the identical conditions. We also organized the results into a statistical chart of the average gap of the fitness values between the two GAs over the series of generations and the number of iterations performed for QPI in each scenario, as shown in Table 2. The numbers shown are the mean and the corresponding sample standard deviation evaluated from the iterative runs. Negative gaps indicate that the plain GA has achieved higher fitness values on average in the corresponding scenarios.

Table 2: Average fitness gap over generations and number of QPI iterations per scenario

Scenario (C/p)	Avg. Fitness Gap	QPI Iteration
35000/1	41295.98 ± 2742.11	60.90 ± 55.88
35000/2	2383.62 ± 6609.35	997.50 ± 324.13
35000/4	-7623.97 ± 11603.66	1100.00 ± 0.00
38000/1	8672.41 ± 5701.13	4.70 ± 3.30
38000/2	10994.14 ± 7267.72	49.70 ± 45.45
38000/4	-1276.50 ± 3285.62	1100.00 ± 0.00
39000/1	5122.43 ± 5245.89	1.10 ± 1.20
39000/2	5958.11 ± 396.83	11.50 ± 10.95
39000/4	2593.67 ± 3592.98	446.60 ± 307.21
39500/1	845.64 ± 2880.91	0.80 ± 1.03
39500/2	868.85 ± 2731.67	3.20 ± 4.89
39500/4	6860.41 ± 1521.55	202.00 ± 157.37
40000/1	358.47 ± 2425.82	1.00 ± 1.05
40000/2	488.43 ± 2561.34	2.20 ± 2.25
40000/4	836.49 ± 3137.56	45.50 ± 42.02

4 DISCUSSION

The performance tendency of the method with varying C and p in the actual experiment appears to support our expectation that, with a considerably high degree of constraint, even a small number of p would create a noticeable performance gap between the two competitors. Note that the scenario 35000/1 achieves the greatest gap in the fitness values throughout the generations, with the mean of 41295.98, which hardly narrows until the end. Although not as dramatic, the scenarios 38000/1, 38000/2, and 39000/2 also show considerable differences in the fitness that last throughout the whole generations. On the other hand, the scenario 35000/4 shows that the QPI-loaded GA underperforms the original after the 32nd generation.

As for the scenarios with $C = 40000$, the overall differences in the fitness are virtually negligible. From the extremely low numbers of QPI iterations for all the three cases, we infer that the constraint limit $C = 40000$ is rather a generous condition for the given dataset, under which both algorithms could easily find multiple copies of the satisfactory individuals. It is also worth mentioning that the scenario 39500/1 and 39500/2 share the similar outcome, perhaps for the same reason. Nonetheless, the scenario 39500/4 achieved a notable difference, probably because securing 4 satisfactory initial individuals was adequate to assist the early stage of optimization under the constraint with comparatively low generosity.

In the last section, we stated that the "zero" individuals that fail to satisfy the constraint condition are omitted in evaluating the average fitness, and therefore their existence is not reflected in the results shown in Figure 1 and Table 2. In other words, the fitness traces in Figure 1 represent the average fitness values of the satisfactory individuals only. Had the unsatisfactory individuals been involved, all the plots of the QPI-loaded GA would have shown a substantial increase at the beginning, since they initially possess only a few suitable individuals in the population. We decided to omit the unsatisfactory individuals in order to more clearly show the effect of QPI upon the optimization performance.

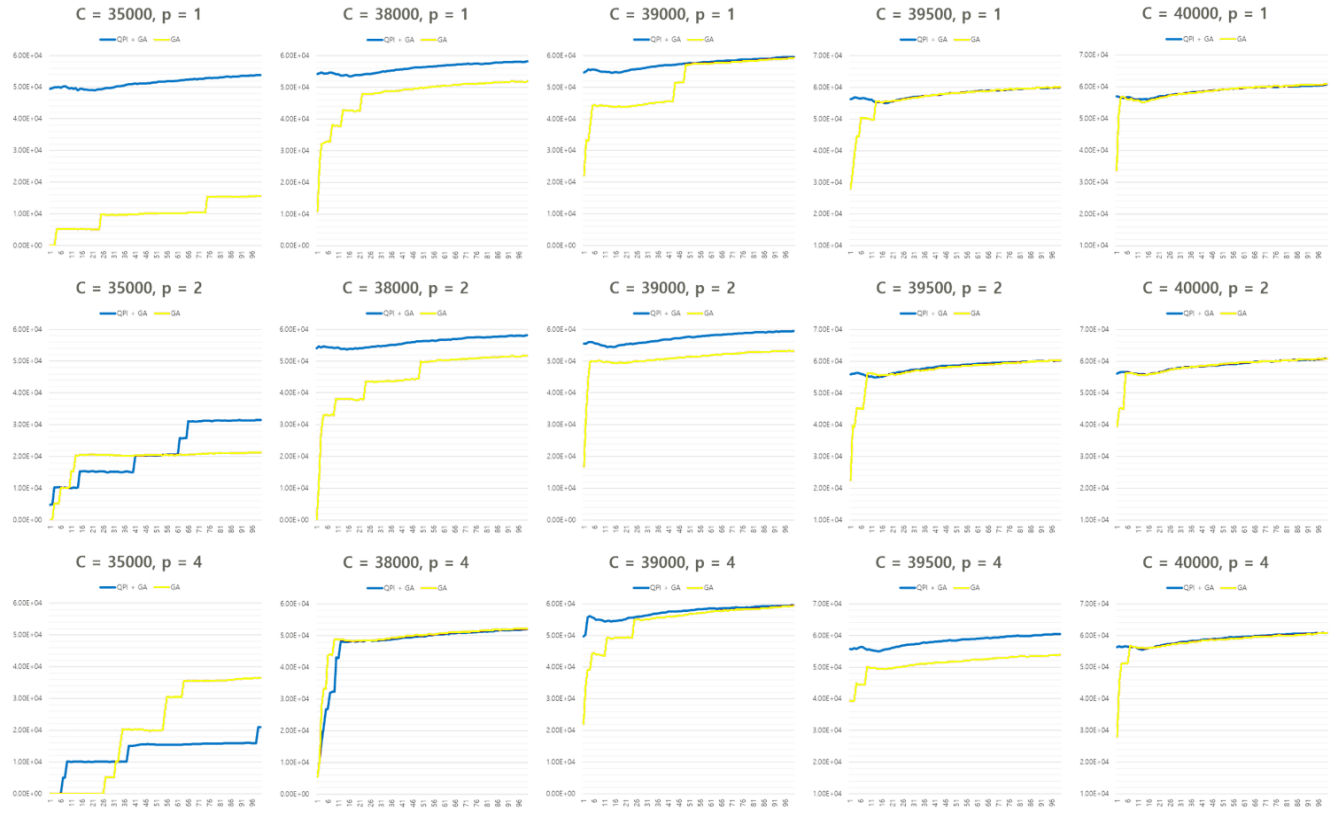

Figure 1: Optimizing performances of a GA with QPI (blue) and a normal GA (yellow) for the 0-1 Knapsack problem in 15 scenarios of different C and p. The x-axis contains the generation counts from 1 to 100, and the y-axis represents the fitness values averaging over 10 iterations. Every plot is drawn to the same scale in terms of the y-axis, which is in a range of 60000.

The running time, in terms of the number of evaluations, of the GA with the proposed method was $(1100 \times \sqrt{n}) + ng \approx 25300$, as opposed to the plain GA that took $ng = 12800$. While we admit such deficiency, the main objective of our study in this paper is to apply the quantum enhancement to the excessively resource-consuming computation procedures, thereby surveying the feasibility of prospective approaches that have been considered undesirable in classical computing and bringing them to the region of actual practicality. Due to the heavy loads of computation required for the real-world problems, we expect that there are diverse aspects in GA or any other heuristic optimization methods that the quadratic or exponential speedup of quantum application could be beneficial.

5 CONCLUSION

The experiment results demonstrate the effectiveness of repetitively counting the number of suitable individuals at the initial population setup as to the matter of optimization, especially under harsh constraint environments in GA. Our proposed strategy is to accelerate the procedure by means of quantum computation, and although there are a number of concerns to address, including properly selecting the values of p with respect to various constraints, we expect that future studies can be conducted to resolve such matters.

REFERENCES

[1] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. 1998. Tight Bounds on Quantum Searching. *Fortschritte der Physik* 46, 4-5 (Jun 1998), 493–505. https://doi.org/10.1002/(sici)1521-3978(199806)46:4/5<493::aid-prop493>3.0.co;2-p

[2] David E. Goldberg. 1988. *Genetic Algorithms in Search, Optimization and Machine Learning* (13 ed.). Addison-Wesley Professional.

[3] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

[4] Kaggle. 2020. *knapsack 2020 | Kaggle.* https://www.kaggle.com/c/knapsack-2020/submissions/final.json?sortBy=date&group=all

[5] Michael A. Nielsen and Isaac L. Chuang. 2004. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (1 ed.). Cambridge University Press.

[6] Weifeng Pan, Kangshun Li, Muchou Wang, Jing Wang, and Bo Jiang. 2014. Adaptive Randomness: A New Population Initialization Method. *Mathematical Problems in Engineering* 2014 (2014).

[7] Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M.A. Salama. 2007. A novel population initialization method for accelerating evolutionary algorithms. *Computers Mathematics with Applications* 53, 10 (2007), 1605–1614. https://doi.org/10.1016/j.camwa.2006.07.013

[8] Noson S. Yanofsky and Mirco A. Mannucci. 2008. *Quantum Computing for Computer Scientists* (1 ed.). Cambridge University Press.

[9] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D. Lukin. 2020. Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices. *Phys. Rev. X* 10 (Jun 2020), 021067. Issue 2. https://doi.org/10.1103/PhysRevX.10.021067