Grover Binary Search for Discrete Quantum Optimization

Ayaz Baykov
Innopolis University
Innopolis, Russia
a.baykov@innopolis.university

Stanislav Protasov
Innopolis University
Innopolis, Russia
s.protasov@innopolis.university

***Abstract*—Contemporary quantum algorithms, being efficient theoretically, fail to run on real QPUs. One reason for these failures is the algorithm's probabilistic nature, which is amplified by imperfections in hardware implementation. Grover search is such a probabilistic method, which enables other methods in quantum optimization and machine learning. In this work we improve the theoretical worst case complexity of Grover Adaptive Search by replacing iterations with binary search. We observe in the experiments that our method shows better success rate in general, and is sufficiently better for a specific type of optimization landscapes with plateaus.**

***Index Terms*—Grover search, quantum computing, quantum optimization, knapsack problem**

I. Introduction

Solving optimization problems is one of the most promising areas of quantum computing applications. Typically, we restate a well-known NP-hard discrete optimization problem using completely different mathematics. This trick allows to make use of operations, which are proven to be faster when run on a quantum processing unit (QPU). Such methods are Shor's algorithm [1], which converts a factorization problem into a frequency estimation task; quantum version of discrete Fourier transform [2], which instead of a straightforward "algorithmic" implementation utilizes unitary property of transformation operator. And, of course, the Grover search [3], which instead of doing naive iterations, benefits from parallel predicate execution for the argument superposition. Grover's algorithm, as well as Shor's algorithm, is not just an algorithm. This is a framework that requires a quantum oracle implementation. These methods can be used for solving various problems, if these problems provide a corresponding oracle. Here we concentrate on the application of Grover search to discrete optimization, and we illustrate our findings with a simplified knapsack packing problem.

II. Previous work

In this paper, we present an improvement for Grover Adaptive Search (GAS) algorithm [4]. The original algorithm is an iterative hybrid general-purpose framework (utilizing both CPU and QPU computations) to solve discrete optimization problems. This framework supposes we can convert a problem statement into the following conditions:

$$S_{low}(x) = \{low < cost(x)\} \wedge valid(x), \tag{1}$$

where x is a candidate solution among N possible (hereinafter we use N to denote a problem search space cardinality), $cost$ is a function to maximize, and $valid$ predicate is responsible for problem constraints. For example, in the packing of knapsacks, given a set A, to choose an optimal subset:

$$\begin{aligned} &x \subseteq A, \\ &N = 2^{|A|}, \\ &valid(x) = weight(x) \leq limit \end{aligned} \tag{2}$$

The predicate $S_{low}(x)$ is then encoded as a *quantum oracle* – special form of a quantum transformation: $(x, t) \rightarrow (x, t \oplus sf(x))$ which satisfies the restriction for any quantum operation to be invertible. It is used inside Grover search algorithm, which solves the problem of satisfiability of $S_{low}(x)$ by providing one of the solutions. If there exists x satisfying $S_{low}(x)$, $cost(x)$ value is used to update low boundary until convergence. Let us define the cost sol of the unknown optimal solution. In the worst case, the lower bound improves by the constant each iteration, thus the classical (CPU) part of the method will trigger the QPU method $\mathcal{O}(sol - low)$ times. In section III we show, how to improve worst case complexity estimation to $\mathcal{O}(\log(\tilde{sol} - low))$. Here $\tilde{sol}$ means the solution estimation from above. We support our theoretical findings with practical observations in section IV. In the discussion section V, we show the cases where this improvement is practically important.

III. Methodology

We call our optimization method Grover Binary Search (GBS). The method relies on the *binary search* algorithm, which for k sorted objects guarantees $\mathcal{O}(\log k)$ complexity on the CPU. It also has a quantum implementation [5] with the same complexity estimation. Here, we propose to use this algorithm instead of the iterative search proposed in the original paper.

If we explicitly reproduce the original idea and replace the GAS iteration with a binary search, we can use a function $count(S_{low}(x))$. The function must return the number of solutions satisfying the oracle. This function is non-increasing for the variable low, so it can be used in binary search. Such counting function can be implemented with quantum counting method [6], which is just an application of Quantum

Phase Estimation [7]. Unfortunately, this method requires doubled number of qubits, $\mathcal{O}(log(N))$ runs of **controlled** Grover iteration, plus $QFT^\dagger$ [2] (which uses order of $2\log N$ gates). Also, this method provides an *approximation* of *count* value, instead of exact answers.

To overcome these complications, we propose replacing the nondecreasing *count* with another nondecreasing function:

$$any_{low}(x) \overset{def}{=} count(S_{low}(x)) > 0. \quad (3)$$

In our method, we search for the smallest low, for which the number of solutions is 0: $\underset{\text{low}}{\operatorname{argmin}}[count(S_{low}(x)) = 0]$. For this condition, both functions $count$ and any return the same answer. To implement the function any, we follow the standard Grover search procedure for unknown number of answers, which requires $\mathcal{O}(\log N)$ **non-controlled** Grover iterations. Any Grover search answer x, which satisfies $S_{low}(x)$, means $any_{low}(x) = True$.

Altogether, our method in the worst case will run $\mathcal{O}(\log_2(\tilde{sol} - low))$ (better than is GAS) iterations of binary search, each of which will run $\mathcal{O}(\log(N))$ Grover iterations (same as in GAS). The pseudocode of our method is shown in the listing 1.

Algorithm 1: Binary Grover search for optimization problem

```
Input: $P$ – problem search space
Result: $R$ – solution
$low \leftarrow Greedy1(P)$;
$high \leftarrow Greedy2(P)$; // here we greedily
    estimate maximum cost(x)
$R \leftarrow \emptyset$;
$\lambda \leftarrow \frac{8}{7}$;
do
    $M \leftarrow 1$;
    while $M \leq \sqrt{|P|}$ do
        $i \leftarrow$ pick uniformly from $\{1...M\}$;
        $mid = \lfloor \frac{low+high}{2} \rfloor$;
        $x \leftarrow GroverSearch^i(P, mid)$;// run $i$
            Grover iterations to find
            $x \in P$
        if $valid(x)$ and $cost(x) > low$ then
            $low \leftarrow cost(x)$;
            $R \leftarrow x$;
        else
            $M \leftarrow M * \lambda$;
        end
    end
    $high \leftarrow mid$;
while $high - low > 1$;
return $R$;
```

IV. Implementation and analysis

We test our solution on a simplified knapsack packing problem: we *assign cost equal to weight* for the sake of a simpler input and circuit. This allows us to run simulator tests for a larger search space while keeping the problem exponentially hard. In a classical 0/1 problem statement, we would need an additional quantum register for weights, limiting our simulation opportunities.

In section I we mentioned, that Grover search is a generic framework, thus it requires additional implementations. First, we need to implement a quantum oracle for the problem. In the case of a simplified knapsack, our oracle should implement the following condition:

$$S_{low}(x) = \{low < \sum_{item \in x} cost(item) \leq limit\}. \quad (4)$$

Note that in a classical 0/1 knapsack packing, the oracle should check $\sum_{item \in x} weight(item) \leq limit$ instead. In our implementation we avoid this by setting $cost = weight$.

We do our implementation and tests using IBM's Qiskit framework v0.31.0 [8], running on x64 Ubuntu Linux. We use the basic noiseless probabilistic simulator `BasicAer`. In this work, we do not consider noise and concentrate on probabilistic effects only.

Circuit 1 represents an example of oracle implementation. We use controlled QFT adders [9] to implement the sum of weights. Uncontrolled adders, together with $CNOT$ applied to a data sign qubit, are used to implement inequalities. This technique was described in chapter 5 of the book [10]. $Uncomputing$ is a block of inverse adders to bring $data$ and key qubits to the initial state. We add $+1$ to lower bound to replace strict inequality $low < cost$ with non-strict $0 \leq cost-(low+1)$, and $+1$ to upper bound to convert $cost \leq limit$ into $cost-(limit+1) < 0$. We avoid unnecessary manipulations by encoding the limit as $low + (limit - low) = 5 + 2$.

We run a simulator experiment to verify the validity of our implementations. Let the available items be $\{6, 3, 5, 4, 2, 3\}$, with an estimated low cost equal to 12, and $limit = 16$. Now, in case of GAS, the oracle would check for answers above 12 and the result of the Grover search would be as in Figure 2a. Most likely it will get answer 14 and repeat the procedure 2b. At the same time, GBS would immediately build an oracle to search for answers above 14. This shows how in binary approach we can reach better values with less iterations for the cases where GAS was not lucky enough.

The main experiment is planned as follows. We test GAS against GBS on a random sets with 4..11 items (which is equivalent to 16..2048 problem search space). Each item's weight is uniformly sampled from the 1..63 integer range. The knapsack limit is uniformly chosen from $[\frac{\sum items}{4}, \sum items]$ integer range. We ran 32 random experiments for each size of the problem. According to the proposed algorithm 1, for the estimation of low, the algorithm $Greedy1$ takes the sum of the first elements that does not exceed $limit$, while $Greedy2$ for the upper bound $high$ takes the value $limit$. Among the values which differ between the methods, we measure number of times QPU circuit is launched, and the correctness of given answers.

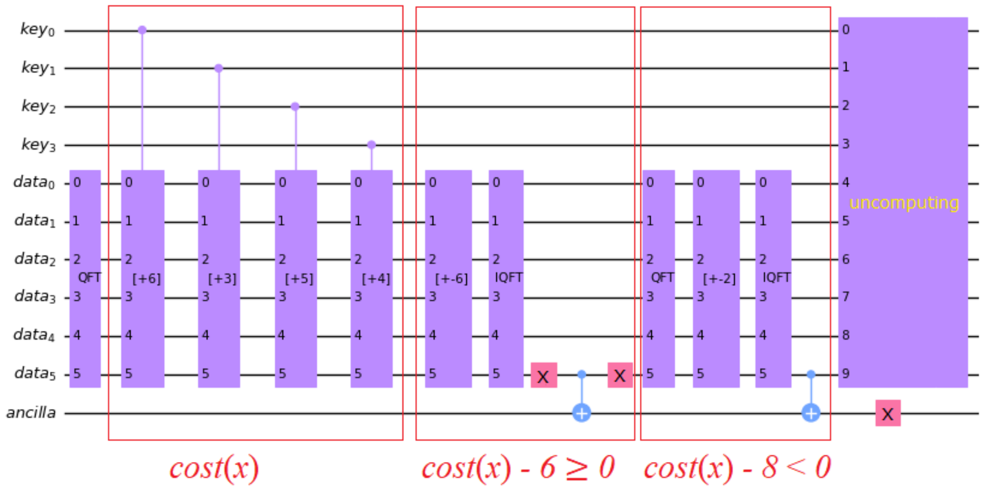
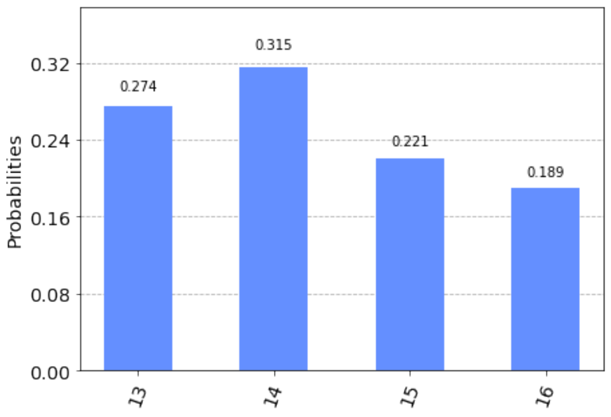
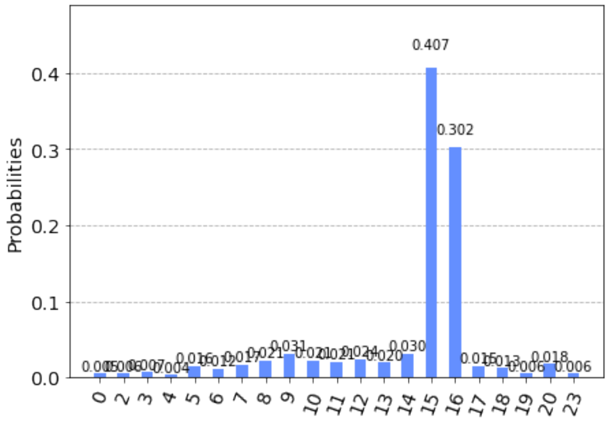


Fig. 1: Example oracle implementation for $S_5(x)$ condition: $5 + 1 \leq cost(x) < (5 + 2) + 1$. Controlled addition operations implement $cost(x)$ function (costs of knapsack candidates), while uncontrolled addition set inequality boundaries.



(a) Results of Grover search for oracle checking $12 < cost(x) <= 16$



(b) Results of Grover search for oracle checking $14 < cost(x) <= 16$

We present our results of QPU launches in figure 3. We observe two major patterns. *First*, the median number of launches for both methods does not increase with the size of the problem. This is a projection of theoretical prediction, that the number of iterations is expected logarithm for GAS and guaranteed logarithmic for GBS, with respect to the upper limit. In our experiment, the maximum limit grows linearly with the number of items, and thus logarithmically for the problem search space size. This means here we observe $\log \log N$ tendency, which grows very slow to catch. *Second* observation is that GBS launches twice as many quantum circuits. This pattern holds for oracle launch counts as well – this metric is proportional to the number of utilized quantum gates. This value is essential in practice, while, for example, the IonQ QPU pricing schema is per gate execution [11]. Code profiling showed that our method can spend a lot of launches to prove that there is no item in the range.

Proposed GBS method inherits the deterministic bisection method, thus incorrect Grover Search is the only source of mistakes in the result. At the same time, the nondeterministic GAS convergence procedure can be an additional source of errors. We observed, that our method has a systematically higher success rate on experimental data (see figure 4). We think that this observation supports the deterministic approach.

V. Discussion

We want to emphasize that our experiments are neither a proof of supremacy in precision nor a justification for the worse gate expenditure. We showed that for a random problem generation with no specific tuning, our method achieves correct solution with a high probability, and shows expected asymptotic complexity characteristics.

Nevertheless, we want to share two observations derived from the experiments. First, both GAS and proposed GBS methods strongly depend on the lower bound estimation at the first iteration. If the constraints on the problem are loose (in our example *low* estimation is unsatisfactory), the Grover search

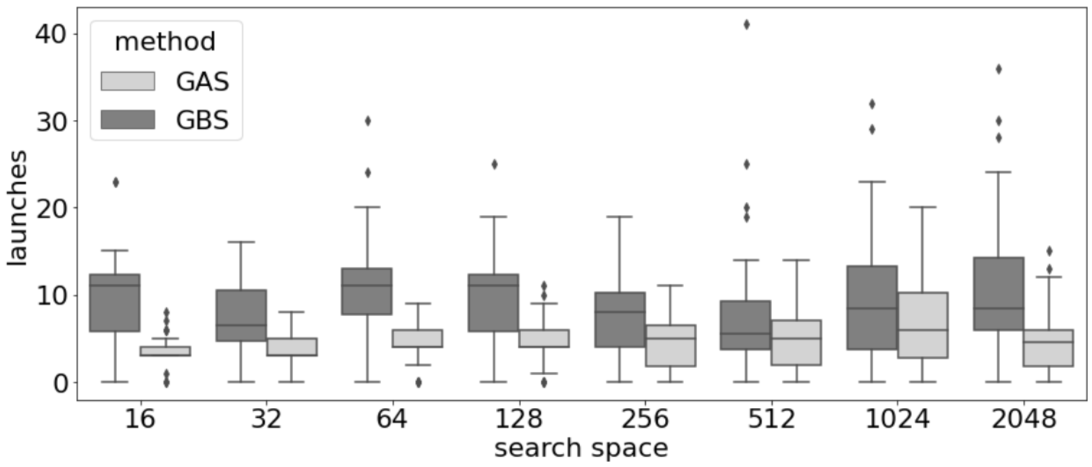


Fig. 3: Comparison or Grover Adaptive Search (GAS) and proposed method (GBS). Number of QPU circuit launches.

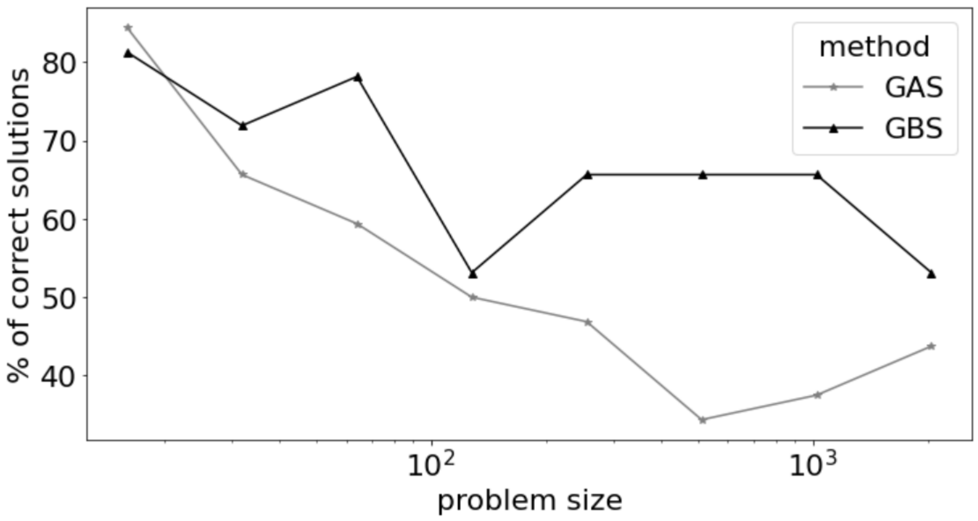


Fig. 4: Success rate of GAS and proposed (GBS) algorithms.

algorithm behaves extremely badly, as it instead amplifies the probabilities of incorrect solutions. In cases where the number of valid solutions is close to half of the search space, Grover search is equivalent to random sampling from the search space. Only a good estimation of a lower bound can kickstart a method, which uses Grover search inside.

Second, GAS and GBS, probably, best suit different problems. Imagine a problem search space with a plateau of multiple similar costs and a single optimal solution above the plateau. The iterative method can easily get stuck on this plateau, making minor improvements at each iteration, while the binary search proceeds if it sees a solution "ahead", with higher costs. To support the observation, we compose a synthetic example, which has a very low chance of succeeding with GAS, but works great for GBS: [21, 19, 1, 1, 1, 1, 1, 1, 1, 1] with the limit of 39. We ran 80 experiments for each method, and GBS found a correct answer in 91% of the launches, while GAS could succeed only once. In the failure cases, the errors of both methods did not exceed 10% (see Figure 5). For practical problems, we recommend using both the GAS and GBS methods if the behavior of the $cost$ function is unknown.

VI. Conclusion

In this paper, we present a novel optimization method, the Grover Binary Search, which combines the brilliant idea of Grover Adaptive Search with the deterministic nature of the binary search. The new method appears to be asymptotically better, but in practice it consumes more quantum gates, which is essential in the NISQ era. However, in the simulations, our method showed a systematically better success rate. Moreover, we could dedicate a separate subset of optimization cost landscapes, which suit much better for GBS method. We support our findings with the experiment, where with GBS we could achieve 91% of success rate, which can be significant for problems where it is crucial to find global optimum.

Additionally, while today's QPU access is mostly cloud-based, we think that it is perspective to design fully-quantum methods to avoid multiple data encoding-decoding. Binary

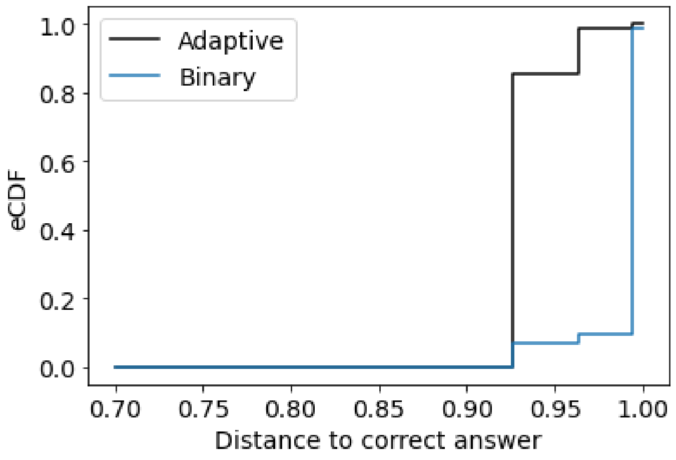


Fig. 5: Empirical cumulative distribution of GAS and GBS answers compared to correct on a synthetic test.

search has a quantum version, thus it is potentially possible to build a quantum-only implementation of GBS. This method will be qubit-hungry, but it can still be preferred to multiple CPU-QPU communication cycles.

References

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, Ieee, 1994, pp. 124–134.

[2] D. Coppersmith, *An approximate fourier transform useful in quantum factoring*, 2002. arXiv: quant-ph/0201067 [quant-ph].

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[4] A. Gilliam, S. Woerner, and C. Gonciulea, "Grover adaptive search for constrained polynomial binary optimization," *Quantum*, vol. 5, p. 428, 2021.

[5] A. M. Childs, A. J. Landahl, and P. A. Parrilo, "Quantum algorithms for the ordered search problem via semidefinite programming," *Phys. Rev. A*, vol. 75, p. 032 335, 3 2007. DOI: 10.1103/PhysRevA.75.032335. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.75.032335.

[6] G. Brassard, P. Høyer, and A. Tapp, "Quantum counting," in *Automata, Languages and Programming*, K. G. Larsen, S. Skyum, and G. Winskel, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 820–831, ISBN: 978-3-540-68681-1.

[7] A. Y. Kitaev, *Quantum measurements and the abelian stabilizer problem*, 1995. arXiv: quant-ph/9511026 [quant-ph].

[8] M. S. A. et al, *Qiskit: An open-source framework for quantum computing*, 2021. DOI: 10.5281/zenodo.2573505.

[9] T. G. Draper, "Addition on a quantum computer," *arXiv preprint quant-ph/0008033*, 2000.

[10] E. R. Johnston, N. Harrigan, and M. Gimeno-Segovia, *Programming Quantum Computers: essential algorithms and code samples*. O'Reilly Media, 2019.

[11] IonQ. "Google cloud platform: Ionq quantum cloud. https://console.cloud.google.com/marketplace/product/ionq-public/ionq." (), [Online]. Available: https://console.cloud.google.com/marketplace/product/ionq-public/ionq. (accessed: 07.02.2022).