Quantifying Grover speed-ups beyond asymptotic analysis

Chris Cade[1,2], Marten Folkertsma[3], Ido Niesen[1,2], and Jordi Weggemans[3]

[1]QuSoft & University of Amsterdam (UvA), Amsterdam, the Netherlands
[2]Fermioniq, Amsterdam, the Netherlands
[3]QuSoft & CWI, Amsterdam, the Netherlands

September 19, 2023

Run-times of quantum algorithms are often studied via an asymptotic, worst-case analysis. Whilst useful, such a comparison can often fall short: it is not uncommon for algorithms with a large worst-case run-time to end up performing well on instances of practical interest. To remedy this it is necessary to resort to run-time analyses of a more empirical nature, which for sufficiently small input sizes can be performed on a quantum device or a simulation thereof. For larger input sizes, alternative approaches are required.

In this paper we consider an approach that combines classical emulation with detailed complexity bounds that include all constants. We simulate quantum algorithms by running classical versions of the sub-routines, whilst simultaneously collecting information about what the run-time of the quantum routine would have been if it were run instead. To do this accurately and efficiently for very large input sizes, we describe an estimation procedure and prove that it obtains upper bounds on the true expected complexity of the quantum algorithms.

We apply our method to some simple quantum speedups of classical heuristic algorithms for solving the well-studied MAX-k-SAT optimization problem. This requires rigorous bounds (including all constants) on the expected- and worst-case complexities of two important quantum sub-routines: Grover search with an unknown number of marked items, and quantum maximum-finding. These improve upon existing results and might be of broader interest.

Amongst other results, we found that the classical heuristic algorithms we studied did not offer significant quantum speedups despite the existence of a theoretical per-step speedup. This suggests that an empirical analysis such as the one we implement in this paper already yields insights beyond those that can be seen by an asymptotic analysis alone.

Contents

1 Introduction

There is growing motivation to design and evaluate quantum algorithms for commercial applications in order to assess the potential impact of quantum computers. To determine whether, or when, a quantum algorithm should be used for a task will involve comparing the candidate quantum algorithm, or set of algorithms, to an existing state-of-the-art classical one. A common approach to benchmark and compare algorithms is to consider their performances on worst-case instances, by providing upper bounds to their runtimes: bounds that hold for every possible instance of the problem the algorithm is designed to solve. In this context, a quantum speedup of a classical algorithm refers to the use of quantum algorithmic techniques that give an improvement over the worst-case runtime of the classical algorithm in question. In some cases, *expected* run-times are considered, where the expectation is taken over the internal (classical or quantum) randomness of the algorithm, and then upper bounds on this expectation are compared. In even rarer cases, it is possible to rigorously analyse the *average*-case complexity of the algorithms, where now the average is taken over the set of inputs [4].

However, such worst-case (and to a lesser extent, average-case) upper bounds can often be misleading: it is not uncommon for algorithms with a large worst-case run-time to perform very well in practice [22, 33].[1] For instance, this is especially true of heuristic algorithms, which are commonly used to solve real-world problems and are often fine-tuned to perform well on instances of interest, rather than on an artificial instance designed to be as difficult as possible for the algorithm but that will likely not appear in a natural setting. As such, much of quantum algorithms research focuses either on exponential quantum speedups (e.g. Shor's algorithm), in which case the speedup obtained by the algorithm is unambiguous; or when only a modest quantum speedup is available, on situations where the run-time of both the quantum and classical algorithms can be determined in a reasonably tight way: the square-root speedup obtained by Grover's algorithm for unstructured search is a simple example of this. For more complicated (quantum and classical) algorithms, however, it might be that the run-time suggested by an asymptotic analysis fails to capture the true complexity of the algorithm on inputs that will be encountered in practice, which can make it difficult to determine the usefulness of a candidate quantum algorithm over a classical one. A similar observation was made in [21], where the authors point out that this disconnect is one of the main reasons that it is so difficult to design quantum algorithms for machine learning and assess their performance relative to their classical counterparts.

In this paper we continue along a line of work that moves beyond performing purely asymptotic analyses of quantum algorithms towards ones of a more empirical nature. In the time before large fault-tolerant quantum computers become readily available, we suggest that for the majority of quantum algorithms, an intermediate form of classical simulation + run-time estimation is possible, and that it can allow for meaningful and informative comparisons to be made. Our particular approach combines tight asymptotic analysis with classical simulation, in an attempt to carefully estimate the run-time of quantum algorithms in lieu of actually being able to run them on a quantum device. Importantly, our methodology is sensitive to the input given to the quantum algorithm.

To verify the utility of our approach, we perform such an empirical analysis for a set of

[1] Interestingly, this observation can actually be made rigorous for some algorithms in certain situations [17].

reasonably simple quantum versions of a classical heuristic algorithm for a particular use-case: that of MAX-k-SAT. The quantum speedups we obtain are quite typical of quantum speedups of classical optimisation algorithms: the classical routine repeats a number of steps, the kth taking some time t_k – which will depend on what happened in previous steps – until convergence; the quantum algorithm does the same, except with each step taking time now $\approx \sqrt{t_k}$. To assess the usefulness of such a quantum algorithm, we must ask: to what extent does this square-root-like speedup manifest in the algorithm when it is actually run to convergence? Moreover, it is likely that the behaviour of the algorithm will differ substantially on different inputs, and hence a further question we should ask is: how much of a speedup does the quantum algorithm obtain on a *representative or real-world* input? We seek to answer such questions with our empirical approach.

1.1 Summary of results

Our main results and contributions are:

- Improved analyses of upper bounds on the expected and worst-case complexities of Grover search when the number of marked items is unknown including log and constant factors, improving upon analyses performed in earlier works (e.g. [34, 7]). We also consider how to optimize the number of classical samples drawn before Grover iterations are used. Sections 2.1, 2.2.

- Upper bounds on the expected complexity of a quantum maximum finding algorithm, improving upon those in previous works (e.g. [14, 1]). Section 2.3.

- An estimation procedure that allows us to use the above bounds to obtain estimates of the expected run-times of repeated calls to a Grover search sub-routine when the number of marked items cannot be computed exactly (in our classical simulations), something that is useful (and indeed necessary) for bench-marking quantum algorithms on very large inputs. The outputs of the procedure come with theoretical guarantees. Section 3.

- A general approach combining the above that allows for rigorous (and efficient) classical estimation of the run-times of quantum algorithms that make repeated calls to Grover sub-routines. This is achieved via classical emulation of the underlying quantum algorithms.

- Two simple quantum heuristic algorithms for MAX-k-SAT, which are basic quantizations of classical 'hill climber' algorithms. Section 4.2.

- We find that the quantum hill climbers obtain favourable scaling compared to their classical counterparts, but that only one of them (the 'simple' hill climber) obtained an absolute speedup for the problem sizes we considered. We observe that some, but not all, of the per-step speedup indicated by an asymptotic analysis manifests in the final behaviours of the algorithms. Section 4.3.

- We verify that our estimation procedure does indeed yield accurate estimates of the expected run-times of our algorithms when compared to an exact method. Section 4.3.

1.2 Concurrent work

In a concurrent work [8], we apply our methodology to help design quantum algorithms for a common task in complex network analysis. The quantitative analysis employed in this other study is notably more comprehensive than the one employed for the elementary hill-climbing algorithm examined in this paper, and thus, the two studies are complementary: the former serves as an introductory exposition to the methodology, while the latter showcases its effectiveness when utilized for developing quantum algorithms for practical problems. More specifically, the other work studies the *Louvain algorithm*[2], which forms one of the main tools for tackling a problem ubiquitous to the study of complex networks: that of *community detection*. Together with its descendants, the Louvain algorithm has successfully been used to study large sparse networks with millions of vertices [6, 13, 20, 27]. In [8], we introduce several quantum versions of the Louvain algorithm, analyse their asymptotic (worst-case) complexities, and investigate numerically how they perform on randomly generated networks, as well as on real-world data sets.

1.3 A broader perspective

We remark that the kind of analysis we perform should in principle be possible for all quantum algorithms that achieve small polynomial speedups over classical ones: we can always 'simulate' the quantum algorithm by running its classical equivalent (which will be only polynomially slower!), and simultaneously estimate how long the quantum routine would have made if it were run instead. All that is required are appropriately tight bounds (including constants, etc.) on the run-times of the quantum sub-routines used by the algorithm. With all of the above in mind, the semi-empirical approach to practical quantum algorithm design and analysis that we use fits into a larger framework that follows the following structure:

- **Design a quantum algorithm** or collection of algorithms, perhaps via speedup of an existing classical algorithm.
- **Choose a measure of complexity** for the algorithms, ideally one that is agnostic about the capabilities of future hardware[3]. This could be for instance the number of time-steps, or the number of queries to the input or to some function.
- **'Simulate' the quantum algorithms on inputs of interest** by replacing the quantum routines with their classical counterparts, and instead collect information to estimate what the quantum complexities would have been if those sub-routines were used instead. This will require one to obtain or prove (ideally tight) bounds on the worst/expected-case complexities of the quantum subroutines used by the algorithms.
- **Use these empirical results** to inform the choice or design of the quantum algorithms. For instance, one might observe that a particular quantum algorithm can be made faster in practice by simplifying it and sacrificing some asymptotic speedup.

[2]The algorithm takes the name of the city in which it was developed. The original paper describing the method has been cited over 15,000 times, and the algorithm itself can be found in all popular graph/network analysis software packages.

[3]For reasons explained in Section 1.4.

As we will see in the sections that follow, there are further considerations that must be taken into account that can prove to be tricky even for simple algorithms such as the ones we consider, suggesting that such an analysis is unlikely to be entirely straightforward in general. Nevertheless, a very fruitful next step could be to build such 'pseudo-simulation' of quantum algorithms into any of the number of quantum programming languages now available [5, 15, 23, 26, 32], which might allow for these sorts of empirical analyses to be performed more quickly and painlessly, and hence facilitate faster quantum algorithm development and prototyping.

1.4 Methodology

Here we describe our methodology for comparing the run-times of quantum and classical algorithms. The most obvious way to do this is to run both algorithms on their respective devices and measure the time they take to run to completion. Unfortunately, quantum hardware is currently not sufficiently developed to be able to run any of the algorithms we describe, and therefore such a comparison is not possible at this point, and will likely not be for the foreseeable future. An alternative approach is to simulate the quantum algorithm at the qubit level on classical hardware, count the number of quantum gates applied and then compare this to the required number of classical gates. This is often the approach currently taken with heuristic quantum algorithms such as VQE [29, 12, 9] and QAOA [35, 30]. However, such simulations are almost always going to be restricted to a few qubits, which means that a comparison between the classical and quantum algorithms can only be made for very small input sizes. Since we are interested in investigating how well the classical and quantum versions of our algorithms compare on actual datasets, which will generally be very large, this method of comparison is insufficient.

Moreover, since quantum technologies are still in their infancy, it is not unlikely that they will improve significantly over the coming years. With this prospect in mind, a comparison that depends heavily on the properties of current-day (or even current-day predictions of) quantum hardware might become obsolete in the near future. For this reason, we aim to make our comparisons *architecture independent*: this will in particular mean not explicitly counting the number of gates needed to implement the algorithms, or taking into account the overheads from error correction. Hence, our comparisons will be of a more qualitative nature than a quantitative one: we are interested, in principle, in how much of the speedup suggested by an asymptotic analysis manifests in the final behaviour of the algorithm – if no speedup appears at this stage, then it certainly won't appear *after* taking into account the aforementioned overheads.

In lieu of estimating actual running times for our algorithms, we fix a suitable notion of complexity and use this to directly compare the classical and quantum algorithms. In particular we opt to count the number of calls made to a particular function (in fact, the very function we are trying to maximise). This essentially equates to measuring query complexity, where we count queries to a function rather than to, say, the input. Counting the number of function calls of course does not capture every costly component of the algorithms that we consider: there are parts that add to the run-time but do not require function calls. However, as is common in the study of query complexity, we choose a suitable measure of complexity so that these parts are those for which we do not obtain any quantum speedup, and hence cost the same quantumly as they do classically – things such as updating what is stored in memory

after completing a step of the algorithm. From the perspective of quantum speedups, comparing the number of function calls made by the quantum and classical algorithms can therefore serve as a proxy for how much of a speedup we can expect to gain on the part of the classical algorithm that admits a speedup. Finally, we note that choosing this complexity measure preserves the architecture independence that we strive for in our analysis, by, for example, ignoring precisely how long a memory update takes, or how many items can be retrieved from memory in a single computational 'step'.

For simplicity, we focus our attention on quantum algorithms that are composed of a number of steps, each of which consists of some classical computation as well as one or more calls to a Grover search on a list containing an unknown number of marked items, and/or to a quantum maximum-finding sub-routine[4]. The quantum algorithms for MAX-SAT discussed in Section 4 are examples of such an algorithm. We consider situations in which the list itself and the number of marked items in it will differ for each step, and in particular will depend on the outcome of the calls that came before it, making the behaviour of the algorithm sensitive to the input itself, as well as the (possibly random) outcomes of the processing during each step. Precisely, we consider quantum algorithms with the structure of Algorithm 1.

Algorithm 1 Generic quantum algorithm structure

1: **Input** X, **Memory** M
2: **for** $k = 1, \ldots, T$ **do**
3: Do some classical processing on X and M, resulting in some list L_k containing t_k marked items.
4: Perform either one or more (perhaps nested) Grover searches with an unknown number of marked items on L_k, or run quantum maximum-finding on the list L_k, to obtain some item x_k.
5: Do some more classical processing given x_k, update M.
6: **end for**

In order to estimate the run-time of such an algorithm given some particular input we would, following our approach, execute all steps except step 4 of Algorithm 1 as they would normally be executed classically, but then replace the step 4 with its classical alternative, and instead estimate how long it *would have* taken if the quantum routine were called. In this way, we can estimate the run-times for different inputs of any quantum algorithm that follows this basic structure.

As we will see in Section 2, even to estimate the complexities of algorithms that make use only of Grover search and quantum maximum finding already requires a somewhat substantial effort. There we prove rigorous upper bounds (including constants) on the expected and worst-case query complexities of Grover search with an unknown number of marked items – something that, to our knowledge, has not been done elsewhere.[5]. Using these bounds, we

[4]I.e. the algorithm looks for either *some* solution at each step (Grover search), or the *best* one (quantum maximum-finding).

[5]Not for the expected complexity, and for the worst-case complexity not in a way that is sufficient to estimate the number of queries made by the quantum algorithm, including all constant and logarithmic overheads, that holds for all input sizes

obtain bounds on the expected complexity of quantum maximum finding, improving upon previous results from the literature.

Our approach can of course be extended to more complicated algorithms that make use of different quantum sub-routines by proving analogous bounds for those sub-routines. Here, however, we keep our focus narrowly on quantum algorithms of the simple structure described above, so that we can apply and demonstrate the usefulness of our approach.

Finally, we note that the run-time estimates for the steps in line 4 will depend on the number t_k of marked items in the list L_k; however, it might well be that we *don't know* how many marked items are there during any one step, and moreover this could be prohibitively time-consuming to compute. In such a situation we may be forced to estimate how many marked items there are, and this will introduce some error into our run-time estimates that we have to handle carefully. For instance, an unbiased estimate of the number of marked items in a list can give us a *biased* estimate for the run-time of Grover search – we discuss this and other considerations in more detail in Section 3.

1.5 Previous work

There have been an increasing number of papers that perform precise resource estimates for a number of quantum algorithms, mostly with a focus on algorithms for simulation of physical systems [28, 18, 31]. Others, such as [3], have investigated what impact overheads such as error-correction might have on potential quantum speedups, in this case concluding that, at least in the near-term, quadratic or small polynomial speedups are unlikely to manifest in practice. Finally, Campbell et al. [11] performed a rigorous analysis of the potential speedups achievable by quantum algorithms for solving constraint satisfaction problems. They considered upper bounds on the run-times of both a naive application of Grover search as well as an optimized implementation of a more sophisticated quantum algorithm for backtracking due to Montanaro [19], taking into account realistic properties of near-term as well as future hardware. They then compared these run-times to the performance of state-of-the-art classical algorithms in an effort to understand when quantum algorithms might provide a performance advantage, and what resources would be required for this.

Our current work is similar in that we also use rigorous upper bounds on the complexities of our quantum sub-routines, although we are often more interested in *expected* complexities. Moreover, we attempt to perform an analysis that is architecture independent, whereas Campbell et al. were interested in hardware properties. We also consider quantum algorithms whose run-times cannot be analysed ahead of time, and which must be implemented, or simulated, in order to discover the speedups (or lack thereof) that they might achieve in practice.

There have also been works that aim to prove rigorous upper bounds on the complexities of various Grover search routines. For example, Zalka [34] performed a careful analysis to upper-bound the number of Grover iterations performed in the worst-case on a list with an unknown number of marked items. We make use of this result, and improve upon it, by extending the analysis to consider the *expected* number of queries made by the algorithm, which requires substantially more effort to bound (tightly). Finally, we note that our analysis holds for all input sizes, whereas (as we understand it) Zalka's result applies only in the limit of large input size.

More recently, an arxiv preprint [24] appeared that claimed that Grover's algorithm offers no quantum advantage. That paper explores the question of whether one would expect a

speedup from applying Grover search in a different way than we do. We consider the speedup obtained by Grover vs. it's classical counterpart - i.e. brute-force search using the same query oracle that Grover has access to. They consider the question of whether Grover's algorithm itself can be classically simulated in practice whilst retaining the square-root speedup, which essentially boils down to studying the difficulty in classically simulating coherent calls to the oracle. It would be interesting (but far beyond the scope of this work) to add this approach to the toolbox when studying whether one can expect to obtain a speedup in practice via application of Grover-type quantum algorithms. The remainder of their paper considers the effects of noise on the performance of Grover's algorithm, which we explicitly avoided in our analysis.

Organization

We begin in Section 2 by explicitly describing an implementation of Grover search with an unknown number of marked items followed by an implementation of a quantum maximum finding routine. We then derive tight upper bounds, including all constants, for the expected and worst-case complexities of these quantum sub-routines. In Section 3, we consider how to apply these bounds for a particular input without knowing ahead of time the parameters needed to compute them, and propose an estimation procedure that deals with this uncertainty. Finally, in Section 4, we apply our methodology to the use-case of MAX-SAT, and present our numerical results.

2 Query complexity bounds

In this section and the next we introduce the tools that form the backbone of our methodology for estimating the run-times of quantum algorithms, in the sense described above. The two main tools that we will require are: a set of rigorous upper bounds on the expected- and worst-case query complexities of Grover search with an unknown number of marked items and quantum maximum-finding (Section 2), and some technical results that allow us to estimate these complexities even when the exact number of marked items is unknown to us (Section 3).

As mentioned, our main quantum sub-routine will be a Grover search with an unknown number of marked items – which we shall refer to as **QSearch** – that can find and return a marked item from a list L of length $|L|$ using an expected $O\left(\sqrt{\frac{|L|}{t}}\right)$ number of queries, when there are t marked items in L:

Lemma 1 (Grover's search with an unknown number of marked items [7])**.** *Let L be a list of items, and t the (unknown) number of 'marked items'. Let $\mathcal{O}_g |x_i\rangle |0\rangle = |x_i\rangle |g(x_i)\rangle$ be an oracle that provides access to the Boolean function $g : [|L|] \to \{0, 1\}$ that labels the items in the list. Then there exists a quantum algorithm* **QSearch**(L, ϵ) *that finds and returns an index i such that $g(x_i) = 1$ with probability at least $1 - \epsilon$ if one exists and requires an expected number $O(\sqrt{N/t}\log(1/\epsilon))$ queries to $\mathcal{O}_g$ and $O(\sqrt{N/t}\log(N/\epsilon))$ other elementary operations. If no such x_i exists, the algorithm confirms this and to do so requires $O(\sqrt{N}\log(1/\epsilon))$ queries to $\mathcal{O}_g$ and $O(\sqrt{N}\log(N/\epsilon))$ other elementary operations.*

We will also find the following variant of Grover search useful in proving our bounds.

Lemma 2 (Exact Grover search [16]). *Let L be a list of items, and $t > 0$ the* known *number of 'marked items'. Let $\mathcal{O}_g |x_i\rangle |0\rangle = |x_i\rangle |g(x_i)\rangle$ be an oracle that provides access to the Boolean function $g : [|L|] \to \{0,1\}$ that labels the items in the list. Then there exists a quantum algorithm* **ExactQSearch**(L,t) *that finds and returns an index i such that $g(x_i) = 1$ with* certainty. *To do so, the algorithm makes $O(\sqrt{N/t})$ queries to $\mathcal{O}_g$ and $O(\sqrt{N}\log(N))$ other elementary operations.*

Finally, we will make use of the quantum maximum-finding algorithm of [14]

Lemma 3 (Quantum maximum-finding [14]). *Let L be a list of items of length $|L|$, with each item in the list taking a value in an ordered set, to which we have coherent access in the form of a unitary that acts on basis states as*

$$\mathcal{O}_L |x\rangle |0\rangle = |x\rangle |L[x]\rangle .$$

Then there exists a quantum algorithm **QMax**(L,ϵ) *that will return* $\arg\max_x L[x]$ *with probability at least* $2/3$ *using at most $O(\sqrt{|L|})$ queries to $\mathcal{O}_L$ (i.e. to the list L) and $O(\sqrt{|L|}\log|L|)$ elementary operations. By repeating the algorithm* $\log(1/\epsilon)$ *times, the probability of success can be amplified to $1-\epsilon$.*

In the sections that follow we carefully bound the expected and worst-case query complexities, including all constants, of **QSearch** on a list with an unknown number of marked items, and then of **QMax**. We consider two different implementations of **QSearch**, one that performs better in the expected case, the other better in the worst case.

The implementation of **QSearch** uses both queries to the function g (classical queries) and the oracle $\mathcal{O}_g$ (quantum queries). Typically, the oracle $\mathcal{O}_g$ can be constructed from a reversible classical circuit implementing g, in which case a single query to $\mathcal{O}_g$ will generally require two queries to g (to compute and uncompute garbage). When we refer to *queries*, we will always mean queries to g itself. A query to $\mathcal{O}_g$ will then correspond to potentially multiple queries to g, and will be weighed with a constant c_q denoting the number of queries made to g per query to $\mathcal{O}_g$. Generally speaking, and for the case of MAX-SAT discussed in Section 4, $c_q = 2$ as mentioned above.

For notation, we will use E to denote an expected number of queries to g, and W for worst-case, with the name of the algorithm in question in the subscript. E.g. if we run **QSearch** on a list L of length $|L|$ with t marked items and a success probability of at least $1-\epsilon$, then the number of queries will be denoted by $E_{\textbf{QSearch}}(|L|,t,\epsilon)$ in the expected case, and by $W_{\textbf{QSearch}}(|L|,\epsilon)$ in the worst-case.

2.1 Expected query complexity of **QSearch**

Our implementation of **QSearch** is based on the implementation of Boyer et al. [7], which takes as input a list L of length $|L|$ and a unitary/oracle that gives coherent access to the function $g : L \to \{0,1\}$. Let $t = |\{x \in L : g(x) = 1\}|$ be the unknown number of marked items in L, which is assumed to be $\leq 3|L|/4$. We first introduce the algorithm **QSearch**$_\infty$, which consists of the following five steps:

1. Set $\lambda = 6/5$, and initialize[6] $m = \lambda$.
2. Choose j uniformly at random from the set of non-negative integers smaller than m.
3. Apply j Grover iterations to the uniform superposition of all items in the list
4. Observe the list register.
5. If the observed item is marked, return it and exit; otherwise, set $m = \min(\lambda m, \sqrt{|L|})$, and go back to step 2.

Note that **QSearch**$_\infty$ will always find a marked item if there is one, but will run forever if there are no marked items. To obtain an algorithm with a finite stopping time one can add an appropriate time-out, in which case the algorithm has some probability of failing (reporting no marked items when there are in fact some). Boyer et al. note that the case $t > 3|L|/4$ can be disposed of in constant time using classical sampling. In order to simulate the behaviour of above algorithm numerically, we must include the classical sampling and time-out features explicitly.

In fact, in Appendix A.1 we show that it is not necessary to assume $0 < t \leq 3|L|/4$, and therefore the classical sampling part is optional. However, in case of many marked items, drawing classical samples is more efficient than applying Grover iterations, and for this reason we keep the classical sampling phase in our implementation of **QSearch** below, with the number of classical samples N_{samples} as a hyperparameter. We discuss how to pick an optimal N_{samples} in Section 2.1.2.

2.1.1 Implementation

We now give our implementation of **QSearch**$(L, N_{\text{samples}}, \epsilon)$ Here, L is the list that contains marked and unmarked items, N_{samples} is the number of classical samples we take and $1 - \epsilon$ is the required lower bound for the success probability. We also define $\lambda = 6/5$ and $\alpha = 9.2$.

Our implementation works as follows. After sampling at most N_{samples} items from L classically, we execute N_{runs} *Grover runs*, where a single *Grover run* is an application of **QSearch**$_\infty$ with a time-out, given by $Q_{\max} = \alpha\sqrt{|L|}$ queries, i.e. lines 8 - 17 of Algorithm 2. The number of runs depends on the desired success probability: $N_{\text{runs}} = \lceil \log_3(1/\epsilon) \rceil$. A single application of **QSearch**$_\infty$ with timeout consists of several *Grover cycles*. That is, for a single run of **QSearch**$_\infty$ with timeout, we first initialize $m = \lambda$, and then repeatedly (i) pick an non-negative integer j less than m, (ii) do j Grover iterations and (iii) measure, and if we don't find a marked item we increase m by a factor of λ. Steps (ii) - (iii) will be referred to as a Grover cycle, see Algorithm 3. Finally, a single application of the Grover iterate[7] will be referred to as a *Grover iteration*.

In Appendix A (precisely Appendices A.1-A.3) we show that **QSearch**, as given by Algorithm 2, has the properties stated in the lemma below.

[6] Boyer et al. initialize $\lambda = 1$, which means they always start with one sample drawn uniformly at random. We choose to start with $\lambda = 6/5$.

[7] The unitary that first reflects through the unmarked states followed by a reflection through the uniform superposition.

Algorithm 2 QSearch

function QSearch(List L, integer N_{samples}, real number $\epsilon > 0$)
  $x \leftarrow$ ClassicalSampling(L, N_{samples})
  if x is marked **then**
    return x
  end if
  $N_{\text{runs}} \leftarrow \lceil \log_3(1/\epsilon) \rceil$, $Q_{\max} \leftarrow \alpha\sqrt{|L|}$, $r \leftarrow 0$ ▷ $\alpha = 9.2$
  while $r < N_{\text{runs}}$ **do**
    $m \leftarrow \frac{6}{5}$, $Q_{\text{sum}} \leftarrow 0$
    Sample a non-negative integer j less than m uniformly at random
    while $Q_{\text{sum}} + j \leq Q_{\max}$ **do**
      $y \leftarrow$ GroverCycle(L, j)
      if y is marked **then**
        return y
      else
        $Q_{\text{sum}} \leftarrow Q_{\text{sum}} + j + 1$ ▷ Add $j + 1$ quantum queries to total
        $m \leftarrow \min(\lambda m, \sqrt{|L|})$ ▷ Update max number of iterations next cycle
        Sample a non-negative integer j less than m uniformly at random
      end if
    end while
    $r \leftarrow r + 1$
  end while
  return No marked item found
end function

Algorithm 3 Subroutines QSearch

function ClassicalSampling(List L, integer N_{samples})
  $k \leftarrow 0$
  while $k < N_{\text{sample}}$ **do**
    Sample an element x from L uniformly at random
    if x is marked **then**
      return x
    end if
    $k \leftarrow k + 1$
  end while
  return No marked item found
end function

function GroverCycle(List L, non-negative integer j)
  Prepare uniform superposition over all elements of L
  Do j Grover iterations
  Measure list-index register
  return measurement outcome
end function

Lemma 4 (Worst-case expected complexity of **QSearch**). *Let L be a list, $g : L \to \{0,1\}$ a Boolean function, $N_{samples}$ a non-negative integer and $\epsilon > 0$, and write $t = |g^{-1}(1)|$ for the (unknown) number of marked items of L. Then, **QSearch**$(L, N_{samples}, \epsilon)$ as described by Algorithm 2 finds and returns an item $x \in L$ such that $g(x) = 1$ with probability at least $1 - \epsilon$ if one exists using an expected number of queries to g that is given by*

$$E_{\textbf{QSearch}}(|L|, t, N_{samples}, \epsilon) = \frac{|L|}{t}\left(1 - \left(1 - \frac{t}{|L|}\right)^{N_{samples}}\right) + \left(1 - \frac{t}{|L|}\right)^{N_{samples}} c_q E_{Grover}(|L|, t)\,, \tag{1}$$

where

$$E_{Grover}(|L|, t) \leq F(|L|, t)\left(1 + \frac{1}{1 - \frac{F(|L|,t)}{\alpha\sqrt{|L|}}}\right), \tag{2}$$

with

$$F(|L|, t) = \begin{cases} \frac{9}{4}\frac{|L|}{\sqrt{(|L|-t)t}} + \left\lceil \log_{\frac{6}{5}}\left(\frac{|L|}{2\sqrt{(|L|-t)t}}\right)\right\rceil - 3 \leq \frac{\alpha\sqrt{|L|}}{3\sqrt{t}} & \text{for} \quad 1 \leq t < \frac{|L|}{4} \\ 2.0344 & \text{for} \quad \frac{|L|}{4} \leq t \leq |L|. \end{cases} \tag{3}$$

If no marked item exists, then the expected number of queries to g equals the number of queries needed in the worst case (denoted by $W_{\textbf{QSearch}}(|L|, N_{samples}, \epsilon)$), which is given by

$$E_{\textbf{QSearch}}(|L|, 0, N_{samples}, \epsilon) = W_{\textbf{QSearch}}(|L|, N_{samples}, \epsilon) \leq N_{samples} + \alpha c_q \lceil \log_3(1/\epsilon)\rceil)\sqrt{|L|}\,. \tag{4}$$

In the formulas above, c_q is the number of queries to g required to implement the oracle $\mathcal{O}_g |x\rangle |0\rangle = |x\rangle |g(x)\rangle$, and $\alpha = 9.2$.

Note that our obtained expression for $E_{\textbf{QSearch}}(|L|, t, N_{\text{samples}}, \epsilon)$ is actually independent of ϵ for $t > 0$ because our upper bound for $E_{\text{Grover}}(|L|, t)$ is independent of ϵ, which is a consequence of the fact that we don't have a lower bound on the failure probability; see Appendix A.3 for details. Also, for the case $1 \leq t \leq |L|$, even though it might appear to be, the expected number of queries for the classical sampling part is actually not linear in $|L|$ since the term $\left(1 - \left(1 - \frac{t}{|L|}\right)^{N_{\text{samples}}}\right)$ contains a factor of $\frac{t}{|L|}$.

2.1.2 Optimizing N_{samples}

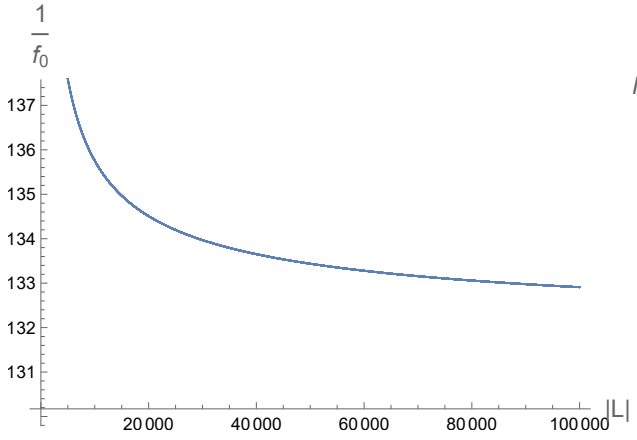
As mentioned in the beginning of Section 2.1, the number of classical samples we use for **QSearch** is a hyperparameter, and in this subsection we discuss how it can be optimized and set to improve the performance of the algorithm for different inputs.

Classical sampling requires fewer queries than Grover search does when a large fraction of the items is marked, whereas it is more efficient to not use classical sampling at all when a small number of them is marked. When a fraction f of items are marked, then an expected $1/f$ classical queries will be required to find one. To determine when classical sampling is more efficient than quantum, we can compare this quantity with the number of queries made by Grover search. That is, given a list L of size L, we want to find out for what value of f we have

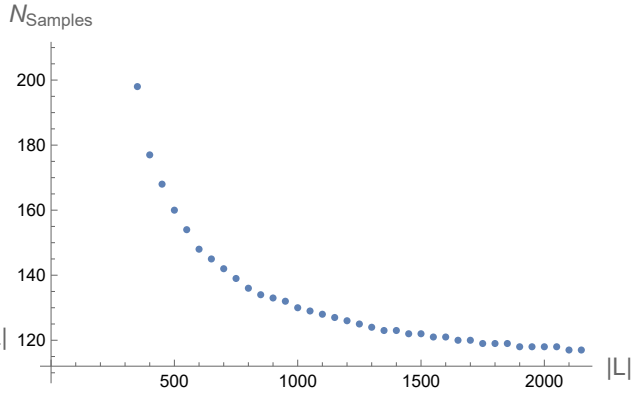
$$\frac{1}{f} = E_{\text{Grover}}(|L|, f|L|). \tag{5}$$

The fraction where equality is attained we call f_0. When $f < f_0$, the Grover part requires fewer queries than classical sampling does, and therefore setting $N_{\text{samples}} > 0$ (i.e. turning the classical sampling part on) increases the query count compared to having $N_{\text{samples}} = 0$. We can numerically compute $f_0(|L|)$ for different values of $|L|$. We observe the following.

- For $|L| \leq 260$, classical sampling always requires fewer queries.
- For $|L| \geq 260$, the value for $1/f_0$ that makes the rightmost inequality in Eq. (5) an equality is plotted as a function of $|L|$ in Fig. 1a.



(a) The value for $1/f_0$ as a function of the list length $|L|$ that marks the point beyond which, in expectation, Grover search requires fewer queries than sampling classically does. In the limit $|L| \to \infty$, there is a horizontal asymptote at $1/f_0 \to 131.665$.



(b) The optimal setting for N_{samples} that minimizes the expected number of queries of **QSearch** when we have no prior knowledge on the number of marked items t, i.e. every value of t is equally likely.

In practice, we do not know what the fraction of marked items $f = t/|L|$ is. For certain algorithms, we might have some information about what f can be, and in such cases this information can be leveraged to our advantage (see e.g. [8]). In case we have *no prior knowledge* on the number of marked items at all, we can assume every value of t is equally likely. In this case, the expected number of queries for **QSearch** is given by

$$\frac{1}{|L|}\sum_{t=1}^{|L|} E_{\textbf{QSearch}}(|L|, t, N_{\text{samples}}, \epsilon).$$

Numerically minimizing[8] the expression above as function of N_{samples} for small list sizes yields the graph in Fig. 1b, which gives an indication for what settings of N_{samples} work well in practice. For our particular numerical simulations in Section 4.3.3, we set $N_{\text{samples}} = 130$.

2.2 Worst-case query complexity of **QSearch**

If we make use of a slightly different implementation of **QSearch** described by Zalka [34], we can obtain a tighter bound on its worst-case performance, which will be useful for some

[8]Recall that our upper bound for $E_{\textbf{QSearch}}(|L|, t, N_{\text{samples}}, \epsilon)$ given by Eq. (1) is independent of ϵ for $t > 0$.

of our applications of **QSearch** where we only care about the worst case (since the expected complexity of this variant is actually worse than that of the **QSearch** implementation described in the previous section). Unfortunately, the bounds given in [34] are only asymptotic and ignore, for example, extra constants arising from rounding integers, and so we briefly re-derive them below. The main upshot is that the dependence on the error becomes quadratically better than the usual implementation, which could end up being a significant improvement for our algorithms. To distinguish this implementation of **QSearch** from the one outlined in Section 2.1, we will refer to this quantum sub-routine as $\mathbf{QSearch}_{\text{Zalka}}$.

As usual, let L be the list of items over which we are searching, and suppose that t of them are marked, and that we want to succeed in finding one if it exists with probability $\geq 1-\epsilon$. The algorithm, based on the one described in [34], consists of the following steps:

1. A preliminary step that checks for a *small* number of marked items, by systematically ruling out $t = 1, t = 2, \ldots, t = t_0$ for (for reasons made clear in Appendix A.4) $t_0 = \lceil \frac{\ln \epsilon}{2\ln(3/4)} \rceil$, by running *exact* Grover search for each value of t. If this step finds a marked item, we return it and stop.

2. A second step where (now with the knowledge from the first step that $t_0 < t$) we repeatedly choose an integer j uniformly at random from the range $[0, \lceil \frac{\pi}{4}\sqrt{\frac{|L|}{t_0}} \rceil]$ and then run Grover search using j iterations. This is done $2t_0$ times (which minimises the part of the complexity that depends on $|L|$). If a marked item is found during any run, we return it and stop. Otherwise, the algorithm returns 'no marked item'.

Run-time analysis The worst-case run-time is clearly when there are no marked items, and hence both steps above are run to the end. In Appendix A.4 we prove the following lemma.

Lemma 5 (worst-case complexity of $\mathbf{QSearch}_{\text{Zalka}}$)**.** *Let L be a list of items, $g : L \to \{0,1\}$ a Boolean function and $\epsilon > 0$, and write c_q for the number of queries to g required to implement the oracle $\mathcal{O}_g |x\rangle |0\rangle = |x\rangle |g(x)\rangle$. Then, with probability of failure at most ϵ, $\boldsymbol{QSearch}_{Zalka}$ requires at most*

$$W_{\boldsymbol{QSearch}_{Zalka}}(|L|, \epsilon) := c_q \left(5 \left\lceil \frac{\ln(1/\epsilon)}{2\ln(4/3)} \right\rceil + \pi \sqrt{|L|} \sqrt{\left\lceil \frac{\ln(1/\epsilon)}{2\ln(4/3)} \right\rceil} \right) \tag{6}$$

queries to g to find a marked item of L, or otherwise to report that there is none.

2.3 Quantum maximum finding **QMax**

We use the quantum maximum finding algorithm from Ahuja and Kapoor [1], described below. We then provide an improved analysis[9] for the expected number of queries made by their quantum maximum finding algorithm.

The input of the algorithm is once again a list L, together with a function $R : L \to \mathbb{R}$ that assigns a value to each item. The output is the index of an element of L that maximises R.

[9] We also believe that there is a slight error in the proof provided by [1], which we fix in our proof.

We assume we have coherent oracle access to the following marking function f defined by

$$f_i(j) = \begin{cases} 1 & \text{if} \quad R(j) > R(i) \\ 0 & \text{otherwise}\,, \end{cases} \tag{7}$$

i.e. access to unitaries $\mathcal{O}_{f_i}$ that acts as

$$\mathcal{O}_{f_i} |j\rangle |0\rangle = |j\rangle |f_i(j)\rangle \ . \tag{8}$$

2.3.1 Infinite-time algorithm

To start with, we define a zero-error infinite-time[10] algorithm for finding the maximum. Afterwards, we will incorporate a time-out in the infinite algorithm, and use Markov's inequality to turn the infinite algorithm into a bounded-error algorithm that terminates in finitely many steps.

Algorithm 4 $\mathbf{QMax}_\infty$

1: **function** $\mathbf{QMax}_\infty$(List L)
2: Choose $i \in L$ uniformly at random and set $y = R(i)$.
3: **while True do**
4: Apply $\mathbf{QSearch}_\infty$ to the list L with the marked items being $f_y^{-1}(1)$.
5: Update $y = R(j)$, where $j \in L$ is the item found by $\mathbf{QSearch}_\infty$.
6: **end while**
7: **return** y
8: **end function**

We say that $\mathbf{QMax}_\infty$ has *found the maximum* when y in Algorithm 4 is equal to an item that maximises R. In Appendix B.1, we prove the lemma stated below.

Lemma 6. *[Expected complexity of $\boldsymbol{QMax}_\infty$] Let L be a list of $|L|$ items. Then, the expected number of queries to any of the f_i (as defined as in Eq. 7) required for $\boldsymbol{QMax}_\infty$ to find the maximum of L is upper bounded by*

$$E_{\boldsymbol{QMax}_\infty}(|L|) \leq c_q \sum_{t=1}^{|L|-1} \frac{F(|L|,t)}{t+1}\,, \tag{9}$$

where $F(|L|,t)$ is defined by Eq. (3). Here, c_q is the number of queries to f_i required to implement the oracle $\mathcal{O}_{f_i}$ (which we assume to be the same for all i).

In case we are interested in queries to R rather than the f_i, then we note that the total number of queries to any of the f_i combined is equal to the total number of queries to R (since for each f_i we need to compute $R(i)$ only once, and this we do anyway at the end of every Grover run to check if the found item is marked), *except* at the very beginning, where in line 2 of Algorithm 4 we need to compute $R(y)$. Therefore, the number of queries to R is upper bounded by Eq. (9) plus one. Consequently, when running $\mathbf{QMax}_\infty$ a total of T times,

[10] $\mathbf{QMax}_\infty$ will not stop running when it has found the maximum: it will continue to run indefinitely because it will be running $\mathbf{QSearch}_\infty$ on a list with no marked items.

if we switch from upper bounding queries to any of the f_i to queries to R, we need to add T to our obtained bound for the former to obtain a bound for the latter.

Next, we can use the bounds for the expected number of queries made by $\mathbf{QSearch}_\infty$ to bound the expected number of queries for $\mathbf{QMax}_\infty$. Using the upper bound in Eq. (3), we have

$$F(|L|,t) \leq \begin{cases} \frac{9.2\sqrt{|L|}}{3\sqrt{t}} & \text{if} \quad 1 \leq t < \frac{|L|}{4} \\ 2.0344 & \text{if} \quad \frac{|L|}{4} \leq t \leq |L| \end{cases} \tag{10}$$

Hence, we obtain

$$E_{\mathbf{QMax}_\infty}(|L|) \leq c_q \left(\frac{9.2\sqrt{|L|}}{3} \sum_{t=1}^{\lceil |L|/4 \rceil - 1} \frac{1}{\sqrt{t}(t+1)} + 2.0344 \sum_{t=\lceil |L|/4 \rceil}^{|L|-1} \frac{1}{t+1} \right) . \tag{11}$$

If the list L is not too large, we can compute the above upper bound by evaluating the sum explicitly. If this computation becomes too time-consuming, we can also resort to bounds that are easier to evaluate. Two such bounds are derived in Appendix B.2, and are given below. We have the following loose upper bound

$$E_{\mathbf{QMax}_\infty}(|L|) \leq c_q \left(6.3505\sqrt{|L|} + 2.8203 \right) ,$$

as well as a tighter upper bound, given by

$$E_{\mathbf{QMax}_\infty}(|L|) \leq c_q \Bigg[\frac{3\sqrt{3}(1+\pi)}{4}\sqrt{|L|} + \frac{\ln(|L|/4)}{2\ln(6/5)} \Big(\ln(|L|/3) + \ln(|L|/4+1) \Big) \\ - 2\ln(|L|/4) + 5.3482 + \frac{\mathrm{Li}_2(-\lceil |L|/4 \rceil + 1)}{2\ln(6/5)} \Bigg] ,$$

where Li_2 is Spence's function, also known as the dilogarithm. For the second bound the leading order term in $\sqrt{|L|}$,

$$c_q \frac{3\sqrt{3}(1+\pi)}{4}\sqrt{|L|} \leq c_q 5.3801\sqrt{|L|} ,$$

has a smaller coefficient than the first upper bound has.

2.3.2 Finite-time bounded-error algorithm

Next, we can introduce a timeout $Q_{\text{timeout}} = 3E_{\mathbf{QMax}_\infty}$ to make $\mathbf{QMax}_\infty$ a finite-time bounded-error algorithm. If X is the random variable corresponding to the number of queries to g made by $\mathbf{QMax}_\infty$ in order to find the maximum of the list L, then by Markov's inequality,

$$\Pr[X \geq Q_{\text{timeout}}] \leq \frac{E_{\mathbf{QMax}_\infty}}{Q_{\text{timeout}}} \leq \frac{1}{3} ,$$

resulting in a maximum-finding algorithm that finds the maximum with probability at least $\frac{2}{3}$ and uses an expected number of queries that is upper-bounded by Q_{timeout}.

We can further boost the success probability to $1-\epsilon$ by repeating the above $\log_3(1/\epsilon)$ times, and picking the largest element (with respect to the function R) out of all repetitions, to obtain the algorithm **QMax** that succeeds with probability at least ϵ and makes at most $\lceil\log_3(1/\epsilon)\rceil Q_{\text{timeout}}$ queries in expectation.

Corollary 1 (Expected complexity of **QMax**)**.** *Let L be a list of items of length $|L|$. Let f_i be the marking functions as defined in Eq. 7. Then the expected number of queries to f_i (for any i) required for **QMax** to find the maximum of L with success probability at least $1-\epsilon$ is $\lceil\log_3(1/\epsilon)\rceil 3E_{\textbf{QMax}_\infty}(|L|)$, where $E_{\textbf{QMax}_\infty}(|L|)$ is given by Eq. (9).*

Using the upper bounds derived for $E_{\textbf{QMax}_\infty}$ derived above, it is sufficient to choose

$$Q_{\text{timeout}} \geq c_q\left(19.0515\sqrt{|L|}+8.4609\right)$$

using the loose upper bound, or

$$Q_{\text{timeout}} \geq c_q\left[\frac{9\sqrt{3}(1+\pi)}{4}\sqrt{|L|}+\frac{3\ln(|L|/4)}{2\ln(6/5)}\Big(\ln(|L|/3)+\ln(|L|/4+1)\Big)-6\ln(|L|/4)\right.$$
$$\left.+16.0466+\frac{3\text{Li}_2(-\lceil|L|/4\rceil+1)}{2\ln(6/5)}\right]$$

using the tight upper bound.

3 Estimating complexities under uncertainty

To estimate the query complexities of **QSearch** and **QMax**, we can use the bounds derived in the previous sections for their expected- and worst-case complexities. These bounds take as input a list L, the desired success probability of the sub-routine, and in case of **QSearch** also the number t of marked items in L. However, the number of marked items in the list will not be known ahead of time, and moreover could be computationally time-consuming to compute classically, especially for very large inputs, which will likely be the ones for which we want to estimate the run-times of quantum algorithms.

Additionally, for algorithms of form of Algorithm 1 that make use of repeated calls to **QSearch** or **QMax**, we would like that with high probability *every* call to either **QSearch** or **QMax** succeeds, which will require boosting their success probabilities to something (inversely) proportional the number of times they are run. However, the number of times each sub-routine is called will often not be known until the algorithm has finished executing, and therefore we will require a reliable upper bound T to the number steps, i.e. the number of times such calls are made.

In this section, we discuss how to deal with both quantities. First, in Section 3.1, we discuss how to use a sampling procedure to estimate the number of marked items using sampling, and consider the extra complications that arise when we use such estimated values to compute (bounds on) the complexities of algorithms. Next, in Section 3.2, we discuss how the total number of steps affects the accuracy of both **QSearch** and **QMax**, as well as the accuracy of the estimates for the expected number of queries made by these quantum routines as obtained through the sampling procedure discussed in Section 3.1.

3.1 Estimating the number of marked items

To use the bounds on the number of queries made by **QSearch** derived in Section 2.1, we need to know how many marked items there are in the list given as input to a **QSearch** call. For sufficiently small lists, we can count the number of marked items exactly at reasonably little computational cost. For longer lists this will become very time consuming, leading to exceedingly slow simulations. When determining the number of marked items exactly becomes infeasible, we can instead estimate the number of marked items. We can do this by counting the number of samples l we need to draw on average before we find a marked vertex.

For a list L with t marked items, the probability that an element of L chosen uniformly at random is marked is $f = t/|L|$. Consequently, the probability that we find a marked item after randomly choosing (with replacement) $k \in \mathbb{N}$ elements of L (the first $k-1$ elements not being marked) is given by

$$\Pr[l = k] = (1-f)^{k-1} f \tag{12}$$

(i.e a geometric distribution with parameter f). We write $l \sim \mathrm{Geo}(f)$ to denote a random variable sampled according to such a distribution, and throughout this section, when we take the expectation value over l it is implied that we do this over the geometric distribution, i.e.:

$$\mathbb{E}[X(l)] = \mathbb{E}_{l\sim \mathrm{Geo}(f)}[X(l)]$$

for any function $X : \mathbb{N} \to \mathbb{R}$. By sampling $l \sim \mathrm{Geo}(f)$, we obtain an unbiased estimate of

$$\mathbb{E}[l] = 1/f = |L|/t\,,$$

which we can use to approximate the expected number of queries made by **QSearch** if it were run on L.

To start we focus on our upper bound for the expected number of (quantum) queries (E_{Grover}) to the oracle $\mathcal{O}_g$ made by **QSearch**. In order to estimate (an upper bound for) $E_{\mathrm{Grover}}(|L|, t)$, we would like an estimator $E^{\mathrm{estimator}}_{\mathrm{Grover}}$ such that the procedure (i) sample $l \sim \mathrm{Geo}(f)$, and then (ii) plug the result into our expression for $E^{\mathrm{estimator}}_{\mathrm{Grover}}(l)$ gives, in expectation (over l), an upper bound to $E_{\mathrm{Grover}}(|L|, t)$; i.e. we want $\mathbb{E}[E^{\mathrm{estimator}}_{\mathrm{Grover}}(l)] \geq E_{\mathrm{Grover}}(|L|, t)$.

A naive attempt at constructing $E^{\mathrm{estimator}}_{\mathrm{Grover}}$ would be to take our upper bound on $E_{\mathrm{Grover}}(|L|, t)$ from Eq. (2) and in this expression replace $1/t$ by $l/|L|$. However, from Eq. (3), we observe that $F(|L|, t)$ contains concave functions like the square-root and the logarithm, which, by Jensen's inequality satisfy

$$\mathbb{E}[\sqrt{l}] \leq \sqrt{\mathbb{E}[l]} = \sqrt{1/f} \quad \text{and} \quad \mathbb{E}[\log_{\frac{6}{5}} l] \leq \log_{\frac{6}{5}} \mathbb{E}[l] = \log_{\frac{6}{5}} 1/f\,.$$

As a consequence, the procedure outlined above (in expectation over l) does *not* give an upper bound to the expected number of queries; instead we obtain a biased estimator that *underestimates* our upper bound for E_{Grover}. Note that the issue of concavity will arise in any approach that tries to simulate a Grover search on an unknown number of marked items by using classical sampling to estimate the fraction of marked items.

We now discuss how to deal with the concavity of the square-root and logarithm, and then describe an estimator that always upper bounds E_{Grover} in expectation.

Upper bound for square-root and log estimates We prove the following two lemmas in Appendices C.1 and C.2:

Lemma 7. *For a random variable X geometrically distributed with parameter f, there exists a constant $d_1 = 4/\pi \approx 1.273$, such that*

$$\sqrt{\mathbb{E}[X]} \leq \mathbb{E}[\sqrt{d_1 X}]$$

for all $f \in (0, 1]$.

Lemma 8. *For a random variable X geometrically distributed with parameter f, there exists a constant $d_2 = e^{\gamma} \approx 1.781$, where γ is the Euler–Mascheroni constant, such that*

$$\log(\mathbb{E}[X]) \leq \mathbb{E}[\log(d_2 X)]$$

for all $f \in (0, 1]$.

Moreover, the proof of Lemma 7 also tells us that in the interesting regime, i.e. where the fraction f is small, the relative error between our upper bound and the actual expected value $\sqrt{1/f}$ goes to zero.

Using the two lemma's above, in Appendix C.3, we show that the following estimator

$$E_{\text{Grover}}^{\text{estimator}}(l) := -1.1272 + \frac{1.7850}{\sqrt{|L|}} + \frac{1.2991}{\sqrt{|L|}} l + \left(5.1962 - \frac{2.5064}{\sqrt{|L|}}\right) \frac{2\sqrt{l}}{\sqrt{\pi}} + \frac{5}{4} \log_{\frac{6}{5}}(e^{\gamma} l) \quad (13)$$

upper bounds E_{Grover} in expectation for all $1 \leq t \leq |L|$ (or $1 \leq \frac{1}{f} \leq |L|$):

$$\mathbb{E}[E_{\text{Grover}}^{\text{estimator}}(l)] \geq E_{\text{Grover}}(|L|, t)\,,$$

where the expectation is taken over the geometric distribution $l \sim \text{Geo}(f)$, with $f = t/|L|$.

Estimation procedure Using the estimator $E_{\text{Grover}}^{\text{estimator}}$, we can construct an estimator that in expectation upper bounds the expected number of queries $E_{\textbf{QSearch}}$ given by Eq. (1). To do so, we require the following two functions on the set of positive integers: $h_1, h_2 : \mathbb{N} \to \mathbb{R}$

$$h_1(l) = \min(l, N_{\text{samples}})\,,$$

and

$$h_2(l) = \begin{cases} 0 & l \leq N_{\text{samples}} \\ 1 & l > N_{\text{samples}} \end{cases}$$

Given h_1 and h_2, in Appendix C.4 we prove the following lemma.

Lemma 9. *The estimator*

$$H(l) = h_1(l) + h_2(l) c_q E_{Grover}^{estimator}(l) \quad (14)$$

upper bounds $E_{\textbf{QSearch}}$ in expectation:

$$\mathbb{E}[H(l)] \geq E_{\textbf{QSearch}}(|L|, t, N_{samples}, \epsilon)$$

for $1 \leq t \leq |L|$ and for all[11] *$\epsilon > 0$, where the expectation value is taken over the geometric distribution $l \sim Geo(f)$, with $f = t/|L|$.*

[11] Note that the derived expression for $E_{\textbf{QSearch}}(|L|, t, N_{\text{samples}}, \epsilon)$ is independent of ϵ for $t \geq 1$.

The above estimator applies to the situation where there is at least one marked item. However in case $t = 0$, in order to determine l we keep drawing samples (with replacement) indefinitely. To make sure our algorithm terminates in finite time, we need to set a maximum $l_{\max}$ such that, if $l = l_{\max}$, we conclude that $t = 0$ and we stop the sampling procedure. The particular choice of $l_{\max}$ will depend on our tolerance for falsely detecting no marked items.

In practice, we use the procedure $\textbf{Estimate}_{\textbf{QSearch}}(L, N_{\text{samples}}, \delta, \epsilon)$ described in Algorithm 5 for estimating (an upper bound to) the expected number of queries to g made by **QSearch**. Recall that c_q is the number of queries to g required to implement the oracle $\mathcal{O}_g$.

Algorithm 5

function $\textbf{Estimate}_{\textbf{QSearch}}$(List L, integer N_{samples}, failure probabilities δ and ϵ)

$l_{\max} \leftarrow \lceil \frac{|L|}{\delta} \rceil$.

Draw samples uniformly at random (with replacement) from L until either finding a marked item, or making $l_{\max}$ samples. Let l be the number of samples taken.

if $l \leq N_{\text{samples}}$ **then**

Then a marked item would have been found classically, in which case

$$E \leftarrow l\,.$$

else if $N_{\text{samples}} < l \leq l_{\max}$ **then**

Then the marked item would not have been found classically, and some Grover iterations would have been performed. In such a case,

$$E \leftarrow N_{\text{samples}} + c_q \Bigg[-1.1272 + \frac{1.7850}{\sqrt{|L|}} + \frac{1.2991}{\sqrt{|L|}} l + \left(5.1962 - \frac{2.5064}{\sqrt{|L|}} \right) \frac{2\sqrt{l}}{\sqrt{\pi}} + \frac{5}{4} \log_{\frac{6}{5}} (e^{\gamma} l) \Bigg]\,.$$

else if $l > l_{\max}$ **then** We conclude $t = 0$, and therefore, by Eq. (4),

$$E \leftarrow N_{\text{samples}} + 9.2 c_q \lceil \log_3(1/\epsilon) \rceil) \sqrt{|L|}\,.$$

end if

return E

end function

Lemma 10. *Let L be a list of items, $g : L \to \{0, 1\}$ a Boolean function and $\epsilon, \delta > 0$. Write $t = |g^{-1}(1)|$ for the (unknown) number of marked items of L. Then, with probability at least $1 - \delta$, the procedure $\textbf{Estimate}_{\textbf{QSearch}}(L, N_{samples}, \delta, \epsilon)$ of Algorithm 5, gives an upper bound to the expected number of queries made by $\textbf{QSearch}(L, N_{samples}, \epsilon)$, in expectation over $l \sim Geo(f)$, where $f = t/|L|$.*

Proof. This follows from Lemma 9, except now we have to take into account the possibility that we sample $l = l_{\max}$ even when $t \geq 1$. Recall that in Algorithm 5 we set $l_{\max} = \lceil \frac{|L|}{\delta} \rceil$. Assuming the worst-case of $f = 1/|L|$, by Markov's inequality, this can happen with probability

at most

$$\Pr[l \geq l_{\max}] \leq \frac{\mathbb{E}[l]}{l_{\max}} \leq \frac{|L|}{l_{\max}} = \delta \,.$$

□

This lemma gives a very rough upper bound on the failure probability as it does not take the fraction of marked elements into account. In practice we found that setting $\delta = \frac{1}{100}$ worked well. Also note that, to get more accurate estimates, rather than sampling l and plugging the expression into H, we can also sample l multiple times and take the sample average of all the corresponding H-values. This will however require a somewhat more elaborate failure probability analysis in case some of the sampled l's are equal to $l_{\max}$.

3.2 Unknown number of steps

When running an algorithm of the form of Algorithm 1, we require that all calls to the quantum subroutines **QSearch** or **QMax** to succeed in order to guarantee that the final algorithm worked correctly. This requires boosting their success probabilities, and, in order to do so, we need to know how many times each one is called, which in our case means knowing how many steps the overall algorithm will take. In case of heuristic algorithms, the number of steps is usually not known. However, it is not uncommon for heuristic algorithms that their *typical* behaviour on certain practical problem instances is known – which is the case for, for example, MAX-SAT and community detection [8].

In case nothing is known about the number of steps, then instead we can (i) guess an upper bound to the number of steps, (ii) run the classical algorithm that emulates the quantum algorithm, and (iii) check retro-actively if indeed we required fewer steps than the guessed upper bound. If our guess was too low, then optionally we can increase it and repeat.

Suppose that T is such a (guessed) upper bound to the total number of steps of an algorithm of the form of Algorithm 1, meaning that we call **QSearch** or **QMax** at most T times. By the union bound, given a desired probability of failure of at most ϵ_{total}, the accuracies of the individual subroutines $\epsilon_{\text{subroutine}}$ should be set such that

$$(1 - \epsilon_{\text{subroutine}})^T \geq 1 - \epsilon_{\text{total}} \,,$$

meaning it is sufficient to choose

$$\epsilon_{\text{subroutine}} \leq 1 - (1 - \epsilon_{\text{total}})^{1/T} \,.$$

The above formula in fact holds for the accuracy of the quantum subroutines (denoted by ϵ in the sections above), as well as the parameter δ (which determines $l_{\max}$) in Lemma 10 for the procedure $\textbf{Estimate}_{\textbf{QSearch}}(L, N_{\text{samples}}, \delta, \epsilon)$, since this estimation procedure is called once per step of the (classical simulation of the) algorithm.

4 Use-case: MAX-k-SAT

In this section, we take the tools developed in Sections 2 and 3 and apply them to a particular heuristic, called a *hill-climber*, for finding (approximate) solutions to Boolean satisfiability problems. The algorithm discussed is of the sort that it admits a quantum speedup that is of

the form of Algorithm 1. This section achieves the modest goal of numerically confirming—that the proposed framework works, but is limited in its depth; for a more comprehensive and detailed numerical study (of a different computational problem) we would like to refer the reader to [8] (see also Section 1.2).

4.1 Propositional Boolean Satisfiability (k-SAT)

k-SAT is a fundamental problem in computer science and artificial intelligence, in which we ask whether a satisfying assignment exists for a given Boolean formula in conjunctive normal form, with the property that each clause contains at most k literals. Whilst k-SAT is an example of a *decision problem*, MAX-k-SAT is an *optimization problem* that generalizes k-SAT: it is the problem of determining the maximum number of clauses, that can be made true by an assignment of truth values to the variables of the formula. Let $\mathbf{x} \in \{0,1\}^n$ be bit strings of length n, $C = \{C_i\}_{i=1}^m$ be a set of m clauses, which each act on at most k literals, and $W = \{w_i\}_{i=1}^m \subseteq \mathbb{R}^m$ a set of weights. The goal of MAX-k-SAT is to solve

$$\max_{\mathbf{x}} \varphi(\mathbf{x}),$$

where $\varphi(\mathbf{x}) = \sum_{i=1}^m w_i C_i(\mathbf{x})$. This problem is NP-hard for any $k \geq 2$.

A straightforward heuristic for solving MAX-k-SAT instances is based on *hill-climbing*: the general idea is to start with some initial bit string, and then look for incremental improvements in the direct neighbourhood of this given bit string. This process is repeated iteratively until it has converged to some local maximum or the maximum number of iterations is reached. Hill-climbing belongs to the family of *local search* methods in mathematical optimization. Local search heuristics have been widely studied for SAT and MAX-SAT (see Ref. [25] for an extensive review on local search methods) and are also yet still being studied: see Refs. [2, 10] for some more recent works.

For MAX-k-SAT , we define the d-level neighbourhood $\mathcal{N}_d(\mathbf{x})$ of some bit string $\mathbf{x}$ as the set of all other bit strings that differ from $\mathbf{x}$ in *at most* d bit flips. The total size of this space is given by

$$|\mathcal{N}_d(\mathbf{x})| = \sum_{i=1}^{d} \binom{n}{i} = \mathcal{O}(n^d).$$

For our hill climber heuristic, we either consider a *simple* hill climber, which greedily moves to an arbitrary neighbouring bit string with a strictly larger objective function value, and the *steep ascent* hill climber, which computes φ on all bit strings in the neighbourhood of the current bit string and picks the one that maximises the increase in φ (assuming its objective function value is strictly larger than that of the current bit string.

If we write T for the number of moves made by either the simple or the steep ascent hill climber (which in general will require different number of steps depending on the problem instance, and in case of the simple hill climber also on the internal randomness of the algorithm) the worst-case time complexities of both algorithms have similar mathematical expressions, given by

$$\sum_{t\in[T]} \mathcal{O}(n^d) = \mathcal{O}(Tn^d)\,, \tag{15}$$

because the per step complexities have the same worst-case upper bounds. However, in practice the *expected* run-time of the simple hill climber depends on the instance and the current state of the algorithm: the more bit strings in its neighbourhood increase the objective function value the faster it completes its local search step in expectation. If, at step t of the algorithm, we write $f_{d,t}$ for the fraction of the number of bit strings in $\mathcal{N}_d(\mathbf{x}_t)$ for which φ assumes a value larger than $\varphi(x_t)$, i.e.

$$f_{d,t} = \frac{|\{x \in \mathcal{N}_d(x_t) : \varphi(x) > \varphi(x_t)\}|}{|\mathcal{N}_d(x_t)|},$$

then we can bound the expected number of steps for the simple hill climber by

$$\sum_{t\in[T]} \mathcal{O}\left(\frac{1}{f_{d,t}}\right). \tag{16}$$

4.2 Quantum heuristics for MAX-k-SAT

Both variants of the hill climber search routines lend themselves to be sped up easily by Grover implementations.

To start with, given a bit string $\mathbf{y}$, we define the function

$$f_{\mathbf{y}}(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \varphi(\mathbf{x}) > \varphi(\mathbf{y}) \\ 0 & \text{otherwise}. \end{cases}$$

We assume that, for every bit string $\mathbf{y} \in \{0,1\}^n$, we have oracle access to $\mathcal{O}_{f_{\mathbf{y}}}$.

Lemma 11 (Simple quantum hill-climber)**.** *Let φ be a* MAX-k-SAT *instance on n variables, and assume oracle access to each of the $\mathcal{O}_{f_y}$ as described above. Then there exists a quantum algorithm* ***Simple quantum hill-climber*** *that with probability $\geq 2/3$, behaves identically to a classical simple hill climber and requires at most an expected number*

$$\sum_{t\in[T]} \tilde{\mathcal{O}}\left(\sqrt{\frac{1}{f_{d,k}}}\right) \tag{17}$$

calls to φ.

Proof. We pick an initial bit string as we would with the classical simple hill climber. Next, suppose that in step t of the algorithm our current best bit string is $\mathbf{x}_t$. Here, we replace the local search step of a d-level neighbourhood in the aforementioned classical simple hill-climber by a single call to **QSearch** using the oracle $O_{f_{\mathbf{x}_t}}$. Writing $f_{d,k}$ for the fraction of neighbours of $\mathbf{x}_t$ for which the objective function value is strictly larger than $\varphi(\mathbf{x}_t)$, by Lemma 1 we require at most an expected number $O(\sqrt{\frac{1}{f_{d,t}}}\log(1/\epsilon))$ queries to $\mathcal{O}_{f_{\mathbf{x}_t}}$ and $O(\sqrt{\frac{1}{f_{d,t}}}\log(n^d/\epsilon))$ other elementary operations to find such a candidate $\boldsymbol{x}_{t+1}$ with probability at least $1-\epsilon$. If we set $\epsilon = 1 - \sqrt[T]{\frac{2}{3}}$, our overall success probability will be at least $(1-\epsilon)^T = 2/3$, as required.

Since each query to any of the $O_{f_{\mathbf{x}_t}}$'s requires $O(1)$ queries to φ, the lemma statement follows. □

Lemma 12 (Steep quantum hill-climber)**.** *Let φ be a* MAX-k-SAT *instance on n variables, and assume oracle access to each of the $\mathcal{O}_{f_y}$ as described above. Then there exists a quantum algorithm* ***Steep quantum hill-climber*** *that with probability $\geq 2/3$, behaves identically to some classical steep hill climber and requires at most an expected number*

$$\sum_{t\in[T]} \tilde{\mathcal{O}}(n^{d/2}) \tag{18}$$

calls to φ.

Proof. The proof is similar to the proof of Lemma 11, except that in this case, instead of the classical maximum finding routine, we make use of the quantum subroutine **QMax** of Lemma 3, which also requires access to each of the $O_{f_{\mathbf{x}_t}}$'s. We can find the maximum in $\mathcal{O}(\sqrt{n^d})\log(1/\epsilon)$ queries to each of the $O_{f_{\mathbf{x}_t}}$'s with probability $\geq 1-\epsilon$. We set ϵ in the same way as we did for the simple quantum hill climber to get the desired success probability. □

4.3 Numerics

In this section, we describe our numerical implementations of the classical and quantum versions of the steep and simple hill climbers. We then compare the expected number of queries for the quantum and classical versions when applied to typical problem instances of MAX-k-SAT using the method developed in Sections 2 and 3.

4.3.1 Algorithmic implementations

Classical hill climbers Both classical algorithms are allowed to sample *without* replacement when searching for a good (or the best) element. Therefore we have that, in the case of a simple hill climber, the number of samples $X_{d,t}$ when searching over a list of $|\mathcal{N}_d(\mathbf{x}_t)|$ elements at step t, of which a fraction of $f_{d,t}$ are 'good elements', has an expected value given by

$$\mathbb{E}[X_{d,t}] = \frac{|\mathcal{N}_d(\mathbf{x})|+1}{|\mathcal{N}_d(\mathbf{x})|f_{d,t}+1}. \tag{19}$$

For the steep hill climber, the classical expected number of queries at every step is always equal to $|\mathcal{N}_d(\mathbf{x})|$.

Quantum hill climbers For the quantum algorithms, we set the desired failure probability ϵ for the entire algorithm to be at most 10^{-5}, which can be achieved by setting the accuracy per step to ϵ/T, with T the maximum total number of steps. Empirically, we found that $T=n$ provides a very loose upper bound on the total number of steps taken by the algorithm. Note that the value of T could be optimised more thoroughly – this leads to a smaller total number of queries needed in the quantum setting – but we leave this for now as this is beyond the main goal of this case study.

For the simple hill climber we use two implementations, one that calculates the number of marked elements t (in this case marked elements correspond to possible moves that increase

the objective function value) exactly at every step, and one that acquires only an estimate of this via sampling.

The exact implementation keeps track of the list of all marked elements at every step, which allows us to use our sharper bounds from Lemma 4 in Section 2.1 to upper bound the expected number of queries made by **QSearch** at each step of the algorithm. From this list of marked elements, we select an element at random and use that as an update step for the classical simulation.

The sampling algorithm, just like its classical counterpart, instead samples in search of elements that give an increase in the cost function. When it finds one, use the number of tries l it took to find a marked item as input to estimate (an upper bound) to the expected number of queries **QSearch** would have made, as described in Section 3, to estimate the run-time of the quantum algorithm. This procedure is just an implementation of Algorithm 5 for the case of MAX-k-SAT .

The steep ascent hill climber also keeps track of the complete list of marked items at every step. From this it selects the item with the maximal function value increase. It uses our bounds from Section 2.3 to attain estimates of the expected number of queries **QMax** would have made for every step, in order to estimate the run-time of the entire algorithm.

4.3.2 Numerical implementation

We write the problem as a matrix multiplication problem and use numpy to solve it, which allows for larger instances to be tested. The assignment of truth values $x \in \{0,1\}^n$ is written as a vector $\tilde{x} \in \{-1,1\}^n$ where -1 is assigned to variables that are false and 1 to those that are true. The clauses C can be written in a similar fashion, $\tilde{C}_i \in \{-1,0,1\}^n$, where -1 is assigned to the negated variables, 0 is assigned to the variables that are not in the clause, and 1 to those that should be true according to the clause. We construct a matrix A with the $\tilde{C}_i$'s as rows. This matrix has an efficient sparse representation since most of it's entries are 0. The objective function $\phi(x)$ becomes the following:

$$\tilde{\phi}(\tilde{x}) = W^T \left(\left\lceil \frac{A\tilde{x} + k}{2k} \right\rceil \right) ,$$

where W^T is row vector containing the weights for each clause and k is the number of variables per clause. The addition and division of k is elements-wise, while $A\tilde{x}$ is matrix vector multiplication. Note that $-k \leq A\tilde{x} \leq k$, where the left-hand inequality is only attained when all variables are incorrectly assigned. In that case the ceiling function returns a 0 and in all other cases it returns a 1, as required. In all numerical simulations d (which determines the level of the neighbourhood considered) is set to 1.

Sampling implementation At every step, the sampling algorithm samples up to d (that determines the size of the neighbourhood of $\tilde{x}$ that the hill climber algorithm can search over; in our case $d = 1$) indices of $\tilde{x}$ and flips their value by multiplying by -1. It then calculates the objective function $\tilde{\phi}(\tilde{x})$ and accepts the changes if the cost increased, and rejects otherwise. This is repeated until the algorithm rejects $10n$ times[12] in a row, at which point we assume that the algorithm has converged.

[12]This is the value of $l_{\max} = 10n$ in Algorithm 5.

Exact implementation At every step, the exact implementation calculates the cost increase of every possible change to $\tilde{x}$. This is done by constructing the matrix

$$B(\tilde{x}) = \sum_{ij} x_i(1 - 2\delta_{ij}),$$

which consists of n copies of $\tilde{x}$ as columns, where we multiply all diagonal elements by -1. This represents all the possible changes of $\tilde{x}$ at a single step. Now we can use $\tilde{\phi}(B(\tilde{x}))$ to calculate the cost of all possible changes simultaneously. This gives a vector $\tilde{y}$ containing the cost value of all the n possible new configurations of $\tilde{x}$ (assuming $d = 1$). These values are compared to the old cost value and those that give a positive increase are saved in a list of marked elements. We consider those variables for which a change (being multiplied by -1) incurs a positive increase in the objective function as marked, the all other variables as unmarked. The size of this list gives the exact value of t, the number of marked variables. The exact implementation of simple quantum hill-climber selects one marked element from the list at random. The steep quantum hill-climber selects the marked element with the highest cost value.

Data structure The exact implementation is feasible due to the fact that we use matrix multiplication to calculate the cost values. However, it can still be quite slow for larger instances. To remedy this, to an extent, we add an extra data structure that keeps track of the list of marked variables, rather than reconstruct it at every step. To do so we use the fact that any update is in some sense local. Let the i'th index of $\tilde{x}$ be the index that is updated. Then there is a subset of clauses $\{C_j | C_{ij} \neq 0\}$ (rows of A where the i'th index of the clause is not zero). These are the only clauses that can change from being satisfied to not satisfied, or from not satisfied to satisfied, by changing the i'th index of $\tilde{x}$. Not all variables of $\tilde{x}$ are contained in these clauses (only k variables get assigned a non-zero value in a clause). Exactly those variables that are, can change from being marked to not and visa versa. Hence we only need to consider this subset of variables when updating the list of marked variables. This severely reduces the computational cost of keeping track of marked items. As it turns out, this is efficient enough to avoid running-time limitations, but instead makes memory limitations the bottleneck.

4.3.3 Results

Here we present our results for estimating the run-times of the two quantum algorithms described previously. Specifically, we estimate the number of queries to any of the marking functions[13] $f_{\mathbf{y}}$ from Section 4.2 by applying the bounds obtained in Section 2. We set $c_q = 2$,

[13]We could have also chosen to count queries to φ instead. Note that, after finding $\mathbf{x}_t$ at step t, we know $\varphi(\mathbf{x}_t)$ from the checking part of **QSearch**$_\infty$ (used as a subroutine for both **QSearch** and **QMax**), so every query to $f_{\mathbf{x}_t}$ corresponds one query to φ. The difference between counting queries to the marking functions versus counting queries to φ occurs at initialisation, where we need one extra query to φ to compute the function value of the initial bit string that is not taken into account when counting queries to the marking functions. Hence, for a total of T calls to either **QSearch** or **QMax**, the number of queries to φ equals the number of queries to the marking functions plus T. This relationship holds for both the classical and quantum query counts. In our comparison, we chose to compare queries to marking functions, because this is where the speedup manifests itself.

since the quantum algorithms for MAX-k-SAT make queries to an oracle $\mathcal{O}_{f_\mathbf{y}}$, which requires 2 queries to $f_\mathbf{y}$ to implement.

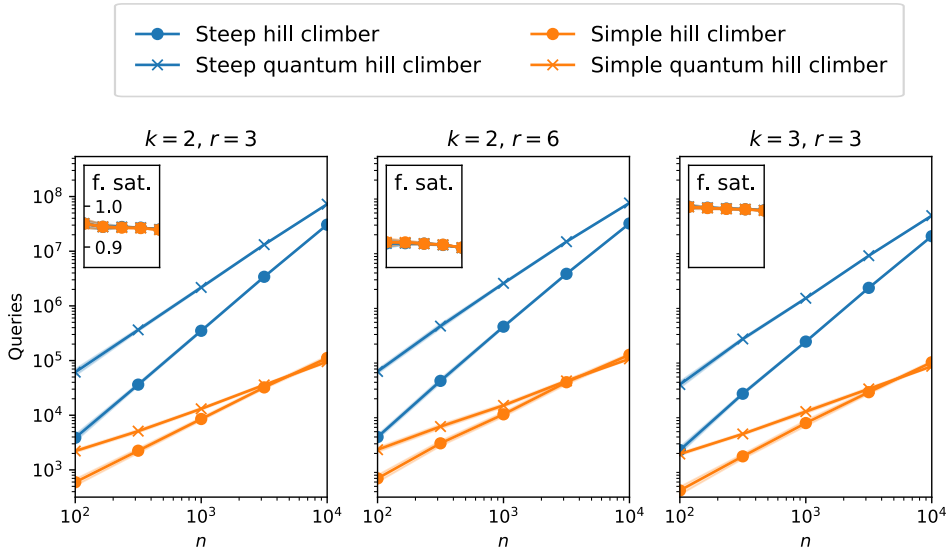


Figure 2: Numerical results for the query counts on randomly generated MAX-k-SAT instances, with n variables and $m = rn$ weighted (uniformly random between 0 and 1) clauses, of the proposed classical and quantum algorithms that implement a hill climber search routine. All hill climbers only consider a $d = 1$ level neighbourhood, so the local search space size is at any step k equal to n. The horizontal axis indicates the total amount of variables n and the vertical axis the amount of queries made to any of the marking functions. Each data point corresponds to the average over 10 randomly generated instances and the shaded area represents one standard deviation. In every sub-figure the inlet plots the fractional number of weighted satisfied clauses, defined as $\varphi(\boldsymbol{x}^*)/W$, where $W = \sum_{i\in[m]} w_i$ is total weight on the m clauses and $\varphi(\boldsymbol{x}^*)$ the objective function value for the obtained solution $\boldsymbol{x}^*$, the x-axis of the subplots is the number of nodes n. The blue and orange lines in the sub-figures are overlapping, this shows that the quality of the solutions found is comparable for the different algorithms. The classical algorithms are indicated by a '•', the respective quantum algorithms by a '×'.

We tested our algorithms on different instances of MAX-k-SAT to see what kind of speed-ups can be attained on average-case instances. The instances were generated using a random assignment of k variables per clause. Figure 2 shows the average number of queries made by our classical and quantum algorithms. There, n is the number of variables, k the number of variables per clause, r is multiplied by n to get the number of clauses $m = rn$. We observe that the behaviour is very similar amongst the different parameter choices in the random MAX-k-SAT generation. We find, as one might expect, that both quantum versions of the steep and simple hill climbers achieve better asymptotic *scaling* when compared to their classical counterparts: here better asymptotic scaling means that we expect that the polynomial which describes the number of queries made to the cost function has a lower degree for the quantum algorithm than it has for the classical one. This is indicated by the difference in slope of the plots in Figure 2, as the number of queries against the problem size is plotted on a log-log scale and thus gives information about the degree of this polynomial, provided n is large enough. On the contrary, only the simple quantum hill climber is able to also beat the classical algorithm in terms the of *absolute* number of queries for the problem

sizes considered (since only in this case the plot corresponding to the number of quantum queries goes below the classical one). However, since it achieves better scaling, we expect that for slightly larger n (larger than 10^4) the steep quantum hill climber will also start to beat the classical counterpart on average, as one would expect. The interesting point here is that, even for a fairly simple model that only takes query counts into consideration, the problem sizes need to already be quite large in order to achieve a quantum speedup.

Table 1 shows the empirically observed asymptotic scaling behaviour of our algorithms. By taking a linear fit in the log-log plot we can estimate the scaling exponents of the different algorithms. In Table 1 we show the relative speedup of our quantum algorithms compared to their classical counterpart. We see that a part of the theoretical speedup is lost. This is likely due to a combination of the fact that the theoretical speedup is a per-step speedup that does not affect the total number of steps taken, only the number of queries required for each individual step, and the fact that on relatively small instances the extra overhead required to run the quantum algorithms is significant.

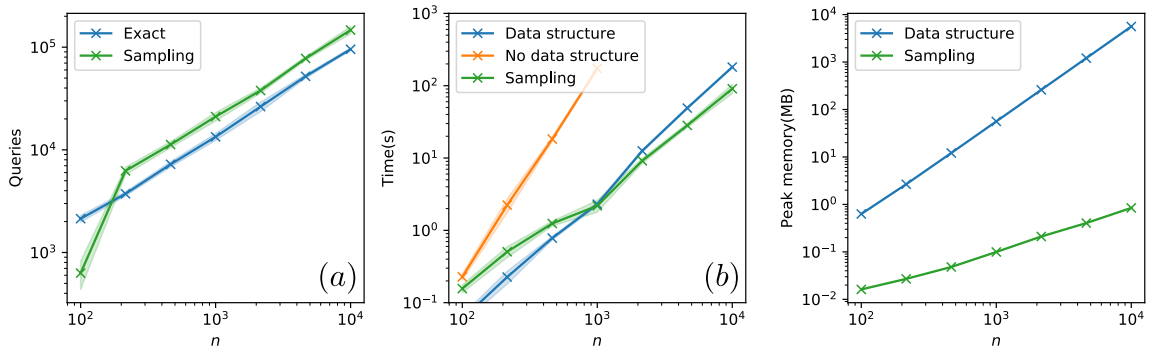


Figure 3: Several numerical results for sampling and exact methods for the simple hill climber on random 2-SAT instances with $r = 3n$ clauses, with $N_{\text{samples}} = 130$ a): average query count, b) average running times and c) peak memory usage. The first plot shows a comparison of query count between the sampling and exact method. Note how for the smallest value of n our sampling method fails to yield a proper upper bound: this is due to the fact that $N_{\text{samples}} > n$, which results in the fact that with high probability we fail to turn on Grover at all, and as a consequence we underestimate the expectation value (since the contribution from Grover to the expectation value is high). The second plot shows a run-time comparison between the exact (with and without data structure) and the sampling method. The third plot shows a comparison of peak memory usage between the exact method (with data structure), and the sampling method. The exact method without data-structure is not shown in the third plot as it has the same memory usage as the sampling method.

As discussed in Section 3, when instances become too large we cannot use an exact method anymore to keep tack of the number of marked items. In Figure 3 we show a comparison between the exact methods and our introduced sampling method for estimating an upper bound on the expected number of queries, for $N_{\text{samples}} = 130$. We find that our estimation method provides a decent upper bound on the number of queries in expectation. For the exact methods we consider two different implementations to acquire the necessary information for calculating the expected number of queries at every step. The first one runs over the entire search space at every step acquiring the number of marked items. The second one uses the

data-structure — described in Section 4.3.2 — that exploits the locality of the instances to update the fraction of 'good elements' in the neighborhood of a given bit string.

Regarding the run-times of our classical simulations, Figure 3 shows that both sampling and the data-structure method considerably outperform the exact implementation that runs over the entire search space. However, the extra added data structure comes at the cost of additional memory requirements, which become the bottleneck as we consider problems at a larger scale. Therefore, for instances where $n > 10^4$, we are limited to the usage of the sampling methods to obtain results. Finally, we note that the data structure method is very context-specific (i.e. here the data structure is specific to MAX-k-SAT) and might not always be possible, whereas the estimation method is applicable generally.

	Classical query complexity per iteration τ	Quantum query complexity per iteration τ	Absolute speed-up observed?	Empirically observed range of polynomial speed-ups
Simple hill climber	$\mathcal{O}\left(\frac{1}{f_\tau}\right)$	$\tilde{\mathcal{O}}(\sqrt{\frac{1}{f_\tau}})$	Yes	1.45-1.72
Steep hill climber	$\mathcal{O}\left(\lvert L\rvert\right)$	$\tilde{\mathcal{O}}(\sqrt{\lvert L\rvert})$	No	1.38-1.60

Table 1: Shown are the theoretically obtained per-iteration complexities of our algorithms compared to their empirically observed speedups across the entire algorithm. Here 'absolute speedup' refers to the quantum algorithm making fewer (estimated) queries than the classical algorithm on the datasets that we considered. The numbers shown in the rightmost column measure the speedup achieved by the quantum algorithm: these are obtained by a linear weighted fit on the plots of Figure 2, which gives the scaling exponent of the expected query counts as a function of the problem size; the number in the table is the classical exponent divided by the corresponding quantum exponent. The numbers are larger than one in all cases, indicating a (modest) quantum speedup. The maximum speedup that can be obtained is 2, since that would correspond to the full quadratic per-step speedup manifesting across the entire run-time. Note that the steep hill-climber would likely also achieve an absolute speed-up if we considered slightly larger problem instances, as it achieves a better scaling than it's classical counterpart.

4.4 Summary of results

Our main findings can be summarised as follows.

- The quantum hill climbers obtain favourable scaling compared to their classical counterparts, but only one of them (the simple hill climber) obtained an absolute (query) speedup compared to its classical counterparts.
- Our estimation procedure gave reliable upper bounds on the complexities of the quantum algorithms as compared to an exact procedure, confirming our theoretical analysis from Section 3.
- Our estimation procedure significantly decreased the computational cost of obtaining run-time estimates in the way considered in this paper. An exact approach that made use of a particular data structure yielded similar results, however it added large memory costs, and such an approach will always be very context-specific and sometimes not possible at all.

- Classical heuristic algorithms tend to work by making many fast-to-compute but small updates to minimize the cost function, a structure that does not lend itself to significant quantum speedups.

References

[1] Ashish Ahuja and Sanjiv Kapoor. A quantum algorithm for finding the maximum. *arXiv:quant-ph/9911082*, 1999. https://doi.org/10.48550/arXiv.quant-ph/9911082 .

[2] Haifa Hamad Alkasem and Mohamed El Bachir Menai. Stochastic local search for partial MAX-SAT: an experimental evaluation. *Artificial Intelligence Review*, 54:2525–2566, 2021. https://doi.org/10.1007/s10462-020-09908-4.

[3] Ryan Babbush, Jarrod R McClean, Michael Newman, Craig Gidney, Sergio Boixo, and Hartmut Neven. Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX Quantum*, 2(1):010103, 2021. https://doi.org/10.1103/PRXQuantum.2.010103.

[4] Shai Ben-David, Benny Chor, Oded Goldreich, and Michel Luby. On the theory of average case complexity. *Journal of Computer and system Sciences*, 44(2):193–219, 1992. https://doi.org/10.1016/0022-0000(92)90019-F.

[5] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. Silq: A high-level quantum language with safe uncomputation and intuitive semantics. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 286–300, 2020. https://doi.org/10.5281/zenodo.3764961.

[6] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P10008, October 2008. https://doi.org/10.1088/1742-5468/2008/10/P10008.

[7] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998. https://doi.org/10.1002/(SICI)1521-3978(199806)46:4/5¡493::AID-PROP493¿3.0.CO;2-P.

[8] Chris Cade, Marten Folkertsma, Ido Niesen, and Jordi Weggemans. Quantum algorithms for community detection and their empirical run-times. `arXiv:2203.06208`, 2022. https://doi.org/10.48550/arXiv.2203.06208.

[9] Chris Cade, Lana Mineh, Ashley Montanaro, and Stasja Stanisic. Strategies for solving the Fermi-Hubbard model on near-term quantum computers. *Physical Review B*, 102(23):235122, 2020. https://doi.org/10.1103/PhysRevB.102.235122.

[10] Shaowei Cai, Chuan Luo, and Haochen Zhang. From decimation to local search and back: A new approach to MAX-SAT. In *IJCAI*, pages 571–577, 2017. https://doi.org/10.24963/ijcai.2017/80.

[11] Earl Campbell, Ankur Khurana, and Ashley Montanaro. Applying quantum algorithms to constraint satisfaction problems. *Quantum*, 3:167, 2019. https://doi.org/10.22331/q-2019-07-18-167.

[12] Pierre-Luc Dallaire-Demers, Jonathan Romero, Libor Veis, Sukin Sim, and Alán Aspuru-Guzik. Low-depth circuit ansatz for preparing correlated fermionic states on a quantum computer. *Quantum Science and Technology*, 4(4):045005, 2019. https://doi.org/10.1088/2058-9565/ab3951.

[13] Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. Generalized louvain method for community detection in large networks. In *2011 11th international conference on intelligent systems design and applications*, pages 88–93. IEEE, 2011. https://doi.org/10.1109/ISDA.2011.6121636.

[14] Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum. `arXiv:quant-ph/9607014`, 1996. https://doi.org/10.48550/arXiv.quant-ph/9607014.

[15] Andrew Hancock, Austin Garcia, Jacob Shedenhelm, Jordan Cowen, and Calista Carey. Cirq: A python framework for creating, editing, and invoking quantum circuits. *URL https://github.com/quantumlib/Cirq*, 2019.

[16] Peter Høyer. Arbitrary phases in quantum amplitude amplification. *Physical Review A*, 62(5):052304, 2000. https://doi.org/10.1103/PhysRevA.62.052304.

[17] Richard M Karp and J Michael Steele. Probabilistic analysis of heuristics. *The traveling salesman problem*, pages 181–205, 1985.

[18] Joonho Lee, Dominic W Berry, Craig Gidney, William J Huggins, Jarrod R McClean, Nathan Wiebe, and Ryan Babbush. Even more efficient quantum computations of chemistry through tensor hypercontraction. *PRX Quantum*, 2(3):030305, 2021. https://doi.org/10.1103/PRXQuantum.2.030305.

[19] Ashley Montanaro. Quantum-walk speedup of backtracking algorithms. *Theory OF Computing*, 14(15):1–24, 2018. http://dx.doi.org/10.4086/toc.2018.v014a015.

[20] Xinyu Que, Fabio Checconi, Fabrizio Petrini, and John A Gunnels. Scalable community detection with the louvain algorithm. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 28–37. IEEE, 2015. https://doi.org/10.1109/IPDPS.2015.59.

[21] Maria Schuld and Nathan Killoran. Is quantum advantage the right goal for quantum machine learning? *Prx Quantum*, 3(3):030101, 2022. https://doi.org/10.1103/PRXQuantum.3.030101.

[22] Daniel A Spielman and Shang-Hua Teng. Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *Communications of the ACM*, 52(10):76–84, 2009. http://doi.acm.org/10.1145/1562764.1562785.

[23] Damian S Steiger, Thomas Häner, and Matthias Troyer. ProjectQ: an open source software framework for quantum computing. *Quantum*, 2:49, 2018. https://doi.org/10.22331/q-2018-01-31-49.

[24] EM Stoudenmire and Xavier Waintal. Grover's algorithm offers no quantum advantage. `arXiv:2303.11317`, 2023. https://doi.org/10.48550/arXiv.2303.11317 .

[25] Thomas Stützle, Holger H. Hoos, and Andrea Roli. A review of the literature on local search algorithms for MAX-SAT. 2001.

[26] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q# enabling scalable quantum computing and development with a high-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pages 1–10, 2018. https://doi.org/10.1145/3183895.3183901.

[27] V. A. Traag, L. Waltman, and N. J. van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9:5233, March 2019. https://doi.org/10.1038/s41598-019-41695-z.

[28] Vera von Burg, Guang Hao Low, Thomas Häner, Damian S Steiger, Markus Reiher, Martin Roetteler, and Matthias Troyer. Quantum computing enhanced computational catalysis. *Physical Review Research*, 3(3):033055, 2021. https://doi.org/10.1103/PhysRevResearch.3.033055.

[29] Dave Wecker, Matthew B Hastings, and Matthias Troyer. Progress towards practical quantum variational algorithms. *Physical Review A*, 92(4):042303, 2015. https://doi.org/10.1103/PhysRevA.92.042303.

[30] Jordi R Weggemans, Alexander Urech, Alexander Rausch, Robert Spreeuw, Richard Boucherie, Florian Schreck, Kareljan Schoutens, Jiří Minář, and Florian Speelman. Solving correlation clustering with QAOA and a Rydberg qudit system: a full-stack approach. *Quantum*, 6:687, 2022. https://doi.org/10.22331/q-2022-04-13-687.

[31] James D Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. Quantum computing resource estimate of molecular energy simulation. *Science*, 309:1704, 2005.

[32] Robert Wille, Rod Van Meter, and Yehuda Naveh. IBM's Qiskit tool chain: Working with and developing for real quantum computers. In *2019 Design, Automation & Test in Europe Conference & Exhibition*, pages 1234–1240. IEEE, 2019. https://doi.org/10.23919/DATE.2019.8715261.

[33] Margaret Wright. The interior-point revolution in optimization: history, recent developments, and lasting consequences. *Bulletin of the American mathematical society*, 42(1):39–56, 2005. https://doi.org/10.1090/S0273-0979-04-01040-7.

[34] Christof Zalka. A Grover-based quantum search of optimal order for an unknown number of marked elements. `arXiv:quant-ph/9902049`, 1999. https://doi.org/10.48550/arXiv.quant-ph/9902049 .

[35] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D Lukin. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *Physical Review X*, 10(2):021067, 2020. https://doi.org/10.1103/PhysRevX.10.021067.

Acknowledgements We would like to thank Harry Buhrman for suggesting the idea of estimating input-dependent run-times, Ian Marshall for helpful discussions, and Quinten Tupker for help with the proof of Lemma 8. The numerics of Section 4.3.3 were carried out on the Dutch national e-infrastructure with the support of the SURF Cooperative.

Funding CC was supported by QuantERA project QuantAlgo 680-91-034, with further funding provided by QuSoft and CWI. MF and JW were supported by the Dutch Ministry of Economic Affairs and Climate Policy (EZK), as part of the Quantum Delta NL programme. IN was supported by the DisQover project: a collaboration between QuSoft and ABN AMRO, and recieved funding from ABN AMRO and CWI.

A Detailed analysis of **QSearch**

In this section of the appendix we give details to support the bounds on the success probability and expected number of queries made by our implementation of **QSearch** given in Section 2.1.

As mentioned in the beginning of Section 2, queries to the quantum oracle $\mathcal{O}_g$ come with a weight of c_q relative to the classical queries to g. In Sections A.1 and A.2, a query refers to a query to $\mathcal{O}_g$. Only in Section A.3 will we include the classical queries, and then the queries to the quantum oracle will by multiplied by an extra factor of c_q (where needed) in the expressions obtained for the expected number of queries to g for **QSearch**.

A.1 Improved bounds

To start with, let us briefly go over the original analysis of Boyer et al. [7], and improve some of the bounds where we can. The analysis in this subsection applies to $\textbf{QSearch}_\infty$.

Suppose we have a list L with t marked items, and let θ be such that

$$\sin^2(\theta) = t/|L|\,.$$

Moreover, let

$$m_t = \frac{1}{\sin(2\theta)} = \frac{|L|}{2\sqrt{(|L|-t)t}}\,.$$

Now, the following lemma provides a lower bound to the success probability of finding a marked item with a single Grover run.

Lemma 13. *(Lemma 2 from [7]). Suppose we have a list L with t marked items, and let θ be such that $\sin^2(\theta) = t/|L|$, $m \in \mathbb{N}_{>0}$ an arbitrary positive integer. Then, the probability P_m of finding a marked element after doing j Grover iterations, where j is a non-negative integer smaller than m chosen uniformly at random, is given by*

$$P_m = \frac{1}{2} - \frac{\sin(4m\theta)}{4m\sin(2\theta)}\,.$$

Consequently, if $m \geq m_t$, then $P_m \geq \frac{1}{4}$.

According to the algorithm description of $\textbf{QSearch}_\infty$, we initialize $m = \lambda$ and $\lambda = 6/5$. After every run, we multiply m by λ. The moment $m > m_t$, we reach the so-called *critical stage*. As Boyer et al. observe, because of Lemma 13, once in the critical stage every run has probability of at least $1/4$ to find a marked item, and this lower bound can be used to upper bound the expected number of Grover iterations required to find a marked item.

The issue with the requirement $m \geq m_t$ is that, when $\theta \to \pi/2$, $m_t \to \infty$. For this reason, Boyer et al. exclude the regime of θ close to $\pi/2$ – which corresponds to the case

of many marked items – by classical sampling. However, as we show below, in the regime $|L|/4 < t \leq |L|$, or $\pi/6 \leq \theta \leq \pi/2$, we actually have $P_m \geq 1/4$ for every integer $m > 0$. More precisely, we have the following lemma.

Lemma 14. *For $|L|/4 < t \leq |L|$, which corresponds to $\pi/6 \leq \theta \leq \pi/2$, we have that*

$$P_m \geq \begin{cases} \frac{1}{4} & \text{for} \quad m = 1 \\ \frac{1}{2} - \frac{1}{1-\frac{\pi^2}{96}} \approx 0.323 & \text{for} \quad m > 1 \end{cases}$$

Proof. Let us start with the easy case of $m = 1$. We have

$$\frac{\sin(4\theta)}{4\sin(2\theta)} = \frac{1}{2}\cos(2\theta)$$

which is upper bounded by $\frac{1}{4}$ for $\pi/6 \leq \theta \leq \pi/2$. Therefore[14], $P_1 \geq \frac{1}{4}$.

Next, we focus on the case $m > 1$. We will prove that

$$\frac{\sin(4m\theta)}{4m\sin(2\theta)} \leq \frac{1}{2\pi}\frac{1}{1-\frac{\pi^2}{96}} \approx 0.177\,.$$

To do so, define $f(\theta) = \frac{\sin(4m\theta)}{4m\sin(2\theta)} = \frac{g(\theta)}{h(\theta)}$, where $g(\theta) = \sin(4m\theta)$ and $h(\theta) = 4m\sin(2\theta)$. We have that $|g(\theta)| \leq 1$. Also, note that $f(\theta)$ is only positive when $g(\theta) \geq 0$. Furthermore, by L' Hôpital's rule,

$$\lim_{\theta\to\frac{\pi}{2}} \frac{\sin(4m\theta)}{4m\sin(2\theta)} = \lim_{\theta\to\frac{\pi}{2}} \frac{4m\cos(4m\theta)}{8m\cos(2\theta)} = -\frac{1}{2}\,.$$

We now distinguish two sub-cases:

Case (a): $\pi/4 \leq \theta \leq \pi/2$. On this interval for θ we have that $h(\theta)$ is a decreasing positive function. The roots of $g(\theta)$ are given by $\theta = \frac{\pi n}{4m}$, $n \in \mathbb{Z}$. In particular, there is one root at $\pi/2$, which corresponds to $n = 2m$. Since $g(\theta)$ is negative just left of $\theta = \frac{\pi}{2}$, the largest value of θ for which $g(\theta)$ is positive is attained when $n \to 2m-1$, which corresponds to $\theta^* = \frac{\pi(2m-1)}{4m}$. Combining the above, we have that

$$f(\theta) \leq \frac{1}{4m\sin(2\theta)} \leq \frac{1}{4m\sin(2\frac{\pi(2m-1)}{4m})} = \frac{1}{4m\sin(\frac{\pi}{2m})}.$$

We now use that $\sin(x) \geq x - \frac{1}{6}x^3$ for $x \geq 0$, which holds since

1. $\sin(x) = x - \frac{1}{6}x^3$ at $x = 0$.
2. $\frac{\partial}{\partial x}\sin(x) = \cos(x) \geq 1 - \frac{1}{2}x^2 = \frac{\partial}{\partial x}\left[x - \frac{1}{6}x^3\right]$, since
   (a) $\cos(x) = 1 - \frac{1}{2}x^2$ at $x = 0$.
   (b) $\frac{\partial}{\partial x}\cos(x) = -\sin(x) \geq -x = \frac{\partial}{\partial x}\left(1 - \frac{1}{2}x^2\right)$ for $x \geq 0$ (using $\sin(x) \leq x$ for $x \geq 0$).

[14] This makes sense, since $m = 1$ corresponds to measuring directly without doing any Grover iterations, and therefore the probability of finding a marked item is at least $\frac{1}{4}$ when $t \geq |L|/4$.

For $m > 1$, we have that $\tilde{m} := \frac{\pi}{2m} \in (0, \frac{\pi}{4}]$, and therefore

$$\frac{1}{4m\sin(\frac{\pi}{2m})} \leq \frac{1}{4m\left(\tilde{m} - \frac{1}{6}\tilde{m}^3\right)} = \frac{1}{4m}\frac{1}{\tilde{m}}\frac{1}{1-\frac{1}{6}\tilde{m}^2} \leq \frac{1}{4m}\frac{2m}{\pi}\frac{1}{1-\frac{\pi^2}{96}}$$
$$= \frac{1}{2\pi}\frac{1}{1-\frac{\pi^2}{96}}\,.$$

Case (b): $\pi/6 \leq \theta \leq \pi/4$. On this interval $h(\theta)$ is an increasing positive function with minimum $\frac{1}{2\sqrt{3}m}$ attained at $\theta = \pi/6$. For $m > 1$, we have that

$$f(\theta) \leq \frac{1}{2\sqrt{3}m} \leq \frac{\sqrt{3}}{12} < \frac{1}{2\pi}\frac{1}{1-\frac{\pi^2}{96}}.$$

Finally, since $P_m = 1/2 - f(\theta)$, we conclude that $P_m \geq \frac{1}{2} - \frac{1}{2\pi}\frac{1}{1-\frac{\pi^2}{96}}$ for $\pi/6 \leq \theta \leq \pi/2$.

□

To start with, let us bound the expected number of queries of $\mathbf{QSearch}_\infty$ to the quantum oracle.

Lemma 15. *The expected number of queries $E^{Quantum}_{\boldsymbol{QSearch}_\infty}$ to the quantum oracle $\mathcal{O}_g$ used by $\boldsymbol{QSearch}_\infty$ when applied to a list L with t marked items can be upper bounded by*

$$E^{Quantum}_{\boldsymbol{QSearch}_\infty}(|L|, t) \leq \begin{cases} \frac{9}{2}m_t + \lceil \log_\lambda(m_t)\rceil - 3 & \text{if} \quad 1 \leq t < \frac{|L|}{4} \\ 2.0344 & \text{if} \quad \frac{|L|}{4} \leq t \leq |L| \end{cases} \tag{20}$$

where $\lambda = \frac{6}{5}$.

Proof. For every Grover cycle of $\mathbf{QSearch}_\infty$, we sample $0 \leq j < m$ uniformly at random. Hence, on average, $j = (\lceil m\rceil - 1)/2$. Because we need one query at the end of each cycle to check if the returned item is marked[15], on average a single Grover cycle uses $(\lceil m\rceil - 1)/2 + 1 \leq m/2 + 1$ queries. Since m is initialized at $m = \lambda$, and multiplied by λ after every run, $m(u) = \lambda^u$ during the u-th cycle.

Now, for $|L|/4 \leq t \leq |L|$ by[16] Lemma 14, writing $p = \frac{1}{2} - \frac{1}{2\pi}\frac{1}{1-\frac{\pi^2}{96}}$, $\mathbf{QSearch}_\infty$ has success probability lower bounded by p and failure probability upper bounded by $\leq 1 - p$. Hence, we can upper bound the expected number of queries for $\mathbf{QSearch}_\infty$ by

$$\sum_{u=1}^{\infty}(1-p)^{u-1}\,p\left(\frac{\lambda^u}{2}+1\right) = \frac{\lambda p}{2}\sum_{u=0}^{\infty}(\lambda(1-p))^u + p\sum_{u=0}^{\infty}(1-p)^u = \frac{\lambda p}{2}\frac{1}{1-\lambda(1-p)} + 1$$
$$= \frac{3}{5}\left(\frac{1}{2} - \frac{1}{2\pi}\frac{1}{1-\frac{\pi^2}{96}}\right)\frac{1}{1-\frac{6}{5}\left(\frac{1}{2}+\frac{1}{2\pi}\frac{1}{1-\frac{\pi^2}{96}}\right)} + 1 \leq 2.0344\,.$$

[15] In terms of queries to the function g, a check actually requires one query to g, not c_q queries to g. Since $c_q \geq 1$, and we are working with upper bounds, and the checking only happens $\lceil \log(m_t)\rceil$ times, we choose to count the check as c_q queries to g to keep the formulas clean. As side note, when we say that we know the value of R for **QMax** or φ for MAX-k-SAT 'from the previous step', this value comes from these classical checks (that occur as a subroutine of both **QSearch** and **QMax**).

[16] Lemma 14 holds for integer m. In our case, $m(u) = \lambda^u$ is not necessarily integer, but since we are picking the integer j less than $m(u)$, we can always replace by $m(u)$ by $\lceil m(u)\rceil$. In particular, since $m(1) = \lambda > 1$, $\lceil m(u)\rceil > 1$ for all u, and we can use the bound in Lemma 14 for $m > 1$.

For $1 \leq t < \frac{|L|}{4}$, we can repeat the analysis of Boyer et al. Before we reach the *critical stage*, i.e. the cycles for which $m < m_t$, the number of queries is upper bounded by

$$\begin{aligned}\sum_{u=1}^{\lceil \log_\lambda(m_t)\rceil - 1} \left(\frac{\lambda^u}{2} + 1\right) &= \frac{1}{2}\frac{\lambda^{\lceil \log_\lambda(m_t)\rceil} - \lambda}{\lambda - 1} + \lceil \log_\lambda(m_t)\rceil - 1 \\ &\leq \frac{1}{2}\frac{\lambda m_t - \lambda}{\lambda - 1} + \lceil \log_\lambda(m_t)\rceil - 1 \\ &= 3m_t + \lceil \log_\lambda(m_t)\rceil - 4\,.\end{aligned}$$

Once $m \geq m_t$, we are in the critical stage, and by Lemma 13, $P_m \geq \frac{1}{4}$. Hence, the expected number of queries is upper bounded by

$$\sum_{u=0}^{\infty} \left(\frac{3}{4}\right)^u \frac{1}{4}\left(\frac{\lambda^{u+\lceil \log_\lambda(m_t)\rceil}}{2} + 1\right) = \frac{\lambda^{\lceil \log_\lambda(m_t)\rceil}}{8}\frac{1}{1 - \frac{3\lambda}{4}} + 1 \leq \frac{\lambda m_t}{8}\frac{1}{1 - \frac{3\lambda}{4}} + 1 = \frac{3}{2}m_t + 1\,.$$

Including the upper bound to the expected number of queries before the critical stage, we arrive[17] at Eq. (20).

□

It should be noted that the bound of $\frac{9}{2}m_t + \lceil \log_\lambda(m_t)\rceil - 3$ actually holds for all t, but only becomes useful when we can further bound m_t (as we do in the next section, which requires that number of marked items is not too large, e.g. $1 \leq t \leq |L|/4$).

A.2 Success probability

Next, we focus on **QSearch** as described by Algorithm 2, which includes a time-out, making it a finite-time bounded-error algorithm whose success probability and complexity we analyse in the subsections that follow.

As a consequence of Lemma 14, we do not need to first sample classically to exclude the case of many marked items before using Grover, because the success probability of a single run is $\geq \frac{1}{4}$ also in the regime of many marked items. Hence, the success probability of **QSearch** only depends on the success probability of the Grover search part (lines 6 - 19 of Algorithm 2), which we investigate next. Note that, in our implementation of **QSearch** given in Algorithm 2, we *do* include the classical sampling part because it can make the algorithm efficient in the regime of many marked items.

If $t = 0$, then the Grover search part will run the maximum number N_{runs} of Grover runs, and return 'no marked item found', which means it will always return the correct answer. Thus, we can restrict ourselves to the case $t > 0$. In this case, the Grover part can only fail when every Grover run fails. For a single Grover run to fail, it has to not find a marked item before the time-out, meaning that $\textbf{QSearch}_\infty$ would have required more than $Q_{\max}$ queries to find a marked item.

[17] The reason that we pick $\lambda = 6/5$ is that it minimizes the coefficient of the dominant term m_t – which by the above two expressions is given by $c(\lambda) = \frac{1}{2}\left(\frac{\lambda}{\lambda-1} + \frac{\lambda}{4-3\lambda}\right)$ – on the interval $\lambda \in (1, 4/3)$. In particular, the choice $\lambda = 6/5$ is optimal.

Let X be the random variable that corresponds to the number of queries to $\mathcal{O}_g$ needed for $\mathbf{QSearch}_\infty$ to find a marked item. We distinguish two cases.[18]

Case 1: $1 \leq t \leq \frac{|L|}{4}$**.** In this case, $\frac{|L|}{|L|-t} \leq \frac{4}{3}$, and therefore[19]

$$m_t = \frac{1}{2}\frac{|L|}{\sqrt{(|L|-t)t}} = \frac{1}{2}\sqrt{\frac{|L|}{|L|-t}}\sqrt{\frac{|L|}{t}} \leq \sqrt{\frac{|L|}{3t}}\,. \tag{21}$$

Now, let us set $Q_{\max} = \alpha\sqrt{|L|}$, where α will be determined below. Using Eq. (20), the probability that $X \geq Q_{\max}$ can be upper bounded by Markov's inequality:

$$\Pr[X \geq Q_{\max}] \leq \frac{\mathbb{E}[X]}{Q_{\max}} \leq \frac{\frac{9}{2}m_t + \lceil \log_\lambda(m_t)\rceil - 3}{\alpha\sqrt{|L|}} \leq \frac{1}{\alpha}\left(\frac{3\sqrt{3}}{2\sqrt{t}} + \frac{\log_\lambda\left(\sqrt{\frac{|L|}{3t}}\right) - 2}{\sqrt{|L|}}\right). \tag{22}$$

We want to make this expression less than or equal to $\frac{1}{3}$ for all $t \geq 1$ and for all $|L|$, which we can accomplished by maximising the above expression with respect to t (which sets $t = 1$ in the above expression), and then choosing α to be

$$\alpha \geq \max_{x\geq 1}\left(\frac{9\sqrt{3}}{2} + 3\frac{\log_\lambda\left(\frac{\sqrt{x}}{3}\right) - 2}{\sqrt{x}}\right) = \frac{9\sqrt{3}}{2} + \frac{25}{36e\ln(\lambda)} \approx 9.1954\,. \tag{23}$$

To keep the notation simple, let us set

$$\alpha = 9.2\,. \tag{24}$$

In particular, setting $\alpha = 9.2$ actually guarantees that

$$\Pr[X \geq Q_{\max}] \leq \frac{1}{3\sqrt{t}}\,, \tag{25}$$

for all $1 \leq t \leq \frac{|L|}{4}$. Indeed, making the rightmost expression in Eq. (22) less than or equal to $\frac{1}{3\sqrt{t}}$ is equivalent to

$$\alpha \geq \left(\frac{9\sqrt{3}}{2} + 3\frac{\left\lceil \log_\lambda\left(\frac{\sqrt{x}}{3}\right)\right\rceil - 3}{\sqrt{x}}\right) \tag{26}$$

for $x = \frac{|L|}{t}$, which holds because of Eq. (23).

Case 2: $\frac{|L|}{4} < t \leq |L|$**.** By Eq. (20),

$$\Pr[X \geq Q_{\max}] \leq \frac{2.0344}{\alpha\sqrt{|L|}}\,. \tag{27}$$

which is also less than or equal to $\frac{1}{3}$ for $\alpha = 9.2$.

[18] We assume that $|L| \geq 4$ throughout this analysis.

[19] Note that the bound $\frac{9}{4}\frac{|L|}{\sqrt{(|L|-t)t}} \leq \frac{9}{2}\sqrt{\frac{|L|}{t}}$ in [7] follows similarly.

In conclusion Given failure probability of at most $\epsilon > 0$, recall that we execute at most $N_{\text{runs}} = \log_3(1/\epsilon)$ Grover runs. Therefore, the probability that **QSearch** succeeds satisfies

$$p^{\text{success}}_{\textbf{QSearch}} \geq \left(1 - \frac{1}{3^{N_{\text{runs}}}}\right) = 1 - \epsilon$$

as required.

A.3 Expected number of queries

In this section of the appendix, we upper bound the expected number of queries to g made by **QSearch**.

A.3.1 Classical sampling part

For fixed $|L|$ and t, the probability that a vertex drawn uniformly at random is marked is given by the fraction $f = t/|L|$. Now, if we draw at most N_{samples} classical samples uniformly at random, and then use Grover search if all N_{samples} samples turn out to be unmarked, this takes a total of

$$E_{\textbf{QSearch}} = \sum_{i=1}^{N_{\text{samples}}} f(1-f)^{i-1} i + (1-f)^{N_{\text{samples}}}(N_{\text{samples}} + c_q E_{\text{Grover}})\,. \tag{28}$$

queries to g in expectation, where E_{Grover} is the expected number of queries to the quantum oracle $\mathcal{O}_g$ made by all N_{runs} Grover runs of **QSearch** combined.

If $t = 0$, then the above expression becomes

$$E_{\textbf{QSearch}} = N_{\text{samples}} + c_q E_{\text{Grover}}\,. \tag{29}$$

If $1 < t \leq |L|$, then we can evaluate the geometric series. We have the following expression for the number of queries made by the classical sampling part

$$E(f) = \sum_{i=1}^{N_{\text{samples}}} f(1-f)^{i-1} i + (1-f)^{N_{\text{samples}}} N_{\text{samples}}$$

where $f = t/|L|$ is the fraction of marked items. The sum above can be evaluated as follows:

$$\begin{aligned} f \sum_{i=1}^{N_{\text{samples}}} (1-f)^{i-1} i &= -f \frac{\partial}{\partial f} \sum_{i=1}^{N_{\text{samples}}} (1-f)^i = f \frac{\partial}{\partial f}\left(\frac{(1-f)^{N_{\text{samples}}+1} - 1 + f}{f}\right) \\ &= -(N_{\text{samples}}+1)(1-f)^{N_{\text{samples}}} - \frac{(1-f)^{N_{\text{samples}}+1}}{f} + \frac{1}{f}. \end{aligned}$$

Adding $(1-f)^{N_{\text{samples}}} N_{\text{samples}}$ gives that $E(f)$ can be rewritten as follows:

$$\begin{aligned} E(f) &= -(1-f)^{N_{\text{samples}}} - \frac{(1-f)^{N_{\text{samples}}+1}}{f} + \frac{1}{f} \\ &= (1-f)^{N_{\text{samples}}}\left(-1 - \frac{1-f}{f}\right) + \frac{1}{f} \\ &= \frac{1}{f}\left(1 - (1-f)^{N_{\text{samples}}}\right)\,. \end{aligned} \tag{30}$$

Hence, for $1 \leq t \leq |L|$, we conclude that[20]

$$E_{\textbf{QSearch}} = \frac{1}{f}(1-(1-f)^{N_{\text{samples}}}) + (1-f)^{N_{\text{samples}}} c_q E_{\text{Grover}} \,. \tag{31}$$

A.3.2 Grover part

Next, we investigate the expected number of queries E_{Grover} to $\mathcal{O}_g$ in the Grover part of **QSearch**.

No marked items in the list If $t = 0$, then every run runs to completion without finding a marked element. In total, a single run executes at most $Q_{\max}$ queries to $\mathcal{O}_g$. Since for $t = 0$ we perform N_{runs}, the expected total number of queries to $\mathcal{O}_g$ in case of no marked items is upper bounded by

$$E_{\text{Grover}} \leq N_{\text{runs}} \alpha \sqrt{|L|} \leq 9.2 N_{\text{runs}} \sqrt{|L|} \,, \tag{32}$$

where in the last inequality we used the expression for α in Eq. (24).

Marked items in the list i.e. $t > 0$. To start with, we want to bound the number of Grover iterations executed in a single run. To do so, we first examine a single run of $\textbf{QSearch}_\infty$, i.e. without a timeout. For $k \in \mathbb{N}$, let us write p_k for the probability that the random variable X – introduced in Appendix A.2 corresponding to the number of queries to $\mathcal{O}_g$ needed for $\textbf{QSearch}_\infty$ to find a marked item – assumes the value k, i.e. the probability that a single run of $\textbf{QSearch}_\infty$ would have found a marked item using a total of k queries. Then, in terms of the probabilities $\{p_k\}_{k\in\mathbb{N}}$, we have

$$\mathbb{E}[X] = \sum_{k=1}^{\infty} k\, p_k \,.$$

A single Grover run fails exactly when $\textbf{QSearch}_\infty$ would have timed-out. Hence, the probability of a single Grover run succeeding is given by

$$q_{\text{success}} = \sum_{k=0}^{\lceil Q_{\max} \rceil - 1} p_k \,.$$

Conditioned on the outcome that step 2 finds a marked item – which happens with probability q_{success}, a single run requires in expectation

$$Q_{\text{success}} = \sum_{k=0}^{\lceil Q_{\max} \rceil - 1} k \, \frac{q_k}{q_{\text{success}}} \tag{33}$$

queries. Due to Lemma 16, we have that

$$Q_{\text{success}} \leq \mathbb{E}[X]. \tag{34}$$

[20] Interestingly, this expression is the same as the one we would have obtained using the following procedure: with probability $(1-f)^{N_{\text{samples}}}$, we do Grover search, which takes N_g^{Grover} steps, and with probability $1-(1-f)^{N_{\text{samples}}}$, we classically sample vertices, of which we need $\frac{1}{f}$. The nice thing about the implementation we use, is that our implementation of **QSearch** mimics this behaviour without knowing f in advance (meaning that we're not flipping a coin that returns heads with probability $(1-f)^{N_{\text{samples}}}$).

Lemma 16. *Let $\mathbb{P} = \{p_k\}_{k=0}^{\infty}$ be a discrete probability on $\mathbb{N}$ and let $E = \sum_{k=0}^{\infty} p_k k$ be the expectation value of sampling a number from $\mathbb{N}$ according to $\mathbb{P}$. If there is a promise that the sampled number k is less than some value $K \in \mathbb{N}$, the resulting probability distribution $\mathbb{P}'$ is renormalized by $p_K = \sum_{k=0}^{K} p_k$, that is: $\mathbb{P}' = \left\{\frac{p_k}{p_K}\right\}_{k=0}^{K}$. Now, we claim that*

$$E' = \sum_{k=0}^{K} \frac{p_k}{p_K} k \leq E,$$

i.e. the expectation value of drawing a number bellow K according to $\mathbb{P}'$ is bounded from above by the original expectation value E.

Proof. We distinguish two cases.
Case 1: $K \leq E$. In this case, we have

$$E' = \sum_{k=0}^{K} \frac{p_k}{p_K} i \leq \sum_{k=0}^{K} \frac{p_k}{p_K} K = K \leq E.$$

Case 2: $K > E$. Now we have

$$E = E' p_K + (1 - p_K) \sum_{k=K+1}^{\infty} \frac{p_k}{1 - p_K} i \geq E' p_K + (1 - p_K) K > E' p_K + (1 - p_K) E\,,$$

which implies

$$E \geq E'\,.$$

□

Similarly, conditioned on the outcome that we do not find a marked item (due to the timeout) – which happens with probability $q_{\text{fail}} = 1 - q_{\text{success}}$, the number of queries Q_{fail} in a single run can trivially be bounded by

$$Q_{\text{fail}} < Q_{\max} = \alpha\sqrt{|L|} = 9.2\sqrt{|L|}\,. \tag{35}$$

Because we execute at most N_{runs} Grover runs, the expected number of queries in its entirety is given by the following sum

$$\begin{aligned} E_{\text{Grover}} &= \sum_{j=0}^{N_{\text{runs}}-1} q_{\text{fail}}^{j}(1 - q_{\text{fail}})(jQ_{\text{fail}} + Q_{\text{success}}) + q_{\text{fail}}^{N_{\text{runs}}} N_{\text{runs}} Q_{\text{fail}} \\ &= Q_{\text{success}}(1 - q_{\text{fail}}) \sum_{j=0}^{N_{\text{runs}}-1} q_{\text{fail}}^{j} + Q_{\text{fail}} \left(q_{\text{fail}}^{N_{\text{runs}}} N_{\text{runs}} + (1 - q_{\text{fail}}) \sum_{j=1}^{N_{\text{runs}}-1} q_{\text{fail}}^{j} j \right) \end{aligned}$$

We can simplify the series above as follows

$$
\begin{aligned}
q_{\text{fail}}^{N_{\text{runs}}} N_{\text{runs}} + (1-q_{\text{fail}}) \sum_{j=1}^{N_{\text{runs}}-1} q_{\text{fail}}^{j}\, j &= q_{\text{fail}}^{N_{\text{runs}}} + q_{\text{fail}} \left(q_{\text{fail}}^{N_{\text{runs}}-1}(N_{\text{runs}}-1) + (1-q_{\text{fail}}) \sum_{j=1}^{N_{\text{runs}}-1} q_{\text{fail}}^{j-1}\, j \right) \\
&= q_{\text{fail}}^{N_{\text{runs}}} + q_{\text{fail}} \left(\frac{1}{1-q_{\text{fail}}} (1 - q_{\text{fail}}^{N_{\text{runs}}-1}) \right) \\
&= \frac{q_{\text{fail}}}{1-q_{\text{fail}}} \left(q_{\text{fail}}^{N_{\text{runs}}-1}(1-q_{\text{fail}}) + 1 - q_{\text{fail}}^{N_{\text{runs}}-1} \right) \\
&= \frac{q_{\text{fail}}}{1-q_{\text{fail}}} (1 - q_{\text{fail}}^{N_{\text{runs}}})
\end{aligned}
$$

where, going from the first to the second line we have used the derived expression for $E(f)$ in Eq. (30) with $f = 1 - q_{\text{fail}}$ and $N_{\text{samples}} = N_{\text{runs}} - 1$. We conclude that, for $t > 0$,

$$
E_{\text{Grover}} = Q_{\text{success}}(1 - q_{\text{fail}}^{N_{\text{runs}}}) + Q_{\text{fail}} q_{\text{fail}} \frac{1 - q_{\text{fail}}^{N_{\text{runs}}}}{1 - q_{\text{fail}}} \,. \tag{36}
$$

Recall that the probability q_{fail} that a single Grover run fails is given by

$$
q_{\text{fail}} = \sum_{k=\lceil Q_{\max} \rceil}^{\infty} p_k \,,
$$

which is bounded by Eq. (25) if $1 \leq t < \frac{|L|}{4}$, and Eq. (27) if $\frac{|L|}{4} < t \leq |L|$ – see Appendix A.2. However, we don't have a lower bound on q_{fail}, and therefore the best we can do with the $1 - q_{\text{fail}}^{N_{\text{runs}}}$ term is to upper bound it by 1, which is equivalent to taking the $N_{\text{runs}} \to \infty$ limit. As a consequence, for $t > 0$, our upper bound for E_{Grover} is independent of the number of Grover runs N_{runs}. The resulting upper bound for E_{Grover} is given by

$$
E_{\text{Grover}} \leq Q_{\text{success}} + q_{\text{fail}} \frac{Q_{\text{fail}}}{1 - q_{\text{fail}}} \,, \tag{37}
$$

where Q_{fail} is bounded by Eq. (35), and Q_{success} given by Eq. (33) will be bounded in the subsection below.

A.3.3 In conclusion

Given a list L with t marked items, a failure probability of ϵ, and a maximum number of classical samples N_{samples}, **QSearch**$(L, N_{\text{samples}}, \epsilon)$ executes at most $N_{\text{runs}} = \lceil \log_3(1/\epsilon) \rceil$ Grover runs.

If $t = 0$, then by Eqs. (29) and (32), the expected total number of queries to g is bounded from above by

$$
E_{\textbf{QSearch}} \leq N_{\text{samples}} + \alpha c_q \lceil \log_3(1/\epsilon) \rceil) \sqrt{|L|} \,.
$$

If $t > 0$, then by Eq. (31) we have

$$
E_{\textbf{QSearch}} = \frac{|L|}{t} \left(1 - \left(1 - \frac{t}{|L|} \right)^{N_{\text{samples}}} \right) + \left(1 - \frac{t}{|L|} \right)^{N_{\text{samples}}} c_q E_{\text{Grover}}
$$

By Eq. (37), E_{Grover} satisfies the bounds below

$$\begin{aligned} E_{\text{Grover}} &\leq Q_{\text{success}} + q_{\text{fail}} \frac{Q_{\text{fail}}}{1 - q_{\text{fail}}} \\ &\leq Q_{\text{success}} + \frac{E[X]}{\alpha\sqrt{|L|}} \frac{Q_{\text{fail}}}{1 - \frac{E[X]}{\alpha\sqrt{|L|}}} \,. \end{aligned}$$

In the second line we have used Eq (22). Now, for Q_{fail}, we have the upper bound from Eq. (35): $Q_{\text{fail}} \leq \alpha\sqrt{|L|}$. Moreover, to bound Q_{success}, we can use Eq. (34), which says that $Q_{\text{success}} \leq E[X]$. Hence, we obtain

$$E_{\text{Grover}} \leq E[X] \left(1 + \frac{1}{1 - \frac{E[X]}{\alpha\sqrt{|L|}}} \right) ,$$

Depending on the number of marked items t, we can bound $E[X]$ as follows.

- If $1 \leq t \leq \frac{|L|}{4}$, then by Eq. (20),

  $$E[X] \leq \frac{9}{4} \frac{|L|}{\sqrt{(|L| - t)t}} + \left\lceil \log_{\frac{6}{5}} \left(\frac{|L|}{2\sqrt{(|L| - t)t}} \right) \right\rceil - 3 \leq \frac{\alpha\sqrt{|L|}}{3\sqrt{t}}.$$

  The rightmost inequality follows from the analysis leading up to Eq. (25).

- In case $\frac{|L|}{4} < t \leq |L|$, by Eq. (20) we have

  $$E[X] \leq 2.0344 \,.$$

In the above formulas, by Eq. (24),

$$\alpha = 9.2 \geq \max_{x \geq 1} \left(\frac{9\sqrt{3}}{2} + 3 \frac{\log_\lambda \left(\frac{\sqrt{x}}{3} \right) - 2}{\sqrt{x}} \right) .$$

The expressions obtained for the expected number of queries to g are presented more concisely in Lemma 4.

A.4 Worst-case behaviour of **QSearch**$_{\text{Zalka}}$

The two steps of this algorithm are given in Section 2.2. Here we analyse the worst-case complexity of that implementation. For the first step, recall that (see Lemma 2) when there are t marked items (and we know t), then exact Grover search[21] can find and return one *with*

[21] We note that exact Grover search is somewhat unrealistic, since it requires arbitrarily precise rotations to be performed in each Grover step. In general this will not be possible, and the best we can hope for is some very close approximation using a sequence of gates taken from some universal gate set. This approximation can be made inverse-exponentially close to the correct rotations with only a polynomial overhead (in terms of the number of gates required), which in particular does not contribute to the query complexity of the algorithm. This will mean that 'exact' Grover search will actually fail with some small probability, but that this probability can be made small enough to be negligible for our purposes (e.g. we can simply ensure that the probability of it failing is smaller than ϵ/t_0 without incurring any extra queries). With this in mind, we ignore such considerations, since there will always be issues of approximation and error arising from the physical implementations of quantum algorithms, and we view such issues as being on the same level as overheads from error correction, which as we have already discussed we will omit from our analysis.

certainty, using precisely $\left\lceil \frac{\pi}{4}\sqrt{\frac{N}{t}-\frac{1}{2}} \right\rceil + 1$ Grover iterations [16].[22] Therefore, step 1 requires at most

$$\begin{aligned}\sum_{t=1}^{t_0}\left(\left\lceil \frac{\pi}{4}\sqrt{\frac{N}{t}}-\frac{1}{2}\right\rceil+1\right) \leq \sum_{t=1}^{t_0}\left(\frac{\pi}{4}\sqrt{\frac{|L|}{t}}+1\right) &= t_0+\frac{\pi}{4}\sqrt{|L|}\sum_{t=1}^{t_0}\frac{1}{\sqrt{t}} \\ &\leq t_0+\frac{\pi}{4}\sqrt{|L|}\int_0^{t_0}\frac{1}{\sqrt{t}}dt \\ &= t_0+\frac{\pi}{2}\sqrt{|L|t_0}\end{aligned}$$

queries to the quantum oracle $\mathcal{O}_g$ (recalling that every Grover iteration takes a single query, and every Grover run requires one at the end to check whether a marked item was found or not). For the second step, clearly there will be at most

$$2t_0\left\lceil \frac{\pi}{4}\sqrt{\frac{|L|}{t_0}}\right\rceil + 2t_0 \leq \frac{\pi}{2}\sqrt{|L|t_0}+4t_0$$

queries (though the probability of doing this many is quite small), and it remains to set the value of t_0. Suppose that whenever we run Grover search with a number of iterations in the range $[0, \lceil \frac{\pi}{4}\sqrt{\frac{|L|}{t_0}}\rceil]$, the probability of detecting a marked item is at least p. Then the probability that this step of the algorithm fails will be no more than $(1-p)^{2t_0}$. If we require that $(1-p)^{2t_0} \leq \epsilon$, then it is sufficient to choose

$$t_0 = \left\lceil \frac{\ln(\epsilon)}{2\ln(1-p)}\right\rceil$$

(where we round up to ensure that t_0 is an integer). Then we can combine the queries from each stage and rewrite the total number made by the algorithm as

$$5\left\lceil \frac{\ln\epsilon}{2\ln(1-p)}\right\rceil + \pi\sqrt{|L|}\sqrt{\left\lceil \frac{\ln\epsilon}{2\ln(1-p)}\right\rceil}.$$

It remains to lower bound the success probability p for different values of t. We can first use a bound derived in [7], which says that $p \geq 1/4$ provided $\left\lceil \frac{\pi}{4}\sqrt{\frac{|L|}{t_0}}\right\rceil \geq \frac{|L|}{2\sqrt{t}\sqrt{|L|-t}}$. Using the fact that $t > t_0$, and further assuming that $t \leq \frac{(a-1)|L|}{a}$ for some $a > 1$ to be chosen, we have

$$\frac{|L|}{2\sqrt{t}\sqrt{|L|-t}} \leq \frac{1}{2}\sqrt{\frac{a|L|}{t_0}}$$

and therefore we can ensure that $\left\lceil \frac{\pi}{4}\sqrt{\frac{|L|}{t_0}}\right\rceil \geq \frac{|L|}{2\sqrt{t}\sqrt{|L|-t}}$ by choosing $a = \frac{\pi^2}{4}$, in which case we obtain $p \geq 1/4$.

Now we can consider the case that $t > \frac{(a-1)|L|}{a}$. In this case, we are still very likely to find a marked item even after applying some Grover iterations that will, in general, rotate the

[22] The notation $\lceil x \rfloor$ represents the closest integer to x.

state away from the marked subspace, and as we show in Lemma 14 we have $p \geq 1/4$ when $t > \frac{\pi^2/4-1}{\pi^2/4} > \frac{|L|}{4}$, and so for both cases we have $p \geq 1/4$. Plugging this lower bound on p into the expression for the total number of Grover iterations, and taking into account that a single query to $\mathcal{O}_g$ corresponds to c_q queries to g, we see that the total number of queries to g made by the algorithm is at most

$$W_{\mathbf{QSearch}_{\text{Zalka}}}(|L|,\epsilon) := c_q \left(5\left\lceil \frac{\ln(1/\epsilon)}{2\ln(4/3)} \right\rceil + \pi\sqrt{|L|}\sqrt{\left\lceil \frac{\ln(1/\epsilon)}{2\ln(4/3)} \right\rceil}\right) . \tag{38}$$

B Detailed analysis of **QMax**

In this section we compute the expected number of queries to g made by $\mathbf{QMax}_\infty$, and we give further upper bounds to the obtained expression for the expected number of queries.

B.1 Expected number of queries

Based on the the proof idea of [1], we provide a more accurate proof and expression of the expected number of queries made by $\mathbf{QMax}_\infty$ when searching the list L with t marked items, as stated in Lemma 6 (restated below for convenience).

Lemma 6. *[Expected complexity of $\boldsymbol{QMax}_\infty$] Let L be a list of $|L|$ items. Then, the expected number of queries to any of the f_i (as defined as in Eq. 7) required for $\boldsymbol{QMax}_\infty$ to find the maximum of L is upper bounded by*

$$E_{\boldsymbol{QMax}_\infty}(|L|) \leq c_q \sum_{t=1}^{|L|-1} \frac{F(|L|,t)}{t+1} , \tag{9}$$

where $F(|L|,t)$ is defined by Eq. (3). Here, c_q is the number of queries to f_i required to implement the oracle $\mathcal{O}_{f_i}$ (which we assume to be the same for all i).

Proof. Let us use the shorthand notation

$$Q(t) = E^{\text{Quantum}}_{\mathbf{QSearch}_\infty}(|L|,t)$$

for the expected number of queries made by $\mathbf{QSearch}_\infty$ to the quantum oracle when searching a list L with t marked items (suppressing the L dependence for notational convenience). Note that by Lemma 15,

$$Q(t) \leq F(|L|,t)$$

where $F(|L|,t)$ is given by Eq. (3). Moreover, let $E(t)$ denote the expected number of queries to the quantum oracles $\mathcal{O}_{f_i}$ for finding the maximum when t items are marked: i.e. the expected number of queries to find the maximum given that y is set to the $t+1$-th item of L when ordered according to R in descending order. We first compute the expected number of queries to the quantum oracles $\mathcal{O}_{f_i}$, and then include the factor of c_q in the end[23].

We have the following recursion relation for $E(t)$:

$$E(t) = \frac{1}{t}\big(E(t-1) + E(t-2) + \ldots + E(1) + E(0)\big) + Q(t) , \tag{39}$$

[23] Since each query to $\mathcal{O}_g$ corresponds to c_q queries to g.

(because, after applying $\mathbf{QSearch}_\infty$, with equal probability we find one of the t marked items in L with a larger value for R than the current index y). Note that $E(0) = 0$.

Using Eq. (39) for t and $t-1$:

$$tE(t) = \sum_{u=1}^{t-1} E(u) + tQ(t)$$
$$(t-1)E(t-1) = \sum_{u=1}^{t-2} E(u) + (t-1)Q(t-1)$$

and subtracting the bottom equation from the top equation and then dividing by t yields

$$E(t) = E(t-1) + Q(t) - \frac{t-1}{t}Q(t-1)\,. \tag{40}$$

Since the above equation holds for every t, we can use the equation for $t-1$ and plug it into Eq. (40), and then do the same for $t-2$, etc., up to $t = 2$. We then obtain

$$E(t) = E(1) + \sum_{u=2}^{t} \left(Q(u) - \frac{u-1}{u}Q(u-1) \right) .$$

We next rewrite[24] the sums above as follows:

$$\begin{aligned} E(t) &= E(1) + \sum_{u=2}^{t} Q(t) - \sum_{u=1}^{t-1} \frac{u}{u+1}Q(u) \\ &= E(1) + Q(t) + \sum_{u=2}^{t-1} Q(t) \left(1 - \frac{u}{u+1}\right) - \frac{1}{2}Q(1) \\ &= Q(t) + \sum_{u=2}^{t-1} \frac{Q(u)}{u+1} + \frac{1}{2}Q(1) \\ &= Q(t) + \sum_{u=1}^{t-1} \frac{Q(u)}{u+1}\,, \end{aligned} \tag{41}$$

where, when going to the third line, we have used that $E(1) = Q(1)$.

Now, since, at initialization, $\mathbf{QMax}_\infty$ chooses an index y uniformly at random, the expected number of queries of $\mathbf{QMax}_\infty$ to the quantum oracles is given by

$$\frac{1}{|L|} \sum_{t=1}^{|L|-1} E(t)\,.$$

[24] This is where the proof of [1] becomes imprecise. The authors use upper bounds for $Q(u)$ and $Q(u-1)$ in order to upper bound $E(t)$. However, in order to obtain an upper bound for $E(t)$, a *lower* bound for the $Q(u-1)$ terms should be used, because they come with a minus sign. The correct way to continue the proof is to postpone using upper bounds for $Q(u)$ until after rewriting the sum.

As before, $E(0) = 0$ and needs not to be included in the sum. Using Eq. (41), we get

$$\begin{aligned}\frac{1}{|L|}\sum_{t=1}^{|L|-1} E(t) &= \frac{1}{|L|}\left(\sum_{t=1}^{|L|-1} Q(t) + \sum_{t=1}^{|L|-1}\sum_{u=1}^{t-1}\frac{Q(u)}{u+1}\right)\\ &= \frac{1}{|L|}\sum_{s=1}^{|L|-1} Q(s)\left(1+\frac{|L|-1-s}{s+1}\right)\\ &= \sum_{s=1}^{|L|-1}\frac{Q(s)}{s+1}\,,\end{aligned}$$

where the expression is the second line can be obtained by collecting all terms $Q(s)$ together for every $s \in \{1, 2, \ldots, |L|-1\}$. Including the factor c_q, and *only now* using the fact that $Q(s) \leq F(|L|, s)$ we obtain our result. □

B.2 Upper bounds to the expected number of queries

Next, we will upper bound the series that upper bounds the expected number of queries to g made by $\mathbf{QMax}_\infty$ by a simpler expression. By Lem. 6 and Eq. (3), we have

$$E_{\mathbf{QMax}_\infty}(|L|) \leq c_q\left(\sum_{s=1}^{\lceil |L|/4\rceil -1}\frac{F(L,s)}{\sqrt{s}(s+1)} + 2.0344\sum_{s=\lceil |L|/4\rceil}^{|L|-1}\frac{1}{s+1}\right). \tag{42}$$

with

$$F(L,s) \leq \frac{9}{4}\frac{|L|}{\sqrt{(|L|-t)t}} + \left\lceil \log_{\frac{6}{5}}\left(\frac{|L|}{2\sqrt{(|L|-t)t}}\right)\right\rceil - 3 \leq \frac{9.2\sqrt{|L|}}{3\sqrt{t}}$$

for $s \leq \lceil |L|/4\rceil - 1$.

Loose upper bound We can upper bound the sums above by integrals. For the second term, we have

$$\sum_{s=\lceil |L|/4\rceil}^{|L|-1}\frac{1}{s+1} \leq \int_{\lceil |L|/4\rceil -1}^{|L|-1} ds\frac{1}{s+1} = \int_{\lceil |L|/4\rceil}^{|L|} dx\frac{1}{x} = \ln(|L|) - \ln(\lceil |L|/4\rceil) \leq \ln(4)\,.$$

For the first term, using $\frac{9.2\sqrt{|L|}}{2\sqrt{s}}$ as an upper bound for $F(L,s)$, we get

$$\sum_{s=1}^{\lceil |L|/4\rceil -1}\frac{1}{\sqrt{s}(s+1)} \leq \frac{1}{2} + \int_1^{\lceil |L|/4\rceil -1} ds\,\frac{1}{\sqrt{s}(s+1} = \frac{1}{2} + 2\arctan(\sqrt{\lceil |L|/4\rceil - 1}) - \frac{\pi}{2}$$

Hence,

$$\begin{aligned}E_{\mathbf{QMax}_\infty}(|L|) &\leq c_q\left(\frac{9.2\sqrt{|L|}}{3}\left(\frac{1-\pi}{2} + 2\arctan(\sqrt{\lceil |L|/4\rceil - 1})\right) + 2.0344\ln(4)\right)\\ &\leq c_q\left(\frac{9.2\sqrt{|L|}}{3}\frac{1+\pi}{2} + 2.0344\ln(4)\right)\\ &\leq c_q\left(6.3505\sqrt{|L|} + 2.8203\right)\,.\end{aligned}$$

Tighter bound We can also use

$$F(L,t) \leq \frac{3\sqrt{3}}{2}\sqrt{\frac{|L|}{t}} + \log_{\frac{6}{5}}\left(\sqrt{\frac{|L|}{3t}}\right) - 2$$

for the $1 \leq t < |L|/4$ regime – obtained from the tighter upper bound for $F(L,t)$ above combined with Eq. (21). In this case, the first term in Eq. (42) can be upper bounded by

$$\sum_{s=1}^{\lceil |L|/4\rceil - 1} \frac{F(L,s)}{s+1} \leq \frac{3\sqrt{3|L|}}{2} \sum_{s=1}^{\lceil |L|/4\rceil - 1} \frac{1}{\sqrt{s}(s+1)} + \left(\frac{1}{2}\log_{\frac{6}{5}}(|L|/3) - 2\right) \sum_{s=1}^{\lceil |L|/4\rceil - 1} \frac{1}{s+1} - \frac{1}{2} \sum_{s=1}^{\lceil |L|/4\rceil - 1} \frac{\log_{\frac{6}{5}}(s)}{s+1}\,.$$

The first term above can be bounded by

$$\frac{3\sqrt{3|L|}}{2}\left(\frac{1-\pi}{2} + 2\arctan\left(\sqrt{\lceil |L|/4\rceil - 1}\right)\right),$$

as before, and the second by

$$\left(\frac{1}{2}\log_{\frac{6}{5}}(|L|/3) - 2\right)\ln(|L|/4)\,.$$

For the third term above, note that the function $\frac{\ln(s)}{s+1}$ is monotonically decreasing for $s \geq 4$. Hence, assuming $\lceil |L|/4\rceil - 1 \geq 4$, we have

$$\begin{aligned}\frac{1}{2\ln(6/5)} \sum_{s=1}^{\lceil |L|/4\rceil - 1} \frac{\ln(s)}{s+1} &\leq \frac{1}{2\ln(6/5)} \sum_{s=1}^{4} \frac{\ln(s)}{s+1} + \int_4^{\lceil |L|/4\rceil - 1} ds \frac{\ln(s)}{s+1} \\ &\leq 2.5279 + \frac{\mathrm{Li}_2(-\lceil |L|/4\rceil + 1) + \ln(\lceil |L|/4\rceil - 1)\ln(\lceil |L|/4\rceil)}{2\ln(6/5)}\end{aligned}$$

where Li_2 is Spence's function, or dilogarithm.

Collecting all terms together, we get

$$\begin{aligned}\sum_{s=1}^{\lceil |L|/4\rceil - 1} \frac{Q(s)}{s+1} \leq{}& \frac{3\sqrt{3|L|}}{2}\left(\frac{1-\pi}{2} + 2\arctan\left(\sqrt{|L|/4}\right)\right) \\ &+ \frac{\ln(|L|/4)}{2\ln(6/5)}\left(\ln(|L|/3) + \ln(|L|/4+1)\right) \\ &- 2\ln(|L|/4) + 2.5279 + \frac{\mathrm{Li}_2(-\lceil |L|/4\rceil + 1)}{2\ln(6/5)}\,.\end{aligned}$$

Using $\arctan(x) \leq \frac{\pi}{2}$, the expected number of queries to g of $\mathbf{QMax}_\infty$ is bounded by

$$\begin{aligned}E_{\mathbf{QMax}_\infty}(|L|) \leq c_q \Bigg[&\frac{3\sqrt{3}(1+\pi)}{4}\sqrt{|L|} + \frac{\ln(|L|/4)}{2\ln(6/5)}\Big(\ln(|L|/3) + \ln(|L|/4+1)\Big) - 2\ln(|L|/4) \\ &+ 5.3482 + \frac{\mathrm{Li}_2(-\lceil |L|/4\rceil + 1)}{2\ln(6/5)}\Bigg]\end{aligned} \tag{43}$$

which has a leading term that grows as

$$c_q \frac{3\sqrt{3}(1+\pi)}{4}\sqrt{|L|} \leq c_q 5.3801\sqrt{|L|}\,.$$

C Estimators for the expected number of queries for **QSearch**

In this section, we provide additional details for our derivation of the estimator in Section 3.1 that overestimates expected number of queries for **QSearch** by sampling items from L uniformly at random and counting after how many samples we find a marked item. We start by proving Lemma 7 and Lemma 8, then construct $E^{\text{estimator}}_{\text{Grover}}$, and finally construct an estimator for the expected number of queries $E_{\textbf{QSearch}}$ for **QSearch**.

C.1 Proof of Lemma 7

Here we prove Lemma 7, restated below for convenience.

Lemma 7. *For a random variable X geometrically distributed with parameter f, there exists a constant $d_1 = 4/\pi \approx 1.273$, such that*

$$\sqrt{\mathbb{E}[X]} \leq \mathbb{E}[\sqrt{d_1 X}]$$

for all $f \in (0, 1]$.

Proof. The expectation value of $\mathbb{E}[\sqrt{d_1 X}]$ is given by

$$\mathbb{E}[\sqrt{d_1 X}] = \sum_{i=1}^{\infty} \sqrt{d_1 i} f(1-f)^{i-1} = \sqrt{d_1}\frac{f\mathrm{Li}_{-\frac{1}{2}}(1-f)}{1-f} =: \sqrt{d_1}h(f),$$

where $\mathrm{Li}_s(z)$ is a polylogarithmic function known as Jonquière's function, given by

$$\mathrm{Li}_s(z) = \sum_{k-1}^{\infty} \frac{z^k}{k^s}.$$

We prove this corollary in three steps: 1) we investigate the limiting behaviour of the ratio of $\sqrt{d_1}h(f)/\sqrt{1/f}$ at the domain boundaries, and show that there exists a constant $d_1 = \frac{4}{\pi}$ such that at both limits the ratio is at least one; 2) we show that this ratio is non-decreasing on this domain; 3) we deduct the implied upper bound and relative errors. We obtain the

limiting behaviour when $f \to 0^+$ as follows:

$$\begin{aligned}
\lim_{f\to 0^+} \frac{\sqrt{d_1}h(f)}{\sqrt{\frac{1}{f}}} &= \lim_{f\to 0^+} \frac{\sqrt{d_1} f \mathrm{Li}_{-\frac{1}{2}}(1-f)}{(1-f)\sqrt{\frac{1}{f}}} \\
&= \sqrt{d_1} \lim_{f\to 0^+} \frac{\mathrm{Li}_{-\frac{1}{2}}(1-f)}{f^{-3/2}(1-f)} \\
&= \sqrt{d_1} \lim_{f\to 0^+} \left[\frac{1}{1-f}\right] \lim_{f\to 0^+} \left[\frac{\mathrm{Li}_{-\frac{1}{2}}(1-f)}{f^{-3/2}}\right] && \text{(Product rule)} \\
&= \sqrt{d_1} \lim_{f\to 0^+} \frac{\frac{\sqrt{\pi}}{2} f^{-3/2} + \mathcal{O}(f^{-1/2})}{f^{-3/2}} && \text{(Series expansion)} \\
&= \sqrt{d_1}\frac{\sqrt{\pi}}{2}.
\end{aligned}$$

Similarly, for $f \to 1$ we get

$$\begin{aligned}
\lim_{f\to 1} \sqrt{d_1}\frac{h(f)}{\sqrt{\frac{1}{f}}} &= \lim_{f\to 1} \sqrt{d_1}\frac{f\mathrm{Li}_{-\frac{1}{2}}(1-f)}{(1-f)\sqrt{\frac{1}{f}}} \\
&= \sqrt{d_1} \lim_{f\to 1} \frac{\mathrm{Li}_{-\frac{1}{2}}(1-f)}{f^{-3/2}(1-f)} \\
&= \sqrt{d_1} \lim_{f\to 1} \left[\frac{1}{f^{-3/2}}\right] \lim_{f\to 1} \left[\frac{\mathrm{Li}_{-\frac{1}{2}}(1-f)}{1-f}\right] && \text{(Product rule)} \\
&= \sqrt{d_1} \lim_{f\to 1} \frac{(1-f) + \mathcal{O}((1-f)^2)}{1-f} && \text{(Series expansion)} \\
&= \sqrt{d_1}.
\end{aligned}$$

Therefore, if we pick $d_1 = (2/\sqrt{\pi})^2 = 4/\pi$, we have that the ratio is at both limits at least one. We will now establish the non-decreasing property of this ratio, already evaluated with our proposed $\sqrt{d_1} = 2/\sqrt{\pi}$, by invoking the first derivative test. Note that

$$\frac{\partial}{\partial f}\frac{2}{\sqrt{\pi}}\frac{h(f)}{\sqrt{\frac{1}{f}}} = -\frac{2f\mathrm{Li}_{-3/2}(1-f) + (f-3)\mathrm{Li}_{-1/2}(1-f)}{\sqrt{\pi}\sqrt{\frac{1}{f}}(1-f)^2},$$

for which the denominator $\sqrt{\pi}\sqrt{\frac{1}{f}}(1-f)^2 \geq 0$ for all $f \in (0,1]$, and the numerator has exactly one root at $f = 1$. Since for any other $\tilde{f} \in (0,1]$ taken left of $f = 1$ the derivative of the ratio is larger than zero, we must have that the ratio is a non-decreasing function. Therefore, we have that $\sqrt{d_1}h(f) \geq \sqrt{1/f}$ for all $f \in (0,1]$.

□

C.2 Proof of Lemma 8

Lemma 8. *For a random variable X geometrically distributed with parameter f, there exists a constant $d_2 = e^\gamma \approx 1.781$, where γ is the Euler–Mascheroni constant, such that*

$$\log(\mathbb{E}[X]) \leq \mathbb{E}[\log(d_2 X)]$$

for all $f \in (0,1]$.

Proof. For random variable X geometrically distributed with parameter f, we have that $\mathbb{E}[X] = \frac{1}{f}$, and

$$\mathbb{E}[\log X] = \sum_{i=1}^{\infty} \log(i) f(1-f)^{i-1}.$$

We will show that

$$\log(\mathbb{E}[X]) - \mathbb{E}[\log X] \leq \gamma \tag{44}$$

from which the statement of the lemma follows:

$$\log(\mathbb{E}[X]) \leq \log(e^{\gamma}) + \mathbb{E}[\log X] = \mathbb{E}[\log(e^{\gamma} X)]\,.$$

In order to prove Eq. (44), let us define $q := 1 - f \in [0,1)$. Now, we have

$$\begin{aligned}
\log(\mathbb{E}[X]) - \mathbb{E}[\log X] &= -\log(1-q) - \sum_{i=1}^{\infty} \log(i)(1-q)q^{i-1} \\
&= -\log(1-q) - \sum_{i=2}^{\infty} \log(i) q^{i-1} + \sum_{i=1}^{\infty} \log(i) q^{i} \\
&= -\log(1-q) - \sum_{i=1}^{\infty} \log(i+1) q^{i} + \sum_{i=1}^{\infty} \log(i) q^{i} \\
&= -\log(1-q) - \sum_{i=1}^{\infty} \log\left(\frac{i+1}{i}\right) q^{i} \\
&= \sum_{i=1}^{\infty} \frac{q^i}{i} - \sum_{i=1}^{\infty} \log\left(\frac{i+1}{i}\right) q^{i} && \text{[Series expansion]} \\
&= \sum_{i=1}^{\infty} \left[\frac{1}{i} - \log\left(\frac{i+1}{i}\right)\right] q^{i}
\end{aligned}$$

Next, we investigate the above series without the q^i's, which turns out to be convergent. Indeed, for $n \in \mathbb{N}$ we have

$$\begin{aligned}
\sum_{i=1}^{n} \left[-\log\left(\frac{1+i}{i}\right) + \frac{1}{i}\right] &= -\log\left(\prod_{i=1}^{n} \frac{1+i}{i}\right) + \sum_{i=1}^{n} \frac{1}{i} \\
&= -\log(n+1) + \sum_{i=1}^{n+1} \frac{1}{i} - \frac{1}{n+1}\,.
\end{aligned}$$

Taking the limit $n \to \infty$ yields

$$\sum_{i=1}^{\infty} \left[-\log\left(\frac{1+i}{i}\right) + \frac{1}{i}\right] = \lim_{n\to\infty} \left[-\log(n+1) + \sum_{i=1}^{n+1} \frac{1}{i}\right] - \lim_{n\to\infty} \left[\frac{1}{n+1}\right] = \gamma \approx 0.5772$$

by definition of the Euler-Mascheroni constant.

Moreover, since

$$\frac{1}{i} - \log\left(\frac{i+1}{i}\right) = \frac{1}{i} - \log\left(1 + \frac{1}{i}\right) \geq 0 \text{ for all } i > 0,$$

and $q^i \in [0,1)$, the following must hold

$$\sum_{i=1}^{\infty} \left[\frac{1}{i} - \log\left(\frac{i+1}{i}\right)\right] q^i \leq \gamma \text{ for all } 0 \leq q < 1\,,$$

and therefore so does Eq. (44).

□

C.3 Estimator for E_{Grover} for all $t \geq 1$

We next construct an estimator that overestimates E_{Grover}. To do so, we will first construct a single function of the form

$$G(f) = c_0 + c_1 \frac{1}{f} + c_2 \sqrt{\frac{1}{f}} + c_3 \log_{\frac{6}{5}}\left(\frac{1}{f}\right)\,, \tag{45}$$

where $f = t/|L|$, that upper bounds $E_{\mathrm{Grover}}(|L|, t)$ on the entire domain $[1, |L|]$ of t — except for the case of $t = 0$, which will be dealt with separately in Section 3.1. Afterwards, we will use Lemmas 7 and 8 to deal with the issue of the concavity of the square root and the logarithm appearing in the expression for G.

Let us start with the regime $1 \leq t \leq |L|/4$. From Eq. (2), using Eq. (21) with $f = \frac{t}{|L|}$, we have

$$\begin{aligned} E_{\mathrm{Grover}}(|L|, t) &\leq \left(\frac{3}{2}\sqrt{\frac{3}{f}} + \log_{\frac{6}{5}}\left(\sqrt{\frac{1}{3f}}\right) - 2\right)\left(1 + \frac{1}{1 - \frac{1}{3\sqrt{f|L|}}}\right) \\ &\leq \left(\frac{3}{2}\sqrt{\frac{3}{f}} + \log_{\frac{6}{5}}\left(\sqrt{\frac{1}{f}}\right) + \log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right) - 2\right)\left(2 + \frac{1}{2\sqrt{f|L|}}\right)\,, \end{aligned} \tag{46}$$

where we have bounded the term $\frac{1}{1-\frac{1}{3\sqrt{f|L|}}}$ by $1 + \frac{1}{2\sqrt{f|L|}}$, which holds since (defining $x := \frac{1}{3\sqrt{f|L|}}$)

$$\frac{1}{1 - \frac{1}{3\sqrt{f|L|}}} = \frac{1}{1-x} = 1 + \frac{x}{1-x} \leq 1 + \frac{3}{2}x = 1 + \frac{1}{2\sqrt{f|L|}}, \tag{47}$$

when $0 \leq x \leq 1/3$. Plugging Eq. (47) into Eq. (46) and expanding all terms we obtain

$E_{\text{Grover}}(|L|, t) \leq G(f)$ by choosing the c_i's as follows.[25]

$$
\begin{aligned}
c_0 &= 2\log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right) - 4 \leq -10.0256 \\
c_1 &= \frac{3\sqrt{3}}{4\sqrt{|L|}} \leq \frac{1.2991}{\sqrt{|L|}} \\
c_2 &= 3\sqrt{3} + \frac{1}{2\sqrt{|L|}}\left(\log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right) - 2\right) \leq 5.1962 - \frac{2.5064}{\sqrt{|L|}} \\
c_3 &= \frac{5}{4}\,.
\end{aligned}
$$

Above, we have absorbed the product of the $\log_{\frac{6}{5}}(\sqrt{1/f})$ and the $1/(2\sqrt{f|L|})$ into c_3, by upper bounding $1/(2\sqrt{f|L|}) \leq 1/2$.

The function $G(f)$ with the constants as given above gives an expression that upper bounds E_{Grover} on the domain $1 \leq t < \frac{|L|}{4}$ (or $4 \leq \frac{1}{f} \leq |L|$), but it does not upper bound E_{Grover} for $\frac{|L|}{4} \leq t \leq |L|$ (or $1 \leq \frac{1}{f} \leq 4$). In order to obtain a single expression that upper bounds E_{Grover} for all $1 \leq t \leq |L|$, or equivalently $1 \leq \frac{1}{f} \leq |L|$, we can take the expression from Eq. (45) and add an unknown constant[26] to c_0 to ensure that the resulting expression also upper bounds E_{Grover} for $\frac{|L|}{4} \leq t \leq |L|$, i.e. $1 \leq \frac{1}{f} \leq 4$. The latter is given by Eq. (2):

$$
2.0344\left(1 + \frac{1}{1 - \frac{2.0344}{\alpha\sqrt{|L|}}}\right)\,. \tag{48}
$$

To obtain a bound that holds for all $1 \leq \frac{1}{f} \leq |L|$, note that Eq. (45) is monotonically increasing in $1/f$. Therefore, all we need is that Eq. (45) upper bounds the expression in Eq. (48) for $\frac{1}{f} = 1$, that is, we require

$$
2.0344\left(1 + \frac{1}{1 - \frac{2.0344}{\alpha\sqrt{|L|}}}\right) \leq c_0 + c_1 + c_2\,, \tag{49}
$$

with

$$
\begin{aligned}
c_0 &= 2\log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right) - 4 + A + \frac{B}{2\sqrt{L}} \\
c_1 &= \frac{3\sqrt{3}}{4\sqrt{|L|}} \\
c_2 &= 3\sqrt{3} + \frac{1}{2\sqrt{|L|}}\left(\log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right) - 2\right)
\end{aligned} \tag{50}
$$

[25] Note that, despite there being a term linear in $\frac{1}{f}$, there number of iterations scales as $\sqrt{|L|}$: indeed, the constant c_1 multiplying the linear term contains a factor of $\frac{1}{\sqrt{|L|}}$, and since $1/f \leq |L|$, we have that $c_1\frac{1}{f} = O(\sqrt{|L|})$.

[26] We can also alter the other coefficients, but that will give a much worse upper bound for the small t regime.

where A and B will be chosen to make Eq. (49) hold.

Using the same reasoning as for Eq. (47) with $x = \frac{2.0344}{\alpha\sqrt{|L|}}$ and $0 \leq x \leq \frac{2.0344}{\alpha}$, we have

$$\begin{aligned} 2.0344\left(1+\frac{1}{1-\frac{2.0344}{\alpha\sqrt{|L|}}}\right) &\leq 2.0344\left(2+\frac{1}{1-\frac{2.0344}{\alpha}}\frac{2.0344}{\alpha\sqrt{|L|}}\right) \\ &= 4.0688+\frac{1}{\sqrt{|L|}}\frac{2.0344^2}{\alpha-2.0344} \end{aligned}$$

In order to make the right-hand side of the above expression less than or equal to $c_0+c_1+c_2$, we can compare the constant terms and the terms multiplying $\frac{1}{\sqrt{|L}}$ on both sides separately. For the constant terms, we require

$$4.0688 \leq 2\log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right)-4+A+3\sqrt{3}$$

which implies that

$$A = 8.8984 \geq 4.0688-2\log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right)+4-3\sqrt{3}$$

works. For the terms multiplying $\frac{1}{\sqrt{|L|}}$, we need

$$\frac{2.0344^2}{\alpha-2.0344} \leq \frac{1}{2}\left(\frac{3\sqrt{3}}{2}+\log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right)-2+B\right)$$

which means we can take

$$B = 3.5700 \geq 2\,\frac{2.0344^2}{\alpha-2.0344}-\left(\frac{3\sqrt{3}}{2}+\log_{\frac{6}{5}}\left(\frac{1}{\sqrt{3}}\right)-2\right).$$

Taking everything together, we obtain the following upper bound

$$E_{\text{Grover}}(|L|,t) \leq -1.1272+\frac{1.7850}{\sqrt{|L|}}+\frac{1.2991}{\sqrt{|L|}}\frac{1}{f}+\left(5.1962-\frac{2.5064}{\sqrt{|L|}}\right)\sqrt{\frac{1}{f}}+\frac{5}{4}\log_{\frac{6}{5}}\left(\frac{1}{f}\right).$$

Hence, the following estimator

$$\tilde{E}^{\text{estimator}}_{\text{Grover}}(l) := -1.1272+\frac{1.7850}{\sqrt{|L|}}+\frac{1.2991}{\sqrt{|L|}}l+\left(5.1962-\frac{2.5064}{\sqrt{|L|}}\right)\sqrt{l}+\frac{5}{4}\log_{\frac{6}{5}}(l)$$

satisfies

$$\tilde{E}^{\text{estimator}}_{\text{Grover}}(\mathbb{E}[l]) = \tilde{E}^{\text{estimator}}_{\text{Grover}}(1/f) \geq E_{\text{Grover}}(|L|,t)\,.$$

for all $1 \leq t \leq |L|$.

Next, we want to construct an estimator $E^{\text{estimator}}_{\text{Grover}}(l)$ that satisfies

$$\mathbb{E}[E^{\text{estimator}}_{\text{Grover}}(l)] \geq \tilde{E}^{\text{estimator}}_{\text{Grover}}(\mathbb{E}[l])\,.$$

Lemma 7 implies that, if we multiply our sample average[27] of $\sqrt{l}$ by $2/\sqrt{\pi} = \sqrt{d_1}$, we obtain an an estimator that in expectation upper bounds $\sqrt{1/f}$. Similarly, from Lemma 8 we gather that, if we multiply l by e^γ in the argument of the logarithm, we obtain an estimator that in expectation upper bounds $\log_{\frac{6}{5}}\left(\frac{1}{f}\right)$. Consequently, the following estimator

$$E_{\text{Grover}}^{\text{estimator}}(l) := -1.1272 + \frac{1.7850}{\sqrt{|L|}} + \frac{1.2991}{\sqrt{|L|}}l + \left(5.1962 - \frac{2.5064}{\sqrt{|L|}}\right)\frac{2\sqrt{l}}{\sqrt{\pi}} + \frac{5}{4}\log_{\frac{6}{5}}\left(e^\gamma l\right)$$

upper bounds E_{Grover} in expectation for all $1 \leq t \leq |L|$ (or $1 \leq \frac{1}{f} \leq |L|$):

$$\mathbb{E}[E_{\text{Grover}}^{\text{estimator}}(l)] \geq E_{\text{Grover}}(|L|, t)\,,$$

where the expectation is taken over the geometric distribution $l \sim \text{Geo}(f)$, with $f = t/|L|$.

C.4 Estimator for expected number of queries of **QSearch**

In this section we will prove that, for a list L with $t \geq 1$ marked items,

$$\mathbb{E}[H(l)] \geq \frac{|L|}{t}\left(1 - \left(1 - \frac{t}{|L|}\right)^N\right) + \left(1 - \frac{t}{|L|}\right)^N c_q E_{\text{Grover}}(|L|, t)\,, \tag{51}$$

where $H(l) = h_1(l) + h_2(l) c_q E_{\text{Grover}}^{\text{estimator}}(l)$, $N = N_{\text{samples}}$ is the number of classical samples taken by **QSearch**, and the expectation over l is taken over the geometric distribution $l \sim \text{Geo}(f)$, with $f = t/|L|$.

As a quick sanity check, if l is geometrically distributed with parameter f (e.g. $\Pr[x = k] = f(1-f)^{k-1}$), then we have

$$\mathbb{E}[l] = \sum_{k=1}^{\infty}(1-f)^{k-1} f k = -f\frac{\partial}{\partial f}\sum_{k=0}^{\infty}(1-f)^k = -f\frac{\partial}{\partial f}\frac{1}{f} = \frac{1}{f}\,,$$

as expected.

Next, we focus on the first term in Eq. (51). Recall from Eq. (30) that for the classical contribution to the queries, we have

$$\frac{1}{f}\left(1 - (1-f)^N\right) = \sum_{i=1}^{N} f(1-f)^{i-1} i + N(1-f)^N\,. \tag{52}$$

Now, for $h_1(l) = \min(l, N)$, we have the following lemma.

Lemma 17. $\mathbb{E}[h_1(l)] = \frac{1}{f}\left(1 - (1-f)^N\right)$. *(Where x is drawn according to the geometric distribution above with parameter f).*

[27] Of sample size 1.

Proof.

$$\begin{aligned}
\mathbb{E}[g(l)] &= \sum_{k=1}^{\infty}(1-f)^{k-1}fh_1(k) = \sum_{k=1}^{N}(1-f)^{k-1}fk + \sum_{k=N+1}^{\infty}(1-f)^{k-1}fN \\
&= \sum_{k=1}^{N}(1-f)^{k}fk + N(1-f)^{N}\sum_{k=1}^{\infty}(1-f)^{k-1}f \\
&= \sum_{k=1}^{N}(1-f)^{k}fk + N(1-f)^{N} \\
&= \frac{1}{f}\left(1-(1-f)^{N}\right),
\end{aligned}$$

where the final equality holds from (52). □

Hence, we can take as an estimator for the number of classical queries just $\min(l, N)$, where l is the number of items sampled from L before finding a marked one.

Now we turn our attention to the quantum contribution to the number of queries, which is given by

$$(1-f)^{N}c_q E_{\text{Grover}}(|L|, t)\,.$$

We already have from Eq. (13) an estimator $E_{\text{Grover}}^{\text{estimator}}(l)$ such that $\mathbb{E}\left[E_{\text{Grover}}^{\text{estimator}}(l)\right] \geq E_{\text{Grover}}(|L|, t)$. We seek a function h_2 such that, when multiplied by $E_{\text{Grover}}^{\text{estimator}}$ we also have $\mathbb{E}\left[h_2(l)E_{\text{Grover}}^{\text{estimator}}(l)\right] \geq (1-f)^{N}E_{\text{Grover}}(|L|, t)$. Toward this end, let

$$h_2(l) = \begin{cases} 0 & l \leq N \\ 1 & l > N. \end{cases}$$

Then

$$\begin{aligned}
\mathbb{E}[h_2(l)] &= \sum_{k=1}^{\infty}(1-f)^{k-1}fh_2(k) = \sum_{k=N+1}^{\infty}(1-f)^{k-1}f \\
&= (1-f)^{N}\sum_{k=1}^{\infty}(1-f)^{k-1}f = (1-f)^{N}\,.
\end{aligned}$$

It remains to show that $\mathbb{E}\left[h_2(l)E_{\text{Grover}}^{\text{estimator}}(l)\right] \geq (1-f)^{N}E_{\text{Grover}}(|L|, t)$.

Lemma 18. *Let $f, g : \mathbb{N} \to \mathbb{R}$ be non-negative non-decreasing functions, and x a random variable on $\mathbb{N}$. Then*

$$\mathbb{E}[f(x)g(x)] \geq \mathbb{E}[f(x)]\mathbb{E}[g(x)]\,.$$

Proof. Fix $x, y, \in \mathbb{N}$. Because f and g are non-decreasing, we have

$$(f(x)-f(y))(g(x)-g(y)) \geq 0\,,$$

and therefore

$$f(x)g(x) + f(y)g(y) \geq f(x)g(y) + f(y)g(x)\,.$$

Because f and g are non-negative functions on $\mathbb{N}$, f, g and the product fg are Lesbesgue integrable with respect to the probability measure. Taking the expectation value over x and y yields

$$\mathbb{E}_x[f(x)g(x)] + \mathbb{E}_y[f(y)g(y)] \geq \mathbb{E}_x[g(x)]\mathbb{E}_y[g(y)] + \mathbb{E}_y[f(y)]\mathbb{E}_x[g(x)]\,,$$

from which the lemma follows. □

Combining Lemma 18 with the observations above, as well as the fact that both h_2 and $E^{\text{estimator}}_{\text{Grover}}$ are non-decreasing, we conclude that the function $H(l) = h_1(l) + h_2(l)c_q E^{\text{estimator}}_{\text{Grover}}(l)$ satisfies

$$\mathbb{E}[H(l)] \geq \frac{1}{f}\left(1 - (1-f)^N\right) + (1-f)^N c_q E_{\text{Grover}}(|L|, t)\,.$$

for $t \geq 1$. Hence, H is an estimator that always upper bounds, in expectation, the number of queries made by QSearch when there is at least one marked item.