Hybrid Quantum Search Algorithms

Inaugural-dissertation

zur

Erlangung des Doktorgrades

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Universität zu Köln

vorgelegt von

Vahideh Eshaghian

aus Sabzevar

angenommen im Jahr 2025

This dissertation has been accepted in the year 2025 by the Faculty of Mathematics and Natural Sciences of the University of Cologne.

I dedicate this work to the two angels of my life, my parents…

Abstract

This thesis presents two main projects focused on hybrid quantum search algorithms.

The first, **Particle-Guided Grover's Search**, applies Bayesian inference to quantum search when the number of solutions is *a priori* unknown. It begins with a prior distribution that encodes our initial "beliefs" about the true number of solutions in the search space, and recursively updates these beliefs using Bayesian inference. Measurement outcomes of the quantum search that do not yield a solution are interpreted as "evidence", guiding the Bayesian update to determine the optimal number of Grover iterations for subsequent rounds.

This approach leverages Particle Filtering – a Bayesian filtering technique – to avoid explicitly storing the entire prior distribution at each step. Instead of requiring $\mathcal{O}(N)$ memory, the method draws a set of "particles" from the prior and updates their associated weights as quantum search failures accumulate. Particle filtering also offers the advantage of incorporating information from all past failures, rather than only the most recent one. The algorithm's performance is benchmarked against a prior work [15].

The second project, **Runtime–Coherence Trade-offs for Hybrid SAT Solvers**, is focused on a class of hybrid k-SAT solvers. These solvers are based on a well-known random walk technique called Schöning's algorithm, which uses two sources of randomness to perform a local search over the space of all assignments for a k-SAT problem.

In this project, we assume that these sources of randomness are encoded into a long bitstring, allowing the random walk to be interpreted as a deterministic function mapping each bitstring to either 0 or 1, depending on whether the walk successfully finds a satisfying assignment. This functional perspective enables us to pose the question: Given a bound on the coherence time of a quantum computer, which portion of the bitstring should we search coherently, and which portion should be left to classical search?

We address this question by introducing a family of hybrid algorithms called "partial Groverizations", which strategically combine quantum and classical resources. We analyze the runtime performance of this family as a function of the available quantum coherence time. The results of this work have been published in [32].

Zusammenfassung

Diese Dissertation stellt zwei Hauptprojekte vor, die sich mit hybriden Quanten-Suchalgorithmen befassen.

Das erste, **Particle-Guided Grover's Search**, wendet Bayessche Inferenz auf die Quanten-Suche an, wenn die Anzahl der Lösungen *a priori* unbekannt ist. Es beginnt mit einer A-Priori-Verteilung, die unsere anfänglichen „Glauben" über die tatsächliche Anzahl von Lösungen im Suchraum kodiert, und aktualisiert diese Glauben rekursiv unter Verwendung der Bayesschen Inferenz. Messergebnisse der Quantensuche, die keine Lösung liefern, werden als „Evidenz" interpretiert, die die Bayessche Aktualisierung bestimmen, um die optimale Anzahl von Grover-Iterationen für nachfolgende Runden zu bestimmen.

Dieser Ansatz nutzt Partikelfilterung – eine Bayessche Filtertechnik –, um zu vermeiden, dass bei jedem Schritt die gesamte A-Prior-Verteilung explizit gespeichert werden muss. Anstatt $\mathcal{O}(N)$ an Speicher zu benötigen, zieht die Methode eine Menge von „Partikeln" aus der A-Priori-Verteilung und aktualisiert deren zugehörige Gewichte jedes Mal, wenn die Quantensuche erfolglos ist. Die Partikelfilterung bietet außerdem den Vorteil, dass sie Informationen aus allen vorherigen erfolglosen Suchvorgängen berücksichtigt und nicht nur aus dem letzten. Die Leistung des Algorithmus wird mit der einer früheren Resultat in der Literatur [15] verglichen.

Das zweite Projekt, **Runtime–Coherence Trade-offs for Hybrid SAT Solvers**, konzentriert sich auf eine Klasse hybrider k-SAT-Solver. Diese Solver basieren auf einer bekannten Random-Walk-Technik namens Schönings Algorithmus, die zwei Zufallsquellen verwendet, um eine lokale Suche über den Raum aller Zuweisungen eines k-SAT-Problem durchzuführen.

In diesem Projekt gehen wir davon aus, dass diese Zufallsquellen in einer langen Bitkette kodiert sind, sodass der Random Walk als deterministische Funktion interpretiert werden kann, die jede Bitkette entweder auf 0 oder 1 abbildet, je nachdem, ob der Walk erfolgreich eine zufriedenstellende Zuweisung findet. Diese funktionale Perspektive ermöglicht es uns, die Frage zu stellen: Angesichts einer Begrenzung der Kohärenzzeit eines Quantencomputers, welchen Teil der Bitfolge sollten wir kohärent durchsuchen und welchen Teil sollten wir der klassischen Suche überlassen?

Wir gehen dieser Frage nach, indem wir eine Familie von Hybridalgorithmen namens „partielle Groverisierungen" einführen, die Quanten- und klassische Ressourcen strategisch kombinieren. Wir analysieren die Laufzeitleistung dieser Familie als Funktion der verfügbaren Quantenkohärenzzeit. Die Ergebnisse dieser Arbeit wurden in [32] veröffentlicht.

Contents

1 Introduction to Quantum Search and Boolean Satisfiability

The aim of this thesis is to study different aspects of quantum search algorithms, and in particular *Grover iterations*. The problem that quantum search algorithms are designed to tackle is (unstructured) *Search Problem*[1]. Take for example Traveling Salesman Problem (TSP), the problem of finding the shortest possible route that visits every given city exactly once, and returns to the starting city. This problem can be formulated as follows, given a length L is there any *loop*, defined as a closed route, that connects all given cities, and its length is at most L? if so which one? This way, the TSP is formulated as a search problem. If one tries to solve this problem naively, one would need to go through all the loops that include every given city, one by one, and calculate their corresponding length to check if the criteria $\text{length}(\text{loop}) \leq L$ is satisfied for any loop. There are in total $n!$ loops that visit every city exactly once for n given cities since fixing the starting point of a route, or choosing the first city, would leave $n-1$ cities to travel to next, and further fixing the second city, would leave $n-2$ cities, and so on.

Assume we had stored all legitimate loops in a list in advance, the brute-force algorithm will require to look at the list as many as $n!/2$-times *on average*, to be able to tell if any of them satisfies the given condition. Depending on the ordering of cities, we might be lucky and find such loop pretty early in the list, or on the contrary, on a bad day we might need to search through them all. The brute-force algorithm is said to *solve* the TSP, since it either confirms that at least one such loop exists, and outputs the corresponding loop, or stops with an output "not found". Each time we look at the list, we request a piece of information from the list. This request is one of the conventional units to measure the performance of algorithms, and is called a *query*.

This example, moreover, illustrates the practice in the computational setting, to classify computational problems and algorithms in terms of how the required effort scales with the problem size. In our particular example, the *size of the problem* is the number of cities n, and the *effort* is the number of times we need to look into the list of legitimate loops before we find one that satisfies the criteria. In the jargon of computer science, the cost (the number of queries) scales factorially with the problem size (number of cities). In this thesis, we adopt this method of comparing algorithms, where an algorithm is more efficient than others, if its scaling is more favourable. From this perspective, a quantum search algorithm remarkably performs better than the brute-force, namely in a manner where the number of queries scales as $\mathcal{O}(\sqrt{n!})$. Although this "efficiency" is comparably modest, and in practice many heuristics perform much better than $n!$, quantum search algorithms –in general– are more powerful in the sense that they can be applied to many classical search algorithms that are based on search heuristics. We will discuss this further, and provide an example of such hybrid algorithms in Chapter 4.

[1] We will soon come back to what is meant by "unstructured".

1.1 Quantum Search Algorithms

Quantum search algorithms are more efficient than similar classical algorithms since the required "effort" to find a distinguished element in a set that contains it, for a quantum search algorithm is squared root of the classical counterpart. The underlying set of elements that we wish to search in, is called the *search space*. To precisely quantify our comparison of algorithms' performance, we will use the following definitions:

Definition 1. *Big-Oh: the function $f : \mathbb{N} \to \mathbb{N}$ is said to be **in the worst case** of order of function $g : \mathbb{N} \to \mathbb{N}$, as its input n grows if*

$$\exists c, n_0 > 0 \qquad \textit{such that} \qquad f(n) \leq cg(n) \qquad \forall n > n_0,$$

and, is written as $f(n) = \mathcal{O}(g(n))$.

Definition 2. *Big-Theta: the function $f : \mathbb{N} \to \mathbb{N}$ is said to be **on average** of order of function $g : \mathbb{N} \to \mathbb{N}$ as its input n grows if*

$$\exists c_1, c_2, n_0 > 0 \qquad \textit{such that} \qquad c_1 g(n) \leq f(n) \leq c_2 g(n) \qquad \forall n > n_0,$$

and, is written as $f(n) = \Theta(g(n))$.

For example, the brute-force algorithm we discussed earlier, needs $\mathcal{O}(n!)$ queries to the list of loops since for $c = 1$ the number of queries is less than $n!$. This condition actually holds for any $n_0 \geq 1$. We say as well that the *runtime* of this algorithm scales as $\mathcal{O}(n!)$. On the contrary, the corresponding quantum search algorithm has a runtime that scales as $\mathcal{O}(\sqrt{n!})$. We will give a formal definition of runtime later in Section 1.2.

In general, the quantum search algorithms provide a quadratic *speedup* over their classical counterpart. In this context, a speedup means any improvement over the current runtime of an algorithm. It is important to note that, this speedup can be gained assuming no inherent structure in the search space to be exploited. If there exists such structure, it might not be possible to design a quantum search algorithm that could take advantage of this structure, and yet provide a quadratic speedup. The Branch and Bound method [43] is an example where the quantum search algorithms could successfully be applied [48]. In short, the Branch and Bound method which is used to tackle combinatorial optimization problems, breaks the problem of finding an optimal configuration to smaller sub-problems (branching), and calculates bounds for the quantity to be optimized (bounding). The method speeds up the search problem by eliminating sub-problems that cannot contain the optimal solution. The Branch and Bound serves as a base for many search heuristics, and provides a strong tool for many industrial applications.

On the contrary, any database that is structured –in some way– would not be a good playground for quantum search algorithms. Because, normally there is a well-tuned classical search algorithm that is designed to take the full advantage of this structure. A nice example is the NASA/IPAC Extragalactic database, a large database that contained 206 million astronomical objects as of 2014. What holds back the application of quantum search here, is the very structure that has been created upon the storage of the data. This structure typically involves some label assignment to the data such that retrieving the data is done very efficiently. Unfortunately, standard

quantum search algorithms are inherently unstructured; they treat all database elements equally by initializing the search in a uniform superposition as we shall see soon. Thus, unless specific structure is embedded into the "oracle" or initial state [2], quantum search algorithms are agnostic to any labelling or preference.

Assume a set that contains N elements with t-many of them being distinguished in someway. We can model this search space by mapping each element to an index from $0, \dots N-1$. The search problem is defined by a "search criteria" that can be represented as a function $f : i \mapsto \{0, 1\}$ with $i \in \{0, \dots N-1\}$. We call an element *marked* if its index is mapped to one, and aim to find such element in a search problem. As it was discussed above, the task of finding one of these marked elements can be approached classically by repeatedly selecting an element, and checking whether it is marked. This algorithm will succeed in $\mathcal{O}(N/t)$ evaluations of f, or equivalently queries. On the other hand, the corresponding quantum search algorithm only needs $\mathcal{O}(\sqrt{N/t})$ evaluation of the "search criteria" to find a marked element.

In the abstract setting, this function is modelled as a *black box* whose internal working is unknown to us, but is capable of recognizing the solution to a search problem by distinguishing between a marked element and a non-marked one. In the case of function f, by mapping them to one and zero, respectively. This black box is called an *oracle*, and although it can recognize a solution when it is presented, this does not mean that it knows the solution *a priori*, or can distinguish it among other elements in the search space.

Since all the operations in quantum circuits (except for the measurements of course) shall be unitary, given a classical oracle, in this case f, we need to represent it as a unitary operator. For this, we consider $\lceil \log_2 N \rceil$ qubits to encode the indices of all the elements in the search space which are called *computational qubits*, plus one auxiliary qubit which we denote as $|q\rangle$. The quantum oracle constructed from f, O_f, can be represented as

$$|x\rangle|q\rangle \xrightarrow{O_f} |x\rangle|q \oplus f(x)\rangle,$$

with $x \in \{0, \dots N-1\}$, and $\oplus$ denoting addition modulo 2. The auxiliary qubit is usually prepared in the state $(|0\rangle - |1\rangle)/\sqrt{2}$, so if x be a solution to the problem, upon application of the oracle the auxiliary qubit will pick a negative sign, and become $(|1\rangle - |0\rangle)/\sqrt{2}$. Conveniently, the oracle leaves auxiliary qubit intact if x is not a solution. It is important to note that the state of the auxiliary qubit will be left unchanged by the action of O_f, it merely introduces a phase shift of π to the whole state $|x\rangle|q\rangle$. For this reason, we can simplify things and instead represent the oracle as

$$|x\rangle \xrightarrow{O_f} (-1)^{f(x)}|x\rangle.$$

This way, we avoid writing the unchanged $|q\rangle$ through the algorithm. This construction of the oracle yields in a unitary operator, O_f. To see this, we note that in the basis $\{|x\rangle\}_{x=0}^{N-1}$, O_f is diagonal, moreover, it is real. In fact the diagonal of O_f is either one or minus one. Therefore, $O^\dagger = O$, further $O^2 = I$ and this completes the proof.

Before technically discussing quantum search algorithms, we need to talk about a subtle point on oracle design. In many search problems, the oracle has a rather complex structure, and would

[2] The notion of "oracle" will be covered soon.

need more than one auxiliary qubit to be implemented. The extra auxiliary qubits provide a workspace for some "scratch" computation, and are called *garbage qubits*. There are two reasons to erase the garbage qubits once the desired computation is over:

1. Resource reuse: To be able to use them again in later calls to the oracle,

2. Preserving quantum interference: To ensure that interference patterns among computational qubits are not disrupted by entanglement with garbage qubits.

In quantum computing, there usually exist some specific interference patterns among computational qubits that are critical for quantum speedup. If the garbage qubits become correlated with the computational qubits, this can affect the entanglement and interference patterns among the computational qubits. Hence, normally we would not want our computational qubits to correlate with the garbage ones.

Therefore, a further step called *uncomputation* needs to be added to the oracle once the phase shift are computed. The goal of uncomputation is to restore these extra auxiliary qubits to their initial states, usually $|0\rangle$.

In the operational level the uncomputation simply means applying the inverse of the oracle to both the computational and garbage qubits. This step clears any unwanted entanglement between the qubits, making sure the computational qubits are ready for the next operation. We do not discuss the details of uncomputation and oracle design here, and refer to [53] for more details.

In the following, we review the principles of Grover iterations, the building blocks of quantum search algorithms. We further discuss their performance analysis, and argue that whether the number of marked elements t, is known or not, the search cost scales as $\sqrt{N/t}$ queries.

1.1.1 Principles of Grover Iterations

As we discussed in the previous section, a search space of size N can be represented by the indices, $0, \ldots N-1$, or alternatively, the basis states of $\lceil \log_2 N \rceil$ qubits. The corresponding basis states $\{|0\rangle,\ldots|N-1\rangle\}$ are called *computational basis*. To encode this search space, the qubits are prepared in the uniform superposition of computational basis, $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$. The uniformity indicates that every base state is as likely as the others to be measured, confirming our assumption on the absence of any structure in the data. Any quantum search algorithm consists of a sequence of rotations, known as *Grover operator*, that aims to rotate this superposition toward the superposition of marked states. [35]

Assume there are t-many marked elements out of the total N. Initial state $|\psi\rangle$, alternatively can be written as the superposition of marked and unmarked states as

$$|\psi\rangle = \sqrt{\frac{N-t}{N}}|\alpha\rangle + \sqrt{\frac{t}{N}}|\beta\rangle,$$

where $|\alpha\rangle, |\beta\rangle$ are the states corresponding to the superposition of unmarked and marked elements, respectively. It is convenient to define the angle θ such that $\sin\theta/2 = \sqrt{\frac{t}{N}}$, consequently $\cos\theta/2 = \sqrt{\frac{N-t}{N}}$.

The Grover operator is designed to rotate $|\psi\rangle$ – in the plane spanned by $|\alpha\rangle$ and $|\beta\rangle$ – towards $|\beta\rangle$. This is done by two consecutive reflections. The first reflection implements the oracle, and is simply a reflection around $|\alpha\rangle$,

$$O = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The state $|\psi\rangle$ represented as the vector $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$, will be mapped to $\begin{pmatrix} \alpha \\ -\beta \end{pmatrix}$ by this reflection.

The second reflection, or the *diffusion* operator, is a reflection around the state $|\psi\rangle$, given as $D \coloneqq 2|\psi\rangle\langle\psi| - I$. The diffusion operator can be represented in the plane of $|\alpha\rangle$ and $|\beta\rangle$ as,

$$D = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}.$$

It turns out that the combined effect of the two reflections—the oracle followed by the diffusion operator—is equivalent to a rotation by an angle θ. This rotation technically defines the Grover operator,

$$G \coloneqq DO = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \tag{1.1}$$

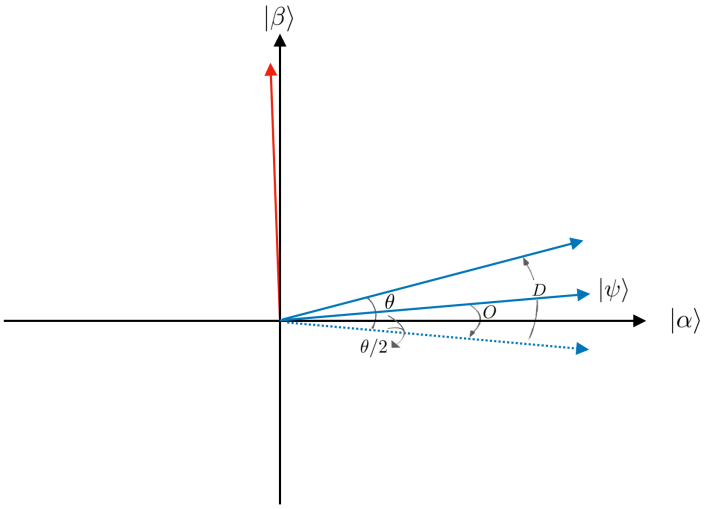
which has the effect of amplifying the amplitudes of marked states while simultaneously reducing the amplitudes of unmarked states. In other words, the application of the Grover operator enhances the overlap between $|\psi\rangle$ and the superposition of marked states. Consequently, the probability of measuring a marked element is increased. The blue parts in figure 1.1 depicts one application of this rotation.

By iteratively applying the Grover operator, we can rotate the initial state $|\psi\rangle$, towards the desired superposition – namely, the superposition of all the marked states $|\beta\rangle$. But how many applications of the Grover operator, or *Grover iterations* are needed? The Grover operator– as discussed above– is a rotation in the plane of marked and unmarked states, and knowing the angle θ we can determine how many rotations are needed to get as close as possible to the desired superposition. It is important to note that since the rotation angle of the Grover operator is twice as the angle between the initial state and the superposition of unmarked states, the rotation angle is already fixed by the parameters of search problem, *i.e.* N and t. Therefore, it is very likely that no combined iterations of Grover can align the initial state perfectly with the superposition of marked states. It is, however, possible to achieve an almost perfect alignment such that measuring the resulting state– most likely– yields in finding a marked element [3]. One can calculate the probability of measuring a marked element after k-many applications of Grover iteration. To see this, we consider the action of k Grover iterations on the initial state,

$$G^k|\psi\rangle = \cos\big(\frac{2k+1}{2}\theta\big)|\alpha\rangle + \sin\big(\frac{2k+1}{2}\theta\big)|\beta\rangle.$$

[3] See [63] for the perfect alignment procedure.

Figure 1.1: Grover rotation shown as two consecutive reflections on the initial state (blue parts). First the oracle operator O, reflects the initial state $|\psi\rangle$, around the superposition of unmarked states $|\alpha\rangle$. Then the diffusion operator D, reflects the resulting state around the initial state. These reflections rotates the initial state towards the superposition of marked states $|\beta\rangle$, by angle θ. The red vector illustrates a *overshooting*, where too many applications of Grover operator will decrease the overlap between the initial state and the marked states.



The probability of finding a marked element by measuring the state $G^k|\psi\rangle$ is given by,

$$p_{\text{succ}}(k) := \sin^2\left(\frac{2k+1}{2}\theta\right)$$

which is defined as the *success probability* of the Grover iterations. Ideally, we want a process that maximizes the success probability. Such rotations shall result in a state which overlaps maximally with $|\beta\rangle$, such that upon measurement, it is most likely to find a solution to the search problem. We call this number the *optimal* number of iterations.

Obviously, $\{|\alpha\rangle, |\beta\rangle\}$ is an orthogonal basis. The optimal number of iterations shall possibly rotate the state $|\psi\rangle$ by $\frac{\pi}{2} - \frac{\theta}{2}$. The criteria to determine the optimum number of iterations reads then,

$$\frac{2k+1}{2}\theta \stackrel{!}{=} \frac{\pi}{2} \Rightarrow 2(2k+1)\arcsin\sqrt{\frac{t}{N}} \simeq \pi \Rightarrow k \simeq \frac{\pi}{4}\sqrt{\frac{N}{t}} \tag{1.2}$$

where we have used the assumption $t \ll N$ to exploit small-angle approximation. This is a reasonable assumptions since if t is not significantly smaller than N, the advantage of Grover iterations diminishes. In such cases, classical algorithms may perform comparably, or even outperform Grover iterations, making the quantum approach less beneficial.

Equation 1.2 also proves that the quantum search solves a search problem with N total elements and t marked ones, in $\mathcal{O}(\sqrt{N/t})$ queries to the oracle. The algorithm achieves this with a con-

stant (*i.e.*, $\mathcal{O}(1)$) probability of success, which can be increased to near certainty with repetition if needed [63].

What if we keep running Grover iterations? Well, the Grover iterations like all other rotations are periodic. By consecutively rotating the initial state it will be mapped to itself at some point, but before that there is a number of rotations that will approximately align this state with the superposition of marked states. Any further iteration would rotate the state further away from the desired superposition, and so-called *overshoot* the rotated state. The red arrow in figure 1.1 depicts this situation.

Therefore, it is very important to apply Grover iterations just as many as the optimum number of iterations. But as we saw earlier the optimum number of iterations is a function of the parameters of the underlying search problem. In many practical search problems, the number of marked elements is not known *a priori*. Can one still take advantage of quantum search algorithms in such cases? The answer to this question is luckily positive. In the next section we will cover different approaches to tackle this problem. The pseudocode 1 outlines the *Quantum Search algorithm*. We adopt the notations $[N]$ to denote a search space of size N, and bin() the binary representation, respectively.

Algorithm 1 Quantum Search

Input: 1) Search space: $[N]$
2) $n+1$ qubits with $n = \lceil \log_2 N \rceil$.
3) A black box oracle O_f, that acts on elements of $[N]$ as $O_f|x\rangle|q\rangle = |x\rangle|q \oplus f(x)\rangle$. $x \in \{0, \dots \text{bin}(N-1)\}$, $f(x) = 1$ if x is a solution to the search problem, otherwise, $f(x) = 0$.
4) An integer k set as the number of Grover iterations.

Output: An element of $[N]$.

1: **Initialization**: $|\psi\rangle|q\rangle = |0\rangle^{\otimes n+1}$
The state of the computational qubits and the one auxiliary qubit are represented by $|\psi\rangle$ and $|q\rangle$, respectively.

2: **Superposition**: $|\psi\rangle|q\rangle \xrightarrow{H^{\otimes n} \otimes HX} |\psi'\rangle|q'\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$
The Hadamard gate $H^{\otimes n}$, and a Pauli x followed by a Hadamard gate HX, are applied to the first n qubits and the last one(auxiliary), respectively.

3: **Grover Iterations**: $|\psi'\rangle|q'\rangle \xrightarrow{(DO_f)^k} \approx |x^*\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$
The Grover operator is applied k-times with $k \simeq \lceil \frac{\pi}{4} \sqrt{N/t} \rceil$.
▷ *The state $|x^*\rangle$ is the superposition of all solutions. Here it is assumed that t is known to the algorithm, if not, the state of the computational qubits could be any superposition of the computational basis.*

4: **Measurement**: The state of computational qubits is measured. With high probability a solution x^* is read.
▷ *Again, if t be unknown, the measurement outcome can be any element of the search space.*

1.1.2 Grover Iterations with unknown number of solutions

How can one take advantage of the quantum search when the number of marked elements is not known in advance? In other words, is there any modification of Grover iterations that can be used when t is unknown, and yet provide the squared root speedup?

There are in general two approaches. The first one utilizes a quantum subroutine called Quantum Counting, and suits the fault-tolerant regime of quantum computing. The second approach proposed by Boyer *et al.* [13] is practical and based on bounding the probability of success from below. We discuss both approaches here, and will propose our approach using Bayesian inference in Chapter 3. The second approach will serve as a baseline to compare our results later.

Quantum Phase Estimation [41] is a quantum subroutine that is able to find the eigenvalues of a unitary operator up to the desired precision. Since all the eigenvalues of a unitary operator have unit modulus, the problem of finding eigenvalues is reduced to finding the corresponding phases of the eigenvalues. In other words, if one manages to estimate the phases, then the corresponding eigenvalues can be approximated easily.

The application of Quantum Phase Estimation to Grover operator G in order to estimate its eigenvalues is called *Quantum Counting* [14]. We now briefly discuss the main idea of Quantum Counting subroutine and refer to [53] for more details. Similar to the electrical circuits, a predetermined memory location in a quantum circuit is called a *register*. In Quantum Phase Estimation two registers are used, the first one is prepared in $|0\rangle^{\otimes d}$, and the second one in the eigenstate $|u\rangle$ of the operator whose eigenvalues we are interested in. In the following we argue why for the Quantum Counting it suffices to prepare the second register in the superposition of computational basis.

Recall the Grover operator from equation 1.1. This operator has the eigenvalues $\lambda_0 = e^{i\theta}$, $\lambda_1 = e^{i(2\pi-\theta)}$, corresponding to the eigenstates $|\nu_0\rangle = \frac{1}{\sqrt{2}}(|\alpha\rangle + i|\beta\rangle)$ $|\nu_1\rangle = \frac{1}{\sqrt{2}}(|\alpha\rangle - i|\beta\rangle)$. The $|\alpha\rangle$ and $|\beta\rangle$ are the superposition of the unmarked and marked elements, respectively. It is not difficult to see that there are $x, y \in \mathbb{C}$ such that

$$x|\nu_0\rangle + y|\nu_1\rangle = \cos(\theta/2)|\alpha\rangle + \sin(\theta/2)|\beta\rangle,$$

with the right hand side being the superposition of the computational basis $|+\rangle^n$. Hence, in Quantum Counting the second register is simply set in the superposition of the computational basis, with the minor change that instead of $n = \lceil \log_2 N \rceil$ computational qubits, totally $n+1$ qubits are used to also include the auxiliary qubit [4].

The number of qubits in the first register d, depends on the desired precision of estimation and the success probability in measuring the phases. We will later discuss how d is related to the probability of success.

The pseudocode 2 presents Quantum Counting subroutine.

The first and second steps set both registers, separately, in the uniform superposition of the computational basis. In the third step, θ and $2\pi - \theta$ the phases of Grover operator's eigenvalues are encoded into the relative phases of the amplitudes of the first register using the controlled-Grover operations. By performing inverse *Quantum Fourier Transform* in the fourth step, the

[4] Here, we assume that only one auxiliary qubit is required to encode the oracle; if more are needed, this number will increase accordingly.

Algorithm 2 Quantum Counting

Input: 1) $d+n+1$ qubits.
2) A black box which performs controlled $-G^j$ operation.

Output: m-bit approximation of θ as $\tilde{\theta}$.

1: **Initialization**: $|0\rangle^{\otimes d}|0\rangle^{\otimes n+1}$

2: **Superposition**: $|0\rangle^{\otimes d}|0\rangle^{\otimes n+1} \xrightarrow{H^{\otimes d}\otimes H^{\otimes n+1}} |+\rangle^{d}|+\rangle^{n+1}$
▷ $|+\rangle^{n+1} = (x|\nu_0\rangle + y|\nu_1\rangle)|+\rangle = x|\nu_0\rangle|+\rangle + y|\nu_1\rangle|+\rangle$. *For simplicity, we restrict our analysis to the branch $|\nu_0\rangle$, and insert the other branch later.*

3: **Controlled-$G^{2^{j-1}}$'s**: $|+\rangle^{d}|\nu_0\rangle \xrightarrow{\prod_{j=1}^{d} C^j G^{2^{j-1}}} (|0\rangle + e^{i\theta 2^{d-1}}|1\rangle)\dots(|0\rangle + e^{i\theta 2^{0}}|1\rangle)|\nu_0\rangle$
$= \sum_{j=0}^{2^d-1} e^{i\theta j}|j\rangle|\nu_0\rangle$.
▷ *The notation C^j means that the control is on the state of the j-th qubit in the first register.*

4: **Inverse Fourier Transform**: $\sum_{j=0}^{2^d-1} e^{i\theta j}|j\rangle \xrightarrow{\mathbf{FT}^\dagger} |\tilde{\theta}\rangle$
On the first register, inverse Fourier transform is applied.
▷ *The full state includes contributions from both branches: $x\sum_{j=0}^{2^d-1} e^{i\theta j}|j\rangle +$ $y\sum_{j=0}^{2^d-1} e^{i(2\pi-\theta)j}|j\rangle$. Applying the inverse FT yields: $x|\tilde{\theta}\rangle + y|2\pi-\tilde{\theta}\rangle$.*

5: **Measurement**: The first register is measured. With high probability $\tilde{\theta}$ or $2\pi-\tilde{\theta}$ are read.

basis of the first register changes from the *Fourier basis* to the computational basis. This allows reading off the phases directly through measurements. *Fourier basis* on a d-dimensional Hilbert space is the orthonormal basis $\{|\tilde{k}\rangle\}_{k=0}^{2^d-1}$ with $|\tilde{k}\rangle = \frac{1}{\sqrt{2^d}}\sum_{j=0}^{2^d-1} e^{2\pi ijk/2^d}|j\rangle$, with $\{|j\rangle\}_{j=0}^{2^d-1}$ being the computational basis and $k \in 0,\dots 2^d-1$. In our case, θ (or similarly $2\pi-\theta$) is encoded as $e^{i\theta j}$ in the amplitudes. By comparing with the standard Fourier basis, we have $\theta = \frac{2\pi k}{2^d}$. The inverse Fourier transform yields a state in the computational basis $|k\rangle_{k=0}^{2^d-1}$, with a peak around the base $|k^*\rangle$ for which $k^* \approx \frac{2^d\theta}{2\pi}$. Consequently, the phase θ can be estimated as $\tilde{\theta} \approx \frac{2\pi k^*}{2^d}$. To keep the pseudocode clear, we refer to the outcome of the inverse FT and the measurement simply as $\tilde{\theta}$.

The quantum Fourier Transform is the quantum counterpart of *discrete Fourier Transform*. We do not discuss this subroutine here, for our purpose it suffices to note that the computational basis $|j\rangle_{j=0}^{2^d-1}$ are mapped to the Fourier basis $|\tilde{k}\rangle_{k=0}^{2^d-1}$ via the quantum FT, and conversely, the inverse quantum FT maps the Fourier basis back to the computational basis. It is worth to mention that both subroutines require $\mathcal{O}(d^2)$ gate operations.

The final stage of Quantum Counting is performing measurement on the first register. Now, if we aim to estimate θ up to $\lceil\frac{n}{2}\rceil+1$ bits of binary precision with $n = \log_2 N$, and with a success probability of at least 0.9, we would need $d = \lceil\frac{n}{2}\rceil+3$ qubits in the first register. This turns out to be sufficient to estimate t, the number of marked elements, reasonably *i.e.*, $|\Delta t| < \mathcal{O}(\sqrt{t})$. (See [53] for detailed analysis). But how about the total number of Grover iterations? In total,

$$\sum_{j=1}^{d} 2^{j-1} = 2^d - 1 = \mathcal{O}(\sqrt{N}),$$

calls to the Grover operator is needed! Therefore, Quantum Counting is not suitable for the current quantum devices, which are far from being fault-tolerant, mainly because it requires $\mathcal{O}(\sqrt{N})$ applications of the Grover operator.

There is a quantum FT-free version of quantum counting that is given by Aaronson *et al.* [1]. This simplified quantum counting procedure is able to estimates the number of marked elements t –from a list of total N elements– in $\mathcal{O}(\sqrt{\frac{N}{t}}\epsilon^{-1}\log(\delta^{-1}))$, with ϵ being the estimation error and δ the probability of failure. This procedure requires $\mathcal{O}(\log N)$ qubits of space.

We now discuss the second approach proposed by Boyer *et al.* [13]. The following lemma captures the core idea underlying their algorithm.

Lemma 1. *Let t be the unknown number of marked elements in a search problem of size N, and $\theta \in [0, \pi/2]$ be the angle such that $\sin(\theta/2) = \sqrt{t/N}$. For an arbitrary positive integer m, and integer j that is drawn from $[0, m-1]$ uniformly, the following holds: If j-many Grover iterations be applied to a register initiated in $|\psi\rangle = \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle$, the average probability of success upon measuring $G^j|\psi\rangle$ is,*

$$\bar{p}_{succ}(m) = \frac{1}{2} - \frac{\sin(2m\theta)}{4m\sin\theta}. \tag{1.3}$$

We observe that in particular, $\bar{p}_{\text{succ}}(m) \geq \frac{1}{4}$ when $m \geq \frac{1}{\sin\theta}$. The proof starts by calculating the average success probability over all the possible *Grover cycle*'s. A *Grover cycle* is defined as one application of Grover iterations to the state $|\psi\rangle$. The average success probability over all possible cycles is,

$$\bar{p}_{\text{succ}}(m) = \frac{1}{m}\sum_{j=0}^{m-1}\sin^2\big((2j+1)\theta/2\big). \tag{1.4}$$

Using trigonometry relations, the right hand side of 1.4 can be simplified to the right hand side of 1.3. Then it is easy to see that for all number of iterations $m \geq \frac{1}{2}\sqrt{\frac{N}{t}}$, the inequality $\frac{\sin(2m\theta)}{4m\sin\theta} \leq \frac{1}{4}$ holds, and this completes the proof.

Using this lemma, Boyer *et al.* could provide a routine that modifies Grover's Search to find a solution to a search problem in $\mathcal{O}(\sqrt{N/t})$ queries to the oracle, even when the number of solutions is not known *a priori*. We call this routine as *Guess-and-Check Grover's Search*. It obviously is probabilistic, and the key insight into its performance is that, after a sufficient number of Grover cycles, the success probability in each cycle becomes at least $1/4$ on average. We do not discuss the runtime analysis here, and refer to [13] for more details.

The algorithm for Guess-and-Check Grover's Search is outlined in the following.

Algorithm 3 Guess-and-Check Grover's Search

Input: 1) $\lceil \log_2 N \rceil + 1$ qubits.
2) Search space: $[N]$.
3) A black box oracle O_f on $[N]$, as described in Algorithm 1.
Output: A marked element x^*.

1: $m \leftarrow 1, \lambda \leftarrow \frac{6}{5}$
2: **Guess number of iterations**: $j \leftarrow$ draw an integer from $[0, m)$ uniformly.
3: **Perform Quantum Search**:
$x \leftarrow$QUANTUMSEARCH$([N], \lceil \log_2 N \rceil + 1$ qubits, $\mathcal{O}_f, j)$ ▷ *This is one Grover cycle.*
4: **if** x is marked **then**
5: **return** x
6: **else**
7: **Expand the interval**: $m \leftarrow \min(\lambda m, \sqrt{N})$
8: **go to** 2

This algorithm will run until it finds a marked element. To avoid an infinite loop when $t = 0$, a stopping time can be introduced such that the runtime of the algorithm still remains $\sqrt{N}$. This way the algorithm might return "no marked element found" incorrectly even though there exists at least one marked element. This is fine since this "failure" error can be made arbitrary small.

To handle the case that the number of solutions in the search problem is relatively large, a classical sampling phase can be introduced. This is necessarily important for $t > \frac{3N}{4}$ since the runtime analysis of the algorithm assumes the condition $t \leq \frac{3N}{4}$.

Guess-and-Check Grover's Search is theoretically simpler than Quantum Counting and also practically better suited for benchmarking the quantum search with unknown t, especially in the near-term and noisy quantum computing regime. But one might still wonder, can one do better than this? The main clue here is that this method remains unaffected by the outcomes of Grover iterations as long as the measurement is unsuccessful. In each Grover cycle, the number of iterations is selected uniformly from a predefined interval, not projecting any *belief* one might have on the potential number of solutions. Furthermore, it does not incorporate the information we might gain from the *failed* attempts or unsuccessful Grover cycles. By interpreting Grover cycles as experiments one can perform on a search space, then it becomes natural to approach the problem from Bayesian inference's perspective.

Additionally, some unstructured search problems, despite an unknown number of marked elements, may provide partial information on the possible number of solutions – a factor that could be leveraged to make such "beliefs" and improve the search efficiency. Studying Grover iterations within the Bayesian framework is the subject of Chapter 3. We will then use Guess-and-Check Grover's Search with slight modifications as a baseline to compare our results. We will review the basics of Bayesian inference in Chapter 2, and in particular will focus on a Bayesian technique called *particle filtering* which comes handy in our later study.

1.2 Boolean Satisfiability Problem

In this section, we discuss what k-SAT problem is, and its significance. k-SAT will be the subject of our study in Chapter 4.

Consider the following problem: in an $N \times N$ chess board with N being a natural number, can one place N queens such that no two queens threaten each other? The above question, called the N-queens problem, is an example of a class of mathematical problems called *Constraint Satisfaction Problem*. In all constraint satisfaction problems, a set of *variables* are given such that each can take values from its corresponding *domain*. The problem then asks whether there exists any way one can *assign* values to all variables such that a set of given limitations, or *constraints*, are simultaneously satisfied. In the N-queens problem, the variables are the N queens, their respective domains are all the $N \times N$ squares of the chess board, and the constraints are

- No two queens can attack each other by being in the same row.
- No two queens can attack each other by being in the same column.
- No two queens can attack each other by being in the same diagonal.

Explicit solutions of N-queens problem except for $N = 2, 3$ were constructed by [36]. However, finding all the solutions is a non-trivial problem.

An important class of constraint satisfaction problems are *Boolean satisfiability* or *SAT problem*. In the SAT problem, the variables are *logical* variables *i.e.* they either take True or False as values, and the constraints are given in the form of a Boolean logic formula defined as follows:

Definition 3. *Boolean logic formula: a formula that takes a set of logical variables*

$$x_i \in \{\textsc{True}, \textsc{False}\} \qquad \textit{for} \qquad i \in \{1, \dots n\},$$

combines them using logical operators And,Or,Not, Implication and Equivalence, and returns either True or False.

Let n be the number of variables that occur in a Boolean logic formula. Since each variable can be interpreted as either True or False, there are in total 2^n ways of assigning values to them. We call each way of assigning values to the variables in a formula, an *assignment*.

An example of a Boolean logic formula is $\phi := ((x_1 \wedge \neg x_3) \Rightarrow x_2) \wedge ((x_2 \vee \neg x_1) \Leftrightarrow x_3)$ that maps the assignment $x_1 = \textsc{True}, x_2 = \textsc{True}, x_3 = \textsc{False}$ to False, and assignment $x_1 = \textsc{False}, x_2 = \textsc{True}, x_3 = \textsc{True}$ to True. We say that the latter assignment *satisfies* the formula ϕ.

We are now in a position to define Boolean satisfiability problem.

Definition 4. *Boolean Satisfiability problem asks whether, given a Boolean logic formula, there exists an assignment of variables that satisfies the formula.*

Normally the Boolean logic formula is transformed into a form that is easier to manipulate algorithmically. Using this form, called *Conjunctive Normal Form* (CNF), any formula can be written as the conjunction of disjunctions, *i.e.* as And of statements composed of Ors of a number of variables or their complements. Each such statement is called a *clause*.

Considering our earlier example of the Boolean logic formula ϕ, it can be transformed to the CNF form as follows,

$$\begin{aligned}\phi &= ((x_1 \wedge \neg x_3) \Rightarrow x_2) \wedge ((x_2 \vee \neg x_1) \Leftrightarrow x_3) \\ &= (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_3).\end{aligned}$$

We do not discuss the details of this transformation here, and refer the curious reader to [61].

1.2.1 k-SAT Problem

A special case of the SAT problem, called the k-SAT problem, is when all clauses – or disjunctions – contains exactly k variables or their negations. A k-SAT formula can be represented in the CNF form as follows,

$$F \coloneqq \bigwedge_{j=1}^{L} C_j \qquad \text{with} \qquad C_j = l_0^{(j)} \vee l_1^{(j)} \vee \ldots l_{k-1}^{(j)},$$

with C_j denoting the j-th clause in the formula, and $l_i^{(j)}$ the $i+1$-th *literal* occurring in the j-th clause. A *literal* is either a Boolean variable or its negation. Using literals, the CNF formulas can be represented in a more neat and compact form.

An example of 3-SAT formula in terms of the Boolean variables is,

$$\begin{aligned}F_1 &\coloneqq \bigwedge_{j=1}^{3} C_j \qquad \text{with,} \\ C_1 &= \neg x_1 \vee x_2 \vee x_3 \\ C_2 &= \neg x_2 \vee x_3 \vee \neg x_4 \\ C_3 &= x_1 \vee \neg x_3 \vee x_4.\end{aligned}$$

Alternatively, F_1 can be represented in terms of literals as,

$$\begin{aligned}F_1 &= \bigwedge_{j=1}^{3} C_j \qquad \text{with,} \\ C_1 &= l_0^{(1)} \vee l_1^{(1)} \vee l_2^{(1)} \\ C_2 &= l_0^{(2)} \vee l_1^{(2)} \vee l_2^{(2)} \\ C_3 &= l_0^{(3)} \vee l_1^{(3)} \vee l_2^{(3)}.\end{aligned}$$

The assignments that satisfy F_1 include $x_1x_2x_3x_4 = \{0000, 0100, 0001, 1100, 1010, 0011, 1110, 1011, 0111, 1111\}$.

Not all SAT formulas have satisfying assignments. Finding a satisfying assignment for a k-SAT formula, or confirming that no such assignment exists, is known to be computationally very hard

for $k \geq 3$. Quantifying the "hardness" of SAT was historically important. To understand this importance, we need to review a few concepts first.

A *decision problem* is any computational problem that can be stated as a YES/NO question. By computational problem we mean any problem that can be modelled on a computer. An example of a decision problem is the primality test of a number stated as: "given a natural number n, is n a prime number?" The TSP problem as it was presented in Section 1 is another example.

We talked earlier about using big-Oh and big-Theta to quantify the performance of some search algorithms. The notion of "complexity" of an algorithm is, however, a much deeper concept. To formalize this notion, and as well quantify the "complexity" of a computational problem, there exists luckily a powerful tool. In 1936 Alan Turing [62] invented a simple abstract machine with the intent of capturing the notion of computability. Later on many attempts could successfully extend this machine to other physical domains of computation like probabilistic, quantum, etc, and added to its strength.

The Turing machine is one of the most prominent inventions of the 20-th century, providing a fundamental measure for both computational problems and algorithms. In what follows we aim to intuitively cover its main idea, and refer the interested reader for a nice and thorough discussion to [4].

The Turing machine is a tuple: a finite set of symbols, a finite set of possible states the machine can be in, and a transition function which consists of a finite set of operations. The machine works on one or more *tapes* that serve as its "memory", and might be of read-only or read/write type. Each tape is a line of cells which is infinite in one direction. For every tape the machine has a *tape head* which enables it to read/write a symbol off/on one cell at a time. Assume the machine works on k tapes. At the i-th time step, k symbols that are directly under the heads are read. Let us call these symbols $c_1, c_2, \ldots c_k$, and the current state machine is at as $s_i \neq s_{\text{HALT}}$. The halt state, s_{HALT}, is an element of the set of possible states for which the machine halts, and returns an output.

Roughly speaking, the transition function at the i-th step does the following:

1. Takes s_i and $c_1, c_2, \ldots c_k$ as inputs.
2. Based on the inputs performs a finite number of operations, and returns potentially new symbols $c'_1, c'_2, \ldots c'_k$
3. Changes the state s_i potentially to a new state from the set of all possible states.
4. Decides the next movement for each head from the set $\{\text{STAY}, \text{RIGHT}, \text{LEFT}\}$.

The RIGHT, LEFT commands mean moving the head of the corresponding tape one cell to the right or left, respectively. The machine continues to operate according to the above instructions until it hits the halt state. Otherwise, it continues forever.

We are now equipped with the promised tool to cover the notion of complexity. An algorithm is formally defined as a Turing machine that computes a problem. A problem, in this context, is typically modelled as a function with a countable domain and range. An algorithm then is a Turing machine that for every input of the function, follows a sequence of steps, and halts with an output of the function. The number of steps such Turing machine takes to solve the problem is called the *running time*, or simply the runtime of the algorithm.

Turing machines are helpful in classifying the computational problems, as well. The first class of functions that was known to be efficiently computable was the *class P*. Class P is defined as all decision problems that are decided correctly (*i.e.* returned a YES/NO) by the Turing machine in a number of time steps that is a polynomial function in the input size. The "P" in the name of this class stands for polynomial time. For example, the problem of determining if a number is prime is shown to be in P [2].

The second studied class of computable functions is the *class NP*, standing for non-deterministic polynomial time. A decision problem is said to be in NP, if and only if there exists a "non-deterministic" Turing machine that can decide it correctly in a number of steps that is bounded by a polynomial function of the input size. A "non-deterministic" Turing machine differs from the standard one mainly in its transition function. The transition function of a non-deterministic Turing machine may prescribe *more than one* action to be performed for any given input configuration – *i.e.*, a state s_i and tape symbols $\{c_j\}_{j=1}^{k}$– at each step. On the contrary, the transition function of a deterministic Turing machine specifies exactly one such action for each input configuration. Intuitively, a non-deterministic Turing machine can be imagined as "trying all possible choices in the actions in parallel" and accepting if any computation path leads to YES.

We are almost where we can talk about the importance of the SAT problem. The final notions we need are "reducibility", "NP-hardness" and "NP-completeness". A decision problem A is said to be *polynomial-time reducible* to another decision problem B if there exists an algorithm (corresponding to a deterministic Turing machine) with polynomial runtime that does the following. For any given input of problem A produces an input for problem B with the below condition: The answer to the original input for A is YES if and only if the answer to the transformed input for B is also YES. And we show it as $A \leq_P B$. Now, a problem B is said to be in the class *NP-hard* if every problem A in NP is polynomial-time reducible to it, *i.e.* $A \leq_P B$. This means that problems in NP-hard are at least as difficult as the hardest problems in NP. Finally, a problem is in the class *NP-complete* if it lies in the intersection of NP-hard and NP.

The long detour was worth taking since the first problem that was proved to be NP-complete, was the Boolean Satisfiability problem [21] and [45]. That is, any problem in NP can be reduced in polynomial time to the Boolean satisfiability problem by a deterministic Turing machine. This result is a well-known theorem in complexity theory, called Cook-Levin. Boolean satisfiability problem is important practically, as well. In fact, many industrial optimization problems can be converted to some form of it. The k-SAT problem will be the focus of our work in Chapter 4, where we will design some class of hybrid (quantum-classical) algorithms that aim to take advantage of the quantum search in some classical SAT-solvers.

2 Introduction to Bayesian Inference

There are in general two different approaches to probabilities in statistics, one is founded by Thomas Bayes in 18th century [12], and formalized later by Laplace [44] while the other was developed much later in the 20th century [33, 51, 52]. The two approaches led to different schools in statistical inference, namely *Bayesian inference* and *frequentist inference*, respectively. Statistical inference is the process of drawing conclusions from observed data, about the quantities that are not observed. To understand this, consider a "random process" we are interested to study through a limited number of "observations" we can afford. To model this process probabilistically there are two main approaches: using Frequentist approach one regards the pieces of information gained from the observations objectively, on the contrary to the Bayesian one where the gained information is interpreted subjectively.

Take for example a biased coin that we aim to guess the probability of it landing on head. Assume we have flipped the coin 14 times, and observed 10 heads and 4 tails. If we ask our frequentist statistician friend, he would consider this probability as a fixed but unknown parameter p, and simply would estimate it by the "sample" proportion of heads. In this approach, it is assumed that the coin "behaves" the same way every time it is tossed, *i.e.*, the chance of getting heads or tails does not change from one toss to the next, and each toss is unaffected by what has happened before or will happen later. The outcome of each tossing of the coin is called a *sample*. Furthermore, he would assume there is an underlying probability distribution that every sample is drawn according to it, in this case probability p, for heads and $1-p$, for tails with $0 \leq p \leq 1$. Our frequentist friend would estimate this probability as $\bar{p} = \frac{10}{14} \approx 0.714$.

Interestingly, if we ask our Bayesian statistician friend, he would take a different approach to this problem. He would assume the probability of coin landing on head –which he would treat as a random variable P–is initially distributed uniformly over $[0, 1]$. In the next step, our friend would wonder how "likely" it is to see the observed data under this assumption. He would consider a function proportional to $f(P) = P^{10}(1-P)^4$, reflecting the dependance of observed data on the variable P. Finally, he would update his initial belief on the uniformity of the distribution of P, using this function and get another distribution which in this case would be the same as $f(P)$ for $0 \leq P \leq 1$. This updated distribution is indeed a probability distribution after normalization (such that the total area under $f(P)$ be one in the interval $[0, 1]$), because $f(P)$ is nonnegative over this interval.

This example clearly illustrates how the two points of view lead to different results, the frequentist approach estimates the unknown probability as a point estimate whereas the Bayesian one provides a probability distribution for it which is regarded as a random variable. Of course, we can calculate the "expected value" of this variable which in this case is $\mathbb{E}[P] \approx 0.688$, but still the two estimated values are different in both nature and quantity. The frequentist estimate treats the probability as a fixed but unknown parameter; the more observations we collect from

the random process, the more accurate this point estimate becomes. This reflects an objective perspective, where the parameter is fixed but unknown. In contrast, the Bayesian approach treats the probability as a random variable, representing our uncertainty or belief about its true value through a probability distribution. As more data become available, this distribution is updated to better capture our knowledge, providing a subjective perspective on the unknown probability.

Both frequentist and Bayesian approaches are essential for drawing scientific conclusions, and deciding which one would be the suitable tool depends on the specific context. For example, the discovery of Higgs boson was established through a frequentist analysis [17], whereas many subsequent studies have employed Bayesian methods to investigate the possibility of new physics (beyond Standard Model) affecting the Higgs field *e.g.* [27].

In Section 1.1.2, we discussed the problem with Grover iterations with an initially unknown number of solutions t. There we looked at a specific algorithm, Guess-and-Check Grover Search, that makes repeated attempts with random assignments of number of iterations, where failed attempts to find a marked element results in new attempts. The question is if we somehow could make active use of these failed attempts in order to gain information about the search space. More specifically, can we gain information on the number of marked elements? Or similarly, we could initially have a "hunch" of what the number of marked elements t are while we are not certain. (One may regard these two questions are related, in the sense that the information gained from a failed attempt potentially could give us such a hunch for the next attempt.) Such questions of partial, or vague knowledge are ideally suited for the framework of Bayesian inference. By interpreting the number of solutions in a search space as a random variable, each time an experiment – in form of Grover iterations – is failed, a new piece of evidence is provided. One can utilize these pieces to update one's beliefs on the distribution of the desired random variable.

In what follows, we begin with the basic concepts from *random variables* all the way to the *Markov Chains*, and will discuss the *Bayes' rule*. In Section 2.2, with the aim of preparing for studying dynamic systems in Bayesian inference, we shortly introduce *Dynamic Bayesian Networks*, and discuss the simplest model of them called *Hidden Markov Model* via an example. In Section 2.3, we discuss different Bayesian marginalization categories to estimate full posterior distributions. Finally, Section 2.4 presents a step by step construction of a Bayesian filtering algorithm known as *Particle Filtering*. To understand why we need to take this long detour, consider the case of the coin example, once more. In this case, there are only two possible outcomes for each experiment, allowing us to represent our beliefs completely. However, in the Grover's search, there are up to N possible hypotheses for the number of solutions, with N being the size of the search space. Therefore, storing a full belief distribution over all possible values of t could require $\mathcal{O}(N)$ memory units. Here is where we resort to *Particle Filtering* which provides a memory-efficient approximation of the belief distribution. Rather than storing probabilities for every possible t, *Particle Filtering* maintains a set of weighted samples– called particles– drawn from an appropriate "importance distribution". These particles are then updated based on all observations made so far, allowing us to capture the most significant features of the evolving belief distribution using a fixed computational budget.

2.1 Basic concepts

Recall the coin example we considered in the previous section. The possible outcomes for throwing the coin were $\{H, T\}$, *i.e.* landing on the head or tail, respectively. Such set that contains all possible outcomes of a random experiment is called *sample space*. A *random variable* is a map from the sample space to the real numbers. In other words, a random variable is the numerical description of the outcomes of a random experiment, which can either be discrete or continuous. In our example, this map can be simply $H \mapsto +1, T \mapsto -1$. A random variable can be considered as a quantity that could take various values, and a "probability distribution" would describe the probability that each value would happen. In the discrete case, the probability distribution is described by a function, which assigns probabilities to each possible value.

An infinite collection of random variables with some index over time, space or another index set, is called a *random* or *stochastic process*. Random processes first came up in physics to describe the random phenomena that their state changes over time. In that context, the dynamic of a system is modelled by a collection of random variables that describe the "state" of a system, and the set of all possible states a system can be in at any point in time is called the *state space*. If the index set be countably infinite, the random process is called *discrete-time*, and is simply an infinite sequence of random variables, *e.g.* $Z_1, Z_2, \dots$. We will only consider discrete-time random processes in this thesis.

The simplest example of a random process is an *i.i.d. process*. A process is said to be *independent* and *identically distributed*(i.i.d.), if it consists of an infinite sequence of random variables that are independent, and follow the same probability distribution. An important i.i.d. process is a *Bernoulli* process with $Z_1, Z_2, \dots$ being independent and following the same Bernoulli distribution. The Bernoulli distribution is a discrete probability distribution over a binary random variable. For example the process of coin flipping we looked at earlier, is a Bernoulli process because the binary random variables $Z_i : \{H, T\} \mapsto \{+1, -1\}$ for $i \in \{1, 2, \dots\}$ are assumed to be independent, and follow the same distribution $P(Z_i = H) = p, P(Z_i = T) = 1 - p$ with $0 \leq p \leq 1$, and for all $i \in \{1, 2, \dots\}$. The independence property of i.i.d. processes means that they are *memoryless* in the sense that for any time step i, the future $Z_{i+1}, Z_{i+2}, \dots$ are independent of the past $Z_1, Z_2, \dots Z_{i-1}$.

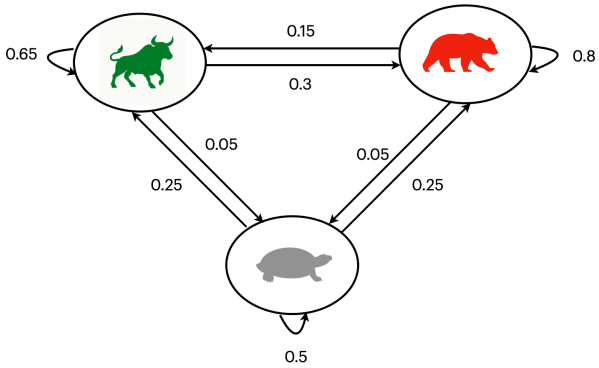
The next simple memoryless random process is a *Markov process* [47]. Before defining a Markov process, we need to discuss the notions of "event" and "conditional probability". An *event* is a subset of sample space to which a probability is assigned. The probability that an event occurs is the sum or the integral of probabilities over this subset depending on the sample space being discrete or continuous. The *conditional probability* is the probability of an event occurring, given that another event is already known to have occurred. The conditional probability of event A "given" event B is defined as $P(A|B) = \frac{P(A \cap B)}{P(B)}$.

A Markov process is a random process $Z_0, Z_1, Z_2, \dots$ such that for all states ζ_i and all $i \geq 0$

$$P(Z_{i+1} = \zeta_{i+1} | Z_i = \zeta_i, \dots Z_0 = \zeta_0) = P(Z_{i+1} = \zeta_{i+1} | Z_i = \zeta_i) \tag{2.1}$$

In other words, the probability distribution of future Z_{i+1} given all the past $Z_0, \dots Z_{i-1}$, and current Z_i, only depends conditionally on the current Z_i. The condition 2.1 is also known as the *Markov property*.

Figure 2.1: A Markov chain model of a hypothetical stock market. There are three possible states in this chain depicted as Bull, Bear and turtle, each indicating an increasing, decreasing or little changes in the market values, respectively. The conditional probabilities are shown by arrows.



The simplest Markov process is called *Markov chain* which is a Markov process with a discrete sample space. A discrete-time Markov chain consists of three components: Initial distribution, the state space and transition matrix. The initial distribution ρ, is the probability distribution of the first random variable in the sequence also known as the *initial state*, the state space that contains all possible values the states can take, and finally, the transition matrix encodes the conditional probabilities. The transition matrix is defined as $\mathbf{P} = (p_{ij})_{i,j=1,\dots n}$ with $p_{ij} \geq 0$ such that $\sum_{i=1}^{n} p_{ij} = 1$. The conditional probability p_{ij} is the probability of the chain transforming from the i-th state to the j-th. The most commonly used Markov chains are discrete-time with a finite state space. In such models, the initial state is described probabilistically by an initial distribution ρ, and the the state of the i-th step is given by $\mathbf{P}^i\rho$. There is a whole field about studying the properties of different Markov chains which is not relevant to our work, hence we leave it. As a simple example of a Markov chain we consider the stock market of a hypothetical economy in a given week. In this model there are totally three possible states: 1-*Bull market*, 2-*Bear market* and 3-*Stagnant market* each representing a trend of increasing, decreasing, or stable average market values, respectively. The transition matrix of this model is given by,

$$\mathbf{P} = \begin{pmatrix} 0.65 & 0.3 & 0.05 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{pmatrix}$$

Figure 2.1 depicts this Markov chain. In many real-world systems, we cannot directly observe the true underlying state, instead we observe some indirect or noisy data that depends on those states. In fact, many dynamic systems modelled by Markov chains require inferring the system's hidden states from observed data using conditional probability. These tasks fall into the category of "inverse problems", where the goal is to recover unobserved causes or states based on observed effects. In the Bayesian framework, this inference is performed using *Bayes' rule*, which allows

the computation of *posterior probabilities* or updated beliefs about the system's state "given" the observations.

Definition 5. *Bayes' rule [12]: Assume a state space $\mathcal{S}$ is partitioned as $H_1, \dots H_n$, i.e.,*

1. $H_k \neq \emptyset \quad \forall k \in \{1, \dots n\}$,
2. $H_k \cap H_l = \emptyset \quad$ *if* $\quad k \neq l$, *and*
3. $\bigcup_{k=1}^{n} H_k = \mathcal{S}$.

Further assume $P(H_k) \neq 0$ and an event E with $P(E) \neq 0$. Then Bayes' rule states that for all $k \in \{1, \dots n\}$

$$P(H_k|E) = \frac{P(E|H_k) \cdot P(H_k)}{\sum_{i=1}^{n} P(E|H_i)P(H_i)}.$$

Each H_k is called a *hypothesis* because the observation of event E might affect its probability. Hence E is called the *evidence*.

In Bayesian inference notation,

- $P(H_k)$ is called *prior probability*, representing the initial belief about the k-th hypothesis before any evidence is observed.
- $P(H_k|E)$ is called the *posterior probability*, which quantifies the updated belief in the k-th hypothesis after considering the evidence E.
- $P(E|H_k)$ is known as the *likelihood*, describing how likely it is to observe the evidence E assuming the k-th hypothesis is true.
- $\sum_{i=1}^{n} P(E|H_i)P(H_i) = P(E)$ is the *marginal likelihood*, and is the probability that the *evidence* is observed irrespect of which hypotheses $1 \dots n$ being true .

Bayes' rule provides a fundamental framework for reasoning about the probability of a cause (hypothesis) given an observed effect (evidence). When dealing with complex systems involving multiple interdependent variables, *Bayesian networks* – also known as *belief networks* – are used to represent and compute joint probability distributions efficiently by exploiting conditional independencies. In the next section, we focus on a temporal extension of these models, known as *dynamic Bayesian networks*, which are particularly relevant for reasoning about systems that evolve over time.

2.2 Dynamic Probabilistic Models in Bayesian Inference

Bayesian networks [50, 54, 55] are probabilistic graphical models that represent a set of random variables, and their conditional dependencies. In such a network the variables are represented as nodes and the conditional dependencies as directed edges. The direction of arrows in a Bayesian network is from a potential cause toward a potential effect. A Bayesian network which models a sequence of variables (over time for example) is called a *dynamic Bayesian network* [23, 24]. In the context of temporal models, this cause-effect structure often corresponds to transitions between

latent(unobserved) system states and their observable outputs. That is, a hidden state at one time step influences the next state as well as the measurements obtained.

The simplest form of dynamic Bayesian networks is represented by the *Hidden Markov Model* (HMM) [7, 8, 9, 10, 11]. HMMs can be seen as a special case of dynamic Bayesian networks where the system evolves over discrete time steps, with hidden states and observable outputs. In a typical HMM, both the hidden states and observations are discrete random variables. While HMMs are limited in structure and assume discrete variables, dynamic Bayesian networks generalize this framework: they allow for more complex state structures, longer-range temporal dependencies, and can model both discrete and continuous variables. For simplicity, we begin by defining HMMs and providing an illustrative example. This helps building intuition for how probabilistic temporal models operate and will form the basis of our discussion on Bayesian sampling in the next section.

A *Hidden Markov Model* consists of an underlying Markov process whose states are not directly observable (hidden), but the process generates observable outputs that depend probabilistically on the hidden states.

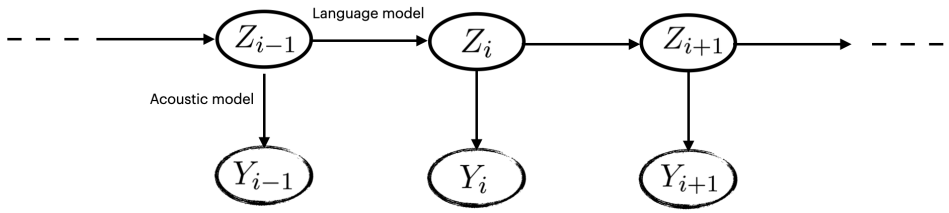
Let Z_i and Y_i be two discrete-time random processes. The pair (Z_i, Y_i) for $i \geq 0$ is called a *hidden Markov model* if

- Z_i be a Markov process whose state is not observable directly,
- $P(Y_i = y_i | Z_0 = \zeta_0, \dots Z_i = \zeta_i) = P(Y_i = y_i | Z_i = \zeta_i)$, $\forall y_i \in \{y_i\}, \forall \zeta_i \in \{\zeta_i\}$ and all $i \geq 0$.

One of the early applications of HMMs was in *speech recognition* ([5, 6]). Speech recognition is a technique that enables the conversion of a piece of speech to text. For many years HMMs were the dominant method in automatic speech recognition systems. Before discussing how this method works we need to talk about a few technicalities. A *phoneme* is the smallest unit of sound in a language such that all the spoken words are built from phonemes. Phonemes are helpful in distinguishing among words, for example the words "cat" and "bat" each consists of three phonemes $/c/æ/t/$ and $/b/æ/t/$, respectively. They differ only in the first phoneme. During the conversion of a speech to text, first the speech or acoustic data is transformed to a "feature" vector in the frequency domain via the Fourier transform. In the next step, one has to "guess" which phoneme the given feature vector most likely corresponds to, at every time step. This is where the main challenge of speech recognition lies. Depending on potential acoustic noises, different accents, and etc the true phonemes can be incorrectly guessed. For example in English $/d/$, $/t/$ and $/\theta/$ (the latter corresponding to the phoneme of th) are similar phonemes, and can be confused during recognition.

The HMMs that are used for speech recognition consists of two statistical models to overcome this issue, namely an *acoustic model*, and a *language model*. An acoustic model is a model that represents the relationship between the phonemes (hidden states) Z_i, and the observable acoustic features, Y_i. Here i serves as an index for time steps. How exactly the acoustic models are created and trained is beyond the scope of this thesis, but mainly they are a probabilistic map between feature vectors and phonemes. For example a particular feature vector could be mapped to either of $/d/$, $/t/$ or $/\theta/$ with probabilities $0.7, 0.2, 0.1$, respectively. Using the observed feature vector is helpful in the estimation of the true phoneme, but one still can do better! Using the structure

Figure 2.2: The graphical representation of the hidden Markov model as the simplest dynamic Bayesian network. In this model there is a hidden state variable, Z_i for each time step which is conditioned on the previous state variable, and is observed via the measured variable Y_i. In the speech recognition process these dependencies are modelled by a language model and an acoustic one.



of the languages, we can refine our guess. In fact, the hidden state Z_i is observed through the feature vector Y_i, and is *conditioned* on the previous state Z_{i-1} in HMMs for speech recognition. For example, in English the probability that a vowel phoneme follows the phoneme $/l/$ is much higher that a consonant phoneme.

At this point, hopefully this example clarifies the dependencies of variables in the HMMs. The following steps summarize the speech recognition procedure at time step i:

1. Using the Fourier transform the (analogue) acoustic data is converted to (digital) feature data, Y_i.

2. A probabilistic function for the state phoneme Z_i "given" the observed feature data Y_i, and the previous phoneme state Z_{i-1} is estimated.

3. The probabilistic function is maximized such that under the assumed models, the observed feature data is most probable.

The dynamical Bayesian network representation of our earlier example from speech recognition is given by figure 2.2.

We will omit the explicit notation of random variables from this point forward for readability and instead refer directly to their corresponding sample space outcomes. The state variable $Z_i = \zeta_i$ with $\zeta_i \in \mathcal{S}$ then would be denoted by ζ_i. We use boldface letters (*e.g.*, $\mathbf{y}$) to emphasize that certain values may be vector-valued. For notational consistency, we note that boldface does not distinguish ζ from ζ; therefore, we assume from this point onward that all state variables are vector-valued, without loss of generality.

Dynamic Bayesian networks define a structured probabilistic model over sequences of variables, representing how hidden states evolve over time and generate observations. This model can be formalized using a probabilistic state space model, which describes the system as follows:

Definition 6 (Probabilistic state space model)**.** *A state space is called probabilistic if ζ_i and $\mathbf{y}_i$, the state of the system and the measurement outcome at time step i, are given according the following conditional probability distributions*

$$\begin{aligned} \zeta_i &\sim p(\zeta_i|\zeta_{i-1}) \quad \zeta_i \in \mathrm{R}^n \\ \mathbf{y}_i &\sim p(\mathbf{y}_i|\zeta_i) \quad \mathbf{y}^m \in \mathrm{R}^m, \end{aligned}$$

with $i = 1, 2, \ldots$.

The notation $x \sim p(x)$ means the variable x is given according to the probability distribution $p(x)$. This definition specifies dynamic Bayesian networks in continuous domains: a Markov process ζ_i governs the evolution of hidden states, while observations $\mathbf{y}_i$ are conditionally dependent on the current state. The conditional probability distribution $p(\zeta_i|\zeta_{i-1})$, describes the "dynamic model" of the system which is stochastic, and $p(\mathbf{y}_i|\zeta_i)$, is the distribution of the measurement outcomes given the state, or simply the "measurement model". According to this model, the state distribution at any time step depends only on the state at the previous step, *i.e.*, $p(\zeta_k|\zeta_{0:k-1}, \mathbf{y}_{1:k-1}) = p(\zeta_k|\zeta_{k-1})$. Further, the measurement model assumes that measurements are conditionally independent of one another and state histories, *i.e.* $p(\mathbf{y}_k|\zeta_{0:k}, \mathbf{y}_{1:k-1}) = p(\mathbf{y}_k|\zeta_k)$. Therefore, this probabilistic state space model is Markovian.

2.3 From Joint Posterior to Bayesian Marginalization

In a Bayesian inference, we often aim to estimate the set of states $\zeta_{0:T} = \{\zeta_0, \zeta_1, \ldots \zeta_T\}$ of a system at time T, given a set of measurement outcomes $\mathbf{y}_{1:T} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots \mathbf{y}_T\}$. Here, $\mathbf{y}_i$ represents all the quantities that are measured at time step i. This is, we want to calculate the joint posterior of all states, considering all the measurements,

$$\begin{aligned} p(\zeta_{0:T}|\mathbf{y}_{1:T}) &= \frac{p(\mathbf{y}_{1:T}|\zeta_{0:T})p(\zeta_{0:T})}{p(\mathbf{y}_{1:T})} \qquad \text{with,} \\ p(\mathbf{y}_{1:T}) &= \int p(\mathbf{y}_{1:T}|\zeta_{0:T})p(\zeta_{0:T})d\zeta_{0:T}. \end{aligned}$$

In this notation, $p(\mathbf{y}_{1:T}|\zeta_{0:T})$ is the likelihood model, $p(\zeta_{0:T})$ the prior distribution, and $p(\mathbf{y}_{1:T})$ the normalization constant. The main struggle is that at each step of the Bayesian update – *i.e.*, every time a new measurement is made– the full posterior is needed to be recomputed. As the number of measurements increases, the computational cost grows, making the calculations intractable.

To address this, a common strategy is to consider a selected marginal distributions of the states rather than the full posterior. Bayesian inference offers three key approaches for this purpose: *Bayesian filtering*, *Bayesian prediction*, and *Bayesian smoothing*.

In what follows, we introduce these three marginal distributions, and in the next section explore one of the Bayesian filtering algorithms called *particle filtering* that we use as a key tool in our analysis later in Chapter 3.

Bayesian Filtering considers the marginal distributions of the current state ζ_i, given the current and previous measurements,

$$p(\zeta_i|\mathbf{y}_{1:i}) \qquad i = 1, \ldots T.$$

Bayesian Prediction considers the marginal distributions of the future state ζ_{i+n} with $n > 0$ given all the current and previous measurements,

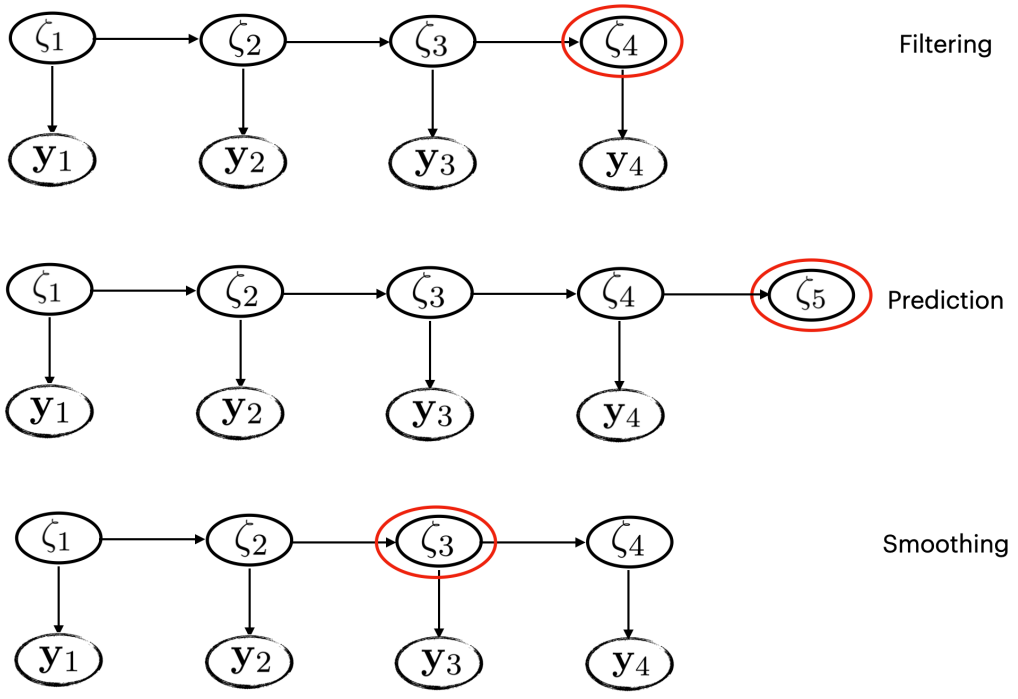
$$p(\zeta_{i+n}|\mathbf{y}_{1:i}) \qquad i = 1, \ldots T, \quad n = 1, 2, \ldots$$

Finally, Bayesian smoothing considers the marginal distributions of a past state ζ_i, given all the current and previous measurements $\mathbf{y}_{1:T}$,

$$p(\zeta_i|\mathbf{y}_{1:T}), \quad 1 \leq i < T.$$

The figure 2.3 presents these three marginal distribution schematically.

Figure 2.3: The graphical representation of Bayesian filtering, prediction and smoothing. In filtering, the posterior distribution at the current state is estimated considering the current and previous measurements, in prediction, the posterior distribution of a future state is estimated given all the measurements so far, and finally in smoothing, the posterior distribution of a past state is estimated given all the measurements so far.



Calculating the marginal distributions associated with Bayesian filtering, prediction, and smoothing is significantly more efficient than computing the full joint posterior over all system states.

Each of these tasks can be approached with various algorithms customized to the structure of the problem. One of the well-known Bayesian filtering algorithms is *Kalman filtering* [37, 38, 60]. The Kalman filter is the solution of Bayesian filtering where the system dynamic and measurement model are both linear, and the noises in both state transitions and measurements are assumed to be Gaussian. The Kalman filter is widely used in GPS navigation systems, for example in position estimation.

2.4 Breaking Down the Posterior: Building Particle Filters Step by Step

Not all Bayesian filtering algorithms are exactly solvable like the Kalman filter. In many practical scenarios, calculating the marginal filtering distribution analytically is not feasible due to nonlinearity or non-Gaussianity in the models. To address this, *Bayesian sampling* methods are employed to approximate these distributions. These methods rely on Monte Carlo approximations that enables practical inference and learning in dynamic systems.

We now introduce how Monte Carlo approximations are used in Bayesian filtering. In Bayesian filtering problems, we often are interested to estimate the expectation value of a function $\mathbf{g} : \mathrm{R}^n \to \mathrm{R}^m$ over a posterior distribution at the current time step T,

$$\mathrm{E}[\mathbf{g}(\zeta)|\mathbf{y}_{1:T}] = \int \mathbf{g}(\zeta) p(\zeta|\mathbf{y}_{1:T}) d\zeta, \tag{2.2}$$

where $p(\zeta|\mathbf{y}_{1:T})$ is the posterior distribution of the state ζ, given the measurements $\mathbf{y}_1, \dots \mathbf{y}_T$ (filtering distribution). Since this integral is generally intractable analytically, numerical methods like Monte Carlo techniques are employed. These algorithms use random sampling to approximate results and, with independent samples and guarantee convergence with an error term of $\mathcal{O}(L^{-1/2})$, with L being the number of samples. The upcoming sections provide three Bayesian sampling algorithms that are used in this context.

The integral in 2.2 can be estimated –using sampling methods– by sum of L independent random samples drawn according to $\zeta^{(i)} \sim p(\zeta|\mathbf{y_{1:T}})$,

$$\mathrm{E}[\mathbf{g}(\zeta)|\mathbf{y}_{1:T}] \approx \frac{1}{L} \sum_{i=1}^{L} \mathbf{g}(\zeta^{(i)}).$$

In what follows, we derive the *Particle Filtering* algorithm step by step, beginning with *Importance Sampling*, then extending it to *Sequential Importance Sampling (SIS)*. Finally, we arrive at *Particle Filtering*, which approximates the posterior distribution using a weighted set of Monte Carlo samples, known as *particles*. The following sections are based on [57, Chapters 1, 6, 11].

2.4.1 Importance Sampling

In many Bayesian filtering problems, it is not possible to sample directly from the posterior distribution due to its complex form. Therefore, rather than sampling from $p(\zeta|\mathbf{y}_{1:T})$, one considers sampling from an *importance distribution*, $\pi(\zeta|\mathbf{y}_{1:T})$ which approximates it, and is easy to sam-

ple from [46]. The choice of an appropriate importance distribution is highly dependent on the specific problem at hand. In Section 2.4.3, we will mention an optimal choice for the importance distribution. The importance distribution is also referred to as the proposal distribution in the literature.

To find out how importance sampling estimates equation 2.2, we assume that L samples are drawn from an importance distribution: $\zeta^{(i)} \sim \pi(\zeta|\mathbf{y_{1:T}})$, with $i = 1, \dots L$. It then follows,

$$\begin{aligned}
\mathrm{E}[\mathbf{g}(\zeta)|\mathbf{y}_{1:T}] &= \int \mathbf{g}(\zeta) p(\zeta|\mathbf{y}_{1:T}) d\zeta \\
&= \frac{\int \mathbf{g}(\zeta) p(\mathbf{y}_{1:T}|\zeta) p(\zeta) d\zeta}{\int p(\mathbf{y}_{1:T}|\zeta) p(\zeta) d\zeta} && (2.3) \\
&= \frac{\int \left[\frac{p(\mathbf{y}_{1:T}|\zeta)p(\zeta)}{\pi(\zeta|\mathbf{y}_{1:T})} \mathbf{g}(\zeta)\right] \pi(\zeta|\mathbf{y}_{1:T}) d\zeta}{\int \left[\frac{p(\mathbf{y}_{1:T}|\zeta)p(\zeta)}{\pi(\zeta|\mathbf{y}_{1:T})}\right] \pi(\zeta|\mathbf{y}_{1:T}) d\zeta} && (2.4)
\end{aligned}$$

To derive equation 2.3, we have used the Bayes' rule to replace the posterior distribution with the likelihood $p(\mathbf{y}_{1:T}|\zeta)$, and prior $p(\zeta)$ which can usually be evaluated easily. To derive 2.4, we have assumed that the importance distribution $\pi(\zeta|\mathbf{y}_{1:T})$, has a greater or equal support compared to the actual posterior, ensuring that the fraction is well-defined.

$$\begin{aligned}
\mathrm{E}[\mathbf{g}(\zeta)|\mathbf{y}_{1:T}] &\approx \frac{\frac{1}{L}\sum_{i=1}^{L} \frac{p(\mathbf{y}_{1:T}|\zeta^{(i)})p(\zeta^{(i)})}{\pi(\zeta^{(i)}|\mathbf{y}_{1:T})} \mathbf{g}(\zeta^{(i)})}{\frac{1}{L}\sum_{i=1}^{L} \frac{p(\mathbf{y}_{1:T}|\zeta^{(i)})p(\zeta^{(i)})}{\pi(\zeta^{(i)}|\mathbf{y}_{1:T})}} && (2.5) \\
&= \sum_{i=1}^{L} w^{(i)} \mathbf{g}(\zeta^{(i)}) && (2.6) \\
\text{with} \qquad w^{(i)} &= \frac{\frac{p(\mathbf{y}_{1:T}|\zeta^{(i)})p(\zeta^{(i)})}{\pi(\zeta^{(i)}|\mathbf{y}_{1:T})}}{\left[\sum_{i=1}^{L} \frac{p(\mathbf{y}_{1:T}|\zeta^{(i)})p(\zeta^{(i)})}{\pi(\zeta^{(i)}|\mathbf{y}_{1:T})}\right]} && (2.7)
\end{aligned}$$

Using Monte Carlo approximation according to the importance distribution, one arrives at 2.5, which can be written as the weighted sample average (see 2.6). The weights $w^{(i)}$, are meant to correct for approximating the target distribution, and are rather easy to calculate. First, one computes the nominator in 2.7, which corresponds to the unnormalized weights, and then divides these unnormalized weights by their sum *i.e.*, denominator, to get the normalized weights.

In summary, the posterior distribution can be approximated by an importance distribution as

$$p(\zeta|\mathbf{y}_{1:T}) \approx \sum_{i=1}^{L} w^{(i)} \delta(\zeta - \zeta^{(i)}), \tag{2.8}$$

with $\delta()$ representing the Dirac delta function.

Given a prior $p(\zeta)$ and a measurement model $p(\mathbf{y}_{1:T}|\zeta)$, one can approximate the posterior by *importance sampling* given in Algorithm 4.

Algorithm 4 Importance Sampling

Input: 1) Prior: $p(\zeta)$
2) Measurement model: $p(\mathbf{y}_{1:T}|\zeta)$
3) Importance distibution: $\pi(\zeta|\mathbf{y}_{1:T})$
4) Number of samples: L

Output: Posterior approximation: $p(\zeta|\mathbf{y}_{1:T})$

function IMPORTANCE SAMPLER(.)
  for $i = 1, \ldots, L$ **do**
    Draw samples $\zeta^{(i)} \sim \pi(\zeta|\mathbf{y}_{1:T})$
    Compute unnormalized weights: $\tilde{w}^{(i)} \leftarrow \frac{p(\mathbf{y}_{1:T}|\zeta^{(i)})p(\zeta^{(i)})}{\pi(\zeta^{(i)}|\mathbf{y}_{1:T})}$,
  Then normalize: $w^{(i)} \leftarrow \frac{\tilde{w}^{(i)}}{\sum_{j=1}^{L}\tilde{w}^{(j)}}$ for $i = 1, \ldots L$.
  return $p(\zeta|\mathbf{y}_{1:T}) \approx \sum_{i=1}^{L} w^{(i)}\delta(\zeta - \zeta^{(i)})$.

2.4.2 Sequential Importance Sampling

When dealing with dynamic systems where the state evolves over time and new measurements are continuously acquired, a one-time application of importance sampling is not sufficient. Instead, one typically employs sequential Monte Carlo methods, such as *Sequential Importance Sampling* (SIS). The main idea behind SIS is to iteratively draw sets of samples (or particles) and update their corresponding weights, with the importance distributions and weights being recursively linked to those of the previous time step.

To derive SIS algorithm, we assume that the underlying state space is probabilistic, and consider the full posterior distribution of the system from $t = 0$ up to a time step $t = k$,

$$p(\zeta_{0:k}|\mathbf{y}_{1:k}) \propto p(\mathbf{y}_k|\zeta_{0:k}, \mathbf{y}_{1:k-1})p(\zeta_{0:k}|\mathbf{y}_{1:k-1}) \tag{2.9}$$
$$= p(\mathbf{y}_k|\zeta_k)p(\zeta_k|\zeta_{0:k-1}, \mathbf{y}_{1:k-1})p(\zeta_{0:k-1}|\mathbf{y}_{1:k-1}) \tag{2.10}$$
$$= p(\mathbf{y}_k|\zeta_k)p(\zeta_k|\zeta_{k-1})p(\zeta_{0:k-1}|\mathbf{y}_{1:k-1}). \tag{2.11}$$

Relation 2.11 is particularly significant because it illustrates how the full posterior distribution up to time $t = k$ is recursively related to that of the previous time step. The proportionality in 2.9 follows from the conditional probability identity $p(a|b, c) = \frac{p(c|a,b)p(a|b)}{p(c|b)}$ [1]. In this case $p(c|b) = p(\mathbf{y}_k|\mathbf{y}_{1:k-1})$, is a constant since the measurements' outcomes $\mathbf{y}_i$, for $i = 1, \ldots k$, are assumed to be fixed and can be dropped. Derivations of 2.10 and 2.11, rely on the Markov properties of the state space model.

[1] To derive this relation, Bayes' rule and chain rule for probabilities are used.

Using the same logic as in Section 2.4.1, one can indeed construct a SIS procedure that samples particles according to a given proposal distribution $\zeta_{0:k}^{(i)} \sim \pi(\zeta_{0:k}|\mathbf{y}_{1:k})$, and computes their corresponding wights. Let us consider the importance weights of the particles at time $t = k$:

$$w_k^{(i)} \propto \frac{p(\zeta_{0:k}^{(i)}|\mathbf{y}_{1:k})}{\pi(\zeta_{0:k}^{(i)}|\mathbf{y}_{1:k})} \tag{2.12}$$

$$\propto \frac{p(\mathbf{y}_k|\zeta_k^{(i)})p(\zeta_k^{(i)}|\zeta_{k-1}^{(i)})p(\zeta_{0:k-1}^{(i)}|\mathbf{y}_{1:k-1})}{\pi(\zeta_{0:k}^{(i)}|\mathbf{y}_{1:k})} \tag{2.13}$$

$$= \frac{p(\mathbf{y}_k|\zeta_k^{(i)})p(\zeta_k^{(i)}|\zeta_{k-1}^{(i)})}{\pi(\zeta_k^{(i)}|\zeta_{0:k-1}^{(i)}, \mathbf{y}_{1:k})} \frac{p(\zeta_{0:k-1}^{(i)}|\mathbf{y}_{1:k-1})}{\pi(\zeta_{0:k-1}^{(i)}|\mathbf{y}_{1:k-1})} \tag{2.14}$$

$$= \frac{p(\mathbf{y}_k|\zeta_k^{(i)})p(\zeta_k^{(i)}|\zeta_{k-1}^{(i)})}{\pi(\zeta_k^{(i)}|\zeta_{0:k-1}^{(i)}, \mathbf{y}_{1:k})} w_{k-1}^{(i)}, \tag{2.15}$$

with $w_{k-1}^{(i)} \propto \frac{p(\zeta_{0:k-1}^{(i)}|\mathbf{y}_{1:k-1})}{\pi(\zeta_{0:k-1}^{(i)}|\mathbf{y}_{1:k-1})}$, being the importance weights of the particles at time $t = k-1$, we have successfully constructed a recursive relation for importance weights.

To arrive at equation 2.13, we have applied relation 2.11. Moreover, in deriving 2.14, we assumed that the importance distribution can be constructed recursively such that

$$\pi(\zeta_{0:k}|\mathbf{y}_{1:k}) = \pi(\zeta_k|\zeta_{0:k-1}, \mathbf{y}_{1:k})\pi(\zeta_{0:k-1}|\mathbf{y}_{1:k-1}) \tag{2.16}$$

We can now summarize one step of the SIS procedure as follows: assume we have already drawn the particles $\zeta_{0:k-1}^{(i)}$ from the importance distribution $\pi(\zeta_{0:k-1}|\mathbf{y}_{1:k-1})$ in the previous steps. Further assume the importance weights from the previous step, $t = k-1$, are given $w_{k-1}^{(i)}$,

1. Draw L new particles for $t = k$ according to $\zeta_k^{(i)} \sim \pi(\zeta_k|\zeta_{0:k-1}^{(i)}, \mathbf{y}_{1:k})$,

2. Calculate the corresponding (unnormalized) weights, $\tilde{w}_k^{(i)} = \frac{p(\mathbf{y}_k|\zeta_k^{(i)})p(\zeta_k^{(i)}|\zeta_{k-1}^{(i)})}{\pi(\zeta_k^{(i)}|\zeta_{0:k-1}^{(i)}, \mathbf{y}_{1:k})} w_{k-1}^{(i)}$,

3. Normalize the weights $w_k^{(i)} = \frac{\tilde{w}_k^{(i)}}{\sum_{j=1}^{L} \tilde{w}_k^{(j)}}$.

The new particles $\{\zeta_k^{(i)}\}_{i=1}^{L}$ with their corresponding importance weights $\{w_k^{(i)}\}_{i=1}^{L}$ represent the approximation to the posterior distribution $p(\zeta_k|\mathbf{y}_{1:k})$.

The algorithm 5 summarizes the full SIS procedure.

Algorithm 5 Sequential Importance Sampling

Input: 1) Prior: $p(\zeta_0)$
2) Measurement model: $p(\mathbf{y}_i|\zeta_i)$
3) Dynamic Model: $p(\zeta_i|\zeta_{i-1})$
4) Importance distribution: $\pi(\zeta_i|\zeta_{0:i-1}, \mathbf{y}_{1:i})$
5) Number of samples: L

Output: Full Posterior approximation: $p(\zeta_k|\mathbf{y}_{1:k})$ for $k = 1, \dots T$

function SIS(.)
  for $i = 1, \dots L$ **do**
    Draw initial samples according to $\zeta_0^{(i)} \sim p(\zeta_0)$,
    Set the initial weights $w_0^{(i)} = 1/L$.
  for $k = 1, \dots T$ **do**
    for $i = 1, \dots L$ **do**
      Draw samples according to $\zeta_k^{(i)} \sim \pi(\zeta_k|\zeta_{0:k-1}^{(i)}, \mathbf{y}_{1:k})$.
      Compute unnormalized weights $\tilde{w}_k^{(i)} \leftarrow \frac{p(\mathbf{y}_k|\zeta_k^{(i)})p(\zeta_k^{(i)}|\zeta_{k-1}^{(i)})}{\pi(\zeta_k^{(i)}|\zeta_{0:k-1}^{(i)}, \mathbf{y}_{1:k})} w_{k-1}^{(i)}$,
    Then normalize $w_k^{(i)} \leftarrow \frac{\tilde{w}_k^{(i)}}{\sum_{j=1}^{L} \tilde{w}_k^{(j)}}$.
    Store: $\{\zeta_k^{(i)}, w_k^{(i)}\}_{i=1}^{L}$
  return $p(\zeta_k|\mathbf{y}_{1:k}) \approx \sum_{i=1}^{L} w_k^{(i)} \delta(\zeta_k - \zeta_k^{(i)})$ for $k = 1, \dots T$.

It is convenient to assume the importance distribution is Markovian in the sense that,

$$\pi(\zeta_k|\zeta_{0:k-1}, \mathbf{y}_{1:k}) = \pi(\zeta_k|\zeta_{k-1}, \mathbf{y}_{1:k}). \tag{2.17}$$

This eliminates the need to store all $\zeta_{0:k}^{(i)}$ throughout the algorithm, hence simplifying the sampling process. This assumption also is often made in the *Particle Filtering* which we are going to discuss next.

2.4.3 Particle Filtering

We finally arrive at the *Particle Filtering*, the Sequential Importance Sampling algorithm with a *resampling* step added to it [26, 34, 42, 56]. SIS can encounter a problem called the *degeneracy problem*. Degeneracy happens when most of the samples (particles) have negligible weights and result in a poor and ineffective representation of the target distribution. This issue has historically limited the practical application of SIS. To address this issue, a *resampling* step is added to the procedure, transforming it into *Sequential Importance Resampling* (SIR) or particle filtering. The resampling step ensures that particles with very small weights are removed, and the particles with considerable weights are duplicated. The algorithm 6 presents how resampling is performed at the time step $t = k$.

It is important to note that resampling does not change the desired expectation value, meaning that $\mathbb{E}_R[\mathbf{g}(\zeta^R)|\mathbf{y}_{1:T}] = \mathbb{E}[\mathbf{g}(\zeta)|\mathbf{y}_{1:T}]$, where R represents the set of resampled indices. We refer to [57] for the proof. Resampling, nevertheless, introduces additional variance to the estimate.

Algorithm 6 Resampling in Particle Filtering

Input: The current particles with corresponding weights: $\{\zeta_k^{(i)}, w_k^{(i)}\}_{i=1}^L$
Output: Resampled particles with corresponding weights: $\{\zeta'^{(i)}_k, w_k'^{(i)}\}_{i=1}^L$

```
function RESAMPLING(.)
    for j = 1, ... L do
        Draw index i ∈ {1, ..., L} with probability w_k^(i)
        ζ'_k^(j) ← ζ_k^(i)
        Set the new weights: w_k'^(j) ← 1/L
    return {ζ'_k^(i), w_k'^(i)}_{i=1}^L
```

This is because resampling is a stochastic process that involves randomly selecting particles based on their weights, which can lead to the loss of diversity among particles. There are different methods to resample new particles efficiently to minimize the variance. We do not discuss them here, and refer again to [57] for details. For our purpose it suffices to note that for this reason, the resampling step is not executed at every time step but only when it is necessary, to balance a trade-off between weight degeneracy and increased variance due to resampling. In many particle filtering implementations, resampling is triggered based on a criterion involving the "effective" number of particles.

The effective number of particles is estimated as,

$$n_{\text{eff}} \approx \frac{1}{\sum_{i=1}^{L} \left(w_k^{(i)}\right)^2},$$

where $w_k^{(i)}$, denotes the normalized weight of the i-th particle at the time step k. Resampling is typically performed when n_{eff} falls below a specified threshold, *e.g.* $< L/10$ with L the total number of particles. This ensures that the particle set remains representative of the underlying distribution. As the variance of the particle weights increases, the denominator in this expression grows, leading to a decrease in n_{eff}. When n_{eff} drops below the threshold, it indicates that a few particles dominate the weight distribution, and many have negligible weights. Resampling addresses this imbalance by replacing the current weighted particle set with a uniformly weighted set, such that the algorithm can concentrate on areas where the true state is more likely to be, based on observations.

Algorithm 7 outlines the particle filtering procedure.

Algorithm 7 Particle Filtering

Input: 1) Prior: $p(\zeta_0)$
2) Measurement model: $p(\mathbf{y}_i|\zeta_i)$
3) Dynamic Model: $p(\zeta_i|\zeta_{i-1})$
4) Importance distribution: $\pi(\zeta_i|\zeta_{i-1}, \mathbf{y}_{1:i})$
5) Number of samples: L

Output: Full Posterior approximation: $p(\zeta_k|\mathbf{y}_{1:k})$ for $k = 1, \dots T$

function PARTICLE FILTER(.)
  for $i = 1, \dots L$ **do**
    Draw initial samples according to $\zeta_0^{(i)} \sim p(\zeta_0)$,
    Set the initial weights $w_0^{(i)} = 1/L$.
  for $k = 1, \dots T$ **do**
    for $i = 1, \dots L$ **do**
      Draw samples according to $\zeta_k^{(i)} \sim \pi(\zeta_k|\zeta_{k-1}^{(i)}, \mathbf{y}_{1:k})$
      Compute $\tilde{w}_k^{(i)} \leftarrow \frac{p(\mathbf{y}_k|\zeta_k^{(i)})p(\zeta_k^{(i)}|\zeta_{k-1}^{(i)})}{\pi(\zeta_k^{(i)}|\zeta_{k-1}^{(i)}, \mathbf{y}_{1:k})} w_{k-1}^{(i)}$,
    Then normalize $w_k^{(i)} \leftarrow \frac{\tilde{w}_k^{(i)}}{\sum_{j=1}^{L} \tilde{w}_k^{(j)}}$.
    $n_{\text{eff}} \leftarrow \frac{1}{\sum_{i=1}^{L} (w_k^{(i)})^2}$
    if $n_{\text{eff}} < L/10$ **then**
      $\{\zeta_k^{(i)}, w_k^{(i)}\}_{i=1}^{L} \leftarrow$ RESAMPLING$(\{\zeta_k^{(i)}, w_k^{(i)}\}_{i=1}^{L})$
    Store: $\{\zeta_k^{(i)}, w_k^{(i)}\}_{i=1}^{L}$
  return $p(\zeta_k|\mathbf{y}_{1:k}) \approx \sum_{i=1}^{L} w_k^{(i)} \delta(\zeta_k - \zeta_k^{(i)})$ for $k = 1, \dots T$.

Here, as mentioned before, we have assumed that the importance distribution is Markovian in the sense of equation 2.17 (see input of Algorithm 7). The performance of Particle Filtering algorithm depends on how well the importance distribution π, can approximate the target distribution. According to [57] the optimal importance distribution[2] that gives the lowest possible variance for the importance weights is,

$$\pi(\zeta_k|\zeta_{0:k-1}, \mathbf{y}_{1:k}) = p(\zeta_k|\zeta_{k-1}, \mathbf{y}_k). \tag{2.18}$$

This is known as the *optimal importance distribution* in particle filtering.[26, 56]

[2] This is indeed the case, for generic non-Markovian importance distributions.

3 From Failure to Insight: Bayesian Strategies in Quantum Search

As we discussed in Section 1.1.2, Guess-and-Check Grover's Search [13] solves a search problem with N total elements and an unknown number of t solutions, in $\mathcal{O}(\sqrt{N/t})$ queries to the oracle. This algorithm does not rely on prior knowledge of the number of candidate solutions; instead, for each Grover cycle, it randomly selects a number from a specified interval and set it as of the proposed number of iterations. In contrast, *Particle-Guided Grover's Search*, the method discussed below, formulates our initial "beliefs" as a prior distribution, which is updated based on the failures of each Grover cycle. This evolving posterior is then used to adaptively determine the number of iterations for the next Grover cycle.

To benchmark the performance of *Particle-Guided Grover's Search*, we compare it against the practical (non-asymptotic) runtime of the Guess-and-Check Grover's Search algorithm. Cade *et al.* [15] have conducted a detailed analysis of the query complexity of this algorithm that goes beyond asymptotic estimates. They introduce a modified version of the Guess-and-Check strategy and provide rigorous upper bounds on both the expected and worst-case query complexities for Grover's search with an unknown number of solutions. We adopt their algorithm as a baseline, applying minor modifications, and implement it using a Monte Carlo approximation. Section 3.1 presents both the original *QSearch* algorithm from Cade *et al.* and the adapted version used in our comparison.

In Section 3.2, we introduce the methodology underlying *Particle-Guided Grover's Search*. Section 3.3 then presents the results and key findings of this approach.

3.1 Related works

In this section we first discuss the *QSearch algorithm*, the modified Guess-and-Check Grover's Search, which is proposed by [15]. Afterwards, we will briefly review the results of the first steps to this work 3.2, which was performed by Robert Derichs [25].

Let us first consider the key modifications introduced in *QSearch algorithm*:

1. A classical sampling step is added prior to calling the Quantum Search algorithm to include the case $t > 3N/4$.

2. A global time-out mechanism is introduced to manage the case $t = 0$, and ensure a lower bound on the success probability.

3. A time-out mechanism is also added to limit the number of quantum queries in each instance of Guess-and-Check Grover's Search, allowing the algorithm to restart once this limit is reached.

Using these modifications to Guess-and-Check Grover's Search, Cade *et al.* have presented algorithm 8. Lines 9-16 correspond to Guess-and-Check Grover's Search algorithm 3. In Guess-and-Check Grover's Search, the algorithm initially performs a Grover cycle with $j = 0$ iterations, which is equivalent to classical sampling. In contrast, QSearch begins with a random choice of $\{0, 1\}$ iteration, since the initial value of m is set to $6/5$ rather than 1 as in the original Guess-and-Check Grover's Search (see line 7). The total number of oracle queries is capped at $9.2\sqrt{N}$. If no marked element is found within this limit, the algorithm resets $m = 6/5$ and restarts Guess-and-Check Grover's Search over. In each Grover cycle, the number of Grover iterations, plus one for measurement, is added to the total number of quantum queries (see line 14).

A global time-out $N_{\text{runs}} = \log_3(1/\epsilon)$ is introduced, with ϵ being the upper bound on the probability of failure (see line 6). Finally, the classical sampling step is included in line 2, with the number of samples $N_{\text{Sample}} = 130$, as specified in [15].

We now modify the QSearch algorithm to enable Monte Carlo simulation. To simulate ClassicalSampling and Quantumsearch, we compare the "true success probability" of each subroutine with a random number $u \sim \mathcal{U}[0, 1)$. Here $\mathcal{U}[.,.)$ represents a uniform distribution over a real-valued half-open interval. For our benchmark, we sample the "true number of solutions" in the search space according to a given distribution and denote it as t_{actual}. This value is used to compute the true success probabilities: $p^{C}_{\text{success}} = t_{\text{actual}}/N$ for classical sampling, and $p^{Q}_{\text{success}}(j) = \sin^2((2j + 1)\arcsin(\sqrt{t_{\text{actual}}/N}))$ for Quantum Search, with j being the number of iterations. A call to either subroutine is considered successful if its corresponding success probability exceeds or equals the random number q.

In this simulation, we adopt an alternative measure of complexity: we count the number of queries made to the classical verification function until a marked element is successfully found. Each query to the Grover oracle typically involves two classical function evaluations, as the oracle implementation generally requires two such calls– one for performing the reflection about the superposition of unmarked states, and another for uncomputation. Accordingly, unless stated otherwise, the runtime throughout this chapter is reported in terms of classical queries.

Further, we set the upper bound on the failure probability to $\epsilon = 0$, meaning that we only count the number of successful attempts in finding a solution.

Pseudocode 9 presents the *QSearch Simulator*. If ClassicalSampler is successful, the solution is returned along with the current query count. Otherwise, the subroutine is considered to have failed, and the preset number of classical sampling attempts is added to the total query count (see line 15).

The average runtime of QSearch Simulator, compared to the expected runtime of QSearch, is shown in Figure 3.1. The average is computed over both the number of solutions t and the size of the search space N, and the results are presented as a function of the ratio t/N. The search space sizes were set to $N \in \{10^3, \ldots 10^6\}$, with increments of 10^3. The number of solutions was restricted to the regime $t \leq N/4$, which is the focus of our study. Values of t were sampled according to $t \sim \mathcal{U}\{1, \lceil N/4 \rceil\}$, with 10^3 samples drawn for a fixed value of N. The average is taken over each fraction of t/N. Here $\mathcal{U}\{a, b\}$ denotes a uniform distribution over the discrete integers from a to b, inclusive. For the expected runtime of QSearch, we have used formulas (1-3) given in [15]. Although both runtimes align overall, the Monte Carlo simulation yields lower query estimates than the theoretical expectation for small values of t/N.

Algorithm 8 QSearch [15]

Input: 1) Search space: $[N]$,
2) Number of classical sampling: N_{Sample},
3) Upper bound on the probability of failure: $\epsilon > 0$.
4) $\lceil \log_2 N \rceil + 1$ qubits.
5) A black box oracle O_f on $[N]$, as described in Algorithm 1.
Output: A marked element or "No marked element found".

function QSearch($[N]$, N_{Sample}, ϵ, $\lceil \log_2 N \rceil + 1$ qubits, O_f)
  $x \leftarrow$ ClassicalSampling($[N]$, N_{Sample})
  if x is marked **then**
    return x
  $N_{\text{runs}} \leftarrow \lceil \log_3(1/\epsilon) \rceil$, $Q_{\text{max}} \leftarrow \alpha\sqrt{N}$, $r \leftarrow 0$ ▷ $\alpha = 9.2$
  while $r < N_{\text{runs}}$ **do**
    $m \leftarrow \frac{6}{5}$, $Q_{\text{sum}} \leftarrow 0$
    $j \leftarrow$ draw an integer from $[0, m)$ uniformly.
    while $Q_{\text{sum}} + j \leq Q_{\text{max}}$ **do**
      $y \leftarrow$QuantumSearch($[N]$, $\lceil \log_2 N \rceil + 1$ qubits, $\mathcal{O}_f$, j) ▷ *One Grover cycle.*
      if y is marked **then**
        return y
      else
        $Q_{\text{sum}} \leftarrow Q_{\text{sum}} + j + 1$ ▷ *Add* $j + 1$ *to quantum queries.*
        $m \leftarrow \min(\lambda m, \sqrt{N})$ ▷ $\lambda = \frac{6}{5}$
        $j \leftarrow$ draw an integer from $[0, m)$ uniformly.
    $r \leftarrow r + 1$
  return "No marked element found"

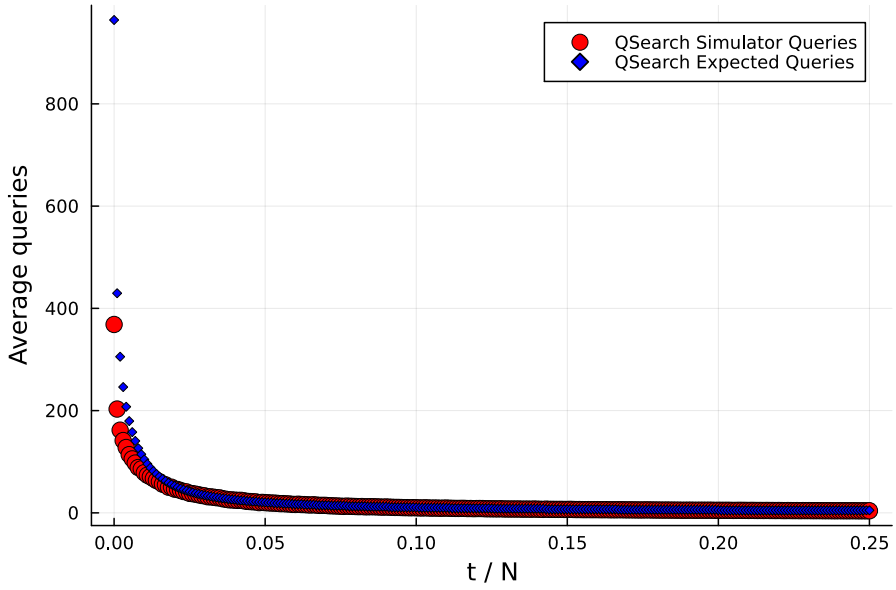
Input: Search space: $[N]$, Number of samples: N_{Sample}
Output: A marked element or "No marked element found"

function ClassicalSampling(N, N_{Sample})
  $k \leftarrow 0$
  while $k \leq N_{\text{Sample}}$ **do**
    $x \leftarrow$ Sample from $[N]$ uniformly at random.
    if x is marked **then**
      return x
    $k \leftarrow k + 1$
  return "No marked element found."

Algorithm 9 QSearch Simulation via Monte Carlo

function QSearch Simulator($N, N_{\text{Sample}}, t_{\text{actual}}$)
  (success, queries) $\leftarrow$ ClassicalSampler($N, N_{\text{Sample}}, t_{\text{actual}}$)
  if success **then**
    return queries
  $Q_{\max} \leftarrow \alpha\sqrt{N}$ ▷ $\alpha = 9.2$
  $r \leftarrow 0$, queries $\leftarrow 0$
  while TRUE **do**
    $m \leftarrow \frac{6}{5}, Q_{\text{sum}} \leftarrow 0$
    $j \leftarrow$ draw an integer from $[0, m)$ uniformly.
    while $Q_{\text{sum}} + j \leq Q_{\max}$ **do**
      success $\leftarrow$ GroverSimulator(N, t_{actual}, j)
      $Q_{\text{sum}} \leftarrow Q_{\text{sum}} + j + 1$
      queries $\leftarrow$ queries $+2j + 1$
      if success **then**
        return queries $+N_{\text{Sample}}$
      else
        $m \leftarrow \min(\lambda m, \sqrt{N})$ ▷ $\lambda \leftarrow \frac{6}{5}$
        $j \leftarrow$ draw an integer from $[0, m)$ uniformly.
function ClassicalSampler($N, N_{\text{Sample}}, t_{\text{actual}}$)
  $p^C_{\text{success}} \leftarrow t_{\text{actual}}/N$
  for $k = 1$ to N_{Sample} **do**
    if $p^C_{\text{success}} \geq$ rand() **then**
      return (TRUE, k)
  return FALSE
function GroverSimulator(N, t_{actual}, j)
  $p^Q_{\text{success}} \leftarrow \sin^2\left((2j+1) \cdot \arcsin(\sqrt{t_{\text{actual}}/N})\right)$
  if $p^Q_{\text{success}} \geq$ rand() **then**
    return TRUE
  else
    return FALSE

Figure 3.1: The average runtime of QSearch Simulator compared to the expected runtime of QSearch. The search space size was varied as $N \in \{10^3, 2 \cdot 10^3, \ldots 10^6\}$, and for each fixed N, the number of solutions was sampled 10^3 times uniformly from $t \leq N/4$. The average is taken over fractions of t/N.



Robert Derichs [25] has previously explored Grover's Search from a Bayesian perspective. His work laid the groundwork for this line of research by providing numerical evidence that incorporating Bayesian reasoning can improve the runtime performance of Grover's Search with unknown number of solutions. In that study, a full prior distribution was maintained and updated at each step, which introduced a scalability limitation– restricting the search space to sizes up to $N = 10^4$ due to computational constraints. In this work, we extend the Bayesian approach to Grover's Search by introducing particle filtering as a scalable alternative, as detailed in the next section.

3.2 Methodology

In this section, we formulate how Grover's Search can be interpreted from a Bayesian inference perspective, given a prior distribution $p(t)$ over the potential number of solutions in a search problem.

In the following, we first review the components of Bayes' rule as applied to Grover's Search in Section 3.2.1. Next, in Section 3.2.2, we will explain the necessity of Particle Filtering for this approach, propose an algorithm, termed *Particle-Guided Grover's Search* that employs Particle Filtering. Finally, in Section 3.3 we present the results.

3.2.1 Grover Iterations with Bayesian Updates

To apply Bayesian inference to Grover iterations with an unknown number of marked elements, the first step is to identify the components of the Bayesian model. The problem can be stated as: assuming a prior distribution over the possible values of t, the unknown number of marked elements, which iteration count j_k in the k-th Grover cycle maximizes the likelihood of measuring a marked element, given that all previous Grover cycles with iteration counts $j_1, j_2, \ldots, j_{k-1}$ have resulted in failure?

The set of hypotheses comprises all possible numbers of marked elements, here taken as all non-negative integers less than $N/4$, where N denotes the size of the search space.[1] The first component to consider is the prior distribution over the potential number of marked elements. The prior distribution, denoted by $p(t)$ is a discrete distribution for the set of hypotheses $t \in [1, \ldots N/4]$.

The second component is the likelihood, which is based on "the piece of evidence" obtained from a failed Grover cycle. Let j denote the number of iterations executed in the most recent Grover cycle, which resulted in failure. We denote this failure event by $\neg j$, indicating that j is unlikely to be the optimal number of iterations for the given search problem. The measurement model or the likelihood describes how likely it is to observe $\neg j$ assuming the hypothesis t is true, and corresponds to the projection of the initial state $|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$, rotated by G^j, onto the superposition of unmarked states, and is expressed as follows:

$$p(\neg j|t) = \cos^2\Big((2j+1)\arcsin\sqrt{t/N}\Big), \tag{3.1}$$

where $\arcsin\sqrt{\frac{t}{N}}$ corresponds to $\frac{\theta}{2}$, given the actual number of marked elements is t.

The posterior follows from Bayes' rule as

$$p(t|\neg j) = \frac{p(\neg j|t)p(t)}{\sum_{t=1}^{N/4} p(\neg j|t)p(t)}.$$

Finally, an update rule is needed to propose a candidate number of Grover iterations for the next round. Numerical simulations indicate that selecting the number of iterations by maximizing the expected success probability per iteration yields superior performance compared to alternative objectives. This quantity is defined as

$$\bar{p}_{\text{success}}(j) := \frac{\mathbb{E}_{p(t)}[p_{\text{success}}(j)]}{j} = \frac{1}{j}\sum_{t=1}^{N/4} p(t)\sin^2\Big((2j+1)\arcsin\sqrt{t/N}\Big),$$

where $p(t)$ denotes the current belief over the number of marked elements, and $\sin^2(\ldots)$ corresponds to the projection of the initial state $|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$, rotated by G^k, onto the superposition of marked states.

Table 3.1 summarizes the components of Grover iterations within the Bayesian framework.

[1] If the number of solutions is larger *i.e.* $t \geq N/4$, classical random sampling may be preferable to Grover's Search.

Table 3.1: The components of Bayesian updates for Grover's Search

Prior	$p(t)$
Likelihood	$p(\neg j\|t) = \cos^2\big((2j+1)\theta/2\big) \quad \text{with} \quad \theta/2 = \arcsin\sqrt{t/N}$
Posterior	$p(t\|\neg j) = p(\neg j\|t)p(t)/\sum_{t=1}^{N/4} p(\neg j\|t)p(t)$
Update rule	$\max_j[\bar{p}_{\text{success}}(j)] \quad \text{with} \quad \bar{p}_{\text{success}}(j) = \frac{1}{j}\sum_{t=1}^{N/4} p(t)\sin^2\big((2j+1)\theta/2\big)$

3.2.2 Particle-Guided Grover's Search

In previous section, we discussed the components of the Bayesian updates for Grover's Search. All distributions involved in this method, in principle, scale up to size $N/4$. This presents challenges for applying Bayesian inference in this context, as it would not only increase the classical runtime of the algorithm to $\mathcal{O}(N^2)$, but also require $\mathcal{O}(N)$ bits to store and update the full prior distribution at each step. In the following section, we address this limitation by introducing *Particle-Guided Grover's Search* – a hybrid algorithm that combines Grover's Search with Bayesian filtering, specifically particle filtering, to achieve computational efficiency. Applying Particle Filtering to Grover's Search offers an additional advantage: by design, Bayesian filtering methods – including Particle Filtering – sequentially estimate the posterior distribution using both current and past observations. This means that all failed Grover cycles up to the current step are naturally incorporated into the belief update process. As a result, the prior at each step becomes "informative", which can enhance the performance of *Particle-Guided Grover's Search*.

Before applying Particle Filtering to Grover's Search, it is important to clarify a key distinction. Bayesian Filtering is traditionally designed to estimate the evolving state of a dynamic system over time. In contrast, the system we address here – the search space – is inherently static.

Nevertheless, there exists substantial research on adapting Sequential Particle Filters to static models (*e.g.*, [19, 20, 26, 39]). Some approaches, such as those in [19, 20], reinterpret the static setting by introducing artificial dynamics. While effective in some cases, such dynamic formulations are unnecessarily complex for our specific application.

What we do here instead, is an adaptation of Particle Filtering, Sequential Importance Resampling (SIR), to our static inference problem. We begin by drawing an initial set of particles from the prior and assigning them uniform weights (**Initialization**). As new evidence of the form $\neg j_k$ becomes available, we update the weights of the existing particles recursively using the likelihood (**Recursive Update**). To reduce weight degeneracy, we include a resampling step, which regenerates particles when the weight variance becomes too large (**Resampling**).

The subroutine *Particle Filtered Posterior Estimation* (Algorithm 10) implements the Sequential Importance Resampling (SIR) algorithm, adapted specifically for Grover's Search. It assumes that the initial set of particles has already been generated in a preceding step, referred to as **Initialization of Particles**. The likelihood function is defined as $p(\neg j|t) = \cos^2\left((2j+1)\theta/2\right)$ with $\theta/2 = \arcsin\sqrt{t/N}$. The function takes the particles and their corresponding weights from previous step, $\{t^{(i)}, w_{k-1}^{(i)}\}_{i=1}^{L}$, and returns particles (potentially new, if resampling is triggered) with the updated weights, $\{t^{(i)}, w_{k}^{(i)}\}_{i=1}^{L}$.

Algorithm 10 Particle Filtered Posterior Estimation

Input: 1) Measurement model (likelihood): $p(\neg j \mid t)$
2) Recent observation: $\neg j_k$
3) Particles and their corresponding weights from previous step: $\{t^{(i)}, w_{k-1}^{(i)}\}_{i=1}^{L}$

Output: Particles (potentially new) and the corresponding updated weights: $\{t^{(i)}, w_{k}^{(i)}\}_{i=1}^{L}$

1: **function** PosteriorEstimator($p(\neg j \mid t)$, $\neg j_k$, $\{t^{(i)}, w_{k-1}^{(i)}\}_{i=1}^{L}$)
2: **for** $i = 1, \ldots, L$ **do**
3: Update weights: $\tilde{w}_k^{(i)} \leftarrow p(\neg j_k \mid t^{(i)}) w_{k-1}^{(i)}$
4: Then normalize: $w_k^{(i)} \leftarrow \tilde{w}_k^{(i)} / \sum_{j=1}^{L} \tilde{w}_k^{(j)}$
5: Compute effective sample size: $n_{\text{eff}} = 1/\sum_{i=1}^{L} (w_k^{(i)})^2$
6: **if** $n_{\text{eff}} < L/10$ **then** ▷ *Resampling step.*
7: $\{t^{(i)}, w_k^{(i)}\}_{i=1}^{L} \leftarrow$ Resampling($\{t^{(i)}, w_k^{(i)}\}_{i=1}^{L}$) (Algorithm 6)
8: **return** $\{t^{(i)}, w_k^{(i)}\}_{i=1}^{L}$

Pseudocode 11 presents the *Particle-Guided Grover's Search* algorithm. The procedure begins by drawing a set of particles from the prior distribution (line 4). Initially, the particles are assigned uniform weights. Using the approximation provided by the particles and their associated weights, a candidate number of Grover iterations is proposed through a subroutine called *Bayesian Update Rule for Grover's Search* (Algorithm 12). If this Grover cycle fails to find a marked element, the algorithm recursively updates the particle weights to incorporate the new evidence from the failed attempt. A resampling step, implemented as part of Algorithm 10, may regenerate a new set of particles if weight degeneracy is detected. This process continues until a marked element is successfully identified.

Algorithm 11 Particle-Guided Grover's Search

Input: 1) Search space: $[N]$,
2) $\lceil \log_2 N \rceil + 1$ qubits,
3) A black box oracle O_f on $[N]$, as described in Algorithm 1.
4) Prior: prior(t)
5) Number of particles: L.
Output: A marked element

function PGGS($[N]$,$\lceil \log_2 N \rceil + 1$ qubits, O_f, prior(t), L)
  $0 \leftarrow k$ ▷ *Initialize time step counter.*
  while TRUE **do**
    if $k = 0$ **then** ▷ *Initialization of Particles.*
      for $i = 1, \dots L$ **do**
        Draw initial particles according to $t^{(i)} \sim$ prior(t)
        Set the initial weights to $w_0^{(i)} \leftarrow 1/L$
      $j_0 \leftarrow$ BAYESDECISION($\{t^{(i)}, w_0^{(i)}\}_{i=1}^L$)
    else
      $\{t^{(i)}, w_k^{(i)}\}_{i=1}^L \leftarrow$ POSTERIORESTIMATOR($p(\neg j \mid t), \neg j_{k-1}, \{t^{(i)}, w_{k-1}^{(i)}\}_{i=1}^L$)
      $j_k \leftarrow$ BAYESDECISION($\{t^{(i)}, w_k^{(i)}\}_{i=1}^L$)
    $y \leftarrow$QUANTUMSEARCH($[N]$, $\lceil \log_2 N \rceil + 1$ qubits, $\mathcal{O}_f, j_k$) ▷ *One Grover cycle.*
    if y is marked **then**
      return y
    $k + 1 \leftarrow k$

Algorithm 12 Bayesian Update Rule for Grover's Search

Input: Set of particles and their corresponding weights: $\{t^{(i)}, w_k^{(i)}\}_{i=1}^L$
Output: Candidate number of Grover iterations for next round: j_{chosen}

function BAYESDECISION($\{t^{(i)}, w_k^{(i)}\}_{i=1}^L$)
  `averagePsuccess` $\leftarrow 0$
  Define $\bar{p}_{\text{success}}(j) = \frac{1}{j} \sum_{i=1}^L [w_k^{(i)} \sin^2\left((2j+1) \arcsin \sqrt{t^{(i)}/N}\right)]$
  for $j = 1, \dots \lceil \sqrt{N} \rceil$ **do**
    if $\bar{p}_{\text{success}}(j) >$ `averagePsuccess` **then**
      `averagePsuccess` $\leftarrow \bar{p}_{\text{success}}(j)$
      $j_{\text{chosen}} \leftarrow j$
  return j_{chosen}

To benchmark the Particle-Guided Grover's Search (Algorithm 11), we perform a Monte Carlo simulation. The subroutine QUANTUMSEARCH is replaced with GROVERSIMULATOR, as defined in Algorithm 9. A variable `queries` is introduced to track the number of queries made to the classical oracle function f. For each Grover cycle with j_k iterations, a total of $2j_k + 1$ queries

are counted in `queries`; $2j_k$ for implementation of the quantum oracle, and one additional query accounts for verifying the measurement outcome.

3.3 Results

In this section, we present the results of benchmarking *Particle-Guided Grover's Search* (PGGS) in comparison to QSearch. Various priors are considered, including truncated Gaussian mixtures, and a uniform prior. The implementation is carried out in *Julia*, and the source code is available at repository [29]. Finally, we discuss the findings in section 3.3.4.

To compare the performances of the two algorithms, besides the uniform distribution, we consider discrete distributions derived from *truncated Gaussians*. A truncated Gaussian distribution over real random variable X with mean μ, variance σ^2, and support restricted to the interval $[a, b]$ is defined as

$$\mathcal{N}_d(x|\mu,\sigma^2)_{[a,b]} := \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} \cdot \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\Phi(\frac{b-\mu}{\sigma})-\Phi(\frac{a-\mu}{\sigma})} & \text{for } x \in [a,b] \\ 0 & \text{otherwise.} \end{cases}$$

Here $x \in \mathbb{R}$, and $\Phi(.)$ is the cumulative distribution function (CDF) of the standard normal distribution. The term $\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})$ in the denominator serves as a normalization constant, ensuring that the truncated Gaussian distribution integrates to one over the interval $[a, b]$. Since our focus is solely on benchmarking the performance of Grover's Search within the Particle Filtering framework against QSearch, we restrict the number of solutions t to the range $[1, \lceil N/4 \rceil]$. This excludes the regime where classical sampling could potentially outperform Grover's Search. Accordingly, we set $a = 1$ and $b = \lceil N/4 \rceil$ throughout this section. Further, we always assume $t \in \mathbb{Z}$, and therefore work exclusively with discrete distributions.

Next, we exploit *univariate Gaussian mixtures* to construct priors that are bimodal. A *univariate Gaussian mixture* is a probabilistic model composed of several Gaussian distributions defined over the same random variable – hence the term univariate. A univariate Gaussian mixture over real random variable X, with K components is defined as,

$$p(x) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x|\mu_k, {\sigma_k}^2)$$

with $x \in \mathbb{R}$, and π_k being the mixing weights, with $\pi_k \geq 0$ and $\sum_{k=1}^{K} \pi_k = 1$. Again, we employ univariate Gaussian mixtures defined over discrete distributions due to $t \in \mathbb{Z}$. For simplicity, we omit explicit references to the variable domain and the truncated interval from this point onward.

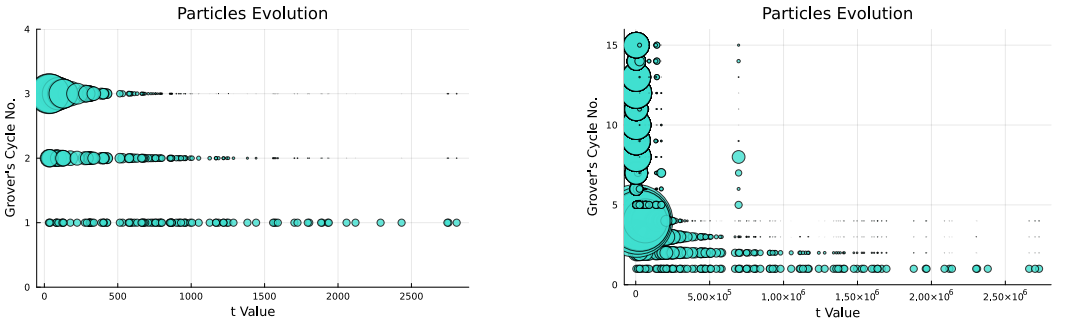
3.3.1 Particles Evolution in PGGS

To demonstrate how the particles, or candidate values for t, evolve over steps of Grover cycles in PGGS we consider two cases given in table 3.2. Here N denotes the size of search space, t_{actual} the true number of solutions and L the number of particles used in PGGS. Case 1 is easier for PGGS, than case 2, since the fraction t/N for the first is much larger, *i.e.* $60/10^6$ versus $60/10^{12}$.

Table 3.2: The two cases considered to demonstrate the particle evolution in PGGS. The evolution is depicted in figure 3.2

Case No.	N	t_{actual}	L	μ	σ
1	10^6	60	100	100	10^3
2	10^{12}	60	100	100	10^6

Figures 3.2 presents how the particles and their corresponding weights evolve by steps of Grover cycles. For the first cycle, all the particles have similar weights and as more cycles fail to find a solution, the weights of particles with values close to t_{actual} grows large, and the weights of particles with large t values shrinks. The procedure continues until the algorithm finds a solution. Figure 3.2b illustrates the degeneracy phenomenon, which becomes apparent around the 5-th cycle. At this point, the variance of the particle weights becomes large, triggering the resampling step. As a result, new particles are drawn from the current weighted distribution, and their weights are reset to uniform values.



(a) Case 1 with prior $\mathcal{N}_d(100, 10^6)$

(b) Case 2 with prior $\mathcal{N}_d(100, 10^{12})$

Figure 3.2: Evolution of particles through Grover cycles for two cases as described in table 3.2.

3.3.2 Comparison

In this section, we evaluate the performance of PGGS in comparison to the QSearch simulator. The evaluation is conducted in a setting where the true number of solutions is sampled from the same prior distribution employed by PGGS. In the next section, we examine the scenario where there is a mismatch between the distribution from which the actual number of solutions is drawn and the prior assumed by PGGS.

Here to benchmark performance, we consider two types of average query counts as key indicators:

1. For each experiment, we draw N_{sample} values of t_{actual} from a given prior. For each sampled value, both PGGS and QSearch are executed and the number of queries is computed. The averages is then taken over unique values of the fraction t/N according to the sampled values for t_{actual}, and the mean is reported for each algorithm.

2. The second indicator involves computing an overall average across all t/N fractions for a fixed prior, thereby yielding a single performance metric per algorithm for each prior.

We benchmark both algorithms under various prior distributions and present the results accordingly. Since the comparative trends remain consistent across different search space sizes, we fix the search space to $N = 10^6$. Nevertheless, simulations with larger spaces—up to 10^{10}—are still feasible on a standard laptop. The number of particles is set to $L = 100$; however, we observe that even with as few as 10 particles, PGGS maintains a performance advantage over QSearch. The number of samples drawn for t_{actual}, denoted N_{sample}, is fixed at 10^3 , as increasing the sample size does not significantly affect the observed behavior of the algorithms.

Table 3.3: Different prior distributions considered to benchmark PGGS against QSearch.

Case No.	Prior	Varied Parameter
1	Unimodal: $\mathcal{N}_d(\mu, \sigma^2)$	σ and μ
2	Bimodal: $\pi_1\mathcal{N}_d(\mu_1, {\sigma_1}^2) + \pi_2\mathcal{N}_d(\mu_2, {\sigma_2}^2)$	π_1/π_2
3	Uniform: $\mathcal{U}\{1, \lceil N/4 \rceil\}$	N

Case 1: Unimodal Distribution

Figures 3.3 and 3.5 present a comparison of average query counts between PGGS and the QSearch simulator under varying standard deviations and means of a unimodal truncated Gaussian prior. The results are shown as a function of the ratio t/N. For each case, the values of t_{actual} are drawn from the corresponding prior distribution, and the average query count is computed for each unique value of t/N.

We investigate the PGGS's performance under varying levels of spread by considering the prior distribution $\mathcal{N}_d(10, \sigma^2)$, with $\sigma \in \{1, 10, \ldots, N/10\}$ increasing on an exponential scale. The mean is fixed at $\mu = 10$ to challenge the algorithm in a more demanding regime.

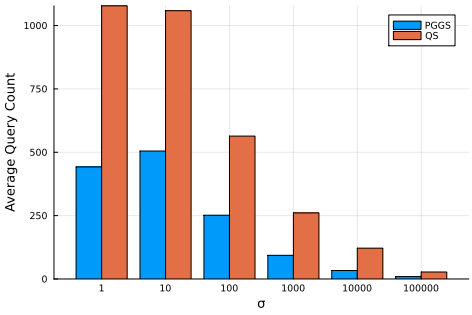
The results indicate that PGGS maintains consistent performance across different spreads, except when $t \simeq 1$. To better visualize this behavior, Figure 3.3b restricts the sampled values of t to ≤ 100 using rejection sampling. Across the relevant regime ($t \leq N/4$, excluding $t \simeq 1$), PGGS exhibits query counts that are largely insensitive to the spread of the prior.

Overall, the results demonstrate a clear improvement in average query efficiency for PGGS over QSearch. Figure 3.3a compares the total average query counts for both methods across different distributions, showing that PGGS achieves roughly a 50% reduction in average queries relative to QSearch.

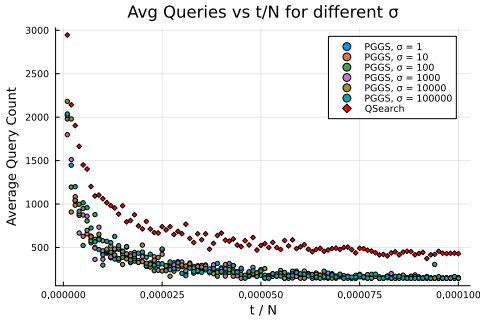
Figure 3.4 illustrates the probability mass functions corresponding to two extreme cases from the variance analysis.

To study how shifting the mean of the Gaussian distribution influences the performance of PGGS, we run another set of simulations. We considered the prior distribution $\mathcal{N}_d(\mu, 10^6)$, with $\mu \in \{1, 10, \ldots, N/100\}$ increasing on an exponential scale.

The results indicate that PGGS keeps the performance advantage across different means, except when $t \simeq 1$. To better visualize this behavior, Figure 3.5b restricts the sampled values of t to ≤ 100 using rejection sampling. Across the relevant regime ($t \leq N/4$, excluding $t \simeq 1$), PGGS exhibits query counts that are largely insensitive to shifting of the prior.

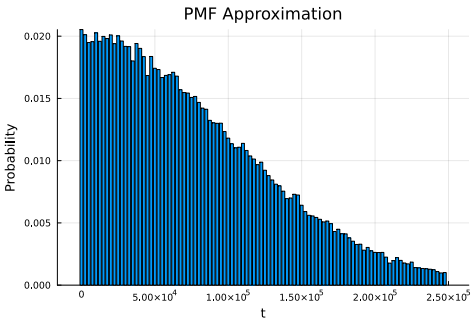


(a)

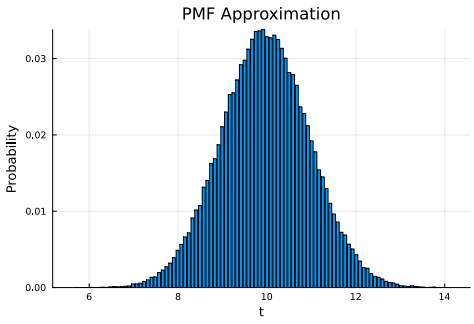


(b) To see a zoomed-in view, the values of t_{actual} were restrictedly sampled from $[1, 100]$.

Figure 3.3: The comparison of PGGS versus QSearch average queries. The prior distribution was taken as $\mathcal{N}_d\left(10, \sigma^2\right)$, varying $\sigma \in \{1, 10, \ldots, N/10\}$ with exponential steps. The settings chosen for this simulation were: $N = 10^6$, $L = 100$, $N_{\text{sample}} = 1000$.



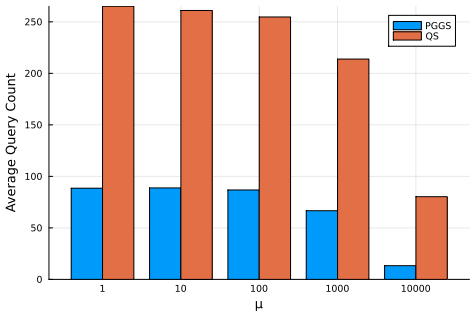
(a)



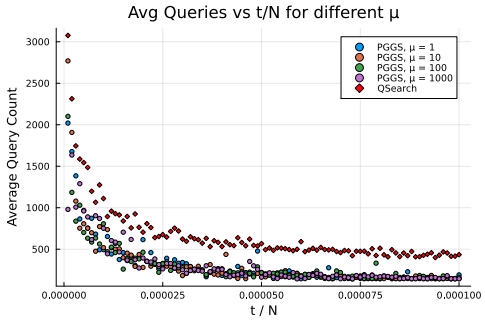
(b)

Figure 3.4: The comparison of probability mass function for two prior distributions $\mathcal{N}_d\left(10, 10^{10}\right)$ (left) and $\mathcal{N}_d(10, 1)$ (right). The size of search space was set to $N = 10^6$.

The numerical simulations again suggest an improvement in average query efficiency for PGGS over QSearch. Figure 3.5a compares the total average query counts for both methods across different distributions, showing that PGGS achieves more than 50% reduction in average queries relative to QSearch.



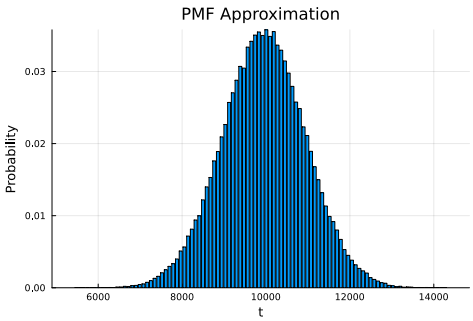
(a)



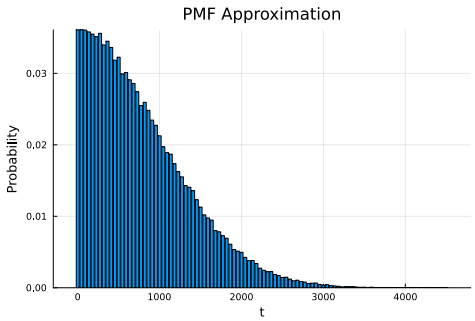
(b) To see a zoomed-in view, the values of t_{actual} were restrictedly sampled from $[1, 100]$.

Figure 3.5: The comparison of PGGS versus QSearch average queries. The prior distribution was taken as $\mathcal{N}_d(\mu, 10^6)$, with varying $\mu \in \{1, 10, \ldots, N/100\}$ with exponential steps (for figure 3.5b the upper bound was set to $N/1000$ to facilitate more efficient rejection sampling.). The settings chosen for this simulation were: $N = 10^6$, $L = 100$, $N_{\text{sample}} = 1000$.

Figure 3.6 depicts the probability mass functions corresponding to two extreme cases from the mean shifting study.



(a)



(b)

Figure 3.6: The comparison of probability mass function for two prior distributions $\mathcal{N}_d(10^4, 10^6)$ (left) and $\mathcal{N}_d(1, 10^6)$ (right). The size of search space was set to $N = 10^6$.

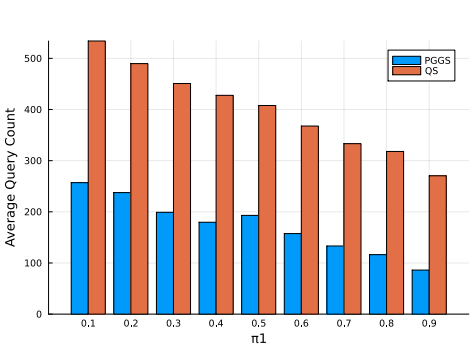
Case 2: Bimodal Distribution

Another relevant setting for comparing the performance of PGGS and QSearch is when the prior distribution is bimodal. The insights obtained in the previous section regarding the effects of shifting the mean and varying the standard deviation are expected to extend to this case as well.

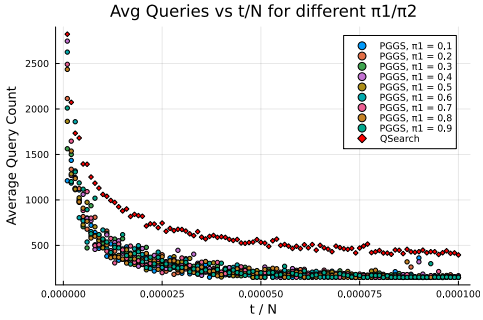
Therefore, we focus here on studying the impact of varying the mixing ratio between the two modes.

To this end, we consider bimodal priors of the form $\pi_1\mathcal{N}_d\big(500, 10^4\big) + \pi_2\mathcal{N}_d\big(10, 10^4\big)$, with $\pi_1 \in \{0.1, 0.2, \ldots, 0.9\}$ and $\pi_2 = 1 - \pi_1$.

Figure 3.7 presents the results. Once again, PGGS outperforms QSearch across all priors (except for $t \simeq 1$), both in terms of average query counts per prior (Figure 3.7a) and per unique ratio t/N (Figure 3.7b). The observed improvement is approximately 50%.



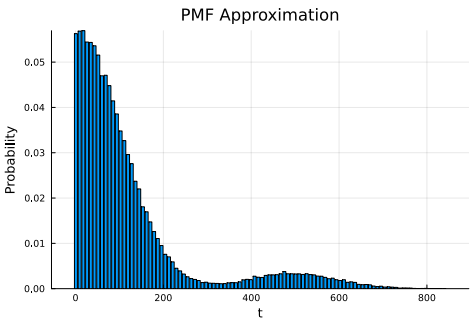
(a)



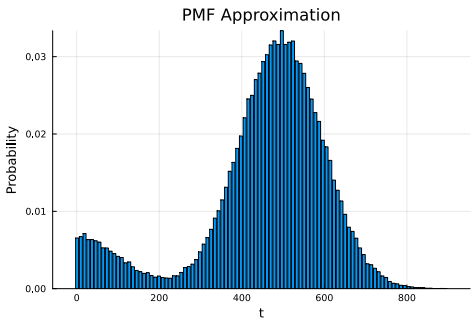
(b) To see a zoomed-in view, the values of t_{actual} were restrictedly sampled from $[1, 100]$.

Figure 3.7: The comparison of PGGS versus QSearch average queries. The prior distribution was taken as $\pi_1\mathcal{N}_d\big(500, 10^4\big) + \pi_2\mathcal{N}_d\big(10, 10^4\big)$, with varying $\pi_1 \in \{0.1, 0.2, \ldots, 0.9\}$. The settings chosen for this simulation were: $N = 10^6$, $L = 100$, $N_{\text{sample}} = 1000$.

Figure 3.8 illustrates the probability mass functions corresponding to two extreme cases from the mixing ratio analysis.



(a)



(b)

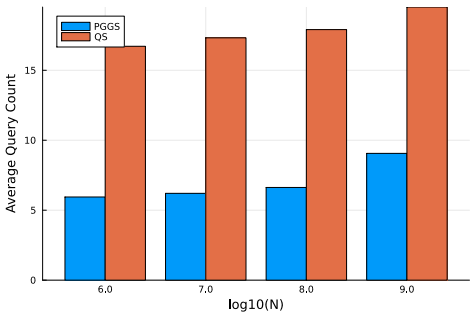
Figure 3.8: The comparison of probability mass function for two prior distributions $0.1\mathcal{N}_d\big(500, 10^4\big) + 0.9\mathcal{N}_d\big(10, 10^4\big)$ (left) and $0.9\mathcal{N}_d\big(500, 10^4\big) + 0.1\mathcal{N}_d\big(10, 10^4\big)$ (right). The size of the search space was set to $N = 10^6$.

Case 3: Uniform Distribution

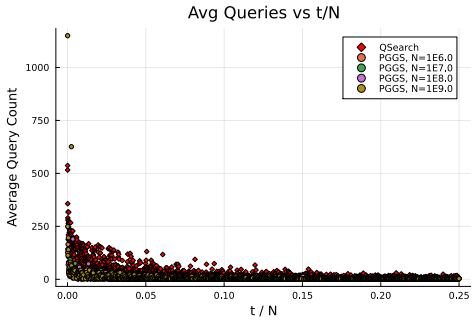
The final distribution we consider to demonstrate PGGS's advantage over QSearch is the uniform distribution. This scenario is particularly noteworthy, as it demonstrates that even starting from a uniform prior, PGGS can incorporate evidence gathered during the search process and, in some cases, outperform methods like QSearch that remain agnostic throughout.

To study this scenario, we use uniform priors $\mathcal{U}\{1, \lceil N/4 \rceil\}$. The values of N were varied over $\{10^6, 10^7, 10^8, 10^9\}$. We first note that the average query counts for both algorithms are significantly lower than previous cases (see Figure 3.9a). This is expected, as the Gaussian priors in earlier experiments were deliberately designed to place more weight on difficult regimes (*e.g.*, small t/N values). In contrast, the uniform prior assigns equal probability to all values, including those corresponding to easier cases (*i.e.*, larger t/N), reducing the overall query count.

Figure 3.9b illustrates the overall performance trends for both algorithms under this setting.



(a) Average query counts for PGGS and QSearch versus N.



(b)

Figure 3.9: The comparison of PGGS versus QSearch average queries. The prior distribution was taken as $\mathcal{U}\{1, \lceil N/4 \rceil\}$ with varying $N \in \{10^6, 10^7, 10^8, 10^9\}$. The settings chosen for this simulation were: $L = 100$, $N_{\text{sample}} = 1000$.

It is remarkable that PGGS still achieves a relative improvement of approximately 50% over QSearch, despite not using a classical sampling subroutine like QSearch does.

This setting is also conceptually important: it suggests that PGGS can offer an improvement even in the complete absence of prior information about the likely number of solutions in the search space – by employing a uniform prior. This approach appears consistent with the *Principle of Indifference*, which recommends assigning equal degrees of belief to all hypotheses when no distinguishing evidence is available. An important question remains, however: can PGGS still provide an improvement using a uniform prior that does not match the distribution from which the actual number of solutions is drawn? We address this question in the next section.

3.3.3 Testing PGGS under Prior Mismatch

In this section, we study the cases where there is a mismatch between the distribution from which the actual number of solutions is drawn, and the prior used by PGGS.

In the Bayesian framework, when genuine prior information is unavailable, it is common to use a *non-informative prior* to minimize the influence of prior assumptions on the inference. The

simplest form of such a prior is the uniform distribution over the parameter space, which reflects equal weighting across all parameter values.

Nevertheless, we explored scenarios in which the number of solutions was sampled from a uniform distribution, yet the prior employed was sharply biased – *e.g.*, a unimodal distribution – contrary to Bayesian principles. In these cases, it is unsurprising that PGGS exhibited poorer performance compared to QSearch.

We also examined scenarios in which the number of solutions was drawn from a unimodal distribution $\mathcal{N}_d(\mu, \sigma^2)$, while PGGS assumed a uniform prior. We divided these into three distinct regimes:

1. $\mu = N/4$
2. $\mu = N/8$
3. $\mu = 10$

For the first two regimes, we examined search spaces of size $N \in \{10^6, \ldots 10^9\}$, using exponentially increasing increments. To clearly illustrate the impact of spread, each simulation employed two distinct deviations – one wide and one relatively narrow. All other simulation parameters (*i.e.* N_{sample} and L) were held constant as before.

The simulation results indicate that when $\mu = N/4$, the average number of queries for both PGGS and QSearch are nearly identical, with PGGS showing a modest advantage. This slight improvement is notable, as this regime typically favors classical sampling – a capability that PGGS lacks as a subroutine.

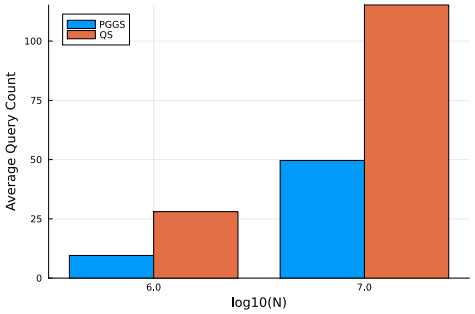
For the case $\mu = N/8$, the ratio of average query counts for PGGS relative to QSearch was approximately 0.5 – same as the advantage PGGS achieves when the prior aligns with the true distribution.

The most intriguing regime occurs when $\mu = 10$. Not only is this the regime where Grover's search achieves its maximum speedup, but previous simulations (Figure 3.3b) also have revealed that PGGS's performance becomes highly sensitive to variations in the spread. For computational efficiency, we restricted our analysis to $N = 10^6$ and 10^7. For each N, we tested two spread values, $\sigma = 10^3$ and 10^5, using $L = 100$ particles as before. The number of samples for the wider spread was maintained at 1000, while for the narrower spread it was reduced to 100 to expedite the process.

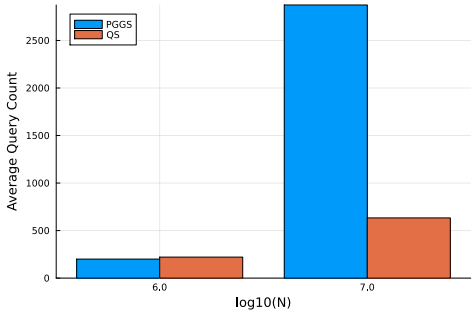
Figure 3.10 illustrates the average performance for both algorithms under this setting.

The case with $\sigma = 10^5$ (left), remains manageable for PGGS, and the average query count for PGGS is approximately half that of QSearch. However, when the distribution is relatively narrow, with $\sigma = 10^3$ (right), PGGS performs significantly worse than QSearch. This outcome is expected, as PGGS, starting from a uniform prior, takes longer to identify that the optimal number of iterations is large. In contrast, QSearch, by exponentially increasing the interval from which the candidate number of iterations is drawn, reaches this optimal value much more quickly.

One may still question why the performance of PGGS changes drastically from $N = 10^6$ to $N = 10^7$ for the narrower spread. The hypothesis is that, since the number of actual solutions is sampled from the same distribution, namely $\mathcal{N}_d(10, 10^6)$, the uniform distribution $\mathcal{U}\{1, \lceil N/4 \rceil\}$ deviates more from $\mathcal{N}_d(10, 10^6)$ when $N = 10^7$ compared to when $N = 10^6$.



(a) Average query counts for PGGS and QSearch for $\sigma = 10^5$ with $N_{\text{sample}} = 1000$.



(b) Average query counts for PGGS and QSearch for $\sigma = 10^3$ with $N_{\text{sample}} = 100$.

Figure 3.10: The comparison of PGGS versus QSearch average queries with a mismatched prior. In this setup, the actual number of solutions was sampled from a unimodal distribution $\mathcal{N}_d(10, \sigma^2)$, while PGGS assumed a uniform prior. The search-space sizes considered were $N \in \{10^6, 10^7\}$, and all simulations used $L = 100$ particles.

To analyze this quantitatively, we introduce a statistical measure known as the *Kullback-Leibler (KL) divergence*, which in this context quantifies how much information is lost when one distribution, P, is approximated by another, Q. In the discrete case, the KL divergence is defined as:

Definition 7. *(Kullback-Leibler divergence or the relative entropy from Q to P)*

$$D_{KL}(P \parallel Q) := \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right),$$

with $\mathcal{X}$ being the sample space both distributions are defined over.

Alternatively, the KL divergence can be written as

$$D_{\text{KL}}(P \parallel Q) = \mathbb{E}_{x \sim P} \Big[\log \left(\frac{P(x)}{Q(x)} \right) \Big],$$

which represents the expected discrepancy in log-probability between the true distribution P and the approximation Q. The KL divergence is a nonnegative quantity that achieves its minimum value, namely zero, if and only if $P = Q$ almost everywhere. This reflects the fact that the best possible approximation of P by Q occurs when the two distributions assign exactly the same probabilities – and thus the same log-likelihoods – to every value x of a random variable. Any deviation from P leads to an increase in information loss, as quantified by the KL divergence. This justifies its alternative name, *relative entropy*.

There are two primary interpretations of relative entropy in this context:

1. **Information Gain in Bayesian Updating**: In the traditional Bayesian framework, relative entropy typically measures the information gain when updating from a prior to a posterior distribution upon observing new data. It quantifies how much our knowledge of the underlying system has increased after incorporating the evidence, reflecting the difference between the prior belief and the posterior distribution.

2. **Loss of Potential Information with Non-Informative Priors**: Alternatively, relative entropy can measure how much potential information is lost when we assume a non-informative or poorly matched prior relative to the true underlying distribution of solution counts in the search problem. In this context, it captures the inefficiency introduced by a prior that does not accurately reflect the characteristics of the search problem, resulting in a suboptimal utilization of available information.

Although these two interpretations are related in the sense that both involve a comparison between distributions, they are fundamentally different. The former focuses on the updating of beliefs in light of observed data, while the latter reflects the misalignment between prior assumptions and the true distribution, leading to a loss of potential information.

Here, we focus on the latter interpretation, using relative entropy to quantify the information that would be lost if non-informative priors fail to align with the true distribution. Mathematica code has been employed to compute the relative entropies for both cases:

$$D_{\mathrm{KL}}\Big(\mathcal{U}\{1, \lceil 10^6/4 \rceil\} \parallel \mathcal{N}_d\big(10, 10^6\big)\Big) = 1.04109 \times 10^4$$
$$D_{\mathrm{KL}}\Big(\mathcal{U}\{1, \lceil 10^7/4 \rceil\} \parallel \mathcal{N}_d\big(10, 10^6\big)\Big) = 1.04165 \times 10^6$$

The larger divergence in the second case may offer an explanation for the observed change in PGGS behavior. However, the exact impact of these quantities on PGGS performance remains an open question. Further investigation is required to explore how the relative entropy, in both of these senses, influences PGGS performance.

3.3.4 Discussions

In this chapter, we benchmarked PGGS, a Bayesian-inspired variant of Grover's search, against QSearch, a relevant approach from the literature. We evaluated their performance across a range of settings.

The results demonstrate that PGGS achieves an overall runtime improvement when the prior matches the distribution from which the number of solutions is drawn. Moreover, PGGS exhibits robustness with respect to the mean and standard deviation in the unimodal case, as well as to variations in the mixing weight ratio – except in the regime where $t/N \simeq 0$. Notably, simulations also indicate that PGGS continues to provide a performance advantage even when the prior does not match the true distribution of the number of solutions, with the same exception in the $t/N \simeq 0$ regime.

Moreover, PGGS incorporates a resampling step as part of the Particle Filtering process. It may be tempting to omit this step, given that the target distribution is ultimately concentrated on a single value of t. One might reason that a concentrated distribution is desirable, and thus, avoiding resampling could be more aligned with the search objective.

To evaluate this, we tested PGGS on several instances both with and without the resampling step (as defined in Line 6 of Algorithm 10). The results indicate that including the resampling step is, in fact, beneficial: PGGS consistently achieved lower average query counts when resampling was retained.

4 Runtime–coherence trade-offs for hybrid SAT-solvers

The present chapter has been published under the title *Runtime–coherence trade-offs for hybrid SAT-solvers* as a pre-print [31], and is published in *IEEE Transactions on Quantum Engineering* [32].

The manuscript has been written together with Sören Wilkening, Johan Åberg, and David Gross (with me as the corresponding, first author). Within the collaboration, I was responsible for obtaining the analytic runtime estimates, which constitute the main result of the paper. The numerical analysis and circuit implementation were carried out by Sören Wilkening.

The analysis was carried out in a mathematical framework where a random walk on the space of assignments is replaced by a simplified model – a random walk on $\mathbb{Z}$. While this model is a standard tool in the field, the pre-print Ref. [31] included a detailed appendix with a rigorous justification. Since the results reproduced here can be interpreted without this argument (by taking the $\mathbb{Z}$ random walk as a highly plausible model, rather than as a source of rigorous bounds), and because the appendix was written by Johan Åberg alone, we have omitted it from this thesis.

4.1 Abstract

Many search-based quantum algorithms that achieve a theoretical speed-up are not practically relevant since they require extraordinarily long coherence times, or lack the parallelizability of their classical counterparts. This raises the question of how to divide computational tasks into a collection of parallelizable sub-problems, each of which can be solved by a quantum computer with limited coherence time. Here, we approach this question via hybrid algorithms for the k-SAT problem. Our analysis is based on Schöning's algorithm, which solves instances of k-SAT by performing random walks in the space of potential assignments. The search space of the walk allows for "natural" partitions, where we subject only one part of the partition to a Grover search, while the rest is sampled classically, thus resulting in a hybrid scheme. In this setting, we argue that there exists a simple trade-off relation between the total runtime and the coherence-time, which no such partition based hybrid-scheme can surpass. For several concrete choices of partitions, we explicitly determine the specific runtime coherence-time relations, and show saturation of the ideal trade-off. Finally, we present numerical simulations which suggest additional flexibility in implementing hybrid algorithms with optimal trade-off.

4.2 Introduction

Consider a quantum algorithm that takes exponential time to run, but still offers a polynomial speed-up over the best classical method. Examples include Grover searches to brute-force a password or for finding the solution for a combinatorial optimization problem for which no classical heuristics exist. Fully quantum implementations might not be desirable for two reasons: (1) Quantum hardware that can sustain very long computations might not be available, and (2) quantum algorithms, like Grover's search, might not be easily amenable to parallelization. One is thus lead to the question of how to best break up such instances into a set of smaller, parallelizable subproblems that can individually be solved on quantum hardware.

We consider the well-known *satisfiability problem* with k the number of literals in each clause, (k-SAT) and focus particularly on 3-SAT since it provides an attractive test bed to investigate such questions. k-SAT is the archetypical combinatorial optimization problem and represents a class of use cases with considerable practical relevance. Moreover, there is a classical randomized algorithm [58, 59] due to Schöning, with a performance close to the best-known algorithms with provable performance, and which furthermore allows for a closed-form asymptotic run-time analysis. And indeed, the algorithm obtained by replacing the classical search of the Schöning-procedure by a Grover search [35] yields a *quantum-Schöning algorithm* with a quadratic improvement *vis-à-vis* its classical counterpart [3]. (Below, we will refer to quantum algorithms that arise this way as *Groverizations* of their classical versions).

However, such "fully quantized" Schöning's SAT-solvers cannot be performed in parallel, which arguably is a relevant feature for algorithms that run in exponential time. Hybrid schemes, based on "partial" Groverizations of Schöning's algorithm, where Grover search procedures are applied only to certain sub-routines, usually do allow for parallelizations.

Starting point of our analysis is the stochastic nature of Schöning's algorithm as a random walk. This point of view yields two classes of hybrid algorithms, where one class Groverizes the random choice of the initial state of the walk, while the other class Groverizes the randomness in the walk

itself. Within an established model of Schöning's algorithm, we optimize the resulting run-times by balancing the resources allocated to the subroutines.

4.2.1 Runtime–Coherence Time Trade-Offs

Before specializing the Schöning process, let us briefly outline the trade-offs between runtime and coherence time that can be expected for quantum search problems. Consider an algorithm that solves instances of size n with runtime $T(n)$. For exponential-time algorithms, we work with a somewhat coarser measure, the *(asymptotic) runtime rate*

$$\gamma = \lim_{n\to\infty} \frac{1}{n} \log T(n),$$

where we drop the base of the logarithm from here on; the base is 2 unless explicitly stated otherwise. In other words, $T \in \mathcal{O}^*(2^{\gamma n})$, where $\mathcal{O}^*$ denotes scaling behavior up to polynomial factors. The aim is to trade it off against the *coherence time* required to run the algorithm. If $C(n)$ is the longest time over which coherence has to be maintained while running the algorithm, then the *coherence time rate* is

$$\chi = \lim_{n\to\infty} \frac{1}{n} \log C(n).$$

Now restrict attention to search algorithms with classical runtime rate γ_{C}. A completely Groverized version runs with rate $\gamma_{\mathrm{G}} = \gamma_{\mathrm{C}}/2$. All of its runtime will be spent coherently, specifically executing Grover iterations. Therefore, $\chi_{\mathrm{G}} = \gamma_{\mathrm{C}}/2$ as well. We can visualize these two points in a "runtime rate vs coherence time rate"-chart, a mode of visualization that we will employ frequently (Fig. 4.1).

To achieve a trade-off between total runtime and coherence time, we will consider algorithms that apply Grover's procedure only to a subset of the search space. It is easy to see that any algorithm which results from such a procedure must have coordinates (χ, γ) that lie on or above the line segment

$$L = \{(\chi, \gamma_{\mathrm{C}} - \chi) \,|\, \chi \in [0, \gamma_{\mathrm{C}}/2]\}$$

that connects the purely classical point $(0, \gamma_{\mathrm{C}})$ to the completely Groverized one $(\gamma_{\mathrm{C}}/2, \gamma_{\mathrm{C}}/2)$.

Indeed, take a partial Groverization that achieves parameters (χ, γ). Then one can replace the Grover part by a classical search. The resulting classical algorithm will have parameters $(0, \gamma + \chi)$, because the Grover search contributed χ to the runtime rate, but its classical simulation will contribute 2χ instead. But if the initial parameters were below the line, i.e. if $\gamma < \gamma_{\mathrm{C}} - \chi$, then the resulting classical algorithm runtime rate is $\gamma + \chi < \gamma_{\mathrm{C}}$, contradicting the assumption that γ_{C} describes the classical complexity of the search.

It is not obvious that, conversely, every point on this optimal line segment can actually be realized, much less with an algorithm that is "natural" or easy to implement. Deciding the parameter ranges for natural partial Groverizations of Schöning's procedure is the main goal of this paper.

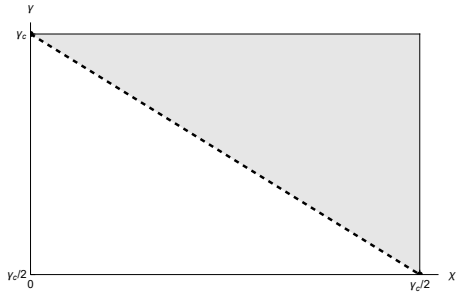


Figure 4.1: We will frequently visualize the behavior of algorithms by indicating their position in a "runtime rate vs coherence time rate"-chart. Classical algorithms require no coherence, and thus lie on the y-axis. In the example given, the point on the upper left hand side represents a classical probabilistic search with runtime rate γ_C. A completely Groverized version has coordinates $(\gamma_C/2, \gamma_C/2)$ (bottom right), meaning that it will spend its entire runtime coherently. Hybrid algorithms that use Grover only for a subset of the search space must lie in the shaded area above or on the dashed line segment connecting these two points.

Related work

Dunjko et al.[28] have previously considered partial Groverizations of Schöning's algorithm. They aimed to minimize a different metric: total number of clean qubits, rather than coherence time. In fact, they work in a highly constrained regime, where the number of available clean qubits only scales as cn, with $0 < c < 1$ and n the number of variables of the given 3-SAT formula. Surprisingly, they show that even this meager allotment of qubits in principle yields a speed-up compared to the classical Schöning's algorithm [1].

Despite the superficial similarities, their and our papers are quite different. We allow for qubit-counts that are quasi-linear in n, i.e. $\mathcal{O}(n \log n)$, reasoning that for exponential-time algorithms, coherence time and parallelizability might be more limiting than the number of available qubits. As it will turn out, the setting considered here can interpolate between the classical and the fully Groverized performance, while the runtime rates obtainable in [28] stay close to the classical ones. While [28] uses de-randomization techniques, our approach builds more directly on the original Schöning's algorithm. This makes our approach technically less involved, and it also makes the lessons learned more widely applicable, since the basic technique of using Grover search over a subset of all variables, directly generalizes to any NP problem, whereas de-randomizations to a larger extent rely on the particular structure of the problem at hand.

[1] According to [28], Supplemental Material Section B.4, the relative speed-up to the classical Schöning's rate is $f(c) = (1 - \log\sqrt{3})\beta(c)$, where the Beta function up to $\mathcal{O}(\frac{\log n}{n})$ is implicitly given as $A\beta(c)\ln\frac{1}{\beta(c)} + B\beta(c) = c$. As mentioned in [28] using a straightforward encoding of each trit into two qubits, one can assume $A = 10$ and $B = 50$. To be consistent with our encoding, we consider $\log_2 3$ qubits to encode a trit and then, calculate the maximum speed-up in the rate, i.e. $f(1) \approx 0.0028$.

4.3 Setting the stage

4.3.1 Schöning's algorithm

Here we provide a very brief introduction to the pertinent aspects of Schöning's 3-SAT solver. For a more thorough review, we refer the reader to [58, 59]. In the 3-SAT problem, we are given a collection of *clauses* $C_1, \ldots, C_L$ on n binary variables, where each clause is of the form $C_j = l_0^{(j)} \vee l_1^{(j)} \vee l_2^{(j)}$, and where each of the *literals* $l_0^{(j)}$, $l_1^{(j)}$, $l_2^{(j)}$ is one of the binary variables or its negation. The 3-SAT formula is the conjunction of all the given clauses, $C := \wedge_{j=1}^{L} C_j$, and the computational task is to determine whether there exists an assignment of the n binary variables that satisfies C. According to Schöning [58], an algorithm exists that, although with run-time that is exponential in n, can perform better than an exhaustive search through all potential assignments.

Schöning's algorithm (Alg. 13) depends on two parameters N, m to be determined later. It begins by choosing an assignment $x \in \{0,1\}^{\times n}$ uniformly at random. The algorithm then performs an m-step *random walk* over the space of n-bit strings (the inner loop in Alg. 13, from Line 5). In every step, it checks (according to a pre-determined order) all the clauses $C_1, \ldots, C_L$. If all are satisfied, then x is a solution and the algorithm terminates. Otherwise, it finds the first unsatisfied clause, chooses one of the three variables corresponding to the literals of that clause uniformly at random. The value of x is then updated, by negating that variable. This concludes the step. If no solution is found after m steps, the walk is terminated. Up to N such walks are attempted (the outer loop in Alg. 13), each time using a fresh uniformly random starting point x.

Algorithm 13 Schöning's Algorithm

```
1:  function SCHOENING(C_1, ..., C_L, N, m)
2:      for i = 1...N do
3:          x ← uniformly random value from {0,1}^{×n}
4:          for j = 1...m do
5:              if x satisfies C_1, ..., C_L then
6:                  return x
7:              else
8:                  k ← index of first unsatisfied clause
9:                  l ← index of one of the three variables occurring in C_k, chosen uniformly at
                    random
10:                 x ← x, with the l-th bit of x flipped
11:     return False
```

4.3.2 Analysis of the run-time of Schöning's algorithm

The analysis of the run-time of Schöning's algorithm is sketched in [58, 59] and a more in-depth analysis can be found in [49]. Here we follow a very similar line of reasoning, with our particular ansatz in mind. In the following, we present an overview, see Appendix [31] for a more detailed account.

Assume that there is at least one satisfying assignment $x^\star$. We first aim to lower-bound the probability that a given random walk finds a solution. Let x_0 be the (random) initial configuration, and x_l the one attained after the l-th step of the random walk. The probability that *any* solution is found during *any* step of the walk is certainly at least as large as the probability $P(x_m = x^\star)$ that the walk finds $x^\star$ at the m-th step. To analyze $P(x_m = x^\star)$, we follow in the steps of Schöning [58, 59], and focus on the evolution of the *Hamming-distance* $d_H(x_l, x^\star)$ between the current configuration and the selected satisfying assignment $x^\star$.

The fundamental insight is that if a clause C_k is violated at the l-th step, then at least one of the three variables that appear in C_k must differ between x_l and the satisfying assignment $x^\star$. Thus, the random flip decreases the Hamming distance to the solution with probability at least $1/3$:

$$P\big(d_H(x_{l+1}, x^\star) = d_H(x_l, x^\star) - 1\big) \geq \frac{1}{3}. \tag{4.1}$$

This suggests to pass from a description of the process on bit-strings to its projection $x_l \mapsto d_H(x_l, x^\star)$ onto $\mathbb{N}$. However, this would generally yield a process that would be no easier to analyze than the original one. One may for example note that although Schöning-process $(x_l)_l$ is Markovian on the space of bit-strings $\{0,1\}^{\times n}$, one cannot generally expect its projection $\big(d_H(x_l, x^\star)\big)_l$ to be Markovian on $\mathbb{N}$.

The general idea for the analysis is to replace (via a coupling) the true projection $\big(d_H(x_l, x^\star)\big)_l$ with another process $(d_l)_l$ on $\mathbb{Z}$, which is Markovian and which moreover upper-bounds the true Hamming-distance,

$$d_H(x_l, x^\star) \leq d_l. \tag{4.2}$$

More precisely, the Markov process $(d_l)_l$ is defined by the transition probabilities

$$P(d_{l+1} = d_l + 1) = \frac{2}{3}, \quad P(d_{l+1} = d_l - 1) = \frac{1}{3}. \tag{4.3}$$

The transition probabilities (4.3) can be interpreted as worst-case scenarios of each step in the Schöning process.

From the bound (4.2) it follows that $P(x_l = x^\star) \geq P(d_l \leq 0)$. In other words, the success probability of the Schöning-process is lower-bounded by the probability that the substitute-process d_l reaches 0.

Given the lower bound $P(d_m \leq 0)$ on the probability of success of each given walk, we expect at least one out of $N = 1/P(d_m \leq 0)$ walks to find $x^\star$. More precisely, if ϵ is the tolerated probability for failure, then the number of repetitions needed in order to find an existing solution satisfies

$$N \geq \frac{\log \epsilon}{\log(1 - P(d_m \leq 0))}. \tag{4.4}$$

The required number N of repetitions will be exponential in n. It is then common to take a coarser point of view, and only analyze the corresponding *rate*

$$\gamma := -\lim_{n\to\infty} \frac{1}{n} \log P(d_m \leq 0), \qquad \text{so that} \qquad N = \mathcal{O}^*(2^{\gamma n}), \tag{4.5}$$

where $\mathcal{O}^*$ denotes scaling behavior up to polynomial factors in order to achieve any constant probability of failure ϵ. With the choice $m = n$ (i.e., the termination time is equal to the number of variables) it turns out [58, 59] that $\gamma \leq \log \frac{4}{3} \approx 0.415$.

It is surprisingly technically difficult to rigorously derive the "global bound" $P(x_l = x^\star) \geq P(d_l \leq 0)$ from the "local bound" (4.1). However, the Markovian version $(d_l)_l$ of the Hamming distance random walk is commonly accepted as a good (in fact, conservative) model of the Schöning-process. In the main body of this paper, we will therefore phrase our arguments in terms of that model. More technical details on the relation between the two processes are given in Appendix [31].

4.3.3 Partial Groverizations: The general idea

For random walks we naturally tend to think of the randomness as being generated whenever needed, like when we assign the initial state, or make the random choices along the path. However, we can alternatively picture the walk as a deterministic process that is fed with an external random string S; a list from which it picks the next entry whenever a random choice is to be made. When the purpose of the walk is to find (an efficiently recognizable) solution to some computational problem, one can thus view the walk as a (deterministic) map that designates each input string S as being "successful" or "unsuccessful", in the sense of the walk reaching the satisfying solution $x^\star$ or not. To this mapping, we can in principle apply a Grover-search procedure, since the walk (as well as the solution-recognition procedure) can be performed via reversible circuitry, and can thus also be implemented coherently.

As described in the previous section, Schöning's algorithm proceeds with an initialization, followed by a random walk on the space of 2^n assignments. The initialization requires n bits of randomness, S_I, since the initial state is selected uniformly over all 2^n strings. A walk of length m requires a string S_W of $m \log 3$ bits to encode the needed randomness. The $\log 3$-factor is due to the fact that, at each step, the algorithm randomly selects which one of the three literals (of the first violated clause) should be flipped. An m-step Schöning-walk can thus be viewed as a map from $S = (S_I, S_W)$ to a binary variable that tells us whether a satisfying assignment has been reached or not.

With a coherent circuit that implements this map, we can thus replace the uniformly distributed random variable S, with a uniform superposition over a corresponding number of qubits, and proceed via standard Grover-iterations [35]. We would expect such a procedure to yield a satisfying assignment at a run-time that scales as $\mathcal{O}^*(2^{n\gamma_G})$ iterations, with $\gamma_G = \frac{1}{2} \log \frac{4}{3} \approx 0.208$ [3], i.e., the standard quadratic speed-up. Up to a few constant qubits, one needs $n+(\log 3+\log L)m$ qubits to encode this map as a quantum circuit, where L is the number of clauses in the 3-SAT formula (more details are given in Section 4.6). Since the number of clauses grows linearly in n for the regime of interest by the SAT phase-transition conjecture [18], and for the Schöning walk $m = n$, the space complexity of such encoding is $\mathcal{O}(n \log n)$.

The view of random walks as maps on random input strings opens up for the concept of partial Groverizations. Nothing would in principle prevent us from regarding only a *part* of the input string S as the input of the Grover-procedure, while keeping the rest of the string classical. Needless to say, one would generally expect the result to be less efficient than the "full" Groverization. However, the gain would be that the partial Groverization breaks the tasks into a collection of subproblems, each of which can be run in parallel on a quantum device that requires shorter coherence time.

Although it seems reasonable to expect that such a division in principle is always possible, one may also expect that it in general would be challenging to find a quantum circuit that implements it in an economical manner. (We can always resort to a full coherent circuit for S in its entirety, putting the "classical part" in a diagonal state.) However, there may be "natural" divisions of the process, which can be exploited. For Schöning's algorithm it is close to hand to consider the division $S = (S_I, S_W)$, i.e., the division of the required randomness into the initialization-part and the walk-part. One can thus consider two particularly natural classes of "partial" Groverizations of Schöning's algorithm. For one of these, the *Groverized Initialization (GI)*, the choice of the initial state is implemented coherently, while the walk is kept "classical". For the *Groverized Walk (GW)*, the choice of initial state is kept classical, while the walk itself is performed coherently.

As described in Section 4.3.2, the actual analysis is based on the random walk $(d_l)_l$ on $\mathbb{Z}$, rather than the true Schöning walk on strings in $\{0,1\}^{\times n}$. The idea is nevertheless the same; the required randomness is divided into the initialization and the walk *per se*, resulting in GI- and GW-processes. As described in Section 4.3.2, the rate of the true Schöning-process can be bounded by the rate of the substitute process $(d_l)_l$. It turns out that a similar argument can be made for GW (see Appendix [31]), thus yielding a rigorous bound for the rate also in this case. However, for the other processes we rather regard the $(d_l)_l$ process as a model of the genuine Schöning-walk, without rigorous guarantees of analogous bounds.

4.4 Partial Groverizations

The previous section introduced two types of partial Groverizations of Schöning's algorithm, GI and GW, based on the division $S = (S_I, S_W)$, i.e. the initial and the walk randomness. In this section, we describe these schemes in detail and further discuss their "fractional" cases.

In the GI scheme, there is an outer loop that classically samples S_W, and is followed by a Grover-search inner loop over the space of all possible S_I. Similarly, GW starts with a classical outer loop that samples S_I and is followed by a Grover-search inner loop over the space of all possible S_W (this space is well-defined as the walk length is fixed). We obtain Fractional Groverized Initialization (FGI) by adapting GI to a regime where only a fraction z of the variables in the initialization can be searched coherently, with $0 \leq z \leq 1$. Fractional Groverized Walk (FGW) is similarly an adaption of GW to a regime where Grover-search can be performed on the randomness of walks of at most m_q steps, with $0 \leq m_q$. In both these fractional schemes, two classical outer loops contain a Grover-search inner loop. The algorithms introduced here depend on parameters (N_1, N_2, etc), that will be specified explicitly in Section 4.5.

All Grover searches will use an oracle derived from the function shown in Alg. 14: It tests whether a Schöning-walk with initial configuration $x \in \{0,1\}^n$ and walk randomness $w \in$

$\{1,2,3\}^m$ will lead to a satisfying assignment. For notational convenience, we let the elements of w take ternary in values, with the interpretation that w_l determines which of the three literals occurring in the first violated clause (if any) in step l of the walk is flipped. For a qubit-based implementation, it is not difficult to re-label the decision variables using $\lceil m \log 3 \rceil$ binary variables.

Algorithm 14 Schöning Walk & Oracle

```
function Oracle(x_0, w)
    return True if SchoeningWalk(x_0, w) satisfies all clauses, else False

function SchoeningWalk(x, w)
    for j = 1...m do
        if x violates one of C_1, ..., C_L then
            k ← index of first unsatisfied clause
            l ← index of the w_j-th variable occurring in C_k
            x ← x, with the l-th bit of x flipped
    return x
```

For the different variants of partial Groverizations discussed below, we will fix a subset of arguments to the oracle, and consider it as a function of the remaining ones. Fixed arguments will be denoted as subscripts, e.g. $\textsc{Oracle}_w : x \mapsto \textsc{Oracle}(x, w)$. With these conventions, we have:

Algorithm 15 Groverized Initialization

```
for i = 1...N_2 do
    w ← uniformly random value from {1,2,3}^{×m}
    x ← Grover-search for ⌊√N_1⌋ iterations using Oracle_w()
    if x satisfies all clauses then
        return x
```

Algorithm 16 Groverized Walk

```
for i = 1...N_1 do
    x_0 ← uniformly random value from {0,1}^{×n}
    w ← Grover-search for ⌊√N_2⌋ iterations using Oracle_{x_0}()
    x ← SchoeningWalk(x_0, w)
    if x satisfies all clauses then
        return x
return False
```

One may note that the Grover search in the Groverized walk only is guaranteed to succeed (with high probability) for a specific collection of initial states. The number of rounds N_1 of the outer loop is selected in such a way that it with high probability hits the set of advantageous initial states at least once, thus allowing the Grover-procedure to reach the satisfying assignment. Similar remarks apply to the other partial Groverizations.

Next, we discuss the "fractional searches". In the first one, the argument x of the oracle is broken up as $x = (x_c, x_q)$ with x_q taking $\lfloor z \cdot n \rfloor$ bits and x_c being $\lceil (1-z) \cdot n \rceil$ bits long. Here, $z \in [0, 1]$ is a free parameter whose value will be determined below.

Algorithm 17 Fractional Groverized Initialization

for $i = 1...N_2$ **do**
  $w \leftarrow$ uniformly random value from $\{1, 2, 3\}^{\times m}$
  for $j = 1...N_1^{(c)}$ **do**
    $x_c \leftarrow$ uniformly random value from $\{0, 1\}^{\times \lceil (1-z)n \rceil}$
    $x_q \leftarrow$ Grover-search for $\left\lfloor \sqrt{N_1^{(q)}} \right\rfloor$ iterations using $\textsc{Oracle}_{(x_c, w)}()$
    $x = (x_c, x_q)$
    if x satisfies all clauses **then**
      return x
return False

The second fractional algorithm breaks up the walk randomness as $w = (w_c, w_q)$ with $w_c \in \{1, 2, 3\}^{m_c}$ and $w_q \in \{1, 2, 3\}^{m_q}$ respectively. Again, the values of m_c, m_q are chosen later.

Algorithm 18 Fractional Groverized Walk

for $i = 1...N_1$ **do**
  $x_0 \leftarrow$ uniformly random value from $\{0, 1\}^{\times n}$
  for $j = 1...N_2^{(c)}$ **do**
    $w_c \leftarrow$ uniformly random value from $\{1, 2, 3\}^{\times m_c}$
    $w_q \leftarrow$ Grover-search for $\left\lfloor \sqrt{N_2^{(q)}} \right\rfloor$ iterations using $\textsc{Oracle}_{(x_0, w_c)}()$
    $w = (w_c, w_q)$
    $x \leftarrow \textsc{SchoeningWalk}(x_0, w)$
    if x satisfies all clauses **then**
      return x
return False

In the final algorithm, a fraction of $z \in [0, 1]$ of both types of variables, the ones corresponding to the initialization and the ones corresponding to the walk, will be treated quantum mechanically.

Algorithm 19 Evenly Fractionalized Grover

for $i = 1 \ldots N^{(c)}$ **do**
  $x_c \leftarrow$ uniformly random value from $\{0,1\}^{\times \lceil (1-z)n \rceil}$
  $w_c \leftarrow$ uniformly random value from $\{1,2,3\}^{\times \lceil (1-z)m \rceil}$
  $(x_q, w_q) \leftarrow$ Grover-search for $\left\lfloor \sqrt{N^{(q)}} \right\rfloor$ iterations using $\textsc{Oracle}_{(x_c,w_c)}()$
  $w = (w_c, w_q)$
  $x_0 = (x_c, x_q)$
  $x \leftarrow \textsc{SchoeningWalk}(x_0, w)$
  if x satisfies all clauses **then**
    return x
return False

4.5 Run-time Analysis

We will now lower-bound the probability of success of the various approaches. As a preparation, in Sec. 4.5.1, we give a brief account of the analysis of the classical case, before moving on to the Groverized versions in Sec. 4.5.2.

4.5.1 The classical Schöning process

The main ideas of the classical analysis are close to their presentation in Refs. [58, 59]. We work in the Markovian model $(d_l)_l$ for the behavior of the Hamming distances, as laid out in Sec. 4.3.2. Frequently, it will be convenient to measure quantities "in units of n or m". For example, we will soon choose a number $\kappa \in [0,1]$ and assume that the initial value d_0 is equal to κn. Of course, this only makes sense if κn is an integer. In order to keep the notation clean, we will implicitly assume that such expressions have been rounded to the next integer.

Choose numbers $\kappa, \nu \in [0,1]$. A given walk $(d_l)_l$ is certainly successful (in the sense that $d_m \leq 0$) if

1. The initial value is $d_0 = \kappa n$,
2. the random walk decreases the Hamming distance in exactly νm of its m steps, and
3. the condition

$$\kappa n \leq (2\nu - 1)m \tag{4.6}$$

holds.

Indeed, the right hand side of (4.6) is the difference between the number of steps where the Hamming distance has been decreased, νm, and the number of steps where the Hamming distance has been increased, $(1-\nu)m$.

For any fixed pair of values κ, ν subject to (4.6), we will now compute the probability of this particular route to success. Denote the first event by E_1 and the second event by E_2. They occur with respective probabilities

$$P(E_1) = \frac{1}{2^n}\binom{n}{\kappa n}, \qquad P(E_2) = \binom{m}{\nu m}\left(\frac{1}{3}\right)^{\nu m}\left(\frac{2}{3}\right)^{(1-\nu)m}. \tag{4.7}$$

Since the two events are independent, the success probability of the walk is lower-bounded by

$$P(x_m = x^\star|\kappa) \geq P(d_m \leq 0|x) \geq P(E_1 \wedge E_2) = P(E_1)P(E_2) \tag{4.8}$$

$$= \frac{1}{2^n}\binom{n}{\kappa n}\binom{m}{\nu m}\left(\frac{1}{3}\right)^{\nu m}\left(\frac{2}{3}\right)^{(1-\nu)m}. \tag{4.9}$$

The various binomial coefficients can be conveniently related to entropies. To this end, recall the definition of the *binary entropy function*

$$H(p) = -p\log p - (1-p)\log(1-p) \quad \text{for} \quad p \in [0,1],$$

and the *relative entropy*

$$D(p \parallel q) = -p\log q - (1-p)\log(1-q) - H(p) \quad \text{for} \quad p, q \in [0,1].$$

Then using the well-known estimate [22, Chapter 11.1]

$$\frac{1}{n+1}2^{nH(\kappa)} \leq \binom{n}{\kappa n} \leq 2^{nH(\kappa)},$$

Equation (4.9) can, after some straight-forward calculations, be concisely rewritten as

$$P(d_m \leq 0|x) \gtrsim 2^{-(1-H(\kappa))n}2^{-D(\nu\|\frac{1}{3})m}, \tag{4.10}$$

where $\gtrsim$ denotes an inequality holds asymptotically, up to a polynomial factor. Equation (4.10) directly gives an upper bound on the rate γ defined in (4.5). Since the rate expresses the logarithm of the complexity "in units of n", it makes sense to also express the length of the walk in terms of $\mu := m/n$. Then:

$$\gamma = -\lim_{n\to\infty}\frac{1}{n}\log P(d_m \leq 0|x) \leq 1 - H(\kappa) + \mu D(\nu \parallel 1/3) =: \gamma(\mu,\kappa,\nu). \tag{4.11}$$

In particular, the infimum of $\gamma(\mu,\kappa,\nu)$ subject to the constraints (4.6) and $0 \leq \mu, 0 \leq \nu, \kappa \leq 1$ is a valid bound for γ. We will perform such optimizations explicitly for the partially Groverized versions in Sec. 4.5.2. For the classical procedure, we just state the final result:

$$\mu = 1, \qquad \kappa = \frac{1}{3}, \qquad \nu = \frac{2}{3}, \qquad \gamma_{\mathrm{C}} = \log\frac{4}{3} \simeq 0.4150. \tag{4.12}$$

Remark: One might be tempted to search a tighter bound by summing the contributions to the probability of success that arise from all consistent values for μ, κ, ν, instead of just considering the extremal value. However, the rate of a sum of exponentially processes is asymptotically determined by the rate of the dominating summand alone, i.e. for all collections of $\gamma_i > 0$, it holds that

$$\lim_{n\to\infty} -\frac{1}{n} \log \sum_i 2^{-\gamma_i n} = \sup_i \gamma_i$$

(assuming convergence). Therefore, considering only the dominating term does not affect the overall asymptotic rate.

4.5.2 Partially Groverized processes

In this section, we derive the main results of this paper: Bounds on the asymptotic rates for partially Groverized versions of Schöning's scheme.

Groverized Initialization, Algorithm 15

For the parameters N_1, N_2, we choose constant multiples of $1/P(E_1), 1/P(E_2)$ respectively. The value of the constant depends on the acceptable probability ϵ of failure, as exhibited in Eq. (4.4). Since this constant does not effect the rate, we will not specify it here. The probabilities do depend essentially on the parameters μ, κ, ν, though. We will therefore write $N_1(\kappa)$ and $N_2(\mu, \nu)$. Because the asymptotic complexity of a Grover search is the square root of the classical complexity, the rate function of GI is then given by

$$\gamma_{\mathrm{GI}}(\mu, \kappa, \nu) = \lim_{n\to\infty} \frac{1}{n} \log\Big(\sqrt{N_1(\kappa)} N_2(\nu, \mu)\Big) = \frac{1 - H(\kappa)}{2} + \mu D(\nu \parallel 1/3). \tag{4.13}$$

Likewise, the required coherence time scales with the number of Grover iterations, i.e. as $\mathcal{O}^*(2^{\chi n})$, for

$$\chi(\kappa) := \lim_{n\to\infty} \frac{1}{n} \log \sqrt{N_1(\kappa)} = \frac{1 - H(\kappa)}{2}. \tag{4.14}$$

The parameters are constrained by

$$0 \le \kappa \le 1, \quad 0 \le \mu, \quad 0 \le \nu \le 1, \quad \frac{\kappa}{2\nu - 1} \le \mu, \tag{4.15}$$

where the final condition is a re-arranged version of the success criterion (4.6).

We now determine the minimal rate γ_{GI} over the consistent parameters. Because relative entropy is non-negative, it is always advantageous to reduce the value of μ until it is minimal subject to the constraints. This is achieved by changing the final inequality in (4.15) to equality. Re-arranging, we arrive at

$$0 \le \kappa \le 1, \quad 0 \le \mu, \quad \nu = \frac{1}{2} + \frac{\kappa}{2\mu}, \tag{4.16}$$

which allows us to eliminate $\nu = \nu(\kappa, \mu)$ from the problem. Varying γ with respect to μ gives rise to the criticality condition

$$0 \overset{!}{=} \partial_\mu \gamma_{\mathrm{GI}}(\kappa, \mu) = \partial_\mu \, \mu D(1/2 + \kappa/(2\mu) \parallel 1/3) \tag{4.17}$$

$$= \frac{1}{2} \log\left(\frac{\mu + \kappa}{\mu}\right) + \frac{1}{2} \log\left(\frac{\mu - \kappa}{\mu}\right) + \log 3 - \frac{3}{2}. \tag{4.18}$$

This can be solved explicitly e.g. using a computer algebra system [30], leading to

$$\mu = 3\kappa \qquad \Rightarrow \qquad \nu = \frac{2}{3}, \quad \mu D(\nu \parallel 1/3) = \kappa. \tag{4.19}$$

Eliminating μ, we get

$$\begin{aligned} \gamma_{\mathrm{GI}}(\kappa) &= \frac{1 - H(\kappa)}{2} + \kappa, \\ \chi_{\mathrm{GI}}(\kappa) &= \frac{1 - H(\kappa)}{2}. \end{aligned} \tag{4.20}$$

The pair of equations (4.20) contain all information about the asymptotic behavior of the Groverized Initialization procedure. Each value of κ gives a solution for the two undetermined constants $N_1(\kappa), N_2(\mu = 3\kappa, \nu = \frac{2}{3})$ in Alg. 15, in such a way that it will run with a small probability of returning a false negative. Varying κ, we thus obtain a family of algorithms that find different compromises between the required coherence time and the total runtime. The achievable pairs of values are shown in Fig. 4.2.

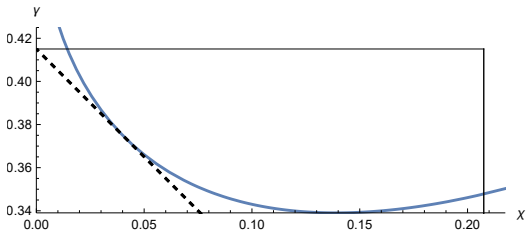


Figure 4.2: Rates $\big(\chi_{\mathrm{GI}}(\kappa), \gamma_{\mathrm{GI}}(\kappa)\big)$ for the required coherence time and the total runtime of the Groverized Initialization algorithm, as the parameter κ is varied. The horizontal bar denotes the runtime rate achieved by the classical Schöning process. In other words, points above this line are uninteresting. The vertical bar denotes the coherence rate that allows one to run a completely Groverized version of the Schöning process. This, arguably, makes points to the right of this line uninteresting as well. Points to the left of the minimum (at $(\gamma, \chi) \simeq (0.339, 0.139)$) can represent advantageous choices if either the total coherence time of a quantum computer is limited, or a larger degree of parallelization is desired. The dashed line is the lower bound on the runtime rate given the coherence time, as introduced in Fig. 4.1. It is achieved for $\kappa = \frac{1}{3}$.

Finally, we explicitly determine the minimal rate achievable in the Groverized Initialization scheme. With the help of a computer algebra system [30], one easily finds

$$0 \stackrel{!}{=} \partial_\kappa \gamma_{\mathrm{GI}}(\kappa) = \frac{1}{2}\log\frac{\kappa}{1-\kappa} + 1 \quad \Leftrightarrow \quad \log\left(\frac{1}{\kappa} - 1\right) = 2 \quad \Rightarrow \quad \kappa = \frac{1}{5} \tag{4.21}$$

which gives

$$\mu = \frac{3}{5}, \qquad \gamma_{\mathrm{GI}} = \frac{3 - \log 5}{2} \approx 0.339, \qquad \chi_{\mathrm{GI}} \simeq 0.139. \tag{4.22}$$

Remark: One can cast the final minimization into the form

$$\gamma_{\mathrm{GI}} = \inf_\kappa \gamma_{\mathrm{GI}}(\kappa) = \inf_\kappa \left(\frac{1 - H(\kappa)}{2} + \kappa\right) = -\sup_\kappa \left(-\kappa - \frac{H(\kappa) - 1}{2}\right).$$

This expression shows that the optimization amounts to computing a Legendre transform. Indeed, with $f(\kappa) := 1/2(H(\kappa) - 1)$, the right hand side equals $-f^*(-1)$. For physicist readers, it might be amusing to note that $S(n\kappa) = nH(n\kappa)$ formally equals the entropy of an n-spin paramagnet as a function of the total magnetization. The Legendre transform of the entropy is a Massieu thermodynamic potential, equal to F/T (with F the free energy) expressed as a function of the inverse temperature [16, Chapter 5.4]. We will, however, not pursue this analogy here.

Groverized Walk, Algorithm 16

The analysis proceeds in close analogy to the above case. The asymptotic rate function of GW is

$$\gamma_{\mathrm{GW}}(\kappa, \mu, \nu) = \lim_{n\to\infty} \frac{1}{n} \log(N_1(\kappa)\sqrt{N_2(\nu, \mu)}) = 1 - H(\kappa) + \frac{\mu}{2} D(\nu \parallel 1/3), \tag{4.23}$$

subject to the set of constraints (4.15). The parameters ν, μ can be treated in exactly the same way as before, leading again to (4.19). In particular, the coherence time rate takes the simple form $\chi = \kappa/2$, which allows us to eliminate κ in favor of χ. We immediately obtain

$$\gamma_{\mathrm{GW}}(\chi) = 1 - H(2\chi) + \chi. \tag{4.24}$$

Again, it is not difficult to solve for the lowest runtime [30]:

$$\mu = 3(\sqrt{2} - 1), \qquad \kappa = \sqrt{2} - 1, \qquad \gamma_{\mathrm{GW}} \approx 0.228, \qquad \chi_{\mathrm{GW}} \simeq 0.2071. \tag{4.25}$$

At the optimal point, the runtime scales with a rate that is very close to the one of a full Groverization of Schöning's process, namely $\gamma_{\mathrm{FG}} = \gamma_{\mathrm{C}}/2 \simeq .2075$. The flip side is that the required coherence times are basically identical:

$$\chi_{\mathrm{FG}} - \chi_{\mathrm{GW}} \simeq 0.0004.$$

The findings are summarized in Fig. 4.3.

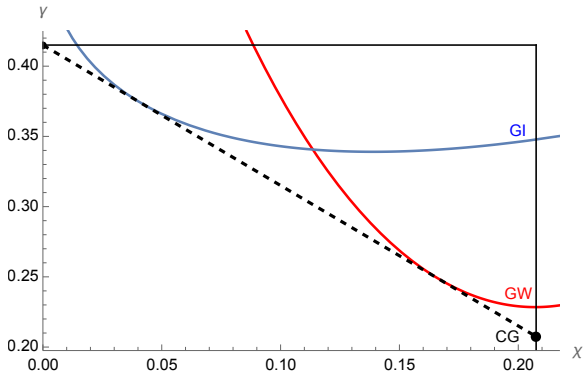


Figure 4.3: The runtime rate vs coherence time rate curves for Groverized Initialization (GI, blue) and Groverized Walk (GW, red). The point marked "CG" at the bottom right of the diagram represents the complete Groverization of the Schöning process. For long coherence times, GW is preferable, while for shorter coherence times GI achieves a lower total runtime.

Fractional Groverized Initialization, Algorithm 17

In the case of Alg. 17, the initial Hamming distance is the sum of two terms $d_0 = \kappa_c(1-z)n + \kappa_q z n$, which model $d_H(x_c, x_c^\star)$ and $d_H(x_q, x_q^\star)$ respectively. Define the analogues

$$P(E_1^c) = \frac{1}{2^{(1-z)n}}\binom{(1-z)n}{\kappa_c(1-z)n}, \qquad P(E_1^q) = \frac{1}{2^{zn}}\binom{zn}{\kappa_q z n}$$

of $P(E_1)$ introduced in Eq. (4.7). Analogous to the discussion in Sec. 4.5.2, the parameters N_1^c, N_1^q are defined as the reciprocals of these probabilities, times a constant that influences the probability of a false negative, but will not be discussed as it has no impact on the asymptotic rates. The success criterion is now

$$(1-z)\kappa_c + z\kappa_q \leq (2\nu - 1)\mu$$

and the other constraints are

$$0 \leq \kappa_c, \kappa_q, \nu, z \leq 1, \qquad 0 \leq \mu.$$

The asymptotic rate function for the runtime of FGI reads

$$\begin{aligned}\gamma_{\mathrm{FGI}}(\kappa_c, \kappa_q, \nu, \mu; z) &= \lim_{n\to\infty} \frac{1}{n} \log\left(N_1^c(\kappa_c; z)\sqrt{N_1^q(\kappa_q; z)}N_2(\nu, \mu)\right) \\ &= (1-z)(1-H(\kappa_c)) + \frac{z}{2}\big(1-H(\kappa_q)\big) + \mu D(\nu \parallel 1/3) \qquad (4.26)\end{aligned}$$

Arguing as in Sec. 4.5.2, the inequality in the success criterion may be replaced by an equality. Solving for ν gives

$$\nu = \frac{1}{2} + \frac{(1-z)\kappa_c + z\kappa_q}{2\mu}.$$

We proceed as in the first two cases. Criticality of $\partial_\mu \gamma_{\mathrm{FGI}}$ with respect to μ occurs at

$$\mu = 3((1-z)\kappa_c + z\kappa_q) \qquad \Rightarrow \qquad \mu D(\nu \parallel 1/3) = (1-z)\kappa_c + z\kappa_q, \quad \nu = \frac{2}{3}.$$

Plugging in, we arrive at

$$\gamma_{\mathrm{FGI}}(\kappa_c, \kappa_q; z) = (1-z)(1 - H(\kappa_c) + \kappa_c) + z\left(\frac{1 - H(\kappa_q)}{2} + \kappa_q\right). \tag{4.27}$$

In other words, the runtime rate function is a convex combination of the ones for the classical Schöning process and for the GI scheme, with weights $(1-z), z$ respectively. Because the classical part does not affect the coherence time, we may set κ_c to its optimal value $\kappa_c^* = 1/3$ (c.f. Eq. (4.12)). Geometrically, as we vary $z \in [0,1]$, Eq. (4.27) describes a line connection $(\chi_{\mathrm{GI}}(\kappa_q), \gamma_{\mathrm{GI}}(\kappa_q))$ with the parameters of the classical Schöning process $(0, \gamma_{\mathrm{C}})$. By the convexity of the GI curve, the fractional algorithm will have a better runtime rate to the left of the value of κ_q at which the line becomes tangent to the curve. In other words, the critical κ_q is defined by the condition

$$\frac{\partial \gamma_{\mathrm{GI}}}{\partial \chi} = \frac{\gamma_{\mathrm{GI}} - \gamma_{\mathrm{C}}}{\chi}.$$

By a computer calculation [30], this happens for $\kappa_q = \frac{1}{3}$ (i.e. equal to κ_c), resulting in the following curve:

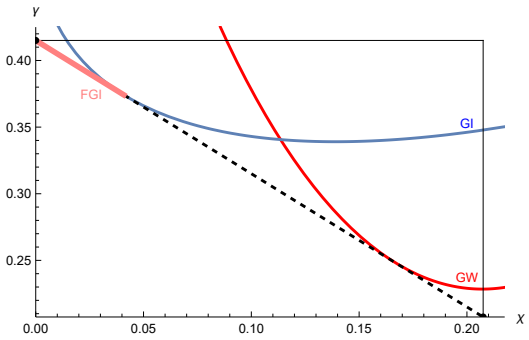


Figure 4.4: The runtime rate vs coherence time rate for the FGI algorithm. This fractional scheme's performance is the convex combination of the classical point $(0, \gamma_{\mathrm{C}})$, and GI at the tangent point to the theoretical lower bound. One can note that the FGI partially saturates the optimal performance relation.

Fractional Groverized Walk, Algorithm 18

In the FGW scheme, we assume that the classical and Groverized walks decrease the Hamming distance in exactly $\nu_c m_c$ and $\nu_q m_q$ steps, respectively, where we have used a subscript to differentiate between the classical and Groverized random walks. The probabilities of such walks occurring is given by:

$$P(E_2^c) = \binom{m_c}{\nu_c m_c}\left(\frac{1}{3}\right)^{\nu_c m_c}\left(\frac{2}{3}\right)^{(1-\nu_c)m_c}, \quad P(E_2^q) = \binom{m_q}{\nu_q m_q}\left(\frac{1}{3}\right)^{\nu_q m_q}\left(\frac{2}{3}\right)^{(1-\nu_q)m_q} \tag{4.28}$$

Analogous to the discussion in Sec. 4.5.2, the parameters N_2^c, N_2^q are defined as the reciprocals of the probabilities $P(E_2^c)$, $P(E_2^q)$, times a constant that influences the probability failure, but will not be discussed as it has no impact on the asymptotic rates. We further parameterize the walk lengths as $m_c = \mu_c n$ and $m_q = \mu_q n$. The runtime rate is

$$\begin{aligned}\gamma_{\mathrm{FGW}}(\kappa, \nu_c, \mu_c, \nu_q, \mu_q) &= \lim_{n\to\infty} \frac{1}{n}\log\left(N_1(\kappa) N_2^c(\nu_c, \mu_c)\sqrt{N_2^q(\nu_q, \mu_q)}\right) \\ &= 1 - H(\kappa) + \mu_c D(\nu_c \parallel 1/3) + \frac{\mu_q}{2} D(\nu_q \parallel 1/3)\end{aligned} \tag{4.29}$$

with parameters subject to the constraints

$$\begin{aligned}&0 \le \kappa \le 1, \\ &0 \le \mu_c, \mu_q, \\ &0 \le \nu_c, \nu_q \le 1, \\ &\kappa \le (2\nu_c - 1)\mu_c + (2\nu_q - 1)\mu_q.\end{aligned} \tag{4.30}$$

The first steps of the analysis should now be familiar. There is no loss of generality in assuming that the final inequality is tight, which can be re-arranged to give

$$\nu_q = \frac{\kappa - (2\nu_c - 1)\mu_c}{2\mu_q} + \frac{1}{2}.$$

The rate γ_{FGW} is stationary as a function of μ_q if

$$\mu_q = 3(\kappa - (2\nu_c - 1)\mu_c) \Rightarrow \nu_q = \frac{2}{3}, \quad \frac{\mu_q}{2} D(\nu_q \parallel 1/3) = 1/2(\kappa - (2\nu_c - 1)\mu_c) = \chi(\kappa, \nu_c, \mu_c).$$

Eliminating κ in favor of the coherence rate χ gives

$$\kappa = 2\chi + (2\nu_c - 1)\mu_c$$

and thus

$$\mu_q = 6\chi, \quad \gamma_{\mathrm{FGW}}(\nu_c, \mu_c; \chi) = 1 - H(2\chi + (2\nu_c - 1)\mu_c) + \mu_c D(\nu_c \parallel 1/3) + \chi.$$

We now need to minimize γ_{FGW} for fixed χ as a function of μ_c, ν_c, subject to

$$\begin{aligned} &0 \leq 2\chi + (2\nu_c - 1)\mu_c \leq 1, \\ &0 \leq \mu_c, \\ &0 \leq \nu_c \leq 1. \end{aligned}$$

We may assume that $\mu_c \neq 0$, for else we are just replicating the GW scheme. A computer calculation [30] gives

$$\partial_{\mu_c}(\gamma \ln 2) + \frac{2 - 4\nu_c}{4\mu_c}\partial_{\nu_c}(\gamma \ln 2) = -\arctan(1 - 2\nu_c) + \ln(3 - 3\nu_c) - \frac{1}{2}\ln 2,$$

which has zeros at $\nu_c = \frac{1}{3}$ and $\nu_c = \frac{2}{3}$.

For $\nu_c = \frac{1}{3}$, one finds

$$\partial_{\mu_c}(\gamma \ln 2) = \frac{2}{3}\arctan(1 - 4\chi + 2/3\mu_c)$$

which has one zero, at $\mu_c = \frac{3}{2}(4\chi - 1)$. The constraint $\mu_c \geq 0$ then implies $\chi \geq \frac{1}{4}$. But this is larger than the coherence time rate $\gamma_{\mathrm{C}}/2 \simeq 0.208$ sufficient to implement a completely Groverized version of Schöning's process, so this solution is not of interest.

We turn to the other solution, $\nu_c = \frac{2}{3}$. For it,

$$\partial_{\mu_c}(\gamma \ln 2) = 1/3(-2\,\mathrm{arctanh}(1 - 4\chi - (2\mu_c)/3) + \ln 2),$$

which has one zero:

$$\mu_c = 1 - 6\chi \qquad \Rightarrow \qquad \mu_q = 6\chi, \quad \nu_c = \nu_q = \frac{2}{3}, \quad \gamma_{\mathrm{FGW}} = \gamma_{\mathrm{C}} - \chi.$$

The runtime vs coherence rate curve for the FGW scheme is given in the following figure:

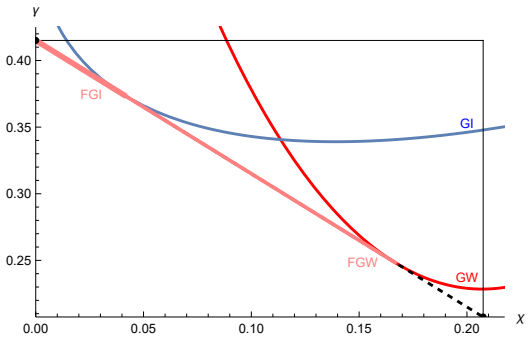


Figure 4.5: The runtime rate vs coherence time rate for the FGW algorithm. This fractional scheme's performance connects the GW curve to the classical Schöning point and is tangent to the curve. It achieves the optimal performance relation partially for a larger regime than FGI and for low coherence times, it comes to lie on top of the FGI line.

Evenly Fractionalized Grover

The runtime rate is

$$\gamma_{\mathrm{EFG}} = (1-z)\Big(1 - H(\kappa_c) + \mu_c D(\nu_c \parallel 1/3)\Big) + z/2\Big(1 - H(\kappa_q) + \mu_q D(\nu_q \parallel 1/3)\Big) \tag{4.31}$$

with success criterion

$$(1-z)\kappa_c + z\kappa_q = (1-z)(2\nu_c - 1)\mu_c + z(2\nu_q - 1)\mu_q,$$

which is in particular true if the following two equations hold

$$\kappa_c = (2\nu_c - 1)\mu_c, \qquad \kappa_q = (2\nu_q - 1)\mu_q.$$

But this is just the convex interpolation between a completely classical and a completely Groverized process. In particular, by choosing the parameters as for the original Schöning process

$$\nu_c = \nu_q = \frac{2}{3}, \quad \kappa_c = \kappa_q = \frac{1}{3}, \quad \mu_c = \mu_q = 1,$$

we obtain a coherence time–runtime rate curve that linearly connects the classical point $(0, \gamma_{\mathrm{C}})$ to the completely Groverized one $(\gamma_{\mathrm{C}}/2, \gamma_{\mathrm{C}}/2)$ (Fig. 4.6).

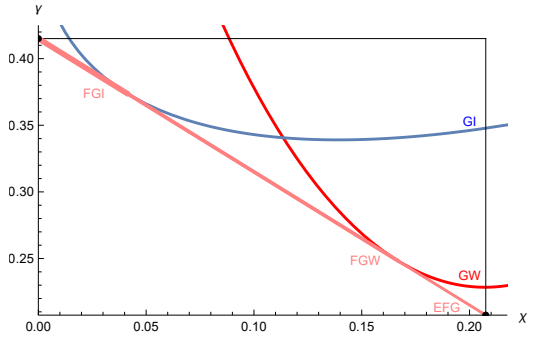


Figure 4.6: Runtime–coherence time rate curves for the covered algorithms. The linear interpolation between the classical and the completely Groverized points are realizable using an increasing number of methods – first only EFG, then also FGW, finally also FGI – as the coherence time decreases.

4.5.3 A heuristic de-randomization of the GI schemes

In this section, we provide evidence that the Groverized initialization schemes can reach further into the γ–χ chart than what the Markovian model suggests. To see why this is plausible, note that the role of randomness for the initial configuration x is very different from the role of randomness for the walk decisions w. In the first case, there is an "absolute measures of the quality of the initial

configuration", namely the Hamming distance to the solution. The probability that the walk does find the solution is quite obviously a function of that metric. Therefore, baring major algorithmic insights, it is unavoidable to consider many different initial configurations before encountering one that will likely lead to a solution.

In contrast, it is not implausible that "every walk works for equally many initial configurations", i.e. that there are no choices for w that are "intrinsically better than others". More precisely, it seems reasonable to assume that for sufficiently large n, and generic SAT formulas, it holds that with high probability in w

$$\begin{aligned} &-\frac{1}{n}\log\Big(\Pr_{x}[\textsc{SchoeningWalk}(x,w)=x^\star \mid d_H(x,x^\star)=h,w]\Big) \\ &\simeq -\frac{1}{n}\log\Big(\Pr_{x,w'}[\textsc{SchoeningWalk}(x,w')=x^\star \mid d_H(x,x^\star)=h]\Big). \end{aligned} \tag{4.32}$$

The right hand side can be easily calculated, as by Ref. [58], for $\mu = 3$,

$$\Pr_{x,w}[\textsc{SchoeningWalk}(x,w)=x^\star \mid d_H(x,x^\star)=h] = 2^{-h}.$$

Under Assumption (4.32), one can restrict the outer loop over w's from Alg. 15 to $N_2 = 1$ iteration, and compensate by increasing the number of Grover iterations for x to $N_1 = \mathcal{O}^*(2^{\gamma_C/2n})$. In other words, the Groverized Initialization scheme with these parameters would lie on the optimal point $(\chi,\gamma) = (\gamma_C/2, \gamma_C/2)$.

Being even bolder, one could then speculate that the analysis of Sec. 4.5.2 carries over and that, as one varies the fraction of initialization bits that are subjected to a Grover search, one could trace out the optimal (χ,γ)-line. In other words, it does not seem impossible that the following Alg. 20, with parameter choice

$$N_1^{(c)} = \mathcal{O}^*(2^{\gamma_C(1-z)n}), \qquad N_1^{(q)} = \mathcal{O}^*(2^{\gamma_C zn/2}),$$

achieves the optimal trade-off.

Algorithm 20 Heuristically De-Randomized Fractional Groverized Initialization

```
w ← uniformly random value from {1,2,3}^{×m}
for j = 1...N_1^{(c)} do
    x_c ← uniformly random value from {0,1}^{×⌈(1−z)n⌉}
    x_q ← Grover-search for ⌊√(N_1^{(q)})⌋ iterations using Oracle_{(x_c,w)}()
    x = (x_c, x_q)
    if x satisfies all clauses then
        return x
return False
```

To gather evidence in favor of Assumption (4.32), we have resorted to numerical methods. A first ansatz is to compute the l.h.s. of Eq. (4.32) exactly, which is possible for small values of n by

iterating over all 2^n assignments to x. Results are shown in Fig. 4.7 for a randomly chosen set of 3-SAT formulas with $n = 20$ variables, $L = 91$ clauses. The number of satisfying assignments t_0 of the formulas are varied. Only the case $t_0 = 1$ can be directly compared to the analytic bounds. However, note that even for this case, the empirically observed rate of $\gamma_{\mathrm{GI}} \simeq .12 \pm .02$ is much lower than the value $\gamma_{\mathrm{C}}/2 \simeq .208$ that we would expect theoretically. Presumably, $n = 20$ is still too small to show the asymptotic behavior.

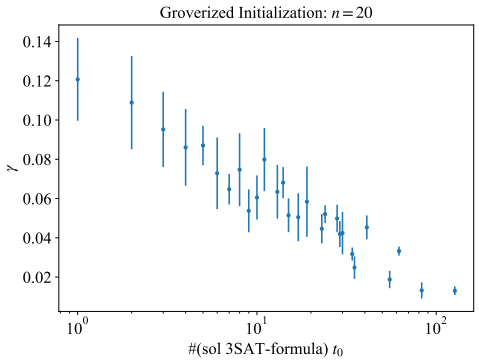


Figure 4.7: Plot of the runtime rate for the heuristically de-randomized GI scheme. Error bars indicate variation as a function of the formulas and the walk variables w. On the x-axis, we show the number of satisfying assignments in the formula. Only the case of $t_0 = 1$ should be directly comparable to the analytic bounds. The empirically observed behavior is much better than the analytic results, suggesting that $n = 20$ is too small to capture the asymptotic behavior.

To test this assumption, we had to turn to numerical heuristics, to at least probe the behavior for much larger values of n, where an exact computation is no longer possible. The results are shown in Fig. 4.8. We used a SAT instance with $n = 1414$ variables that we believe to have a single satisfying assignment $x^\star$ which is explicitly known. To generate the instance, a 128-bit plain text was encoded by a 128-bit key using the XTEA block cipher truncated to three rounds. The formula represents the conditions on an input key to map the known plain text to the known ciphertext. The clauses are designed such that they enforce the correct evaluation of bit-wise operations of the algorithm with respect to the given input and output. XTEA was restricted to three rounds in order to keep the size of the formula manageable. While we have no formal proof, it is reasonable to assume that there is a unique key that satisfies the formula. This is supported by consistency checks in terms of running SAT solvers on a version of this problems with even fewer rounds [40].

Let us denote the sphere of strings with Hamming distance h from $x^\star$ by $M^h(x^\star)$. For a fixed walk randomness w, and for $h = 1, \ldots 11$, we have drawn x uniformly from $M^h(x^\star)$. In order to compare the numerical results to the theory prediction, we have to use the value of the right hand side of Assumption (4.32) for non-asymptotic values of n.

The following plot shows the empirically estimated probabilities of Schöning's walk (with $\mu = 3$) arriving at the solution, when starting from a random initial configuration of given Hamming distance. The findings show the expected behavior of averaging over w, already for a fixed random value of w. In this sense, they are compatible with Assumption (4.32). We note, however, that we were not able to probe the assumption for larger values of h. Garnering a better understanding

for the concentration properties of the Schöning walk as a function of the walk choices remains therefore an open question.

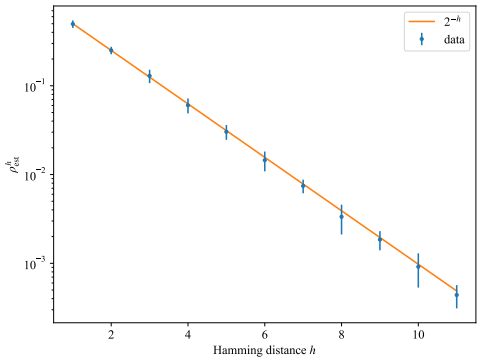


Figure 4.8: Estimated probability for a uniformly random initial configuration x with Hamming distance h to be mapped to $x^\star$ under a Schöning walk, for a fixed, randomly chosen set of walk decisions w (c.f. Alg. 20). The SAT instance has $n = 1414$ variables and is believed to have a unique satisfying assignment [40]. For each data point, 10^4 initial configurations x, were sampled uniformly from the Hamming distance sphere $M^h(x^\star)$. The results agree well with the theoretical prediction under Assumption (4.32) (orange line).

4.6 Circuits

In this section, we discuss an implementation of the partial Groverization schemes and present the main building blocks of their quantum circuits. Given n variables and the length of Schöning's walk m, the quantum implementation requires $n + m \log 3$ qubits to encode the initializations and walk randomness. The oracles of the partial Groverization schemes are some adaptation of one or more Schöning walks, and regardless of the search space they act on, the label of the violated clause at each step needs to be stored in their workspaces. This is necessary since such oracles are typically realized using uncomputation, therefore, $\log L$ extra auxiliary qubits are needed at each step, amounting to $m \log L$ qubits in total for the workspace. As a result, encoding any Groverization of Schöning's algorithm asymptotically needs $n + (\log 3 + \log L)m$ qubits.

Figure 4.9 represents a single step of Schöning walk, schematically. The first register encodes the space of all possible initialization. The gates ev_j, for $j \in \{1, .., L\}$, act on the first two registers. Each gate consists of a few controlled-gates where the control qubits correspond to the three variables in the j-th clause, and the target qubit is the second register. The second register is an auxiliary qubit, initially set to $|0\rangle$, and is negated as soon as the first violated clause is detected. The third register consists of $\log L$ auxiliary qubits that are used to count the number of clauses from where the first violated clause has happened. The last register is a qutrit providing the randomness of the corresponding walk step. The controlled-gates ch_j, for $j \in \{1, .., L\}$ act on the first three registers, and take care of variable flipping wherever the first violated clause is detected. The $0 \vee 1 \vee 2$ block represents a triple controlled-gate where the control qutrit is the subspaces corresponding to the computational basis states $|0\rangle, |1\rangle, |2\rangle$. Figure 4.10 depicts the controlled-gates

including ch_j, in detail. The sub-figure on the right shows the corresponding controlled-gate for GI, where the walk randomness is fed classically to the last register.

All partial Groverization of Schöning algorithm can be implemented using slight modifications. For the GW algorithm, the n-qubit variable register will not be initialized in the uniform superposition of all possible assignments $|+\rangle^{\otimes n}$, but rather in a state with classically randomly defined variables $|x_1 \cdots x_n\rangle$. For the GI algorithm the qutrit within every Schöning's step can be removed since we can, for every Schöning's step, generate a random number $r \in \{0, 1, 2\}$ and apply only the X gates based on the classically determined r (see figure 4.10).

4.7 Summary & Outlook

This work considers hybrid schemes for search-based quantum algorithms, with the aim to allow for parallelizability, and to reduce the need for long coherence times. The basic gist is to partition the randomness of an underlying classical probabilistic algorithm into a part that is subject to Grover search, while the rest is sampled classically. Such "partial Groverizations" allow for parallelization of the classical sampling, as well as enable adaption to available coherence times. We consider exponential-time algorithms, why our analysis focuses on the asymptotic run-time rates and coherence-time rates. We argue that these two types of rates are bounded by a general trade-off relation that no hybrid-scheme can beat. For our concrete analysis, we consider hybrid schemes based on Schöning's algorithm, where the latter solves 3-SAT (or more generally k-SAT) problems by random walks in the space of assignments. The walk-procedure allows for several partial Groverization-schemes. We determine the corresponding run-times and coherence-times of these schemes, and demonstrate saturation of the general trade-off relation. Many of these partial Groverizations intuitively lend themselves for efficient circuit implementations, and we provide the main building blocks of these. On a more speculative note, we present numerical evidence that the GI scheme can be partially de-randomized, in the sense that a single "typical" instance of the classical randomness of the walk appears to mimic the effects of the repeated sampling. This would open for an additional flexibility in the implementation of these hybrid-schemes, still maintaining the optional trade-off.

In this investigation, we have focused on partial Groverizations of Schöning's algorithm. However, this approach should in principle be applicable to any classical probabilistic search scheme, since it essentially only rests on partitions of the underlying randomness. The main concern would be to find "natural" partitions that are algorithmically accessible, in the sense that the partial Groverization can be implemented efficiently. Explicit run-time and coherence-time rates would also require a classical scheme, as well as partitions, that are sufficiently tractable for analysis, unless one would resort to numerical estimates.

The partial de-randomization of GI-scheme that is suggested by our numerical explorations, would deserve further investigations. In particular, the question is to what extent, and in what sense, the hypothetical relation (4.32) would be true. Moreover, one may ask if something similar also would apply to fractional GI. For numerical investigations, it would be relevant to extend to larger Hamming distances, further classes of 3-SAT instances, as well as problem sizes. This would likely involve challenges to design reliable numerical estimates, since exact calculations by

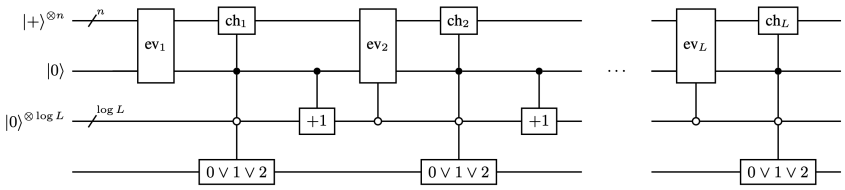


Figure 4.9: The quantum implementation of a single Schöning's step for a general implementation of the partial Groverization of Schöning's algorithm. The ev_j gates evaluate the j-th clause on the corresponding variables and the controlled-gates containing ch_i and $0 \vee 1 \vee 2$ act on all the registers and check if the j-th clause is the first violated clause and if so, flip one of three variables in it based on the randomness provided by the if-statement, $0 \vee 1 \vee 2$. Here $0 \vee 1 \vee 2$ represents a triple controlled-gate where the control qutrit is the subspaces of the computational basis (visualized in figure 4.10). The $\log L$ auxiliary qubits are needed for uncomputation.

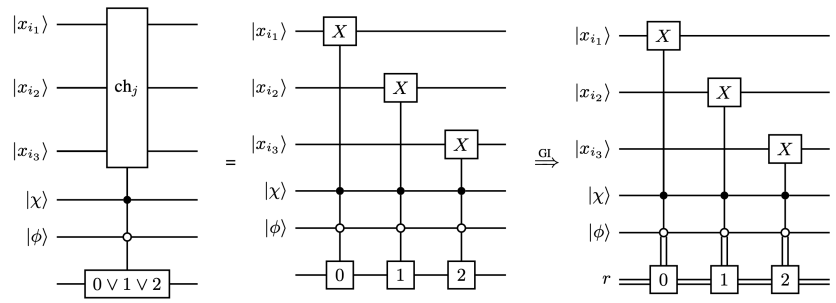


Figure 4.10: Implementation of the variable flips of Schöning's walk within amplitude amplification. Here, x_{i_1}, x_{i_2} and x_{i_3} are the variables of the j-th clause. The $0 \vee 1 \vee 2$ block represents a triple controlled-gate where the control qutrit is the subspaces of the basis $|0\rangle, |1\rangle, |2\rangle$. For the GI algorithm, the walk randomness can be provided by fixing a random number $r \in \{0, 1, 2\}$ for every walk step.

the very nature of the problem quickly becomes intractable. For purely analytical approaches, some notion of concentration of measure of walks, would be interesting.

In the spirit of [58, 59] we have in this investigation employed "the walk on $\mathbb{Z}$" as a model of the true Schöning-procedure. In Appendix of [31] (see also [49]) we additionally provide bounds for the true rates of Schöning-procedure and the GW-procedure, in terms of the mirroring processes on $\mathbb{Z}$. It would be relevant to obtain similar bounds also for the GI-process, as well as for the various fractional schemes.

4.8 Acknowledgement

We thank Phillip Keldenich for kindly providing us with the SAT instance that was used for numerical simulations resulted in figure (4.8). We acknowledge support from Bundesministerium für Bildung und Forschung - BMBF under project QuBRA. The UzK team was also supported by Germany's Excellence Strategy – Cluster of Excellence Matter and Light for Quantum Computing (ML4Q) EXC 2004/1 (390534769).

5 Conclusions

Applying Bayesian inference to Grover's Search when the number of solutions is unknown presents two main challenges:

1. representing the prior knowledge over the solution space, which may require storing an array of size $\mathcal{O}(N)$, and

2. assuming the availability of a meaningful prior distribution in the first place.

To address these challenges, we employ Particle Filtering, a Bayesian filtering method. While typically used in dynamic systems with evolving latent states, Particle Filtering can also be beneficial in static settings such as search problems. Specifically, it allows one to approximate the prior distribution with a manageable number of weighted particles, thereby circumventing the need for large memory and enabling adaptation based on observed evidence.

We implement this approach in the Particle-Guided Grover's Search (PGGS) algorithm and benchmark it against QSearch through numerical simulations. Various experimental settings were explored. Results suggest that PGGS achieves a consistent speedup – roughly halving the average query count – whenever the particle distribution aligns well with the actual distribution from which the number of solutions is drawn, regardless of the type of distribution.

PGGS addresses the first challenge by approximating the prior with a finite set of weighted particles that evolve over time based on observations. The second challenge is partially addressed: PGGS performs well unless the true distribution over the number of solutions is sharply peaked in region $t/N \simeq 0$. In such cases, a mismatch between the prior and reality leads to a worser performance.

In studying the trade-off between coherence and overall runtime for hybrid SAT solvers, we focused on Schöning's random walk over assignments of a 3-SAT formula. The walk begins at a uniformly random assignment and then, at each step, checks whether the given formula is satisfied. If not, it picks the first unsatisfied clause – according to a predefined order – and randomly flips one of its literals. Thus, there are two sources of randomness: (1) choosing the initial assignment (which may be repeated if the walk exceeds a predefined step threshold), and (2) picking which literal to flip at each failed step.

We recast this randomized procedure as a deterministic function that maps a single "randomness string" (encoding both the initial assignment and the sequence of literal-flips) to either success (finding a satisfying assignment) or failure. In other words, each bitstring fully specifies one run of the walk, and its output indicates whether that run finds a solution. This viewpoint reduces "finding a satisfying assignment" to the search problem "find a bitstring that the deterministic function maps to success."

Once we treat the walk's two randomness-choices as two contiguous sub-bitstrings, we can ask: which sub-bitstring is most expensive to search (assuming a limited quantum resources)? If

coherence time allows us to search only a fraction of the whole bitstring, it makes sense to allocate that quantum resource to the sub-bitstring requiring the largest amount of work. Moreover, one can ask whether correlating the two sub-bitstrings – rather than choosing them independently – yields any advantage when only a fraction of the bits can be searched coherently.

To answer these questions, we designed a family of hybrid SAT-solvers based on Schöning's walk, each parameterized by how it allocates the quantum coherence budget between the two sub-bitstrings. We analyzed the resulting trade-offs: given a coherence-time limit of zn qubits (with n variables in the 3-SAT instance and $0 \leq z \leq 1$), how should we allocate those zn qubits across the "initialization" bits and the "flip-choice" bits? Our results show that, when a full quantum search over the bitstring is infeasible, the optimal strategy is to allocate qubits fractionally equally between the two sub-bitstrings – an algorithm we name *Evenly Fractionalized Grover*. In other regimes, two variants – *Fractional Groverized Walk* (allocating the qubits to a fraction of flip-choices) and *Fractional Groverized Initialization* (allocating the qubits to a fraction of the initial assignment) – can perform just as well as the "evenly fractionalized" version.

Admittedly, these polynomial-speedup hybrid strategies do not dramatically improve practical 3-SAT solving. This is because 3-SAT is NP-complete and heuristics based on *Davis-Putnam-Logemann-Loveland* algorithm (DPPL), a backtracking algorithm, or its other variant *Conflict-Driven Clause Learning* (CDCL), and stochastic local search, already outperform such hybrid algorithms for practical cases. Nonetheless, this line of inquiry is academically valuable: it offers a systematic framework for designing and analyzing hybrid quantum-classical algorithms, a field still in its infancy. By identifying which components of a probabilistic algorithm's encoding benefit most from limited quantum search, one can gain valuable insight into how to optimally partition and allocate scarce coherence resources in hybrid algorithms derived from that probabilistic framework.

6 Outlook

Chapter 3 presented the application of Bayesian filtering to Grover's Search when the number of solutions is unknown. Benchmark results demonstrated an improvement over *QSearch* [15].

The relative entropy, both as a measure of information gain in Bayesian updating and as an indicator of potential information loss when non-informative priors are employed, could serve as key factors in understanding PGGS' performance. This raises an important open question that warrants further investigation in future work.

Further, identifying the minimal number of particles necessary to capture the essential characteristics of the prior remains an unanswered question which is crucial to answer, especially since the runtime of PGGS scales polynomially with the number of particles, L.

Here, PGGS has been evaluated in settings where the algorithm runs until a marked element is found. An alternative evaluation strategy could introduce a fixed runtime cutoff, allowing for a more nuanced comparison with QSearch in time-constrained scenarios. Additionally, a formal runtime analysis of PGGS would be beneficial to estimate the classical resources required by this hybrid algorithm.

Also, the performance of PGGS in the regime $t/N \simeq 0$ appears unstable under various parameter settings. Since – as expected by Grover's Search – both QSearch and PGGS exhibit their highest average query counts at this point, this regime needs further investigation.

We have tested PGGS under Gaussian and uniform distributions; however, particle filters are particularly valuable for handling non-Gaussian and more complex distributions. Investigating the behavior of PGGS under such priors could present a direction for future work.

Lastly, the current upper bound on the number of Grover iterations—set to $\sqrt{N}$, following the suggestion by Boyer *et al.* [13]– requires further investigation in the Bayesian context, where adaptive strategies might yield more efficient performance.

Chapter 4 examined the trade-off between runtime and coherence time in a family of hybrid SAT solvers, developed by analytically bounding the success probability of the underlying random walk – namely, Schöning's algorithm. There, we introduced a key technique: by encoding the two sources of randomness used in the random walk into a single long bitstring, we reformulated the randomized algorithm as a deterministic function that maps each bitstring to either 0 or 1, depending on whether the walk successfully finds a satisfying assignment. This reformulation allowed us to identify which parts of the bitstring are computationally more expensive to search, enabling us to allocate the more powerful search tool – namely, quantum search – to those components. This approach provides a generalizable framework that can be applied to other probabilistic algorithms, particularly in the context of hybrid algorithm design, to maximize performance under constrained quantum resources.

Further, the bounds provided in 4 are asymptotic and expressed in terms of success rates. While numerical simulations were conducted to evaluate how well these theoretical bounds hold in practice, further numerical investigations are necessary to study these rate estimates.

Acknowledgements

Throughout my school years, I was always drawn to speed runs in sports classes, while I found the strength ones far less appealing. I often questioned the purpose of pushing myself through long distances, where the goal seemed ill-defined and the effort, exhausting.

As I grew older, I never fully understood the value of these challenges–until very recently.

In many ways, the PhD journey has reminded me of a long, endurance-based strength run, much like a marathon, one I have yet to attempt. Over the course of this journey, I have experienced some of the most extreme highs and lows of my life, which, in turn, have made it even more meaningful to me.

I have been fortunate to receive comprehensive and continuous support from my advisor, Prof. David Gross. His vivid imagination – often illuminating what felt like a long and dark tunnel – his sense of humor, and his remarkable patience made every discussion not only insightful but also enjoyable. David, I am deeply grateful for everything you have done for me throughout this journey. I am honored to call you my Doktorvater.

I am deeply grateful to Dr. Johan Åberg, who supported me like a coach throughout this long process. He was always there to cheer me up, to answer all my (often naïve) questions, and to engage in countless discussions without ever losing patience. The lessons I learned from those conversations have been truly invaluable. Johan, it has been a real pleasure working with you. Thank you for everything– and especially for your exceptional kindness.

I wish to express my gratitude to the other members of my defense committee, PD Dr. Rochus Klesse and Prof. Andreas Zilges, for kindly agreeing to evaluate this manuscript.

I would also like to thank all the members of the Quantum Information group for their support and companionship throughout this time. In particular, I am grateful to Mariela Boevska for her kindness and steady support. My nice officemates – Felipe Montealegre, Markus Heinrich, and Christian Gorjaew – helped make every day in the office both enjoyable and smooth. Special thanks to Christian for the humorous (and perhaps life-changing) realization that I may be addicted to sugar!

Laurens Ligthart and Paulina Goedicke brought vibrancy to the office atmosphere through their engaging and often entertaining discussions. I am especially grateful to Laurens for our many off-topic scientific conversations, which helped expanding the boundaries of my imagination. I am thankful to Lionel Dmello, Fabian Henze, Taylor Garnowski, Lilith Höddinghaus, David Wierichs, Tanmay Singal, Tjark Eilts, Mark Goh, Yaiza Aragonés-Soria, Arne Heimendahl, and Julius Zeiss for contributing to such a friendly and vibrant office environment.

A special thank you goes to Prof. Ralf Bulla, whose presence was always a pleasure and whose kindness was a quiet lesson in itself. I also thank Elisa Linnartz for her warm company.

My sincere gratitude goes to the wonderful ML4Q office team – Beate Saal, Andrea Bliesener, Marian Barsoum, Magda Baer Radermacher, Eva Kanis, and Yan Chen – for consistently adding

joy, laughter, and energy to the workplace. I warmly thank Maedeh Rezaei for her cheerful companionship.

I am grateful to Petra Neubauer-Guenther, Mr. Sindermann and Konstantinia Schäfer for their generous support and kindness.

I would like to thank my collaborators–David Gross, Johan Åberg, Sören Wilkening, and Robert Derichs–for the many engaging and fruitful discussions we shared.

I would like to express my sincere gratitude for the financial support received from Bundesministerium für Bildung und Forschung (BMBF) under the QuBRA project, and from Germany's Excellence Strategy via the Cluster of Excellence – Matter and Light for Quantum Computing (ML4Q).

I gratefully acknowledge the authors and contributors of the latex-mimosis template for providing a well-designed framework that greatly facilitated the preparation of this thesis.

During some of the most challenging days of self-doubt on this journey, I found strength in recalling the kind words of Mrs. Sha'ban Pour (my sports teacher), Mrs. Kiani (my English teacher), Mr. Parsa (my geometry teacher), and Prof. Bernd Rosenow, my undergraduate advisor. Their encouragement stayed with me over the years, and my appreciation for them is truly beyond words.

I am also deeply grateful to Prof. Aghil Yousefi-Koma, my former boss at the University of Tehran, as well as to Prof. Mahmoud Mani and Prof. Ali Nobari at the Polytechnic University of Tehran, for their guidance and support during a time when I was uncertain about changing my field of study from Aerospace Engineering to Physics – a decision I now look back on with great satisfaction.

I would like to thank my dear friends Shadi Tasdighi, Bahareh Esteki, Helma Vakili, Ameneh Sheikhan and Zeinab Shafiee for their kind companionship throughout this long journey. Their presence made even the most difficult days more bearable.

Last but certainly not least, I extend my deepest gratitude to my family – having them in my life is truly the greatest stroke of luck I could have ever asked for. I am especially grateful to my mother, whose unwavering support and unconditional love have carried me through the toughest times. Her strength, wisdom, and compassion have taught me how to survive when everything seemed impossible. I am also thankful from the bottom of my heart to my sister, brothers, and my nephew and niece – without their kindness, encouragement, and love, I could not have made it through this journey and even life.

To the love of my life, Reza: Thank you for always standing by my side, especially during the darkest days. Your presence has been a mirror, allowing me to see myself with greater clarity. Thank you for never giving up, even when that reflection was challenging for both of us. I am incredibly fortunate to have you in my life, and I deeply appreciate you for being such a kind and loving soul– one who has expanded the boundaries of kindness in my heart and mind.

Finally, I hold the loving memory of my father close to my heart. A man driven by a rare curiosity for truth and the fearless spirit of a fighter. Though he is no longer here so I could kiss his hands in gratitude, his presence – and even his absence – have deeply shaped who I am. I am immensely proud to be his daughter, and the strength of his memory continues to guide me on my path...

Bibliography

1. S. Aaronson and P. Rall. "Quantum Approximate Counting, Simplified". In: *Symposium on Simplicity in Algorithms*. Society for Industrial and Applied Mathematics, 2020, pp. 24–32. ISBN: 9781611976014. DOI: 10.1137/1.9781611976014.5. URL: http://dx.doi.org/10.1137/1.9781611976014.5.
2. M. Agrawal, N. Kayal, and N. Saxena. "PRIMES Is in P". *Annals of Mathematics* 160:2, 2004, pp. 781–793. ISSN: 0003486X. URL: http://www.jstor.org/stable/3597229.
3. A. Ambainis. "Quantum search algorithms". *SIGACT News* 35:2, 2004, pp. 22–35. ISSN: 0163-5700. DOI: 10.1145/992287.992296. URL: https://doi.org/10.1145/992287.992296.
4. S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. 1st. Cambridge University Press, USA, 2009. ISBN: 0521424267.
5. J. Baker. "The DRAGON system–An overview". *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23:1, 1975, pp. 24–29. DOI: 10.1109/TASSP.1975.1162650.
6. J. Baker. "More Visible Speech". *Journal of The Acoustical Society of America - J ACOUST SOC AMER* 52, 1972. DOI: 10.1121/1.1982148.
7. L. E. Baum. "An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process". In: 1972. URL: https://api.semanticscholar.org/CorpusID:60804212.
8. L. E. Baum and J. A. Eagon. "An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology". *Bulletin of the American Mathematical Society* 73, 1967, pp. 360–363. URL: https://api.semanticscholar.org/CorpusID:14153120.
9. L. E. Baum and T. Petrie. "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". *The Annals of Mathematical Statistics* 37:6, 1966, pp. 1554–1563. DOI: 10.1214/aoms/1177699147. URL: https://doi.org/10.1214/aoms/1177699147.
10. L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains". *The Annals of Mathematical Statistics* 41:1, 1970, pp. 164–171. DOI: 10.1214/aoms/1177697196. URL: https://doi.org/10.1214/aoms/1177697196.
11. L. E. Baum and G. R. Sell. "Growth transformations for functions on manifolds." *Pacific Journal of Mathematics* 27:2, 1968, pp. 211–227.
12. T. Bayes. "An essay towards solving a problem in the doctrine of chances". *Philosophical Transactions of the Royal Society of London* 53, 1763, pp. 370–418.

13. M. Boyer, G. Brassard, P. Høyer, and A. Tapp. "Tight Bounds on Quantum Searching". *Fortschritte der Physik* 46:4–5, 1998, pp. 493–505. ISSN: 1521-3978. DOI: 10.1002/(sici)1521-3978(199806)46:4/5<493::aid-prop493>3.0.co;2-p.

14. G. Brassard, P. Høyer, and A. Tapp. "Quantum counting". In: *Automata, Languages and Programming: 25th International Colloquium, ICALP'98 Aalborg, Denmark, July 13–17, 1998 Proceedings 25*. Springer. 1998, pp. 820–831.

15. C. Cade, M. Folkertsma, I. Niesen, and J. Weggemans. "Quantifying Grover speed-ups beyond asymptotic analysis". *Quantum* 7, 2023, p. 1133. ISSN: 2521-327X. DOI: 10.22331/q-2023-10-10-1133.

16. H. B. Callen. *Thermodynamics and an introduction to thermostatistics;2nd ed.* Wiley, New York, NY, 1985. URL: https://cds.cern.ch/record/450289.

17. S. Chatrchyan et al. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". *Physics Letters B* 716:1, 2012, pp. 30–61. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.021. URL: https://www.sciencedirect.com/science/article/pii/S0370269312008581.

18. P. Cheeseman, B. Kanefsky, and W. M. Taylor. "Where the really hard problems are". In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'91. Morgan Kaufmann Publishers Inc., Sydney, New South Wales, Australia, 1991, pp. 331–337. ISBN: 1558601600.

19. N. Chopin. "A Sequential Particle Filter Method for Static Models". *Biometrika* 89, 2001. DOI: 10.1093/biomet/89.3.539.

20. N. Chopin and O. Papaspiliopoulos. *An introduction to sequential monte carlo*. 2020. ISBN: 978-3-030-47844-5.

21. S. A. Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. Association for Computing Machinery, Shaker Heights, Ohio, USA, 1971, pp. 151–158. ISBN: 9781450374644. DOI: 10.1145/800157.805047. URL: https://doi.org/10.1145/800157.805047.

22. T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006. ISBN: 0471241954.

23. P. Dagum, A. Galper, and E. Horvitz. "Dynamic network models for forecasting". In: *Uncertainty in artificial intelligence*. Elsevier. 1992, pp. 41–48.

24. P. Dagum, A. Galper, E. Horvitz, and A. Seiver. "Uncertain reasoning and forecasting". *International Journal of Forecasting* 11:1, 1995, pp. 73–87.

25. R. Derichs. "Quantum Search Optimisation for Known Solution Distributions". Bachelor thesis. Universität zu Köln, 2023.

26. A. Doucet, N. Freitas, and N. Gordon. "An Introduction to Sequential Monte Carlo Methods". *Sequential Monte Carlo Methods in Practice. Springer, Berlin*, 2001. DOI: 10.1007/978-1-4757-3437-9_1.

27. B. Dumont, S. Fichet, and G. Von Gersdorff. "A Bayesian view of the Higgs sector with higher dimensional operators". *Journal of High Energy Physics* 2013:7, 2013, pp. 1–37.

28. V. Dunjko, Y. Ge, and J. I. Cirac. "Computational Speedups Using Small Quantum Devices". *Phys. Rev. Lett.* 121, 25 2018, p. 250501. DOI: 10.1103/PhysRevLett.121.250501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.121.250501.

29. V. Eshaghian and D. Gross. *Data for Benchmarking PGGS agains QSearch*. 2025. URL: https://github.com/VahidehE/Particle-Guided-Grover-s-Search-.git.

30. V. Eshaghian, S. Wilkening, J. Åberg, and D. Gross. *Data for explicit run-times for hybrid SAT-solvers*. 2024. URL: https://github.com/SoerenWilkening/QuantumSchoening.

31. V. Eshaghian, S. Wilkening, J. Åberg, and D. Gross. *Runtime-coherence trade-offs for hybrid SAT-solvers*. 2024. arXiv: 2404.15235 [quant-ph]. URL: https://arxiv.org/abs/2404.15235.

32. V. Eshaghian, S. Wilkening, J. Åberg, and D. Gross. "Runtime–coherence trade-offs for hybrid SAT-solvers". *IEEE Transactions on Quantum Engineering*, 2025, pp. 1–25. DOI: 10.1109/TQE.2025.3563805.

33. R. Fisher and F. Yates. *Statistical methods for research workers*. Edinburgh Oliver & Boyd, 1925.

34. N. Gordon, D. Salmond, and A. Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". *IEE Proceedings F (Radar and Signal Processing)* 140, 2 1993, pp. 107–113. DOI: 10.1049/ip-f-2.1993.0015. eprint: https://digital-library.theiet.org/doi/pdf/10.1049/ip-f-2.1993.0015. URL: https://digital-library.theiet.org/doi/abs/10.1049/ip-f-2.1993.0015.

35. L. K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Association for Computing Machinery, Philadelphia, Pennsylvania, USA, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

36. E. J. Hoffman, J. C. Loessi, and R. C. Moore. "Constructions for the Solution of the m Queens Problem". *Mathematics Magazine* 42:2, 1969, pp. 66–72. DOI: 10.1080/0025570X.1969.11975924. eprint: https://doi.org/10.1080/0025570X.1969.11975924. URL: https://doi.org/10.1080/0025570X.1969.11975924.

37. R. E. Kalman and R. S. Bucy. "New results in linear filtering and prediction theory", 1961.

38. R. E. Kalman. "A new approach to linear filtering and prediction problems", 1960.

39. N. Kantas, A. Doucet, S. S. Singh, J. Maciejowski, and N. Chopin. "On Particle Methods for Parameter Estimation in State-Space Models". *Statistical Science* 30:3, 2015. ISSN: 0883-4237. DOI: 10.1214/14-sts511. URL: http://dx.doi.org/10.1214/14-STS511.

40. P. Keldenich. private communication. 2024.

41. A. Y. Kitaev. "Quantum measurements and the Abelian Stabilizer Problem", 1995. arXiv: quant-ph/9511026 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/9511026.

42. G. Kitagawa. "Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models". *Journal of Computational and Graphical Statistics* 5:1, 1996, pp. 1–25. DOI: 10.1080/10618600.1996.10474692. eprint: https://www.tandfonline.com/doi/pdf/10.1080/10618600.1996.10474692. URL: https://www.tandfonline.com/doi/abs/10.1080/10618600.1996.10474692.

43. A. H. Land and A. G. Doig. "An Automatic Method of Solving Discrete Programming Problems". *Econometrica* 28:3, 1960, pp. 497–520. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1910129.

44. P. S. Laplace. "Memoir on the Probability of the Causes of Events". *Statistical Science* 1:3, 1986, pp. 364–378. ISSN: 08834237, 21688745. URL: http://www.jstor.org/stable/2245476.

45. L. A. Levin. "Universal Sequential Search Problems". *Probl. Peredachi Inf.* 9:7, 1973, pp. 115–116.

46. J. Liu, R. Chen, and T. Logvinenko. "A theoretical framework for sequential importance sampling and resampling". *Sequential Monte Carlo Methods in Practice*, 2001, pp. 1–24.

47. A. Markov. "Extension of the law of large numbers to dependent quantities". *Izvestiia Fiz.-Matem. Obsch. Kazan Univ.,* 1906, pp. 135–156.

48. A. Montanaro. "Quantum speedup of branch-and-bound algorithms". *Physical Review Research* 2:1, 2020. ISSN: 2643-1564. DOI: 10.1103/physrevresearch.2.013056. URL: http://dx.doi.org/10.1103/PhysRevResearch.2.013056.

49. R. A. Moser. "Exact algorithms for constraint satisfaction problems". PhD thesis. ETH Zurich, 2013. ISBN: 978-3-8325-3369-4. DOI: 10.3929/ETHZ-A-009755667. URL: https://d-nb.info/1032358084.

50. R. E. Neapolitan. "Probabilistic reasoning in expert systems - theory and algorithms". In: 1989. URL: https://api.semanticscholar.org/CorpusID:5473785.

51. J. Neyman and E. S. Pearson. "On the problem of the most efficient tests of statistical hypotheses". *Philosophical Transactions of the Royal Society of London*, 1933, pp. 289–337.

52. J. Neyman and E. S. Pearson. "On the Use and Interpretation of Certain Test Criteria for Purposes of Statistical Inference". *Philosophical Transactions of the Royal Society of London*, 1928. DOI: 10.2307/2331945.

53. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

54. J. Pearl. "Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning". In: *Proc. of Cognitive Science Society (CSS-7)*. 1985.

55. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988. ISBN: 1558604790.

56. B. Risti, M. Arulampalam, and A. Gordon. *Beyond Kalman Filters: Particle Filters for Target Tracking*. 2004.

57. S. Särkkä and L. Svensson. *Bayesian Filtering and Smoothing*. 2nd ed. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2023.

58. T. Schöning. "A probabilistic algorithm for k-SAT and constraint satisfaction problems". In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*. 1999, pp. 410–414. DOI: 10.1109/SFFCS.1999.814612.

59. U. Schöning and J. Torán. *The Satisfiability Problem: Algorithms and Analyses*. Mathematik für Anwendungen. Lehmanns Media, 2013. ISBN: 9783865416483. URL: https://books.google.de/books?id=VnHoCQ%C2%A7QBAJ.

60. P. Swerling. "Optimum linear estimation for random processes as the limit of estimates based on sampled data". *IRE Wescon Convention Record* 4, 1958, pp. 158–163.

61. G. S. Tseitin. "On the complexity of derivation in propositional calculus". *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, 1983, pp. 466–483.

62. A. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society* 42:1, 1936, pp. 230–265. DOI: 10.2307/2268810.

63. C. Zalka. "A Grover-based quantum search of optimal order for an unknown number of marked elements", 1999. arXiv: quant-ph/9902049 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/9902049.