Assessing fault-tolerant quantum advantage for k-SAT with structure

Martijn Brehm[1] and Jordi Weggemans[2]

[1]University of Amsterdam, Amsterdam, the Netherlands

[2]QuSoft & CWI, Amsterdam, the Netherlands

For many problems, quantum algorithms promise speedups over their classical counterparts. However, these results predominantly rely on asymptotic worst-case analysis, which overlooks significant overheads due to error correction and the fact that real-world instances often contain exploitable structure. In this work, we employ the hybrid benchmarking method to evaluate the potential of quantum Backtracking and Grover's algorithm against the 2023 SAT competition main track winner in solving random k-SAT instances with tunable structure, designed to represent industry-like scenarios, using both T-depth and T-count as cost metrics to estimate quantum run times. Our findings reproduce the results of Campbell, Khurana, and Montanaro (Quantum '19) in the unstructured case using hybrid benchmarking. However, we offer a more sobering perspective in practically relevant regimes: almost all quantum speedups vanish, even asymptotically, when minimal structure is introduced or when T-count is considered instead of T-depth. Moreover, when the requirement is for the algorithm to *find* a solution within a single day, we find that only Grover's algorithm has the potential to outperform classical algorithms, but only in a very limited regime and only when using T-depth. We also discuss how more sophisticated heuristics could restore the asymptotic scaling advantage for quantum backtracking, but our findings suggest that the potential for practical quantum speedups in more structured k-SAT solving will remain limited.

1 Introduction

Quantum computing holds the promise of revolutionizing numerous fields by solving computational problems faster than classical computers. Optimization problems—which have a special status as they arise almost everywhere in practice—are also commonly identified as candidates to benefit from quantum computing [1].

But how does one show that such a quantum speedup over classical algorithms exists? In theoretical computer science, the de facto way to do this is to compare *asymptotic worst-case complexity*, where the complexity can be measured at different levels of abstraction (like the number of gates, queries to a certain function, etc.). Take, for example, constraint satisfaction problems (CSPs), a broad class of combinatorial problems where one must find an assignment to variables that satisfies a given set of local constraints.

Martijn Brehm: m.a.brehm@uva.nl, https://staff.science.uva.nl/m.a.brehm

Jordi Weggemans: https://jordiweggemans.github.io/research/

arXiv:2412.13274v3 [quant-ph] 15 Jan 2026

Boolean satisfiability k-SAT is a canonical CSP: each constraint is a clause containing at most k literals, and the task is to decide whether a single assignment satisfies all clauses simultaneously. For k-SAT, the best *known* classical algorithms that succeed on every input yield provable upper bounds (up to polynomial factors) on the number of elementary classical gates of $2^{n(1-c/k+o(1/k))}$ where $c > 0$ is an algorithm-dependent constant [30, 31, 33]. In contrast, for any fixed k, Grover's algorithm solves k-SAT with bounded error using at most $2^{n/2}$ quantum gates (again up to polynomial factors), giving for large constant k an as-good-as quadratic *asymptotic* improvement in the number of "native operations".

However, such an asymptotic worst-case analysis has two major shortcomings when it comes to what one wants to know in *practice*:

(i) one might not be interested in the worst-case performance—which considers the worst possible performance amongst all problem instances—as the algorithm only has to perform well on a small subset of all possible instances that are encountered in real-life scenarios;

(ii) quantum and classical computers use different types of hardware with different 'unit costs' (i.e. implementation resources), so X quantum gates cannot be directly compared with Y classical gates. Moreover, these abstract measures of complexity might not be easily translatable to universal measures like time, memory, energy usage, etc., one actually is interested in.

To cover item (i) classical algorithms are often compared empirically by measuring their performance on benchmark sets. Through decades of progress, we have classical algorithms that can exploit the structure in real-world SAT instances well-enough to solve many large instances efficiently [20].

This is not yet feasible for most quantum algorithms, as quantum hardware is still in its infancy. How then can we study the performance of quantum algorithms on instances of real-life interest, while also making assumptions about hardware that does not exist as of yet to take into account item (ii)?

To tackle point (ii), Campbell, Khurana and Montanaro proposed an elegant approach to empirically study the performance of a quantum backtracking algorithm [29] and Grover's algorithm [22] on random instances of CSPs (k-SAT and graph colourability), using a cost model that accounts for error-correction [12]. To go beyond asymptotic worst-case analysis, they write out the constants in the query complexity upper bound of their algorithms, which include as parameters the number of variables n and (for backtracking) the size of the classical backtracking tree $\mathcal{T}$. To estimate the scaling of $\mathcal{T}$ for a class of random problem instances, they sample random instances and compute a fit of the median value of $\mathcal{T}$, giving them query complexity upper bound in terms of just n for both algorithms. They then argue that (if one does not account for the classical processing cost required to decode the surface code) there is a regime such that the estimated number of queries translates into a quantum runtime of ~ 1 day, while classical would require anywhere between 15 and 500 years [12].

While this seems to show the possibility of significant quantum speedups on these problems, there are several objections one could make regarding Ref. [12]. First, the work only considers random instances at the threshold value a_k of the clauses-to-variable ratio (defined below): SAT instances at the threshold are believed (and experimentally verified) to be amongst the hardest instances [13], implying that they effectively exhibit no structure. Indeed, most classical SAT solvers are empirically tuned to exploit structure, meaning that the algorithms are not optimized to solve these types of problems. Second, Ref. [12] employs a median fit to estimate a parameter involved in the query complexity

upper bound: this approach *underestimates* this "upper bound", due to an analogue of Jensen's inequality for medians [28]. Third, the quantum algorithm is highly parallelized in many aspects—most notably in its T-gate implementation, allowing T-depth to be considered instead of T-count—an advantage not given to the classical solver. It is known that allowing a moderate amount of classical parallelization can make even a fully quadratic speedup fail to result in a quantum advantage when considering a reasonable maximum on the allowed computing time [6]. Fourth, the quantum backtracking algorithm only *detects* a solution, but does not find it. This makes the algorithm far less useful than its classical counterpart, in particular for practical applications.[1] Using binary search this can be turned into a search algorithm, but this would involve an additional $\mathcal{O}(\log \mathcal{T}) = \mathcal{O}(n)$ multiplicative overhead. Fifth and final, the detection algorithm requires an upper bound on $\mathcal{T}$ to run correctly, which costs an additional $\mathcal{O}(n)$ runs of the binary search algorithm to estimate.[2] This additional $\mathcal{O}(n^2)$ slowdown could offset the observed speedup.

More recently, Cade, Folkertsma, Niesen and Weggemans proposed an approach that overcomes some of the previously mentioned obstacles [10, 11]. In this approach, coined *hybrid benchmarking* in Ref. [5], one tightens a query complexity upper bound by parametrizing in many more features of the input instance (i.e. beyond just input size). As long as these features are efficiently *classically* computable one can evaluate the query complexity upper bound exactly (as opposed to estimating these features, yielding a lower bound only applicable to random data [12]). While Ref. [12] gives a somewhat positive view of the possibilities of achieving a quantum speedup on instances that are solvable within a day, the works of [5, 10, 11] give a much more sobering picture for the problems they study.

This work aims to study the performance of quantum backtracking and Grover's algorithm on k-SAT by combining the best of both approaches[3]: with the hybrid benchmarking of [5, 10, 11] we can evaluate relatively tight query complexity upper bound of any problem instance (solving the first and second objection listed above), which we use to study random k-SAT distributions that have a tunable amount of "structure", specifically designed to mimic industry-like instances [3, 21]. We then turn the query complexity upper bound into an upper bound on the quantum runtime using the setup of [12], where we consider T-depth as well as T-count (solving the third objection). Finally, we use binary search to turn the detection algorithm into a search algorithm, and use the search algorithm to estimate the required tree size (solving the fourth and fifth objections).

While Ref. [12] shows that on completely *random* instances at the satisfiability threshold (which are believed to have no exploitable structure) Grover's algorithm and a quantum backtracking detection algorithm can beat classical solvers when the maximum allowed computing time is a day, our guiding question will be:

---

[1]Even in the main track of the SAT competition, whose winner Ref. [12] compares against, it is required to provide certificates in the satisfiable and even the unsatisfiable cases.

[2]In Ref. [12] they do estimate a fit of $\mathcal{T}$, which could in principle be used to configure the detection algorithm. There are two potential problems with this: first, being a fit of the median value for $\mathcal{T}$, we should expect this bound to be incorrect half of the time, so performance guarantees will only be valid on half of the random instances. Second, this means the algorithm can only work when applied to instances from the family it was trained on; in particular, this means that we cannot be sure whether the algorithm works on a specific instance encountered in practice, as we do not have an estimate for its value of $\mathcal{T}$.

[3]The quantum algorithms considered in [5, 10, 11] were essentially classical algorithms with Grover subroutines, meaning that at most quadratic speedups could be achieved only at the subroutine level. Therefore, focusing on quantum backtracking and vanilla Grover as case studies, as done in Ref. [12], seems to have more potential since both algorithms are fully quantum.

Can fault-tolerant quantum algorithms provide relevant speedups in more structured (and thus more practically motivated) instances of k-SAT?

Contributions   Our main contributions are as follows.

- We provide explicit query complexity upper bound of the quantum walk detection algorithm by Belovs (i.e., including all constants and other relevant factors), lowering the complexity by a factor of approximately 100 over previously known configurations [7, 12, 29]. We provide a similar result for a binary search algorithm based on this detection algorithm.

- These query complexity upper bound have been implemented into code to estimate the performance on arbitrary k-SAT instances. This is achieved by running a classical backtracking algorithm to collect all the required parameters involved in the bounds; these are then combined with the surface code resource estimations from Ref. [12] to estimate a quantum runtime. We applied the same approach to evaluate query complexity upper bound of Grover's algorithm as presented in [10].

- We present results of our method applied to random SAT instances with tunable structure, reproducing the results of Ref. [12] when using T-depth (which requires high parallelization) and completely unstructured SAT instances. Beyond this regime, however, we find that almost all quantum speedups vanish, even asymptotically, when minimal structure is introduced or when T-count is considered instead of T-depth.

- When the objective is to *find* a solution within a single day, we find that only Grover's algorithm has the potential to outperform classical algorithms. However, this occurs only in a very limited regime and only when using T-depth. We show that if current quantum algorithms are restricted from using such parallelization, there is no potential for a speedup that is useful within a one-day computing limit. We further argue that even with improved heuristics, the potential for quantum backtracking algorithms remains limited.

Related work   There are several lines of work that aim to assess the performance of quantum algorithms beyond standard worst-case asymptotic analysis. Since we cannot (yet) run or simulate quantum algorithms on large, arbitrary inputs, each of these approaches proposes a different way to work around this limitation.

Above, we already mentioned a number of works, conceptually similar to ours, that fall under the umbrella of *hybrid benchmarking* [5, 10–12]. These works parametrise complexity upper bounds of quantum algorithms in terms of additional, efficiently computable features of the input, yielding much tighter expressions. Whenever these features can be computed classically, one can evaluate the quantum complexity on arbitrary inputs. However, computing the features typically requires solving the problem classically, so this approach is particularly suitable for problems where the quantum advantage is expected to be small (for instance, polynomial). A related work studies the performance of the Grover Quantum Approximate Optimization Algorithm on random 3-SAT instances, giving numerical evidence for a quadratic speed-up over classical random sampling [36]. The authors already note, however, that this advantage is expected to diminish when compared to practical classical SAT solvers, except perhaps on highly unstructured instances—an observation consistent with our conclusions.

Another approach is to simulate quantum algorithms directly at the circuit level. For exact simulation (i.e., state-vector simulation), the required memory scales exponentially with the number of qubits, which already limits the size of the classical problem instances one can explore: solving an n-bit CSP with a quantum algorithm requires at least n qubits merely to write down the output string, making large-scale simulations infeasible. Approximate simulation methods, such as tensor-network techniques [26], can handle somewhat larger instances, but only when the quantum computation generates limited entanglement. Generally, deeper circuits tend to produce higher entanglement, and since the algorithms we study have very large circuit depth, we expect these methods to perform poorly on the instances we consider (although it may still be worthwhile to test this).

Organisation of paper In the next section, we introduce the studied algorithms and for each provide a query complexity upper bound, we then introduce the distributions of SAT instances that we study, and motivate our cost model for the quantum algorithms (which we alter slightly from Ref. [12]). In the section after, we present our results: the observed complexity of the various algorithm on SAT instances with increasing structure. In the final section we conclude.

2 Algorithms, instances and cost model

Classical backtracking

Backtracking is a classical technique for solving constraint satisfaction problems (CSPs).

Problem 1 (Constraint satisfaction problem)**.** *Given a predicate $P : \{0,1\}^n \to \{\text{true, false}\}$, find an $x \in \{0,1\}^n$ such that $P(x) = \text{true}$, or output "not found" if $P(x) = \text{false}$ for all $x \in \{0,1\}^n$.*

We call the bit strings $x \in \{0,1\}^n$ *assignments*. Solving a CSP naively requires checking all 2^n assignments in the worst case. However, for some CSPs you can (sometimes) detect whether a partial assignment can or cannot be extended to a solution. Formally, this extends the predicate to $P : \{0,1,*\}^n \to \{\text{true, false, indeterminate}\}$, where an '$*$' means a variable is unassigned. When you realize an assignment cannot be extended into a solution, you no longer need to check all assignments that extend it. This is the idea underlying *backtracking*.

Specifically, a backtracking algorithm comes equipped with an implementation of the predicate[4] $P : \{0,1,*\}^n \to \{\text{true, false, indeterminate}\}$ and a function $h : \{0,1,*\}^n \to \{1,\dots,n\}$, called a *heuristic*, which, given a partial assignment, returns the next variable to check. The backtracking algorithm starts by creating an empty assignment $x = * \cdots *$. It then runs $h(x)$, which outputs a variable (e.g., variable 2), and x is updated so that this variable is set to true (e.g., set $x := *1* \cdots *$). It then calls $P(x)$: if $P(x) = \text{true}$, x is a solution, and we output it; if $P(x) = \text{indeterminate}$, we extend x using h (i.e., call $h(x)$ to obtain another variable and set it to true). However, if $P(x) = \text{false}$, the current assignment x cannot be extended to a solution. This means we can invert our previous decision (e.g., set variable 2 to false instead) and continue. Eventually, the algorithm either finds a solution or exhausts all possibilities and ends up with an empty assignment again, concluding "not found."

---

[4]Note that the specific implementation matters. A more advanced predicate could recognize that a partial assignment is mapped to true or false earlier than a more basic one, at the cost of additional processing time.

One can view this procedure as exploring a tree, where each node corresponds to a partial assignment x. The root node corresponds to the empty assignment $* \cdots *$, and each node has two children: one where the next variable has been set to true, and one where it is false. Leaves correspond to assignments where $P(x)$ returned true or false (dead-ends, complete assignments, or solutions). If $\mathcal{T}$ is the size of the tree, then the backtracking algorithm makes $\mathcal{T}$ queries to P and h, and the hope is that $\mathcal{T} \ll 2^n$, so that a large improvement is made over brute-force search.[5]

2.1 Quantum backtracking (detection)

By a result of Montanaro, any such classical backtracking algorithm exploring a tree of size $\mathcal{T}$ can be turned into a quantum algorithm that detects the existence of a solution with only $\mathcal{O}(\sqrt{\mathcal{T}n})$ queries to P and h, which is almost quadratically better [29]. This is achieved by running a quantum walk algorithm starting from the root of the backtracking tree, using a quantum walk algorithm due to Belovs that can start from an arbitrary starting distribution [7]. We provide the details in Section A; summarizing, Belovs' result states that, given access to an oracle that determines whether a vertex in a graph is a solution (P in this case), the quantum walk algorithm detects the existence of a solution using $\mathcal{O}(\sqrt{RW})$ queries, where W is the total weight of the graph, and R is the effective resistance from the support of the starting distribution to the set of solutions. This algorithm requires a priori knowledge of upper bounds on R and W, which determine the complexity of the algorithm. Since our tree has unit weights, and since the effective resistance from root to leaves in a tree is at most the depth n of the tree, we recover the complexity $\mathcal{O}(\sqrt{\mathcal{T}n})$.

In Section A.2, we derive an explicit query complexity upper bound of the quantum walk detection algorithm by Belovs (i.e., including all constants and other relevant factors). Given a desired error probability δ, we carefully balance the number of parallel repetitions of the algorithm, the precision of the quantum phase estimation subroutine, and constants in the quantum walk unitary, in order to minimize the total number of queries. This already yields an query complexity upper bound that is a factor ~ 100 better over previously known configurations [7, 12, 29].[6]

Recall that we want to evaluate this query complexity upper bound for specific SAT instances, given a backtracking algorithm for SAT (i.e. given P and h), meaning we need to compute W and R. We will attempt to compute W and R exactly. Note that this is unrealistic: in practice, one would have to estimate W and R, which requires additional queries to do, and may not yield W and R exactly, but a loose upper bound on them. We will account for these overheads in the next section when we turn detection into search. Hence, we think of the complexity of the detection algorithm as an optimal baseline.

We can estimate W and R by running the classical backtracking algorithm in time $\mathcal{O}(\mathcal{T})$ to construct the backtracking tree. We can then count the nodes to give us $W = \mathcal{T}$ in time $\mathcal{O}(\mathcal{T})$ and compute the effective resistance R in time $\mathcal{O}(\mathcal{T}^3)$.[7] Unfortunately, this

---

[5]Note that a satisfiable instance of your CSP might be solved using far fewer than $\mathcal{T}$ queries, as $\mathcal{T}$ is the size of the entire tree, while one might find a particular solution without searching the entire tree.

[6]This is in terms of total queries. If we allow parallelisation and compare the maximum number of sequential queries, the speedup will be smaller than 100. Indeed, we could have optimised for less sequential queries, but that would have resulted in far more total queries. As explained in the introduction, we aim to study the total T-count, so we optimised for that.

[7]The effective resistance of a graph can be computed by inverting the Laplacian matrix of the graph (see e.g. Theorem 2.2 in [17]), which takes $\mathcal{O}(\mathcal{T}^3)$ using a naive matrix multiplication algorithm.

last step quickly becomes infeasible for large instances, so we settle for the upper bound $R \leq n$ in all our experiments. We note that we can reduce the complexity of computing R to just $\mathcal{O}(t^3)$, where t is the number of solutions, however in practice this was still too slow.[8]

Our choice of P and h, i.e., our actually used backtracking algorithm, is the same as in Ref. [12]. The heuristic h checks variables in order of their number of appearances: e.g., if variable x_i occurs most often in our formula (negated or not), followed by x_j, and so on, the first variable we consider is x_i, the second is x_j, and so on. We always first set a variable to false. The predicate P is defined to iterate over all clauses and literals in order. If one finds a literal is indeterminate, we consider $P(x)$ indeterminate (i.e., even if a later literal in the same clause is satisfied). Otherwise, it behaves as expected: if all literals in all clauses are unsatisfied, we output false, and if one literal in each clause is satisfied, we output true. Our motivation for this choice is to keep P and h as simple as possible, which we come back to in the discussion.

2.2 Quantum backtracking (search)

The above quantum backtracking algorithm only *detects* whether there exists a solution, i.e. whether a SAT instance is satisfiable or not. For most practical applications it is desirable to also find the solution if the formula is satisfiable. In our backtracking context (where the graph is a tree), we can turn detection into search by a standard binary search algorithm at the cost of a $\mathcal{O}(\log \mathcal{T}) = \mathcal{O}(n)$ overhead. However, to guarantee the n consecutive detection runs are still correct with the desired success probability, we also need to amplify the success probability to $\mathcal{O}(1/n)$, requiring $\mathcal{O}(\log n)$ parallel repetitions.

Given this search algorithm, we can estimate the upper bound on $\mathcal{T}$ which the detection algorithm needs to run correctly. This estimation relies on the fact that the detection algorithm requires the upper bound on $\mathcal{T}$ only in the negative case, i.e., for unsatisfiable instances. Hence, when the algorithm outputs "no marked elements exist" we can always trust it (with the given error probability): if we had given the algorithm an unsatisfiable instance then it would be correct, but if we had given the algorithm a satisfiable instance then it wouldn't have output negatively, regardless of $\mathcal{T}$.

Thus, we should only doubt the correctness of the detection algorithm when it outputs "marked elements exist". But our binary search outputs a candidate solution. If the detection algorithm tells us "marked elements exist", we can force it to find a solution, which we can verify using one query to P. If this solution is not valid, we can conclude "no marked elements exist" with small failure probability.

We can now estimate $\mathcal{T}$ as follows. Define $\mathcal{T}' = 1$ and run binary search. If it outputs "no marked elements exist", we trust it. Otherwise, we check the candidate solution x by running $P(x)$. If it is correct, we output it. Otherwise, we double $\mathcal{T}'$ and repeat. Once $\mathcal{T}' \geq \mathcal{T}$ becomes a correct upper bound, the procedure is guaranteed to work with the configured success probability, so we require $\lceil \log \mathcal{T} \rceil \in \mathcal{O}(n)$ repetitions. This algorithm is due to Ref. [29, p. 9].

This yields a usable search algorithm which makes $\mathcal{O}(\sqrt{\mathcal{T}}nn^2 \log n)$ queries to P and h and which does not require any prior knowledge to run (unlike the detection algorithm). In Section A.3, we derive an explicit expression of this bound keeping track of all relevant factors (i.e., without asymptotic notation). To evaluate this query complexity upper

---

[8]This relies on two observations. First, we can remove branches leading to non-solution leaves. Second, long straight paths can then be reduced to single edges. This leaves a system of $\mathcal{O}(t)$ equations, instead of $\mathcal{O}(\mathcal{T})$ equations.

bound for specific SAT instances, recall that we can run the classical backtracking algorithm to determine $\mathcal{T}$. In addition, we determine the depth of the first solution that the backtracking algorithm finds (note that we can compute this easily using h). This allows us to bound the number of repetitions of the detection algorithm in the final iteration. This can have a significant effect, as the last run dominates the complexity since earlier runs have complexity $\mathcal{O}(\sqrt{\mathcal{T}'n})$ with $\mathcal{T}'$ exponentially smaller than $\mathcal{T}$.

2.3 Grover's algorithm

Grover's quantum search algorithm allows one to search a list of N elements with $\mathcal{O}(\sqrt{N})$ queries to the list. Specifically, given oracle access to a function $f : \{0, 1, ..., N-1\} \to \{0, 1\}$, with high probability the algorithm outputs some x such that $f(x) = 1$ if it exists, or detects that no such x exists, using only $\mathcal{O}(\sqrt{N})$ queries to f. In the case of SAT, we would search over all $N = 2^n$ assignments, yielding a search algorithm which makes $\mathcal{O}(\sqrt{2^n})$ queries to the predicate P.

In Section B, we state a very optimised explicit version of this query complexity upper bound due to Ref. [10]. This bound is parameterised in terms of both the list size N and the number of solutions t in the list. Hence, to evaluate this bound for a given SAT instance, we need to compute the total number of satisfying assignments to a given SAT instance. Since we already constructed the backtracking tree, we can compute t by taking all solution leaves and counting the number of ways in which this assignment can be extended.

2.4 Classical SAT solver

Historically, many cutting-edge classical SAT solvers have been backtracking algorithms, one example being the famous DPLL algorithm [14, 15]. However, since the mid-90s, SAT solvers have evolved beyond this framework to a framework called *conflict-driven clause learning* (CDCL). Upon reaching a contradiction, they extend the formula with a clause expressing what combination of variables led to the contradiction, so that the mistake is prevented later on. Moreover, instead of simply backtracking up one level, they can realize that several previous levels cannot contain a solution, and they "backjump" up multiple levels. This more general approach yields better performance in practice [27]. This means our classical baseline should not be a backtracking algorithm, but rather one of these more general CDCL solvers. Importantly, this implies that the almost quadratic speedup that quantum backtracking yields is relative to a subpar classical algorithm, harming the possibility of a quantum speedup.

As a classical baseline, we chose the SBVA-CaDiCaL solver: the winner of the 2023 SAT competition, which was the most recent SAT competition at the time.[9] This algorithm first pre-processes a SAT instance using the SBVA algorithm [23], and then solves the processed instance using the CaDiCaL 1.9.5 solver [8]. Our experiments involve SAT instances with varying and increasing clause sizes k. We noticed that for $k \geq 4$, the pre-processor started to dominate the runtime and slow down the solving process. Hence, we decided to drop the pre-processing and simply run CaDiCaL 1.9.5 by itself. During our work, version 2.0.0 of CaDiCaL was released, so we opted to use this version instead [9]. For this algorithm, the complexity measure will simply be the observed runtime.

---

[9]Results of the 2023 SAT competition can be found at https://satcompetition.github.io/2023/results.html.

k	3	4	5	6	7	8	9	10	11	12
α_k	4.27	9.93	21.12	43.27	87.79	176.54	354.01	708.92	1418.71	2838.28

Table 1: Satisfiability threshold ratios $\alpha_k = m/n$ of clauses and variables in a random k-SAT instance, as taken from Table 10 in Ref. [12].

2.5 Instances

We consider random k-SAT instances, which we generate using the CNFgen tool.[10] Specifically, each clause is picked uniformly randomly and independently from all possible clauses, excluding those which contain duplicate variables. In addition, we consider random SAT instances that have an increasing amount of "structure." There is an increasing body of work substantiating the notion of structure in SAT instances [3]. We use a simplified version of the similarity-popularity model from [21], which are specifically designed to exhibit some characteristics of many real-world SAT instances, which works as follows. Say we wish to sample a SAT formula with n variables and m clauses. Assign each variable x_i and clause j a uniformly random angle $\theta \in [0, 2\pi)$. To sample the formula, iterate over each clause j and sample its k variables according to the following distribution:

$$\mathbb{P}[x_i \text{ in clause } j] = \frac{1}{1 + \left(\frac{i(\pi - |\pi - |\theta_i - \theta_j||)}{N}\right)^{1/\beta}},$$

where N is a normalization coefficient. When a variable is added to a clause, it is negated with probability $1/2$. Intuitively, the occurrence of i in the denominator means we expect variable x_i to occur much more often than variable x_{i+1} ("popularity"), while $|\theta_i - \theta_j|$ means variables and clauses with similar angles are more likely to end up together ("similarity"). We get to set the parameter β.[11] As we increase β, we decrease the popularity and similarity effects, and hence the exploitable structure in the instances. In particular, as $\beta \to \infty$, we recover the uniformly random model outlined above.

For both models, we consider 3, 4, ..., 12-SAT, choosing the number of clauses m such that the ratio m/n is the threshold of satisfiability α_k. This is defined so that as $n \to \infty$, a random k-SAT instance is satisfiable with probability 1 for any $\alpha < \alpha_k$ and with probability 0 for any $\alpha > \alpha_k$. The existence of such thresholds (known as the Satisfiability Conjecture) has been proven for large k [16]; however, for small values of $k \geq 3$, only bounds are known. We list the ratios we use in Table 1, which we took from Table 10 in Ref. [12].

We argue that increasing k is another method of reducing exploitable structure. First, SAT solvers have a harder time making use of larger clauses, as finding contradictions (i.e., clauses that are unsatisfied by a current assignment) or inferring that certain variables must be true (i.e., through unit propagation – the observation that if a clause contains a single non-unsatisfied literal, that literal must be set to true) will be harder. Additionally, the satisfiability threshold is known to grow exponentially with k, implying that (for random k-SAT), increasing k allows you to add exponentially more constraints without harming satisfiability. Finally, this is reflected in the so-called *strong exponential time hypothesis* (SETH) [24]. Let $s_k = \inf \{c : k\text{-SAT can be solved in } 2^{cn}\text{poly}(m) \text{ time}\}$. The

---

[10]See https://massimolauria.net/cnfgen/.

[11]In [21], β is denoted by T as it can intuitively be understood as some notion of "temperature," but we have switched notation to prevent confusion with the "T-gate." Similarly, they denote the normalization coefficient N by R, which we have changed to prevent confusion with the effective resistance R.

strong exponential time hypothesis asserts that for every $\epsilon > 0$, there exists a k such that $s_k \geq 1 - \epsilon$, i.e. , $\lim_{k \to \infty} s_k = 1$. Therefore, as k becomes very large, this hypothesis conjectures that the optimal SAT solver approaches a scaling of $\sim 2^n \text{poly}(m)$, matching brute-force search.

Hence, varying k and/or β allows us to create distributions of SAT instances that are expected to have varying amounts of exploitable structure.

2.6 Cost model

The cost of the classical SAT solver will simply be the measured runtime. For the quantum algorithms, we evaluate a query complexity upper bound for the given SAT instance, which for the considered algorithms can be directly translated to the T-count or T-depth, due to the results in Ref. [12]. While Ref. [12] takes T-depth of the resulting quantum circuit as their measure of complexity, we instead optimize for the both the T-count and T-depth. Let us summarize briefly what these two measures are.

We assume the quantum circuit implementing the algorithm is decomposed in Clifford, Toffoli and T-gates, and will be encoded using the surface code [19]. Clifford gates can be implemented using state injection, which can be done parallel to the implementation of a subsequent Tofolli or T gate. Since Toffoli gates can be implemented using a single layer of T gates [34] (using a total of 7 T-gates), Ref. [12] argues that the only relevant gate count for this cost model is then the so-called T-depth of the circuit, which is defined as the total number of groups of T-gates which can be performed simultaneously. By using time-optimal methods [18], T-gates can be implemented fault-tolerantly using a state injection technique where a T-state is prepared offline. This comes at the cost of a huge overhead in the total number of qubits that are needed, but this means that every single layer of T-gates can be implemented at a time roughly on the order of the measurement time. Hence, the total time cost in this model is simply assumed to be the T-depth multiplied by the measurement cost. They also assume that repetitions of the quantum algorithms (to amplify success probability) are done in parallel, adding nothing to the total runtime. By considering measurement times of $10^{-7}, 10^{-8}$ and 10^{-9} seconds (which they call plausible, realistic, and optimistic) they give indications of the quantum runtime.

Recall that we use the same P and h oracles as Ref. [12]. As such, we can use the circuit implementations of Ref. [12] to translate our query complexity upper bound to a gate complexity upper bound. However, we believe T-count, which is simply the total number of T gates, might be a fairer cost model than T-depth, as using the latter requires significant parallelization, which is not granted to the classical algorithms. However, to allow for fair comparison with Ref. [12] and to leave the final judgement of what cost model is most relevant to the reader, we will present results on both cost models.

We will write c_q for the time needed to implement a single (layer of) T-gate(s). Finally, we assume repetitions of the quantum algorithms (which we need to do to guarantee the required success probability, see Section A) are all done in serial instead of parallel, as done in Ref. [12], to further limit the parallelization in the quantum algorithm.

3   Benchmarking results

We sampled SAT instances for the values of n, β, k listed in Section E, all at the satisfiability threshold ratio between the number of variables and the number of clauses, as given

10

in Table 1.[12] For an equal number of satisfiable and unsatisfiable instances (30 each for backtracking, 50 each for CaDiCaL), we compute the median runtime (CaDiCaL) or median T-depth and T-count (quantum algorithms) for each n, and then take a linear least squares fit of the logarithm of these medians. We configure the quantum backtracking algorithms to run with error probability at most $\delta = 1/1000$.[13]

This ends up giving us a grid over k and β. For each entry (k, β) we have a fit of the classical runtime, denoted as $2^{s_c \cdot n + i_c}$, and a fit of the quantum algorithm's T-depth or T-count denoted $2^{s_q \cdot n + i_q}$ (where we use this notation for both T-count and T-depth, though from context it will be clear which we mean), where $s_c, i_c, s_q, i_q \in \mathbb{R}$. The T-depth or T-count can then directly be converted to a run-time estimate by $c_q 2^{s_q \cdot n + i_q}$ for some constant $c_q > 0$.

Recall that we think of increasing k and β as decreasing the exploitable structure in the instances. Hence, we think of the top-left portion of this grid as being very structured, and the bottom-right portion as highly unstructured. It is exactly this bottom-right (i.e. random instances for $k = 12$) where [12] observe a quantum speedup. The question becomes: what happens as we exit this bottom-right part, and increase the amount of exploitable structure? Studying these results yields three main conclusions.

Our code and data is attached as Ref. [2].

3.1 The four regimes

Consider Table 2a, which displays which algorithm achieves the best scaling in terms of runtime (classical) and T-depth (quantum): the lowest constant s in $2^{s \cdot n + i}$. Across the grid, we observe 4 somewhat distinct regimes of SAT instances, reminiscent of a *phase diagram* as encountered in physics. For highly structured instances (top-left) the classical SAT solver easily scales better than all quantum algorithms (**blue**). Moving to the bottom-right, removing some structure, we enter a regime where the quantum backtracking detection algorithm (studied in [12]) starts to scale better than classical (**red**). However, recall that this algorithm does not find a solution unlike all the other considered algorithms, and requires a priori knowledge of the tree size $\mathcal{T}$ to run. To fix this, we needed to turn it into a binary search algorithm, allowing us to then estimate $\mathcal{T}$, suffering an $\mathcal{O}(n^2 \log n)$ overhead. As we remove even more structure from the instances there comes a regime where this binary search algorithm also starts to scale better than the classical SAT solver (**green**). Finally, as we go towards instances that contain barely any exploitable structure, we reach a point where Grover's algorithm starts to scale better than any of the other algorithms (**yellow**).[14]

These four regimes are in accordance with the SETH: as k grows we expect the classical scaling should approach $\sim 2^n \mathrm{poly}(m)$ for both the modern SAT solver as well as any backtracking algorithm. Quantum backtracking detection will gain a less-than quadratic speedup over this, while the search algorithm will scale even slightly worse. Grover will then eventually scale best, as it achieves a full quadratic speedup over classical. We can think of increasing β as having a similar effect: removing structure harms the modern

---

[12]We note that the lower bound on n for fitting the classical runtime is often quite high. For small k, the runtime initially grows quite fast, and only then stabilises into a pattern of the form $2^{a \cdot n + b}$. We attempted to lower bound n to exclude the small n which behave erratically. This is visualised in Fig. 1.

[13]E.g. Ref. [12] uses $\delta = 1/10$, providing a bigger advantage to the quantum algorithms. We recover their results despite this extra disadvantage.

[14]Except for the one outlier exactly at the right-bottom corner.

T-depth

(a) Scaling (b) One day, $c_{\mathsf{q}} = 10^{-9}$ (c) One day, $c_{\mathsf{q}} = 10^{-6}$

T-count

(d) Scaling (e) One day, $c_{\mathsf{q}} = 10^{-9}$ (f) One day, $c_{\mathsf{q}} = 10^{-6}$
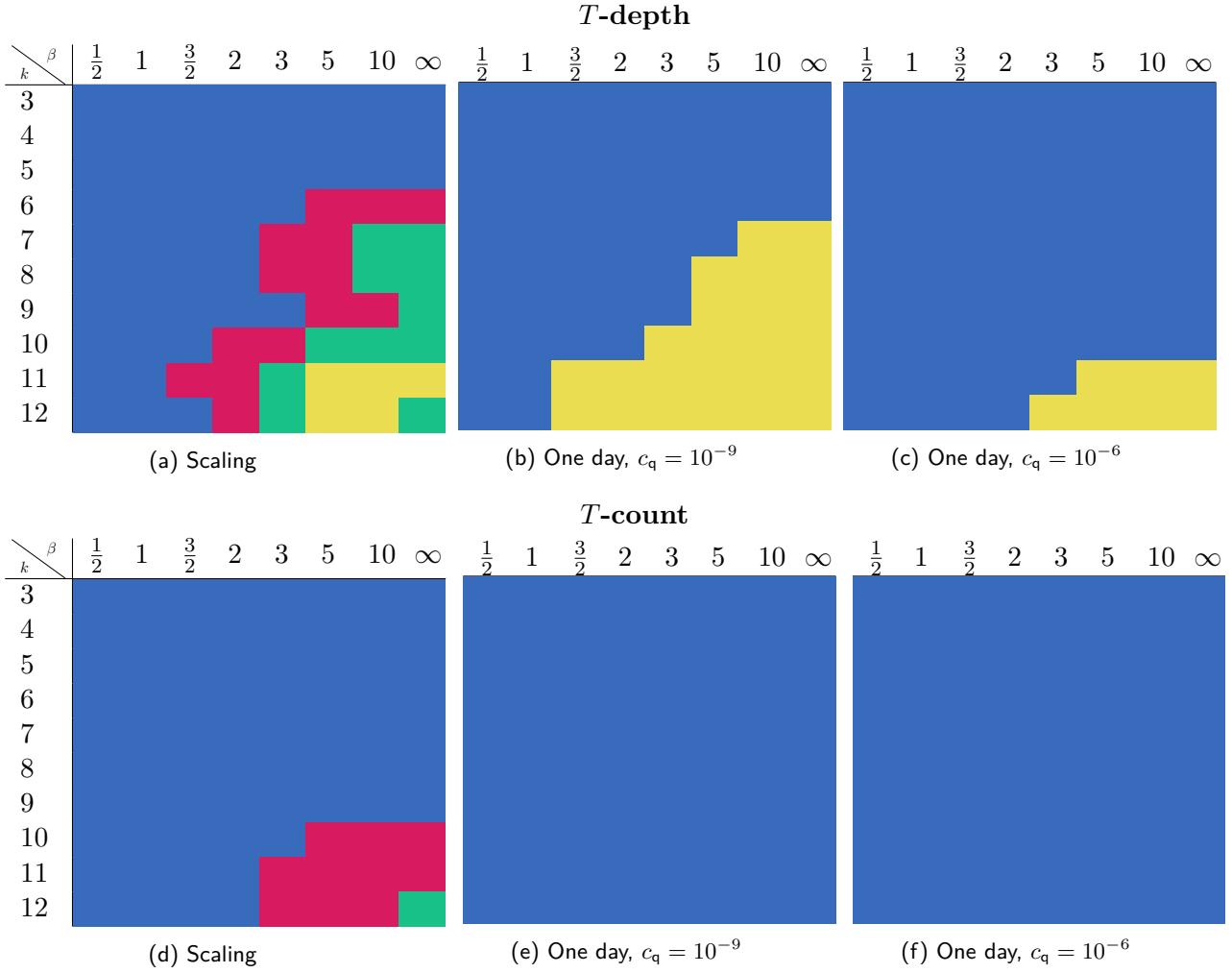
Table 2: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain an equal number of satisfiable and unsatisfiable instances (where lower β means more structure). Cells are coloured based on the best algorithm for the given performance measure. In Tables 2a and 2d we colour a cell **blue** if the classical algorithm *scales* best (i.e. $s_{\mathsf{c}} < s_{\mathsf{q}}$), **red** if the quantum backtracking *detection* algorithm scales best, **yellow** if Grover scales best, and **green** if quantum backtracking *search* scales better than classical and Grover. In the other tables, we translate T-depth and T-count to runtime using two measurement times $c_{\mathsf{q}} = 10^{-9}$ and $c_{\mathsf{q}} = 10^{-6}$ and colour based on which algorithm can solve the largest instances in one day given their fits (using the same colour codes).

classical solver most, then the detection algorithm, and then the search algorithm, while Grover is agnostic to all of this.

Some representative examples of these regimes are given in Table 3. This table shows the estimated *scaling* of the four different algorithms (in terms of runtime for classical and in terms of T-depth for quantum) for several selected values of β. These four regimes indicate that the classical algorithm outperforms the quantum algorithms on the more structured instances. Though the quantum backtracking algorithm is also able to exploit structure, much of this gain is lost when one is also interested in finding the solution. Only when barely any structure is left does Grover start beating the other algorithms. This means there might be a place for quantum backtracking to solve semi-structured practical SAT instances that are too large for classical solvers.

	High ($\beta = 0.5$)	Middle ($\beta = 2$)	Low ($\beta = 3$)	None ($\beta = \infty$)
Classical	**0.0804**	0.432	0.507	0.563
Detection	0.282	**0.368**	0.394	0.424
Search	0.366	0.461	**0.46**	0.514
Grover	0.509	0.511	0.513	**0.509**

Table 3: Exponent s of the estimated runtime 2^{sn+i} for classical algorithm and T-depth for quantum algorithms on 11-SAT with an equal mix of satisfiable and unsatisfiable instances. For the constant overhead i suffered by the algorithms, see Table 9. The values of β refer to the amount of structure in the SAT instances. Recall that in the "High" structure regime, the classical solver scales best. In the "Middle" structure regime detection scales best. In the "Low" structure regime, binary search starts to beat classical and Grover (though it never beats detection, as it runs detection multiple times). Finally, in the "None" structure case Grover starts scaling better than both binary search and classical.

	High ($\beta = 0.5$)	Middle ($\beta = 2$)	Low ($\beta = 3$)	None ($\beta = \infty$)
$c_q = 10^{-6}$	n.a.	$\approx$ universe age	$\gg$ universe age	7 hours
$c_q = 10^{-7}$	n.a.	3 millennia	$\gg$ universe age	always
$c_q = 10^{-8}$	n.a.	5 hours	$\gg$ universe age	always

Table 4: Crossover times as per Eq. (1) for different values of c_q. In every column, we compare the classical algorithm to the quantum algorithm representative of the given regime, i.e. whose scaling is bolded in Table 3. We translate the estimates of the T-depth of the quantum algorithms to runtime by multiplying with c_q. The values of β refer to the amount of structure in the SAT instances.

However, so far we only discussed asymptotic scaling. Converting the T-depth estimates to actual estimated runtimes, the potential for quantum backtracking quickly diminishes. Consider three different T-gate implementation times c_q: $10^{-6}, 10^{-7}$ and 10^{-8} seconds.[15] In Table 4 we use these three values to compute the crossover time for the representative quantum algorithms in these four regimes. The *crossover time* is the time at which the classical runtime matches the quantum runtime (quantum suffers more constant overhead than classical but scales better, so after enough time has passed quantum should obtain an advantage). Formally, we solve $2^{s_c \cdot n + i_c} = c_q \cdot 2^{s_q \cdot n + i_q}$ for n and then evaluate either side for that n.

Although with a very optimistic $c_q = 10^{-8}$ the detection algorithm can achieve a crossover time of a few hours, for larger c_q it quickly grows to be infeasible. It may seem surprising that the crossover time can change so drastically when c_q drops by one order of magnitude. However, as we show in Section C, solving the above for the crossover time gives:

$$t_{\text{crossover}} = c_q^{\frac{s_q}{s_c - s_q} + 1} \left(2^{\frac{s_q(i_q - i_c)}{s_c - s_q} + i_q} \right). \tag{1}$$

This shows that when the scaling of the quantum and classical algorithm are close, small changes in c_q can drastically change (1). Far more sobering, the more practically usable backtracking search algorithm has a crossover time far exceeding the age of the universe for all constants. This was not just the case in this one example of the regimes where binary search scales better than classical: in all such regimes in Table 9 we observe a crossover time of at least 10000 millennia. Even when going to measurement time $c_q = 10^{-9}$, the crossover time is still at least a year, and often still many, many millennia.

---

[15]We generally omit the unit 'seconds" when talking about values of c_q from now on.

In contrast, Grover's algorithm has a much smaller amount of overhead, leading to an immediate crossover point for the smallest two values of c_q and a crossover point of a few hours for the larger value of c_q.[16] However, this is only for the most random instances ($\beta = 0$), and in the cases where Grover scales best but β is larger, the advantage of Grover over classical is much smaller resulting in a crossover time of at least several years.

The above discussions is only limited to using T-depth as the estimated cost metric of the quantum algorithms. Studying the T-count yields an even more sobering picture. Table 2d shows the best scaling algorithms in terms of T-count. Just as before we observe that classical scales best initially, while for less structured SAT instances detection starts scaling better, and for even less structured instances binary search also start scaling better than classical. As argued above, by the SETH, we expect to observe the same four regimes for T-count, but we conjecture that the values of k we considered are still too small to fully observe all regimes. That is, we imagine the regimes to be shifted down as compared to the T-depth (as T-depth is a more quantumly favourable cost model).

Note that since speedups in the T-count model only start appearing for much larger k—so for much less structured instances—this limits the practical utility of these quantum speedups. Even worse, for *every* regime where a quantum algorithm scaled better than the classical algorithm in terms of T-count, a quick calculation shows that the crossover time was larger than the age of the universe!

3.2 No quantum advantage potential for one day search with structure

We saw that with respect to scaling there exist four regimes: one where classical scales best, one where quantum backtracking detection scales best, one where quantum backtracking search beats classical and Grover, and one where Grover scales best.

In Table 4 we considered the crossover time in terms of T-depth. We found that only Grover's algorithm seems to offer practical utility, as the binary search algorithm required millennia to start beating classical. To test this, we used the fits of the median runtime T-depth to compute the size of the largest instance that classical and quantum algorithms can solve in one day, for measurement times 10^{-9} and 10^{-6}. We then again coloured the grid based on which algorithm can solve the largest instance in one day.

In Table 2b and Table 2c, we see the results for $c_q = 10^{-9}$ and $c_q = 10^{-6}$, respectively. We immediately observe that the quantum backtracking algorithms disappeared, showing that they are not able to beat classical solvers or Grover's algorithm, even on the least structured instances. Instead, we see that for almost all instances, the classical solver is the fastest in one day. Only for the least structured instances does Grover's algorithm outperform the classical solver, and Grover's regime grows in size as c_q becomes smaller. However, since $c_q = 10^{-9}$ is already extremely close to the clock speed of the laptop these experiments were run on, and the implementation time of a fault-tolerant quantum operation is always limited by the classical clock speed, it feels safe to assume that we are already very close to the best possible scenario here.

Moving to T-count, we repeat this test for the same measurement times in Tables 2e and 2f. We see that no quantum algorithm is able to beat the classical solver within a day, even on the most unstructured instances.

T-depth

(a) Scaling, satisfiable   (b) Scaling, unsatisfiable
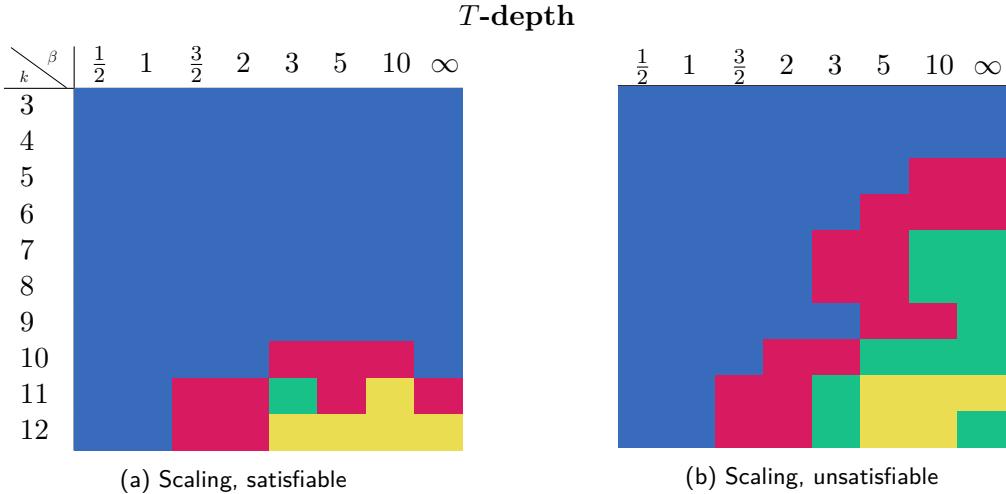
Table 5: These tables are defined the same way as Table 2a: each grid is coloured based on the scaling of the runtime and T-depth of the algorithms. Here (a) is restricted to satisfiable instances and (b) to unsatisfiable.

3.3 Getting an advantage is more difficult on satisfiable instances

Finally, we observe a sharp difference in the potential for quantum speedups between satisfiable and unsatisfiable instances. All the previous results are for an equal mix of satisfiable and unsatisfiable instances. Let us now split these up. In Table 5b we colour the grid based on which algorithm scales best on unsatisfiable instances for T-depth. We see that the results look remarkably similar to the result for a mix of satisfiable and unsatisfiable which we saw in Table 2a. In contrast, Table 5a shows the best scaling algorithm on satisfiable instances using T-depth, and the classical algorithm occupies a much larger area. We do the same for T-count in Tables 8a and 8b in the Appendix, which yields a similar result.

We believe there are two possible explanations for this: (i) at least in comparison with Grover, classical algorithms might scale better as the number of solutions grows as it might terminate earlier, for which we give a mathematical argument in Section D; and (ii) the CaDiCaL algorithm might be empirically optimized to quickly work towards solutions on structured instances, which might only help if there is a solution to begin with.

Other data and results We listed more detailed results in Section E. Specifically, we show the grids over β and k containing the actual fits (i.e. as in Table 3) one for only satisfiable instances, one for unsatisfiable, one for an equal mix, and all of these for T-count, for T-depth and for query complexity.

4 Conclusion and discussion

Taking into account an overhead for error correction and a reasonable maximum allowed computing time, we have seen that only Grover's algorithm is able to achieve practical speedups, albeit on highly unstructured SAT instances. However, once instances with even little structure are considered and we do not grant the quantum algorithms paral-

---

[16] Note that we observe this immediate speedup for Grover for the smallest possible instance, i.e. with $n = 14$ for $k = 11$. However, since $\alpha_{11} = 1418.71$, this means this instance already has over 1400 clauses.

lelization for the T-gate implementation, the classical SAT solvers perform much better than Grover's algorithm. The quantum backtracking search algorithm fails to achieve a relevant speedup in any regime of considered SAT instances. This is despite backtracking showing an asymptotic advantage (i.e. better scaling with n) in a slim regime of SAT instances with little structure. In this final section, we discuss three potential objections one might make against our findings.

4.1   What about better heuristics?

We saw that there was indeed a small regime of slightly structured SAT instances where quantum backtracking offered an *asymptotic* advantage over both classical SAT solvers and Grover, which shows that at least *in principle* quantum backtracking is able to exploit structure in SAT instances better than our current classical solvers, even when using a rather "unsophisticated" heuristic h and predicate P.

The usage of more advanced heuristics and predicates could drastically reduce the tree size $\mathcal{T}$, and hence the scaling of quantum backtracking algorithm. For such a choice of P and h, one could then also design quantum circuits to implement these functions similar to what was done in Ref. [12], and perform similar experiments to what we have done to if there is now a speedup on more structured SAT instances. However, more complicated heuristics and predicates would likely result in significantly larger gate complexity (and thus T-count) than the simple choice of h and P we considered. Hence, there is be trade-off in achieving a potentially better scaling at the cost of increasing the circuit overhead.

However, based on our results, we expect even for the smartest choices of P and h it might still be difficult to achieve a practical speedup when the maximum allowed computing time is only a day.

The argument is as follows. First, we will consider the T-depth as our cost metric. Consider again Table 3, where for $\beta = 3$ we found that quantum backtracking search scaled better than the classical solver. In Table 4 we found that even for the very small value of $c_{\mathrm{q}} = 10^{-8}$ the crossover time was larger than the age of the universe. Now suppose that we choose a smarter P and h to improve the scaling of the quantum algorithm. What kind of scaling would we need to achieve to match the size of instances that the classical solver can handle in one day? Assuming that the quantum algorithm achieves a perfect quadratic speedup (recall that we don't have a full quadratic speedup, as we lose a factor $\mathcal{O}(n^2)$, which in practice consistently gives a speedup of order $1.4 - 1.45$) over the classical backtracker, and using the overheads from Section E (which is an extremely unrealistic assumption), we can compute the required scaling of the classical backtracker to match this performance. For a more reasonable $c_{\mathrm{q}} = 10^{-6}$, we find that the classical backtracker would need to have a scaling exponent of 0.4385, meaning that this "simple" classical backtracker would have outperformed last year's SAT competition winner! For $c_{\mathrm{q}} = 10^{-7}$, we find an exponent of 0.564, which is still small but not unreasonable. However, recall that this is all under the very unrealistic assumption that using better heuristics does not lead to increased overhead. Doing the same calculation using T-count (again for $\beta = 3$), we find that the overhead for $c_{\mathrm{q}} = 10^{-6}$ is already so big that a we can never find a speedup (for any type of scaling) and that for $c_{\mathrm{q}} = 10^{-9}$ we have a scaling exponent of 0.148, which would give a breakthrough even in 3-SAT solving (using an 11-SAT algorithm).

Hence, we conclude that it might be difficult to find an improved heuristic that would allow the quantum backtracker to become useful when the maximum allowed computing time is only one day.

4.2 What about improved quantum backtracking algorithms?

An extension of the quantum backtracking algorithm we use is given by Jarret and Wang [25], who propose an efficient method to estimate the effective resistance of the graph, reducing the $\sqrt{n}$ dependence in the complexity and replacing the complexity due to binary search with $\mathcal{O}(\log(Rt))$, where R is the actual effective resistance of the graph and t is the number of solutions. We expect that the extra overhead from having to run amplitude estimation will not significantly change the above picture.

Additionally, Ambainis and Kokainis show how to change the dependence on the tree size $\mathcal{T}$ to only the subtree that the backtracking algorithm explored before finding the first solution, offering a speedup for satisfiable instances [4] and addressing the argument that classical algorithms benefit more from multiple solutions by terminating once one has been found. Moreover, since we found that the potential for a quantum speedup was much smaller for satisfiable instances compared to unsatisfiable instances, this modification has the potential to improve the competitiveness of quantum backtracking algorithms. However, since the potential for a speedup even on unsatisfiable instances (the most competitive regime for quantum) is minimal, we believe that this improvement by itself won't lead to different conclusions.

Finally, Piddock expanded Belovs' underlying quantum walk algorithm to handle search in general graphs (recall that the above algorithm only handles search on trees), building heavily on ideas from Jarret and Wang [25]. This replaces the n overhead for binary search with a $\mathcal{O}(\log^3(t))$ overhead, where t is the number of solutions [32]. However, a priori $\mathcal{O}(\log^3(t))$ is not obviously lower than $\mathcal{O}(n)$, as we have no rigorous relation between the number of variables and the number of solutions in our SAT distributions. Preliminary results from applying our methodology to Piddock's algorithm on random 3-SAT show slightly better scaling in practice than our binary search approach, but significantly higher constant overhead. The similarity of Piddock's algorithm to Jarret and Wang's further supports our claim that their algorithm most likely won't improve the competitiveness of quantum.

4.3 How reliable are our results?

An objection one might make is that our results do not include any quantification of the errors[17], so how likely is it that a reproduction of our experiments leads to the same conclusions? First, it's important to note that we employed a large number of different distributions over SAT instances (parameterized by k and β), sampling multiple instances for each distribution across various instance sizes n. For each n, we calculated the median and used it in a linear least-squares fit of the logarithm of the median. Consistently, we found that the interquartile range relative to the median was at most a few percent, with some larger outliers. However, these outliers were partially mitigated by their role in determining an exponential fit over multiple medians for different values of n. The robustness of our approach is further enhanced by the fact that the results are presented on a grid. Even if individual points contain errors, the overall depiction of the different regimes remains largely unaffected.

More importantly, the aim of this work is not to claim that we can make precise *quantitative* predictions of quantum run times for specific instances, but rather to sufficiently rigorously explore the broader *qualitative* picture. Subjecting our numbers, for example

---

[17]This also holds for Campbell, Khurana and Montenaro [12], which forms the starting point of this work.

the specific crossover times mentioned in Table 4 or the data to used the colouring of the grid in Table 2a, to large perturbations due to errors will still yield the same qualitative conclusions, as it is quantitatively consistent over all grids in Section E.

Acknowledgments

References

[1] Amira Abbas, Andris Ambainis, Brandon Augustino, Andreas Bärtschi, Harry Buhrman, Carleton Coffrin, Giorgio Cortiana, Vedran Dunjko, Daniel Egger, Bruce Elmegreen, Nicola Franco, Filippo Fratini, Bryce Fuller, Julien Gacon, Constantin Gonciulea, Sander Gribling, Swati Gupta, Stuart Hadfield, Raoul Heese, and Christa Zoufal. Challenges and opportunities in quantum optimization. *Nature Reviews Physics*, 6, 10 2024. DOI: 10.1038/s42254-024-00770-9.

[2] All the data used in this paper, and all the code used to generate this data, can be found in the following repository. URL https://github.com/martijnmartijnmartijnmartijn/Quantum-advantage-k-SAT-with-structure.

[3] Tasniem Nasser Alyahya, Mohamed El Bachir Menai, and Hassan Mathkour. On the structure of the boolean satisfiability problem: A survey. *ACM Comput. Surv.*, 55 (3), mar 2022. ISSN 0360-0300. DOI: 10.1145/3491210.

[4] Andris Ambainis and Martins Kokainis. Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 989–1002, 2017. DOI: 10.1145/3055399.3055444.

[5] Sabrina Ammann, Maximilian Hess, Debora Ramacciotti, Sándor P Fekete, Paulina LA Goedicke, David Gross, Andreea Lefterovici, Tobias J Osborne, Michael Perk, Antonio Rotundo, et al. Realistic runtime analysis for quantum simplex computation, 2023.

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

[7] Aleksandrs Belovs. Quantum walks and electric networks, 2013.

[8] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020. DOI: 10138/318754.

[9] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. DOI: 10.1007/978-3-031-65627-9̇7.

[10] Chris Cade, Marten Folkertsma, Ido Niesen, and Jordi Weggemans. Quantifying Grover speed-ups beyond asymptotic analysis. *Quantum*, 7:1133, oct 2023. ISSN 2521-327X. DOI: 10.22331/q-2023-10-10-1133.

[11] Chris Cade, Marten Folkertsma, Ido Niesen, and Jordi Weggemans. Quantum algorithms for community detection and their empirical run-times. *Quantum Information and Computation*, 24(5&6):0361–0410, 2024. DOI: 10.26421/QIC24.5-6-1.

[12] Earl Campbell, Ankur Khurana, and Ashley Montanaro. Applying quantum algorithms to constraint satisfaction problems. *Quantum*, 3:167, jul 2019. ISSN 2521-327X. DOI: 10.22331/q-2019-07-18-167.

[13] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, page 331–337, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1558601600.

[14] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, jul 1960. ISSN 0004-5411. DOI: 10.1145/321033.321034.

[15] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, jul 1962. ISSN 0001-0782. DOI: 10.1145/368273.368557.

[16] Jian Ding, Allan Sly, and Nike Sun. Proof of the satisfiability conjecture for large k. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, STOC '15, pages 59–68. ACM, 2015. DOI: 10.1145/2746539.2746619.

[17] W. Ellens, F.M. Spieksma, P. Van Mieghem, A. Jamakovic, and R.E. Kooij. Effective graph resistance. *Linear Algebra and its Applications*, 435(10):2491–2506, 2011. ISSN 0024-3795. DOI: https://doi.org/10.1016/j.laa.2011.02.024. Special Issue in Honor of Dragos Cvetkovic.

[18] Austin G Fowler. Time-optimal quantum computation, 2012.

[19] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012. DOI: 10.1103/PhysRevA.86.032324.

[20] John Franco and John Martin. Chapter 1. A History of Satisfiability. In Armin Biere, Marijn Heule, Hans Van Maaren, and Toby Walsh, editors, *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2021. ISBN 978-1-64368-160-3 978-1-64368-161-0. DOI: 10.3233/FAIA200984.

[21] Jesús Giráldez-Cru and Jordi Levy. Locality in random sat instances. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 638–644, 2017. DOI: 10.24963/ijcai.2017/89.

[22] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325–328, jul 1997. DOI: 10.1103/PhysRevLett.79.325.

[23] Andrew Haberlandt, Harrison Green, and Marijn J. H. Heule. Effective Auxiliary Variables via Structured Reencoding. In *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*, volume 271 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:19, 2023. DOI: 10.4230/LIPIcs.SAT.2023.11.

[24] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, mar 2001. DOI: 10.1006/jcss.2000.1727.

[25] Michael Jarret and Kianna Wan. Improved quantum backtracking algorithms using

effective resistance estimates. *Phys. Rev. A*, 97:022337, feb 2018. DOI: 10.1103/Phys-RevA.97.022337.

[26] Igor L Markov and Yaoyun Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, 2008. DOI: 10.1137/050644756.

[27] Joao Marques-Silva, Ines Lynce, and Sharad Malik. Chapter 4. Conflict-Driven Clause Learning SAT Solvers. In Armin Biere, Marijn Heule, Hans Van Maaren, and Toby Walsh, editors, *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2021. ISBN 978-1-64368-160-3 978-1-64368-161-0. DOI: 10.3233/FAIA200987.

[28] Milan Merkle. Jensen's inequality for medians. *Statistics & probability letters*, 71(3): 277–281, 2005. DOI: 10.1016/j.spl.2004.11.010.

[29] Ashley Montanaro. Quantum-walk speedup of backtracking algorithms. *Theory of Computing*, 14(15):1–24, 2018. DOI: 10.4086/toc.2018.v014a015.

[30] Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, October 19-22, 1997*, pages 566–574. IEEE Computer Society, oct 1997. DOI: 10.1109/SFCS.1997.646146.

[31] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *J. ACM*, 52(3):337–364, may 2005. ISSN 0004-5411. DOI: 10.1145/1066100.1066101.

[32] Stephen Piddock. Quantum walk search algorithms and effective resistance, 2019.

[33] Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999, New York, NY, USA, October 17-18, 1999*, pages 410–414. IEEE Computer Society, oct 1999. DOI: 10.1109/SFFCS.1999.814612.

[34] Peter Selinger. Quantum circuits of t-depth one. *Phys. Rev. A*, 87:042302, apr 2013. DOI: 10.1103/PhysRevA.87.042302.

[35] Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *45th Symposium on Foundations of Computer Science, FOCS 2004, Rome, Italy, October 17-19, 2004, Proceedings*, FOCS '04, pages 32–41. IEEE Computer Society, 2004. DOI: 10.1109/FOCS.2004.53.

[36] Zewen Zhang, Roger Paredes, Bhuvanesh Sundar, David Quiroga, Anastasios Kyrillidis, Leonardo Duenas-Osorio, Guido Pagano, and Kaden RA Hazzard. Grover-qaoa for 3-sat: Quadratic speedup, fair-sampling, and parameter clustering. *Quantum Science and Technology*, 10(1):015022, 2024. DOI: 10.1088/2058-9565/ad895c.

A   Explicit query complexity upper-bounds for quantum walk algorithms

This section provides explicit expressions of the query complexity of the quantum walk algorithms used in this work. We start by providing some background information on Belovs' quantum walk detection algorithm (which we defined in Section 2.1). We then provide an explicit expression of the exact query complexity of this detection algorithm. Finally, we give an upper-bound on the expected query complexity of the detection-based binary search algorithm defined in the quantum walk search algorithm Section 2.2.

A.1   Background Belovs' quantum walk

The two quantum walk algorithms solve the following problem.

Problem 2 (The graph search problem). *Let $G = (V, E, w)$ be an weighted undirected graph, where $w : E \to \mathbb{R}_+$. Let $M : V \to \{0, 1\}$ be a binary function telling you whether any vertex is "marked" or not. The goal is to determine whether there exist marked vertices, i.e. whether $|M^{-1}(1)| > 0$ or not (decision version), or if M is non-empty, to find a marked vertex, i.e. find some $v \in V$ such that $M(v) = 1$ (search version).*

A simple classical algorithm for this problem is a random walk/Markov chain. Sample a starting vertex from σ, check if this vertex is marked, and if not, choose a neighbouring vertex to move to with probability proportional to its weight, and repeat. The expected number of queries to M that such an algorithm requires is called the *hitting time* of the Markov chain from σ to M, denoted $HT_{\sigma,M}$.[18]

Szegedy showed how to take any Markov chain, and turn it into a quantum algorithm for the same problem that requires quadratically less queries to M. Specifically, he showed that given a Markov chain, one can construct a unitary W with the property that a specific quantum state $|\phi\rangle$ is invariant under W if and only if no marked vertices exist. Thus, repeatedly applying W to this state should reveal which case one is in. Szegedy showed that this requires $\mathcal{O}(\sqrt{HT_{\phi,M}})$ applications of W (i.e. queries to M) [35].

Two main drawbacks of this algorithm are that 1) it only solves the detection problem (for general graphs at least) and 2) it requires one to construct $|\phi\rangle$, which is a superposition over the entire graph (preventing usage when one only has local access to a graph, i.e., oracle access to neighbouring vertices).

Using a beautiful connection between classical Markov chains and electrical network theory, Belovs showed that the second drawback can be solved [7]. In electrical network theory, one views a graph as an electrical circuit, with edge weight corresponding to resistance at that edge. Imagine that one sends a current into the graph according to the starting distribution, i.e., a current of $\sigma(v)$ enters vertex v, and that current has to leave the graph at the marked vertices. If p_e is the current at some edge e and w_e is the weight (resistance) of that edge, recall then that the energy at that edge is p_e^2/w_e. The energy of the entire flow of current is the sum of the energy at each edge. The minimal energy of any flow of current from the starting distribution σ to the marked vertices M is called the *effective resistance*, denoted $R_{\sigma,M}$. Call the electrical flow the flow through the graph that generates the least energy. Belovs shows that the quantum state $|\phi\rangle$ describing this electrical flow (i.e. amplitude at an edge is equal to the amount of current at that edge under the electrical flow) is invariant under the quantum walk unitary if and only if marked elements exist.

---

[18]In this notation the second subscript usually refers to the set of vertices that you are trying to hit with your Markov chain; we allow ourselves to abuse this notation and write instead our oracle M that checks whether a vertex is marked.

Using this insight, he then proposes the following alteration: for each vertex v in the support of the starting distribution σ, add a new vertex v' and a new edge between v and v' with very small positive weight. Note that energy is current squared over resistance: since the weight (i.e. resistance) at these edges is small, that means the energy will be large. Hence, in the positive case, $|\phi\rangle$ will have large overlap with our starting state $|\sigma\rangle$, so that running quantum phase estimation (QPE) will output this phase 0 with probability proportional to the square of the overlap $\langle\sigma|\phi\rangle$ (independent of the number of bits of precision). To also make this work in the negative case, we would want to measure something other than phase 0, and Belovs shows that if we use $m \in O(\log\sqrt{HT_{\sigma,M}})$ bits of precision this occurs with high probability.

Thus, his algorithm is simple: run QPE on W (defined using the altered graph) and $|\sigma\rangle$ with $m \in O(\log\sqrt{HT_{\sigma,M}})$ bits of precision and output "marked elements exist" if and only if you measure phase 0. Belovs notes that $HT_{\sigma,M} = 2WR_{\sigma,M}$, where W is the sum of weights of the graph and $R_{\sigma,M}$ is the effective resistance, the sum of energy of the electrical flow. By expressing the query complexity required by the precision m in terms of the number unitary applications in QPE (which is $\mathcal{O}(2^m)$), the query complexity of Belovs' algorithm becomes $\mathcal{O}(\sqrt{WR_{\sigma,M}})$. Finally note that this means you need to know (upper-bounds) on W and $R_{\sigma,M}$ to run this algorithm.

Theorem 1 (Ref. [7], Theorem 4). *For any starting distribution σ Belovs' quantum walk (configured with upper-bounds on W and $R_{\sigma,M}$) solves the Detection problem with probability $> 1/2$ with $\mathcal{O}\left(\sqrt{WR_{\sigma,M}}\right)$ queries to M.*

A.2 Query complexity and optimization of quantum walk detection

To move beyond asymptotics, let us consider how to exactly configure Belovs' algorithm, and study the resulting query complexity. There are two questions at play here. Assume some desired success probability $1 - \delta$.

(1) How small do we make the weights at the newly added edges, i.e. how large do we want to set $\langle\sigma|\phi\rangle$? This essentially sets the success probability in the positive case, and changes the quantum walk unitary.

(2) What exact number of bits $m \in O(\log\sqrt{WR_{\sigma,M}})$ do we use for QPE? This essentially sets the success probability in the negative case. We will see that the outcome of (1) also influences the number of bits m.

Once we answer these questions, the resulting complexity is easy to express: it is exactly $2^m - 1$ calls to W and hence queries to M.

For (1), recall that in the positive case QPE outputs phase 0 with probability at least $|\langle\sigma|\phi\rangle|^2$ (i.e. regardless of m). In the proof of his Theorem 4, Belovs shows that this probability is at least $\frac{C_1}{1+C_1}$ where $C_1 > 0$ is a constant which sets the weight of the newly added edges. We therefore set C_1 such that $\frac{C_1}{1+C_1} \geq 1 - \delta$.

For (2), recall that in the negative case, we want QPE to output anything other than phase 0. Belovs shows that the overlap between the starting state $|\sigma\rangle$ and the eigenvectors of W with phase smaller than some angle Θ can be upper-bounded by $1/(2C_2)^2$ where $C_2 > 0$ is another constant. This is shown by defining

$$\Theta = \frac{1}{C_2\sqrt{1 + C_1RW}}\ ,$$

where R, W are upper-bounds on the effective resistance from σ to M and sum of weights of the graph, respectively. This means we can lower-bound the probability that QPE

outputs (the approximation) of a non-zero phase by $1 - (2C_2)^2$. However, unlike the phase 0, these small non-zero phases can't be expressed in any number of bits: this means that we need to set m such that $2^m \leq \Theta$.

However, setting m this large is not necessarily sufficient to guarantee an error probability of at most $1/(2C_2)^2$. The problem is that this only bounds the probability that QPE outputs an approximation of a non-zero phase. This leaves the possibility that such an approximation of a non-zero phase could be 0, and we need to bound the probability of this happening.

We note that Belovs' doesn't provide such an analysis, but one is given in Ref. [12, § 4.1]. However, this analysis omits the above two constants by implicitly setting $C_1 = C_2 = 1$, which yields an algorithm with a relatively small base success probability. This is compensated for by repeating the algorithm many times. Asymptotically this makes sense: it is well known that reducing the error probability of an algorithm to δ by repeating it requires $\mathcal{O}(\log 1/\delta)$ repetitions. Conversely, rewriting the above expressions involving C_1, C_2 to improve the error probability of the base quantum walk algorithm to δ requires a $\mathcal{O}(1/\delta)$ factor slowdown: exponentially worse than repeating. However, we found that the constant in the latter case is much larger than in the former case. As a result, optimizing the number of repetitions as well as the value of the constants C_1, C_2 ends up yielding an algorithm that is in our considered setting roughly a factor 100 more efficient (i.e. as compared to setting $C_1 = C_2$ as done in Ref. [12]).

Let us analyze the success probability of the algorithm with these constants included. Start by expressing the decomposition of our starting state in the eigenbasis of the quantum walk unitary as $|\sigma\rangle = \sum_j \alpha_j |\phi_j\rangle$, where $|\phi_j\rangle$ is an eigenvector of W with eigenphase θ_j. The state just before applying the inverse QFT during QPE is then given by

$$\frac{1}{\sqrt{2^m}} \sum_j \alpha_j \left(\sum_{x=0}^{2^m-1} e^{2\pi i \theta_j x} |x\rangle \right) |\phi_j\rangle.$$

As in Ref. [12], we replace the inverse QFT in QPE with a Hadamard transform, relying on the fact that we only need to distinguish a 0-eigenphase from those bounded away from 0. To see why this works, note that applying a Hadamard transform to the first register of the state above yields

$$\frac{1}{\sqrt{2^m}} \sum_j \alpha_j \left(\sum_{x=0}^{2^m-1} e^{2\pi i \theta_j x} H^{\otimes m} |x\rangle \right) |\phi_j\rangle = \frac{1}{2^m} \sum_j \alpha_j \left(\sum_{x=0}^{2^m-1} \sum_{y=0}^{2^m-1} e^{2\pi i \theta_j x} (-1)^{x \cdot y} |y\rangle \right) |\phi_j\rangle.$$

This is the state we measure, and the algorithm accepts if and only if the first register is measured as $|0^m\rangle$. By the no-signaling principle, any measurement performed on the second register does not affect the probability distribution over outcomes on the first register. If we perform a measurement on the second register in the eigenbasis $\{|\phi_j\rangle\langle\phi_j|\}$, the second register collapses to $|\phi_j\rangle$ with probability $\mathbb{P}[\phi_j] = |\alpha_j|^2$. Therefore, conditioning on the outcome ϕ_j, the probability of measuring $|0^m\rangle$ in the first register is then[19]

$$\mu_j := \left| \frac{1}{2^m} \sum_{x=0}^{2^m-1} e^{2\pi i \theta_j x} (-1)^{x \cdot 0} \right|^2.$$

By the total law of probability, the overall acceptance probability is

$$\mathbb{P}[\text{accept}] = \sum_j \mathbb{P}[\text{accept}|\phi_j]\mathbb{P}[\phi_j] = \sum_j |\alpha_j|^2 \mu_j.$$

---

[19]If we directly wrote $\mathbb{P}[\text{accept}]$ by fixing $y = 0$, we would obtain $|\sum_j \alpha_j|^2$ instead of $\sum_j |\alpha_j|^2$. This form allows us to separate the sum over j, which is necessary.

In the positive case (where a marked element exists), the starting state has a significant overlap with the eigenvector corresponding to eigenphase 0, denoted ϕ_0. Specifically, we have $\alpha_0 = \langle \phi_0 | \sigma \rangle \geq \sqrt{\frac{C_1}{1+C_1}}$. Thus, with probability $\frac{C_1}{1+C_1}$, the measurement yields an approximation of the phase $\theta_0 = 0$. Since this approximation is always 0, we succeed with the same probability, as required.

The negative case is more complicated, as we don't just want to upper bound the probability of ending up with a non-zero phase, but also that the approximations of these non-zero phases are non-zero. To achieve this, we split up the sum over j based on the size of the corresponding phase θ_j. The point of this is that we can use the effective spectral gap lemma to tightly bound the overlap between our starting state and the eigenvectors with small phases $\leq \Theta$ (i.e. close to 0). We can then deal with the large phases with a rougher bound, as these large phases are very unlikely to yield an approximation equal to 0. Specifically, we write:

$$\mathbb{P}[\text{accept}] = \frac{1}{2^{2m}} \left(\sum_{j, |\theta_j| \leq \Theta} |\alpha_j|^2 \mu_j + \sum_{j, |\theta_j| > \Theta} |\alpha_j|^2 \mu_j \right) .$$

The spectral gap lemma [7, Lemma 2] tells us that overlap between our starting state $|\sigma\rangle$ and eigenvectors with phases $|\theta_j| \leq \Theta = \frac{1}{C_2 \sqrt{1 + C_1 RW}}$ is at most $\frac{\Theta}{2} \sqrt{1 + C_1 RW}$. Note also that we always have $\mu_j \leq 1$. This means we can upper bound the first sum as $\left(\frac{1}{C_2 \sqrt{1 + C_1 RW}} \right)^2$.

We bound the second sum as $\frac{1}{2^{2m} \Theta^2 (1 - \Theta^2/6)}$ using $\mu_j \leq \frac{1}{2^{2m} \theta_j^2 (1 - \theta_j^2/6)}$ [12, § 4.1], combined with the fact that this upper bound on μ decreases as $|\theta_j|$ grows (i.e. is maximized at $\theta_j = \Theta$) and that $\sum_j |\alpha_j|^2 = 1$. Writing this out gives us:

$$\mathbb{P}[\text{accept}] \leq \frac{\Theta^2}{4}(1 + C_1 RW) + \frac{1}{2^{2m} \Theta^2 (1 - \Theta^2/6)}$$

To simplify the above we define

$$\Theta^2 = \frac{4a}{1 + C_1 RW} \quad \text{and} \quad M^2 = 2^{2m} = \frac{1 + C_1 RW}{4b} .$$

Note that before Θ depended on the constant C_2, but now depends on a. As such, from now on we write $C := C_1$. We can now simplify p as

$$\mathbb{P}[\text{accept}] \leq a + \frac{b}{a \left(1 - \frac{2}{3} \frac{a}{1 + CRW}\right)} = a + \frac{b}{a \left(1 - o(1)\right)} .$$

The above is essentially $a + b/a$, and to minimize this over a we set $1 - b/a^2 = 0$, which gives $b/a^2 = 1$, so that we set $a = \sqrt{b}$.

Recall that the success probability in the positive case is at least $C/(1+C)$, where we can pick $C > 0$. The above expression for the error probability p in the negative case contains the constant b. Let us require that the negative success probability $1 - p$ is at least $C/(1+C)$, and solve for b in terms of C:

$$\frac{C}{1+C} \leq 1 - \left(\sqrt{b} + \frac{\sqrt{b}}{1 - \frac{2}{3} \frac{\sqrt{b}}{1 + CRW}} \right) ,$$

which is the same as

$$\frac{1}{1+C} \geq \sqrt{b} + \frac{\sqrt{b}}{1 - \frac{2}{3} \frac{\sqrt{b}}{1 + CRW}} \geq \sqrt{b} + \sqrt{b}\left(1 + \frac{2}{3} \frac{\sqrt{b}}{1 + CRW}\right) \geq 2\sqrt{b},$$

where the one-to-last inequality follows because $1 + x \leq \dfrac{1}{1-x}$ since $x \in [0,1]$.

Thus, to guarantee success probability $1 - \delta$ in the positive case we pick $C > 0$ such that $\dfrac{C}{1+C} = 1 - \delta$ (and construct the quantum walk unitary). We can then set $\sqrt{b} = \dfrac{1}{2(1+C)}$ to guarantee the same success probability in the negative case (and determine the number of bits needed for phase estimation). Having guaranteed the right error probability, the query complexity becomes $2^m - 1$, where we had

$$M = 2^m = \sqrt{\frac{1 + CRW}{4\dfrac{1}{4 + 4C^2}}} = \sqrt{(1+C^2)(1 + CRW)}$$

so that we need to set $m = \lceil \log \sqrt{(1+C^2)(1+CRW)} \rceil$ (i.e. we might slightly round up m by 1 so that the query complexity is between M and $2M$).

This is all for a single run of the detection algorithm. We now consider doing l repetitions of the algorithm, which yields query complexity $l(2^m - 1)$. We now want to minimize $l(2^m - 1)$ subject to the choices of (C, l) guaranteeing error probability $\leq \delta$. We could do so analytically by appealing to, say, a Chernoff bound. For the sake of efficiency, let us work out the optimum numerically. That is, write out the error probability of a majority vote of l runs of the algorithm as

$$\delta \leq (1-p)^l \sum_{i=0}^{\lfloor l/2 \rfloor} \binom{l}{i} \left(\frac{p}{1-p} \right)^i = \frac{1}{(1+C)^l} \sum_{i=0}^{\lfloor l/2 \rfloor} \binom{l}{i} \left(\frac{\frac{C}{1+C}}{\frac{1}{1+C}} \right)^i = \frac{1}{(1+C)^l} \sum_{i=0}^{\lfloor l/2 \rfloor} \binom{l}{i} C^i \ .$$

To optimize, simply increment $l = 1, 3, 5...$, exponentially dropping the error probability. For each l, solve the above inequality for C, i.e. determine C which gives you exactly the required error probability $1 - \delta$. Then compute the number of queries $l(2^m - 1)$. Once the number of queries increases from one iteration to the next, output l and C corresponding to the smallest number of queries.

The reason this works is that we must always have $C > 1$, as otherwise the success probability of the base algorithm is $\leq 1/2$. At some point, the number of repetitions alone guarantees the required success probability, so that we can't decrease C further. At this point, the number of queries will start to grow with each iteration, whereas before that the cost of increasing n was offset in part by decreasing C.

There is one slight caveat here, which is that the query complexity we are minimizing depends on R and W, i.e. upper-bounds that are dependent on the given problem instance. Although the above finds an optimum relative to some R and W, it is not clear that this will also be optimal (or even decent) for other R and W. However, we note that $\sqrt{(1+C^2)(1+CRW)} \leq \sqrt{1+C^2} \left(1 + \sqrt{CRW} \right) = \sqrt{C(1+C^2)}\sqrt{RW} + \sqrt{1+C^2}$ so that we can rewrite the query complexity as:

$$l \cdot (2^m - 1) = l \cdot \left(2^{\lceil \log \sqrt{(1+C^2)(1+CRW)} \rceil} - 1 \right) \tag{2}$$

$$\leq l \cdot \left(2^{\lceil \log \sqrt{C(1+C^2)}\sqrt{RW} + \sqrt{1+C^2} \rceil} - 1 \right)$$

$$\leq 2 \cdot l \cdot \sqrt{C(1+C^2)}\sqrt{RW} + 2 \cdot l \cdot \sqrt{1+C^2} - l \ .$$

We run the above procedure, optimizing just for $2 \cdot l \cdot \sqrt{C(1+C^2)}\sqrt{RW}$, as the two other terms are tiny constants. This means we obtain a globally optimum choice (C, l), with the only exception being tiny instances (which we don't care about regardless).

Table 6 contains optimal (C, l) for a selection of error probabilities δ. As a comparison, the configuration of the detection algorithm given in Ref. [12] sets $\delta = 1/10$ and ends up requiring $2528\sqrt{RW}$ queries, roughly 100 more than our configuration of $26\sqrt{RW}$.

δ	C	l	Queries to quantum walk unitary
1/10	4.107	3	$25.701\sqrt{RW + 0.243} - 3$
1/100	3.698	13	$95.751\sqrt{RW + 0.270} - 13$
1/1000	3.735	23	$171.894\sqrt{RW + 0.268} - 23$
1/10000	3.778	33	$250.710\sqrt{RW + 0.265} - 33$
1/100000	3.813	43	$331.004\sqrt{RW + 0.262} - 43$
1/1000000	3.742	55	$412.068\sqrt{RW + 0.267} - 55$

Table 6: An overview of configurations of Belovs' quantum walk detection algorithm. Herem δ is a provable upper-bound on the error probability of the algorithm, C is a constant used in the definition of the quantum walk unitary, l is the number of parallel repetitions. Recall that Eq. (2) gives the number of queries in terms of C and l.

A.3 Query complexity and optimization of quantum walk detection

We now turn the above detection algorithm into a usable search algorithm. As explained, we simply perform binary search on a tree, and since the algorithm requires an upper bound on the tree size (as we use unit weights), we repeat the binary search algorithm to estimate this tree size. See Section 2.2 for the details. The overhead is a factor $\mathcal{O}(n^2 \log n)$, where n is the depth of the tree. Our question is now what the exact overhead is, including constants.

If our tree has depth n, binary search will make at most n repetitions of the detection algorithm. Each individual run of the detection algorithm is guaranteed to succeed with probability $1 - \delta$. To make sure that the entire sequence of $\leq n$ runs of the detection algorithms succeeds with the desired probability $1 - \delta$, we would need to reduce the error probability to $\mathcal{O}(1/n)$, requiring $\mathcal{O}(\log n)$ repetitions. More precisely, we should repeat l' times, where l' is the solution to

$$1 - \sqrt[n]{1 - \delta} \leq \delta^{l'} \sum_{i=0}^{\lfloor l'/2 \rfloor} \binom{l'}{i} \left(\frac{1 - \delta}{\delta} \right)^i ,$$

and this is something we can compute in code for our target δ. Suppose we have a satisfiable instance. Then there exists a specific first solution that we find, say at depth $d \leq n$ (this is because we use a fixed heuristic h). The overhead caused by binary search is then $d \cdot l' = O(n \log n)$. Instead, if we have an unsatisfiable instance, we would detect this at the root, and only have an overhead of $l' = O(\log n)$.

However, this is assuming we are in the final iteration where we have $\mathcal{T}' \geq \mathcal{T}$. Let us now consider the overhead caused by the previous iterations where we use $\mathcal{T}'$ that is too small. Clearly, we need exactly $\lceil \log \mathcal{T} \rceil$ iterations before we guarantee $\mathcal{T}' \geq \mathcal{T}$. In each such iteration we perform binary search. In principle, such a binary search procedure could terminate at a leaf at any depth (or even at the root). However, since the correctness guarantees of the detection algorithm don't hold when $\mathcal{T}'$ is too small, we can't assume that the number of steps is indeed small [20]. We have to assume the worst case of n full

---

[20]One might object that the correctness of the detection in the positive case doesn't rely on the upper bound $\mathcal{T}$. Indeed, given a satisfiable instance, the first call of the detection algorithm will detect this

iterations for each binary search call. Finally, note that the query complexity depends on the given value of $\mathcal{T}'$ and is therefore smaller for these initial iterations. Specifically, recalling the query complexity of the detection algorithm from Eq. (2), we can write the final upper-bound on the query complexity as:

$$l' \sum_{i=1}^{\lceil \log \mathcal{T} \rceil} n \cdot l \cdot \left(2^{\lceil \log \sqrt{(1+C^2)(1+CR \cdot 2^i)} \rceil} - 1 \right) ,$$

where in the positive case we can replace the factor n in the final iteration with a factor d, and in the negative case we can remove the factor n in the final iteration (as we detect unsatisfiability at the root, and therefore don't run binary search).

B   Explicit expected query complexity upper-bounds for Grover

This section provides explicit expressions for the expected query complexity of Grover's algorithm. We take this expression from Ref. [10].

Lemma 1 (Adapted from Lemma 4 in Ref. [10]). *Let L be a list, $g : L \to \{0,1\}$ a Boolean function, $N_{samples}$ a non-negative integer and $\delta > 0$, and write $t = |g^{-1}(1)|$ for the (unknown) number of marked items of L. Then, **QSearch**$(L, N_{samples}, \delta)$ finds and returns an item $x \in L$ such that $g(x) = 1$ with probability at least $1 - \delta$ if one exists using an expected number of queries to g that is given by*

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

where

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

with

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

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

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

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

---

correctly. However, as part of binary search, we then run the detection algorithm at one of the children. Although one of the children is guaranteed to contain a solution in this case, the other child needn't. We can therefore always end up running detection on a tree with on solutions, after which our correctness guarantee fades away.

In our setting, we take $N_{\text{samples}} = 0$ and a failure probability of $\delta = 1/1000$. The upper bound on the expected number of queries made by Grover's algorithm as given by Lemma 1 then simplifies to

$$E_{\text{Grover}}(N, t) \leq \begin{cases} 9.2 \lceil \log_3(1/\delta) \rceil \sqrt{N} = 64.4\sqrt{N} & \text{for} \quad t = 0 \\ \frac{9}{4} \frac{N}{\sqrt{(N-t)t}} + \left\lceil \log_{\frac{6}{5}} \left(\frac{N}{2\sqrt{(N-t)t}} \right) \right\rceil - 3 \leq \frac{3.1\sqrt{N}}{\sqrt{t}} & \text{for} \quad 1 \leq t < \frac{N}{4} \\ 2.0344 & \text{for} \quad \frac{N}{4} \leq t \leq N. \end{cases}$$

C The influence of the T-gate overhead factor on the cross-over time

For our estimates of classical and quantum scaling in solving a specific class of SAT instances, we focus on the *crossover point*. This is defined as the time at which the expected execution times for solving a given instance size with both quantum and classical algorithms are equal, assuming the quantum algorithm offers a scaling advantage over the classical one. However, the relation between the quantum "clock speed" for T-gates, needed to primarily account for error correction, and this crossover point is not straightforward, as our results in Section E show. To explain this, let us do a simple calculation which explains this behaviour.

For a given class of SAT instances, parametrized in the number of variables n, recall that we estimate the classical run time t_c as

$$t_{\text{c}} = 2^{s_c \cdot n + i_c},$$

where s_c and i_c are parameters obtained through our fitting. Similarly, for the estimated T-count of the quantum algorithm, defined as T_{q}, we have

$$T_{\text{q}} = 2^{s_q \cdot n + i_q},$$

where again s_q and i_q are obtained parameters. We can now estimate the quantum run time t_q as

$$t_{\text{q}} = c_{\text{q}} T_{\text{q}},$$

where c_{q} is the constant which characterizes the time needed to implement a single T gate. The crossover point $t_{\text{crossover}}$ is then determined as the time such that

$$t_{\text{q}} = t_{\text{c}},$$

which can be solved for n to obtain

$$n^* = \frac{\log_2 c_{\text{q}} + i_q - i_c}{s_{\text{c}} - s_{\text{q}}}.$$

Plutting this in our expression for T_{q} gives

$$\begin{aligned} t_{\text{crossover}} &= c_{\text{q}} 2^{s_q \left(\frac{\log_2 c_q + i_q - i_c}{s_c - s_q} \right) + i_q} \\ &= c_{\text{q}} \left(2^{\log_2 c_q} \right)^{\frac{s_q}{s_c - s_q}} \left(2^{\frac{s_q(i_q - i_c)}{s_c - s_q} + i_q} \right) \\ &= c_{\text{q}}^{\frac{s_q}{s_c - s_q} + 1} \left(2^{\frac{s_q(i_q - i_c)}{s_c - s_q} + i_q} \right), \end{aligned}$$

which depends non-trivially on c_q and the fitting parameters.

D The benefit of stopping earlier with multiple solutions

In this appendix we present an argument as to why classical algorithms seem to perform much better on random satisfiable instances as compared to unsatisfiable instances, comparing both to Grover. Consider a search problem over N elements where t are promised to be marked, where instances are chosen uniformly at random. The query complexity of Grover is the same for all these instances, and scales as $\mathcal{O}(\sqrt{N/t})$. Hence, the expected query complexity is also $\mathcal{O}(\sqrt{N/t})$. However, for a deterministic classical solver, it is easy to show that the expected query complexity is $\mathcal{O}(N/t)$, which means it scales quadratically better in t as compared to Grover. We take as our brute force search algorithm the naive algorithm that simply checks for every element in $[N]$ whether it is marked in their standard ordering, and terminates once it sees a marked item. This means that the query complexity is equal to the index of the smallest marked elements, which is for picking t out of N elements uniformly at random equal to i with probability

$$P(i) = \frac{\binom{N-i}{t-1}}{\binom{N}{t}}.$$

Hence, the expected query complexity is equal to

$$\sum_{i=1}^{N} iP(i) = \sum_{i=1}^{N} i\frac{\binom{N-i}{t-1}}{\binom{N}{t}} = \frac{N+1}{t+1} = \mathcal{O}(N/t),$$

which scales quadratically better than Grover in terms of t.

E Additional numerical results

β \ k	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						160 - 279	160 - 262	160 - 250
						10 - 56	10 - 55	10 - 49
4					64 - 130	64 - 113	64 - 103	64 - 94
					10 - 45	10 - 42	10 - 41	10 - 40
5				47 - 119	47 - 99	47 - 86	47 - 78	47 - 71
				10 - 45	10 - 40	10 - 41	10 - 39	10 - 34
6			30 - 91	30 - 85	30 - 71	30 - 46	30 - 60	30 - 54
			10 - 42	10 - 42	10 - 36	10 - 35	10 - 34	10 - 28
7	25 - 77	25 - 105	25 - 83	25 - 71	25 - 60	25 - 53	25 - 49	25 - 44
	10 - 47	10 - 40	10 - 35	10 - 35	10 - 31	10 - 29	10 - 29	10 - 28
8	20 - 84	20 - 80	20 - 63	20 - 61	20 - 52	20 - 47	20 - 40	20 - 40
	11 - 40	11 - 38	11 - 33	11 - 30	11 - 29	11 - 25	11 - 26	10 - 25
9	15 - 75	15 - 66	15 - 54	15 - 45	15 - 39	15 - 36	15 - 33	15 - 37
	12 - 27	12 - 24	12 - 21	12 - 21	12 - 21	12 - 21	12 - 21	11 - 24
10	15 - 102	15 - 73	15 - 54	15 - 47	15 - 42	15 - 38	15 - 36	15 - 33
	13 - 31	13 - 27	13 - 24	13 - 23	13 - 22	13 - 21	13 - 21	12 - 22
11	14 - 67	14 - 50	14 - 40	14 - 37	14 - 34	14 - 33	14 - 32	14 - 32
	14 - 21	14 - 20	14 - 20	14 - 20	14 - 19	14 - 19	14 - 19	13 - 20
12	15 - 56	15 - 41	15 - 35	15 - 33	15 - 28	15 - 29	15 - 28	15 - 30
	15 - 19	15 - 18	15 - 18	15 - 18	15 - 18	15 - 17	15 - 17	14 - 19

Table 7: Ranges of number of variables n considered for each SAT distribution (k, β). The top values in each cell are the range of n for CaDiCaL 2.0.0 and the bottom values the range of n for the backtracking algorithm. We consider every integer in these ranges. For each n, for CaDiCaL 2.0.0 we always sample 50 satisfiable and 50 unsatisfiable instance. For small k and small n the runtime initially grows quite fast, and only then stabilises into a pattern of the form $2^{a \cdot n + b}$. The relatively large lower-bounds on n is due to us selecting the point where the runtime stabilises into this pattern. No such phenomenon occurs for backtracking. For backtracking we sample 30 satisfiable and 30 unsatisfiable instances. Where the ranges overlap it therefore possible that we use different instances for the two solvers.
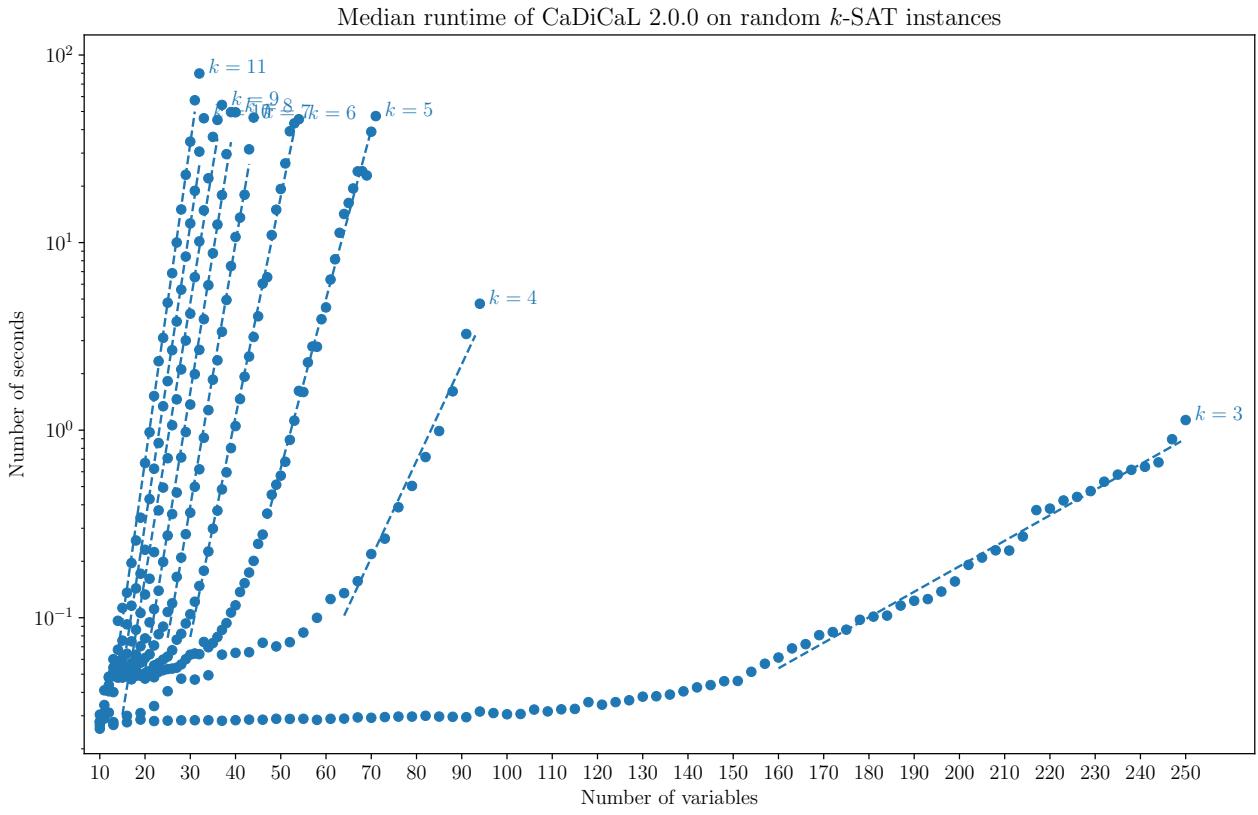
Figure 1: Scatter plot of median runtime for the CaDiCaL 2.0.0 solver on random k-SAT instances at the satisfiability threshold, see Table 1. The dotted line indicates the exponential fit of the running time. For small k you can see that we only start fitting the runtime for larger n, as the runtime seems to behave differently for small n.

T-count

(a) Scaling, satisfiable	(b) Scaling, unsatisfiable
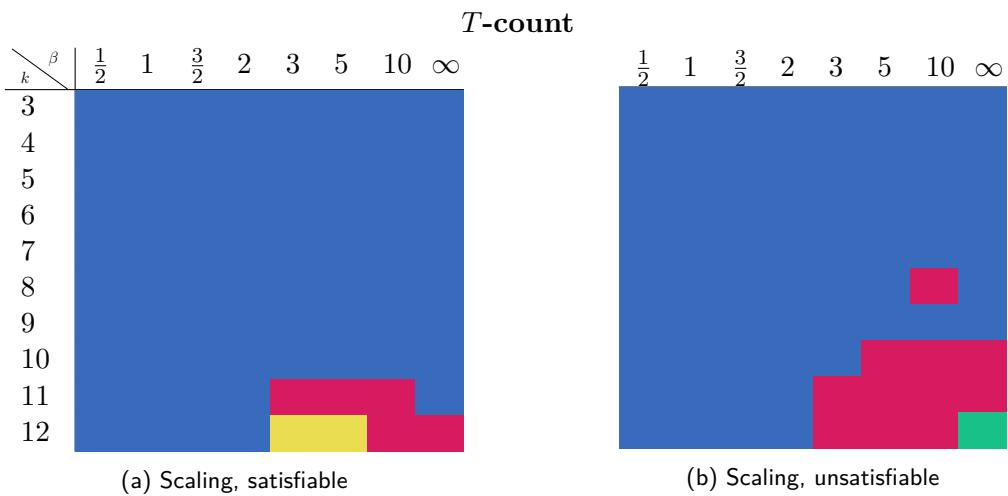
Table 8: These tables are defined the same way as Table 2a: each grid is coloured based on the scaling of the runtime and T-count of the algorithms. Here (a) is restricted to satisfiable instances and (b) to unsatisfiable.

k \ β	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						$.0369n-11.0$ $.186n+20.7$ $.237n+25.4$ $.508n+9.73$	$.0502n-12.6$ $.195n+20.6$ $.247n+25.3$ $.508n+9.72$	$.0452n-11.4$ $.203n+20.4$ $.255n+25.1$ $.507n+9.25$
4					$.0724n-9.6$ $.226n+20.5$ $.287n+25.0$ $.511n+9.79$	$.13n-12.7$ $.247n+20.4$ $.309n+24.9$ $.512n+9.77$	$.165n-14.4$ $.26n+20.3$ $.322n+24.8$ $.512n+9.77$	$.171n-14.2$ $.263n+20.3$ $.323n+24.8$ $.509n+9.36$
5				$.0951n-9.27$ $.235n+20.7$ $.294n+25.3$ $.509n+10.1$	$.18n-12.7$ $.268n+20.4$ $.333n+24.9$ $.51n+10.1$	$.245n-14.7$ $.286n+20.3$ $.351n+24.7$ $.51n+10.1$	$.29n-15.9$ $.299n+20.2$ $.362n+24.7$ $.51n+10.1$	$.3n-15.7$ $.31n+20.1$ $.385n+24.3$ $.507n+9.77$
6			$.081n-7.4$ $.24n+20.9$ $.303n+25.4$ $.509n+10.2$	$.164n-10.1$ $.266n+20.7$ $.331n+25.1$ $.509n+10.2$	$.249n-12.2$ $.3n+20.4$ $.372n+24.6$ $.51n+10.2$	$.327n-13.8$ $.319n+20.2$ $.394n+24.4$ $.51n+10.2$	$.34n-13.5$ $.331n+20.1$ $.404n+24.4$ $.51n+10.2$	$.39n-15.4$ $.343n+20.0$ $.425n+24.2$ $.506n+9.89$
7	$.00334n-4.19$ $.157n+22.1$ $.214n+26.7$ $.507n+10.3$	$.0411n-5.58$ $.225n+21.3$ $.287n+25.8$ $.509n+10.3$	$.163n-9.37$ $.274n+20.8$ $.349n+25.0$ $.509n+10.3$	$.244n-11.3$ $.298n+20.6$ $.367n+24.9$ $.509n+10.3$	$.329n-12.9$ $.326n+20.3$ $.401n+24.6$ $.51n+10.2$	$.396n-14.1$ $.347n+20.1$ $.426n+24.3$ $.51n+10.2$	$.444n-14.9$ $.358n+20.1$ $.439n+24.2$ $.51n+10.2$	$.466n-15.3$ $.367n+20.0$ $.449n+24.1$ $.511n+10.2$
8	$.00234n-3.95$ $.194n+21.9$ $.263n+26.3$ $.508n+10.4$	$.0877n-6.53$ $.248n+21.4$ $.311n+25.9$ $.508n+10.3$	$.212n-9.42$ $.295n+20.8$ $.371n+25.1$ $.509n+10.3$	$.288n-10.9$ $.32n+20.6$ $.394n+24.9$ $.509n+10.3$	$.368n-12.2$ $.344n+20.3$ $.421n+24.6$ $.51n+10.3$	$.433n-13.3$ $.368n+20.1$ $.464n+24.0$ $.51n+10.3$	$.474n-13.9$ $.377n+20.1$ $.465n+24.1$ $.51n+10.3$	$.492n-14.1$ $.389n+19.9$ $.473n+24.0$ $.511n+10.3$
9	$.00513n-3.81$ $.235n+21.6$ $.314n+25.7$ $.508n+10.6$	$.115n-6.42$ $.291n+21.0$ $.383n+25.0$ $.509n+10.5$	$.225n-8.22$ $.329n+20.6$ $.411n+24.8$ $.506n+10.6$	$.281n-8.95$ $.351n+20.4$ $.428n+24.7$ $.506n+10.6$	$.345n-9.82$ $.371n+20.2$ $.478n+24.0$ $.506n+10.6$	$.401n-10.6$ $.386n+20.1$ $.478n+24.0$ $.506n+10.6$	$.443n-11.3$ $.396n+20.0$ $.528n+23.3$ $.506n+10.6$	$.486n-12.3$ $.401n+20.0$ $.484n+24.1$ $.508n+10.5$
10	$.0196n-3.67$ $.237n+22.0$ $.301n+26.6$ $.507n+10.6$	$.182n-6.98$ $.297n+21.3$ $.368n+25.6$ $.508n+10.6$	$.299n-8.9$ $.335n+20.8$ $.423n+24.9$ $.508n+10.6$	$.366n-9.81$ $.356n+20.5$ $.44n+24.7$ $.506n+10.6$	$.439n-10.7$ $.378n+20.3$ $.462n+24.4$ $.507n+10.6$	$.499n-11.5$ $.396n+20.1$ $.49n+24.1$ $.508n+10.6$	$.543n-12.2$ $.405n+20.1$ $.506n+23.9$ $.508n+10.6$	$.529n-12.2$ $.413n+20.0$ $.494n+24.2$ $.506n+10.6$
11	$.0804n-4.04$ $.282n+21.6$ $.366n+25.8$ $.509n+10.7$	$.245n-6.81$ $.327n+21.0$ $.389n+25.5$ $.511n+10.6$	$.366n-8.63$ $.358n+20.7$ $.479n+24.2$ $.511n+10.6$	$.432n-9.52$ $.368n+20.6$ $.461n+24.7$ $.511n+10.6$	$.507n-10.5$ $.394n+20.3$ $.46n+24.7$ $.513n+10.6$	$.543n-10.8$ $.409n+20.1$ $.517n+23.9$ $.513n+10.6$	$.576n-11.2$ $.417n+20.1$ $.531n+23.7$ $.513n+10.6$	$.563n-11.8$ $.424n+20.0$ $.514n+24.1$ $.509n+10.7$
12	$.0888n-2.24$ $.299n+21.7$ $.389n+25.8$ $.515n+10.6$	$.249n-5.02$ $.343n+21.1$ $.497n+24.1$ $.566n+9.66$	$.355n-6.63$ $.356n+20.9$ $.447n+25.1$ $.52n+10.5$	$.417n-7.53$ $.386n+20.5$ $.447n+25.1$ $.52n+10.5$	$.512n-9.01$ $.402n+20.3$ $.447n+25.1$ $.52n+10.5$	$.555n-9.47$ $.423n+20.1$ $.656n+21.9$ $.525n+10.4$	$.587n-9.87$ $.43n+20.0$ $.656n+21.9$ $.525n+10.3$	$.593n-11.3$ $.431n+20.1$ $.502n+24.4$ $.512n+10.7$

Table 9: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain only an equal mix of *satisfiable* and *unsatisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum T-*depth* of the form $2^{a \cdot n + b}$. Each cell contains the exponent $a \cdot n + b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover.

k \ β	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						$.0299n-9.89$ $.187n+20.8$ $.237n+25.6$ $.44n+6.82$	$.0356n-10.5$ $.196n+20.7$ $.248n+25.4$ $.443n+6.8$	$.0327n-9.93$ $.205n+20.5$ $.258n+25.1$ $.436n+6.5$
4					$.054n-8.35$ $.227n+20.6$ $.291n+25.0$ $.477n+6.97$	$.0582n-7.84$ $.247n+20.4$ $.309n+24.9$ $.472n+7.08$	$.0571n-7.32$ $.26n+20.3$ $.324n+24.7$ $.479n+7.0$	$.0617n-7.25$ $.262n+20.3$ $.323n+24.8$ $.465n+6.81$
5				$.0713n-7.98$ $.235n+20.8$ $.295n+25.3$ $.483n+7.46$	$.103n-8.92$ $.268n+20.4$ $.332n+24.9$ $.495n+7.17$	$.0892n-7.25$ $.286n+20.3$ $.351n+24.7$ $.485n+7.45$	$.0885n-6.93$ $.299n+20.2$ $.366n+24.6$ $.485n+7.44$	$.0587n-5.12$ $.309n+20.1$ $.385n+24.3$ $.474n+7.34$
6			$.065n-6.83$ $.239n+20.9$ $.3n+25.4$ $.484n+7.66$	$.126n-8.91$ $.267n+20.7$ $.33n+25.1$ $.489n+7.54$	$.174n-10.0$ $.299n+20.4$ $.372n+24.7$ $.494n+7.41$	$.187n-9.88$ $.32n+20.2$ $.393n+24.5$ $.488n+7.72$	$.187n-9.64$ $.331n+20.1$ $.404n+24.4$ $.501n+7.37$	$.167n-8.64$ $.344n+20.0$ $.425n+24.2$ $.47n+7.53$
7	$.00228n-4.15$ $.158n+22.1$ $.217n+26.6$ $.458n+8.12$	$.0336n-5.35$ $.224n+21.3$ $.289n+25.8$ $.487n+7.62$	$.138n-8.77$ $.274n+20.8$ $.351n+25.0$ $.487n+7.73$	$.205n-10.5$ $.297n+20.6$ $.367n+24.9$ $.483n+7.87$	$.256n-11.4$ $.326n+20.3$ $.401n+24.6$ $.496n+7.55$	$.264n-11.1$ $.348n+20.1$ $.427n+24.3$ $.492n+7.72$	$.295n-11.7$ $.358n+20.1$ $.439n+24.2$ $.473n+8.12$	$.255n-10.2$ $.368n+20.0$ $.449n+24.1$ $.5n+7.71$
8	$.00223n-3.97$ $.196n+21.9$ $.264n+26.3$ $.466n+8.21$	$.073n-6.2$ $.249n+21.4$ $.311n+25.9$ $.488n+7.86$	$.183n-8.95$ $.295n+20.8$ $.371n+25.1$ $.489n+7.83$	$.244n-10.2$ $.319n+20.6$ $.391n+24.9$ $.478n+8.12$	$.288n-10.7$ $.344n+20.3$ $.421n+24.6$ $.496n+7.76$	$.328n-11.5$ $.368n+20.1$ $.464n+24.0$ $.497n+7.69$	$.349n-11.7$ $.377n+20.1$ $.465n+24.1$ $.491n+7.93$	$.3n-10.2$ $.389n+19.9$ $.473n+24.0$ $.481n+8.11$
9	$.00368n-3.83$ $.23n+21.7$ $.314n+25.7$ $.452n+8.79$	$.1n-6.29$ $.295n+21.0$ $.383n+25.0$ $.511n+7.73$	$.196n-7.97$ $.329n+20.6$ $.411n+24.8$ $.504n+7.84$	$.226n-8.3$ $.35n+20.4$ $.428n+24.7$ $.447n+8.63$	$.277n-8.93$ $.37n+20.2$ $.478n+24.0$ $.499n+7.79$	$.292n-9.07$ $.385n+20.1$ $.478n+24.0$ $.41n+9.24$	$.332n-9.8$ $.396n+20.0$ $.528n+23.3$ $.486n+8.28$	$.343n-10.1$ $.401n+20.0$ $.484n+24.1$ $.494n+8.09$
10	$.016n-3.73$ $.24n+22.0$ $.308n+26.5$ $.484n+8.24$	$.172n-7.24$ $.298n+21.2$ $.368n+25.6$ $.492n+8.0$	$.281n-9.14$ $.335n+20.8$ $.423n+24.9$ $.499n+7.89$	$.339n-10.0$ $.357n+20.5$ $.44n+24.7$ $.516n+7.73$	$.389n-10.6$ $.378n+20.3$ $.462n+24.4$ $.463n+8.64$	$.439n-11.4$ $.396n+20.1$ $.49n+24.1$ $.449n+8.73$	$.466n-11.8$ $.405n+20.1$ $.506n+23.9$ $.509n+7.87$	$.412n-10.8$ $.412n+20.0$ $.494n+24.2$ $.488n+8.29$
11	$.0838n-4.77$ $.278n+21.7$ $.366n+25.8$ $.507n+7.9$	$.243n-7.56$ $.321n+21.2$ $.389n+25.5$ $.534n+7.41$	$.359n-9.47$ $.356n+20.7$ $.479n+24.2$ $.482n+8.25$	$.414n-10.1$ $.366n+20.6$ $.461n+24.7$ $.494n+8.24$	$.485n-11.2$ $.393n+20.3$ $.46n+24.7$ $.579n+6.62$	$.508n-11.3$ $.409n+20.1$ $.517n+23.9$ $.612n+6.06$	$.534n-11.7$ $.417n+20.1$ $.531n+23.7$ $.487n+8.31$	$.464n-10.9$ $.424n+20.0$ $.514n+24.1$ $.487n+8.33$
12	$.0991n-3.37$ $.315n+21.4$ $.389n+25.8$ $.664n+5.18$	$.26n-6.21$ $.348n+21.0$ $.497n+24.1$ $.502n+8.16$	$.372n-8.12$ $.353n+21.0$ $.447n+25.1$ $.477n+8.51$	$.447n-9.39$ $.377n+20.7$ $.447n+25.1$ $.667n+5.3$	$.538n-10.8$ $.402n+20.3$ $.447n+25.1$ $.426n+9.35$	$.58n-11.4$ $.422n+20.1$ $.656n+21.9$ $.505n+8.11$	$.556n-10.8$ $.431n+20.0$ $.656n+21.9$ $.505n+8.11$	$.539n-11.4$ $.431n+20.1$ $.502n+24.4$ $.478n+8.47$

Table 10: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain only *satisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum T-*depth* of the form $2^{a\cdot n+b}$. Each cell contains the exponent $a\cdot n+b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover.

k \ β	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						$.0412n-11.6$ $.186n+20.6$ $.236n+25.3$ $.509n+10.6$	$.0586n-13.8$ $.195n+20.5$ $.247n+25.2$ $.509n+10.6$	$.0529n-12.5$ $.201n+20.4$ $.26n+24.9$ $.509n+10.1$
4					$.0843n-10.4$ $.225n+20.5$ $.281n+25.1$ $.511n+10.7$	$.146n-13.6$ $.247n+20.4$ $.313n+24.8$ $.512n+10.6$	$.188n-15.7$ $.26n+20.2$ $.324n+24.7$ $.513n+10.6$	$.19n-15.2$ $.264n+20.2$ $.327n+24.6$ $.51n+10.2$
5				$.102n-9.57$ $.235n+20.7$ $.294n+25.3$ $.509n+11.0$	$.192n-13.1$ $.267n+20.4$ $.33n+24.9$ $.511n+10.9$	$.258n-15.1$ $.287n+20.3$ $.351n+24.7$ $.51n+10.9$	$.301n-16.0$ $.3n+20.2$ $.366n+24.6$ $.511n+10.9$	$.318n-16.2$ $.31n+20.0$ $.384n+24.3$ $.508n+10.6$
6			$.0882n-7.62$ $.241n+20.9$ $.304n+25.3$ $.509n+11.1$	$.171n-10.1$ $.266n+20.7$ $.333n+25.0$ $.509n+11.1$	$.258n-12.4$ $.3n+20.3$ $.372n+24.6$ $.511n+11.0$	$.337n-13.9$ $.319n+20.2$ $.394n+24.4$ $.511n+11.0$	$.35n-13.5$ $.332n+20.1$ $.407n+24.4$ $.511n+11.0$	$.407n-15.8$ $.343n+20.0$ $.427n+24.1$ $.507n+10.7$
7	$.00394n-4.21$ $.155n+22.1$ $.21n+26.8$ $.508n+11.2$	$.0473n-5.76$ $.226n+21.3$ $.288n+25.8$ $.509n+11.1$	$.172n-9.55$ $.273n+20.8$ $.346n+25.0$ $.51n+11.1$	$.252n-11.3$ $.298n+20.5$ $.365n+25.0$ $.51n+11.1$	$.334n-12.9$ $.327n+20.3$ $.401n+24.6$ $.511n+11.1$	$.402n-14.1$ $.347n+20.1$ $.427n+24.3$ $.512n+11.1$	$.451n-14.9$ $.358n+20.1$ $.439n+24.2$ $.512n+11.1$	$.476n-15.4$ $.367n+20.0$ $.449n+24.1$ $.512n+11.1$
8	$.00241n-3.93$ $.192n+22.0$ $.255n+26.5$ $.509n+11.2$	$.0994n-6.8$ $.247n+21.4$ $.311n+25.8$ $.509n+11.2$	$.223n-9.54$ $.294n+20.8$ $.372n+25.1$ $.509n+11.2$	$.297n-10.9$ $.32n+20.6$ $.394n+24.9$ $.51n+11.2$	$.374n-12.2$ $.345n+20.3$ $.421n+24.6$ $.511n+11.2$	$.437n-13.2$ $.368n+20.1$ $.455n+24.2$ $.512n+11.2$	$.487n-14.0$ $.378n+20.1$ $.465n+24.1$ $.512n+11.2$	$.495n-14.0$ $.389n+19.9$ $.481n+23.9$ $.512n+11.2$
9	$.0057n-3.78$ $.237n+21.6$ $.3n+26.1$ $.509n+11.4$	$.126n-6.56$ $.289n+21.0$ $.383n+25.0$ $.51n+11.4$	$.238n-8.34$ $.329n+20.6$ $.411n+24.8$ $.508n+11.4$	$.29n-8.96$ $.351n+20.4$ $.428n+24.7$ $.508n+11.4$	$.352n-9.8$ $.371n+20.2$ $.478n+24.0$ $.508n+11.4$	$.404n-10.4$ $.386n+20.1$ $.478n+24.0$ $.508n+11.4$	$.45n-11.2$ $.396n+20.0$ $.528n+23.3$ $.508n+11.4$	$.49n-12.3$ $.401n+20.0$ $.487n+24.1$ $.51n+11.4$
10	$.0229n-3.66$ $.235n+22.1$ $.308n+26.4$ $.507n+11.5$	$.19n-6.91$ $.296n+21.3$ $.368n+25.6$ $.509n+11.5$	$.308n-8.83$ $.333n+20.8$ $.423n+24.9$ $.509n+11.5$	$.375n-9.76$ $.355n+20.6$ $.44n+24.7$ $.507n+11.5$	$.441n-10.6$ $.378n+20.3$ $.462n+24.4$ $.509n+11.5$	$.494n-11.2$ $.396n+20.1$ $.49n+24.1$ $.51n+11.5$	$.541n-11.9$ $.405n+20.1$ $.506n+23.9$ $.51n+11.5$	$.53n-12.1$ $.413n+20.0$ $.494n+24.2$ $.508n+11.5$
11	$.0822n-3.78$ $.283n+21.6$ $.359n+26.0$ $.51n+11.5$	$.247n-6.51$ $.329n+21.0$ $.389n+25.5$ $.512n+11.5$	$.368n-8.41$ $.357n+20.7$ $.46n+24.6$ $.512n+11.5$	$.432n-9.23$ $.372n+20.5$ $.461n+24.7$ $.512n+11.5$	$.504n-10.2$ $.395n+20.2$ $.46n+24.7$ $.515n+11.4$	$.535n-10.4$ $.408n+20.1$ $.517n+23.9$ $.515n+11.4$	$.578n-11.0$ $.417n+20.1$ $.531n+23.7$ $.515n+11.4$	$.563n-11.7$ $.424n+20.0$ $.514n+24.1$ $.511n+11.5$
12	$.0974n-2.22$ $.281n+22.0$ $.289n+27.4$ $.516n+11.5$	$.252n-4.78$ $.335n+21.2$ $.497n+24.1$ $.522n+11.4$	$.357n-6.44$ $.357n+20.9$ $.497n+24.1$ $.522n+11.4$	$.419n-7.35$ $.389n+20.5$ $.447n+25.1$ $.522n+11.4$	$.508n-8.74$ $.401n+20.3$ $.447n+25.1$ $.522n+11.4$	$.546n-9.14$ $.422n+20.1$ $.656n+21.9$ $.527n+11.3$	$.579n-9.55$ $.429n+20.1$ $.656n+21.9$ $.527n+11.3$	$.592n-11.2$ $.43n+20.1$ $.502n+24.4$ $.514n+11.5$

Table 11: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain only *unsatisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum T-*depth* of the form $2^{a \cdot n+b}$. Each cell contains the exponent $a \cdot n + b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover.

Table 12 caption and grid. Each cell lists four values in order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover.

k \ β	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						$.0369n-11.0$ $.29n + 29.9$ $.341n + 34.7$ $.548n + 15.7$	$.0502n-12.6$ $.3n + 29.8$ $.352n + 34.5$ $.549n + 15.7$	$.0452n-11.4$ $.32n + 29.3$ $.372n + 34.0$ $.53n + 12.6$
4					$.0724n - 9.6$ $.346n + 29.5$ $.407n + 33.9$ $.557n + 17.2$	$.13n - 12.7$ $.373n + 29.2$ $.436n + 33.7$ $.56n + 17.1$	$.165n - 14.4$ $.388n + 29.0$ $.451n + 33.5$ $.561n + 17.1$	$.171n - 14.2$ $.398n + 28.8$ $.458n + 33.3$ $.515n + 14.1$
5				$.0951n-9.27$ $.351n + 29.9$ $.41n + 34.4$ $.558n + 18.6$	$.18n - 12.7$ $.393n + 29.3$ $.458n + 33.8$ $.562n + 18.5$	$.245n - 14.7$ $.41n + 29.3$ $.475n + 33.7$ $.561n + 18.5$	$.29n - 15.9$ $.427n + 29.1$ $.49n + 33.6$ $.563n + 18.5$	$.3n - 15.7$ $.459n + 28.3$ $.534n + 32.5$ $.506n + 15.5$
6			$.081n - 7.4$ $.353n + 30.3$ $.416n + 34.8$ $.56n + 19.9$	$.164n - 10.1$ $.38n + 30.1$ $.445n + 34.4$ $.56n + 19.9$	$.249n - 12.2$ $.424n + 29.5$ $.497n + 33.8$ $.567n + 19.7$	$.327n - 13.8$ $.446n + 29.3$ $.52n + 33.6$ $.568n + 19.7$	$.34n - 13.5$ $.46n + 29.2$ $.532n + 33.5$ $.57n + 19.7$	$.39n - 15.4$ $.509n + 28.0$ $.59n + 32.2$ $.501n + 16.9$
7	$.00334n-4.19$ $.251n + 32.2$ $.308n + 36.9$ $.555n + 21.2$	$.0411n-5.58$ $.329n + 31.3$ $.391n + 35.8$ $.562n + 21.1$	$.163n - 9.37$ $.385n + 30.6$ $.461n + 34.8$ $.568n + 21.0$	$.244n - 11.3$ $.409n + 30.4$ $.479n + 34.7$ $.568n + 21.0$	$.329n - 12.9$ $.444n + 30.0$ $.519n + 34.3$ $.574n + 20.9$	$.396n - 14.1$ $.47n + 29.7$ $.549n + 33.9$ $.577n + 20.8$	$.444n - 14.9$ $.481n + 29.7$ $.562n + 33.8$ $.577n + 20.8$	$.466n - 15.3$ $.493n + 29.5$ $.575n + 33.7$ $.579n + 20.8$
8	$.00234n-3.95$ $.281n + 32.9$ $.35n + 37.2$ $.561n + 22.3$	$.0877n-6.53$ $.337n + 32.3$ $.4n + 36.8$ $.563n + 22.3$	$.212n - 9.42$ $.391n + 31.6$ $.467n + 35.8$ $.569n + 22.2$	$.288n - 10.9$ $.42n + 31.2$ $.494n + 35.5$ $.573n + 22.1$	$.368n - 12.2$ $.447n + 31.0$ $.523n + 35.2$ $.575n + 22.1$	$.433n - 13.3$ $.479n + 30.6$ $.575n + 34.5$ $.582n + 22.0$	$.474n - 13.9$ $.486n + 30.6$ $.573n + 34.7$ $.58n + 22.0$	$.492n - 14.1$ $.503n + 30.4$ $.587n + 34.4$ $.584n + 21.9$
9	$.00513n-3.81$ $.325n + 33.3$ $.404n + 37.4$ $.576n + 23.2$	$.115n - 6.42$ $.387n + 32.6$ $.479n + 36.6$ $.581n + 23.2$	$.225n - 8.22$ $.432n + 32.1$ $.514n + 36.3$ $.588n + 23.1$	$.281n - 8.95$ $.454n + 31.8$ $.531n + 36.2$ $.588n + 23.1$	$.345n - 9.82$ $.473n + 31.7$ $.581n + 35.5$ $.588n + 23.1$	$.401n - 10.6$ $.489n + 31.6$ $.581n + 35.5$ $.588n + 23.1$	$.443n - 11.3$ $.499n + 31.5$ $.631n + 34.8$ $.588n + 23.1$	$.486n - 12.3$ $.499n + 31.5$ $.582n + 35.7$ $.584n + 23.1$
10	$.0196n - 3.67$ $.31n + 35.0$ $.373n + 39.6$ $.568n + 24.6$	$.182n - 6.98$ $.375n + 34.1$ $.446n + 38.5$ $.573n + 24.5$	$.299n - 8.9$ $.418n + 33.6$ $.506n + 37.7$ $.579n + 24.4$	$.366n - 9.81$ $.441n + 33.3$ $.525n + 37.5$ $.58n + 24.4$	$.439n - 10.7$ $.465n + 33.1$ $.549n + 37.2$ $.582n + 24.3$	$.499n - 11.5$ $.484n + 32.9$ $.578n + 36.9$ $.585n + 24.3$	$.543n - 12.2$ $.494n + 32.8$ $.595n + 36.6$ $.585n + 24.3$	$.529n - 12.2$ $.502n + 32.7$ $.584n + 36.9$ $.585n + 24.3$
11	$.0804n - 4.04$ $.363n + 35.5$ $.447n + 39.7$ $.582n + 25.5$	$.245n - 6.81$ $.409n + 34.9$ $.471n + 39.4$ $.584n + 25.4$	$.366n - 8.63$ $.44n + 34.5$ $.561n + 38.1$ $.584n + 25.4$	$.432n - 9.52$ $.45n + 34.5$ $.543n + 38.5$ $.584n + 25.4$	$.507n - 10.5$ $.476n + 34.1$ $.542n + 38.5$ $.586n + 25.4$	$.543n - 10.8$ $.491n + 34.0$ $.599n + 37.8$ $.586n + 25.4$	$.576n - 11.2$ $.499n + 33.9$ $.614n + 37.6$ $.586n + 25.4$	$.563n - 11.8$ $.508n + 33.9$ $.597n + 37.9$ $.587n + 25.4$
12	$.0888n - 2.24$ $.377n + 36.7$ $.467n + 40.8$ $.584n + 26.6$	$.249n - 5.02$ $.42n + 36.1$ $.574n + 39.1$ $.64n + 25.5$	$.355n - 6.63$ $.433n + 36.0$ $.524n + 40.1$ $.586n + 26.5$	$.417n - 7.53$ $.463n + 35.5$ $.524n + 40.1$ $.586n + 26.5$	$.512n - 9.01$ $.479n + 35.4$ $.524n + 40.1$ $.586n + 26.5$	$.555n - 9.47$ $.499n + 35.1$ $.732n + 36.9$ $.589n + 26.5$	$.587n - 9.87$ $.507n + 35.1$ $.732n + 36.9$ $.589n + 26.3$	$.593n - 11.3$ $.51n + 35.1$ $.581n + 39.4$ $.586n + 26.5$

Table 12: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain an equal mix of *satisfiable* and *unsatisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum T-*count* of the form $2^{a \cdot n + b}$. Each cell contains the exponent $a \cdot n + b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover.

k \ β	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						.0299n−9.89 .291n + 30.1 .341n + 34.8 .48n + 12.8	.0356n−10.5 .301n + 29.9 .354n + 34.6 .483n + 12.8	.0327n−9.93 .322n + 29.4 .374n + 34.0 .458n + 9.87
4					.054n − 8.35 .348n + 29.5 .411n + 33.9 .524n + 14.3	.0582n−7.84 .374n + 29.2 .435n + 33.7 .521n + 14.4	.0571n−7.32 .389n + 29.0 .453n + 33.5 .528n + 14.3	.0617n−7.25 .397n + 28.8 .458n + 33.3 .472n + 11.5
5				.0713n−7.98 .351n + 29.9 .411n + 34.4 .532n + 15.9	.103n − 8.92 .394n + 29.3 .458n + 33.8 .547n + 15.6	.0892n−7.25 .41n + 29.3 .475n + 33.7 .537n + 15.9	.0885n−6.93 .427n + 29.1 .494n + 33.5 .538n + 15.8	.0587n−5.12 .458n + 28.3 .534n + 32.5 .473n + 13.1
6			.065n − 6.83 .353n + 30.3 .414n + 34.8 .536n + 17.3	.126n − 8.91 .38n + 30.1 .443n + 34.5 .541n + 17.2	.174n − 10.0 .424n + 29.5 .496n + 33.8 .551n + 17.0	.187n − 9.88 .446n + 29.3 .519n + 33.6 .546n + 17.3	.187n − 9.64 .459n + 29.2 .532n + 33.5 .561n + 16.9	.167n − 8.64 .51n + 28.0 .59n + 32.2 .466n + 14.5
7	.00228n−4.15 .253n + 32.2 .311n + 36.8 .506n + 19.0	.0336n−5.35 .328n + 31.3 .392n + 35.8 .541n + 18.4	.138n − 8.77 .386n + 30.6 .463n + 34.8 .545n + 18.4	.205n − 10.5 .409n + 30.4 .479n + 34.7 .542n + 18.6	.256n − 11.4 .444n + 30.0 .519n + 34.3 .561n + 18.2	.264n − 11.1 .471n + 29.7 .55n + 33.9 .559n + 18.3	.295n − 11.7 .481n + 29.7 .562n + 33.8 .539n + 18.7	.255n − 10.2 .493n + 29.5 .575n + 33.7 .568n + 18.3
8	.00223n−3.97 .284n + 32.8 .351n + 37.2 .519n + 20.2	.073n − 6.2 .338n + 32.2 .4n + 36.8 .542n + 19.8	.183n − 8.95 .391n + 31.6 .467n + 35.8 .549n + 19.7	.244n − 10.2 .42n + 31.2 .491n + 35.6 .542n + 19.9	.288n − 10.7 .447n + 31.0 .523n + 35.2 .561n + 19.5	.328n − 11.5 .479n + 30.6 .575n + 34.5 .569n + 19.3	.349n − 11.7 .486n + 30.6 .573n + 34.7 .561n + 19.6	.3n − 10.2 .502n + 30.4 .587n + 34.4 .555n + 19.7
9	.00368n−3.83 .321n + 33.4 .404n + 37.4 .52n + 21.5	.1n − 6.29 .391n + 32.6 .479n + 36.6 .584n + 20.3	.196n − 7.97 .432n + 32.1 .514n + 36.3 .585n + 20.3	.226n − 8.3 .453n + 31.9 .531n + 36.2 .528n + 21.1	.277n − 8.93 .473n + 31.7 .581n + 35.5 .581n + 20.3	.292n − 9.07 .488n + 31.6 .581n + 35.5 .491n + 21.7	.332n − 9.8 .499n + 31.5 .631n + 34.8 .567n + 20.8	.343n − 10.1 .499n + 31.5 .582n + 35.7 .569n + 20.7
10	.016n − 3.73 .313n + 35.0 .38n + 39.4 .545n + 22.2	.172n − 7.24 .377n + 34.1 .446n + 38.5 .557n + 21.9	.281n − 9.14 .418n + 33.6 .506n + 37.7 .57n + 21.7	.339n − 10.0 .442n + 33.3 .525n + 37.5 .59n + 21.4	.389n − 10.6 .465n + 33.1 .549n + 37.2 .538n + 22.3	.439n − 11.4 .484n + 32.9 .578n + 36.9 .525n + 22.4	.466n − 11.8 .494n + 32.8 .595n + 36.6 .585n + 21.5	.412n − 10.8 .502n + 32.7 .584n + 36.9 .567n + 21.9
11	.0838n − 4.77 .359n + 35.6 .447n + 39.7 .581n + 22.7	.243n − 7.56 .403n + 35.0 .471n + 39.4 .607n + 22.2	.359n − 9.47 .438n + 34.6 .561n + 38.1 .556n + 23.1	.414n − 10.1 .448n + 34.5 .543n + 38.5 .567n + 23.1	.485n − 11.2 .475n + 34.1 .542n + 38.5 .652n + 21.4	.508n − 11.3 .492n + 34.0 .599n + 37.8 .685n + 20.9	.534n − 11.7 .499n + 33.9 .614n + 37.6 .56n + 23.1	.464n − 10.9 .508n + 33.9 .597n + 37.9 .564n + 23.1
12	.0991n − 3.37 .393n + 36.4 .467n + 40.8 .733n + 21.2	.26n − 6.21 .425n + 36.0 .574n + 39.1 .568n + 24.2	.372n − 8.12 .43n + 36.0 .524n + 40.1 .543n + 24.5	.447n − 9.39 .455n + 35.7 .524n + 40.1 .733n + 21.3	.538n − 10.8 .479n + 35.4 .524n + 40.1 .492n + 25.4	.58n − 11.4 .498n + 35.1 .732n + 36.9 .568n + 24.2	.556n − 10.8 .507n + 35.1 .732n + 36.9 .568n + 24.2	.539n − 11.4 .51n + 35.1 .581n + 39.4 .552n + 24.4

Table 13: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain only *satisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum T-count of the form $2^{a \cdot n + b}$. Each cell contains the exponent $a \cdot n + b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover.

k \ β	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						$.0412n-11.6$ $.289n+29.8$ $.34n+34.5$ $.549n+16.6$	$.0586n-13.8$ $.3n+29.8$ $.352n+34.5$ $.55n+16.5$	$.0529n-12.5$ $.318n+29.3$ $.377n+33.8$ $.531n+13.5$
4					$.0843n-10.4$ $.345n+29.5$ $.401n+34.0$ $.558n+18.0$	$.146n-13.6$ $.373n+29.2$ $.439n+33.6$ $.561n+18.0$	$.188n-15.7$ $.389n+29.0$ $.453n+33.5$ $.562n+17.9$	$.19n-15.2$ $.399n+28.7$ $.463n+33.1$ $.516n+14.9$
5				$.102n-9.57$ $.351n+29.8$ $.41n+34.4$ $.558n+19.5$	$.192n-13.1$ $.393n+29.3$ $.456n+33.8$ $.563n+19.4$	$.258n-15.1$ $.41n+29.3$ $.475n+33.7$ $.562n+19.4$	$.301n-16.0$ $.427n+29.1$ $.494n+33.5$ $.564n+19.3$	$.318n-16.2$ $.459n+28.3$ $.533n+32.5$ $.507n+16.4$
6			$.0882n-7.62$ $.354n+30.3$ $.418n+34.7$ $.561n+20.7$	$.171n-10.1$ $.38n+30.1$ $.447n+34.4$ $.561n+20.7$	$.258n-12.4$ $.425n+29.5$ $.497n+33.8$ $.568n+20.6$	$.337n-13.9$ $.445n+29.3$ $.52n+33.6$ $.569n+20.6$	$.35n-13.5$ $.461n+29.2$ $.535n+33.5$ $.57n+20.5$	$.407n-15.8$ $.508n+28.0$ $.593n+32.1$ $.503n+17.7$
7	$.00394n-4.21$ $.249n+32.3$ $.304n+36.9$ $.556n+22.1$	$.0473n-5.76$ $.329n+31.3$ $.392n+35.8$ $.563n+21.9$	$.172n-9.55$ $.385n+30.6$ $.458n+34.8$ $.569n+21.8$	$.252n-11.3$ $.41n+30.4$ $.476n+34.8$ $.569n+21.8$	$.334n-12.9$ $.444n+30.0$ $.519n+34.3$ $.575n+21.7$	$.402n-14.1$ $.47n+29.7$ $.55n+33.9$ $.578n+21.7$	$.451n-14.9$ $.481n+29.7$ $.562n+33.8$ $.578n+21.7$	$.476n-15.4$ $.493n+29.5$ $.575n+33.7$ $.58n+21.6$
8	$.00241n-3.93$ $.279n+32.9$ $.342n+37.4$ $.561n+23.2$	$.0994n-6.8$ $.337n+32.3$ $.4n+36.7$ $.563n+23.2$	$.223n-9.54$ $.39n+31.6$ $.468n+35.8$ $.57n+23.0$	$.297n-10.9$ $.42n+31.2$ $.494n+35.5$ $.574n+23.0$	$.374n-12.2$ $.447n+31.0$ $.523n+35.2$ $.576n+22.9$	$.437n-13.2$ $.479n+30.6$ $.566n+34.7$ $.583n+22.8$	$.487n-14.0$ $.486n+30.6$ $.573n+34.7$ $.581n+22.8$	$.495n-14.0$ $.503n+30.4$ $.595n+34.3$ $.586n+22.8$
9	$.0057n-3.78$ $.328n+33.3$ $.391n+37.7$ $.577n+24.1$	$.126n-6.56$ $.385n+32.6$ $.479n+36.6$ $.583n+24.0$	$.238n-8.34$ $.432n+32.1$ $.514n+36.3$ $.589n+23.9$	$.29n-8.96$ $.454n+31.8$ $.531n+36.2$ $.589n+23.9$	$.352n-9.8$ $.473n+31.7$ $.581n+35.5$ $.589n+23.9$	$.404n-10.4$ $.489n+31.6$ $.581n+35.5$ $.589n+23.9$	$.45n-11.2$ $.499n+31.5$ $.631n+34.8$ $.589n+23.9$	$.49n-12.3$ $.499n+31.5$ $.585n+35.6$ $.585n+24.0$
10	$.0229n-3.66$ $.308n+35.0$ $.38n+39.4$ $.568n+25.4$	$.19n-6.91$ $.374n+34.2$ $.446n+38.5$ $.574n+25.3$	$.308n-8.83$ $.416n+33.6$ $.506n+37.7$ $.58n+25.2$	$.375n-9.76$ $.44n+33.3$ $.525n+37.5$ $.582n+25.2$	$.441n-10.6$ $.464n+33.1$ $.549n+37.2$ $.584n+25.2$	$.494n-11.2$ $.484n+32.9$ $.578n+36.9$ $.586n+25.1$	$.541n-11.9$ $.494n+32.8$ $.595n+36.6$ $.586n+25.1$	$.53n-12.1$ $.502n+32.7$ $.584n+36.9$ $.587n+25.1$
11	$.0822n-3.78$ $.364n+35.5$ $.439n+39.9$ $.583n+26.3$	$.247n-6.51$ $.411n+34.9$ $.471n+39.4$ $.586n+26.3$	$.368n-8.41$ $.439n+34.6$ $.541n+38.5$ $.586n+26.3$	$.432n-9.23$ $.453n+34.4$ $.543n+38.5$ $.586n+26.3$	$.504n-10.2$ $.477n+34.1$ $.542n+38.5$ $.588n+26.3$	$.535n-10.4$ $.491n+34.0$ $.599n+37.8$ $.588n+26.3$	$.578n-11.0$ $.499n+33.9$ $.614n+37.6$ $.588n+26.3$	$.563n-11.7$ $.508n+33.9$ $.597n+37.9$ $.588n+26.2$
12	$.0974n-2.22$ $.359n+37.0$ $.367n+42.4$ $.585n+27.4$	$.252n-4.78$ $.412n+36.2$ $.574n+39.1$ $.588n+27.4$	$.357n-6.44$ $.434n+35.9$ $.574n+39.1$ $.588n+27.4$	$.419n-7.35$ $.466n+35.5$ $.524n+40.1$ $.588n+27.4$	$.508n-8.74$ $.477n+35.4$ $.524n+40.1$ $.588n+27.4$	$.546n-9.14$ $.498n+35.1$ $.732n+36.9$ $.59n+27.4$	$.579n-9.55$ $.506n+35.1$ $.732n+36.9$ $.59n+27.4$	$.592n-11.2$ $.51n+35.1$ $.581n+39.4$ $.588n+27.4$

Table 14: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain only *unsatisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum T-count of the form $2^{a \cdot n + b}$. Each cell contains the exponent $a \cdot n + b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover.

β \ k	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						$.0369n-11.0$ $.183n+11.2$ $.234n+15.9$ $.499n+5.15$	$.0502n-12.6$ $.192n+11.1$ $.243n+15.8$ $.499n+5.15$	$.0452n-11.4$ $.2n+10.9$ $.252n+15.6$ $.499n+5.15$
4					$.0724n-9.6$ $.221n+11.1$ $.282n+15.5$ $.5n+5.14$	$.13n-12.7$ $.242n+10.9$ $.304n+15.4$ $.499n+5.14$	$.165n-14.4$ $.255n+10.8$ $.317n+15.3$ $.499n+5.14$	$.171n-14.2$ $.259n+10.8$ $.319n+15.3$ $.499n+5.14$
5				$.0951n-9.27$ $.229n+11.3$ $.289n+15.8$ $.5n+5.14$	$.18n-12.7$ $.262n+11.0$ $.327n+15.4$ $.499n+5.14$	$.245n-14.7$ $.281n+10.8$ $.345n+15.2$ $.499n+5.14$	$.29n-15.9$ $.294n+10.7$ $.356n+15.2$ $.499n+5.14$	$.3n-15.7$ $.306n+10.5$ $.382n+14.8$ $.499n+5.15$
6			$.081n-7.4$ $.234n+11.4$ $.297n+15.9$ $.499n+5.14$	$.164n-10.1$ $.261n+11.2$ $.325n+15.6$ $.499n+5.14$	$.249n-12.2$ $.293n+10.9$ $.366n+15.2$ $.499n+5.14$	$.327n-13.8$ $.313n+10.7$ $.388n+15.0$ $.499n+5.15$	$.34n-13.5$ $.326n+10.6$ $.398n+14.9$ $.499n+5.15$	$.39n-15.4$ $.34n+10.5$ $.422n+14.6$ $.498n+5.16$
7	$.00334n-4.19$ $.152n+12.6$ $.208n+17.2$ $.499n+5.14$	$.0411n-5.58$ $.218n+11.8$ $.28n+16.3$ $.499n+5.14$	$.163n-9.37$ $.267n+11.3$ $.343n+15.5$ $.499n+5.15$	$.244n-11.3$ $.291n+11.1$ $.36n+15.4$ $.499n+5.15$	$.329n-12.9$ $.321n+10.8$ $.396n+15.0$ $.499n+5.15$	$.396n-14.1$ $.342n+10.6$ $.421n+14.8$ $.499n+5.15$	$.444n-14.9$ $.353n+10.5$ $.434n+14.7$ $.499n+5.15$	$.466n-15.3$ $.362n+10.5$ $.444n+14.6$ $.499n+5.15$
8	$.00234n-3.95$ $.188n+12.4$ $.256n+16.8$ $.499n+5.14$	$.0877n-6.53$ $.241n+11.9$ $.304n+16.4$ $.499n+5.14$	$.212n-9.42$ $.289n+11.3$ $.365n+15.6$ $.499n+5.14$	$.288n-10.9$ $.314n+11.0$ $.389n+15.3$ $.499n+5.15$	$.368n-12.2$ $.339n+10.8$ $.415n+15.1$ $.499n+5.15$	$.433n-13.3$ $.361n+10.6$ $.457n+14.5$ $.499n+5.15$	$.474n-13.9$ $.371n+10.5$ $.458n+14.6$ $.499n+5.15$	$.492n-14.1$ $.382n+10.4$ $.466n+14.5$ $.498n+5.16$
9	$.00513n-3.81$ $.229n+12.1$ $.308n+16.2$ $.499n+5.15$	$.115n-6.42$ $.285n+11.5$ $.376n+15.5$ $.499n+5.15$	$.225n-8.22$ $.321n+11.1$ $.403n+15.3$ $.498n+5.16$	$.281n-8.95$ $.343n+10.8$ $.42n+15.1$ $.498n+5.16$	$.345n-9.82$ $.363n+10.7$ $.47n+14.4$ $.498n+5.16$	$.401n-10.6$ $.378n+10.6$ $.47n+14.4$ $.498n+5.16$	$.443n-11.3$ $.388n+10.5$ $.52n+13.7$ $.498n+5.16$	$.486n-12.3$ $.394n+10.5$ $.477n+14.6$ $.498n+5.16$
10	$.0196n-3.67$ $.233n+12.4$ $.296n+17.0$ $.499n+5.14$	$.182n-6.98$ $.292n+11.7$ $.363n+16.0$ $.499n+5.14$	$.299n-8.9$ $.328n+11.2$ $.417n+15.3$ $.499n+5.15$	$.366n-9.81$ $.349n+11.0$ $.433n+15.2$ $.499n+5.15$	$.439n-10.7$ $.37n+10.8$ $.455n+14.9$ $.499n+5.15$	$.499n-11.5$ $.387n+10.6$ $.481n+14.6$ $.498n+5.15$	$.543n-12.2$ $.396n+10.6$ $.498n+14.4$ $.498n+5.15$	$.529n-12.2$ $.405n+10.5$ $.486n+14.6$ $.498n+5.16$
11	$.0804n-4.04$ $.274n+12.1$ $.358n+16.3$ $.499n+5.15$	$.245n-6.81$ $.318n+11.5$ $.38n+16.0$ $.499n+5.15$	$.366n-8.63$ $.348n+11.1$ $.469n+14.7$ $.499n+5.15$	$.432n-9.52$ $.359n+11.1$ $.451n+15.1$ $.499n+5.15$	$.507n-10.5$ $.382n+10.8$ $.448n+15.2$ $.498n+5.15$	$.543n-10.8$ $.397n+10.6$ $.505n+14.4$ $.498n+5.15$	$.576n-11.2$ $.405n+10.6$ $.52n+14.3$ $.498n+5.15$	$.563n-11.8$ $.414n+10.5$ $.503n+14.6$ $.498n+5.16$
12	$.0888n-2.24$ $.289n+12.1$ $.379n+16.3$ $.499n+5.15$	$.249n-5.02$ $.329n+11.6$ $.483n+14.7$ $.544n+4.29$	$.355n-6.63$ $.342n+11.5$ $.433n+15.6$ $.498n+5.15$	$.417n-7.53$ $.372n+11.0$ $.433n+15.6$ $.498n+5.15$	$.512n-9.01$ $.388n+10.9$ $.433n+15.6$ $.498n+5.15$	$.555n-9.47$ $.404n+10.7$ $.638n+12.5$ $.498n+5.16$	$.587n-9.87$ $.412n+10.6$ $.638n+12.5$ $.498n+5.01$	$.593n-11.3$ $.419n+10.6$ $.491n+14.9$ $.498n+5.15$

Table 15: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain an equal mix of *satisfiable* and *unsatisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum *query complexity* of the form $2^{a \cdot n + b}$. Each cell contains the exponent $a \cdot n + b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover.

k \ β	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						$.0299n-9.89$ $.184n+11.3$ $.234n+16.0$ $.43n+2.24$	$.0356n-10.5$ $.193n+11.2$ $.245n+15.8$ $.433n+2.22$	$.0327n-9.93$ $.202n+11.0$ $.254n+15.6$ $.427n+2.4$
4					$.054n-8.35$ $.223n+11.1$ $.286n+15.5$ $.466n+2.32$	$.0582n-7.84$ $.242n+10.9$ $.304n+15.4$ $.46n+2.45$	$.0571n-7.32$ $.255n+10.8$ $.319n+15.3$ $.466n+2.37$	$.0617n-7.25$ $.258n+10.8$ $.319n+15.3$ $.456n+2.59$
5				$.0713n-7.98$ $.229n+11.3$ $.29n+15.8$ $.474n+2.49$	$.103n-8.92$ $.262n+11.0$ $.327n+15.4$ $.484n+2.24$	$.0892n-7.25$ $.28n+10.8$ $.345n+15.2$ $.475n+2.51$	$.0885n-6.93$ $.293n+10.8$ $.36n+15.1$ $.474n+2.51$	$.0587n-5.12$ $.305n+10.6$ $.382n+14.8$ $.467n+2.73$
6			$.065n-6.83$ $.234n+11.5$ $.295n+16.0$ $.475n+2.61$	$.126n-8.91$ $.261n+11.2$ $.324n+15.6$ $.48n+2.49$	$.174n-10.0$ $.293n+10.9$ $.366n+15.2$ $.484n+2.39$	$.187n-9.88$ $.314n+10.7$ $.387n+15.0$ $.477n+2.7$	$.187n-9.64$ $.325n+10.7$ $.398n+14.9$ $.49n+2.35$	$.167n-8.64$ $.341n+10.4$ $.422n+14.6$ $.463n+2.8$
7	$.00228n-4.15$ $.153n+12.6$ $.211n+17.1$ $.45n+2.96$	$.0336n-5.35$ $.218n+11.8$ $.282n+16.3$ $.478n+2.49$	$.138n-8.77$ $.268n+11.3$ $.345n+15.5$ $.476n+2.63$	$.205n-10.5$ $.291n+11.1$ $.36n+15.4$ $.473n+2.76$	$.256n-11.4$ $.321n+10.8$ $.396n+15.0$ $.486n+2.45$	$.264n-11.1$ $.342n+10.6$ $.422n+14.7$ $.481n+2.64$	$.295n-11.7$ $.352n+10.5$ $.434n+14.7$ $.461n+3.03$	$.255n-10.2$ $.362n+10.5$ $.444n+14.6$ $.488n+2.64$
8	$.00223n-3.97$ $.19n+12.4$ $.258n+16.7$ $.457n+3.0$	$.073n-6.2$ $.242n+11.8$ $.304n+16.4$ $.479n+2.66$	$.183n-8.95$ $.289n+11.3$ $.365n+15.6$ $.48n+2.63$	$.244n-10.2$ $.314n+11.0$ $.385n+15.4$ $.468n+2.94$	$.288n-10.7$ $.339n+10.8$ $.415n+15.1$ $.486n+2.59$	$.328n-11.5$ $.361n+10.6$ $.457n+14.5$ $.485n+2.53$	$.349n-11.7$ $.371n+10.5$ $.458n+14.6$ $.48n+2.77$	$.3n-10.2$ $.381n+10.4$ $.466n+14.5$ $.469n+2.97$
9	$.00368n-3.83$ $.225n+12.1$ $.308n+16.2$ $.443n+3.38$	$.1n-6.29$ $.288n+11.4$ $.376n+15.5$ $.501n+2.34$	$.196n-7.97$ $.321n+11.1$ $.403n+15.3$ $.496n+2.42$	$.226n-8.3$ $.342n+10.9$ $.42n+15.1$ $.439n+3.21$	$.277n-8.93$ $.363n+10.7$ $.47n+14.4$ $.491n+2.37$	$.292n-9.07$ $.378n+10.6$ $.47n+14.4$ $.402n+3.82$	$.332n-9.8$ $.388n+10.5$ $.52n+13.7$ $.477n+2.86$	$.343n-10.1$ $.394n+10.5$ $.477n+14.6$ $.484n+2.71$
10	$.016n-3.73$ $.236n+12.4$ $.303n+16.9$ $.477n+2.75$	$.172n-7.24$ $.293n+11.7$ $.363n+16.0$ $.483n+2.54$	$.281n-9.14$ $.329n+11.2$ $.417n+15.3$ $.49n+2.42$	$.339n-10.0$ $.35n+11.0$ $.433n+15.2$ $.509n+2.24$	$.389n-10.6$ $.37n+10.8$ $.455n+14.9$ $.455n+3.17$	$.439n-11.4$ $.387n+10.6$ $.481n+14.6$ $.439n+3.28$	$.466n-11.8$ $.396n+10.6$ $.498n+14.4$ $.499n+2.42$	$.412n-10.8$ $.405n+10.5$ $.486n+14.6$ $.48n+2.81$
11	$.0838n-4.77$ $.27n+12.1$ $.358n+16.3$ $.497n+2.4$	$.243n-7.56$ $.312n+11.6$ $.38n+16.0$ $.522n+1.94$	$.359n-9.47$ $.347n+11.2$ $.469n+14.7$ $.47n+2.78$	$.414n-10.1$ $.356n+11.1$ $.451n+15.1$ $.482n+2.77$	$.485n-11.2$ $.381n+10.8$ $.448n+15.2$ $.564n+1.19$	$.508n-11.3$ $.398n+10.6$ $.505n+14.4$ $.597n+0.632$	$.534n-11.7$ $.405n+10.6$ $.52n+14.3$ $.472n+2.88$	$.464n-10.9$ $.414n+10.5$ $.503n+14.6$ $.476n+2.84$
12	$.0991n-3.37$ $.305n+11.9$ $.379n+16.3$ $.648n-0.28$	$.26n-6.21$ $.334n+11.5$ $.483n+14.7$ $.48n+2.79$	$.372n-8.12$ $.34n+11.5$ $.433n+15.6$ $.455n+3.14$	$.447n-9.39$ $.364n+11.2$ $.433n+15.6$ $.645n-0.0727$	$.538n-10.8$ $.388n+10.9$ $.433n+15.6$ $.404n+3.98$	$.58n-11.4$ $.404n+10.7$ $.638n+12.5$ $.478n+2.83$	$.556n-10.8$ $.413n+10.6$ $.638n+12.5$ $.478n+2.83$	$.539n-11.4$ $.419n+10.6$ $.491n+14.9$ $.464n+2.97$

Table 16: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain only *satisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum *query complexity* of the form $2^{a \cdot n + b}$. Each cell contains the exponent $a \cdot n + b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover.

k \ β	$\frac{1}{2}$	1	$\frac{3}{2}$	2	3	5	10	∞
3						$.0412n-11.6$ $.183n+11.1$ $.233n+15.7$ $.5n+6.01$	$.0586n-13.8$ $.192n+11.0$ $.243n+15.7$ $.5n+6.01$	$.0529n-12.5$ $.198n+10.9$ $.257n+15.4$ $.5n+6.01$
4					$.0843n-10.4$ $.22n+11.1$ $.276n+15.6$ $.5n+6.01$	$.146n-13.6$ $.242n+10.9$ $.308n+15.3$ $.5n+6.01$	$.188n-15.7$ $.255n+10.8$ $.319n+15.2$ $.5n+6.01$	$.19n-15.2$ $.26n+10.7$ $.324n+15.1$ $.5n+6.01$
5				$.102n-9.57$ $.23n+11.2$ $.288n+15.8$ $.5n+6.01$	$.192n-13.1$ $.262n+11.0$ $.324n+15.4$ $.5n+6.01$	$.258n-15.1$ $.281n+10.8$ $.345n+15.2$ $.5n+6.01$	$.301n-16.0$ $.294n+10.7$ $.36n+15.1$ $.5n+6.01$	$.318n-16.2$ $.307n+10.5$ $.38n+14.8$ $.5n+6.01$
6			$.0882n-7.62$ $.235n+11.4$ $.298n+15.9$ $.5n+6.01$	$.171n-10.1$ $.26n+11.2$ $.327n+15.5$ $.5n+6.01$	$.258n-12.4$ $.294n+10.9$ $.366n+15.1$ $.5n+6.01$	$.337n-13.9$ $.313n+10.7$ $.388n+15.0$ $.5n+6.01$	$.35n-13.5$ $.327n+10.6$ $.401n+14.9$ $.5n+6.01$	$.407n-15.8$ $.34n+10.5$ $.424n+14.6$ $.5n+6.01$
7	$.00394n-4.21$ $.149n+12.6$ $.205n+17.2$ $.5n+6.01$	$.0473n-5.76$ $.219n+11.8$ $.281n+16.3$ $.5n+6.01$	$.172n-9.55$ $.267n+11.3$ $.34n+15.5$ $.5n+6.01$	$.252n-11.3$ $.292n+11.1$ $.358n+15.5$ $.5n+6.01$	$.334n-12.9$ $.321n+10.8$ $.396n+15.0$ $.5n+6.01$	$.402n-14.1$ $.341n+10.6$ $.422n+14.8$ $.5n+6.01$	$.451n-14.9$ $.353n+10.5$ $.434n+14.7$ $.5n+6.01$	$.476n-15.4$ $.361n+10.5$ $.444n+14.6$ $.5n+6.01$
8	$.00241n-3.93$ $.186n+12.4$ $.249n+17.0$ $.5n+6.01$	$.0994n-6.8$ $.241n+11.9$ $.304n+16.3$ $.5n+6.01$	$.223n-9.54$ $.289n+11.3$ $.366n+15.5$ $.5n+6.01$	$.297n-10.9$ $.315n+11.0$ $.389n+15.3$ $.5n+6.01$	$.374n-12.2$ $.339n+10.8$ $.415n+15.1$ $.5n+6.01$	$.437n-13.2$ $.361n+10.6$ $.448n+14.7$ $.5n+6.01$	$.487n-14.0$ $.371n+10.5$ $.458n+14.6$ $.5n+6.01$	$.495n-14.0$ $.382n+10.4$ $.474n+14.4$ $.5n+6.01$
9	$.0057n-3.78$ $.232n+12.0$ $.295n+16.5$ $.5n+6.01$	$.126n-6.56$ $.282n+11.5$ $.376n+15.5$ $.5n+6.01$	$.238n-8.34$ $.322n+11.1$ $.403n+15.3$ $.5n+6.01$	$.29n-8.96$ $.343n+10.8$ $.42n+15.1$ $.5n+6.01$	$.352n-9.8$ $.363n+10.7$ $.47n+14.4$ $.5n+6.01$	$.404n-10.4$ $.378n+10.5$ $.47n+14.4$ $.5n+6.01$	$.45n-11.2$ $.389n+10.5$ $.52n+13.7$ $.5n+6.01$	$.49n-12.3$ $.394n+10.5$ $.48n+14.6$ $.5n+6.01$
10	$.0229n-3.66$ $.231n+12.5$ $.303n+16.8$ $.5n+6.01$	$.19n-6.91$ $.29n+11.7$ $.363n+16.0$ $.5n+6.01$	$.308n-8.83$ $.327n+11.3$ $.417n+15.3$ $.5n+6.01$	$.375n-9.76$ $.348n+11.0$ $.433n+15.2$ $.5n+6.01$	$.441n-10.6$ $.37n+10.8$ $.455n+14.9$ $.5n+6.01$	$.494n-11.2$ $.387n+10.6$ $.481n+14.6$ $.5n+6.01$	$.541n-11.9$ $.396n+10.6$ $.498n+14.4$ $.5n+6.01$	$.53n-12.1$ $.405n+10.5$ $.486n+14.6$ $.5n+6.01$
11	$.0822n-3.78$ $.276n+12.0$ $.351n+16.4$ $.5n+6.01$	$.247n-6.51$ $.32n+11.5$ $.38n+16.0$ $.5n+6.01$	$.368n-8.41$ $.347n+11.2$ $.45n+15.1$ $.5n+6.01$	$.432n-9.23$ $.362n+11.0$ $.451n+15.1$ $.5n+6.01$	$.504n-10.2$ $.383n+10.8$ $.448n+15.2$ $.5n+6.01$	$.535n-10.4$ $.397n+10.6$ $.505n+14.4$ $.5n+6.01$	$.578n-11.0$ $.405n+10.6$ $.52n+14.3$ $.5n+6.01$	$.563n-11.7$ $.414n+10.5$ $.503n+14.6$ $.5n+6.01$
12	$.0974n-2.22$ $.271n+12.4$ $.279n+17.9$ $.5n+6.01$	$.252n-4.78$ $.321n+11.7$ $.483n+14.7$ $.5n+6.01$	$.357n-6.44$ $.343n+11.5$ $.483n+14.7$ $.5n+6.01$	$.419n-7.35$ $.375n+11.0$ $.433n+15.6$ $.5n+6.01$	$.508n-8.74$ $.386n+10.9$ $.433n+15.6$ $.5n+6.01$	$.546n-9.14$ $.404n+10.7$ $.638n+12.5$ $.5n+6.01$	$.579n-9.55$ $.411n+10.6$ $.638n+12.5$ $.5n+6.01$	$.592n-11.2$ $.419n+10.6$ $.491n+14.9$ $.5n+6.01$

Table 17: Each grid in these tables represents a distribution of k-SAT instances with differing amounts of structure β, fixed to contain only *unsatisfiable* instances (where lower β means more structure). For each grid we sample SAT instances and fit the classical runtime and quantum *query complexity* of the form $2^{a \cdot n + b}$. Each cell contains the exponent $a \cdot n + b$ in the order: classical runtime, quantum backtracking detection, quantum backtracking search, Grover. We color a cell blue if the classical algorithm scales best (i.e. has the lowest a value), orange if the detection algorithm scales best, yellow if Grover scales best, and green if search scales better than classical and Grover. Query complexity of Grover depends only on n (and the number of solutions, which is constant for unsatisfiable instances), not on k, β and hence is constant.