# Quantum Time-Space Tradeoffs for Exponential Dynamic Programming

**Susanna Caroppo**
Roma Tre University, Department of Engineering, Rome, Italy

**Jevgēnijs Vihrovs**
Faculty of Science and Technology, University of Latvia, Riga, Latvia

**Dārta Zajakina**
Faculty of Science and Technology, University of Latvia, Riga, Latvia

**Aleksejs Zajakins**
Faculty of Science and Technology, University of Latvia, Riga, Latvia

## Abstract

We investigate the quantum algorithms for dynamic programming by Ambainis et al. (SODA'19). While giving provable complexity speedups and applicable to a variety of NP-hard problems, these algorithms have a notable drawback: they require a large amount of Quantum Random Access Memory (QRAM), which potentially could be very challenging to implement in a physical quantum computer. In this work, we study how the space complexity can be improved by trading it for time, while still retaining a speedup over the classical algorithms. We show novel quantum time-space tradeoffs by combining different classical approaches with quantum techniques. For instance, we show that the Travelling Salesman Problem can be solved quantumly in $\widetilde{O}(1.859^n)$ time and $\widetilde{O}(1.315^n)$ QRAM space.

**2012 ACM Subject Classification** Theory of computation → Dynamic programming; Theory of computation → Divide and conquer; Theory of computation → Quantum complexity theory

**Keywords and phrases** Quantum Algorithms, Time-Space Tradeoffs, NP-Hard Problems, Dynamic Programming, Divide & Conquer

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2026.37

**Related Version** *Full Version*: https://arxiv.org/abs/2604.02233

**Funding** *Jevgēnijs Vihrovs*: Supported by the 1.1.1.9 Research application No 1.1.1.9/LZP/2/25/206 of the Activity "Post-doctoral Research" "Practical applications and limitations of quantum search".

## 1 Introduction

In this work, we investigate quantum algorithms designed to address NP-hard problems more efficiently than the best known classical algorithms. The most well-known example of such an algorithm is Grover's search algorithm [13], which can be applied to explore all possible $2^n$ assignments of a SAT formula with $n$ variables. The resulting $\widetilde{O}(\sqrt{2^n})$ time complexity constitutes a quadratic speedup over the respective classical algorithm.

For many other NP-hard problems, however, simply applying Grover's search over all possible options doesn't result in any interesting quantum speedup. Ambainis et al. showed that Grover's search can be combined with the dynamic programming (DP) of Bellman, Held and Karp [5, 15] to obtain quantum speedups for a multitude of NP-hard problems [3]. This, notably, gave an $\widetilde{O}(1.728^n)$ time algorithm for the Travelling Salesman Problem and $\widetilde{O}(1.817^n)$ time algorithms for general graph vertex ordering problems. Subsequently, these ideas have been applied for many other problems, for example, see [7, 9, 18, 20, 23, 24].


Although these algorithms are very useful for obtaining sound theoretical quantum speedups, they typically consume a large amount of Quantum Random Access Memory (QRAM), which is a quantum analogue of RAM. In a nutshell, given $N$ memory data items $D_1, \ldots, D_N$, a QRAM read operation implements the unitary mapping $|i\rangle |0\rangle \mapsto |i\rangle |D_i\rangle$. As a consequence, the advantageous difference from classical RAM is that the memory can be accessed in superposition, which is crucial to the algorithms above.

The complexity of such "quantum dynamic programming" algorithms is measured by the number of gates in the QRAM circuit model, which is the standard circuit model augmented with QRAM gates. The underlying assumption for the quantum speedups above is that an implementation of a QRAM gate is "cheap" and its running time is comparable to a RAM gate in classical computation. On one hand, there are theoretical proposals for QRAM architectures requiring only $\mathrm{O}(\mathrm{poly}\log N)$ time for a QRAM operation [10]. On the other hand, there is an active debate about whether this assumption is reasonable [16], and the progress on the physical realization of QRAM has been limited [22].

Taking into account the various technical difficulties, it is reasonable to assume that if cheap QRAM is attainable, then the size of the available memory will be small, at least in the near term. Another plausible scenario is that only "semi-cheap" QRAM physically is possible, having a running time of an operation between $\mathrm{O}(\mathrm{poly}\log N)$ and trivial $\mathrm{O}(N)$. In this paper we investigate whether a quantum speedup of the exponential quantum dynamic algorithms is possible in the setting with QRAM of limited size. More specifically, we study the time-space tradeoffs by fixing the amount of available QRAM memory and investigating the best possible running time under this restriction. The problem of computing with limited memory is also relevant classically, and such time-space tradeoffs have been studied for classical exponential-time algorithms [19].

## 1.1 Our Results

The algorithms of [3] are based on the classical dynamic programming over subsets [5, 15]. Conceptually, given a problem with an $n$ element input, these algorithms compute some function $f : 2^{[n]} \to \mathbb{N}$ for all subsets $S \subseteq [n]$. The dynamic programming approach computes the values of $f(S)$ from smaller to larger size of $S$ by using the previously computed values; the answer to the problem is usually given by $f([n])$.

On a high level, all of these quantum algorithms share a common structure: in the first stage, they run a classical dynamic programming to compute the values $f(S)$ for $S$ of small size, $|S| \leq \alpha n$, storing all of them in QRAM memory. The second stage runs a quantum search procedure based on Grover's algorithm and uses the fact that $f(S)$ for small $S$ can be retrieved from memory in only $\mathrm{O}(\mathrm{poly}\, n)$ time (using the cheap-QRAM assumption). The optimal time complexity of such an algorithm is achieved when $\alpha$ is chosen so that the complexities of the two stages are balanced. On the other hand, computing the optimal running time of the algorithm when $\alpha$ is fixed naturally gives a time-space tradeoff. Of course, the actual algorithms have more than just one parameter $\alpha$ and the best running time depends on the choice of their values.

In this paper, we pursue two directions:

1. **Optimization tradeoffs.** Calculate the quantum time-space tradeoffs of the existing algorithms by optimizing the relevant parameters.
2. **Improved tradeoffs.** Find novel quantum time-space tradeoffs that are more efficient than the optimization tradeoffs.

There are two different algorithms in [3] for two classes of tasks: *divide & conquer* problems and *permutation* problems. We examine both of these classes separately.

**Divide & Conquer Problems.** Broadly, a problem of this type satisfies the divide & conquer recurrence

$$f(S) = \min_{\substack{X \subset S \\ |X|=k}} g(f(S \setminus X), f(X)), \tag{1}$$

for any value of $k \in [|S|-1]$, where $g$ is some function that can be computed in $\mathrm{O}(\mathrm{poly}\, n)$ time. The answer then is given by $f([n])$. Examples of such problems include the Travelling Salesman Problem and Feedback Arc Set. For these problems [3] provides an algorithm with time and space complexity $\widetilde{\mathrm{O}}(1.728^n)$.

First, we calculate the optimization tradeoff for this algorithm:

▶ **Theorem 1.** *Suppose that the maximum available amount $\mathcal{S}^n$ of QRAM space is given by $\mathcal{S} = 2^{\mathrm{H}(\alpha)} \leq 1.728$ for some $\alpha \in [0, 1/2]$. Let $k$ be the unique integer such that $\frac{1}{2^{k+1}} \leq \alpha < \frac{1}{2^k}$. Then there is a quantum algorithm for the divide & conquer problems with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} = 2^{\max\left\{1-\frac{2-\mathrm{H}(2^k\alpha)}{2^{k+1}}, \mathrm{H}(\alpha)\right\}}$.*

When $\alpha = 1/2^{k+1}$, time complexity simplifies to $\mathcal{T} = 2^{\max\{1-\alpha, \mathrm{H}(\alpha)\}}$, so the optimization tradeoff is lower bounded by $\mathcal{T} = 2^{1-\mathrm{H}^{-1}(\log_2 \mathcal{S})}$ for all $\mathcal{S}$.

Next, we show an improved quantum time-space tradeoff:

▶ **Theorem 2.** *For any maximum available amount $\mathcal{S}^n$ of QRAM space between $\widetilde{\mathrm{O}}(\mathrm{poly}\, n)$ and $\widetilde{\mathrm{O}}(1.728^n)$, there is a quantum algorithm for the divide & conquer problems with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} \in [2/\mathcal{S}^{0.268}, 2/\mathcal{S}^{0.201}]$.*

The improved tradeoff is based on the well-known classical time-space tradeoff for such problems by Gurevich and Shelah [14] with time complexity $\widetilde{\mathrm{O}}(4^n 2^{-s})$ and space complexity $\mathrm{O}(2^s)$, for any $s = n/2, n/4, n/8, \ldots$ Their technique combines divide & conquer over subsets with dynamic programming for subproblems of size $s$. Our result is achieved by combining this algorithm with the quantum dynamic programming algorithm of [3]. Notably, the $\widetilde{\mathrm{O}}(1.728^n)$ algorithm is already based on the same idea: hence, the improved tradeoff is obtained by utilizing the divide & conquer *twice*!

Figure 1 shows the full graph of the two time-space tradeoffs. Their complexities coincide when $\mathcal{S} \in [1.438, 1.728]$, reason being that they compile to the same algorithm in that range. The explicit formula for the time complexity for the improved tradeoff is derived in Section 3. Interestingly, the improved tradeoff follows a fractal pattern – we call this property *fractalization* and explain it in Lemma 5. Note that for all [3]-style algorithms the time complexity is always at least the space complexity because of the first precomputation stage. Therefore, no tradeoff will go below the $\mathcal{T} = \mathcal{S}$ curve at any point.

We note, however, that Theorem 1 has a significant practical advantage over Theorem 2. The first tradeoff requires only QRAM "read" operations on the fully classical data (qROM model). In contrast, the improved tradeoff requires quantum data and also "write" QRAM operations. Nonetheless it shows that fully writable QRAM with quantum data (qRAM) can provide a computational advantage.

**Permutation Problems.** The second type of problems can be informally described as finding the minimum of some function $f : S_n \to \mathbb{N}$, where $S_n$ is the set of permutations on $n$ elements. These problems reduce to computing values of some function $f(L, M, R) = \min_{\pi \in S_M} \mathrm{cost}(L, \pi, R)$, where $L \cup M \cup R = [n]$ is a partition, cost is some function that depends on the problem and $S_M$ is the set of permutations of the elements of $M$. Formally, we require the condition

$$f(L, M, R) = \min_{\substack{X \cup Y = M \\ |X|=k}} g(f(L, X, Y \cup R), f(L \cup X, Y, R)), \tag{2}$$

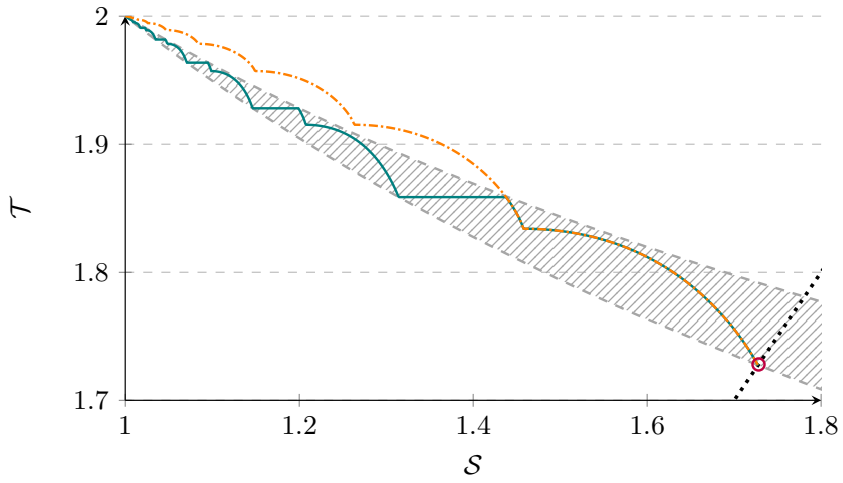



**Figure 1** Time-space tradeoffs for the divide & conquer problems. A point $(\mathcal{S}, \mathcal{T})$ denotes an algorithm with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$ that uses at most $\widetilde{\mathrm{O}}(\mathcal{S}^n)$ QRAM space. Orange line denotes the optimization tradeoff of Theorem 1. Teal line denotes the improved tradeoff of Theorem 2. It is bounded below by $\mathcal{T} = 2/\mathcal{S}^{0.268}$ and above by $\mathcal{T} = 2/\mathcal{S}^{0.201}$ (dashed gray). Dotted line shows the function $\mathcal{T} = \mathcal{S}$, giving a barrier to improving the tradeoffs after $\mathcal{S} \approx 1.728$, which corresponds to the quantum dynamic programming algorithm of [3] (red circle).

for some problem-specific function $g$ computable in $\mathrm{O}(\operatorname{poly} n)$ time. The answer then is given by $f(\varnothing, [n], \varnothing)$. In other words, this condition says that the optimal permutation can be found by decomposing it into subsequences, for which we can find the cost independently and combine them afterwards.

For such problems, Bellman, Held and Karp dynamic programming gives a classical algorithm with $\widetilde{\mathrm{O}}(2^n)$ time and space complexity. This class includes a multitude of graph vertex ordering problems like Treewidth, Pathwidth, Cutwidth, Minimum Fill-In, Optimal Linear Arrangement etc. [6].

As introduced in [3], such problems can be modeled by the Hypercube Path query problem: given a subgraph $G$ of the $n$-dimensional Boolean hypercube accessible by queries to the edges of the hypercube, determine whether a path exists between $0^n$ and $1^n$ in $G$. A quantum algorithm for this problem gives an algorithm with only an $\mathrm{O}(\operatorname{poly} n)$ factor increase in the time and space complexity for any of the permutation problems. The time and space complexity of their quantum algorithm is $\widetilde{\mathrm{O}}(1.817^n)$.

While optimizing the tradeoff of the original divide & conquer quantum algorithm (Theorem 1) is relatively straightforward, it becomes quite challenging in the case of the Hypercube Path. The quantum algorithm of [3] is recursive: it calls itself on subcubes of the original hypercube. If the sole goal is to minimize the time complexity of the algorithm, we need to solve a short system of equations with a handful of parameters, which can be computed numerically without much effort (see Section 3.1 in [3]).

Since we are interested in the time complexity of the algorithm with limited space, the situation is much more difficult. By the construction of the algorithm, each recursive level has a constant number of recursive calls and the depth of the recursion is $\mathrm{O}(\log n)$. Therefore, if the allowed space amount is $\widetilde{\mathrm{O}}(\mathcal{S}^n)$, then each recursive call is still allowed to use $\widetilde{\mathrm{O}}(\mathcal{S}^n / \operatorname{poly}(n)) = \widetilde{\mathrm{O}}(\mathcal{S}^n)$ space (since in total there are $c^{\mathrm{O}(\log n)} = \mathrm{O}(\operatorname{poly} n)$ recursive calls). That means that while the dimension of the subcubes decreases, the total amount of available space essentially does not decrease. Thus, each recursive call must have completely different parameters from its parent call, considerably amplifying the amount of independent parameters and making it unfeasible for numerical optimization.

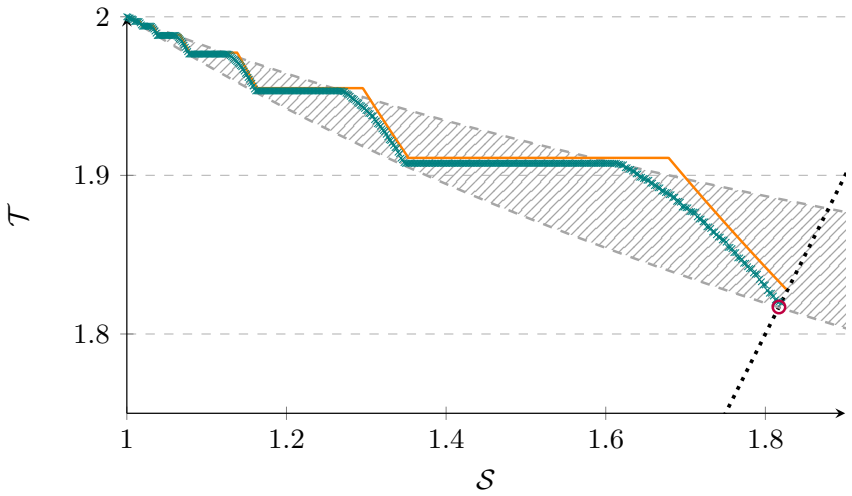



**Figure 2** Time-space tradeoffs for the permutation problems. A point $(\mathcal{S}, \mathcal{T})$ denotes an algorithm with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$ that uses at most $\widetilde{\mathrm{O}}(\mathcal{S}^n)$ QRAM space. The discrete teal plot denotes the optimization tradeoff of Theorem 3. It is bounded below by $\mathcal{T} = 2/\mathcal{S}^{0.161}$ and above by $\mathcal{T} = 2/\mathcal{S}^{0.099}$ (dashed gray). The orange line denotes the quantum pairwise scheme tradeoff of Theorem 4. Dotted line shows the function $\mathcal{T} = \mathcal{S}$, giving a barrier to improving the tradeoffs after $\mathcal{S} \approx 1.817$, which corresponds to the quantum Hypercube Path algorithm of [3] (red circle).

Nonetheless, we find a way to compute this time-space tradeoff in a feasible time by using dynamic programming itself to compute the required time complexities. The approach is described in Section 4, by which we obtain the following result:

▶ **Theorem 3.** *For any maximum available amount $\mathcal{S}^n$ of QRAM space between $\widetilde{\mathrm{O}}(\mathrm{poly}\, n)$ and $\widetilde{\mathrm{O}}(1.817^n)$, there is a quantum algorithm for the permutation problems with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} \in [2/\mathcal{S}^{0.161}, 2/\mathcal{S}^{0.099}]$.*

Then we examine the classical time-space tradeoff called the *pairwise scheme* introduced in [21]. We combine it with a quantum algorithm for dynamic programming in $n$-dimensional lattice graphs from [12] to obtain the following time-space tradeoff:

▶ **Theorem 4.** *For any maximum available amount $\mathcal{S}^n$ of QRAM space in the range $\mathcal{S} \in [1.631, 1.826]$, there is a quantum algorithm for the permutation problems with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} = 2.511/\mathcal{S}^{0.527}$. By fractalizing this algorithm, we obtain a tradeoff for any space amount $\widetilde{\mathrm{O}}(\mathrm{poly}\, n)$ and $\widetilde{\mathrm{O}}(1.826^n)$ with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} \in [2/\mathcal{S}^{0.151}, 2/\mathcal{S}^{0.088}]$.*

The two tradeoffs are depicted in Figure 2, the first one being more optimal. Since it is obtained numerically, the graph is not continuous but given by a discrete set of points. Both tradeoffs follow the fractal-like pattern because of Lemma 5.

While the second tradeoff has a strictly larger time complexity, it has a few benefits: first, it is quite close to the first tradeoff. Secondly, we can write down an explicit formula for its running time and thirdly, the corresponding algorithm is more concise and has a smaller number of parameters. Both of the tradeoffs use QRAM with "write" operations and quantum data (qRAM).

**Our Techniques.** Here we briefly summarize the techniques we use to obtain our tradeoffs.

The divide & conquer tradeoff of Theorem 1 we obtain by optimizing the parameters in the algorithm of [3]. The improved tradeoff of Theorem 2 we obtain by combining the Gurevich and Shelah classical tradeoff [14] with the quantum dynamic programming algorithm for TSP [3]: this tradeoff seems surprising, as the Gurevich and Shelah tradeoff is already being applied once in the original quantum algorithm.

The optimization tradeoff of Theorem 3 is obtained by optimizing the (many) parameters of the Hypercube Path quantum dynamic programming algorithm [3]. This optimization is complicated, and we provide an efficient dynamic programming procedure that allows us to numerically calculate the complexities. The slightly less efficient, but cleaner tradeoff of Theorem 4 we obtain by applying the *quantum dynamic programming on the multidimensional lattices* of [12] to the *pairwise scheme* classical tradeoff [21]. The latter connection looks interesting, as it shows that some tradeoffs (the pairwise scheme in this case) may restrict the problem to searching for a path in a subgraph of the hypercube, where the structure of this subgraph still allows for quantum speedups.

## 1.2 Related Work

Recently, Jeffery and Pass have demonstrated a quantum time-space tradeoff for directed *st*-connectivity [17]. While the Hypercube Path problem is an instance of directed *st*-connectivity, the time complexity of their algorithm in this case is superexponential, trading off with classical space but always using only $O(n^2)$ quantum space. However, a simple application of Grover's search over a path vertex in the middle layer of the hypercube and applying the algorithm recursively already gives an $\widetilde{O}(2^n)$ quantum time algorithm with only $O(\operatorname{poly} n)$ quantum space complexity (which corresponds to $\mathcal{S} = 1$ and $\mathcal{T} = 2$ in Figure 2).

While our work to the best of our knowledge is the first that examines the quantum time-space tradeoffs for the exponential dynamic programming, we've learned that the long-standing classical tradeoffs of Koivisto and Parviainen [19] were recently improved, independently and concurrently by [4] and [8].

# 2 Preliminaries

Our algorithms work in the standard quantum circuit model, augmented with QRAM gates (Quantum Random Access Memory). We denote by qROM the QRAM memory that stores classical data and allows to read it in superposition; by qRAM we denote the QRAM memory that stores quantum data and allows for both read and write operations. The detailed notation and lemmata are given in Appendix A.

We also need the following simple useful general property of time-space tradeoffs:

▶ **Lemma 5** (Fractalization). *For any divide & conquer or permutation problem, a quantum time-space tradeoff $(\mathcal{T}, \mathcal{S})$ implies a quantum $(\sqrt{2\mathcal{T}}, \sqrt{\mathcal{S}})$ tradeoff. All tradeoffs obtained by repeated fractalization from $(\mathcal{T}, \mathcal{S})$ adhere to a function $\mathcal{T} = 2/\mathcal{S}^c$ for some $c$.*

The proof is deferred to Appendix A. This lemma explains why the tradeoffs in Figure 1 and Figure 2 follow a fractal-like pattern. Moreover, this shows why the coinciding part of the two tradeoffs of Theorem 1 and Theorem 2 (with $\mathcal{S} \in [1.438, 1.728]$) repeats for smaller values of $\mathcal{S}$, with horizontal plateaus in between, see Figure 1: this happens because the improved tradeoff can be obtained by fractalizing this segment.

# 3 Divide & Conquer Tradeoffs

## 3.1 Quantum Dynamic Programming

Our tradeoffs are based on the original algorithm of [3] for Travelling Salesman Problem, which we retrace here. Note that for the Travelling Salesman Problem the recurrences below include additional parameters, but the recurrences are essentially the same and the overall complexities stay identical, up to $O(\operatorname{poly} n)$ factors, so we omit these details for the sake of simplicity.

**Algorithm Description.** The algorithm performs in two parts, the classical precomputation and the quantum search stages. For the precomputation, we use Equation (1) condition with $k = 1$ to get the recurrent relation $f(S) = \min_{x \in S} g(f(S \setminus \{x\}), f(\{x\}))$. Assuming $f(\{x\})$ is trivially computable for all $x \in [n]$, the precomputation computes the value of all $f(S)$ in order of increasing size of $S$ until $|S| \leq \alpha n$, using dynamic programming. All of these values are then stored in qROM.

In the quantum search stage, we use Equation (1) with $k = |S|/2$ to get

$$f(S) = \min_{X \subset S, |X| = |S|/2} g(f(S \setminus X), f(X)).$$

We start our algorithm with $S = [n]$ and we run Grover's search (see Theorem 8) over the sets $X$. For each of these sets, we apply Grover's search recursively using the same recurrence.

The recursion continues until $|S| \leq 2\alpha n$, when we apply Grover's search to Equation (1) with $k = \alpha n$, $f(S) = \min_{X \subset S, |X| = \alpha n} g(f(S \setminus X), f(X))$. Here the values of $f(S \setminus X)$ and $f(X)$ can be obtained from qROM in constant time.

## 3.2 Optimization Tradeoff

Varying the parameter $\alpha$ naturally gives a time-space tradeoff. By analyzing the time and space complexity, we obtain the following result:

▶ **Theorem 1.** *Suppose that the maximum available amount $\mathcal{S}^n$ of QRAM space is given by $\mathcal{S} = 2^{\mathrm{H}(\alpha)} \leq 1.728$ for some $\alpha \in [0, 1/2]$. Let $k$ be the unique integer such that $\frac{1}{2^{k+1}} \leq \alpha < \frac{1}{2^k}$. Then there is a quantum algorithm for the divide & conquer problems with time complexity $\widetilde{O}(\mathcal{T}^n)$, where $\mathcal{T} = 2^{\max\left\{1 - \frac{2 - \mathrm{H}(2^k \alpha)}{2^{k+1}}, \mathrm{H}(\alpha)\right\}}$.*

**Proof.** To compute $f(S)$ for a single $S$ during the precomputation stage, we need to examine all $x \in S$, which are $O(n)$. The total number of sets examined during the precomputation is $\binom{n}{\leq \alpha n} := \sum_{i=0}^{\alpha n} \binom{n}{i}$. Thus the time and space complexity at the first stage are equal to

$$T_P = \widetilde{O}\left(\binom{n}{\leq \alpha n}\right) = \widetilde{O}\left(2^{\mathrm{H}(\alpha) n}\right) \tag{3}$$

by the well-known binary entropy approximation (see Theorem 7).

In the second stage, since $\alpha \geq 1/2^{k+1}$, then $k$ is the depth of the recursion (i.e. for the last level, $|S| = n/2^k$). If $|S| = n/2^i$ for $i < k$, then Grover's algorithm searches over $n/2^{i+1}$ options. By Theorem 7 and Theorem 8, the number of iterations of Grover's search on the $i$-th level then is

$$O\left(\sqrt{\binom{n/2^i}{n/2^{i+1}}}\right) = O\left(\sqrt{2^{\mathrm{H}(1/2) n/2^i}}\right) = O(2^{n/2^{i-1}}). \tag{4}$$

At the last recursive call, the number of options to examine is $\binom{n/2^k}{\alpha n}$. Thus the number of Grover's iterations is

$$\mathrm{O}\left(\sqrt{\binom{n/2^k}{\alpha n}}\right) = \mathrm{O}\left(2^{\mathrm{H}(\alpha 2^k)n/2^k}\right). \tag{5}$$

The overall time complexity of the search part then is equal to

$$T_Q = \widetilde{\mathrm{O}}\left(2^{\frac{n}{2}\left(1+\frac{1}{2}+\ldots+\frac{\mathrm{H}(\alpha 2^k)}{2^k}\right)}\right) = \widetilde{\mathrm{O}}\left(2^{n\left(1-\frac{2-\mathrm{H}(2^k\alpha)}{2^{k+1}}\right)}\right). \tag{6}$$

Note that the $\mathrm{O}(\mathrm{poly}(n))$ factor here comes from two places: first, only a single $\mathrm{O}(n)$ overhead factor from the topmost Grover's search; secondly, a factor of $\mathrm{O}(c^{\log n}) = \mathrm{O}(\mathrm{poly}\, n)$ comes from the $\mathrm{O}(\log n)$ recursive applications of Grover's search.

The total space complexity of the search part is only $\mathrm{O}(\mathrm{poly}\, n)$, thus negligible compared to the precomputation (moreover, it doesn't use QRAM). Altogether, the total space complexity is equal to $\widetilde{\mathrm{O}}(\mathcal{S}^n) = T_P$ and the time complexity of the algorithm is equal to $\widetilde{\mathrm{O}}(\mathcal{T}^n) = T_P + T_Q = \mathrm{O}(\max(T_P, T_Q))$. By balancing the two terms, the smallest time complexity is obtained at $\alpha \approx 0.236$ and achieves $\mathcal{T} = \mathcal{S} \approx 1.728^n$ (this gives the algorithm of [3]).

For greater $\alpha$, we have $\mathcal{S} = \mathcal{T} > 1.728$ which is unfavorable. For other $\alpha$, this gives a time-space tradeoff with

$$\mathcal{S} = 2^{\mathrm{H}(\alpha)}, \quad \mathcal{T} = 2^{\max\left\{1-\frac{2-\mathrm{H}(2^k\alpha)}{2^{k+1}}, \mathrm{H}(\alpha)\right\}}. \tag{7}$$

Finally, it is also clear the the algorithm requires only qROM memory. ◀

## 3.3 Improved Tradeoff

The key idea for the improved tradeoff is to use the Gurevich and Shelah approach [14]. Classically, this algorithm runs the divide & conquer strategy recursively until the size of the set is $s = n/2^k$, at which point the answer is computed using the exponential dynamic programming. The time complexity can be calculated to be $\widetilde{\mathrm{O}}(4^n 2^{-s})$ and the space complexity is $\mathrm{O}(2^s)$, since the memory is reused for the final subproblems. Our quantum tradeoff employs the same strategy, except (a) applying Grover's search in the divide & conquer process and (b) running the quantum algorithm from Theorem 1 for the subproblems at the end of the recursion.

**Algorithm Description.** The algorithm is given two parameters, $\alpha \in [0, 1/2]$ and $\beta \in [0, 1]$. Suppose that the algorithm is given a set $S$ and has to compute $f(S)$:

1. If $|S| > 2\beta n$, then it uses Grover's search over sets of size $k = |S|/2$ by Equation (1) and calls itself recursively.
2. If $\beta n < |S| \leq 2\beta n$, it applies Grover's search over sets of size $k = \beta n$ by Equation (1) and calls itself recursively.
3. If $|S| \leq \beta n$, then the algorithm calculates $f(S)$ using the algorithm of Theorem 1 with parameter $\alpha$.

Note that since the algorithm of Theorem 1 is called inside the application of Grover's search, we now need qRAM instead of qROM to be able to write in the same memory at different quantum branches of the computation.

**Complexity.** By analyzing the asymptotic complexity, we obtain the following estimates:

▶ **Theorem 6.** *Suppose that $k \geq 1$, $m \geq 0$ are integers and $\alpha \in [1/2^{k+1}, 1/2^k]$, $\beta \in [1/2^{m+1}, 1/2^m]$ real numbers. Let $\mathrm{R}(\rho, \ell) := 1 - \frac{2-\mathrm{H}(2^\ell \rho)}{2^{\ell+1}}$. There exists a quantum algorithm for the divide & conquer problems with time and space complexities $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, $\widetilde{\mathrm{O}}(\mathcal{S}^n)$, where*

$$\mathcal{T} = 2^{\mathrm{R}(\beta,m)+\beta \max\{\mathrm{R}(\alpha,k),\mathrm{H}(\alpha)\}}, \quad \mathcal{S} = 2^{\beta\,\mathrm{H}(\alpha)}. \tag{8}$$

**Proof.** The total time complexity of the Item 1 and Item 2 has already been analyzed in Theorem 1, as in the second stage of that algorithm we run Grover's search with divide & conquer until sets of size $\alpha n$. In this case, it is equal to $\widetilde{\mathrm{O}}\left(2^{n\mathrm{R}(\beta,m)}\right)$. The algorithm requires only $\mathrm{O}(\mathrm{poly}(n))$ memory because of Grover's searches.

Item 3 requires $\widetilde{\mathrm{O}}(2^{\beta n \max\{\mathrm{R}(\alpha,k),\mathrm{H}(\alpha)\}})$ time and $\widetilde{\mathrm{O}}(2^{n\beta\,\mathrm{H}(\alpha)})$ space by Theorem 1. By multiplying the obtained complexities, the statement follows. ◀

By numerically analyzing this result for different $\alpha$ and $\beta$, we conclude that competitive tradeoffs are obtained only when $\beta = 1/2^m$ for some $m \geq 0$. In that case

$$\mathrm{R}(\beta, m) = 1 - \frac{2 - \mathrm{H}(2^m \cdot 1/2^m)}{2^{m+1}} = 1 - \beta \tag{9}$$

and the time complexity simplifies to

$$\mathcal{T} = 2^{\max\left\{1-\beta\frac{2-\mathrm{H}(2^k\alpha)}{2^{k+1}}, 1-\beta+\beta\,\mathrm{H}(\alpha)\right\}}. \tag{10}$$

The Pareto-optimal points give the teal curve in Figure 1.

This line can also be obtained by fractalization, which also implies the following result, giving easy-to-compute lower and upper bounds for the tradeoff:

▶ **Theorem 2.** *For any maximum available amount $\mathcal{S}^n$ of QRAM space between $\widetilde{\mathrm{O}}(\mathrm{poly}\, n)$ and $\widetilde{\mathrm{O}}(1.728^n)$, there is a quantum algorithm for the divide & conquer problems with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} \in [2/\mathcal{S}^{0.268}, 2/\mathcal{S}^{0.201}]$.*

**Proof.** By Lemma 5, the original $(1.728, 1.728)$ algorithm implies a $(\sqrt{2 \cdot 1.728}, \sqrt{1.728}) \approx (1.859, 1.315)$ tradeoff, and this one implies $(1.928, 1.147)$ tradeoff, and so on. The described points satisfy some curve $2/\mathcal{S}^c = \mathcal{T}$ by Lemma 5. By solving $2/1.727391^c = 1.727391$, we obtain $c \approx 0.268$.

Similarly we can examine the leftmost point at which the two curves in Figure 1 coincide. We can solve this numerically and find that it happens at $\mathcal{S} \approx 1.438$, with $\mathcal{T} \approx 1.859$. By fractalization, solving $2/1.438^c = 1.859$ gives a curve $2/\mathcal{S}^{0.201}$ on which lie all the tradeoff points that can be obtained from $(1.859, 1.438)$ as described in the beginning of the proof. We can then verify that this gives an upper bound on the improved time-space tradeoff. ◀

## 4 Permutation Problem Tradeoffs

In this section, we investigate time-space tradeoffs for permutation problems through two frameworks. The first is the Hypercube Path problem introduced by [3]. The second is the pairwise scheme of [19].

## 4.1 Hypercube Path

We revisit the Hypercube Path problem along with its original quantum algorithm, analyzing its performance under restricted quantum memory. In this setting, we derive new bounds on the achievable time–space tradeoffs and examine the outcomes of this optimization. Furthermore, in Appendix C, we present a dynamic programming method that computes the optimal running time for any specified space budget.

**Problem Definition.** In the Hypercube Path problem, we are given query access to a subgraph $G$ of the directed $n$-dimensional hypercube $Q_n$. The vertices of $Q_n$ are the bit strings in $\{0,1\}^n$ and a directed edge $x \to y$ exists if $y$ can be obtained from $x$ by flipping a single bit from 0 to 1. The goal is to determine whether there exists a directed path from the vertex $0^n$ to $1^n$. The graph $G$ is given by an oracle that, given two vertices $x$ and $y$, returns whether there is an edge $x \to y$.

**Algorithm Description.** Let $k \in \mathbb{N}$ be a parameter of the algorithm and $0 < \alpha_1 < \cdots < \alpha_k < \alpha_{k+1} = 1/2$ be constants to be defined later. Let $A_i$ denote the set of vertices of $G$ with Hamming weight $\lfloor \alpha_i n \rfloor$. The algorithm proceeds in three main stages. In the first stage, the algorithm checks if there exists an index $i$ such that $\lfloor \alpha_i n \rfloor = \lfloor \alpha_{i+1} n \rfloor$. This means that the dimension of the hypercube is below some small threshold and the solution is computed classically via dynamic programming. In the second stage, for each vertex $x$ with Hamming weight at most $\lfloor \alpha_1 n \rfloor$ the algorithm computes whether it is reachable from $0^n$. It uses dynamic programming and stores this reachability information for all vertices $x \in A_1$ in QRAM. Analogously, it checks reachability from layer with Hamming weight $n - \lfloor \alpha_1 n \rfloor$ to $1^n$.

In the third stage, the algorithm executes Grover's search (see Theorem 8) over all vertices $x$ in the middle layer $A_{k+1}$ of the hypercube. For each candidate $x$, it checks if there exists a path from $0^n$ to $x$ (the existence of a path from $x$ to $1^n$ is checked analogously). The verification proceeds recursively, starting from a target vertex in the highest layer $A_{k+1}$ and decomposing the path step-by-step until it reaches the precomputed base layer $A_1$.

Consider a vertex $x$ in an arbitrary layer $A_i$ and denote by $x \preceq y$ that $x_i \leq y_i$ for all $i \in [n]$. To determine if there is a path from $0^n$ to $x$, the algorithm must find an intermediate vertex $y$ in the preceding layer $A_{i-1}$ such that: $y \preceq x$, there exists a path from $0^n$ to $y$, and there exists a path from $y$ to $x$ in $G$ in the subcube $\{z \in \{0,1\}^n \mid y \preceq z \preceq x\}$.

This search is implemented quantumly using Grover's search over all potential $y \in A_{i-1}$ with $y \preceq x$. The recursion unfolds as follows:

- **Base case ($i = 1$).** For a vertex $x \in A_1$, the answer is directly retrieved from the classically precomputed lookup table.
- **Recursive step ($i > 1$).** For a vertex $x \in A_i$, the algorithm uses Grover's search to find a valid $y \in A_{i-1}$. For each candidate $y$, it makes two recursive calls. The first checks whether $0^n$ is connected to $y$. The second calls the main Hypercube Path algorithm on the subcube between $y$ and $x$.

For the optimal choice of $k = 6$ parameters $\alpha_i$, the overall time and space complexity is therefore $\widetilde{\mathrm{O}}(1.817^n)$.

## 4.2 Optimization Tradeoff

In contrast to the original algorithm, where the parameters $\alpha_i$ can be found by solving a comparatively simple system of equations, our constrained-memory setting requires a far more complex approach. Suppose that our memory budget is given by $\widetilde{\mathrm{O}}(\mathcal{S}^n)$. Since

the maximum depth of the recursion is $O(\log n)$ and the number of recursive calls at each level of recursion is constant, the total number of recursive subproblems solved by the algorithm is $O(\text{poly}\, n)$. The allowed memory budget at each recursive subproblem is then still $\widetilde{O}(\mathcal{S}^n/\,\text{poly}\, n) = \widetilde{O}(\mathcal{S}^n)$.

Therefore, the dimension of the subcubes decreases in the recursion, while the memory budget stays essentially the same. Thus each recursive call operates on a subproblem that may require a distinct set of optimal parameters $\alpha_i$, independent of those in the parent call. This recursive dependency forces us to optimize over several parameters at every level of the recursion, rendering a direct numerical approach computationally intractable. Consequently, solving this generalized time-space tradeoff problem is fundamentally more complex than the parameter optimization for the original algorithm, which assumed unbounded space.

We analyze such a tradeoff for all values of $k = 1, \ldots, 6$, where $k$ denotes the number of layers used to partition the hypercube. For each such $k$ and for any value of $\mathcal{S}^n$, we determine the time complexity $\mathcal{T}^n$ of solving the Hypercube Path problem. To formalize this, we introduce a parameter $c \in [0, 1]$ representing the relative dimension of the hypercube: at the beginning of the computation we have $c = 1$, and at each recursive step the dimension of the hypercube shrinks, decreasing the value of $c$. We denote by $\mathcal{T}(c)^n$ the time complexity of solving the problem on a hypercube of relative dimension $c$, under fixed values of $k$ and $\mathcal{S}$. In particular, $\mathcal{T}(1)^n = \mathcal{T}^n$, so that $\mathcal{T}(c)^n$ will be used when analyzing intermediate subproblems, while $\mathcal{T}^n$ refers to the overall time complexity of the algorithm.

We now formalize the memory constraint and the associated objective function to be minimized. In Appendix C, we detail our methodology for calculating the optimal value $\mathcal{T}^n$, for all $\mathcal{S}^n$ and $k$.

**Single Layer Analysis.** In the case $k = 1$, the hypercube is partitioned into two layers: $A_1$ consisting of vertices of Hamming weight $\lfloor \alpha_1 cn \rfloor$, and $A_2$ consisting of vertices of weight $\lfloor \alpha_2 cn \rfloor$ with $\alpha_2 = \frac{1}{2}$ (see Figure 3a). Let $\lambda_c$ be the parameter determining the maximum allowed size of the DP table computed during preprocessing. This parameter must be chosen to satisfy the condition $\mathcal{S}^{cn} = \binom{cn}{\leq \lambda_c cn}$. Subsequently, the parameter $\alpha_1$ is chosen to satisfy the inequality $\alpha_1 \leq \lambda_c$. The time complexity $\mathcal{T}^n$ is computed as follows.

**Base case ($cn \leq \alpha_1 n$).** The entire dynamic programming table fits into the memory, so the solution can be directly computed via the classical algorithm, giving

$$\mathcal{T}(c)^n = 2^{cn}. \tag{11}$$

**Recursive case ($cn > \alpha_1 n$).** In this case, the available memory $\mathcal{S}^{cn}$ is insufficient for storing the whole dynamic programming table, necessitating a recursive approach. While the algorithm can precompute up to $\mathcal{S}^{cn}$ entries, the optimal strategy may not use the entire memory budget. The optimal time complexity is therefore given by minimizing over all possible choices of $\alpha_1 \in (0, \lambda_c]$ as follows.

$$\mathcal{T}(c)^n = \min_{\alpha_1 \in (0, \lambda_c]} \max \begin{cases} \binom{cn}{\leq \alpha_1 cn}, \\ \sqrt{\binom{cn}{\alpha_2 cn}} \cdot \sqrt{\binom{\alpha_2 cn}{\alpha_1 cn}} \cdot \mathcal{T}\left((\alpha_2 - \alpha_1)c\right)^n. \end{cases} \tag{12}$$

The first term in the max function represents the classical preprocessing time. The second term describes the quantum recursive time, which is a product of three components: (i) $\sqrt{\binom{cn}{\alpha_2 cn}}$ for Grover's search over the middle layer $A_2$; (ii) $\sqrt{\binom{\alpha_2 cn}{\alpha_1 cn}}$ for Grover's search over

the predecessors in layer $A_1$; and (iii) $\mathcal{T}\left((\alpha_2-\alpha_1)c\right)^n$ for the recursive call on a subcube of relative dimension $(\alpha_2-\alpha_1)c$. For a fixed $\alpha_1$, the time complexity is determined by the slower of the two phases: preprocessing or recursion. The overall optimal time $\mathcal{T}(c)^n$ is found by minimizing this value over all possible $\alpha_1$.

The recurrence for the recursive cases can be expressed asymptotically using the binary entropy function to approximate the binomial coefficients, as stated in Theorem 7. This entropy-based formulation will be used for all subsequent cases $k = 2, \ldots, 6$. Moreover, since the time and space complexities grow exponentially with $n$, we analyze their base-2 exponents for simplicity. Defining the time exponent as $\tau(c) = \log_2 \mathcal{T}(c)$, and noting its independence from the absolute input dimension, we omit explicit dependence on $n$ hereafter. The recurrence for $k = 1$ can be rewritten as:

$$\tau(c) = \min_{\alpha_1 \in (0,\lambda_c]} \max \begin{cases} \mathrm{H}(\alpha_1)c, \\ \frac{1}{2}(\mathrm{H}(\alpha_2)c + \mathrm{H}(\frac{\alpha_1}{\alpha_2})\alpha_2 c) + \tau\left((\alpha_2-\alpha_1)c\right). \end{cases} \tag{13}$$

The general case follows similarly, but is more cumbersome, see Appendix B. We calculate the optimal complexities in Appendix C, which results in the graph seen in Figure 2. The following theorem follows using the calculated values.

▶ **Theorem 3.** *For any maximum available amount $\mathcal{S}^n$ of QRAM space between $\widetilde{\mathrm{O}}(\mathrm{poly}\, n)$ and $\widetilde{\mathrm{O}}(1.817^n)$, there is a quantum algorithm for the permutation problems with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} \in [2/\mathcal{S}^{0.161}, 2/\mathcal{S}^{0.099}]$.*

**Proof.** We can see that fractalization is built-in in the Hypercube Path tradeoff in the following way. It conforms to the described approach as for the top recursive level we can take $\alpha_1 = \ldots = \alpha_k = 0$ and $\alpha_{k+1} = 1/2$ (the algorithm as stated, formally requires $\alpha_i$ values to be distinct, but we can take them to be arbitrarily close to each other). Thus if $(\mathcal{T}_k, \mathcal{T}_k)$ is the time-optimal algorithm for a fixed $k$, then all the tradeoffs obtained by fractalization will be present in our analysis; indeed, these correspond to the leftmost points of all horizontal plateaus in Figure 4. By Lemma 5, for each $k$ these points will lie on a curve $2/\mathcal{S}^{c_k} = \mathcal{T}$ for some constant $c_k$. For $k = 6$, we can solve $2/1.816905^{c_k} = 1.816905$ and get $c_k \approx 0.161$. Hence, the time-space tradeoff is lower bounded by $2/\mathcal{S}^{0.161}$. Similarly we can also find that it is upper bounded by $2/\mathcal{S}^{0.099}$ using the data calculated numerically. ◀

## 4.3 Pairwise Scheme

The pairwise scheme, originally introduced by [19] in the classical setting, provides a general framework for solving permutation problems. Here we show how we can improve it quantumly.

**Classical Algorithm.** First we describe how the pairwise scheme algorithm works to solve the Hypercube Path problem classically. A valid path from $0^n$ to $1^n$ can be described by a permutation $\pi = (\pi_1, \ldots, \pi_n)$, meaning that the $i$-th edge connects two bit strings that differ in the $\pi_i$-th coordinate. Let $k \in [0, n/2]$ be an integer parameter and for each pair of integers $\{2i-1, 2i\}$, where $i \in [k]$, fix their relative order in this permutation. Each of the $X = 2^k$ resulting partial orders defines a single subproblem, which can be evaluated via dynamic programming as follows.

For a set $S \subseteq [n]$, let $1_S$ denote an $n$-bit string where $x_i = 1$ iff $i \in S$. Then let $f(S) = 1$ iff $1_S$ is reachable from $0^n$ in $G$ and 0 otherwise. If a set $S \subseteq [n]$ satisfies the current partial order, we can compute $f(S)$ using $f(S) = \vee_{i \in S}(f(S \setminus i) \wedge (1_{S \setminus i}, 1_S) \in G)$. If $Y$ is the total number of sets $S$ satisfying the partial order, then using dynamic programming we can

compute $f(S)$ for all such $S$ in time $\widetilde{\mathrm{O}}(Y)$ time and space (the sets can be listed in complexity $\widetilde{\mathrm{O}}(Y)$ using, for example, breadth-first search).

To calculate $Y$, note that for each pair $\{2i-1, 2i\}$, only three of its subsets conform to the partial order: if the order of two elements $a$ and $b$ is fixed to $a \to b$, then the valid subsets are $\varnothing$, $\{a\}$ and $\{a, b\}$ ($\{b\}$ does not conform to the partial order, as $b$ cannot precede $a$). For all other $n-2k$ elements, they can either appear in $S$ or not without restrictions. Therefore, $Y = 3^k \cdot 2^{n-2k}$.

This yields an overall complexity of $\widetilde{\mathrm{O}}((3/2)^k 2^n)$ time and $\widetilde{\mathrm{O}}((3/4)^k 2^n)$ space, for any $k \in [n/2]$, producing a practical scheme that operates within moderately exponential space $\widetilde{\mathrm{O}}(\mathcal{S}^n)$, where $\sqrt{3} \le \mathcal{S} \le 2$.

**Quantum Algorithm.** We combine the pairwise scheme with a quantum algorithm for dynamic programming in $n$-dimensional lattice graphs from [12] to obtain the following quantum time-space tradeoff:

▶ **Theorem 4.** *For any maximum available amount $\mathcal{S}^n$ of QRAM space in the range $\mathcal{S} \in [1.631, 1.826]$, there is a quantum algorithm for the permutation problems with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} = 2.511/\mathcal{S}^{0.527}$. By fractalizing this algorithm, we obtain a tradeoff for any space amount $\widetilde{\mathrm{O}}(\mathrm{poly}\, n)$ and $\widetilde{\mathrm{O}}(1.826^n)$ with time complexity $\widetilde{\mathrm{O}}(\mathcal{T}^n)$, where $\mathcal{T} \in [2/\mathcal{S}^{0.151}, 2/\mathcal{S}^{0.088}]$.*

**Proof.** We will speed up two parts of the classical algorithm quantumly. First, we can run Grover's search over the subproblems, requiring only $\mathrm{O}(\sqrt{X}) = \mathrm{O}(\sqrt{2^k})$ Grover's iterations.

Secondly, we will reduce the dynamic programming in a single subproblem to detecting a path in a *multi-dimensional lattice*. Indeed, examine a graph on the set of vertices

$$V = \{0, 1, 2\}^k \times \{0, 1\}^{n-2k}, \tag{14}$$

in which the first $k$ coordinates correspond to the $k$ pairs with the fixed relative order and the other $n-2k$ coordinates correspond to elements $[2k+1, n]$. Denote the *weight* of a vertex $x \in V$ by $|x| = \sum_{i=0}^{n-k} x_i$. Let a directed edge go from $x$ to $y$ in this graph iff $|x| + 1 = |y|$.

Since we are interested in $f([n])$, then the classical dynamic programming over subsets can be reduced to detecting a path from $0^{n-k}$ to $2^k 1^{n-2k}$. Indeed, for the first $k$ coordinates, the values 0, 1 and 2 denote the subsets $\varnothing$, $\{a\}$ and $\{a, b\}$. The other coordinates have values 0 and 1 if the corresponding element is contained or not in the set.

The work [12] proposed quantum algorithms for detecting such a path in a hyperlattice. In our case, we have a hyperlattice with $k$ dimensions of size 3 and $n-2k$ dimensions of size 2. For such a graph, their algorithm achieves time and space complexity

$$\widetilde{\mathrm{O}}(2.65907^k \cdot 1.82653^{n-2k}) = \widetilde{\mathrm{O}}(1.82653^n / 1.25465^k) = \widetilde{\mathrm{O}}(\mathcal{S}^n), \tag{15}$$

see `solverD2K5.nb` in [11].

Our algorithm then requires the same amount of space, and its time complexity is

$$\widetilde{\mathrm{O}}(\sqrt{2^k} \cdot 1.82653^n / 1.25465^k) = \widetilde{\mathrm{O}}(1.82653^n \cdot 1.12718^k) = \widetilde{\mathrm{O}}(\mathcal{T}^n). \tag{16}$$

We can calculate that the smallest amount of memory that this tradeoff can be applied to is $1.82653^n / 1.25465^{n/2} \approx 1.631^n$. When expressing $\mathcal{T}$ with $\mathcal{S}$, we obtain that $\mathcal{T} = 2.511/\mathcal{S}^{0.527}$.

Finally, we can extend this tradeoff for all values of $\mathcal{S}$ using Lemma 5. We can calculate that the obtained tradeoff is lower and upper bounded by functions $2/\mathcal{S}^{0.151}$ and $2/\mathcal{S}^{0.088}$. ◀

The graph of this tradeoff is shown in Figure 2. While slightly less efficient than Theorem 3, in this case, the algorithm is conceptually simpler and it also comes with an explicit upper bound on its time and space asymptotic complexity.

## References


1 Jonathan Allcock, Jinge Bao, Aleksandrs Belovs, Troy Lee, and Miklos Santha. On the Quantum Time Complexity of Divide and Conquer. In *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025)*, volume 334 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ICALP.2025.9.

2 Jonathan Allcock, Jinge Bao, Joao F. Doriguello, Alessandro Luongo, and Miklos Santha. Constant-depth circuits for Boolean functions and quantum memory devices using multi-qubit gates. *Quantum*, 8:1530, 2024. doi:10.22331/q-2024-11-20-1530.

3 Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. Quantum Speedups for Exponential-Time Dynamic Programming Algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2019, pages 1783–1793, USA, 2019. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611975482.107.

4 Afrouz Jabal Ameli, Jesper Nederlof, and Shengzhe Wang. Improved Space-Time Tradeoffs for Permutation Problems via Extremal Combinatorics, 2026. doi:10.48550/arXiv.2604.05661.

5 Richard Bellman. Dynamic Programming Treatment of the Travelling Salesman Problem. *J. ACM*, 9(1):61–63, 1962. doi:10.1145/321105.321111.

6 Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. A Note on Exact Algorithms for Vertex Ordering Problems on Graphs. *Theory of Computing Systems*, 50(3):420–432, 2012. doi:10.1007/s00224-011-9312-0.

7 Susanna Caroppo, Giordano Da Lozzo, and Giuseppe Di Battista. Quantum algorithms for one-sided crossing minimization. *Theor. Comput. Sci.*, 1052:115424, 2025. doi:10.1016/J.TCS.2025.115424.

8 Justin Dallant and László Kozma. Improved space-time tradeoff for TSP via extremal set systems, 2026. doi:10.48550/arXiv.2604.05645.

9 Serge Gaspers and Jerry Zirui Li. Quantum Algorithms for Graph Coloring and Other Partitioning, Covering, and Packing Problems. In *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 69:1–69:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ICALP.2024.69.

10 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum Random Access Memory. *Phys. Rev. Lett.*, 100:160501, 2008. doi:10.1103/PhysRevLett.100.160501.

11 Adam Glos, Martins Kokainis, Ryuhei Mori, and Jevgēnijs Vihrovs. The numerical results for the complexity of the quantum algorithm for dynamic programming on $n$-dimensional lattice graph, 2021. doi:10.5281/zenodo.4603688.

12 Adam Glos, Martins Kokainis, Ryuhei Mori, and Jevgēnijs Vihrovs. Quantum Speedups for Dynamic Programming on $n$-Dimensional Lattice Graphs. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.50.

13 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC 1996, pages 212–219, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237866.

14 Yuri Gurevich and Saharon Shelah. Expected Computation Time for Hamiltonian Path problem. *SIAM Journal on Computing*, 16(3):486–502, 1987. doi:10.1137/0216034.

15 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10(1):196–210, 1962. doi:10.1145/800029.808532.

16 Samuel Jaques and Arthur G Rattew. QRAM: A Survey and Critique. *Quantum*, 9:1922, 2025. doi:10.22331/Q-2025-12-02-1922.

17 Stacey Jeffery and Galina Pass. A Quantum Time-Space Tradeoff for Directed *st*-Connectivity, 2025. arXiv:2510.08403.

18 Vladislavs Kļevickis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. Quantum Speedups for Treewidth. In *17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022)*, volume 232 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.TQC.2022.11.

19 Mikko Koivisto and Pekka Parviainen. *A Space–Time Tradeoff for Permutation Problems*, pages 484–492. SODA 2010. Society for Industrial and Applied Mathematics, USA, 2010. doi:10.1137/1.9781611973075.41.

20 Masayuki Miyamoto, Masakazu Iwamura, Koichi Kise, and François Le Gall. Quantum Speedup for the Minimum Steiner Tree Problem. In *Computing and Combinatorics*, pages 234–245. Springer International Publishing, 2020. doi:10.1007/978-3-030-58150-3_19.

21 Pekka Parviainen and Mikko Koivisto. Exact structure discovery in Bayesian networks with less space. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI'09, pages 436–443. AUAI Press, 2009. doi:10.5555/1795114.1795165.

22 Fanhao Shen, Yujie Ji, Debin Xiang, Yanzhe Wang, Ke Wang, Chuanyu Zhang, Aosai Zhang, Yiren Zou, Yu Gao, Zhengyi Cui, Gongyu Liu, Jianan Yang, Yihang Han, Jinfeng Deng, Anbang Wang, Zhihong Zhang, Hekang Li, Qiujiang Guo, Pengfei Zhang, Chao Song, Liqiang Lu, Zhen Wang, and Jianwei Yin. Experimental realization of the bucket-brigade quantum random access memory, 2025. arXiv:2506.16682.

23 Kazuya Shimizu and Ryuhei Mori. Exponential-Time Quantum Algorithms for Graph Coloring Problems. In *LATIN 2020: Theoretical Informatics*, pages 387–398. Springer International Publishing, 2020. doi:10.1007/978-3-030-61792-9_31.

24 Seiichiro Tani. Quantum algorithm for finding the optimal variable ordering for binary decision diagrams. *Theoretical Computer Science*, 1041:115230, 2025. doi:10.1016/j.tcs.2025.115230.


## A Extended Preliminaries

**Notation.** Throughout the paper, $n$ is used to denote the size of the problem. We use the shorthand $\mathrm{O}(\mathrm{poly}\, n)$ to denote a function in $\mathrm{O}(n^c)$ for some constant $c$. We will often use the $\widetilde{\mathrm{O}}(f(n))$ notation to hide $\mathrm{O}(\mathrm{poly}\log(f(n)))$ factors. If an algorithm uses $\widetilde{\mathrm{O}}(\mathcal{T}^n)$ time and $\widetilde{\mathrm{O}}(\mathcal{S}^n)$ space, we say that it is an $(\mathcal{T}, \mathcal{S})$ time-space tradeoff.

**Model.** In QRAM, we assume that we have a memory of size $N = \widetilde{\mathrm{O}}(\mathcal{S}^n)$ data items denoted by $|D_1\rangle, \ldots, |D_N\rangle$. We will assume that each data item is supported on $\mathrm{O}(\mathrm{poly}\, n)$ qubits (or bits, if the data is classical). This is similar to the classical word RAM model (here we have algorithms of exponential complexity; hence, the size of the word is polylogarithmic of that).

A QRAM "read-only" operation implements a unitary mapping

$$\sum_{i=1}^{N} \alpha_i |i\rangle_a |b\rangle_t |D_1, \ldots, D_N\rangle_m \mapsto \sum_{i=1}^{N} \alpha_i |i\rangle_a |b \oplus D_i\rangle_t |D_1, \ldots, D_N\rangle_m , \tag{17}$$

where subscripts $a$, $t$ and $m$ denote *address*, *target* and *memory* registers. On the other hand, a stronger QRAM "read-write" operation implements a unitary mapping

$$\sum_{i=1}^{N} \alpha_i |i\rangle_a |b\rangle_t |D_1, \ldots, D_N\rangle_m \mapsto \sum_{i=1}^{N} \alpha_i |i\rangle_a |D_i\rangle_t |D_1, \ldots, D_{i-1}, b, D_{i+1}, \ldots, D_N\rangle_m . \tag{18}$$

As we can see, there are four distinct QRAM models an algorithm can employ: classical or quantum data, read-only or read-write operations. In this paper we use only two: classical data, read-only operations (*qROM*); quantum data, read-write operations (*qRAM*). The terminology of QRAM is not fully consistent throughout the literature, see e.g. [16, 2] for more definitions and discussion.

Throughout the paper, we assume that QRAM operations take unit time.

**Tools.** We frequently use the following well-known approximation:

▶ **Theorem 7** (Entropy approximation). *For any $k \in [0, 1]$, we have*

$$\binom{n}{\leq k} = \sum_{i=0}^{k} \binom{n}{k} \leq 2^{\mathrm{H}(\frac{k}{n})n}, \tag{19}$$

*where $\mathrm{H}(x) = -(x \log_2 x + (1-x)\log_2(1-x))$ is the binary entropy function.*

The key subroutine of our tradeoffs is Grover's search. We use its updated version:

▶ **Theorem 8** (Quantum minimum finding with erroneous oracle, Corollary 10 in [1]). *Let $\mathcal{A} : [N] \to [\Sigma]$ be a bounded-error quantum algorithm with running time $T$ (correct with constant probability). Then there exists a bounded-error quantum algorithm that computes $\min_{i\in[N]} \mathcal{A}(i)$ in $\mathrm{O}(\sqrt{N}(\log N + T))$ time.*

We briefly mention here the benefits of this new variation. The quantum algorithms of [3] (and consequently, our tradeoffs) call Grover's algorithm recursively, some of them to up to $\mathrm{O}(\log n)$ recursive depth. Since the error probability of inputs is required to be reduced to $\mathrm{O}(1/c^n)$ for each of Grover's search applications, this introduces a factor of $n^{\mathrm{O}(d)}$ in the overall complexity (if $d$ is the depth of the recursion), which can be as large as $n^{\mathrm{O}(\log n)}$. This factor is subexponential but superpolynomial in $n$. Although it doesn't affect the fact that the asymptotic complexity of the algorithms is subexponential, it still can introduce a substantial multiplicative factor.

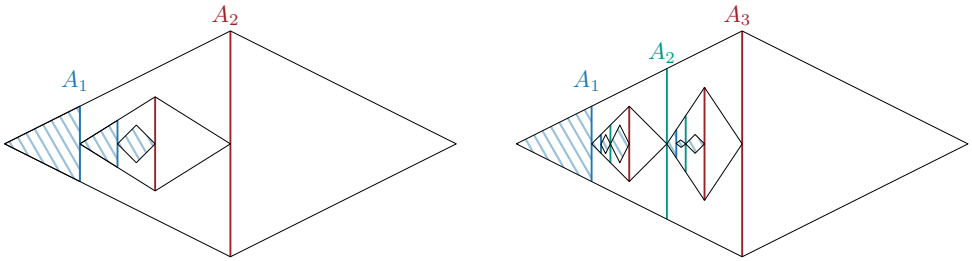
Note that complexity-wise, $\mathrm{O}(c^n f(n)) = \mathrm{O}((c+\varepsilon)^n)$ for any $\varepsilon > 0$ if $f(n)$ is subexponential in $c$. Hence, for example, complexity $\mathrm{O}(1.728^n)$ can be slightly deceptive, as the more accurate asymptotics are described by $n^{\mathrm{O}(\log n)}1.727391...^n$. Theorem 8 gives an improved complexity and a clearer view: since now no error reduction is necessary, only a single additional $\mathrm{O}(n)$ factor comes up in the complexity from Grover's searches. This lets us write our asymptotic complexities as $\widetilde{\mathrm{O}}(c^n)$, which highlights that the hidden factor is only polynomial in $n$.

**Fractalization.** We restate the lemma from the preliminaries:

▶ **Lemma 5** (Fractalization). *For any divide & conquer or permutation problem, a quantum time-space tradeoff $(\mathcal{T}, \mathcal{S})$ implies a quantum $(\sqrt{2\mathcal{T}}, \sqrt{\mathcal{S}})$ tradeoff. All tradeoffs obtained by repeated fractalization from $(\mathcal{T}, \mathcal{S})$ adhere to a function $\mathcal{T} = 2/\mathcal{S}^c$ for some $c$.*

**Proof.** If the problem is of the divide & conquer type, suppose we run a single Grover's search over the sets of size $n/2$ using Equation (1) and then apply the given tradeoff for those sets. In the case of a permutation problem, consider running Grover's search over all possible sets of size $n/2$ as the first half of the permutation, and then applying the given tradeoff to find the optimal permutation of each half. In both cases the time complexity is given by

$$\widetilde{\mathrm{O}}\left(\sqrt{\binom{n}{n/2}}\mathcal{T}^{n/2}\right) = \widetilde{\mathrm{O}}(\sqrt{2\mathcal{T}}^n) \tag{20}$$




**(a)** A 1-layer decomposition of the hypercube. **(b)** A 2-layer decomposition of the hypercube.

**Figure 3** Illustration of the recursive hypercube layering strategy. Figure 3a depicts the case for $k = 1$ layers. The initial preprocessing layer $A_1$ is highlighted in blue. Its size, determined by parameter $\alpha_1$, is constrained by the available memory, represented by the blue striped region. The middle layer $A_2$ is highlighted in red. The diagram shows three levels of recursion. At the final level, the subcube is small enough that its entire solution fits within the memory budget and can be computed classically. Figure 3b illustrates the analogous structure for $k = 2$ layers. The intermediate layer $A_2$ is highlighted in green.

and the space complexity is only $\widetilde{\mathrm{O}}(\mathcal{S}^{n/2}) = \widetilde{\mathrm{O}}(\sqrt{\mathcal{S}}^n)$.

The second part follows since if $2/\mathcal{S}^c = \mathcal{T}$, then $2/\sqrt{\mathcal{S}}^c = \sqrt{2\mathcal{T}}$ is the same identity. ◀

Classically, fractalization from $(\mathcal{T}, \mathcal{S})$ implies a $(2\sqrt{\mathcal{T}}, \sqrt{\mathcal{S}})$ tradeoff. Thus, as a side note, one can fractalize the classical time-space tradeoffs of [19] to achieve the time-space product less than 4 at almost all points $\mathcal{S} \in [1, 2]$ instead of just the segment $\mathcal{S} \in [1.452, 2]$.

## B General Case Analysis

We now generalize our Hypercube Path tradeoff analysis to an arbitrary number of layers $k$. In the $k$-layer configuration, we have layers $A_1, A_2, \ldots, A_{k+1}$, where the layer $A_1$ determines the size of the DP table computed in the preprocessing step, $A_{k+1}$ is the middle layer (with $\alpha_{k+1} = 1/2$), and all others are intermediate layers (see Figure 3b). The time complexity $\mathcal{T}(c)$ is computed as follows.

**Recursive case ($c > \alpha_1$).** The available memory is insufficient for a direct solution, necessitating a recursive approach. The optimal strategy may not use the entire memory budget, and the optimal values for the parameters $\alpha_1, \alpha_2, \ldots, \alpha_k$ are not known a priori. The optimal time complexity is therefore given by minimizing over all valid sequences of parameters $\alpha_1, \ldots, \alpha_k$ satisfying the chain $0 < \alpha_1 < \alpha_2 < \cdots < \alpha_k < \alpha_{k+1}$. For succinctness, we denote this multiple minimization by

$$\min_{\alpha_1,\ldots,\alpha_k} = \min_{\alpha_1 \in (0,\lambda_c]} \min_{\alpha_2 \in (\alpha_1,\alpha_{k+1})} \cdots \min_{\alpha_k \in (\alpha_{k-1},\alpha_{k+1})}.$$

The recurrence relation is given by:

$$\tau(c) = \min_{\alpha_1,\ldots,\alpha_k} \max \begin{cases} \mathrm{H}(\alpha_1)c, \\ \frac{1}{2}\left(\mathrm{H}(\alpha_{k+1})c + \sum_{j=i}^{k} \mathrm{H}\left(\frac{\alpha_j}{\alpha_{j+1}}\right)\alpha_{j+1}c\right) + \tau((\alpha_{i+1} - \alpha_i)c), \quad \text{for all } i \in [1,k]. \end{cases} \tag{21}$$

The terms in the max function generalize the structure from the $k = 1$ case. The first term represents the exponent of the classical preprocessing time. The inner maximization over $i$ selects the most costly recursive path, where each path corresponds to recursing at a different

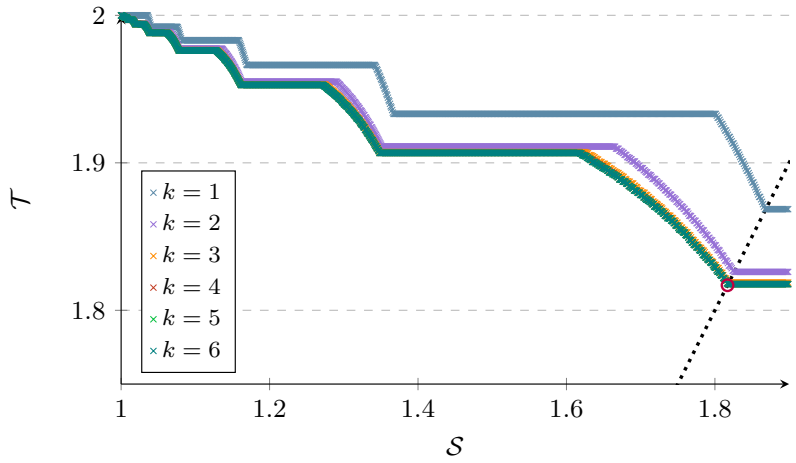



**Figure 4** Quantum time-space tradeoffs for the HYPERCUBE PATH problem. A point at coordinates $(\mathcal{S}, \mathcal{T})$ denotes an algorithm with time complexity $\widetilde{\mathcal{O}}(\mathcal{T}^n)$ and QRAM space complexity $\widetilde{\mathcal{O}}(\mathcal{S}^n)$. Each colored curve corresponds to a different recursive strategy parameterized by $k$, the number of layers split per recursive call. Higher values of $k$ yield better tradeoffs. Dotted line shows the function $\mathcal{T} = \mathcal{S}$, giving a barrier to improving the tradeoffs after $\mathcal{S} \approx 1.817$, which corresponds to the quantum HYPERCUBE PATH dynamic programming algorithm of [3] (red circle).

layer $i$. For a fixed path $i$, the terms represent: Grover's search over the middle layer $A_{k+1}$; the sum of Grover's search exponents over the predecessors in each subsequent layer down to $A_i$; and the recursive call on the subcube between layers $A_i$ and $A_{i+1}$.

Our analysis of the time and space complexity of the HYPERCUBE PATH algorithm for $k = 1, \ldots, 6$ yields a family of tradeoff curves $\mathcal{T}$, parameterized by the memory usage $\mathcal{S}$. These curves quantify the optimal achievable time complexity under a constrained quantum memory budget. The results of this computation are illustrated in Figure 4.

Consistent with the original algorithm, the curves for $k \geq 6$ tend to converge, as further increasing the number of layers does not yield a significant improvement in time complexity. Table 1 presents the resulting optimal time complexities for a representative sample of $\mathcal{S}$ values across all $k = 1, \ldots, 6$. Note that there is a small discrepancy between the constant 1.816905... obtained in [3] and our calculations in which we obtain 1.817776...: this is due to the accumulation of small imprecisions in our calculations.

**Table 1** Representative optimal time complexities illustrating the time-space tradeoff for selected values of the memory parameter $\mathcal{S}$ across different numbers of layers $k = 1, \ldots, 6$. The last line shows the smallest time complexity for the quantum HYPERCUBE PATH algorithm with $k$ layers.

| | $k = 1$ | $k = 2$ | $k = 3$ | $k = 4$ | $k = 5$ | $k = 6$ |
|---|---|---|---|---|---|---|
| $\mathcal{S} = 1.0$ | 2 | 2 | 2 | 2 | 2 | 2 |
| $\mathcal{S} = 1.2$ | 1.966319 | 1.955016 | 1.953075 | 1.952799 | 1.952799 | 1.952799 |
| $\mathcal{S} = 1.4$ | 1.933180 | 1.911044 | 1.907250 | 1.906712 | 1.906712 | 1.906712 |
| $\mathcal{S} = 1.6$ | 1.933180 | 1.911044 | 1.907250 | 1.906712 | 1.906712 | 1.906712 |
| $\mathcal{S} = 1.8$ | 1.931984 | 1.843999 | 1.830741 | 1.828918 | 1.828918 | 1.828918 |
| $\mathcal{S} = \mathcal{T}$ | 1.868583 | 1.826044 | 1.818802 | 1.817776 | 1.817776 | 1.817776 |

## C Computing the Optimal Values

A central step in our analysis is the computation of $\mathcal{T}$ for all $\mathcal{S} \in [1, 2]$ and $k = 1, \ldots, 6$. The parameter $\mathcal{S}$ takes values in a continuous interval, necessitating a discrete approximation with a finite granularity. A naive approach that employs $k$ nested loops over the discrete values of $\mathcal{S}$ is computationally infeasible, as the precision of the results depends critically on a high-resolution discretization (requiring a granularity of at least 1000 points for a good approximation). This would impose a prohibitive computational burden even on modern parallel architectures. To overcome this limitation, we employ a dynamic programming strategy that stores intermediate results to avoid redundant calculations.

This approach revolves around two interacting DP tables: the *optimal complexity* table $\mathtt{T}$ and the *partial complexity* table $\mathtt{P}$. Since both the time complexity $\mathcal{T}^n$ and space complexity $\mathcal{S}^n$ are exponential in $n$, we simplify our analysis by focusing on their exponents. The entries in our dynamic programming tables $\mathtt{T}$ and $\mathtt{P}$ therefore store the base-2 logarithm of the complexity; that is, an entry contains the value $\tau(c) = \log_2 \mathcal{T}(c)$. Specifically, an entry $\mathtt{T}[r, s]$ contains the optimal time complexity for determining whether a path exists in the hypercube using a recursion depth of at most $r$ and a memory budget parameterized by $s$, where $\mathcal{S}^n = 2^{sn}$. Meanwhile, an entry $\mathtt{P}[r, s, \alpha_1, i, \alpha_i]$ contains the partial complexity for determining the existence of a path from $0^n$ to a vertex in layer $A_i$ (with parameter $\alpha_i$), a preprocessing layer $A_1$ with parameter $\alpha_1$, a recursion depth $r$, and a memory parameter $s$. The values of $\mathtt{P}$ are defined only for $r \geq 1$ and $i \geq 2$; the case $i = 1$ is trivial ($\mathtt{P}[\cdot, \cdot, \cdot, 1, \cdot] = 0$) as the reachability of vertices in $A_1$ is known from the precomputation step.

We now formalize the base and recursive cases for filling these tables. For a fixed number of layer $k$, we distinguish the $(k + 1)$-th layer by setting $\alpha_{k+1} = \frac{1}{2}$, as this layer always corresponds to the middle of the hypercube where the final Grover's search is performed.

**Base case $\mathtt{T}$ ($r = 0$).** In the base case with no recursion that uses layers $A_i$, for all values of $s$, the algorithm resorts to a direct quantum search, yielding:

$$\mathtt{T}[0, s] = 1. \tag{22}$$

This corresponds to the exponent of the $\widetilde{\mathrm{O}}(2^n)$-time, $\widetilde{\mathrm{O}}(1)$-space quantum algorithm.

**Recursive case $\mathtt{T}$ ($r > 0$).** The optimal time complexity is obtained by minimizing over all valid choices of the first layer parameter $\alpha_1$, constrained by the available memory $s = \mathrm{H}(\lambda_c)$.

$$\mathtt{T}[r, s] = \min_{\alpha_1 \in (0, \lambda_c]} \left\{ \max \left\{ \mathrm{H}(\alpha_1), \frac{1}{2} + \mathtt{P}[r, s, \alpha_1, k + 1, 1/2] \right\} \right\}. \tag{23}$$

This recurrence mirrors the structure of Equation (21), balancing the cost of classical preprocessing against the quantum recursive cost of verifying paths from $A_1$ to the middle layer.

**Base case $\mathtt{P}$ ($i = 2$).** The complexity of checking if there is a path to layer $A_2$ is computed as follows.

$$\mathtt{P}[r, s, \alpha_1, 2, \alpha_2] = \frac{1}{2} \mathrm{H}\left(\frac{\alpha_1}{\alpha_2}\right) + (\alpha_2 - \alpha_1)\mathtt{T}\left[r - 1, \min\left\{\frac{s}{\alpha_2 - \alpha_1}, 1\right\}\right]. \tag{24}$$

The first term represents the exponent for Grover's search over layer $A_1$ to find a predecessor of a vertex in $A_2$. The second term is the recursive cost of verifying the path within the subcube of relative dimension $(\alpha_2 - \alpha_1)$ between the two layers.

**Recursive case** $\mathtt{P}$ **(**$i \geq 3$**).** For general $i$, the cost of reaching layer $A_i$ from $A_1$ is computed by minimizing over all possible intermediate parameters $\alpha_{i-1} \in (\alpha_1, \alpha_i)$ of the preceding layer.

$$\mathtt{P}[r, s, \alpha_1, i, \alpha_i] = \min_{\alpha_{i-1} \in (\alpha_1, \alpha_i)} \left\{ \frac{1}{2} \mathrm{H}\left(\frac{\alpha_{i-1}}{\alpha_i}\right) + \max\left\{ \mathtt{P}[r, s, \alpha_1, i-1, \alpha_{i-1}],\ (\alpha_i - \alpha_{i-1})\mathtt{T}\left[ r-1, \min\left\{ \frac{s}{\alpha_i - \alpha_{i-1}}, 1 \right\} \right] \right\} \right\}. \tag{25}$$

The first term in the sum is the cost of Grover's search for a predecessor in $A_{i-1}$. The max operator ensures the total cost is dominated by the slower of two processes: the accumulated cost of reaching layer $A_{i-1}$ itself and the recursive cost of exploring the subcube between $A_{i-1}$ and $A_i$.

In summary, this recursive formulation of $\mathtt{T}$ and $\mathtt{P}$ provides a computationally tractable framework for exploring the time-space tradeoff. It avoids the infeasibility of a brute-force discretization over $s$ while enabling precise exploration of the time–memory tradeoff in the recursive quantum algorithm. The tradeoffs calculated by our approach are shown in Figure 4.