A Compressive Sensing Inspired Monte-Carlo Method for Combinatorial Optimization

Baptiste Chevalier[1],* Shimpei Yamaguchi[1],† Wojciech Roga[1],‡ and Masahiro Takeoka[1,2§]
[1] *Department of Electronics and Electrical Engineering, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama 223-8522, Japan and*
[2] *Advanced ICT Research Institute, National Institute of Information and Communications Technology (NICT), Koganei, Tokyo 184-8795, Japan*

In this paper, we present the Monte-Carlo Compressive Optimization algorithm, a new method to solve a combinatorial optimization problem that is assumed compressible. The method relies on random queries to the objective function in order to estimate generalized moments. Next, a greedy algorithm from compressive sensing is repurposed to find the global optimum when not overfitting to samples. We provide numerical results giving evidences that our methods overcome state-of-the-art dual annealing. Moreover, we also give theoretical justification of the algorithm success and analyze its properties. The practicality of our algorithm is enhanced by the ability to tune heuristic parameters to available computational resources.

I. INTRODUCTION

Optimization problems stay in the very core of many important modern world applications including engineering, finance, or physics. However, finding efficient ways to solve these problems is an important concern, since computational time usually increases exponentially with the size of the instance. For instance, the minimization of the cost function in neural networks is done over the space of network parameters and finding a global extremum becomes harder as the network depth increases. However, if the cost functions are continuous and differentiable, or even convex, there are several powerful methods such as gradient-based methods including Stochastic Gradient Descent [1] or gradient-free methods including COBYLA or Nelder-Meads [2, 3]. On the other hand, combinatorial optimization, which we focus on in this paper, turns out to be much more difficult task. Studies from the last decades have identified extremely challenging tasks under the form of cost function that can be expressed as the minimization of a generalized Ising Hamiltonian [4–7]. Indeed, Ising problems have a discrete structure and when the interaction between non-nearest neighbor spins contribute to the energy, dynamic programming methods [8–10] stop working (spin-glass or random energy problems). The optimization becomes extremely challenging. The best methods known today rely on meta-heuristics, as in the case of simulated annealing [5, 11] or genetic algorithms. The cost function in which this complicated structures appears are not limited to physical scenarios but are, in fact, present in many NP problems including NP-complete problems as the knapsack problem or the traveling salesman problem.

In this work, we propose a novel optimization algorithm which can be used to handle combinatorial optimization tasks of high complexity as long as the problem is compressible. Loosely speaking, we say a problem is compressible if there exists a short description which determines all values of the function which needs to be optimized. This implies that an embedding can be found from the compressed description space to the optimization space which maps all the relevant information into the values of the cost function. The optimization method proposed in this paper relies on the principles of compressive sensing [12–15] and statistical compressive learning [16] as well as on Monte-Carlo methods [17, 18].

Compressive sensing is a well established field which allows for recovering a high dimensional signal from small number of linear measurements given that one knows a basis in which the signal has a sparse representation. The recovery procedure finds the sparse representation explicitly, from which the high dimensional signal if derived by the known basis transform. The number of measurements needed to recover the total signal scales as a polynomial function of the number of the non-zero entries of the sparse representation which is much smaller than the dimensionality of the signal. Compressive learning is a relatively new idea. There, a sample from a distribution is mixed by a sketching function and a recovery algorithm which is applied to the sketching image returns a distribution which matches the original distribution better than the estimates based on the sample directly. Monte Carlo is a known approach which uses stochastic algorithms to explore properties of high dimensional functions intractable by rigorous analytical approaches. The combinatorial optimization method we propose here—Monte Carlo Compressive Optimization combines appealing features of the three fields in a novel way. It is applicable if we can assume the existence of the sparse representation of a function, although this representation does not need to be known explicitly. It learns the properties of the maximum value of the function from sketches applied to random samples from the function, although the maximum may never be explored during the sampling process. The approach can be useful for a wide range of applications including reinforcement learning and graph problem optimization. A numerical analysis shows the per-

* chevalier.baptiste@keio.jp
† shimpeiyamaguchi@keio.jp
‡ wojciech.roga@keio.jp
§ takeoka@elec.keio.ac.jp

formance of our method compared to the general state-of-the-art method applicable to this kind of problems: dual annealing (simulated + fast annealing) [19–21].

The paper is structured as follows. In Section II, we will set the framework and define a complicated compressible combinatorial problem determined by a few rules. In Section III, we define and discuss our algorithm, the Monte-Carlo Compressive Optimization algorithm. The Section IV shows the performance of our algorithm obtained through numerical simulation and compare it to the state-of-the-art optimization method: dual annealing. Section V is dedicated to a theoretical justification of the algorithm. The Section VI discusses additional properties of the algorithm such as stability of the algorithm and uniqueness of the solution. Finally, in the last section we discuss applications of our method in Reinforcement Learning and finding the ground states of complex Hamiltonians; the role quantum computers can play in the optimization will be discussed as well.

II. OPTIMIZATION AND COMPRESSIBLE COMBINATORIAL PROBLEMS

An optimization problem consists of finding the optimum (minimum or maximum) of a given real-valued function f, namely the objective, cost, loss or reward function. In mathematical terms, this is to find $x^* = \text{argmin}_x f(x)$ (resp. argmax). To evaluate the performance of an optimization method, one can consider the query complexity framework where calls to the cost function are made through an oracle at constant cost $O(1)$. For combinatorial optimization, the solution space is the space of all possible bit-strings of a given length: $\{0,1\}^N$. This explains why naive methods—brute-force, linear search—have a cost of $O(2^N)$. Optimization is generally difficult but in some cases one can exploit the structure of f to overcome the difficulty as it is the case of gradient-methods for continuous/convex functions. For discrete binary cost functions, a similar desirable feature is the recursivity as it allows one to use dynamic programming methods and changes time complexity into space complexity.

Consider a compressible objective function $f_{\mathcal{R}} : x \mapsto f_{\mathcal{R}}(x)$, where $x \in \{0,1\}^N$ and $f_{\mathcal{R}}$ is uniquely determined by a few rules $\mathcal{R}$. A rule (r, ω_r) can be defined as a bit-string $r \in \mathcal{R}$ of a given length $k \leq N$ and an associated reward $\omega_r \in \mathbb{R}$. Only if the input bit-string x contained the bit string r as a sub-string, then the total cost $f_{\mathcal{R}}(x)$ increases by ω_r.

For a given input x, the function is defined as:

$$f_{\mathcal{R}}(x) = \sum_{r\in\mathcal{R}} \sum_{i=1}^{N-k+1} \omega_r \delta_{h(x_{i\to i+k-1},r),0} \tag{1}$$

where $x_{i\to i+k}$ is a k-length sub-string of the bit-string x starting from the i-th bit, δ is the Kronecker delta and h is the Hamming distance. From now let us use the notation $\delta_r(x_{i\to i+k-1}) := \delta_{h(x_{i\to i+k-1},r),0}$

The function is compressible in the way that all the information is contained in a few values and there exists a (non-linear) mapping $F_N : \mathcal{R} \to \mathbb{R}^{2^N}$ embedding the rules into the optimization space. This mapping is given by the following transform:

$$F_N(r) = \sum_{x\in\{0,1\}^N} \sum_{i=1}^{N-k+1} \delta_r(x_{i\to i+k-1})\mathbf{x} \tag{2}$$

where $\mathbf{x}$ is a vector of which the x-th element is 1 and the rest is 0. The transform F_N maps a rule r to a 2^N real values vector that contains the value taken by our function $f_{\mathcal{R}}$ on $\{0,1\}^N$. We can define the vector:

$$f_{\mathcal{R}} = \sum_{r\in\mathcal{R}} \omega_r F_N(r), \tag{3}$$

as a linear combination of the transform of all rules in $\mathcal{R}$ and thus redefine the function $f_{\mathcal{R}}(x)$ as the x-entry of the vector $f_{\mathcal{R}}$.

To show the difficulty in optimizing such functions, one can rewrite $f_{\mathcal{R}}$ as a diagonal matrix, it reads:

$$\begin{aligned}\mathcal{H} &= \text{diag}(f_{\mathcal{R}}) \\ &= \sum_{r\in\mathcal{R}} \omega_r \text{diag}(F_N(r)) \\ &= \sum_{x\in\{0,1\}^N} \sum_{r\in\mathcal{R}} \sum_{i=1}^{n-k+1} \omega_r \delta_r(x_{i\to i+k-1}) \text{diag}(\mathbf{x})\end{aligned}$$

Physicist may not have failed noticing the similarities between the diagonal form of our cost function described above and the mathematical description of an Ising problem including non-linear coupling terms, i.e. the value of each ω_r depends on the spin configuration $\omega_r(x)$. Using the conventional mapping from QUBO (or general Binary problem) to Ising Hamiltonian for the non-zero contributions, we have:

$$\mathcal{H} = \sum_{r\in\mathcal{R}} \sum_{i=1}^{N-k+1} \omega_r \delta_r(x_{i\to i+k-1}) \bigotimes_{j=i}^{i+k} \Big(\frac{\mathbb{I} + (-1)^{x_j}\sigma_j^z}{2}\Big) \tag{4}$$

where x_i is the i-th bit of x, the tensor term implicitly contains tensor identities for subspaces that are not covered by j. Developing the right part, this is exactly the form of a general Ising Hamiltonian where interaction are taken between the k-closest spins (tensor product of k terms σ^z with consecutive indices) and coupling are non-linear (for more detailed discussion, see Section VII).

Our goal is to find the best input bit-string x^* that maximizes the reward function $f_{\mathcal{R}}(x^*)$. We define it as

Problem I $\quad x_I^* = \arg\max f_{\mathcal{R}}(x).$

The association done above with a general Ising Hamiltonian allows us to state that, in general, this is a problem belonging to the complexity class NP-complete. This is a really complicated problem since the reward function does not present any particular properties such as continuity, convexity or differentiability and the solution space is of size 2^N where N is the length of any input bit-string. When the cost function does not present any simplifying property, it should be considered as a black box and state-of-the-art methods rely on meta-heuristics like simulated annealing or quantum annealing.

III. MONTE-CARLO COMPRESSIVE OPTIMIZATION ALGORITHM

In this section, we present our main result which is a new algorithm to solve combinatorial optimization problems like Problem I, i.e. compressible in the way described in Section II.

Consider the cost function over the set of all possible bit-strings of length N. It is a real-valued function $f : \{0,1\}^N \to \mathbb{R}^+$. It is also useful to consider the probability distribution π associated with the density function $f(x)$ (that is normalized).

Algorithm 1
Inputs: sample size n, threshold parameter t, sketch fucntion Φ, decoding procedure Δ.

1. Compute a sample of f called Sf. To do so: First, draw a sample $\mathcal{I}$ of n indices from the uniform distribution. Then apply the sampling operator $S_{\mathcal{I}}$ defined as $S_{\mathcal{I}} : x \mapsto x\delta_{x,y\in\mathcal{I}}$ to f.

2. Apply a *hard thresholding* operator T_t such that in T_tSf all values $Sf(x)$ lower than a given t are removed from the sample.

3. Apply a *sketching map* (also known as a measurement map or a feature map) Φ to get $y = \Phi(T_tSf)$. We call y the sketch vector and it contains an empirical estimation of *generalized moments* of the distribution π.

4. Apply a decoding procedure Δ to y and get $\tilde{\pi} = \Delta(y)$. The decoding procedure is taken from compressive sensing **greedy methods** (Matching pursuit, Orthogonal Matching Pursuit ...).

5. The largest line of $\tilde{\pi}$ is used as the maximum estimate.

The Monte-Carlo method [17] can be used to approximate generalized moments similar to the one from [16]. This is a well known result from the probabilistic computation theory (or ergodic theory), about a large number M of samples sampled independently from a probability distribution π,

$$\lim_{M\to\infty} \frac{1}{M} \sum_{x_{(i)}\sim\pi} g(x_{(i)}) = \sum_{x_{(i)}} g(x_{(i)})\pi(x_{(i)}) \tag{5}$$

$$\approx \frac{S}{M} \sum_{x_{(i)}\sim\mathcal{U}} g(x_{(i)})\pi(x_{(i)}), \tag{6}$$

where S is the size of the domain. Which means that a moment of a function g of a variable x with respect to a measure $\pi(x)$ can be approximated by sampling from the uniform distribution and computing the value of the density function $\pi(x)$ in those points.

We now discuss the choice of the sketching function Φ. We know from the compressive sensing theory that functions Φ which respect some properties such as the Restricted Isometry Property (RIP) are good candidates—in practice, often randomized linear maps are used. However, structured deterministic measurement functions can also work [15]. Also, for instance in [22, 23], structured measurement functions with different advantages were used. Moreover, the sketching function does not necessarily need to be linear, generalized moments of the distribution can be of different degrees. Introducing non-linearities such as a hard thresholding process [15] in practice gives us better performance with respect to random linear map alone. In addition, the choice of the sketching function directly impact the optimization step within the recovery algorithm. The freedom in the choice of the sketch function allows for flexibility in the algorithm complexity, which can be tailored to match one's available computational resources. Indeed, this turned out to be an essential element of the practical implementation and its justification presented in the next sections.

Lastly we underline the importance of using greedy methods as the matching pursuit rather than ℓ_1 minimization method such as the basis pursuit. Indeed, ℓ_1 recovery methods aim at recovering the exact sparse signal by putting more emphasis on the constraints. Even though this might be a desirable behavior in compressive sensing it is not here in Monte-Carlo Compressive Optimization. When putting more emphasis on the constraints, we are learning about the particular features from the selected sample rather than the global features from the distribution. This phenomenon is known in learning theory as overfitting. On the other hand, greedy methods like matching pursuit put emphasis on the sparsity. These methods focus on finding the one line that match the best all constraints and this line often matches with the maximum line of the distribution.

In section IV, we use several sketching functions to compute generalized moments as well as the Orthogonal Matching Pursuit for the decoding procedure. The sketching functions we used are defined as follows:
Quadruplets

Each line of the sketching matrix is defined by

$$\begin{aligned}\Phi^{i,i+1,i+2,i+3}_{x_1,x_2,x_3,x_4} &= 1_1 \otimes \ldots \otimes 1_{i-1} \\ &\otimes (1-x_1, x_1) \otimes (1-x_2, x_2) \\ &\otimes (1-x_3, x_3) \otimes (1-x_4, x_4) \\ &\otimes 1_{i+5} \otimes \ldots \otimes 1_N\end{aligned}$$

for every starting position $1 \leq i \leq N-3$ and x_is take all binary values.

Quintuples

Each line of the sketching matrix is defined by

$$\begin{aligned}\Phi^{i,i+1,i+2,i+3,i+4}_{x_1,x_2,x_3,x_4,x_5} &= 1_1 \otimes \ldots \otimes 1_{i-1} \\ &\otimes (1-x_1, x_1) \otimes (1-x_2, x_2) \\ &\otimes (1-x_3, x_3) \otimes (1-x_4, x_4) \\ &\otimes (1-x_5, x_5) \otimes 1_{i+6} \\ &\otimes \ldots \otimes 1_N\end{aligned}$$

for every starting position $1 \leq i \leq N-4$ and x_is take all binary values.

Random sketch A random binary matrix.

In Section V, we will focus on using the Matching Pursuit algorithm for decoding procedure. This defines the following problem for the first crucial step of Matching Pursuit:

Problem II $\quad x^*_{II} = \arg\max \Phi^T \Phi T S f_{\mathcal{R}}.$

IV. NUMERICAL RESULTS: COMPARISON WITH OTHER OPTIMIZATION METHODS

In this section, we numerically compare the performances of the Monte-Carlo Compressive Optimization algorithm when using different sketching functions Φ. We compare their performances with each other and with the dual annealing algorithm, known as the best heuristic method to handle general combinatorial optimization problems.

Setup : In order to work with reasonable computational time, we choose a problem of relatively small size, bit-strings of length $N = 12$ (space dimension is 2^N). This also allows for visualization of the problem structure when having a look at its, usually unknown, spectrum. Even though the case with longer length will be the subject of further study, the theoretical arguments convince us that similar results will still be valid. The problem of interest is shown in Figure 3 and was randomly drawn from the set of possible problems. Note that the problem belongs to a general class of Ising-like cost functions that are complicated to solve even for current state-of-the-art heuristic methods like dual annealing.

We chose a set of possible rules described by sequences of four, five or six bits. The number of each rules, as well as each pattern, are chosen randomly. The metric used in Figure 4 is the average distance (over the choice of samples) to the real optimum $f(x^*) - f(\hat{x})$. We compare this distance for different sample size n going from 50 to 300.

Additionally, Figure 1 gives the distribution of this same distance for different estimates coming from distinct samples. Another interesting metric is the Hamming distance of the estimate string $\hat{x}$ to the optimal one x^*. This Hamming distance distribution for different estimates coming from distinct sample appears in Figure 2.

In the case of the dual annealing, we chose the number of iterations to be of the same order as the sample size. Indeed, this is a fair comparison since the number of calls to the cost function f, supposedly costly, will remain the same in both cases. In addition, the annealing method is in fact similar to a sampling method—it can be associated with the Metropolis-Hastings algorithm [24] where the sampling domain is reduced at each iteration.

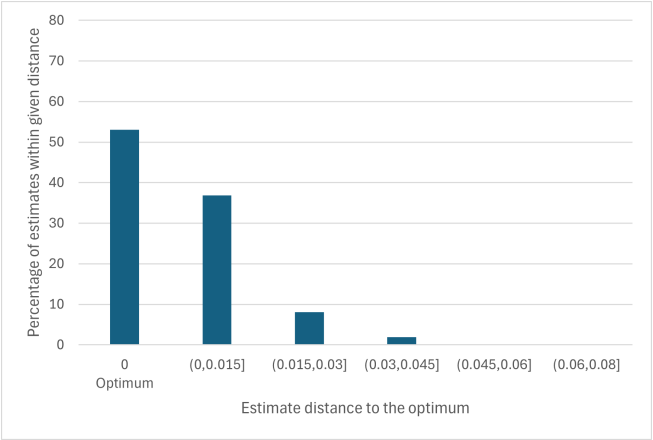
Results interpretation: The results are shown in Figure 4. First observation is that all three sketching functions reduce the distance to the optimum in a polynomial way. Quadruplets and quintuplets methods perform well compared to the random linear sketch function but also to the annealing. Indeed, for quadruplets and quintuplets methods, the distance to the optimum reduces in a similar or faster manner with respect to the annealing method. This is strong evidence that, on this class of compressible cost functions, our method can outperform the state-of-the-art dual annealing method, especially in the useful case of the small sample size solution.

Figure 1 shows that in most cases (more than 50% cases for non-random sketch method) we are able to find the actual optimum. Even when failed, the suggested solutions are close to the minimum within a tight interval from $f(x^*)$. Figure 2 shows that most of the time, for a random sample, the corresponding estimate belongs to the neighborhood of the solution x^* (for the metric space $\mathbb{F}_2$ with the Hamming distance). Even in the worst case scenario, the Hamming distance never drops below half its maximum, i.e. the algorithm never performs worse than a random choice.

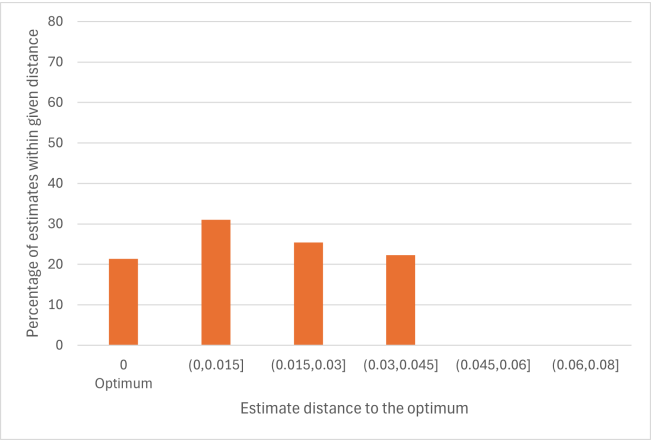
V. JUSTIFICATION OF ALGORITHM 1

Although, currently, we do not have the rigorous theoretical bounds on the probability with which Algorithm 1 approaches the correct solution with a given error, we can provide a heuristic justification of the algorithm supported by numerical simulation and rigorous proofs of lemmas and propositions whenever possible.

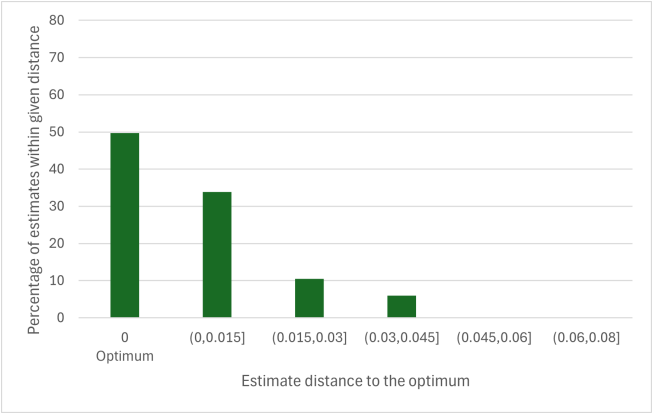
Consider a simplified situation with a single 3-bit rule $r = r_1 r_2 r_3$. We define our target problem, which is to find a binary string x^*_I that contains as many sub-string identical to r as possible. Notice that r is unknown and does not need to be known, while x^*_I is learned based on rewards given to randomly selected binary strings, where



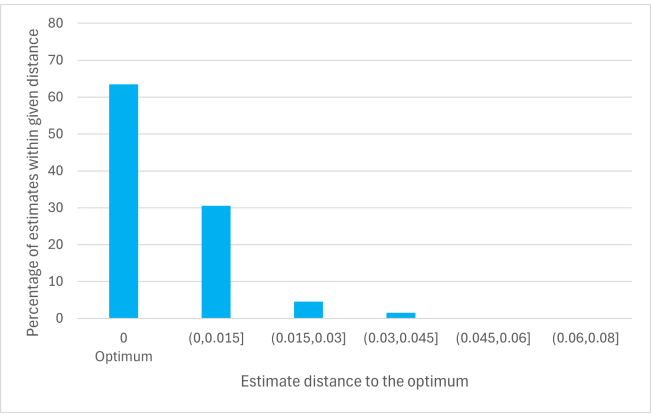
(a) Annealing



(b) Random sketch



(c) Quadruplets sketch



(d) Quintuples sketch

FIG. 1: Distribution of the distance between estimate solutions $f(x)$ and the optimum $f(x^*)$. On each plot, the percentage of estimate solutions that lies in given distance interval. The methods used are: (a) annealing and (b)(c)(d) Monte-Carlo Compressive Optimization with different sketch functions. $N = 12$ and $n = 250$

for a given string the reward is equal to the number of times the rule r appears in it.

In Algorithm 1, we solve Problem II where the rows of matrix Φ, are given by

Duplets

$$\Phi^{i,i+1}_{x_1,x_2} = 1_1 \otimes \ldots \otimes 1_{i-1} \otimes (1-x_1, x_1) \otimes (1-x_2, x_2) \otimes 1_{i+2} \otimes \ldots \otimes 1_N \quad (7)$$

for the $[4(i-1) + 2x_1 + x_2 + 1]$-th row and x_1 and x_2 are binary variables.

Define variable $\mu^i_x = \Phi^i_x Y$, where

$$\Phi^i_x = 1_1 \otimes \ldots \otimes 1_{i-1} \otimes (1-x, x) \otimes 1_{i+1} \otimes \ldots \otimes 1_N$$

is a **Singulet** map and

$$Y = (1-y_1, y_1) \otimes \ldots \otimes (1-y_n, y_n),$$

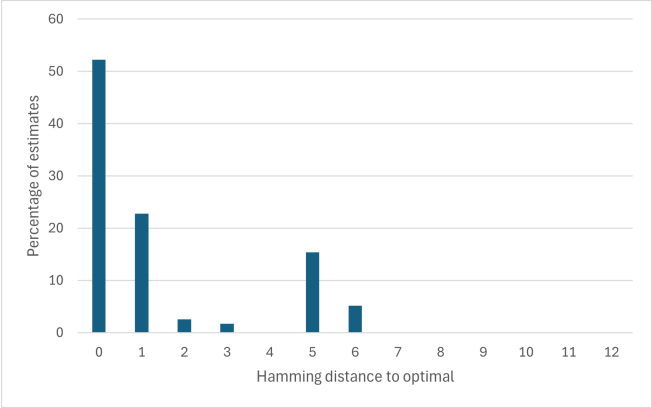
where is the zero vector except at the position given by the binary variables $y_i \in \{0, 1\}$. If vectors Y are randomly chosen from a distribution, the variable μ^i_x is a random variable. Assume Y are defined by arguments of the function $T_t S f_{\mathcal{R}}$—lines sampled uniformly from $f_{\mathcal{R}}$ and thresholded.

Moments $\varphi^{i,i+1}_{x_1,x_2} = \langle \mu^i_{x_1} \mu^{i+1}_{x_2} \rangle$ are calculated as $\Phi^{i,i+1}_{x_1,x_2} T_t S f_{\mathcal{R}}$, where $\Phi^{i,i+1}_{x_1,x_2}$ is given in (7). Up to normalization, these moments express averages over sampled elements of $f_{\mathcal{R}}$ with weights corresponding to the value each element. Analogously, we consider moments $\varphi^{i,i+1,i+2}_{x_1,x_2,x_3} = \langle \mu^i_{x_1} \mu^{i+1}_{x_2} \mu^{i+2}_{x_3} \rangle$. These averages are related to appropriate correlation functions

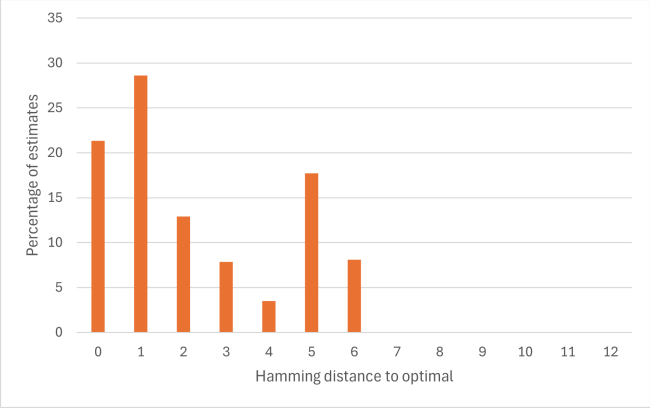
$$corr^{i_1 \ldots i_n}_{x_{i_1},\ldots,x_{i_n}} = \frac{\langle (\mu^{i_1}_{x_{i_1}} - \langle \mu^{i_1}_{x_{i_1}} \rangle) \ldots (\mu^{i_n}_{x_{i_n}} - \langle \mu^{i_n}_{x_{i_n}} \rangle) \rangle}{\langle \mu^{i_1}_{x_{i_1}} \rangle \ldots \langle \mu^{i_n}_{x_{i_n}} \rangle}.$$

Therefore, loosely speaking, the moments express correlations between a given bit configuration $x_1 \ldots x_n \in \Sigma_n$ and the appearance of this configuration in important elements of the sample from $f_{\mathcal{R}}$. The importance is specified by values of $f_{\mathcal{R}}$ that remain after sampling and thresholding.

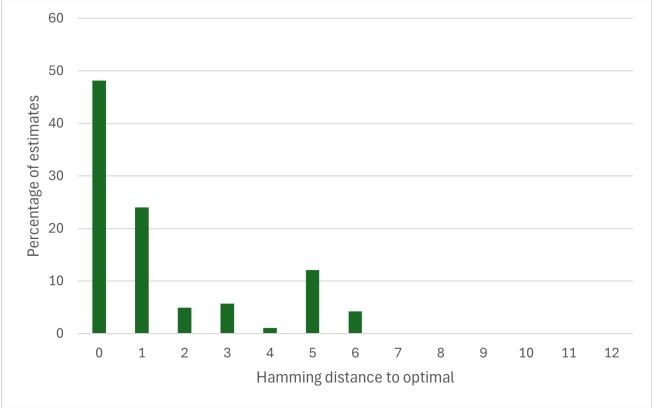
In Algorithm 1 we solve Problem II, which is, we find



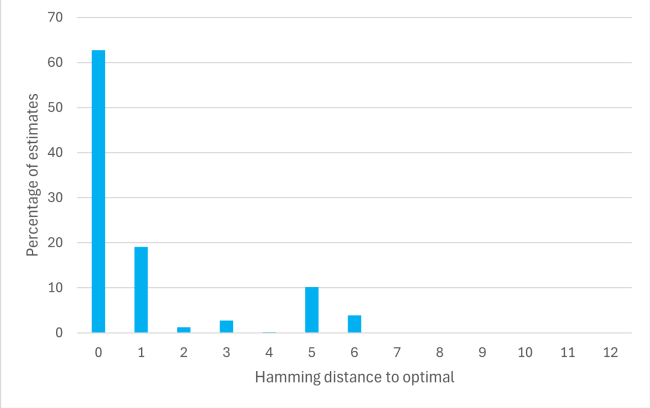
(a) Annealing



(b) Random sketch



(c) Quadruplets sketch



(d) Quintuples sketch

FIG. 2: Distribution of the Hamming distance between estimate solutions x and the optimum x^*. On each plot, the percentage of estimate solutions for a given hamming distance. The methods used are: (a) annealing and (b)(c)(d) Monte-Carlo Compressive Optimization with different sketch functions.$N = 12$ and $n = 250$

a string x^*_{II} for which the sum of correlations for neighboring variables is maximized,

$$x^*_{II} = \arg\max \sum_{i=1}^{N-1} \varphi^{i,i+1}_{x_1,x_2}. \tag{8}$$

Moreover, the way Algorithm 1 assigns the values $\varphi^{i,i+1}_{x_1,x_2}$ induces correlations between these values.

Our numerical analysis shows that solving Problem II often suffices to solve Problem I. Because x^*_I contains several times $r_1r_2r_3$ it is true that r_2 follows r_1 more often than in a random string (the same for the pair r_3 and r_2), and obviously, r_3 follows the pair r_1r_2 more often than in random sequences. Therefore, the minimum requirement for any solver of Problem I should be to propose candidate solutions x which:

- Condition 1: possibly often contain pairs of bits r_1r_2 and r_2r_3, and
- Condition 2: the pairs appear in proper order, r_2r_3 likely appear after r_1r_2.

Notice that the solution of Problem II is a binary string x^*_{II} which is defined by the lower indices of the variables $\mu^i_{x_i}$ for which the sum of correlations $\varphi^{i,i+1}_{x_1,x_2} = \langle \mu^i_{x_1} \mu^{i+1}_{x_2} \rangle$ is maximum (as in Equation (8)). In the following proposition (proven in Appendix A), we show that pairs of variables with x_1x_2 matching substrings of r have larger expected moments than bits that does not match r.

Proposition 1. Let $\varphi^{i,i+1}_x = \Phi^{i,i+1}_x f_{\mathcal{R}}$,

(i) for the problem with single rule r, for any $2 \leq i \leq N-2$, if r' is a substring of r, and z is not, then

$$\varphi^{i,i+1}_{r'} \geq \varphi^{i,i+1}_z,$$

(ii) for the problem with single rule r, for any i, there exists at least one r' substring of r such that, for any string z:

$$\varphi^{i,i+1}_{r'} \geq \varphi^{i,i+1}_z,$$

(iii) for the problem with single rule r, for any i, there exists at least one r' substring of r such that,

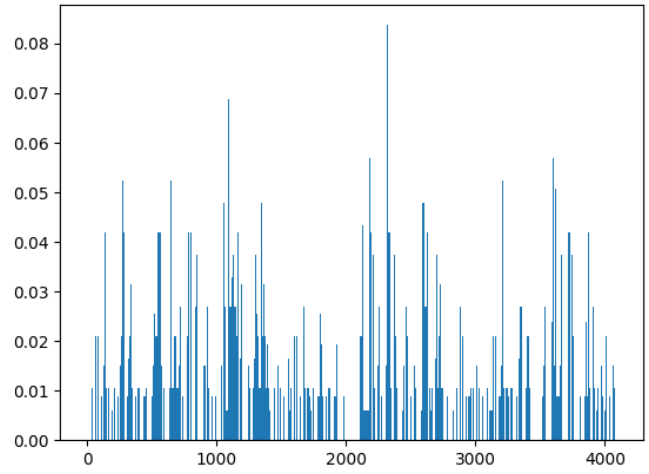


FIG. 3: Example of $f_{\mathcal{R}}$. On the x axis, we list all binary sequences. Their respective rewards are shown on the y axis.

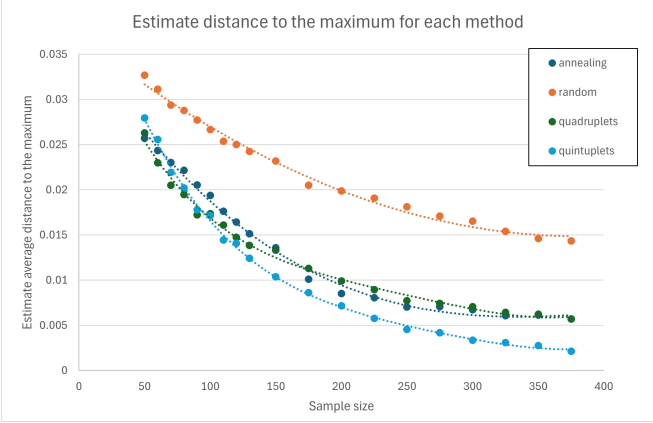


FIG. 4: Estimate distance to the minimum when the sample size increases. Comparison of dual annealing (dark blue) with our methods using 3 different sketch functions: (orange) random sketch, (green) quadruplets sketch, (light blue) quintuplets sketch.

for any string z the expectation values $\langle\varphi_r^{i,i+1}\rangle = \Phi_x^{i,i+1}T_tSf_{\mathcal{R}}$ satisfy:

$$\langle\varphi_{r'}^{i,i+1}\rangle \geq \langle\varphi_z^{i,i+1}\rangle.$$

Therefore, solvers of Problem II (like Algorithm 1) promotes outputs x_{II}^* including substrings of r, hence Condition 1 is satisfied. Condition 2 is also expected in the output x_{II}^* of Problem II solvers. This comes from the consistency of the bits in $\varphi_{x_1,x_2}^{i,i+1}$ and $\varphi_{x_2,x_3}^{i+1,i+2}$ — in the correct ordering bit x_2 is shared. In the opposite order, x_3 and x_1 cannot always be shared, and the bits x_3x_1 may not be among preferred substrings of r. The right order is also expected based on the effect of thresholding. It guaranties that in all addresses of randomly selected lines of $f_{\mathcal{R}}$, if r_1r_2 is found in position $i, i+1$ in any of the lines the probability that r_3 follows is high or increases with increasing threshold (this property is discussed in Appendix B). This in turn is reflected in high values of both the correlation $\varphi_{x_1,x_2}^{i,i+1}$ and $\varphi_{x_2,x_3}^{i+1,i+2}$. The thresholding argument penalizes the incorrect order of the correlation functions. Moreover, it penalizes strings with repeated subsings r_1r_2 not followed by r_3.

Therefore, we argue that Algorithm 1 which formally solves Problem II satisfies the minimum requirements of solver of Problem I. Thus, the output x_{II}^* of Algorithm 1 is also, with high probability, a solution of Problem I, as confirmed in the numerical simulations.

The arguments presented here are focused on providing an understanding based on simplified assumptions including a single rule of size three. The generalization for multiple different rules which is still confirmed by the numerical tests is in many cases a complicated problem. The rigorous proof can go along the same line of reasoning. However, calculating related probabilities rigorously can require solving counting satisfying assignments of logical formulas which are $\#P$ problems. Computing upper and lower bounds on the success probability remains an open problem.

VI. STABILITY AND UNIQUENESS

In what follows, we support the justification of Algorithm 1 with additional results demonstrating its robustness with respect to the choice of the sample and its responsiveness to small changes of the rules. These arguments justify the stability under sampling and uniqueness of the solution of Problem I and Problem II for a specific set of rules.

A. Moment concentration—Stability

Let $\mathcal{R}$ be a given set of rule and $f_{\mathcal{R}}(x)$ its associated objective function. Let Φ be a random linear sketching function. Let $\{x_{(i)}\}_{i\in I}$ be a sample of size n from the uniform distribution.

Theorem 1. [Moment concentration]
We call $y = \mathbb{E}[\Phi(f_{\mathcal{R}}(x))]$ the "true moment" of the function $f_{\mathcal{R}}$ and $\hat{y}_n = \frac{1}{n}\sum\Phi(f_{\mathcal{R}}(x_{(i)}))$ the estimated moment of the function $f_{\mathcal{R}}$.
The following inequality holds for any y and $\hat{y}_n$:

$$\mathbb{P}(|y - \hat{y}_n| \leq \epsilon) \geq 1 - \frac{\sigma^2}{\epsilon^2 n} \tag{9}$$

where $\sigma^2 = \mathrm{Var}[\Phi(f_{\mathcal{R}}(x))]$ and $\epsilon > 0$ is a positive constant.

See Appendix C 1 for the proof.

The value $\hat{y}_n$ is also called a Monte Carlo estimation of y.

In our algorithm, the sketching function is replaced by a linear map computing M distinct generalized moments. Thus, Theorem 1 can be generalized, using the triangle inequality, as follows:

$$\mathbb{P}(\hat{y}_n \in \mathcal{B}_\epsilon) \geq 1 - \frac{\mathrm{Tr}\{\Sigma\}}{\epsilon^2 n}, \tag{10}$$

where $\mathrm{Tr}\{\Sigma\}$ is the sum of the variances associated with each sketching function and $\mathcal{B}_\epsilon$ is the M dimensional ball centered at y of radius ϵ (which generalized $|y - \hat{y}_n| \leq \epsilon$).

Now looking at Theorem 1 (or equation (10) for the general case), implies that there are two ways to increase the probability of concentration for a given $\epsilon > 0$. The first one, however not desirable in our case, would be to increase the sample size n. Indeed, sampling from our objective function is assumed to be costly and we want to limit the number of calls. The second way would require to reduce the variance σ^2. In fact, if one can find a similar function to $f_\mathcal{R}$ where the maximum is preserved but whose variance is slightly lowered, one could increase the probability to concentrate. Such a function could be found by thresholding the original function. In consequence, we can show that a clever use of thresholding usually results in lowering the variance, and thus achieves our goal.

We define a thresholded moment $y^{(t)} = \mathbb{E}[\Phi(T_t f_\mathcal{R}(x))]$, where T_t is the thresholding operator which sets every value lower than t to 0, i.e.

$$T_t : x \mapsto \begin{cases} x \text{ if } x \geq t \\ 0 \text{ else} \end{cases} .$$

For the rest of the section, we assume that the threshold parameter is not too large in such a way that the thresholded moment $y^{(t)}$ would at any time remain close to the real moment y.

Theorem 2. [Threshold increases probabilities]

Let ${\sigma^{(t)}}^2 = \mathrm{Var}[\Phi(T_t f_\mathcal{R}(x))]$ be the variance of the thresholded function. Recall, σ^2 is the variance of the original function $f_\mathcal{R}$.

There exists a value t for the thresholding parameter for which the variance decreases:

$${\sigma^{(t)}}^2 \leq \sigma^2 \tag{11}$$

Moreover, for any $k > 0$ and such that $1 - \frac{1}{k^2} \geq 1 - \frac{\sigma^2}{\epsilon^2 n}$, there exists a parameter t such that:

$$\mathbb{P}(\hat{y}_n \in \mathcal{B}_\epsilon) \geq 1 - \frac{1}{k^2} \geq 1 - \frac{\sigma^2}{\epsilon^2 n} \tag{12}$$

which improves the bound on $\mathbb{P}(\hat{y}_n \in \mathcal{B}_\epsilon)$ increasing each probability.

See Appendix C 2 for the proof.

From now on, rather than looking at the probability to find $\hat{y}_n$ in a given radius ϵ, we focus on radius sizes having the same probability bound for different sets of rules. Let $k > 0$ and we define $\epsilon^{(t)}$ to be $\epsilon^{(t)} = \frac{k\sigma^{(t)}}{\sqrt{n}}$ because Theorem 1 needs to hold. This means the value of $\epsilon^{(t)}$ now changes as $\sigma^{(t)}$ decreases (when threshold increases).

Theorem 3. [Arbitrary closeness]

Let $\mathcal{R}_i$ be a given set of rules and $\hat{y}_n(\mathcal{R}_i)$ be the moments computed from the function $f_{\mathcal{R}_i}$. Let $\epsilon^{(t)}(\mathcal{R}_i)$ the radius of the ball $\mathcal{B}_{\epsilon^{(t)}(\mathcal{R}_i)}$ that satisfies

$$\mathbb{P}(\hat{y}_n(\mathcal{R}_i) \in \mathcal{B}_{\epsilon^{(t)}(\mathcal{R}_i)}) \geq 1 - \frac{1}{k^2}. \tag{13}$$

Then, there exists $D^{(t)} \in \mathbb{R}^+$ such that

$$\mathbb{P}(\hat{y}_n(\mathcal{R}_i) \in \mathcal{B}_{D^{(t)}/2}) \geq 1 - \frac{1}{k^2} \tag{14}$$

where D is common to all $\mathcal{R}_i$ and is given by :

$$D^{(t)} = 2 \max_{\mathcal{R}_i} \{\epsilon(R_i)\}. \tag{15}$$

Moreover, $D^{(t)}$ becomes arbitrary small as t increases.

Proof. Because for each set of rules the moments concentrate in a ball of radius $\epsilon(\mathcal{R}_i)$, then they also all belong to the ball of radius $D^{(t)}/2$. From a corollary of Theorem 2, if k is fixed then ϵ decreases with thresholding for any set of rules. So $D^{(t)}$ can go arbitrary small as the threshold parameter changes. □

From Theorem 3, we conclude that thresholding the objective function $f_\mathcal{R}$ allows for a reduction of the variance and thus a better concentration of the moments. Moreover, an interesting property of thresholding is that it can commute with sampling; indeed, a threshold applied on the value sampled from $f_\mathcal{R}$ is equivalent to sampling from the function $f_\mathcal{R}$ after it was thresholded. Thus, it makes it practical to apply threshold on the samples rather than the full function, allowing to benefit from the concentration in our method.

B. Rules distinguishability—Uniqueness

In this section, we will show that the distinguishability between two rules embedded by F_N, as described in Section II, can be increased arbitrary by increasing N. To do so, we will study the transform F_N that embeds the rules into the optimization space and was defined in equation 2.

Let's start with some properties of the transform F_N:

Claim 1. Given a string a (a rule) of size k:

1. $\|F_N(a)\| \geq 0$
2. $F_k = id$ (if $n = k$, the transform is the identity), i.e., $F_k(a) = \mathbf{a}$.
3. F_N can be written as a $2^N \times 2^k$ rectangular matrix.

The proofs of Claim 1 are trivial. Additional properties of the F_N Transform can be found in Appendix E
From the properties of Claim 1, we now prove the following two lemma.

Lemma 1.

$$F_N(a) = F_N(b) \iff a = b \tag{16}$$

See Appendix D 1 for the proof.

Lemma 2. When $a \neq b$,

$$\begin{aligned}&\|F_{k+l}(a) - F_{k+l}(b)\|_2^2 \\ &\geq \begin{cases} 2^{l+1}\left(\frac{3}{5}l - \frac{1}{9} - \frac{\frac{4}{3}k + \frac{2}{3}l + \frac{20}{9}}{2^{k-2}}\right) & (k : \text{even}) \\ 2^{l+1}\left(\frac{3}{5}l - \frac{1}{9} - \frac{\frac{5}{3}k + \frac{1}{3}l + \frac{16}{9}}{2^{k-2}}\right) & (k : \text{odd}) \end{cases}\end{aligned}$$

See Appendix D 2 for the proof.

Lemma 1 shows the injectivity of the F_N transform, meaning that two distinct rules necessarily have different transforms. Lemma 2 expresses the separation between the transform of different rules in the ℓ_2 space. It shows that two distinct rules are necessarily apart by a distance that increases exponentially with the parameter N determining the size of the optimization space.

Theorem 4. [Rules separations]
For any $a, b \neq a \in \{0,1\}^k$ and for any $d \geq 0$, there exists N such that

$$\|F_N(a) - F_N(b)\|_2 \geq d.$$

Proof. We combine both previous lemmas. Lemma 2 tells us that the distance between a and b increases exponentially when N grows. Thus, for any $d \geq 0$, there exists N such that $\|F_N(a) - F_N(b)\|_2 \geq d$. □

Theorem 5. [Moments arbitrary distance]
Let $\Phi : \mathbb{R}^{2^N} \to \mathbb{R}^M$ be a map computing M generalized moments of the function π whose density is $f_{\mathcal{R}}$. Assume Φ follows the Restricted Isometry Property (RIP). For any $a, b \neq a \in \{0,1\}^k$ and for any $D \geq 0$, there exists N such that:

$$\|\Phi(F_N(a)) - \Phi(F_N(b))\|_2 \geq D \tag{17}$$

Proof. Since Φ respects the RIP, we have:

$$\begin{aligned}(1-\delta)\|F_N(a) - F_N(b)\|_2 &\leq \|\Phi(F_N(a) - F_N(b))\|_2 \\ &\leq (1+\delta)\|F_N(a) - F_N(b)\|_2\end{aligned}$$

where $\delta \in [0,1]$ is the fixed Restricted Isometry constant associated to Φ. In particular:

$$\|F_N(a) - F_N(b)\|_2 \leq \frac{1}{1-\delta}\|\Phi(F_N(a) - F_N(b))\|_2 \tag{18}$$

However, from Theorem 4 we know that there exists N such that:

$$d \leq \|F_N(a) - F_N(b)\|_2 \tag{19}$$

for any d. And so, for the same N, and because Φ is linear:

$$d(1-\delta) \leq \|\Phi(F_N(a)) - \Phi(F_N(b))\|_2$$

Because this holds for any d, one just need to chose it in such a way that $D/(1-\delta) \leq d$ to get the desired property. □

Theorem 4 and Theorem 5 are the main results of this section. Theorem 4 tells us that distinct rules are separated in the transformed space, i.e. optimization space, by an arbitrary distance as long as the space is large enough. Theorem 5 extends this property to any generalized moment computed from these rules as long as the sketching function Φ respects the RIP.

C. Final arguments

We know from Theorem 5 that the moments computed from distinct rules can be arbitrary distant which reads:

$$\|\Phi(F_N(a)) - \Phi(F_N(b))\| \geq D. \tag{20}$$

Because the optimization problem is fixed, so is the dimension of the space, meaning N is given. We chose for D appearing in Theorem 5 the greatest possible values that N satisfies. Next having a look at Theorem 3, we know that there exist $D^{(t)}$ such that for both set of rules $\{a\}$ and $\{b\}$:

$$\mathbb{P}(\hat{y}_n \in \mathcal{B}_{D^{(t)}/2}) \geq 1 - \frac{1}{k^2}. \tag{21}$$

and we chose k for the right hand side to be a large probability. Moreover, $D^{(t)} = 2\max\{\epsilon^{(t)}(a), \epsilon^{(t)}(b)\}$ is decreasing with the threshold parameter t. Thus, we chose t for $D^{(t)}$ to be smaller than D. We end up with the situation where moments y_a and y_b from different rules are separated by distance D and each Monte-Carlo approximation of these moments lies, with high probability $1 - \frac{1}{k^2}$, within the ball of radius $D/2$. One clearly sees that no approximation of a moment from the rule a can be misunderstood as a moment from the rule b since they live in different balls that do not intersect.

Since our approximate moments contains features of the true distribution it should be, in principle, possible to recover features of it, such as its maximum. Remember, the moment given by any sample and the moments from the full distribution are all similar. Since we do not use any specific feature of the sample other than the sketching function, we will tend to recover solutions that are representative of the distribution global features and mostly, when combined with Section V, recover a solution containing the maximum line. This will work as

long as we are not overfitting the sample—too many moment computed by the sketching function will determine features of this exact sample rather than global features.

VII. DISCUSSION ON APPLICATIONS AND QUANTUM COMPUTERS

In Section II we defined combinatorial optimization problems that are compressible. In Section III we gave an algorithm to solve such problems and Section IV showed the performance of our method compared to the best existing method—dual annealing. Finally, theoretical justification of this algorithm was discussed in Section V and Section VI. In this last section, we show that compressible combinatorial problems naturally show up in many fields by giving two examples. One is the problem of learning optimal policy in Reinforcement Learning. The other is the problem of finding the ground state of an Ising Hamiltonian. Note that compressible problems are not restricted to these two fields but rather may appear in other contexts: flow problems, knapsack problem, traveling salesman problem, machine learning, etc...

A. Reinforcement Learning

Reinforcement Learning (RL) is a crucial technique in today's machine learning due to its central role in several applications, such as optimal control theory, robotics, or game theory. It studies the behavior of an agent who aims to learn the nearly optimal policy for a given task. We can think of this task as a game. Given the set of game's states and actions, the policy dictates the choice of the next action for a given state. Reinforcement Learning at its beginning focuses on the exploitation of dynamic programming methods as imagined by Richard Bellman. A typical example is the Q-learning algorithm [25], which makes use of the Bellman equation to update the so-called Q-table. However, when the space of states and actions becomes too large, current methods rely on stochastic principles. A clever use of the so-called policy networks and of the stochastic gradient descent allowed for achieving remarkable results [26]. This approach avoids the bottleneck of dynamical methods, but remains close to Bellman's original ideas; the policy is refined iteratively from partial returns until it eventually converges to nearly optimal solution. Another way to tackle Reinforcement Learning tasks are known as Monte-Carlo methods. In that case, we try to solve the reinforcement learning task based on complete returns obtained from full episodes. The name Monte-Carlo comes from the fact that one is sampling through the space of complete returns. This is the case of the method we presented in Section III. Both approaches have benefits and drawbacks, and the choice of which method to use is problem-dependent. Let's consider the following "game". At each step, the agent has to choose between two possible actions labeled 0 and 1. Only after one complete episode, which consists of a sequence of N actions, a total reward is granted. The goal is to find a nearly optimal policy maximizing the total reward. Since one has only access to the final returns, with the reward mechanism in each episode acting like a black-box, the use of Monte-Carlo methods seems reasonable in this context. Thus, this RL scenario indeed exhibit a problem structure similar to the combinatorial optimization of a compressible function. Our approach can be use here as a competitive method in a context where policy network could not be exploited.

B. Ising Hamiltonians and Quantum Computers

The Ising model has been studied by physicist to understand the behavior of certain phenomena as ferromagnetism. The energy of a given configuration is given by the Hamiltonian $\mathcal{H}$ of the system. With the emergence of quantum computing, as well as the needs for simulating quantum physics, analogy are often made between Hamiltonians and cost functions. In this context, we could refer to $\frac{1}{2}$-spins to talk about binary variables and spin-configuration to talk about a given bit-string. The Hamiltonian of an Ising spin-chain is given by:

$$\mathcal{H}(\sigma) = -\sum_i h_i\sigma_i - \sum_{\langle i,j\rangle} J_{i,j}\sigma_i\sigma_j \tag{22}$$

where $\langle i,j\rangle$ means nearest neighbor pairs of spins. Also notice that, in typical cases from physics, h_i and $J_{i,j}$ are independent of the actual configuration of σ_i and σ_j but only linked to relative positions i and j. We will call this kind of typical Ising Hamiltonians "linear" since all coupling terms are linear. The nearest-neighbor case is solvable by classical computers in polynomial time by using dynamic programming methods due to its recursive property when computing the cost of a given configuration. However, this is not the case in more general Ising problems. Indeed, Ising problems such as frustrated ferromagnetic models, spin-glasses, random energy models or non-linear Ising models are known to be intractable for classical computers. A random energy Ising is given by:

$$\mathcal{H}(\sigma) = \sum_{\sigma_s\in P} \omega_s\sigma_s \tag{23}$$

where σ_s are Pauli-Z strings within the Pauli group P of given length s pauli strings.

We already made a parallel between such Random Energy Model Ising Hamiltonian and our cost function described in section II. Indeed, our method is a new promising approach to tackle the problem of finding the ground state of such Ising-like Hamiltonians as no efficient classical methods are known (and there won't be if $P \neq NP$).

Quantum computers also seem to have a crucial role to play in combinatorial optimization, especially since so

many cost function can me mapped to Ising-like Hamiltonians. Indeed, quantum phenomena exhibit structure from complicated Ising problems—for instance spin-glass—and even though quantum computers might not be able to solve the optimization problem in polynomial time (if NP $\not\subset$ BQP) [27] they might provide good heuristics to tackle those kind of optimization problems (see quantum annealing [6, 7], QAOA [28–30], HVA [31], Imaginary time evolution [32], dissipative dynamics [33, 34]...)

In this last section, we connect our new results to the previous work to see how quantum computers can contribute to Monte-Carlo Compressive Optimization. In our last study, we demonstrated how a quantum computer could improve compressive sensing methods [23]. The method consists in using structured patterns for the measurement map Φ in such a way that the optimization step in matching pursuit is reduced to finding the ground state of an Ising Hamiltonian. One relevant aspect is that the use of structured pattern allows us to save memory as we do not need to store an exponentially large random matrix anymore. Another aspect was that the crucial optimization step of the matching pursuit algorithm reduces to finding the solution of an Ising problem. If the patterns used for Φ corresponded to fixed nearest-neighbor bits in the binary sequences, the optimization could be handled by a dynamic programming algorithm, while for more complex patterns (like patterns corresponding to distant pairs of bits) the solution can be obtained using a quantum computer apriori faster than using known algorithms on a classical computer. Indeed, the optimization for the distant-pairs measurement patterns reduces to a corresponding spin-glass problem and can be approached using QAOA or quantum annealing. As these kinds of systems seem to be quantum mechanical by nature, we have good reasons to think that quantum computer can perform better which is in accord with our previous results.

As it was briefly mentioned above, we can think of the rules as encoding a random energy Ising problem with non-linear coupling terms. This is, generally speaking, a really hard problem. However, when applying the Monte-Carlo Compressive Optimization algorithm with a sketching function made of structured patterns, we indeed map the complex Ising problem into a spin-chain or spin-glass Ising problem where important features of the problem such as its optima can be preserved. Moreover, the mapping is done using a sample of the solutions space, hence it is computable in reasonable time. This really is a major phenomenon since the spin-glass or spin-chain problems are much easier to solve and, in many cases, we care more about the optima of a function than we do for its other properties.

ACKNOWLEDGMENTS

This work was supported by JST Moonshot R&D, Grant No. JPMJMS226C and Grant No. JPMJMS2061, JST ASPIRE, Grant No. JPMJAP2427, JST COI-NEXT Grant No. JPMJPF2221, JST SPRING Grant No. JPMJSP2123.

[1] H. Robbins and S. Monro, The Annals of Mathematical Statistics **22**, 400 (1951), publisher: Institute of Mathematical Statistics.
[2] J. A. Nelder and R. Mead, The Computer Journal **7**, 308 (1965), _eprint: https://academic.oup.com/comjnl/article-pdf/7/4/308/1013182/7-4-308.pdf.
[3] M. J. D. Powell, in *Advances in Optimization and Numerical Analysis*, edited by S. Gomez and J.-P. Hennart (Springer Netherlands, Dordrecht, 1994) pp. 51–67.
[4] A. Lucas, Frontiers in Physics **2** (2014), 10.3389/fphy.2014.00005, arXiv:1302.5843 [cond-mat].
[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Science **220**, 671 (1983).
[6] T. Kadowaki and H. Nishimori, Physical Review E **58**, 5355 (1998).
[7] M. Ohzeki and H. Nishimori, "Quantum annealing: An introduction and new developments," (2010), arXiv:1006.1696 [cond-mat].
[8] R. Bellman, Bulletin of the American Mathematical Society **60**, 503 (1954).
[9] F. Barahona, Journal of Physics A: Mathematical and General **15**, 3241 (1982).
[10] N. Schuch and J. I. Cirac, Phys. Rev. A **82**, 012314 (2010).
[11] D. Delahaye, S. Chaimatanan, and M. Mongeau, in *Handbook of Metaheuristics*, edited by M. Gendreau and J.-Y. Potvin (Springer International Publishing, Cham, 2019) pp. 1–35.
[12] E. Candes, J. Romberg, and T. Tao, IEEE Transactions on Information Theory **52**, 489 (2006).
[13] D. Donoho, IEEE Transactions on Information Theory **52**, 1289 (2006).
[14] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin, Constructive Approximation **28**, 253 (2008).
[15] S. Foucart and H. Rauhut, in *A Mathematical Introduction to Compressive Sensing*, edited by S. Foucart and H. Rauhut (Springer, New York, NY, 2013) pp. 61–75.
[16] R. Gribonval, G. Blanchard, N. Keriven, and Y. Traonmilin, "Compressive Statistical Learning with Random Feature Moments," (2021), arXiv:1706.07180 [stat].
[17] N. Metropolis and S. Ulam, Journal of the American Statistical Association **44**, 335 (1949), publisher: [American Statistical Association, Taylor & Francis, Ltd.].
[18] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, The Journal of Chemical Physics **21**, 1087 (1953).
[19] Y. Xiang, D. Y. Sun, W. Fan, and X. G. Gong, Physics Letters A **233**, 216 (1997).
[20] C. Tsallis, Journal of Statistical Physics **52**, 479 (1988).
[21] C. Tsallis and D. A. Stariolo, Physica A: Statistical Mechanics and its Applications **233**, 395 (1996).

[22] K. V. Jacob, E. Kaur, W. Roga, and M. Takeoka, Phys. Rev. A **102**, 032403 (2020).
[23] B. Chevalier, W. Roga, and M. Takeoka, Physical Review A **110**, 062410 (2024), publisher: American Physical Society.
[24] W. K. Hastings, Biometrika **57**, 97 (1970).
[25] C. J. C. H. Watkins and P. Dayan, Machine Learning **8**, 279 (1992).
[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, Nature **518**, 529 (2015), publisher: Nature Publishing Group.
[27] S. Aaronson, "NP-complete Problems and Physical Reality," (2005), arXiv:quant-ph/0502072.
[28] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum Computation by Adiabatic Evolution," (2000), arXiv:quant-ph/0001106.
[29] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," (2014), arXiv:1411.4028 [quant-ph].
[30] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, Physical Review X **10**, 021067 (2020).
[31] C.-Y. Park and N. Killoran, Quantum **8**, 1239 (2024), publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften.
[32] S. McArdle, T. Jones, S. Endo, Y. Li, S. C. Benjamin, and X. Yuan, npj Quantum Information **5**, 1 (2019).
[33] T. S. Cubitt, "Dissipative ground state preparation and the Dissipative Quantum Eigensolver," (2023), arXiv:2303.11962 [quant-ph].
[34] Y. Zhan, Z. Ding, J. Huhn, J. Gray, J. Preskill, G. K.-L. Chan, and L. Lin, "Rapid quantum ground state preparation via dissipative dynamics," (2025), arXiv:2503.15827 [quant-ph].

Appendix A: Proof of Proposition 1—Larger momemnts of substrings of the rules

Proof. Formally,

$$\varphi_x^{i,i+1} = (\Phi_x^{i,i+1})^T F_N(r),$$

where

$$\Phi_x^{i,i+1} = (1_1 \otimes \ldots \otimes 1_{i-1} \otimes x \otimes 1_{i+2} \otimes \ldots \otimes 1_N)$$

and

$$F_N(r) = r \otimes 1_4 \otimes \ldots \otimes 1_N + 1_1 \otimes r \otimes 1_5 \otimes \ldots \otimes 1_N ... + 1_1 \otimes \ldots \otimes 1_{N-3} \otimes r.$$

i) For any position $2 \leq i \leq N-2$,
For a measurement bits x_1x_2 to match the rules at those positions, there are four possibilities shown in Table I:

- If x_1x_2 matches either $r_1r_2r_1r_2r_2r_3 \rightarrow$ the energy increases by 2^{N-3}
- If x_1 matches $r_3 \rightarrow$ the energy increases by 2^{N-4}
- If x_2 matches $r_1 \rightarrow$ the energy increases by 2^{N-4}
- Any other case $\rightarrow$ the energy increases by 0

So in total, any sketching pattern that are substring of a rule (r_1r_2 or r_2r_3) will bring 2^{N-3} energy. The other most energetic case is $x_1x_2 = r_3r_1$ which brings $2 \times 2^{N-4} = 2^{N-3}$ energy (same amount).
All other cases bring less energy. So we have indeed for all subtring r and any z $m_r^{i,i+1} \geq m_z^{i,i+1}$. If it's true for all, then it's also true for at least one.
ii) We study matching at the beginning and at the end
a) matching in position 1,2, (Table II).

.	.	x_1	x_2	.	.
r_1	r_2	r_3	.	.	.
.	r_1	r_2	r_3	.	.
.	.	r_1	r_2	r_3	.
.	.	.	r_1	r_2	r_3

TABLE I: Configurations in which two bits x_1 and x_2 inside a string can mach the rule $r_1r_2r_3$.

x_1	x_2	.	.	.	.
r_1	r_2	r_3	.	.	.
.	r_1	r_2	r_3	.	.
.	.	r_1	r_2	r_3	.
.	.	.	r_1	r_2	r_3

TABLE II: Configurations in which two bits x_1 and x_2 at the beginning of a string can mach the rule $r_1r_2r_3$.

.	.	.	.	x_1	x_2
r_1	r_2	r_3	.	.	.
.	r_1	r_2	r_3	.	.
.	.	r_1	r_2	r_3	.
.	.	.	r_1	r_2	r_3

TABLE III: Configurations in which two bits x_1 and x_2 at the end of a string can mach the rule $r_1r_2r_3$.

- If x_1x_2 match $r_1r_2 \rightarrow$ the energy increases by 2^{N-3}
- If x_2 matches $r_1 \rightarrow$ the energy increases by 2^{N-4}
- Any other case $\rightarrow$ the energy increases by 0

So there is one substring r , namely r_1r_2, that brings more energy than any other string.
The other substring all bring 0 energy for this position.
b) matching in position N-1,N (Table III).

- If x_xm_2 match $r_2r_3 \rightarrow$ the energy increases by 2^{N-3}
- If x_1 matches $r_3 \rightarrow$ the energy increases by 2^{N-4}
- Any other case $\rightarrow$ the energy increases by 0

So there is one substring r , namely r_2r_3, that brings more energy than any other string.
The other substring all bring 0 energy for this position.
Part (iii) easily follows from (ii). □

For the problems with multiple rules the above reasoning should be completed by multiplying powers of two by appropriate factors proportional to values of rewards. The statements regarding the ordering of $< \varphi_x^{i,i+1} >$ for x belonging to different rules, or appearing in different rules, or not belonging to any of them is more complicated. However, similar arguments to Propositions 1 supported by numerical simulations allow us to conjecture that Algorithm 1 assigns higher values $\langle \varphi_x^{i,i+1} \rangle$ to pairs of bits x which are parts of the rules.

Appendix B: Probability of realizing the rules at a given position knowing it appears q times

We want to compute the probability for a rule of size m to realize after matching $m-1$ bits given that the strings of interest all contains exactly q times the rules. We compute for a simple example as it is already complicated. In our cases we take the rule 001 of length $m = 3$ and the string contains a single time $q = 1$ the rule.
We will proceed as follow:

1. Count the string that contains a single 001
2. Belong them estimate the number of that contains L times 00 as a substring. The probability in that string to find 001 after 00 is given by $\frac{1}{L}$
3. Compute the expectated probability for all L

1. Counting the string that contains a single 001

We start by counting the string that contains at least one 001 as a substring. We want to count all the strings of a given length N which contains at least one '001' substring. We define $E_N = \{s \text{ — } s \text{ has at least one substring equals 001 and } |s| = N\}$ The goal is to compute its cardinality $e_N := \#E_N$.

Claim 2. There is an explicit expression for e_N given by

$$e_N = \sum_{k=1}^{\lfloor N/3 \rfloor} (-1)^{k+1} \binom{N-2k}{k} 2^{N-3k} \tag{B1}$$

Proof. Let $A_i = \{s \text{ — } s \text{ have its } s_i \text{ substring equals to } 001\}$. By definition we have $E_N = \bigcup_{i=1}^{N-2} A_i$. We now use the inclusion-exclusion principle (see below) to get

$$e_N = \sum_{\neq J \subseteq \{1,\dots,n\}} (-1)^{|J|+1} \# \bigcap_{j \in J} A_j$$

Because the string '001' does not overlap with itself: if $i, j \in J$ are less than 3 apart (length of the substring) that is $|i-j| < 3$, then $\# \bigcap_{j \in J} A_j = 0$ for this given J. So contribution only comes when choosing the element of J to be at least 3 apart. Let's consider all the J of a given size k ex: $k = 1$ then $J = \{1\}$ or $\{2\} \dots$ For a given k, we fix $3k$ bits in the string s and there are $N - 3k$ free bits. This give a cardinality of $\#A_j = 2^{N-3k}$, for $j \in J$ and $\#J = k$ (elements chosen 3 apart). Finally the number of ways to chose k starting position for '001' in s of length $N$is given by $\binom{N-2k}{k}$. All together we have

$$e_N = \sum_{k=1}^{\lfloor N/3 \rfloor} (-1)^{k+1} \binom{N-2k}{k} 2^{N-3k}$$

Because all sets A_j have the same cardinality (2^{N-3k}) for a given k, we just need to count them and multiply their contributions. This is a common situation when the result depends only on the number of sets in the intersection and not on which sets appears. □

We can now count the N length strings which contains a single time 001 as a substring. We want to compute the number of string of length N which contains '001' a single time as a substring. We call $S_N = \{s \text{ — } s \text{ has exactly one substring equals 001 and } |s| = N\}$ The goal is to compute its cardinality $s_N := \#S_N$

Claim 3. There exists a recursive formula for s_N given by

$$s_{N+1} = 2s_N - s_{N-2} + 2^{N-2} - e_{N-2} \tag{B2}$$

where

$$e_{N-2} = \sum_{k=1}^{\lfloor (N-2)/3 \rfloor} (-1)^{k+1} \binom{N-2(k+1)}{k} 2^{N-3k-2} \tag{B3}$$

Proof. First we compute the first terms: $s_1 = 0$ and $s_2 = 0$ $s_3 = 1$ because $S_3 = \{001\}$ $s_4 = 4$ because $S_4 = \{001*, *001\}$ Going from N to $N+1$ we add strings to S_{N+1} from two sources:

1. For all string $m \in S_N$ we can build a new string $m' = m \cdot *$ If m doesn't end by '00' then m' contains a single time '001' because m itself contains a single time '001'. Because for $m \in S_N$, if m end by '00' the others $N-2$ bits are free and should contains '001' a single time. The number of string m ending by '00' is exactly s_{N-2}. The total contribution is then $2s_N - s_{N-2}$.

2. We need to add all the other strings that end by '001' and did not contains any '001' before. Because the last 3 bits are fixed to '001' the remaining $N-2$ bits are free and should not contain any '001' . Thus there are #(string of length N-2) $- e_{N-2}$ such strings (where e_N is defined in the previous section) The total contribution is then $2^{N-2} - e_{N-2}$.

Combining (1) and (2) achieves the proof. □

Claim 4. There exists an explicit formula for s_N given by

$$s_N = \sum_{k=1}^{\lfloor N/3 \rfloor} (-1)^{k+1} \binom{N-2k}{k} k 2^{N-3k} \tag{B4}$$

Proof. (by induction) We want to show the following property

$$(h_N) \quad s_N = \sum_{k=1}^{\lfloor N/3 \rfloor} (-1)^{k+1} \binom{N-2k}{k} k 2^{N-3k}$$

Initialization : Using the formula we can check

1. $s_1 = 0$
2. $s_2 = 0$
3. $s_3 = 1$

Induction :
Suppose (h_N) is true for N, let's show (h_{N+1}) is also Using the recursive formula from Claim 3 we have

$$s_{N+1} = 2s_N - s_{N-2} + 2^{N-2} - e_{N-2}$$

by hypothesis on s_N

$$s_{N+1} = 2 \sum_{k=1}^{\lfloor N/3 \rfloor} (-1)^{k+1} \binom{N-2k}{k} k 2^{N-3k} \tag{B5}$$

$$- \sum_{k=1}^{\lfloor N-2/3 \rfloor} (-1)^{k+1} \binom{N-2(k+1)}{k} k 2^{N-3k-2} \tag{B6}$$

$$+ 2^{N-2} \tag{B7}$$

$$- \sum_{k=1}^{\lfloor (N-2)/3 \rfloor} (-1)^{k+1} \binom{N-2(k+1)}{k} 2^{N-3k-2} \tag{B8}$$

We rewrite line (1) as

$$\sum_{k=1}^{K} (-1)^{k+1} \binom{N-2k}{k} k 2^{N+1-3k}$$

$$= (N-2)2^{N-2} + \sum_{k=2}^{K} (-1)^{k+1} \binom{N-2k}{k} k 2^{N+1-3k}$$

adding with line (3) we get

$$(N-1)2^{N-2} + \sum_{k=2}^{K} (-1)^{k+1} \binom{N-2k}{k} k 2^{N+1-3k}$$

Let's call $K = \lfloor N/3 \rfloor$
We distinguish two cases:

1. $\lfloor (N+1)/3 \rfloor = K$
2. $\lfloor (N+1)/3 \rfloor = K+1$

Case 1:
Step 1: We add lines (2) and (4) together. Because $\lfloor (N+1)/3 \rfloor = K$ thus $\lfloor (N-2)/3 \rfloor$ has to be $K-1$.

$$-\sum_{k=1}^{K-1}(-1)^{k+1}\binom{N-2(k+1)}{k}2^{N-2-3k}(k+1)$$

Let $k' = k+1$ ($k = k'-1$) so the sum become

$$-\sum_{k'=2}^{K}(-1)^{k'}\binom{N-2k'}{k'-1}2^{N+1-3k'}k'$$

The term (-1) in front can be inserted to change the alternating term: (k' is written as k again for readability)

$$\sum_{k=2}^{K}(-1)^{k+1}\binom{N-2k}{k-1}2^{N+1-3k}k$$

Using Pascal formula for binomial coefficient we have $\binom{N-2k}{k-1} = \binom{N+1-2k}{k} - \binom{N-2k}{k}$

$$\sum_{k=2}^{K}(-1)^{k+1}\binom{N+1-2k}{k}2^{N+1-3k}k - \sum_{k=2}^{K}(-1)^{k+1}\binom{N-2k}{k}2^{N+1-3k}k$$

Step 2: We add everything together (1) + (2) + (3) + (4)

$$\begin{aligned}
&(N-1)2^{N-2}\\
&+\sum_{k=2}^{K}(-1)^{k+1}\binom{N-2k}{k}k2^{N+1-3k}\\
&+\sum_{k=2}^{K}(-1)^{k+1}\binom{N+1-2k}{k}2^{N+1-3k}k\\
&-\sum_{k=2}^{K}(-1)^{k+1}\binom{N-2k}{k}2^{N+1-3k}k
\end{aligned}$$

First and last lines contains the same sums so they cancel out

$$(N-1)2^{N-2}+\sum_{k=2}^{K}(-1)^{k+1}\binom{N+1-2k}{k}2^{N+1-3k}k$$

Step 3: Notice $(N-1)2^{N-2}$ is the term when $k=1$ of the right sum

$$(-1)^{1+1}\binom{N+1-2\cdot 1}{k}2^{N+1-3\cdot 1}\cdot 1 = (N-1)2^{N-2}$$

Step 4: In the end we get

$$\sum_{k=1}^{K}(-1)^{k+1}\binom{N+1-2k}{k}k2^{N+1-3k}$$

which conclude the induction.
Case 2:
Step 1: We add lines (2) and (4) together. Because $\lfloor (N+1)/3 \rfloor = K+1$ thus $\lfloor (N-2)/3 \rfloor$ has to be K.

$$-\sum_{k=1}^{K}(-1)^{k+1}\binom{N-2(k+1)}{k}2^{N-2-3k}(k+1) \quad (2)+(4)$$

Let $k' = k+1$ ($k = k'-1$) so the sum become

$$-\sum_{k'=2}^{K+1}(-1)^{k'}\binom{N-2k'}{k'-1}2^{N+1-3k'}k'$$

The term (-1) in front can be inserted to change the alternating term: (k' is written as k again for readability)

$$\sum_{k=2}^{K+1}(-1)^{k+1}\binom{N-2k}{k-1}2^{N+1-3k}k$$

Using Pascal formula for binomial coefficient we have $\binom{N-2k}{k-1} = \binom{N+1-2k}{k} - \binom{N-2k}{k}$

$$\sum_{k=2}^{K+1}(-1)^{k+1}\binom{N+1-2k}{k}2^{N+1-3k}k - \sum_{k=2}^{K+1}(-1)^{k+1}\binom{N-2k}{k}2^{N+1-3k}k$$

Step 2: We add everything together (1) + (2) + (3) + (4)

$$\begin{aligned}
&(N-1)2^{N-2}\\
&+\sum_{k=2}^{K}(-1)^{k+1}\binom{N-2k}{k}k2^{N+1-3k}\\
&+\sum_{k=2}^{K+1}(-1)^{k+1}\binom{N+1-2k}{k}2^{N+1-3k}k\\
&-\sum_{k=2}^{K+1}(-1)^{k+1}\binom{N-2k}{k}2^{N+1-3k}k
\end{aligned}$$

First and last lines contains the same sums except the last term so they partially cancel out

$$(N-1)2^{N-2}+\sum_{k=2}^{K+1}(-1)^{k+1}\binom{N+1-2k}{k}2^{N+1-3k}k+(-1)^{K+1}\binom{N-2(K+1)}{K+1}2^{N+1-3(K+1)}(K+1)$$

Step 3: Notice $(N-1)2^{N-2}$ is the term when $k=1$ of the right sum

$$(-1)^{1+1}\binom{N+1-2\cdot 1}{k}2^{N+1-3\cdot 1}\cdot 1 = (N-1)2^{N-2}$$

Notice as well that the coefficient binomial of the last term is $\binom{N-2K-2}{K+1}$ where $K = \lfloor (N-1)/3 \rfloor$. So by definition $N+1 = 3K+r$ where $0 \leq r \leq 2$. The upper term becomes $N-2K-2 = 3K+r-1-2K-2 = K+r-3$ because $r \leq 2$ we have $K+r-3 \leq K-1$. So we notice the upper term of the binomial coefficient is greater than the lower term because $N-2K-2 \leq K-1 \leq K+1$ so in the end $\binom{N-2K-2}{K+1} = 0$. Step 4: In the end we get

$$\sum_{k=1}^{K+1}(-1)^{k+1}\binom{N+1-2k}{k}k2^{N+1-3k}$$

which conclude the induction. □

2. Counting the number string with exactly L 00 in a string that contains a single 001

We define $Z_L = \{s \in S_N | s \text{ as exactly } L \text{ substrings equal } 00\}$. The goal is to compute the its cardinality $z_L = \#Z_L$.

Claim 5. There is an explicit expression for z_L given by

$$z_L = \sum_{\mathrm{k}=1}^{N-2}\left(\mathbf{f}_{\mathrm{k}-L+1}\mathbf{f}_{N-\mathrm{k}} + \sum_{\ell=2}^{L}\mathbf{f}_{\mathrm{k}-L+\ell}\mathbf{f}_{N-\ell-\mathrm{k}-1}\right) \tag{B9}$$

where $\mathbf{f}_k$ is the k-th term of the Fibonacci sequence (or 0 for negative k).

The following lemma is used for the proof of Claim 5

Lemma 3. Let $A_N = \{s \in \Sigma_N \text{ — } s \text{ does not contains any substrings equal to } 00\}$. Let $a_N = \#A_N$.
Then $a_N = \mathbf{f}_{N+2}$ where $\begin{cases} \mathbf{f}_n \text{ if } n \geq 1 \text{ is the } n\text{-th term of the fibonacci sequence.} \\ 0 \text{ else} \end{cases}$
In addition, let $A_N^* = \{s \in A_N \text{ — } s \text{ does not end by } 1\}$. Let $a_N^* = \#A_N^*$.
Then $a_N^* = \mathbf{f}_{N+1}$.

Proof. Let's define A_N^0 and A_N^1 as the subsets of A_N that ends respectively by a 0 and a 1. We call a_N^0 and a_N^1 their cardinality. We have $A_N = A_N^0 \cap A_N^1$ and $a_N = a_N^0 + a_N^1$.

Let's compute the first terms for each of them:

$$\begin{aligned} A_1 = \{0,1\} \quad A_1^0 = \{0\} \quad A_1^1 = \{1\} \\ A_2 = \{01,10,11\} \quad A_2^0 = \{10\} \quad A_2^1 = \{01,11\} \\ A_3 = \{010,011,101,110,111\} \quad A_3^0 = \{010,110\} \quad A_3^1 = \{101,011,111\} \end{aligned}$$

As well as the cardinality:

$$\begin{aligned} a_1 = 2 \quad a_1^0 = 1 \quad a_1^1 = 1 \\ a_2 = 3 \quad a_2^0 = 1 \quad a_2^2 = 2 \\ a_3 = 5 \quad a_3^0 = 2 \quad a_3^1 = 3 \end{aligned}$$

One can see the Fibonacci sequence appearing, let's prove it formally.

- For each $s \in A_N^0$, the $N+1$ length string can only end by 1 to avoid 00 substrings. Thus $a_{N+1}^1 += a_N^0$
- For each $s \in A_N^1$, the $N+1$ length string can end by both 0 and 1. Thus $a_{N+1}^1 += a_N^1$ and $a_{N+1}^0 += a_N^1$

All in all, this reads:

$$\begin{cases} a_{N+1}^1 = a_N^0 + a_N^1 \\ a_{N+1}^0 = a_N^1 \end{cases} \tag{B10}$$

Let's compute the term for $N+2$:

$$\begin{aligned} a_{N+2} &= a_{N+2}^0 + a_{N+2}^1 \\ &= a_{N+1}^1 + a_{N+1}^0 + a_{N+1}^1 \\ &= a_{N+1} + a_N^0 + a_N^1 \\ &= a_{N+1} + a_N \end{aligned}$$

and

$$\begin{aligned} a_{N+2}^* &= a_{N+2}^0 \\ &= a_{N+1}^1 \\ &= a_N^1 + a_N^0 \\ &= a_{N+1}^0 + a_N^0 \\ &= a_{N+1}^* + a_N^* \end{aligned}$$

So both (a_N) and (a_N^*) are described by Fibonacci sequences but with different first terms. Indeed, a_N start at $a_1 = 3$ so $a_N = \mathbf{f}_{N+2}$. On the other hand, a_N^* start at $a_1^* = 1$ then $a_2^* = 2$ so $a_N^* = \mathbf{f}_{N+1}$. □

Proof. of Claim 5 We start by computing the specific case of z_2. Because $s \in Z_2$ has a single substring equal 001, we can assume its starting position to be $1 \leq k \leq N$

$$s = \ldots\ldots \underset{\uparrow}{001} \ldots\ldots$$
$$k$$

Case 1:
Assume we have a zero in position $k-1$ so we have two 00 substrings.

$$s = \underbrace{\ldots\ldots}_{k-2} \underset{\uparrow}{0}\underset{k}{0}01 \underbrace{\ldots\ldots}_{N-(k+2)}$$

The string $s \in Z_2$ is of the form $s = w_1 \cdot 0001 \cdot w_2$ where:

- $w_1 \in A^*_{k-2}$
- $w_2 \in A_{N-(k+2)}$

So the number of all such strings s is $a^*_{k-2}a_{N-k-2} = \mathbf{f}_{k-1}\mathbf{f}_{N-k}$.
Case 2:
Assume we have two zeros starting in position $N-1$ so we have two 00 substrings in total.

$$s = \underbrace{\ldots\ldots}_{k-1} \underset{\uparrow}{0}\underset{k}{0}1 \underbrace{\ldots\ldots}_{N-2-(k+2)} 00$$

The string $s \in Z_2$ is of the form $s = w_1 \cdot 001 \cdot w_2 \cdot 00$ where:

- $w_1 \in A^*_{k-1}$
- $w_2 \in A^*_{N-k-4}$

So the number of all such strings s is $a^*_{k-1}a^*_{N-k-4} = \mathbf{f}_{k-1}\mathbf{f}_{N-k-3}$.
There are no other cases because s contains a single 001 so if any other places contains 00, following by a 1 bring to case 1 and following by 00 increase the number of 00 substrings by one, making it not belong to z_2. We just need to sum over all starting position k to get the expression of z_2:

$$z_2 = \sum_{k=1}^{N-2} \mathbf{f}_{k-1}\mathbf{f}_{N-k} + \mathbf{f}_k\mathbf{f}_{N-k-3} \tag{B11}$$

We now do the general case of z_L for a given L such that $2 \le L \le N-1$. As before, because $s \in Z_L$ has a single substring equals 001, we can assume its starting position to be $1 \le k \le N$

$$s = \ldots\ldots \underset{\uparrow}{0}\underset{k}{0}1 \ldots\ldots$$

We now reproduce the same two previous case and add intermediates ones.
Case 1:
Assume we have $L-1$ zeros in positions $k-L$ to $k-1$ so we have exactly L substrings equals 00.

$$s = \underbrace{\ldots}_{k-L}\underbrace{0\ldots0}_{L-1}\underset{\uparrow}{0}\underset{k}{0}1 \underbrace{\ldots\ldots}_{N-(k+2)}$$

The string $s \in Z_L$ is of the form $s = w_1 \cdot \underbrace{0\ldots0}_{L-1}001 \cdot w_2$ where:

- $w_1 \in A^*_{k-L}$
- $w_2 \in A_{N-(k+2)}$

So the number of all such strings s is $a^*_{k-L}a_{N-(k+2)} = \mathbf{f}_{k-L+1}\mathbf{f}_{N-k}$.
Case 2:
Assume we have L zeros starting in position $N-L+1$ so we have L substrings equals 00 in total.

$$s = \underbrace{\ldots\ldots}_{k-1} \underset{\uparrow}{0}\underset{k}{0}1 \underbrace{\ldots}_{N-L-k-2} \underbrace{0\ldots0}_{L}$$

The string $s \in Z_L$ is of the form $s = w_1 \cdot 001 \cdot w_2 \cdot \underbrace{0\ldots0}_{L}$ where:

- $w_1 \in A^*_{k-1}$
- $w_2 \in A^*_{N-L-k-2}$

So the number of all such strings s is $a^*_{k-1}a^*_{N-L-k-2} = \mathbf{f}_k\mathbf{f}_{N-L-k-1}$.
Intermediates Cases:
We can go incrementally from case 1 to case 2 by removing one zero at the head of the $\underbrace{0\dots0}_{L-1}$ string and adding one (except at the first time where we need two) at the end of the string s.

$$
\begin{array}{l}
\underbrace{\dots}_{k-L}\underbrace{0\dots0}_{L-1}\underset{\uparrow}{0}01\ \underbrace{\dots\dots}_{N-(k+2)} \\
\underbrace{\dots}_{k-L}\underbrace{0\dots0}_{L-2}\underset{\uparrow}{0}01\ \underbrace{\dots\dots}_{N-2-(k+2)}\ 00 \\
\underbrace{\dots}_{k-L}\underbrace{0\dots0}_{L-3}\underset{\uparrow}{0}01\ \underbrace{\dots\dots}_{N-3-(k+2)}\ 000 \\
\qquad\qquad\vdots \\
\underbrace{\dots\dots}_{k-L}\underset{\uparrow}{0}001\ \underbrace{\dots}_{N-(L-1)-k-2}\ \underbrace{0\dots0}_{L-1} \\
\underbrace{\dots\dots}_{k-1}\underset{\uparrow}{0}01\ \underbrace{\dots}_{N-L-k-2}\ \underbrace{0\dots0}_{L}
\end{array}
$$

we see the pattern being revealed, each time we remove one zero before 001 and add one more at the end. This can be written as the next expression:

$$\mathbf{f}_{\mathrm{k}-L+1}\mathbf{f}_{N-\mathrm{k}} + \sum_{\ell=2}^{L}\mathbf{f}_{\mathrm{k}-L+\ell}\mathbf{f}_{N-\ell-\mathrm{k}-1} \tag{B12}$$

where the first term in front reduces to Case 1 (it is a bit different to others since we add two zeros next), all the intermediate cases are computed until $\ell = L$ which is Case 2. Finally, we sum over all starting positions k which achieves the proof.

□

3. Computing the probability

Theorem 6. The probability P_N for the rule 001 of size $m = 3$ to realize after matching $m - 1 = 2$ bits 00 given that the strings of interest all contains exactly $q = 1$, a single times the rule is given by

$$P_N = \sum_{L=1}^{N-1}\frac{1}{L}\frac{z_L}{S_N} \tag{B13}$$

where the expression of S_N is given in Claim 4 and the one of z_L is given in Claim 5

This laborious development gives us in the end a literal expression for the computation of P_N. The function is strictly decreasing and we are able to check that for the string containing a single time 001 as a substring, the probability to find 001 after 00 is greater than $\frac{1}{2}$ for $N \leq 25$. The upper bound $\mathbf{N} = 25$ is expected to increase whenever we are interested to the string containing q times 001 for $q \geq 2$. This way, we have an intuition of the fact that a given rules has chance greater than $\frac{1}{2}$ to realize after matching $m - 1$ bits when we are guarantee that the string contains exactly q times the rule and N is not too large. This probability even increases for larger q.

Appendix C: Properties of the moment concentration

1. Proof of Theorem 1

Proof. [Moment concentration] First, one uses the Chebyshev's inequality (the distance of a sample to its mean) and gets

$$\mathbb{P}(|\hat{y}_n - \langle \hat{y}_n \rangle| \geq k\tilde{\sigma}) \leq \frac{1}{k^2}$$

where $k \in \mathbb{N}$ and $\tilde{\sigma}^2 = \mathrm{Var}(\hat{y}_n)$.
Next $\tilde{\sigma}$ and σ can be related as follows:

$$\begin{aligned}
\mathrm{Var}(\hat{y}_n) &= \mathrm{Var}[\frac{1}{n}\sum_i \Phi(f_{\mathcal{R}}(x_{(i)}))] \\
&= \frac{1}{n^2}\mathrm{Var}[\sum_i \Phi(f_{\mathcal{R}}(x_{(i)}))] \\
&= \frac{1}{n^2} \cdot n \cdot \mathrm{Var}[\Phi(f_{\mathcal{R}}(x_{(i)}))] \\
&= \frac{1}{n}\mathrm{Var}[\Phi(f_{\mathcal{R}}(x))] \\
&= \frac{\sigma^2}{n},
\end{aligned}$$

where we successively use the properties of the variance of the sum of independent random variables.

Using now the property of Monte-Carlo estimation, we have $\langle \hat{y}_n \rangle = y$. So we can rewrite the first expression as:

$$\mathbb{P}(|\hat{y}_n - y| \geq \frac{k\sigma}{\sqrt{n}}) \leq \frac{1}{k^2}.$$

Using the law of total probability, we can change the inequality into:

$$\mathbb{P}(|\hat{y}_n - y| \leq \frac{k\sigma}{\sqrt{n}}) \geq 1 - \frac{1}{k^2}.$$

Finally, let $\epsilon = \frac{k\sigma}{\sqrt{n}}$ and substitute k to get the desired result:

$$\mathbb{P}(|y - \hat{y}_n| \leq \epsilon) \geq 1 - \frac{\sigma^2}{\epsilon^2 n}.$$

□

2. Proof of Theorem 3

Lemma 4. [Concentration Bound with Thresholding]

Let $\sigma^{(t)^2} = \mathrm{Var}[\Phi(T_t f_{\mathcal{R}}(x))]$ be the variance of the thresholded function. The following expression holds:

$$\sigma^{(t)^2} = p(t)\sigma_1^2(t) + \mu_1^2(t)(1 - p(t)p(t)) \tag{C1}$$

where $p(t)$ is the probability to sample from the non-zero region of $T_t f_{\mathcal{R}}$ after thresholding, μ_1 and σ_1^2 are respectively the mean and variance of this non-zero region.

Proof. To express $\sigma^{(t)^2}$ we use the law of total variance. For a given t, let's divide the space of $\Phi(T_t f_{\mathcal{R}}(x))$ into its zero part ($G = 0$) and its non-zero part ($G = 1$). The law of total variance allows us to write the variance $\mathrm{Var}[\Phi(T_t f_{\mathcal{R}}(x))]$ (which we will call $\mathrm{Var}(X)$ for convenience) as follows:

$$\mathrm{Var}(X) = \mathbb{E}[\mathrm{Var}[X|G]] + \mathrm{Var}[\mathbb{E}[X|G]],$$

where G is our grouping variable separating the zero part from the non-zero part. Assume

- $\mu_0 = 0, \quad \sigma_0 = 0$
- $\mu_1, \quad \sigma_1$

are the mean and variance of respectively the zero and non-zero parts. The first term of $\mathrm{Var}(X)$ (within group variance) turns into $\mathbb{E}[\mathrm{Var}[X|G]] = p_1\sigma_1^2$.

The second term (between group variance) can be computed as well: we have $\mu = p_0\mu_0 + p_1\mu_1$ for the total mean

$$\begin{aligned}\mathrm{Var}[\mathbb{E}[X|G]] &= p_0(\mu_0 - \mu) + p_1(\mu_1 - \mu) \\ &= p_0(\mu_0 - \mu) + p_1(\mu_1 - \mu) \\ &= p_0\mu_1^2 p_1^2 + p_1\mu_1^2(1-p_1)^2 \\ &= \mu_1^2 p_0 p_1.\end{aligned}$$

Finally we get:

$$\sigma^{(t)} = \sqrt{\mathrm{Var}(x)} = \sqrt{p(t)\sigma_1^2(t) + \mu_1^2(t)(1 - p(t)p(t)},$$

□

We now prove the Theorem 2

Proof. [Threshold increases probability] (intuition behind the proof)

Let's show that the theorem holds for some nontrivial case with moderate t. First, we notice that the terms in Lemma 4 do not depend on permutations of the domain of $f_{\mathcal{R}}$, therefore for this analysis we can reorder the function according to decreasing values. Let's assume that $f_{\mathcal{R}}(x)$ can be approximated by an exponential law $f_{\mathcal{R}}(x) = \lambda e^{-\lambda x}$ (after reordering the values). Therefore, we can approximate each terms from lemma 4:

$$p(t) = \frac{-\log(t/\lambda)}{\lambda}$$

$$\begin{aligned}\mu_1(t) &= \int_0^{p(t)} x \cdot \lambda e^{-\lambda x} dx \\ &= \frac{1}{\lambda} - \frac{t}{\lambda^2} + \frac{t\log(t/\lambda)}{\lambda^2}\end{aligned}$$

$$\begin{aligned}\sigma_1(t) &= \int_0^{p(t)} x^2 \cdot \lambda e^{-\lambda x} - \mu^2 \\ &= \frac{(\lambda^2 - t^2 + 2t^2\log[t/\lambda] - t(\lambda + t)\log[t/\lambda]^2)}{\lambda^4}\end{aligned}$$

Putting everything together in Lemma 4, we obtain an explicit expression for the value $\sigma^{(t)^2}$. For a fixed λ, the function in t increases before reaching a maximum and then decreases. One can also check $\lim_{t\to\infty} \sigma^{(t)^2} = 0$. Finally, since the function $f_{\mathcal{R}}$ is continuous, the Intermediate Value Theorem tells us that there exists a value of t for which $\sigma^{(t)^2} \leq \sigma^2$. This proves equation 11.

Now consider $k \in \mathbb{R}^+$ such that $1 - \frac{1}{k^2} \geq 1 - \frac{\sigma^2}{\epsilon^2 n}$. From Theorem 1 the moment computed from the thresholded function read as:

$$\mathbb{P}(\hat{y}_n \in \mathcal{B}_\epsilon) \geq 1 - \frac{\sigma^{(t)^2}}{\epsilon^2 n}$$

Hence we want:

$$1 - \frac{\sigma^{(t)^2}}{\epsilon^2 n} \geq 1 - \frac{1}{k^2} \geq 1 - \frac{\sigma^2}{\epsilon^2 n} \tag{C2}$$

We focus on

$$\begin{aligned} 1 - \frac{{\sigma^{(t)}}^2}{\epsilon^2 n} &\geq 1 - \frac{1}{k^2} \\ \frac{{\sigma^{(t)}}^2}{\epsilon^2 n} &\leq \frac{1}{k^2} \\ {\sigma^{(t)}}^2 &\leq \frac{\epsilon^2 n}{k^2} \end{aligned}$$

Through the same reasoning as equation 11, there exist a t that fit this scenario. And so the same t works as well for equation C2 which certify equation 12 and achieves the proof.

□

Appendix D: Properties of the transform and distinguishability

1. Proof of Lemma 1

The following claim is prerequisite to show Lemma 1.

Claim 6.

$$a \neq 0^{(k)} \implies F_N(a)[i_a] = 1 \tag{D1}$$

Proof. The proof is by contradiction.
$a = a_1 \cdots a_k$ matches with $i_a = 0^{(N-k)} a_1 \cdots a_k$ at the last k bits. If we assume $F_N(a)[i_a] > 1$, then, a matches with i_a at somewhere else, say $a_1 \cdots a_k = 0^{(l)} a_1 \cdots a_{k-l}, (l \in \mathbb{Z}^+, 1 \leq l \leq k-1)$. Then we get $a = 0^{(k)}$ by solving

$$\begin{aligned} a_1 &= 0, \\ &\vdots \\ a_l &= 0, \\ a_{l+1} &= a_1, \\ &\vdots \\ a_k &= a_{k-l}. \end{aligned}$$

This contradicts with $a \neq 0^{(k)}$. □

We now prove the Lemma 1

Proof. [Injectivity] The proof that $a = b \implies F_N(a) = F_N(b)$: is trivial from the definition of F_N. Thus we only need to prove $F_N(a) = F_N(b) \implies a = b$. To do so, we use the contraposition that is, we want to prove that $a \neq b \Longrightarrow F_N(a) \neq F_N(b)$.
Set

$$\begin{aligned} &a = a_1 \cdots a_k, (a_i \in \{0,1\}, i = 1, \cdots k), \\ &b = b_1 \cdots b_k, (b \in \{0,1\}, i = 1, \cdots k), \\ &0^{(m)} = \underbrace{0 \cdots 0}_{m}, \\ &i_a = 0^{(N-k)} a_1 \cdots a_k, \\ &i_b = 0^{(N-k)} b_1 \cdots b_k, \end{aligned}$$

and $F_N(a)[i]$ where $i \in \{0,1\}^n$ denotes the i-th element of the vector $F_N(a)$.

Case i) When $a = 0^{(k)}$
$F_N(a)[0^{(N)}] = N - k + 1$, by the definition of F_N. $F_N(b)[0^{(N)}] = 0$, since $b \neq a = 0^{(k)}$. Thus, $F_N(a) \neq F_N(b)$.

Case ii) When $a \neq 0^{(k)}$
Let $b \neq 0^{(k)}$, since the case of $b = 0^{(k)}$ can be proved completely the same as the case of $a = 0^{(k)}$. We prove by contradiction. Assume $F_N(a) = F_N(b)$. This gives

$$F_N(a)[i_a] = F_N(b)[i_a], \tag{D2}$$
$$F_N(a)[i_b] = F_N(b)[i_b]. \tag{D3}$$

From Claim 6 and equation (D2), $F_N(b)[i_a] = 1$. Thus, for an integer $k_1,\ 1 \leq k_1 \leq k-1$,

$$b_1 \cdots b_k = 0^{(k_1)} a_1 \cdots a_{k-k_1} \tag{D4}$$

since $0 \neq b \neq a$.
From Claim 6 and equation (D3), $F_N(a)[i_b] = 1$. Thus, for an integer $k_2,\ 1 \leq k_2 \leq k-1$,

$$\begin{aligned} a_1 \cdots a_k &= 0^{(k_2)} b_1 \cdots b_{k-k_2} \\ &= \begin{cases} 0^{(k_2)} 0^{(k_1)} a_1 \cdots a_{k-k_1-k_2} & \text{if } k - k_2 \geq k_1 + 1 \\ 0^{(k_2)} 0^{(k-k_2)} & \text{if } k - k_2 \leq k_1 \end{cases} \end{aligned}$$

by equation (D4) and $0 \neq a \neq b$. By considering the same way in the proof of Claim 6, a will be $0^{(k)}$ in both cases. This contradicts with $a \neq 0^{(k)}$. Therefore, $F_N(a) \neq F_N(b)$. □

2. Proof of Lemma 2

Proof.

$$\begin{aligned} &\|F_N(a) - F_N(b)\|_2^2 \\ =&F_N(a) \cdot F_N(a) + F_N(b) \cdot F_N(b) - 2F_N(a) \cdot F_N(b) \end{aligned}$$

Thus, we will analyze $F_N(a) \cdot F_N(b)$. From

$$F_{k+l}(a) = \sum_{i=1}^{l+1} \left(\mathbf{1}_2^{\otimes(i-1)} \otimes |a\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i)} \right),$$

the inner product becomes

$$\begin{aligned} &F_{k+l}(a) \cdot F_{k+l}(b) \\ =&\sum_{i=1}^{l+1} \left(\mathbf{1}_2^{\otimes(i-1)} \otimes |a\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i)} \right)^{\mathsf{T}} \sum_{i'=1}^{l+1} \left(\mathbf{1}_2^{\otimes(i'-1)} \otimes |b\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i')} \right) \\ =&\sum_{i=i'} \left(\mathbf{1}_2^{\otimes(i-1)} \otimes |a\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i)} \right)^{\mathsf{T}} \left(\mathbf{1}_2^{\otimes(i'-1)} \otimes |b\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i')} \right) \\ +&\sum_{i \neq i'} \left(\mathbf{1}_2^{\otimes(i-1)} \otimes |a\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i)} \right)^{\mathsf{T}} \left(\mathbf{1}_2^{\otimes(i'-1)} \otimes |b\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i')} \right). \end{aligned}$$

For the first term,

$$\begin{aligned} &\sum_{i=i'} \left(\mathbf{1}_2^{\otimes(i-1)} \otimes |a\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i)} \right)^{\mathsf{T}} \left(\mathbf{1}_2^{\otimes(i'-1)} \otimes |b\rangle \otimes \mathbf{1}_2^{\otimes(l+1-i')} \right) \\ =&\sum_{i=1}^{l+1} \left(\mathbf{1}_2^{\otimes(i-1)\mathsf{T}} \mathbf{1}_2^{\otimes(i-1)} \right) \langle a|b\rangle \left(\mathbf{1}_2^{\otimes(l+1-i)\mathsf{T}} \mathbf{1}_2^{\otimes(l+1-i)} \right) \\ =&\begin{cases} 0 & (a \neq b) \\ (l+1)2^l & (a = b) \end{cases}, \end{aligned}$$

since $\langle a|b\rangle = \delta_{a,b}$.
For the second term with $a \neq b$, the positions of $|a\rangle$ and $|b\rangle$ differ, so the calculation of the inner product will not be easy as the case of the first term. Here I assume that $l \geq k$ and will divide the second term into two cases; (i) when the positions of $|a\rangle$ and $|b\rangle$ do not overlap, and (ii) when they overlap. The case of $l < k$ can be regarded as only considering Case (ii).
Case i): when the positions of $|a\rangle$ and $|b\rangle$ do not overlap
It can be illustrated as $(l-k+1)$ cases shown below (there are $\mathbf{1}_2$s in the blank places):

$$
\begin{aligned}
&a_1a_2\cdots a_{k-1}a_k \\
&\qquad\qquad b_1b_2\cdots b_{k-1}b_k, \\
&a_1a_2\cdots a_{k-1}a_k \\
&\qquad\qquad \underbrace{\quad}_{1} b_1b_2\cdots b_{k-1}b_k, \\
&a_1a_2\cdots a_{k-1}a_k \\
&\qquad\qquad \underbrace{\quad\quad}_{2} b_1b_2\cdots b_{k-1}b_k, \\
&\qquad\qquad \vdots \\
&\qquad\qquad \vdots \\
&a_1a_2\cdots a_{k-1}a_k \\
&\qquad\qquad \underbrace{\quad\quad\quad}_{l-k} b_1b_2\cdots b_{k-1}b_k.
\end{aligned}
$$

Here I showed the case when a_k is in front of b_1. The opposite case (when b_k is in front of a_1) can be considered in the completely same way. For all of them, the inner product will be 2^{l-k}, because the inner product of $|a_i\rangle$ (or $|b_i\rangle$) and $\mathbf{1}_2$ will be 1 for all $1 \leq i \leq k$ (there are $2k$ of this inner products), and the inner product of $\mathbf{1}_2$ and $\mathbf{1}_2$ will be 2 (there are $(l-k)$ of this inner product). When there is g-length gap between the positions of a_k and b_1, there are $(l-k-g+1)$ options for the position of a_1. Thus, the sum of inner products is

$$
2 \cdot 2^{l-k} \sum_{g=0}^{l-k} (l-k-g+1) = 2^{l-k}(l-k+1)(l-k+2).
$$

Case ii): when the position of $|a\rangle$ overlaps with the position of $|b\rangle$
These cases will be illustrated as below; when a_1 is in front of b_1 (Case ii-1)

$$
\begin{aligned}
&a_1a_2\cdots a_{k-1}a_k \\
&\quad b_1b_2\cdots \quad b_{k-1}b_k, \\
&a_1a_2a_3\cdots a_{k-1}a_k \\
&\qquad b_1b_2\cdots \quad b_{k-2}b_{k-1}b_k, \\
&\vdots \\
&\vdots \\
&a_1a_2\cdots a_{k-1}a_k \\
&\qquad\qquad b_1b_2\cdots b_{k-1}b_k,
\end{aligned}
$$

and when b_1 is in front of a_1 (Case ii-2)

$$\begin{array}{l} \quad a_1a_2 \quad \cdots \quad a_{k-1}a_k \\ b_1b_2\cdots \quad b_{k-1}b_k, \end{array}$$

$$\begin{array}{l} \quad a_1a_2a_3\cdots a_{k-2}a_{k-1}a_k \\ b_1b_2b_3 \quad \cdots \quad b_k, \end{array}$$

$$\vdots$$

$$\vdots$$

$$\begin{array}{l} \quad a_1a_2\cdots a_{k-1}a_k \\ b_1b_2\cdots b_{k-1}b_k. \end{array}$$

For both (Case ii-1) and (Case ii-2) the jth pattern represents the case when the positions of a_1 and b_1 are j-separated. For j-separated case, the sum of the inner product is

$$\begin{cases} (l-j+1)2^{l-j} & \text{when overlapping part of } a \text{ and } b \text{ is same,} \\ 0 & \text{else.} \end{cases}$$

This value depends on a and b. We will calculate the maximum value of the inner product of this part. The task is to find the possible combination which gives the maximum inner product value from the below table.

j	Case ii-1	Case ii-2
1	$l2^{l-1}$	$l2^{l-1}$
2	$(l-1)2^{l-2}$	$(l-1)2^{l-2}$
$\vdots$	$\vdots$	$\vdots$
$k-1$	$(l-k+2)2^{l-k+1}$	$(l-k+2)2^{l-k+1}$

TABLE IV: Maximum inner product value for each separation value j

First, choose $l2^{l-1}$ and $l2^{l-1}$. This is because

$$\begin{aligned} &(l-1)2^{l-2}+\cdots+(l-k+2)2^{l-k+1} \\ =&\sum_{j=2}^{k-1}(l-j+1)2^{l-j} \\ =&2^{l-1}\left[l-2-\left(\frac{1}{2}\right)^{k-2}(k+l+2)\right] \\ \leq&2^{l-1}l, \end{aligned}$$

so you get more than half of the possible inner product value by choosing them. If you choose these two, $a_2\cdots a_k = b_1\cdots b_{k-1}$ and $a_1\cdots a_{k-1} = b_2\cdots b_k$ have to be simultaneously held. You can easily see that this condition gives

$$\begin{aligned} a &= 01010101\cdots \\ b &= 10101010\cdots \end{aligned}$$

or the opposite. This also satisfies the conditions for j: odd. You can see it by sliding a (or b) to the right for j: odd positions. If you try to satisfy any of the condition for j: even for the above a and b, it will immediately gives you $a = b = 0000\cdots$ (or $1111\cdots$).

Finally, the maximum inner product value for case (ii) is

$$\begin{aligned}
&2\sum_{j:\text{ odd}}(l-j+1)2^{l-j}\\
&=\begin{cases}2\sum_{m=1}^{n}\left[l-(2m-1)+1\right]2^{l-(2m-1)} & (k=2n)\\ 2\sum_{m=1}^{n-1}\left[l-(2m-1)+1\right]2^{l-(2m-1)} & (k=2n-1)\end{cases}\\
&=\begin{cases}\frac{2^{l+2}}{9}\left[3l-2+\frac{6n-3l+2}{4^n}\right]\\ \frac{2^{l+2}}{9}\left[3l-2+\frac{6n-3l-4}{4^{n-1}}\right]\end{cases}\\
&=\begin{cases}\frac{2^{l+2}}{9}\left[3l-2+\frac{3k-3l+2}{2^k}\right] & (k:\text{even})\\ \frac{2^{l+2}}{9}\left[3l-2+\frac{3k-3l-1}{2^{k-1}}\right] & (k:\text{odd})\end{cases}
\end{aligned}$$

For the second term with $a=b$, the sum of inner product value will be the sum of

$$2^{l-k}(l-k+1)(l-k+2)$$

which is from Case (i) and

$$2\sum_{j=1}^{k-1}(l-j+1)2^{l-j}=2^l\left[2l-2-\left(\frac{1}{2}\right)^{k-2}(k+l+2)\right]$$

which is the sum of all the values in TABLEIV from Case (ii). We also observe that $F_N(a)\cdot F_N(a)$ does not depend on a. Thus, we can rewrite $\|F_N(a)-F_N(b)\|_2^2$ as

$$\|F_N(a)-F_N(b)\|_2^2=2[F_N(a)\cdot F_N(a)-F_N(a)\cdot F_N(b)]$$

Therefore we have to check the difference between $F_N(a)\cdot F_N(a)$ and $F_N(a)\cdot F_N(b)$. For the first term, the difference is $(l+1)2^l$. For the second term, the difference appears in Case (ii), and the minimum difference is

$$\begin{aligned}
&2^l\left[2l-2-\left(\frac{1}{2}\right)^{k-2}(k+l+2)\right]\\
&-\begin{cases}\frac{2^{l+2}}{9}\left[3l-2+\frac{3k-3l+2}{2^k}\right] & (k:\text{even})\\ \frac{2^{l+2}}{9}\left[3l-2+\frac{3k-3l-1}{2^{k-1}}\right] & (k:\text{odd})\end{cases}
\end{aligned}$$

Add these differences and double it, and we get Theorem2. □

Appendix E: Additional properties

Definition 1. [ℓ_1-Norm of the transform]

$$\|F_N(a)\|_1=\sum_{x\in\{0,1\}^N}\sum_{i=1}^{N-k+1}\delta_{h(x_i,a),0}. \tag{E1}$$

We need the following for the proof of Lemma 5

Claim 7.

$$\begin{aligned}
F_{N+1}(a)&=\begin{pmatrix}F_N(a)\\ F_N(a)\end{pmatrix}+\mathbf{a}\otimes\mathbf{1_{2^{N-k+1}}}\\
&=\mathbf{1}_2\otimes F_N(a)+\mathbf{a}\otimes\mathbf{1_{2^{N-k+1}}},
\end{aligned}$$

where $\otimes$ is the tensor product, $\mathbf{1}_n$ denotes n length column vector with all elements 1, and $\mathbf{a}$ is the 2^k-length column vector of which a-th element is 1 and the rest is 0.

Proof. An $(N+1)$-bit string $x^{(N+1)} \in \{0,1\}^{N+1}$ can be made by putting 0 or 1 in front of an N-bit string $x^{(N)} \in \{0,1\}^N$. We can divide $\{0,1\}^N$ into two sets: A set X which consists of N-bit strings which start with $(k-1)$ bits matching the $(k-1)$ last bits of a, i.e. $a_2 \cdots a_k$. Its complement set $\bar{X}$ made of the remaining N-bit strings. Put a_1 in front of $x^{(N)} \in X$ and generate $(N+1)$-bit string $a_1 x^{(N)}$. Then the number of k-bit strings in $a_1 x^{(N)}$ that matches with a increases by one from the number of k-bit strings in $x^{(N)}$ that matches with a, because $x^{(N)} \in X$ starts with $a_2 \cdots a_k$. In other cases, i.e.,

- putting $1 - a_1$ in front of $x^{(n)} \in X$ and
- putting a_1 or $1-_1^a$ in front of $x^{(n)} \in \bar{X}$,

the number of k-bit strings that matches with a does not change. Writing this explicitly,

$$F_{N+1}(a)[yx^{(N)}] = \begin{cases} F_N(a)[x^{(N)}] + 1 & \text{if } y = a_1 \wedge x^{(N)} \in X, \\ F_N(a)[x^{(N)}] & \text{else.} \end{cases}$$

Therefore, we can calculate $F_{N+1}(a)$ by the following procedure;

1. Concatenate $F_N(a)$ and $F_N(a)$,
2. Add 1 to the elements of which indexes start from k-bit string a.

The 2^{N+1} length vector indicating the position of $(N+1)$-bit strings which start from a can be written as $\mathbf{a} \otimes \mathbf{1_{2^{N-k+1}}}$, since there are 2^{N+1-k} strings like that. Thus, the equation in Claim 7 holds. □

Lemma 5. [ℓ_1-Norm Recursive Expression]

$$\|F_{N+1}(a)\|_1 = 2\|F_N(a)\|_1 + 2^{N-k+1}. \tag{E2}$$

Proof. By taking the ℓ_1-norm on each side of the equation from Claim 7, it can easily be shown. (both vectors on the right hand side of the equation are vectors with non-negative elements.) □

Lemma 6. [ℓ_1-Norm Explicit Expression]

$$\|F_{k+l}(a)\|_1 = 2^l(l+1). \tag{E3}$$

Notice that all elements a have the same norm!

Proof. By substituting N in Lemma 5 to $k+l$,

$$\|F_{k+l+1}(a)\|_1 = 2\|F_{k+l}(a)\|_1 + 2^{l+1}.$$

Set $\|F_{k+l}(a)\|_1 = s_l$. Then,

$$\begin{aligned} s_{l+1} &= 2s_l + 2^{l+1}, \\ \frac{s_{l+1}}{2^{l+1}} &= \frac{s_l}{2^l} + 1. \end{aligned}$$

Thus,

$$\begin{aligned} \frac{s_l}{2^l} &= \frac{s_0}{2^0} + l, \\ \frac{s_l}{2^l} &= 1 + l, \quad (\because s_0 = \|F_k(a)\|_1 = \|\mathbf{a}\|_\mathbf{1} = \mathbf{1}) \\ \|F_{k+l}(a)\|_1 &= s_l \\ &= 2^l(l+1). \end{aligned}$$

□

Lemma 7. [Distance expression] The distance between the transform of two different rules a and b of length k can be expressed as follow:

$$\|F_N(a) - F_N(b)\|_2 = 2\Big[\|F_N(a)\|_2 - \mathcal{C}_N(a,b)\Big], \tag{E4}$$

where

$$\mathcal{C}_N(a,b) = \sum_{x \in \mathrm{supp}(F_N(a)) \cap \mathrm{supp}(F_N(b))} \min(F_N(a)[x], F_N(b)[x])$$

is the number of common elements of both vectors. (It is the number of times both strings a and b appear as a sub-string of x for all x.)

Proof. We want to compute $\|F_n(a) - F_N(b)\|_2$, using the polarization identity for ℓ_2-distance we have :

$$\|F_N(a) - F_N(b)\|_2 = \|F_N(a)\|_2 + \|F_N(b)\|_2 - 2F_N(a) \cdot F_N(b).$$

Then we use the fact that all rule have the same ℓ_2-norm to get

$$\|F_N(a) - F_N(b)\|_2 = 2\Big[\|F_N(a)\|_2 - F_N(a) \cdot F_N(b)\Big] \tag{E5}$$

Finally, $F_N(a) \cdot F_N(b) = \sum_x (F_N(a)[x], F_N(b)[x])$ is non-zero iff $x \in \mathrm{supp}(F_N(a)) \cap \mathrm{supp}(F_N(b))$. □