Learning Complexity of Simulated Annealing

Avrim Blum
TTIC

Chen Dan
CMU

Saeed Seddighin
TTIC

Abstract

Simulated annealing is an effective and general means of optimization. It is in fact inspired by metallurgy, where the temperature of a material determines its behavior in thermodynamics. Likewise, in simulated annealing, the actions that the algorithm takes depend entirely on the value of a variable which captures the notion of temperature. Typically, simulated annealing starts with a high temperature, which makes the algorithm pretty unpredictable, and gradually cools the temperature down to become more stable.

A key component that plays a crucial role in the performance of simulated annealing is the criteria under which the temperature changes, namely, *the cooling schedule*. Motivated by this, we study the following question in this work: "*Given enough samples to the instances of a specific class of optimization problems, can we design near-optimal cooling schedules that minimize the runtime or maximize the success rate of the algorithm on average when the underlying problem is drawn uniformly at random from the same class?*"

We provide positive results both in terms of *sample complexity* and *simulation complexity* [1]. For sample complexity, we show that $\widetilde{O}(\sqrt{m})$ samples suffice to find an approximately optimal cooling schedule of length m. We complement this result by giving a lower bound of $\widetilde{\Omega}(m^{1/3})$ on the sample complexity of any learning algorithm that provides an almost optimal cooling schedule. These results are general and rely on no assumption. For simulation complexity, however, we make additional assumptions to measure the success rate of an algorithm. To this end, we introduce *the monotone stationary graph* that models the performance of simulated annealing. Based on this model, we present polynomial time algorithms with provable guarantees for the learning problem.

1 Introduction

The goal of this work is to better understand how we can design efficient simulated annealing (SA) algorithms. Simulated annealing is a well-known heuristic method to tackle hard problems. Term annealing originates from thermodynamics, referring to the way that metals cool and anneal. Instead of the energy of the material, simulated annealing utilizes the objective function of an optimization problem. Surprisingly, the implementation of SA is very simple as it is very similar to hill-climbing. The only difference is that instead of picking the best move in every step, simulated annealing picks a random move. If the selected move improves the quality of the solution, then the move is always accepted. Otherwise, the algorithm makes the move anyway with some probability less than 1. The probability decreases exponentially with the *badness* of the move, which is the amount by which the solution is worsened. This is shown by $\Delta(E)$. One example of the annealing criteria is given below:

$$\Pr[\text{accepting downhill move at step } i] \simeq 1-e^{\Delta(E)/t_i}.$$

where parameter t_i is the temperature of the algorithm at step i which is used to determine this probability. The t_i parameter is analogous to temper-

[1] We call the overall runtime of the algorithm that determines the cooling schedule the simulation complexity

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

Sample complexity		
Upper bound:	$\widetilde{O}(\sqrt{m})$	
Lower bound: (for our discretization approach)	$\widetilde{\Omega}(\sqrt{m})$	
Lower bound: (for any learning algorithm)	$\widetilde{\Omega}(m^{1/3})$	
Simulation complexity		
identical paths	separate paths	separate paths + all-satisfied
exact solution in time poly$(m, n, \lvert T\rvert)$	exact solution in time poly$(m, n, \lvert T\rvert^n)$	$(O(\log n\lvert T\rvert), 0)$ approximation in time poly$(m, n, \lvert T\rvert)$

Table 1: Overview of our results. Here, m denote the length of cooling schedule. In the computational results, T is the set of the discretized temperatures and n is the number of samples used in the learning algorithm.

ature in an annealing system at time step i. At higher values of temperature, downhill moves are more likely to occur. As the temperature tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. In a typical SA optimization, the temperature starts at a high value and is gradually decreased according to a cooling schedule. Simulated annealing is used for a broad class of computational problems ranging from SAT to travelling salesman problem, to VLSI routing, etc. as experiments strongly support the efficiency of simulated annealing in practice Aragon et al. (1984); Kirkpatrick et al. (1983); Ingber (1993); Nieto-Vesperinas et al. (1988), and still achieving empirical success in various applications recently Mafarja and Mirjalili (2017); Fathollahi-Fard et al. (2019); Wang et al. (2019); Ezugwu et al. (2017); Lu et al. (2017); Matejka and Fitzmaurice (2017).

Indeed the efficiency of an SA algorithm significantly depends on its cooling schedule Lam and Delosme (1988b,a); Nourani and Andresen (1998); Kirkpatrick et al. (1983); Sacco (1990); Nulton and Salamon (1988); Triki et al. (2005); Azizi and Zolfaghari (2004); Aarts and Korst (1988). One simple cooling schedule is to start with a single temperature t_0 and decrease the temperature linearly with a rate of α to obtain lower temperatures gradually. We use this simple cooling strategy to present illustrating examples, nonetheless we consider a more generalized setting in this work. The literature has also gone beyond simple cooling schedules and several non-linear methods have been proposed so far Lam and Delosme (1988b,a); Nourani and Andresen (1998); Kirkpatrick et al. (1983); Sacco (1990); Nulton and Salamon (1988); Triki et al. (2005); Azizi and Zolfaghari (2004); Aarts and Korst (1988). It is not hard to imagine that even for different instances of the same problem, the optimal cooling schedules may vary significantly.

In our setting, we focus on a family of problem instances (available via sample access to an unknown distribution) and tend to maximize the average score over all instances. This approach is known as the PAC-style modeling which has been used to analyze a variety of other application-specific algorithms, including branch and bound algorithms, center-based/linkage-based clustering, and combinatorial auctionsGupta and Roughgarden (2017); Balcan et al. (2017, 2018, 2019a). We refer the readers to Roughgarden (2020) for a more comprehensive review of this area. Generally in algorithm design, we need to incorporate an unknown set of instances, since otherwise for any given instance there is an algorithm that has the solution memorized. Typically one does this worst-case over an instance family, and with this analysis we are at least aiming to bring that closer to the specific instances by performing near-optimally with respect to the actual distribution over instances at hand.

Therefore in this work, we take a learning approach towards designing simulated annealing algorithms, using the PAC-style model for data-driven algorithm design introduced in Gupta and Roughgarden (2017) and used to analyze a wide range of important families of algorithms and heuristics in Balcan et al. (2017, 2018, 2019a, 2020). In brief, we consider a distribution $\mathcal{D}$ over a specific class of instances of a presumably hard problem (such as SAT) and aim to design near-optimal cooling schedules for such instances, analyzing both sample complexity (the number of instances from $\mathcal{D}$ we need to observe) and simulation complexity (runtime) needed for learning.

Our approach is particularly motivated by the work of Balcan et al. (2017).

2 Prelimnaries

As aforementioned, an SA algorithm makes a random walk on the nodes of a search graph. Each node of this graph represents a potential (not necessarily optimal) solution for the underlying problem and the energy of a node is a value reflecting how close this solution is to an optimal solution. We assume that for each node, its energy and neighbors are available via oracle queries. One thing to keep in mind is that the number of nodes in this huge search graph may be exponentially large and that we only have local views on the nodes of the graph. For instance, when the underlying problem is SAT, we may have 2^k nodes where k is the number of variables in the SAT problem and each node represent an assignment of true/false to the variables.

Crucial to any cooling schedule are the parameters that maximize its performance. This could be as simple as just a real value specifying the cooling rate or as complicated as a sequence of variables determining the exact value of the temperature at every step. Take for instance, the simplest case in which a parameter t_0 and linear cooling rate α formulate the temperature at every step. In this case, at step i, $t_i = t_0(1 - \alpha i)$ formulates the temperature. Therefore, the learning algorithm has to find the optimal pair (t_0, α) that maximizes efficiency. It is an easy exercise to see that the learning problem is actually not very challenging in this case. Although this simple formulation involves infinitely many (t_0, α) pairs that need to be searched over, via careful discretization techniques, one can narrow down the set of possible (t_0, α) pairs to polynomially many candidates and iterate over them to find the optimal cooling schedule[2]. Samples are used to determine how well each cooling schedule performs in practice. More precisely, samples are used to approximate the score of a cooling schedule.

However, we go beyond linear cooling schedules and include more sophisticated systems (*i.e.,* non-linear cooling schedules). Our setting is pretty general: we denote the cooling schedule by a vector $\mathcal{E} = \langle t_1, t_2, \ldots, t_m \rangle$ where m is the number of steps our algorithm takes and t_i specifies the temperature at time i. Any non-increasing sequence of values makes a valid cooling schedule. The problem becomes more challenging with this representation; Even after discretizing the temperatures, still there are exponentially many cooling schedules and determining an approximately optimal schedule is non-trivial.

Recall that each node of the search graph corresponds to a potential solution for the underlying problem. In case of SAT for instance, each node can be an assignment of the true/false values to the variables. We label a subset of nodes in the search graph as *acceptable solution nodes*. These nodes correspond to solutions that are acceptable for the underlying problem. In the case of SAT, a node whose corresponding solution satisfies all of the clauses is a solution node. The score of an SA algorithm with a specific cooling schedule is the likelihood of reaching an acceptable solution node after a fixed number of steps. We would like to point out that although a reasonable energy function for the nodes of the search graph gives higher energies to the acceptable solution nodes, we make no particular assumption on the energies in our setting. We remark that if the cooling schedule is available, it is computationally easy to evaluate the score of the algorithm. We run the SA algorithm according to the cooling schedule and once it terminates we find out if the solution found by the algorithm is acceptable for the problem. By repeating this process enough times, we can estimate the score of the cooling schedule very accurately.

Indeed the optimal parameters may vary for different problems or even for different instances of the same problem and therefore we need to also incorporate the problem instances in our setting. To illustrate the importance of the cooling schedule, consider the example shown in Figure 2. In the example shown in Figure 2, there is one solution node (colored in red) which has an energy of $3n$ and the search graph consists of a clique of vertices with distinct energies one of whose vertices has a path to the solution node. The energies of the nodes of this path are increasing. Let us assume that the initial state of the algorithm is the node colored in green. It is easy to verify that an extreme strategy that never accepts downhill moves has zero chance of reaching the solution node and another extreme strategy that always accepts all downhill moves requires a cubic number of steps to reach the solution node. However, a strategy that accepts each downhill move with probability $1/2$ only requires $\mathcal{O}(n^2)$ steps in expectation to reach the solution node.

Motivated by this example, we define our general problem in the following way:

[2]This improvement comes with a small error to the quality of the solution.

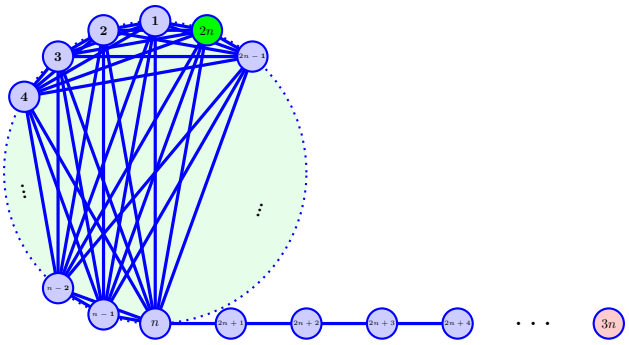

Figure 1: A search graph with $3n$ nodes is illustrated in this figure. The numbers on the nodes show their energy or in other words goodness of the nodes. In this example, we consider the red node to be the only solution of the problem.

Problem Let $\mathcal{D}$ be a distribution over a specific class of instances[3] of a hard problem (such as SAT). Denote the set of valid (combination of) parameters for the SA algorithm by $\mathcal{F} = \{\mathcal{E}_1, \mathcal{E}_2, \dots, \}$. Moreover, let for an instance $\mathsf{I} \sim \mathcal{D}$ and a set of parameters $\mathcal{E} \in \mathcal{F}$, $\mathsf{score}(\mathsf{I}, \mathcal{E})$ be a function that reflects how well an SA algorithm with parameters $\mathcal{E}$ works on instance I. This is basically the likelihood of finding a solution, if our SA algorithm uses $\mathcal{E}$ as its cooling schedule. Our goal is to find a set of parameters $\mathcal{E} \in \mathcal{F}$ that maximizes efficiency. In other words,

$$\mathbb{E}_{\mathsf{I} \sim \mathcal{D}}[\mathsf{score}(\mathsf{I}, \mathcal{E})]$$

is (approximately) maximized. We assume throughout this paper that the scores improve as energy *increases*. Also, a downhill move is a move which hurts the energy of a node and thus is accepted with some probability smaller than 1. However, whenever the score of a node does not hurt in a move, such a move is always made.

We clarify the notation by a simple example. Let us go back to the basic setting in which we formulate the temperature at each step with a pair (t_0, α). In this case, $\mathcal{F} = \mathbb{R}^+ \times (0, 1/m)$ would be the set of all valid parameters. Moreover, a natural example for score is the probability of finding a correct solution after a given (say m) number of steps. This way, the problem is to find a temperature t_0 and a cooling rate α that maximize the success probability after performing m moves of SA. Our attention in this work is focused on an even more general setting. We denote the cooling schedule by a sequence of non-increasing temperatures $\langle t_1, t_2, \dots, t_m \rangle$ for a fixed m. Thus, in our setting we have $\mathcal{F} \subseteq (\mathbb{R}^+)^m$ subject to the temperatures being non-increasing. For simplicity, and without loss of generality, we assume that the energies of the nodes are integer numbers in range $\{1, 2, \dots, \mathsf{e}_{\max}\}$.

Any SA algorithm basically makes a random walk on a search graph in which every node represents a potential/partial solution for the problem. For instance, when the underlying problem is SAT, every node of the search graph is a true/false assignment to the variables of the program. The energy of each node is a local guess on how well that solution satisfies the goals of the problem. For the case of SAT for instance, one simple energy function for a node is the number of clauses the corresponding solution satisfies. In our setting, we make no assumption on the energy of the nodes though in practice we expect that a higher energy signals a better solution. Since an SA algorithm makes a random walk, its state at every step can be shown via a distribution over the nodes of the search graph. Initially, this distribution shows the likelihood of each node being used as the starting solution and as the algorithm proceeds, the distribution changes based on the criteria of the random walk. The final state of the algorithm represents the likelihood of each node reported as the final solution. Thus, we wish the final distribution of our algorithm to be highly concentrated on the solution nodes.

We evaluate our learning algorithm based on two quantities: *sample complexity* and *simulation complexity*[4]. The former measures the number of samples one needs in order to find an (approximately) optimal cooling schedule and the latter measures the runtime of the learning algorithm in order to find an optimal cooling schedule.

Our main results are concerned with the sample complexity of the learning problem. As a typical challenge for learning problems, we have to face the issue that the space of the problem is infinitely large as there are infinitely many cooling schedules for an SA algorithm. In order to prove a bound on the sample complexity, the first step is to show that by losing a small additive error, we can bound the space of the solutions to a finite set. We begin by explaining this in Section 3. We also propose a computational model for evaluating the simulation complexity of the problem and design efficient algorithms with theoreteical guarantees in Section 4.

[3] For instance industrial instances of SAT.

[4] This is equivalent to the notion of running time if we assume that our SA procedure halts after a polynomial number of steps.

3 Sample Complexity

In this section, we give an analysis for the sample complexity of the problem. Recall that, for any problem instance I and any sequence of m temperatures $\mathcal{E} = \langle t_1, t_2, \ldots, t_m \rangle$, we define $\mathsf{score}(\mathsf{I}, \mathcal{E})$ to be the probability of finding an acceptable solution of I using temperatures in $\mathcal{E}$. We say $\mathcal{E}$ is ε-approximately optimal, if $\mathbb{E}_{\mathsf{I}\sim\mathcal{D}}\mathsf{score}(\mathsf{I}, \mathcal{E}) \geq \sup_{\mathcal{E}'} \mathbb{E}_{\mathsf{I}\sim\mathcal{D}}\mathsf{score}(\mathsf{I}, \mathcal{E}') - \varepsilon$. That is, no other cooling schedule of the same length can achieve a significantly higher success rate. Our goal is to provide upper and lower bounds on the number of i.i.d. samples from $\mathcal{D}$ required for learning an ε-approximately optimal cooling schedule.

3.1 Warm-Up: A Crude Upper Bound

We first show that although the space of the problem is infinitely large, only a polynomial number of samples suffice to approximate the optimal solution within desirable guarantees. This step is quite classic as discretization is the typical approach to bound the solution set. One of the difficulties in finding near-optimal cooling schemes is that there are infinitely many options available. We show that by discretizing the temperatures into $\widetilde{O}(m/\varepsilon)$ different values, we only lose an additive error of ε in the success rate when running the algorithm on any instance of the problem. Note that, we are not making any assumptions yet: we only rely on the fact that the algorithm is evaluated based on the success rate. Discretizing the temperature makes designing efficient algorithms possible too as we will show in Section 4. Our main result is an upper bound of $\widetilde{O}(\sqrt{m})$ for the sample complexity which is explained in details later in Section 3.3. Here we start as a warm-up by giving an upper bound of $\widetilde{O}(m)$.

Theorem 3.1. *The sample complexity of computing an ε-approximately optimal cooling schedule with length m is bounded by $O\left(\varepsilon^{-2}\left(m\log(\frac{m\mathsf{e}_{max}}{\varepsilon})\right)\right)$.*

Roughly speaking, the total number of samples we need in order to approximate the optimal cooling schedule is logarithmic in terms of the number of *candidate solutions* we have. Initially, the space of cooling schedules is infinitely large, however, a discretization technique can reduce the space of candidate solutions to $2^{\widetilde{O}(m)}$ many. More precisely, we define a discretized temperature set T whose size is $\widetilde{O}(m)$ and show that there is an almost optimal solution that only uses the temperatures in T. This reduces the space of candidate solutions to $(O(m))^m \leq 2^{\widetilde{O}(m)}$ which implies that the sample complexity is bounded by $\widetilde{O}(m)$.

The only non-trivial part of the above analysis is to show that a discretized set of temperatures with size $\widetilde{O}(m)$ is enough to approximate the optimal cooling schedule within an arbitrarily small additive error. Let us fix an $\epsilon > 0$ and assume that the goal is to construct a discretized set of temperatures T such that there is a cooling schedule that only uses the temperatures of T and its score is at most ϵ smaller than the optimal solution. One convenient way to construct such a set is to make sure for each $t > 0$ there is a $t' \in T$ such that for any $1 \leq x \leq \mathsf{e}_{\max}$ we have $|e^{-x/t} - e^{-x/t'}| \leq \epsilon/m$. Then we can imply that if we replace every temperature t_i of the optimal solution with its corresponding t'_i of the discretized set, each step we make a different decision with probability at most ϵ/m and thus the total error is bounded by ϵ. That is, with probability $1-\epsilon$ our algorithm traverses the exact same path as had we not modified the optimal cooling schedule. It is not hard to prove that such a condition can be met by having $O(m\log \mathsf{e}_{\max})$ elements in $|T|$ which gives us an almost linear bound on the sample complexity.

Up to this point we show that an almost linear number of queries is sufficient for approximating an optimal cooling schedule. This raises two questions: i) Can we improve the bound such that the dependence on m is subpolynomial? In particular, do polylogarithmically many samples suffice for our purpose? ii) If the answer to the first question is negative, can we prove a linear lower bound on the sample complexity? As we show in the following, the answer to both questions is negative!

3.2 A Polynomial Lower Bound

We present a negative answer to the first question above. Although this section gives us a lower bound, our improved upper bound in the next section is actually inspired by this lower bound. The first attempt to prove a lower bound is to understand the limit of the discretization technique explained above. Therefore, we ask the following question: "assuming that our algorithm first constructs a discretized set of temperatures and then seeks to find an optimal solution that only uses the discretized temperatures, how many samples do we need?" Indeed, the answer to this question does not imply a lower bound in general, but it does give us an insight into the problem which leads to a general lower bound.

To answer the above question, we need to understand what is the smallest set T of temperatures that can be used to make a cooling schedule whose score is very close to the optimal solution? The search graph shown in Figure 2 proves that $|T|$ should be at least as large as $\widetilde{\Omega}(\sqrt{m})$, otherwise the guarantee may not hold.

In the search graph of Figure 2, we set $m' = m/100$. For a fixed τ, we set x in a way that $e^{-x/\tau} = 1/2$, that is if the temperature is equal to τ the probability of making a downhill move is exactly equal to $1/2$[5]. The goal of this search graph is to start the SA algorithm from the initial node and the only acceptable solution node is the final node.

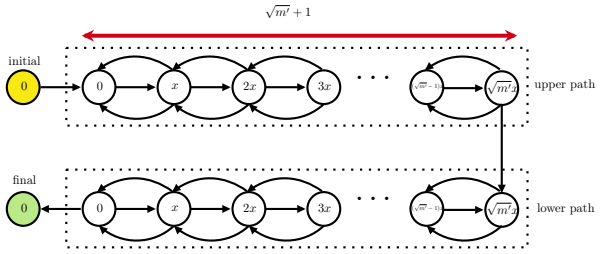

Figure 2: The search graph is depicted for a fixed temperature τ. $x = \tau \ln 0.5$ is chosen in a way that $e^{-x/\tau} = 1/2$ holds.

The search graph of Figure 2 is particularly interesting because of the following observations: i) A cooling schedule of length m only having temperature τ is guaranteed to reach the final node with high probability. ii) A cooling schedule of length m that does not contain any temperature in range $[\tau(1 - \widetilde{\Omega}(m^{-1/2})), \tau(1 + \widetilde{\Omega}(m^{-1/2}))]$ has very little chance to reach the final node. As a consequence, if the multiplicative distance between two consecutive temperatures in our discretized set is more than $1+\widetilde{\Omega}(m^{-1/2})$, one can delicately design such a search graph for which our discretization performs poorly while the optimal solution gets a score close to 1. This implies that the size of the discretized set has to be at least $\widetilde{\Omega}(\sqrt{m})$ to prove a bound.

While the above argument shows that our specific algorithm definitely needs $\widetilde{O}(\sqrt{m})$ samples[6], it does not give a lower bound for general algorithms beyond our discretization approach. However, we show in Section B with a slightly more advanced analysis that any algorithm requires at least $\widetilde{\Omega}(m^{1/3})$ samples from the distribution in order to guarantee a non-trivial bound. While the heart of the proof is based on the same search graph, in order to extend the observation to all algorithms, we slightly lose on the exponent of m in the lower bound.

Theorem 3.2. *Even if* $\mathsf{e}_{max} = 2^{\widetilde{\Theta}(1)}$*, any learning algorithm requires at least* $\widetilde{\Omega}(m^{1/3})$ *samples from the distribution in order to obtain an additive error less than* 0.5*.*

Before proceeding to the main result ($\widetilde{O}(\sqrt{m})$ sample complexity upper bound), we would like to note an implication of this result in the context of simulated annealing. There have been several attempts in the literature to understand the complexity of simulated annealing. One question asked in the literature from both theoretical and practical standpoints is if there is a meaningful difference between Simulated Annealing and the Metropolis Algorithm Wegener (2005); Hajek (1988). Metropolis is a special case of simulated annealing where the temperature does not change by time. That is the cooling schedule repeats a single temperature m times. While this observation was made previously, our lower bound also implies that (from a theoretical standpoint) there is a meaningful difference between the two algorithms as Metropolis can be learned with much fewer samples which shows there are cases for which simulated annealing performs much better. Another example is when the temperature drops linearly for which the sample complexity is small. More generally, this lower bound actually shows a gap between SA and any special case of SA whose cooling schedule has complexity smaller than $m^{1/3}$. For instance, it shows that an extended version of Metropolis that uses $m^{1/3-\epsilon}$ many different temperatures in the cooling schedule is not competitive with the general SA algorithm.

3.3 Tightening the Upper Bound

Perhaps the more surprising result of this paper is that the sample complexity can be improved to $\widetilde{O}(\sqrt{m})$:

Theorem 3.3. *The sample complexity of computing an* ε*-approximately optimal cooling schedule with length* m *is bounded by* $O(\varepsilon^{-2}\sqrt{m}\log(\frac{m\mathsf{e}_{max}}{\varepsilon}))$*.*

Our algorithm is almost identical to the one explained in Section 3.1 except that we construct a smaller set T whose size is bounded by $\widetilde{O}(\sqrt{m})$. Then we argue that the total number of cooling

[5]For now, we assume x can be an arbitrary real number but this comes without loss of generality.

[6]See the proof of Theorem 3.3 for more details.

schedules with this temprature set is bounded by $2^{\widetilde{O}(\sqrt{m})}$ which leads to sample complexity $\widetilde{O}(\sqrt{m})$.

Below we provide a sketched version of the proof. A complete version can be found in appendix. The first pointer to this result is that there is no clear way to improve the lower bound of Section 3.2. Keep in mind that for the lower bound, we construct a search graph for which a particular cooling schedule works well, but if we multiply (or divide) each temperature by a small factor $1+\widetilde{\Omega}(m^{-1/2})$, the score of the algorithm drops significantly. Obviously, if one comes up with a better search graph for which a multiplicative factor of $1+\widetilde{\Omega}(m^{-1/2-\epsilon})$ breaks the solution, then it shows that it is impossible to obtain an upper bound of $\widetilde{O}(\sqrt{m})$ with the discretization technique. Failure to make a better bad instance brings us to the possibility that maybe massaging each temperature by a multiplicative factor of $1+\widetilde{O}(m^{-1/2})$ cannot hurt the score of the cooling schedule significantly. We show that this is indeed the case!

Recall that in Section 3.1, in order to prove that the discretized cooling schedules perform almost optimally, we show that there is a discretized cooling schedule that behaves the same as the optimal cooling schedule with probability $1-\epsilon$. That is, in the unlikely event of making a different decision (we call it a *mistake*) we give 0 credit to our discretized cooling schedule, yet we prove that the score is pretty close to that of the optimal. Clearly, this is a loose upper bound as we do not expect to lose too much by making a single mistake.

We illustrate the idea with a toy problem. Consider a complete binary tree of depth m. The root has depth 0 and the leaves have depth m. Each leaf is attributed to a score which is either 0 or 1. The score of each non-leaf node is the average of the scores of its children. In other words, if we make a random walk towards the leaves with equal probability of going to each child, the score of a node is equal to the probability of reaching a leaf with score 1 using the random-walk. Let us call this *even-random-walk* and consider a different type of random-walk, namely *uneven-random-walk*. The uneven-random-walk is pretty much the same as the even random walk, except that at some depth i uniformly drawn from $[1, m]$, an adversary may change the decision of which child to go to. The toy-problem is to understand how much the score of a node hurts by replacing even-random-walk by uneven-random-walk.

To study this, we attribute to each node a *deviation value* which is equal to the absolute value of the difference between the scores of it children. This roughly captures an upper bound of the score we lose, if we traverse the edges of that node with a different criteria (other than $1/2, 1/2$). Thus, we need to know what is the average deviation values of the nodes in an even-random-walk? This roughly tells us how much we lose in the score, if an adversary changes the criteria of the walk at some random point!

The upper bound on the answer is $O(1/\sqrt{m})$ no matter how the leaves are scored. It goes beyond the scope of this paper, but we mention the idea in the hope that it helps a mindful reader decipher some of steps that we take in the proof of Lemma C.1. Define a deviation function $f(x) : [0,1] \to [0, 0.25] = x - x^2$. One can show by induction that starting from each node v of depth i, the average sum of deviations in a random walk is bounded by $O((f(s_v) + (m-i)/m)\sqrt{m})$ where s_v is the score of node v (obtained via even-random-walk).

The toy problem illustrates that in the event that our optimal solution makes decisions with probability $1/2, 1/2$ (which is indeed the case for our lower bound), we can afford to make $O(\sqrt{m})$ mistakes and not lose much in the average score. This does not hold if the decisions are made with different probabilities. To see this, consider the case that only the rightmost leaf has a score 1 and the rest of the leaves have scores 0. Moreover, the probability of going to the right child in the random walk is $1-\epsilon/m$ and the probability of going to the left child is ϵ/m. In this case, the average deviation is $\Omega(1)$ when we start from the root and make a random walk according to the probabilities.

The next observation is that when the decisions are not necessarily $1/2, 1/2$ say $p, 1-p$, multiplying the temperature by a factor of $1+x$ changes the probabilities by at most $\max\{\ln 1/p, 1\} \min\{p, 1-p\}x$ (see Observation C.5). That is, as the probabilities deviate from $1/2$, the probability of making a "mistake" drops linearly. More precisely, the multiplicative term $\min\{p, 1-p\}$ gives us extra power to deal with these situations. For instance, if $p < 1/\sqrt{m}$ or $p > 1 - 1/\sqrt{m}$ then the probabilities change by an additive error of $\widetilde{O}(1/m)$ when we multiply the temperature by a factor of $1+\widetilde{O}(m^{-1/2})$. This error is tolerable since we can afford to have an error of ϵ/m for each decision we make.

The proof is based on the above ideas but the analysis is quite involved and rather cryptic by nature. We show in Section C that if the temperatures in the discretized set are at most $1 + \widetilde{O}(m^{-1/2})$ away from each other (multiplicative), then one can make

a cooling schedule by the discretized temperatures whose score is arbitrarily close to that of the optimal solution. This then can be used to obtain an upper bound of $\widetilde{O}(\sqrt{m})$ on the sample complexity of the problem.

4 Simulation Complexity

The second part of the paper is concerned with the computational aspects of the learning problem. Although we prove that the sample complexity is polynomial without any assumptions, it seems that extra assumptions are necessary for the runtime concerns. Notice that we make no assumption on the underlying problem and the only information available to us when we sample an instance of the problem is a huge search graph containing exponentially many vertices. Even if we bring the underling problem into the setting, it is not clear how we can make use of the conditions of a problem such as SAT to find the right cooling schedule. Keep in mind that the complexity of the underlying problem is the reason we use simulated annealing in the first place. Therefore, we introduce a stylized model to make the problem more tractable. We call our model *the monotone stationary graph*. Although the model relies on extra assumptions, it features nice properties that make it particularly suitable for our purpose.

First, it gives a compact representation for every instance of the problem. Up to this point, we treated each problem instance as a huge search graph with exponentially many vertices which is too big to store in the memory let alone optimizing the solution over it. Our model represents the search graphs in a more efficient way. Next, notice that even if we fix a well-defined representation for a search graph, one should be able to recover the new representation of a problem instance without spending too much time (and of course without taking a complete look at the already exponentially large search graph). Our model makes it possible to recover the stationary graph in polynomial time. Finally, the any model used for our problem has to give us enough structure so that finding an approximately optimal cooling schedule becomes polynomially tractable in the new setting. This is the most important feature of our model.

In our model, we represent each instance of the problem as a graph. Vertices of this graph correspond to the temperatures in our discretized set. Intuitively, for a temperature $t \in T$, its corresponding vertex in the graph represent the state of an SA algorithm that runs infinitely many steps with temperature t. Thus, when the state of our algorithm is close to such a stationary distribution, we assume that our algorithm is pointing at the corresponding vertex in the monotone stationary graph. We draw edges between the vertices to specify how many steps we need to take in the SA algorithm to move between the stationary distributions. Since in our model, the state of an algorithm can be approximated with a node in this graph, we can also determine its score by examining the corresponding stationary distribution.

Therefore, given n instances of the underlying problem, our goal is to find a cooling schedule that obtains the highest average score for these instances by our model. We consider the following three settings and provide a solution for each one of them:

(i). identical-paths: in this setting, we assume that the optimal cooling schedule traverses the same path for all n instances.

(ii). separate-paths: in this setting, we allow the optimal solution to use different paths for different instances.

(iii). separate-paths + all-satisfied: This is a special case of the second setting where we know that there exists a cooling schedule of length m that is optimal for all instances and brings us to the last node for each monotone stationary graph.

We refer the readers to Section D for the details of the computational model. To obtain polynomial time solutions, we introduce the notion of an (α, ϵ)-approximate cooling schedule. In such a solution we allow the cooling schedule to violate the size constraint by a factor of α with the promise that its score is no more than ϵ smaller than the score of the optimal cooling schedule of length m. With this notation, we present computational results below (also summarized in Table 1):

(i). identical-paths: This is the simplest one among the three settings - we achieve an *exact* algorithm which runs in $\mathsf{poly}(m, n, |T|)$ and maximizes the average score for the n instance.

(ii). separate-paths: in this setting, we again achieve an *exact* algorithm which runs in $\mathsf{poly}(m, |T|)$ for any fixed n. However, the runtime is exponential in n.

(iii). separate-paths + all-satisfied: To overcome the exponential dependency in n, we design an efficient approximation algorithm which runs in $\mathsf{poly}(m, n, |T|)$, but achieve an $O(\log(n|T|), 0)$ approximation instead of exact solution in the

previous settings. The algorithm is based on LP relaxation of an integer program.

Due to the space limitations, we refer the readers to Section E in the appendix for the details of the algorithm and analysis.

5 Conclusion

In this paper, we proposed a PAC-Learning framework for estimating a near optimal cooling schedule in simulated annealing. We provided non-trivial upper and lower bounds on the sample complexity. Our techniques may also be relevant to other problems with a random walk on search space. We also introduced the monotone stationary graph model for which we are able to find a near-optimal cooling schedule in polynomial time based on rounding linear programs.

We conclude with two open questions due to the limitations of this work. First, finding near optimal cooling schedules in poly-time (beyond our monotone stationary graph model) is an important question in both theory and practice. Our statistical results showed that the sample complexity of learning a good schedule is polynomial in m, however, the analysis is based on a discretization over the search space which has a size exponential in m and it's unclear how to design an efficient algorithm based on the large discretization. This is, in fact, a common issue in the data-driven algorithm design community: the sample complexity scales logarithmically with the search space, while the search space may be exponentially large. See e.g. Page 5 of Balcan et al. (2019b), and Theorem 17 of Balcan et al. (2017) where their sample complexity is $O(n)$, but computational complexity is $\Omega(n^2 3^{2n})$. In certain simpler settings, it's possible to polynomially upper bound the search space and get efficient algorithms, e.g. it was shown in Balcan et al. (2019b) that learning a single hyperparameter in linkage-based clustering can be solved in $O(n^8)$, but even in their setting the algorithm is based on enumeration. Since simulated annealing is a very complicated problem and it's unlikely to have efficient learning algorithm for the general setting, we seek to find approximation algorithms under additional assumptions and restrictions in the monotone stationary graph model. Designing efficient algorithms beyond this model is definitely an important future direction, and perhaps of interest broadly to the data-driven algorithm design community as well.

Second, designing a better energy function for simulated annealing remains an open question. Throughout the paper, we assumed that the energy function is given apriori, and our goal is to learn a cooling schedule using the given energy function so that the score of learned schedule is at least $\text{OPT} - \varepsilon$. While the sample complexity bounds derived in this paper depend on $\mathrm{e}_{\max}$, which varies from different choices of energy function, we believe that the choice of energy function plays a more important role, as it decides the optimal score OPT among all possible cooling schedules $\mathcal{E}$. Consequently, we should expect OPT to be smaller (and empirical performance to be better) when energy function is chosen carefully. Since we are only considering additive approximation to OPT, the value of OPT itself has no effect on the sample complexity, but it is definitely a useful quantity in practice. Thus, finding a better energy function (with a smaller OPT) is an interesting direction for future works.

Acknowledgement We thank the annonymous reviewers for many helpful suggestions for improving the exposition of this work. Part of this work was done when CD and SS were visiting students at TTIC. This work was supported in part by the National Science Foundation under grants CCF-1733556, CCF-1800317, and CCF-1815011.

References

Aarts, E. and Korst, J. (1988). Simulated annealing and boltzmann machines.

Aragon, C., Johnson, D., McGeoch, L., and Schevon, C. (1984). Simulated annealing performance studies. In *Workshop on Statistical Physics in Engineering and Biology*, pages 865–892.

Azizi, N. and Zolfaghari, S. (2004). Adaptive temperature control for simulated annealing: a comparative study. *Computers & Operations Research*, 31(14):2439–2451.

Balcan, M.-F., DeBlasio, D., Dick, T., Kingsford, C., Sandholm, T., and Vitercik, E. (2019a). How much data is sufficient to learn high-performing algorithms? *arXiv preprint arXiv:1908.02894*.

Balcan, M.-F., Dick, T., and Lang, M. (2019b). Learning to link. *arXiv preprint arXiv:1907.00533*.

Balcan, M.-F., Dick, T., and Lang, M. (2020). Learning to link. In *ICLR*.

Balcan, M.-F., Dick, T., Sandholm, T., and Vitercik, E. (2018). Learning to branch. In *International Conference on Machine Learning*, pages 353–362.

Balcan, M.-F., Nagarajan, V., Vitercik, E., and White, C. (2017). Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In *Conference on Learning Theory*, pages 213–274.

Ezugwu, A. E.-S., Adewumi, A. O., and Frîncu, M. E. (2017). Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem. *Expert Systems with Applications*, 77:189–210.

Fathollahi-Fard, A. M., Govindan, K., Hajiaghaei-Keshteli, M., and Ahmadi, A. (2019). A green home health care supply chain: New modified simulated annealing algorithms. *Journal of Cleaner Production*, 240:118200.

Gupta, R. and Roughgarden, T. (2017). A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017.

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of operations research*, 13(2):311–329.

Ingber, L. (1993). Simulated annealing: Practice versus theory. *Mathematical and computer modelling*, 18(11):29–57.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Lam, J. and Delosme, J.-M. (1988a). An efficient simulated annealing schedule: derivation. *Yale University, New Haven, Connecticut, Technical Report*, 8816.

Lam, J. and Delosme, J.-M. (1988b). Performance of a new annealing schedule. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pages 306–311. IEEE Computer Society Press.

Lu, F., Bi, H., Huang, M., and Duan, S. (2017). Simulated annealing genetic algorithm based schedule risk management of it outsourcing project. *Mathematical Problems in Engineering*, 2017.

Mafarja, M. M. and Mirjalili, S. (2017). Hybrid whale optimization algorithm with simulated annealing for feature selection. *Neurocomputing*, 260:302–312.

Matejka, J. and Fitzmaurice, G. (2017). Same stats, different graphs: generating datasets with varied appearance and identical statistics through simulated annealing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 1290–1294.

Nieto-Vesperinas, M., Navarro, R., and Fuentes, F. (1988). Performance of a simulated-annealing algorithm for phase retrieval. *JOSA A*, 5(1):30–38.

Nourani, Y. and Andresen, B. (1998). A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373.

Nulton, J. D. and Salamon, P. (1988). Statistical mechanics of combinatorial optimization. *Physical Review A*, 37(4):1351.

Roughgarden, T. (2020). *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press.

Sacco, M. (1990). Stochastic relaxation, gibbs distributions and bayesian restoration of images.

Triki, E., Collette, Y., and Siarry, P. (2005). A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. *European Journal of Operational Research*, 166(1):77–92.

Wang, L., Goh, M., Ding, R., and Pretorius, L. (2019). Improved simulated annealing based risk interaction network model for project risk response decisions. *Decision Support Systems*, 122:113062.

Wegener, I. (2005). Simulated annealing beats metropolis in combinatorial optimization. In *International Colloquium on Automata, Languages, and Programming*, pages 589–601. Springer.