Self-Improving Algorithms *

Nir Ailon† Bernard Chazelle† Seshadhri Comandur† Ding Liu†

Abstract

We investigate ways in which an algorithm can improve its expected performance by fine-tuning itself automatically with respect to an *arbitrary, unknown* input distribution. We give such *self-improving* algorithms for sorting and clustering. The highlights of this work: (i) a sorting algorithm with optimal expected limiting running time; and (ii) a k-median algorithm over the Hamming cube with linear expected limiting running time. In all cases, the algorithm begins with a learning phase during which it adjusts itself to the input distribution (typically in a logarithmic number of rounds), followed by a stationary regime in which the algorithm settles to its optimized incarnation.

1 Introduction

The classical approach to analyzing algorithms draws a familiar litany of complaints: Worst-case bounds are too pessimistic in practice, say the critics, while average-case complexity too often rests on unrealistic assumptions. The charges are not without merit. It's hard enough to argue that the only permutations we ever want to sort are random; it's a different level of implausibility altogether to pretend that the sites of a Voronoi diagram should always follow a Poisson process or that ray tracing in a BSP tree should be spawned by a Gaussian. Efforts have been made to analyze algorithms under more complex models (eg, Gaussian mixtures, Markov model outputs) but with limited success and lingering doubts about the choice of priors.

Ideally, one would like to compute a function f with the help of a *self-improving* algorithm. Upon receiving its first input instance I_0, such an algorithm would compute $f(I_0)$ with, say, good worst-case guarantees and nothing more. Think of newly installed software that knows nothing about the user and runs in its "vanilla" configuration. Subsequently, as it is called upon to compute $f(I_k)$ for $k = 1, 2, \ldots$, the algorithm would gradually improve its performance through automatic finetuning. Intuitively, if the I_k's are drawn from a low-entropy distribution, the algorithm should be able to spot that and *learn* to be more efficient.

The obvious analogy is data compression, which seeks to exploit low entropy to minimize encoding size. The analogue of Shannon's noiseless coding theorem would be here: Given an *unknown* distribution $\mathcal{D}$, design a self-improving algorithm that converges to one with optimal expected running time. The second goal, which is to optimize the convergence speed, is more strictly speaking of a machine learning nature. One of the surprises to us—but perhaps not to machine learning experts—is how fairly naive distribution learning suffices for dramatic self-improvement.

The starting point of this work is the observation that, trimmed of noise, real-world data is often of much lower entropy than size alone suggests. For example, Takens' embedding theorem asserts that univariate time series obtained from deterministic dynamical systems can be geometrized *canonically* as a (usually) low-dimensional attractor set in finite-dimensional space [45]. Hidden Markov models for speech recognition can be remarkably effective with only a few thousand states. Anecdotal evidence can also be gleaned from the current trend toward personalization in the design of web tools (search engines, recommendation systems, etc). Input data is often lodged in a tiny slice of input space that cannot be captured by closed-form distributions. To make predictions about the slice is the essence of machine learning [16, 28, 39]. To take computational advantage of the slice is what self-improving algorithms are all about. There is an intriguing connection with *online learning*, and several of our algorithms can, indeed, be interpreted as prediction from expert advice [11, 12, 14, 18, 20, 27, 29, 33, 34, 36]. This connection will be explored in the full paper.

Our Results The performance of a self-improving algorithm is measured with respect to an unknown memoryless random source $\mathcal{D}$ of input instances. (See discussion of the model's merits below). The algorithm is given instances $I_0, I_1, \ldots$, which it must solve one at a time in batch mode with: (1) no prior knowledge of future instances; and (2) no a priori knowledge about $\mathcal{D}$. The algorithm may store auxiliary information to help improve its performance. (Unlike self-organizing data structures, however, none of that information should be

*This work was supported in part by NSF grants CCR-998817, 0306283, ARO Grant DAAH04-96-1-0181.

†Dept. Comp. Sci., Princeton University, { nailon, chazelle, csesha, dingliu }@cs.princeton.edu

necessary for the algorithm to complete its task.)

Our first result is, in some sense, the first truly optimal sorter. While the prospect of "beating $n \log n$" might excite only true-blue sorting devotees, the topic is the ideal introduction to the concept of self-improvement. First, some notation. Given a source $\mathcal{D}$ of numbers $\{x_1, \ldots, x_n\}$ to sort, let $\mathcal{D}_<$ be the distribution over the symmetric group induced by the ranks of the x_i's (using the indices i to break ties). The complexity of our algorithm depends on the entropy $H(\mathcal{D}_<)$, which, we happily note, can be much smaller than the entropy of the source (while, of course, never exceeding it).

- Sorting: We give a self-improving algorithm with a limiting running time[1] of $O(H(\mathcal{D}_<)+n)$ and prove that it is optimal. If the input $\{x_1, \ldots, x_n\}$ to be sorted is obtained by drawing each x_i independently from some distribution $\mathcal{D}_i$, then the storage is $O(n^{1+\varepsilon})$, for any fixed $\varepsilon > 0$. The storage is optimal in the worst case. We also show that independence is necessary: Without it, the storage *must* be exponential in n.

Our second result addresses a classical NP-hard optimization problem.

- Clustering: We consider the k-median problem over the d-dimensional Hamming cube. Assuming a distribution of n points, each one drawn independently from its own unknown (arbitrary) random source, we give a self-improving $(1+\varepsilon)$-approximation algorithm that runs in $O(dn)$ limiting running time and $O(d)$ space. Errors occur with arbitrarily small probability. The algorithm can be made Monte Carlo (ie, with error probability independent of the input) with some extra storage *independent* of n.

Previous Work Related concepts have been studied before. List accessing algorithms and splay trees are textbook examples of how simple updating rules can speed up searching with respect to an adversarial request sequence [1, 13, 25, 43, 44]. It is interesting to note that self-organizing data structures were investigated over stochastic input models first [2, 10, 21, 35, 41, 42]. It was the observation [9] that memoryless sources for list accessing are not terribly realistic that partly motivated work on the adversarial models. It is highly plausible that both approaches are superseded by more sophisticated stochastic models: for example, hidden Markov models for gene finding or speech recognition or time-coherent models for self-customized BSP trees [6]. Nonparametric learning avoids the limitations of closed-form distributions but its aims are predictive rather than prescriptive and algorithmic speedups are usually not among them.

Algorithmic self-improvement differs from past work on self-organizing data structures and online computation in two fundamental ways: (i) self-improving algorithms operate offline and do not lend themselves to competitive analysis; (ii) they do not exploit structure within any given input but, rather, within the ensemble of input distributions. For example, suppose that the distribution consists of two random but *fixed* permutations. Our self-improving sorter will run in linear time, whereas *any* self-organizing data structure for searching would require $\Omega(n \log n)$ time.

A Bayesian version of self-improvement would be to postulate a prior and treat the I_k's as data conditioning a posterior distribution. While this paper is concerned with memoryless sources, it is easy to imagine extensions to higher-order models. One could also consider time-varying distributions or Markov models. We believe that sticking to memoryless sources for self-improving algorithms is far less restrictive than doing the same for online computation. Take speech for example: The weakness of a memoryless model is that the next utterance is highly correlated with the previous ones; hence the use of Markov models. A self-improving algorithm would operate at the level of a sentence or a paragraph—not an utterance—where correlations are more diffuse and a memoryless source might be a good first approximation.

2 A Self-Improving Sorter

For simplicity, we break up our discussion into a non-learning and a learning phase. First, we assume that the distribution $\mathcal{D} = \prod_i \mathcal{D}_i$ is known ahead of time and that we are allowed some amount of preprocessing before having to deal with the first input instance (§2.1). Both assumptions are unrealistic, so we show how to remove them to produce a bona fide self-improving sorter (§2.2). The surprise is how strikingly little of the distribution needs to be learned for effective self-improvement.

Remark: Much research has been done on adaptive sorting [17]; in particular, on algorithms that exploit near-sortedness. Our approach is conceptually different. We seek to exploit properties, not of individual inputs, but of their distribution. In particular, our sorter runs in linear time for permutations drawn from a linear-entropy source, even though any individual input might be a *perfectly random* permutation. We are not aware

[1] If $T_k(n)$ is the expected running time for solving I_k—over the input distribution and the random bits of the algorithm—then, as k grows, $T_k(n)$ (or a function bounding it) converges to the *limiting running time*.

of any previous algorithm that can achieve that.

THEOREM 2.1. *There exists a self-improving sorter that runs in $O(H(\mathcal{D}_<)+n)$ limiting running time, for any input distribution $\mathcal{D}=\prod_i \mathcal{D}_i$. Its worst case running time is $O(n\log n)$. The storage requirement is $O(n^{1+\varepsilon})$, for any fixed $\varepsilon>0$, which is optimal in the worst case. The algorithm reaches its limiting complexity within $O(\log n)$ sorting rounds.*

Can we hope for a similar result if we drop the independence assumption? The answer is no.

LEMMA 2.1. *There exists an input distribution $\mathcal{D}$ such that any self-improving sorter that runs in $O(H(\mathcal{D}_<)+n)$ limiting running time requires at least $\Omega(2^{H(\mathcal{D}_<)})$ storage.*

Proof. Consider the set of all $n!$ permutations. Every subset S of 2^n permutations induces a distribution $\mathcal{D}^S$ such that $\mathcal{D}^S_<$ has every permutation in S with equal probability (and only permutations in S). $H(\mathcal{D}^S_<)$ is n, and by definition any optimal self-improving sorter will (eventually) sort a permutation from $\mathcal{D}^S$ in expected cn time, for some constant c. The total number of such distributions is $\binom{n!}{2^n} > (n!/2^n)^{2^n}$. Consider a self-improving sorter A that uses s units of storage. Let the algorithm that results from setting this space with the string v be A_v. A distribution $\mathcal{D}^S$ is *handled* by A_v, if A_v sorts a permutation from $\mathcal{D}^S$ in expected cn time. By the pigeon-hole principle, there exists some string w such that A_w handles at least $(n!/2^n)^{2^n}2^{-s}$ distributions.

Define a permutation to be *easy* if A_w sorts the permutation in $2cn$ time. Consider some $\mathcal{D}^S$ that is handled by A_w. Then, by Markov S has at least $2^n/2$ permutations which are easy for A_w. A simple information-theoretic argument shows that there exist at most 2^{2cn} easy permutations. Any handled distribution is generated from a set of permutations with at least $2^n/2$ permutations from the set of easy permutations; the rest can be anything. Therefore the total number of handled distributions is at most -

$$\binom{n!}{2^n/2}\binom{2^{2cn}}{2^n/2} < (n!)^{2^n/2}2^{cn2^n}$$

This must be greater than $(n!/2^n)^{2^n}2^{-s}$. Therefore $s=\Omega(2^n n\log n)$. Since $H(\mathcal{D}^S_<)=n$, the theorem is proved.

□

2.1 Sorting with Full Knowledge We consider the problem of sorting $I=\{x_1,\ldots,x_n\}$, where each x_i is drawn from a distribution $\mathcal{D}_i$, which is specified by a vector $(p_{i,1},\ldots,p_{i,N})$, where $p_{i,j}=\text{Prob}[\,x_i=j\,]$. We can assume without loss of generality that all the x_i's are distinct. (If not, simply replace x_i by nx_i+i-1 for tie-breaking purposes and enlarge N to $n(N+1)$. All probabilities and entropies remain the same.)

- THE V-LIST: Fix an integer parameter $\lambda=c\log n$, for large enough c, and sample λ input instances from $\prod\mathcal{D}_i$. Form their union and sort the resulting λn-element multiset it a single list $u_1\le\cdots\le u_{\lambda n}$. Next, extract from it every λ-th item and form the list $V=(v_0,\ldots,v_{n+1})$, where $v_0=0$, $v_{n+1}=\infty$, and $v_i=u_{i\lambda}$ for $0<i\le n$. Keep the V*-list* in a sorted table as a snapshot of a "typical" input instance. We will prove the remarkable fact that, with high probability, locating each x_i in the V-list is linearly equivalent to sorting I. We cannot afford to search the V-list directly, however. To do that, we need auxiliary search structures.

- THE D_i-TREES: For any $i>0$, let pred^v_i be the predecessor[2] of a random y from $\mathcal{D}_i$ in the V-list, and let H^v_i be the entropy of pred^v_i (which cannot exceed the entropy of $\mathcal{D}_i$). The D_i-tree is an optimum binary search tree [37] over the keys of the V-list, where the access probability of v_k is $\sum_j\{\,p_{i,j}\mid v_k\le j<v_{k+1}\,\}$, for any $0\le k\le n$: the same distribution used to define H^v_i. This allows us to compute pred^v_i using $H^v_i+O(1)$ expected comparisons. We can reduce the size of each D_i-tree to $O(n^\varepsilon)$, for any fixed $\varepsilon>0$, without losing more than a constant factor in the running time. (Details omitted.)

To sort I, first we search for each x_i in the V-list using the previous technique. This allows us to partition I into groups $G_1<G_2<\cdots$ of x_i's sharing the same predecessor in the V-list. The sorting of I is complete once we go through each G_j and quicksort their elements. The first phase of the algorithm takes $O(n+\sum_i H^v_i)$ expected time[3]. What about the second? Its complexity is $O(n)$, as follows from:

LEMMA 2.2. $\mathbf{E}\,|\,\{\,i\mid v_k\le x_i<v_{k+1}\,\}\,|^2=O(1)$, *for any $0\le k\le n$.*

To prove Lemma 2.2, we introduce an analytical device in the form of an idealized V-list. Given a random $\{x_1,\ldots,x_n\}$ drawn from $\mathcal{D}$ and a real z, let $\rho(z)=$

[2]Throughout this paper, the predecessor of y in a list refers to the index of the largest list element $\le y$; it does not refer to the element itself.

[3]The H^v_i's themselves are random variables depending on the choice of the V-list. Therefore, this is a conditional expectation.

$\mathbf{E}\,|\{\, i \,|\, x_i \le z \,\}|$. The function ρ grows monotonically from 0 to n and $\rho(z+1) \le \rho(z)+1$; therefore, for each $0 \le k \le n$, there is a maximum integer $b_k \le N$ such that $\lceil \rho(b_k) \rceil = k$. Call $b_0 < \cdots < b_n = N$ the *B-list.* The expected number of x_i's in $[b_k, b_{k+1})$ is less than 3; therefore $\sum_i q_i < 3$, where $q_i = \mathrm{Prob}\,[\, b_k \le x_i < b_{k+1} \,]$. So, for any $0 \le k < n$, using (pairwise) independence, $\mathbf{E}\,|\{\, i \,|\, b_k \le x_i < b_{k+1} \,\}|^2 = \sum_i q_i + 2\sum_{i<j} q_i q_j \le \sum_i q_i + (\sum_i q_i)^2 < 12$. With this inequality, Lemma 2.2 follows from the first part of:

LEMMA 2.3. *With probability at least $1 - 1/n^3$, (i) no interval $[v_k, v_{k+1})$ contains more than three b_j's; and (ii) no interval $[b_k, b_{k+1})$ contains more than three v_j's.*

Proof. Suppose that some $[v_k, v_{k+1})$ contains at least $b_j, b_{j+1}, b_{j+2}, b_{j+3}$. If Y is the number of elements in the original sample $u_1, \ldots, u_{\lambda n}$ that lie in $[b_j, b_{j+3}]$, then $Y \le \lambda$. But $Y = \sum_l y_l$, where y_l is the indicator variable for $u_l \in [b_j, b_{j+3}]$. Let $\alpha_l = \mathrm{Prob}\,[\, y_l = 1 \,]$, $\alpha = (\sum_l \alpha_l)/m$, and $m = \lambda n$. Note that $2\lambda \le \mathbf{E}\,Y \le 4\lambda$. By Chernoff's bound [4] (page 268), $Y < \alpha m - a$ with probability less than $e^{-a^2/2\alpha m}$. Setting $a = \lambda/2$ proves that, with probability $1 - 2^{-\Omega(\lambda)}$, $[v_k, v_{k+1})$ contains no more than three b_j's. With $\lambda = c \log n$, for c large enough, the union bound ensures that, with probability at least $1 - n^{-3}$, this holds true of every interval $[v_k, v_{k+1})$.

To prove (ii), suppose now that some $[b_k, b_{k+1})$ contains at least $v_j, v_{j+1}, v_{j+2}, v_{j+3}$,. Define $Z = \sum_l z_l$, where z_l is the indicator variable for $u_l \in [b_k, b_{k+1})$. Let $\beta_l = \mathrm{Prob}\,[\, z_l = 1 \,]$ and $\beta = (\sum_l \beta_l)/m$. Note that $Z \ge 3\lambda$ and yet $\mathbf{E}\,Z \le 2\lambda$. We can actually assume that $\mathbf{E}\,Z = 2\lambda$, since it is the worst case. Again, by Chernoff's bound, $Z \ge \beta m + a$ with probability less than $e^{-a^2/2\beta m + a^3/2(\beta m)^2}$. Setting $a = \lambda$ proves that, with probability $2^{-\Omega(\lambda)}$, $[b_k, b_{k+1})$ contains no more than three v_j's. We complete the proof by repeating the previous union bound argument. $\square$

We have shown that the algorithm takes $O(n + \sum_i H_i^v)$ expected time (given a fixed V-list). Given the artificial nature H_i^v, there is no obvious reason why this should be optimal. Indeed, to prove that it is requires a little effort. We sketch the proof. Let pred_i^f be the predecessor of a random y from $\mathcal{D}_i$ in $\{0, x_1, \ldots, x_{i-1}\}$, and let H_i^f be the entropy of pred_i^f. Fredman proved that the expected time of any comparison-based sorter is $\Omega(n + \sum_i H_i^f)$ [19]. This result together with the following lemma tells us that the algorithm runs in optimal time on expectation.

LEMMA 2.4.
$$\mathbf{E}\left[n + \sum_i H_i^v \right] = O\Big(n + \sum_i H_i^f \Big) \;.$$

Proof. We introduce two new entropies: Define the D-list $\{d_k = b_{2k}\}$ to consist of the even-indexed entries of the B-list. Let pred_i^x and pred_i^d be the predecessors of a random y from $\mathcal{D}_i$ in $\{0, x_1, \ldots, x_n\}$ and in the D-list, respectively. (Note that y and x_i are drawn *independently* from the same source.) We define H_i^x (resp. H_i^d) to be the entropy of pred_i^x (resp. pred_i^d). The predecessor of y in $\{0, x_1, \ldots, x_n\}$ can be inferred from three pieces of information: (i) y's predecessor in $\{0, x_1, \ldots, x_{i-1}\}$; (ii) y's predecessor in $\{0, x_{i+1}, \ldots, x_n\}$; and (iii) whether $y < x_i$, $y = x_i$, or $y > x_i$. By using bijectivity between the input instances $(x_1, \ldots, x_n)$ and $(x_n, \ldots, x_1)$, it then follows that $\sum_i H_i^x = O(n + \sum_i H_i^f)$. Given a random $(x_1, \ldots, x_n)$ drawn from $\mathcal{D}$, let $\chi_k = 1$ if $[d_k, d_{k+1})$ is empty, ie, does not contain any x_i, and 0 otherwise, and let $q_{i,k} = \mathrm{Prob}\,[\, d_k \le x_i < d_{k+1} \,]$. Note that the expected number of x_i's in that interval is at least 1; therefore, $\mathrm{Prob}\,[\, \chi_k = 1 \,] = \prod_i (1 - q_{i,k}) \le e^{-\sum_i q_{i,k}} \le e^{-1}$. For a fixed $(x_1, \ldots, x_n)$, glue together consecutive empty intervals, ie, remove d_k if both $[d_{k-1}, d_k)$ and $[d_k, d_{k+1})$ are empty. Let $d'_1 < d'_2 < \cdots$ be the subset of $\{d_i\}$ thus left, and let $\mathrm{pred}_i^{d'}$ be the predecessor of a random y from $\mathcal{D}_i$ among $\{d'_k\}$. Finally, let ξ be 0 if y lies in an interval $[d'_k, d'_{k+1})$ that coincides with one of the nonempty $[d_j, d_{j+1})$'s, and 1 otherwise; and denote $\mathrm{Prob}\,[\, \xi = 1 \,]$ by p_ξ. Using standard information theory, (2.1)
$$H_i^d = H(\mathrm{pred}_i^d) \le H(\mathrm{pred}_i^{d'}) + p_\xi H(\, \mathrm{pred}_i^d \,|\, \xi = 1\,);$$
$$\begin{aligned}\mathbf{E}\left\{ p_\xi\, H(\, \mathrm{pred}_i^d \,|\, \xi = 1\,) \right\} &= \mathbf{E} \sum_k \chi_k q_{i,k} \log \frac{p_\xi}{q_{i,k}} \\ &\le \sum_k (\mathbf{E}\,\chi_k) q_{i,k} \log \frac{1}{q_{i,k}} \le e^{-1} \sum_k q_{i,k} \log \frac{1}{q_{i,k}} \\ &= H_i^d / e.\end{aligned}$$

For any fixed $(x_1, \ldots, x_n)$, $\mathrm{pred}_i^{d'}$ can be inferred from pred_i^x with one extra bit. This remains true on average, and so by (2.1)
$$\begin{aligned}H_i^d &= \mathbf{E}\, H(\mathrm{pred}_i^{d'}) + \mathbf{E}\left\{ p_\xi\, H(\, \mathrm{pred}_i^d \,|\, \xi = 1\,) \right\} \\ &\le \frac{e}{e-1}\, \mathbf{E}\, H(\mathrm{pred}_i^{d'}) \le \frac{e}{e-1}\, (H_i^x + 1);\end{aligned}$$

and therefore, $H_i^d = O(H_i^x + 1)$. By Lemma 2.3, with probability $p_\zeta \ge 1 - 1/n^3$, no interval $[d_k, d_{k+1})$ contains more than six v_j's. Let $\zeta = 1$ if that is the case, and 0 otherwise. If $\zeta = 1$, then pred_i^v can be inferred from pred_i^d with only three extra bits. Therefore,
$$\begin{aligned}\mathbf{E}\, H_i^v &\le \mathbf{E}[H_i^v | \zeta = 1] + n^{-3} \mathbf{E}[H_i^v | \zeta = 0] \\ &\le H_i^d + 3 + n^{-3} \log n \;.\end{aligned}$$

The lemma follows now from the relations above among H_i^x, H_i^f, H_i^d. $\square$

We can show that the storage cannot be reduced to linear. In particular, our $O(n^{1+\varepsilon})$ bound is optimal for some distributions.

LEMMA 2.5. *For any $H \le bn\log n$, with small enough constant $b > 0$, there exists a distribution $\mathcal{D} = \prod_i \mathcal{D}_i$ of entropy H such that any comparison-based algorithm that can sort a random permutation from $\mathcal{D}$ in expected time $O(H+n)$ requires a data structure of bit size $\Omega(2^{H/n} n\log n)$.*

Proof. The basic idea of the proof is the same as the proof of Lemma 2.1. Let $h = 2^{\lfloor H/n \rfloor}$. We define $\mathcal{D}_i$ by choosing h distinct integers in $[1, n]$ and making them equally likely to be picked as x_i. This leads to $\binom{n}{h}^n > (n/h)^{hn}$ choices of distinct distributions $\mathcal{D}$. Suppose that there is a data structure of size s that can accommodate any such distribution with an expected running time of $O(H+n)$. Then one such data structure $\mathcal{S}$ must be able to accommodate this running time for a set $\mathcal{G}$ of at least $(n/h)^{hn}2^{-s}$ distributions $\mathcal{D}$. Each $\mathcal{D}_i$ is characterized by a vector $v_i = (a_{i,1}, \ldots, a_{i,h})$, so that $\mathcal{D}$ itself is specified by $v = (v_1, \ldots, v_n) \in \mathbf{R}^{nh}$. (From now on, we view v both as a vector and a distribution of input instances.) Define the j-th projection of v as $v^j = (a_{1,j}, \ldots, a_{n,j})$. Viewed as an input instance, we say that v^j is *easy* if $\mathcal{S}$ sorts it within $C(H+n)$ time, for large enough C. Of course, even if $v \in \mathcal{G}$, it could well be that none of the projections of v are easy. However, if we consider the projections obtained by permuting the coordinates of each vector $v_i = (a_{i,1}, \ldots, a_{i,h})$ in all possible ways, we enumerate each input instance from v the same number of times; therefore, there exists a choice of permutations for which at least half of the projections are easy. Let I_v denote the corresponding input instances.

How many distributions v can each generate at least $h/2$ input instances from I_v. There are fewer than $|I_v|^{h/2}$ choices of such instances and, for any such choice, each $v_i = (a_{i,1}, \ldots, a_{i,h})$ has half its entries already specified, so the remaining choices are fewer than $n^{hn/2}$. This gives an upper bound of $n^{hn/2}|I_v|^{h/2}$ on the number of distributions. This number cannot be smaller than $(n/h)^{hn}2^{-s}$; therefore

$$|I_v| \ge n^n h^{-2n} 2^{-2s/h}. \tag{2.2}$$

In a comparison-based decision tree model, each input instance is associated with the leaf of a tree of depth at most $C(H+n)$, ie, with one of at most $2^{C(H+n)}$ leaves. This would give us a lower bound on s if each instance was assigned a distinct leaf. But this may not be the case. However, we have a *collision* bound, saying that at most $4^n/2$ instances can be mapped to the same leaf. This implies that $|I_v|4^{-n} \le 2^{C(H+n)}$; and by (2.2), $s = \Omega(hn\log n)$; hence the lemma.

To prove the collision bound, we use the tie-breaking rule of §2.1 to map each input instance to a permutation (the one induced by the map $x_i \mapsto nx_i + i - 1$). It is clear that two instances mapping to two distinct permutations must lead to two different leaves of the decision tree. So the only question left is to bound the number of instances mapping to a given permutation. Let $(x_1, \ldots, x_n)$ be an input instance (no tie-breaking). For $i = 2, \ldots, n$, define $\alpha_i = 0$ if $x_i = x_{i-1}$, and 1 otherwise. For $j = 1, \ldots, n$, define $\beta_j = 0$ if $x_j = i$ for some j, and 1 otherwise. For example $(3, 3, 3, 5, 5, 3, 7, 7)$ gives 0010110 for the α_i's and 11010101 for the β_j's, and the induced permutation is $(1, 2, 3, 5, 6, 4, 7, 8)$. It is elementary to see that any input instance can be fully recovered if we are given its tie-breaking induced permutation together with the bit vectors α_i, β_i. This proves the collision bound. $\square$

2.2 Learn-And-Sort It takes $O(\log n)$ rounds to build the V-list. Instead of a table, we use a perfect binary search tree to store the V-list. The D_i-trees require dynamic updates, so we turn to splay trees [44]. We maintain a buffer of size $M = n^\varepsilon(\varepsilon\log n + \log\delta^{-1})$, for some constant $\delta > 0$, in which we record the first M outcomes of pred_i^v. We then initialize the D_i-splay tree only over the set of recorded predecessors. It can be shown that, with probability at least $1-\delta$, all b_k's such that $\mathrm{Prob}\,[\, b_k \le x_i < b_{k+1}\,] \ge n^{-\varepsilon}$, for some $1 \le i \le n$, are encountered in the first M rounds. An argument similar to the one used in §2.1 shows that the total storage requirement is still $O(n^{1+\varepsilon})$.

3 Self-Improving Clustering

Clustering arises in quantitative data analysis [22, 38] under many different formulations [3, 7, 8, 15, 23, 24, 26, 31, 46]. Our approach in this section is not wedded to any specific one, but, for concreteness, we focus on the k-median problem over the Hamming cube: Given a set I of n points in $\{0,1\}^d$, find a set $\mathcal{C} \subseteq \{0,1\}^d$ ($|\mathcal{C}| \le k$) that minimizes $\sum_{x\in I} d(x, C)$, where $d()$ is the L^1 distance. Our ideas for the two-center case easily extend to the general problem, so we restrict our discussion to the case $k = 2$. Formulated as a maximization problem, NP-completeness was proven by Kleinberg et al [30], who also gave a constant factor approximation algorithm, later improved to a PTAS by Alon and Sudakov [5]. Using dimension reduction methods, Ostrovsky and Rabani [40] derived a PTAS for the corresponding minimization problem (note that approximate solutions might be quite different) with a running time[4] of $O(n^{poly(\varepsilon^{-1})}d^2)$. By sampling over the points rather than the dimensions, Kumar et al. [31] produced a different algorithm (henceforth denoted mod-KSS) that has the advantage of running in linear time for any fixed ε. Although designed originally for the Euclidean k-means problem, it can be easily adapted to the problem at hand: Its complexity becomes $\varepsilon^{-O(\varepsilon^{-2})}dn$.

Is there a self-improving version of mod-KSS? The method identifies a small set of candidate center pairs from which, with high probability, a suitable pair will be an approximate solution. The construction is non-oblivious, however, and it appears unlikely that a single set can accommodate the bulk of input instances from a random source. But by modifying the random sampling used in mod-KSS (in a way somewhat reminiscent of what we did in the sorting section §2), we are able to construct a suitable set $\mathcal{P}$ of candidate pairs. With this in hand, we can build the necessary machinery to achieve self-improvement. We omit many of the technical details from this abstract.

Theorem 3.1. *Let $\mathcal{D} = \prod_{i=1}^{n} \mathcal{D}_i$ be a random source of n-point sets in $\{0,1\}^d$, where each $\mathcal{D}_i$ has positive entropy and minimum point-wise probability $p > 0$ on its support. Assume that $n = \tilde{\Omega}(\frac{d}{\varepsilon^4 p})$. There exists a self-improving $(1+\varepsilon)$-approximation algorithm for the 2-median problem that runs in linear limiting running time and $O(d)$ space. Errors occur with arbitrarily small probability. The algorithm can be made Monte Carlo (ie, with error probability independent of the input) with $\exp(poly(d, p^{-1}, \varepsilon^{-1}))$ extra storage.*

Note that by the assumptions of the theorem, p must be at most 1/2.

3.1 Construction of candidate pairs We describe the construction of $\mathcal{P}$ (Figure 1). As usual, the input is a set $I = \{x_1, \ldots, x_n\}$ drawn randomly from an unknown source $\mathcal{D} = \prod_i \mathcal{D}_i$, ie, each $x_i \in \{0,1\}^d$ is drawn independently from $\mathcal{D}_i$ (assumed to be of positive entropy). Some notation :

- $\mathtt{OPT}_2\,(I, c_1, c_2) = \sum_{x \in I} \min_i d(x, c_i)$.
- Given $S \subseteq \{0,1\}^d$, $\mathrm{Maj}\,(S)$ is the point obtained by taking majority coordinate-wise in S. Note that the 1-median of I has cost $\mathtt{OPT}_1\,(I, \mathrm{Maj}\,(S)) = \sum_{x \in I} \min_i d(x, \mathrm{Maj}\,(S))$.
- $Y_i = \mathrm{support}\,(\mathcal{D}_i)$ and $Y = \cup_{i=1}^{n} Y_i$ (Y and its subsets are multisets).
- Given any $Z \subseteq Y$, $\mathtt{IND}\,(Z) = \{\, i \,|\, Z \cap Y_i \neq \phi\}$.

Our objective is to obtain a $(1+\varepsilon)$-approximation for $\mathtt{OPT}_2\,(I) = \min_{c_1, c_2} \mathtt{OPT}_2\,(I, c_1, c_2)$.

Lemma 3.1. *With probability $> 1 - 2^{-s}$, a random $I \in \mathcal{D}$ satisfies $\mathtt{OPT}_2\,(I, c_1, c_2) \leq (1 + 3\varepsilon)\,\mathtt{OPT}_2\,(I)$ for some $(c_1, c_2) \in \mathcal{P}$; furthermore, $|\mathcal{P}| = (p\varepsilon)^{-O(s)} \log n$ ($s = b_0 \varepsilon^{-2}$ for some constant b_0).*

We now sketch the proof of this lemma. Let I_1, I_2 be the two clusters producing $\mathtt{OPT}_2\,(I)$, with $|I_1| \geq |I_2|$.

Claim 3.1. *With probability $1 - 2^{-\Omega(\lambda)}$, for some primary center c_1, $\min_c \mathtt{OPT}_2\,(I, c_1, c) \leq (1+\varepsilon)\mathtt{OPT}_2\,(I)$.*

Proof. Let S be the s random points picked in forming any one of the primary centers. Clearly,

$$\mathrm{Prob}\,[\, S \subseteq I_1\,] = \mathrm{Prob}\,[\, S \subseteq I_1 \,|\, \mathtt{IND}\,(S) \subseteq \mathtt{IND}\,(I_1)\,] \times \mathrm{Prob}\,[\, \mathtt{IND}\,(S) \subseteq \mathtt{IND}\,(I_1)\,] \geq (p/2)^s.$$

Conditioned upon being a subset of $\mathtt{IND}\,(I_1)$, the indices of $\mathtt{IND}\,(S)$ are uniformly distributed within $\mathtt{IND}\,(I_1)$. By a sampling argument (omitted from this abstract), we can prove that $\mathtt{OPT}_1\,(I_1, \mathrm{Maj}\,(S)) \leq (1+\varepsilon)\mathtt{OPT}_1\,(I_1, \mathrm{Maj}\,(I_1))$, with probability $\geq \frac{1}{2}(p/2)^s$. If so, we set c_1 to $\mathrm{Maj}\,(S)$ (henceforth, c_1 will denote this point). Picking $\lambda(2/p)^s$ primary centers ensures the existence of such a c_1 with probability $1 - 2^{-\Omega(\lambda)}$. Let (I_1', I_2') be the two clusters induced by $\min_c \mathtt{OPT}_2\,(I, c_1, c)$. Note that $\min_c \mathtt{OPT}_2\,(I, c_1, c) \leq \mathtt{OPT}_2\,(I, c_1, \mathrm{Maj}\,(I_2')) \leq (1+\varepsilon)\,\mathtt{OPT}_2\,(I)$. □

Let $t = d(c_1, \mathrm{Maj}\,(I_2'))$ and $Q_i = \{\, x \in I \,|\, d(c_1, x) \geq t/2\,\}$. By the triangular inequality, $I_2' \subseteq Q_i$, and, using a standard irreducibility argument [8, 31], we can assume that $|I_2'| \geq \frac{\varepsilon}{2}|Q_i|$. Indeed, if that were not the case, then mapping all of I_2' to c_1 instead of to its own majority point, would incur an additive cost $\leq \frac{\varepsilon}{2}t|Q_i|$. But then, $\mathtt{OPT}_2\,(I, c_1, \mathrm{Maj}\,(I_2')) \geq (1 - \varepsilon/2)|Q_i|t/2$; therefore, using c_1 as the sole center would give us a cost $\leq (1 + 3\varepsilon/2)\,\mathtt{OPT}_2\,(I, c_1, \mathrm{Maj}\,(I_2')) \leq (1 + 3\varepsilon)\,\mathtt{OPT}_2\,(I)$, which would be acceptable.

Definition 3.1. *Input I is* typical *if, regardless of the choice of c_1 (in the construction of $\mathcal{P}$), $2^{j-1} \leq |I \cap P_j| \leq 2^{j+1}$ for all $j > j_0$.*

Claim 3.2. *Typicality is violated with probability $\lambda(2/p)^s 2^{-\Omega(p2^{j_0})} \leq 2^{-b_0 s}$.*

[4] All the algorithms in this section are probabilistic.

The Set $\mathcal{P}$ of Center Pairs

1. Primary centers c_1: Fix three large enough parameters b_0, s, λ, where $s = b_0\varepsilon^{-2}$, $\lambda \le 2^{s/p}$, and $b_0\varepsilon$ is small enough. Repeat $\lambda(2/p)^s$ times: Sample I from $\mathcal{D}$; pick a set S of s random points from I; and output $\mathrm{Maj}\,(S)$.
2. Secondary centers c_2: For each c_1, sort Y by decreasing distance from c_1 and, for $j = 0, 1, \ldots, \log n$, form the prefix P_j of Y of cumulative probability 2^j, ie, $\sum_{y_l \in P_j} \mathrm{Prob}\,[y_l] = 2^j$ (where probability defined with respect to the $\mathcal{D}_i$ that y_l is drawn from). Let $j_0 = \log(s/p^2) + b_0$. (Round to nearest integer whenever needed.)
   (a) For all subsets S of at most s points from P_{j_0}, output $\mathrm{Maj}\,(S)$.
   (b) For each $j > j_0$, repeat $\lambda(8/p^2\varepsilon)^s$ times: Pick s random indices from $\mathtt{IND}\,(P_j)$ and, for each such index i, sample one point from $\mathcal{D}_i$ to form S: Output $\mathrm{Maj}\,(S)$.

Figure 1: Construction of the set $\mathcal{P}$ of center pairs

Proof. Since $\mathbf{E}\,|I \cap P_j| = \sum_{y_l \in P_j} \mathrm{Prob}\,[y_l] = 2^j$ and $\mathtt{IND}\,(P_j) \le |P_j| \le 2^j/p$, (by Chernoff) a random $I \in \mathcal{D}$ is not typical (for fixed c_1) with probability $\sum_{j > j_0} 2^{-\Omega(p2^j)} \le 2^{-\Omega(p2^{j_0})}$. By a union bound over all c_1's, we get the desired bound. $\square$

Let P_{j_i} denote the smallest $P_j \supseteq Q_i$. We now prove that together with c_1, some c_2 (secondary center) satisfies the condition given in the lemma. We distinguish between two cases:

- $j_i \le j_0$: All subsets of $|I'_2|$ of size $\le s$ are enumerated, and we already know that one of them, S, must satisfy $\mathtt{OPT}_1\,(I'_2, c_2) \le (1+\varepsilon)\mathtt{OPT}_1\,(I'_2, \mathrm{Maj}\,(I'_2))$, where $c_2 = \mathrm{Maj}\,(S)$. This implies that $\mathtt{OPT}_2\,(I, c_1, c_2) \le (1+\varepsilon)\mathtt{OPT}_2\,(I, c_1, \mathrm{Maj}\,(I'_2)) \le (1+3\varepsilon)\,\mathtt{OPT}_2\,(I)$.
- $j_i > j_0$: Assume that I is typical. Since $Q_i \supset I \cap P_{j_i - 1}$, $|Q_i| \ge 2^{j-2}$; therefore $|I'_2| \ge \varepsilon 2^{j-3}$. In view of $\mathtt{IND}\,(P_j) \le 2^j/p$, we find that $|I'_2| \ge \frac{p\varepsilon}{8}\,\mathtt{IND}\,(P_j)$. The sampling in 2(b) is uniform over the indices of $\mathtt{IND}\,(P_j)$; therefore,

  $$\begin{aligned}\mathrm{Prob}\,[\,S \subseteq I'_2\,] = &\\ &\mathrm{Prob}\,[\,S \subseteq I'_2 \mid \mathtt{IND}\,(S) \subseteq \mathtt{IND}\,(I'_2)\,] \times\\ &\mathrm{Prob}\,[\,\mathtt{IND}\,(S) \subseteq \mathtt{IND}\,(I'_2)\,] \ge (p^2\varepsilon/8)^s.\end{aligned}$$

  Again, using the uniformity of the distribution within $\mathtt{IND}\,(I'_2)$, we find that $\mathtt{OPT}_1\,(I'_2, \mathrm{Maj}\,(S)) \le (1+\varepsilon)\mathtt{OPT}_1\,(I'_2, \mathrm{Maj}\,(I'_2))$ with probability $\ge \frac{1}{2}(p^2\varepsilon/8)^s$. Picking $\lambda(8/p^2\varepsilon)^s$ secondary centers (for fixed c_1) ensures, with probability $1 - 2^{-\Omega(\lambda)}$, the existence of such a c_2, where $\mathtt{OPT}_2\,(I, c_1, c_2) \le (1+\varepsilon)^2\,\mathtt{OPT}_2\,(I)$. Removing the typicality assumption brings the probability of success down to $1 - 2^{-b_0 s} - 2^{-\Omega(\lambda)}$.

The size of $\mathcal{P}$ is $\le (|P_{j_0}|^s + \lambda(8/p^2\varepsilon)^s)\lambda(2/p)^s \log n$. Setting $\lambda = b_0 s$ and plugging in $|P_{j_0}| \le 2^{j_0}/p$ and $j_0 = \log(s/p^2) + b_0$ gives $|\mathcal{P}| = (p\varepsilon)^{-O(s)} \log n$ (and the desired probability of error).

Learning phase (on first $O(\log n)$ inputs):
```
0.construct P
1.for first N = O(log n) inputs I
  (a) calculate Z_{c1,c2} for all (c1,c2) ∈ P
  (b) run mod-KSS to output a (1+ε)-
      approximation for I
2.for each (c1,c2) ∈ P compute average Ẑ_{c1,c2}
  over the first N samples of Z_{c1,c2}
3.choose c1*, c2* with minimum Ẑ_{c1*,c2*}
```

Normal phase (on all other inputs):
```
  for all inputs I, return c1*, c2*
```

Figure 2: Self-Improving algorithm for clustering

3.2 Learn-And-Cluster For fixed c_1, c_2, the random variable $Z_{c_1,c_2} = \mathtt{OPT}_2\,(I, c_1, c_2)$ is highly concentrated around its mean. Indeed, $Z_{c_1,c_2} = \sum_{i=1}^{n} z_i$, where $z_i = \min_j d(x_i, c_j)$ for random $x_i \in \mathcal{D}_i$. We may assume that $\mathbf{E}\,Z_{c_1,c_2} = \Omega(pn)$. Indeed, if $\mathbf{E}\,Z_{c_1,c_2} \le n/2$

```
Normal phase (Monte-Carlo):
  for input I
    if I is nonhot or nontypical then
      return mod-KSS(I)
    else
      return c1*, c2*
```

Figure 3: No input left behind

then with probability at least $1 - e^{-\Omega(n/d)}$ we will have $Z_{c_1,c_2} \leq 3n/5$, in which case computing the optimal solution is trivial.

Let $\mathcal{E}_{c_1,c_2}$ denote the event

$$|Z_{c_1,c_2} - \mathbf{E}\, Z_{c_1,c_2}| \leq \varepsilon \mathbf{E}\, Z_{c_1,c_2} \ ,$$

that is, Z_{c_1,c_2} is close to its mean (up to a relative error of ε). By using a generalized Chernoff bound (for sums of variables with different variances) and a convexity argument, we find that

$$\begin{aligned}\mathrm{Prob}_{I\in\mathcal{D}}\left[\mathcal{E}_{c_1,c_2}\right] &\geq 1 - 2^{-\Omega(\varepsilon^2(\sum_i \mathbf{E}\, z_i)^2/\sum_i \mathrm{Var}\, z_i)} \\ &\geq 1 - 2^{-\Omega(\varepsilon^2 pn/d)}.\end{aligned}$$

Definition 3.2. *An input I is* hot *if $\mathcal{E}_{c_1,c_2}$ occurs for all $(c_1, c_2) \in \mathcal{P}$.*

Taking a union bound over all center pairs in $\mathcal{P}$, we know that this happens (by Lemma 3.1) with probability at least $1 - (p\varepsilon)^{-O(\varepsilon^{-2})}\, 2^{-\Omega(\varepsilon^2 pn/d)} \log n > 1 - 2^{-s}$, for $n = \tilde{\Omega}(d/\varepsilon^4 p)$.

We know that with probability $> 1 - 2^{1-s}$, an input $I \in \mathcal{D}$ is hot and satisfies the condition specified in Lemma 3.1. Therefore, there exists some $(c_1, c_2) \in \mathcal{P}$ such that $\mathtt{OPT}_2\,(I, c_1, c_2) \leq (1 + 3\varepsilon)\,\mathtt{OPT}_2\,(I)$. Also, for all $(c_1', c_2') \in \mathcal{P}$, $\mathtt{OPT}_2\,(I, c_1', c_2') \leq (1+\varepsilon)\mathbf{E}\, Z_{c_1',c_2'}$. If we can find a pair $(c_1^*, c_2^*) \in \mathcal{P}$, such that $\mathbf{E}\, Z_{c_1^*,c_2^*}$ is at most $(1+\varepsilon)\min_{(c_1',c_2')\in\mathcal{P}} \mathbf{E}\, Z_{c_1',c_2'}$, then (c_1^*, c_2^*) is a $(1+O(\varepsilon))$-approximate solution for I. This is what the learning phase of the self-improving algorithm (Figure 2) does. It will run at a cost of $\varepsilon^{-O(\varepsilon^{-2})}dn$ steps per input (for running mod-KSS). The pair (c_1^*, c_2^*) chosen at the end will satisfy the condition given above with probability $1-1/\mathrm{poly}(n, \varepsilon^{-1}, p^{-1})$. With probability $> 1-2^{1-s}$, the clustering is a $(1+O(\varepsilon))$-approximation of the optimum. (Rescale ε to get a $(1+\varepsilon)$ factor.)

3.3 Monte-Carlo clustering The benefit of the above trimmed-down approach is that it requires no extra space on top of the space used by mod-KSS (note that $\hat{Z}_{c_1,c_2}$ for $(c_1, c_2) \in \mathcal{P}$ can be calculated after the first $O(\log n)$ inputs are handled, without the need of using $\Omega(|\mathcal{P}|)$ storage). There is a drawback, however: The error probability, though arbitrarily small, is due not only to the randomization but also to the distribution $\mathcal{D}$, which runs contrary to the "no input left behind" philosophy of Monte Carlo algorithms.

We describe how to fix this, thereby completing the proof of Theorem 3.1.

The cause of errors due to the random source is two-fold: (i) nontypicality and (ii) nonhotness, both occurring with probability $\leq 2^{-s}$. It is not ruled out that nontypicality and nonhotness are a property of a (rare) input I, regardless of however many parallel independent learning phases are performed. If we could spot both cases, we could safely run mod-KSS on such inputs, since the expected running time would then be linear: $2^{1-s}\times$ mod-KSS $= O(dn)$ for $b_0 = \log \varepsilon^{-1}$. The new normal phase of the self-improving algorithm is given in Figure 3. Since typicality needs to be enforced only for one center pair, (c_1^*, c_2^*), it can be detected in linear time and $O(\log n)$ space.

Much tougher to handle is hotness detection. Recall non-hotness means that c_1^*, c_2^* is not an ε-solution for I. This could be detected if we had a procedure that estimates the cost of the optimal cost (without actually returning the centers). Fix $(c_1, c_2) \in \mathcal{P}$ and suppose that I is typical but not hot. For notational convenience, we relax the hotness parameter from ε to $O(\varepsilon)$. We reduce the problem to nearest neighbor (NN) searching in L_1 with wildcards. We apply dimension reduction and then use standard NN technology. We begin with the observation that only hotness on the low end of the tail matters: Just check if I is hot for (c_1^*, c_2^*) and go the mod-KSS route if not. Then clearly the only cause of error would be if $\mathtt{OPT}_2\,(I, c_1, c_2)$ lay on the low tail of Z_{c_1,c_2}, for some unknown (c_1, c_2). This reduces hotness detection to a nearest neighbor problem. Indeed, define the "distance" between I and (c_1, c_2) as $\mathtt{OPT}_2\,(I, c_1, c_2)$. This is not a metric, but we can turn it into L_1 with wildcards. Define $v_{c_1,c_2} \in \{0,1\}^{2dn}$ as $(c_1, c_2, \ldots, c_1, c_2)$, and $w_I \in \{0,1\}^{2dn}$ as $(x_1, x_1, x_2, x_2, \ldots, x_n, x_n)$. Given $v, w \in \{0,1\}^{2dn}$, let

$$\Delta(v, w) = \min_{u\in U} \sum_{i=1}^{2dn} u_i d(v_i, w_i) \ ,$$

where U is the set of all $u \in \{0,1\}^{2dn}$ satisfying $\prod_{2dj<i\leq 2dj+d} u_i + \prod_{2dj+d<i\leq 2d(j+1)} u_i = 1$ for $0 \leq j < n$ (the wildcards). Note that $\mathtt{OPT}_2\,(I, c_1, c_2) = \Delta(v_{c_1,c_2}, w_I)$. Next step: dimension reduction. Take a sample R of $r = O(b_0 d/\varepsilon^4 p)$ random points from I and check if the unbiased estimator $\frac{n}{r}\,\mathtt{OPT}_2\,(R, c_1, c_2)$ differs from $\mathbf{E}\, Z_{c_1,c_2}$ by a factor $> 1+\varepsilon$. We declare I non-hot if it does. Applying generalized Chernoff to the worst-case distribution of individual variances shows that detection will fail with probability $\leq 2^{-\Omega(\varepsilon^2 r \mathbf{E}\, Z_{c_1,c_2}/dn)} \leq 2^{-\Omega(\varepsilon^2 rp/d)} \leq 2^{-s}$.

To get rid of the wildcards, we observe that $\Delta(v, w) = \min_{v' \in B_v} d(v', w) - K$, where K is the number of ones among the points of R, and B_v consists of all the points obtained by zeroing out one coordinate in every consecutive pair in v. (By abuse of notation, we still use v_{c_1,c_2} and w_I but think of these vectors as living in $\{0,1\}^{2dr}$.) The problem is to compute $\min_{c_1,c_2} \Delta(v_{c_1,c_2}, w_I) = \min\{\, d(v', w_I) \,|\, v' \in B_{v_{c_1,c_2}}\,\} - K$. This is the nearest neighbor problem in dimension $2dr$ with $2^r|\mathcal{P}| = 2^{O(b_0 d/(\varepsilon^4 p))}\log n$. This can be solved exactly in $O(dr) = O(\frac{b_0 d^2}{\varepsilon^4 p})$ time, using $2^{O(dr)} = 2^{O(b_0 d^2/(\varepsilon^4 p))}$ storage. (The storage can be decreased by using approximate NN searching over the Hamming cube [32], but to deal with the additive K requires a bit of work.) In summary, setting $b_0 = O(\log \varepsilon^{-1})$ ensures a limiting running time of $O(dn)$, using $\exp(\mathrm{poly}(d, p^{-1}, \varepsilon^{-1}))$ storage. (Note that the exponential dependency on ε is to be expected.)

Acknowledgments We wish to thank Loukas Georgiadis, Sariel Har-Peled, Piotr Indyk, Rob Schapire, Bob Tarjan, and Renato Werneck for their helpful comments.

References

[1] S. Albers and J. Westbrook. *Self organizing data structures*, Online Algorithms: The state of the art, 1442:13–41, 1998.

[2] B. Allen and I. Munro. *Self-organizing binary search trees*, Journal of the ACM, 25:526–535, 1978.

[3] N. Alon, S. Dar, M. Parnas, and D. Ron. *Testing of clustering*, Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2000.

[4] N. Alon and J. Spencer. *The probabilistic method*. John Wiley, 2nd edition, 2000.

[5] N. Alon and B. Sudakov. *On two segmentation problems*, Journal of Algorithms, 33:173–184, 1999.

[6] S. Ar, B. Chazelle, and A. Tal. *Self-customized bsp trees for collision detection*, Comput. Geom.: Theory and Applications, 10:23–29, 2000.

[7] M. Badoiu and K. L. Clarkson. *Smaller core-sets for balls*, In Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003.

[8] M. Badoiu, S. Har-Peled, and P. Indyk. *Approximate clustering via core-sets*, In Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), pages 250–257, 2002.

[9] J.L. Bentley and C.C. McGeoch. *Amortized analysis of self-organizing sequential search heuristics*, C. ACM, 28:404–411, 1985.

[10] J.R. Bitner. *Heuristics that dynamically organize data structures*, SIAM J. Comput., 8:82–110, 1979.

[11] A. Blum. *On-line algorithms in machine learning*, Online Algorithms: The state of the art, 1441, 1998.

[12] A. Blum, S. Chawla, and A. Kalai. *Static optimality and dynamic search-optimality in lists and trees*, In Proc. 13th SODA, pages 1–8, 2002.

[13] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*, 1998.

[14] N. Cesa-Bianchi, Y. Freund, D. Haussler, D.P. Helmbold, R.E. Schapire, and M.K. War. *How to use expert advice*, Journal of the ACM, 44(3):427–485, 1997.

[15] W.F. De la Vega, M. Karpinski, C. Kenyon, and Y. Rabani. *Approximation schemes for clustering problems*, In Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), pages 50–58, 2003.

[16] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley & Sons, 2nd edition, 2000.

[17] V. Estivill-Castro and D. Wood. *A survey of adaptive sorting algorithms*, ACM Computing Surveys, 24:441–476, 1992.

[18] A. Flaxman, A. Kalai, and B. McMahan. *Online convex optimization in the bandit setting: gradient descent without a gradient* , In Proc. 16th SODA, 2005.

[19] Fredman, M.L. *How good is the information theory bound in sorting?*, Theoretical Computer Science 1 (1976), 355–361.

[20] Y. Freund and R.E. Schapire. *Adaptive game playing using multiplicative weights*, Games and Economic Behavior, 29:79–103, 1999.

[21] G.H. Gonnet, J.I. Munro, and H. Suwanda. *Exegesis of self-organizing linear search*, SIAM J. Comput., 10:613–637, 1981.

[22] S. Har-Peled. *Clustering motion*, In Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pages 84–93, 2001.

[23] S. Har-Peled and A. Kushal. *Smaller core-sets for k-median and k-means clustering*, In Proceedings of the 21st Annual ACM Symposium on Computational Geometry (SoCG), 2005.

[24] S. Har-Peled and S. Mazumdar. *Core-sets for k-means and k-median clustering and their applications*, In Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), pages 291–300, 2004.

[25] J.H. Hester and D.S. Hirschberg. *Self-organizing linear search*, ACM Comp. Surv., 17:295–311, 1985.

[26] M. Inaba, N. Katoh, and H. Imai. *Applications of weighted voronoi diagrams and randomization to variance-based k-clustering*, In Proceedings of the 10th Annual ACM Symposium on Computational Geometry (SoCG), pages 332–339, 1994.

[27] A. Kalai and S. Vempala. *Efficient algorithms for the online decision problem*, In Proc. 16 COLT, 2003.

[28] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*, MIT Press, 1994.

[29] M. Kearns, R. Schapire, and L. Sellie. *Towards efficient agnostic learning*, Machine Learning 17, pages 115–141, 1994.

[30] J. Kleinberg, C. Papadimitriou, and R. Prabhakar. *Segmentation problems*, In Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC), pages 473–482, 1998.

[31] A. Kumar, Y. Sabharwal, and S. Sen. *A simple linear time (1+ϵ)-approximation algorithm for k-means clustering in any dimension*, In Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pages 454–462, 2004.

[32] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. *Efficient search for approximate nearest neighbor in high dimensional spaces*, SIAM J. Comput. 30 (2000), 457-474.

[33] N. Littlestone. *Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm*, Machine Learning, 2:285–318, 1987.

[34] N. Littlestone and M.K. Warmuth. *The weighted majority algorithm*, Information and Computation, 108(2):212–261, 1994.

[35] J. McCabe. *On serial files with relocatable records*, Operations Research, 13:609–618, 1965.

[36] B. McMahan and A. Blum. *Online geometric optimization in the bandit setting against an adaptive adversary*, In Proc. 17th COLT, 2004.

[37] K. Mehlhorn. *Data structures and algorithms 1: Sorting and searching*, EATCS Monographs on Theoretical Computer Science, 1984.

[38] N. Mishra, D. Oblinger, and L. Pitt. *Sublinear approximate clustering*, In Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2001.

[39] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[40] R. Ostrovsky and Y. Rabani. *Polynomial time approximation schemes for geometric k-clustering*, J. ACM, 49:139–156, 2002.

[41] R. Rivest. *On self-organizing sequential search heuristics*, C. ACM, 19:63–67, 1976.

[42] Albers S. and M. Mitzenmacher. *Average case analyses of list update algorithms*, Algorithmica, 21:312–329, 1998.

[43] D.D. Sleator and R.E. Tarjan. *Amortized efficiency of list update and paging rules*, C. ACM, 28:202–208, 1985.

[44] D.D. Sleator and R.E. Tarjan. *Self-adjusting binary search trees*, J. ACM, 32:652–686, 1985.

[45] F. Takens. *Detecting strange attractors in turbulence*, Dynamical Systems and Turbulence, 898:366–381, 1981.

[46] M. Thorup. *Quick k-median, k-center and facility location for sparse graphs*, In Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP), volume 2076 of Lecture Notes in Computer Science. Springer, 2001.