Parameter-free Locality Sensitive Hashing for Spherical Range Reporting *

Thomas D. Ahle, Martin Aumüller, and Rasmus Pagh

IT University of Copenhagen, Denmark, {thdy, maau, pagh}@itu.dk

December 1, 2016

Abstract

We present a data structure for *spherical range reporting* on a point set S, i.e., reporting all points in S that lie within radius r of a given query point q (with a small probability of error). Our solution builds upon the Locality-Sensitive Hashing (LSH) framework of Indyk and Motwani, which represents the asymptotically best solutions to near neighbor problems in high dimensions. While traditional LSH data structures have several parameters whose optimal values depend on the distance distribution from q to the points of S (and in particular on the number of points to report), our data structure is essentially parameter-free and only takes as parameter the space the user is willing to allocate. Nevertheless, its expected query time basically matches that of an LSH data structure whose parameters have been *optimally chosen for the data and query* in question under the given space constraints. In particular, our data structure provides a smooth trade-off between hard queries (typically addressed by standard LSH parameter settings) and easy queries such as those where the number of points to report is a constant fraction of S, or where almost all points in S are far away from the query point. In contrast, known data structures fix LSH parameters based on certain parameters of the input alone.

The algorithm has expected query time bounded by $O(t(n/t)^\rho)$, where t is the number of points to report and $\rho \in (0, 1)$ depends on the data distribution and the strength of the LSH family used. The previously best running time in high dimensions was $\Omega(tn^\rho)$, achieved by traditional LSH-based data structures where parameters are tuned for outputting a single point within distance r. Further, for many data distributions where the intrinsic dimensionality of the point set close to q is low, we can give improved upper bounds on the expected query time. We finally present a parameter-free way of using multi-probing, for LSH families that support it, and show that for many such families this approach allows us to get expected query time close to $O(n^\rho + t)$, which is the best we can hope to achieve using LSH.

1 Introduction

Range search is a central problem in computational geometry, see e.g. the survey [1]. Given a set S of n points in $\mathbb{R}^d$, the task is to build a data structure that answers queries of the following type: For a region R (from a predefined class of regions), *count* or *report* all points from S that belong to R. Examples for such classes of regions are simplices [25], halfspaces [12], and spheres [9].

In this paper we study the *spherical range reporting (SRR) problem*: Given a distance parameter r and a point set S, build a data structure that supports the following queries: Given a point q, report all points in S within distance r from q. This problem is closely related to *spherical range counting* ("return the number of points") and *spherical range emptiness* ("decide whether there is a point at distance at most r"). Solving spherical range searching problems in time that is truly sublinear in the point set size $n = |S|$ seems to require space exponential in the dimensionality of the point set S. This phenomenon is an instance of the *curse of dimensionality*, and is supported by popular algorithmic hardness conjectures (see [2, 29]).

For this reason, most algorithms for range searching problems involve approximation of distances: For some approximation parameter $c > 1$ we allow the data structure to only distinguish between distance $\leq r$ and $> cr$, while points at distance in between can either be reported or not. We refer to this relaxation as c-approximate SRR. Approximate range reporting and counting problems were considered by Arya et al. in [9], by Indyk in his Ph.D. thesis [19] as "enumerating/counting point locations in equal balls" and by Andoni in his Ph.D. thesis [3] as "randomized R-near

*The research leading to these results has received funding from the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.

neighbor reporting". In low dimensions, tree-based approaches allow us to build efficient data structures with space usage $\tilde{O}(n\gamma^{d-1}(1+(c-1)\gamma^2))$ and query time $\tilde{O}(1/((c-1)\gamma)^{d-1})$ for an approximation factor $1 < c \leq 2$ and trade-off parameter $\gamma \in [1, 1/(c-1)]$, see [9]. The exponential dependency of time and/or space on the dimension makes these algorithms inefficient in high dimensions.

Our approach uses the *locality-sensitive hashing* (LSH) framework [20], which hashes points into some smaller space such that close points are more likely to hash to the same value than distant points. We will introduce in Section 2. Using this technique to solve SRR is not new: Both Indyk [19] and Andoni [3] described extensions of the general LSH framework to solve this problem. As we will show, their approaches yield running times of $\Omega(tn^\rho)$, where t is the number of elements at distance at most cr from the query, and $\rho \in (0,1)$ is a parameter that depends on the distance r, the approximation factor c, and the LSH family used to build the data structure. When the output size t is large this leads to running time $\Omega(n^{1+\rho})$, which is worse than a linear scan! Indyk [19] also describes a reduction from spherical range counting to the (c,r)-approximate near neighbor problem that asks to report a *single* point from the result set of c-approximate SRR. The reduction uses $O(\log^2 n/(c-1)^3)$ queries of independently built (c,r)-ANN data structures, giving a running time of $O(n^\rho \log^2 n/(c-1)^3)$. Building upon Indyk's technique, Chazelle et al. [12] proposed a data structure that solves approximate halfspace range queries on the unit sphere by applying a dimension reduction technique to Hamming space. All of these algorithms use a standard LSH index data structure in a black-box fashion. We propose a data structure that is almost similar to a standard LSH data structure, but query it in an adaptive way. Our guarantees are probabilistic in the sense that each close point is present in the output with constant probability.

Using LSH-based indexes for range reporting means that we report each point closer than distance r with a certain probability, as well as some fraction of the points with distance in the range (r, cr). When c is large, this can have negative consequences for performance: a query could report nearly every point in the data set, and any performance gained from approximation is lost. When the approximation factor c is chosen close to 1, data structures working in high dimensions usually need many independent repetitions to find points at distance r. This is another issue with such indexes that makes range reporting hard: very close points show up in every repetition, and we need to remove these duplicates.

A natural approach to overcome the difficulties mentioned above is to choose the approximation factor c such that the cost of duplicated points roughly equals the cost of dealing with far points. For LSH-based algorithms, many papers explain an offline approach of finding the "optimal" value of c for a data set [4, 10, 15, 28] which usually involves sampling, or making assumptions on the data distribution. However, the best value of c depends not only on the data set, but also on the *query*. This situation is depicted in Figure 1. In this paper, we provide a query algorithm that adapts to the input and finds a near-optimal c at *query time*. We manage to do this in time proportional to the number of points eventually returned for the optimal parameters, making the search essentially free.

Output-sensitivity. To illustrate the improvement over standard fixed parameter LSH, we propose hard data sets for spherical range reporting. In these data sets, we pick $t-1$ very close points that show up in almost every repetition, one point at distance r, and the remaining points close to distance cr. In this case LSH would need $\Theta(n^\rho)$ repetitions to retrieve the point at distance r with constant probability, where e.g. $\rho = 1/c$ in Hamming space [20] and $\rho = 1/c^2$ in Euclidean space [5]. This means that the algorithm considers $\Theta(tn^\rho)$ candidate points, which could be as large as $\Theta(n^{1+\rho})$ for large t. In Section 5 we describe and analyze two algorithms *Adaptive Single-probe* and *Adaptive Multi-probe* that mitigate this problem. The basic idea is that these algorithms "notice" the presence of many close points, and respond by choosing c more lenient, allowing for t far points being reported per repetition in addition to the t close points. This in turn allows us to do only $\Theta((n/t)^\rho)$ repetitions, for a total candidate set of size $\Theta(t(n/t)^\rho)$, which is never larger than n. In general, the number of points between distance r and cr have a linear influence on these running times. This is made precise in Section 4.

Multi-probing. When we stick to the LSH framework, the best running time we could hope for is $\Theta(n^\rho + t)$, giving the optimal output sensitive running time achievable by (data independent) LSH. In order to get closer to this bound, we analyze the *multi-probing approach* for LSH data structures, introduced in [26] and further developed in [24]. The idea is that LSH partitions the space in many buckets, but usually only examines the exact bucket in which the query point falls in each repetition. Multi-probing considers all buckets "sufficiently correlated" with the query bucket to increase the likelihood of finding close points. To our knowledge, multi-probing has always

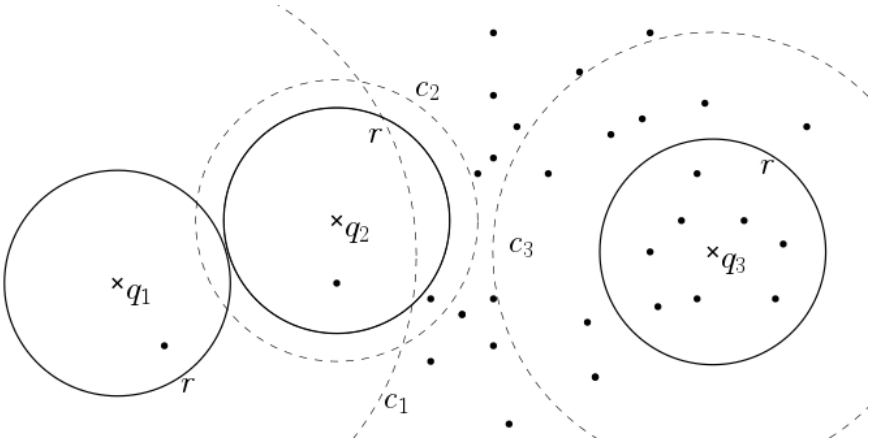

Figure 1: Three queries at radius r centered around points q_1, q_2, and q_3. The dashed circles around the queries contain about twice as many points as the inner solid circles. The ratios between the two radii are called c_1, c_2, and c_3, respectively. Note that the ratios are different across queries on the same dataset. This means that the running time to answer these queries can differ because larger c values allow faster query times.

been applied in order to save space by allowing a smaller number of repetitions to be made and trading this for an increase in query time. Our motivation is different: We want to take advantage of the fact that each of the very close points can only be in one bucket per repetition. Hence by probing multiple buckets in each repetition, *we not only save memory, but also gain a large improvement in the dependency on t in our running time*. We do this by generalizing the adaptive single-probe algorithm to not only find the optimal c for a query, but also the optimal number of buckets to probe. As we show in Section 5, we are able to do this in time negligible compared to the size of the final candidate set, making it practically free. The algorithm works for any probing sequence supplied by the user, but in Section 6 we provide a novel probing sequence for Hamming space and show that it strictly improves the query time compared to the non-multi-probing variant. For large values of t, we show that the running time matches the target time $O(n^\rho + t)$. An overview of the exact running time statements of the algorithms proposed here with a comparison to standard LSH, a linear scan, and the optimal running time for LSH-based algorithms is depicted in Figure 2.

Techniques. The proposed data structure is very similar to a standard LSH data structure as described in [20]. In such a data structure, k locality-sensitive hash functions are concatenated to increase the gap between the collision probability of "close" points and "far away" points. A certain concatenation length k is fixed according to the number of points in the data set, the approximation factor c, and the strength of the hash family at hand. From the value k and the hash family one can compute how many repetitions (using independent hash functions) have to be made to guarantee that a close point is found with, say, constant probability. We give a detailed review of this approach in Section 2. Instead of building the data structure for only one particular k, we build a multi-level variant that encompasses all lengths $1, \ldots, k$ at the same time. At query time, we do an efficient search over the parameter space to find the provably best level. The algorithm then retrieves only the candidates from this level and filters far away points and duplicates. The reason we are able to do an efficient search over the parameter space is that certain parts of the output size can be estimated very quickly when storing the size of the hash table buckets in the LSH data structure. For example, when considering very large c, though the output may be large, there are only few repetitions to check. Gradually decreasing c, which technically means increasing the length k in the LSH data structure, we will eventually have to check so many repetitions that the mere task of iterating through them would be more work than scanning through the smallest candidate set found so far. Since the number of repetitions for each value of c grows geometrically, it ends up being bounded by the last check, which has size not larger than the returned candidate set. For multi-probing it turns out that a similar strategy works, but the search problem is now two-dimensional.

Additional Related Work. Our approach to query-sensitivity generalizes and extends the recent work of Har-Peled and Mahabadi [18] which considers

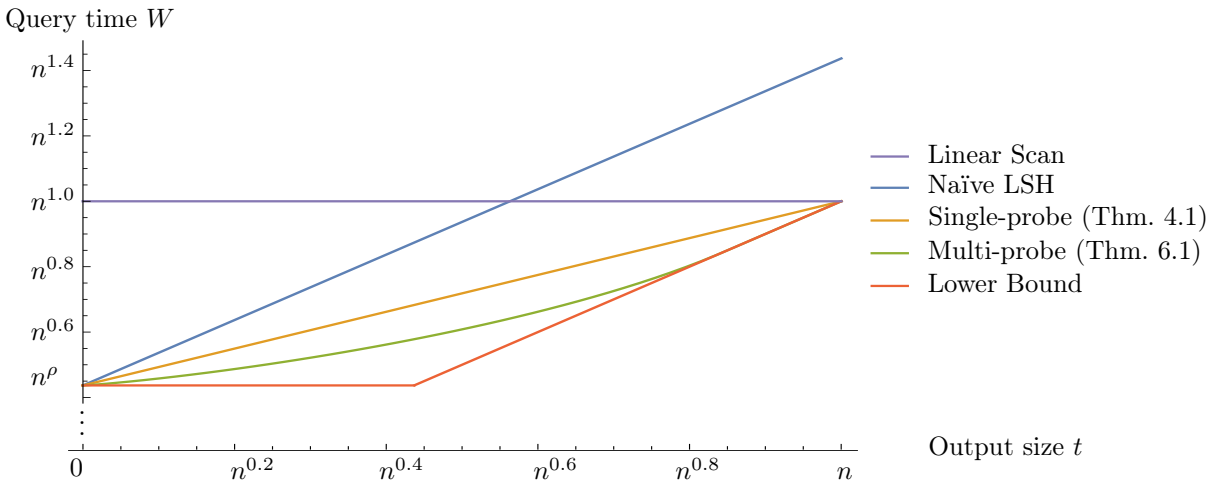

Figure 2: Overview of the running time guarantees of the proposed algorithms "Adaptive Single-probe" and "Adaptive Multi-probe" for collision probabilities $p_1 = 0.8$ and $p_2 = 0.6$ in d-dimensional Hamming space such that $\rho \approx 0.436$. The x-axis shows the output size t as a function of n, the y-axis shows the expected work W, i.e., the expected number of points the algorithm retrieves from the hash tables. For comparison, we plotted the target time of $O(n^\rho + t)$, the running time $O(tn^\rho)$ of a naive LSH approach, and the running time $O(n)$ of a linear scan.

approximate near neighbors. Our method applies to every space and metric supported by the LSH framework while [18] is presented for Hamming space.

The proposed single-probing algorithm can be thought of as an adaptive query algorithm on the trie-based LSH forest introduced by Bawa et al. in [10] for the related approximate k-nearest neighbor problem. The authors of [10] make significant assumptions on the distance distribution of approximate nearest neighbors. The algorithm proposed in [10] always looks at all $n^{f(c)}$ repetitions where $f(c)$ depends on the largest distance r supported by the algorithm and the approximation factor. It collects points traversing tries synchronously in a bottom-up fashion. By looking closer at the guarantees of LSH functions, we show that one can gradually increase the number of repetitions to look at and find the best level to query directly. We hope that the insights provided here will shed new light on solving approximate nearest neighbors beyond using standard reductions as in [17]. We note that very recent work of Andoni et al. [**?**] provides new guarantees for LSH forest in worst case settings — it would be interesting to see if this could lead to improvements of our results.

Combining results of very recent papers [23, 13, 7] on space/time-tradeoffs makes it possible to achieve running times that are asymptotically similar to our results with respect to multi-probing. We give a detailed exposition in Appendix E.

2 Preliminaries

Our data structures can be implemented in a standard model of computation that allows unit cost retrieval of a memory word. For simplicity we will assume that hash function evaluation as well as distance computation can also be done in unit time — results in more general settings follows by reduction.

Let (X, dist) be a metric space over X with distance function $\mathrm{dist}\colon X^2 \to \mathbb{R}$. In this paper, the space usually does not matter; only the multi-probing sequence in Section 6 is tied to Hamming space.

Definition 1. (Spherical Range Reporting) *Given a set of points $S \subseteq X$ and a number $r \geq 0$, construct a data structure that supports the following queries: Given a point $q \in X$, report each point $p \in S$ with $\mathrm{dist}(p, q) \leq r$ with constant probability.*

Note that we consider the exact version of SRR, where distances are not approximated, but allow point-wise probabilistic guarantees. Of course, repeating an algorithm that solves SRR $\Theta(\log |S|)$ times yields an algorithm that outputs each close point with high probability.

Definition 2. (LSH Family, [11]) *A locality-sensitive hash (LSH) family $\mathcal{H}$ is family of functions $h\colon X \to R$, such that for each pair $x, y \in X$ and a random $h \in \mathcal{H}$, for arbitrary $q \in X$, whenever $\mathrm{dist}(q, x) \leq \mathrm{dist}(q, y)$ we have* $\Pr[h(q) = h(x)] \geq \Pr[h(q) = h(y)]$.

Usually the set R is small, like the set $\{0,1\}$. Often we will concatenate multiple independent hash functions from a family, getting functions $h_k\colon X \to R^k$. We call this a *hash function at level k*.

Having access to an LSH family $\mathcal{H}$ allows us to build a data structure with the following properties.

THEOREM 2.1. ([17, THEOREM 3.4]) *Suppose that for some metric space (X, dist) and some factor $c > 1$, there exists an LSH family such that $\Pr[h(q) = h(x)] \geq p_1$ when $\mathrm{dist}(q,x) \leq r$ and $\Pr[h(q) = h(x)] \leq p_2$ when $\mathrm{dist}(q,x) \geq cr$ with $p_1 > p_2$. Then there exists a data structure such that for a given query q, it returns with constant probability a point within distance cr, if there exists a point within distance r. The algorithm uses $O(dn + n^{1+\rho})$ space and evaluates $O(n^\rho)$ hash functions per query, where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$.*

It is essential for understanding our algorithms to know how the above data structure works. For the convenience of the reader we provide a description of the proof next.

Proof. Given access to an LSH family $\mathcal{H}$ with the properties stated in the theorem and two parameters L and k (to be specified below), repeat the following process independently for each i in $\{1, \ldots, L\}$: Choose k hash functions $g_{i,1}, \ldots, g_{i,k}$ independently at random from $\mathcal{H}$. For each point $p \in S$, we view the sequence $h_i(p) = (g_{i,1}(p), \ldots, g_{i,k}(p)) \in R^k$ as the hash code of p, identify this hash code with a bucket in a table, and store a reference to p in bucket $h_i(p)$. To avoid storing empty buckets from R^k, we resort to hashing and build a hash table T_i to store the non-empty buckets for S and h_i.

Given a query $q \in X$, we retrieve all points from the buckets $h_1(q), \ldots, h_L(q)$ in tables $T_1, \ldots, T_L$, respectively, and report a close point in distance at most cr as soon as we find such a point. Note that the algorithm stops and reports that no close points exists after retrieving more than $3L$ points, which is crucial to guarantee query time $O(n^\rho)$.

The parameters k and L are set according to the following reasoning. First, set k such that it is expected that at most one distant point at distance at least cr collides with the query in one of the repetitions. This means that we require $np_2^k \leq 1$ and hence we define $k = \lceil \frac{\log n}{\log(1/p_2)} \rceil$. To find a close point at distance at most r with probability at least $1 - \delta$, the number of repetitions L must satisfy $\delta \leq (1 - p_1^k)^L \leq \exp(-p_1^k \cdot L)$. This means that L should be at least $p_1^{-k} \ln \delta$ and simplifying yields $L = O(n^\rho)$. Note that these parameters are set to work even in a worst-case scenario where there is exactly one point at distance p and all other points have distance slightly larger than cr. □

For ease of presentation, we assume that the collision probability p_1 is $\Theta(1)$ throughout the paper.

The LSH framework can easily be extended to solve SRR. We just report all the points that are in distance at most r from the query point in the whole candidate set retrieved from all tables $T_1, \ldots, T_L$ [3]. For the remainder of this paper, we will denote the number of points retrieved in this way by W ("work"). It is easy to see that this change to the query algorithm would already solve SRR with the guarantees stated in the problem definition. However, we will see in Section 4 that its running time might be as large as $O(n^{1+\rho})$, worse than a linear scan over the data set.

3 Data Structure

We extend a standard LSH data structure in the following way.

Assume we are given a set $S \subseteq X$ of n points, two parameters r and L, and access to an LSH family $\mathcal{H}$ that maps from X to R. Let $\mathrm{reps}(k) = \lceil p_1^{-k} \rceil$ where p_1 is a lower bound on the probability that points at distance r collide under random choice of $h \in \mathcal{H}$. Let K be the largest integer such that $\mathrm{reps}(K) \leq L$. A *Multi-level LSH data structure* for S is set up in the following way: For each $k \in \{0, \ldots, K\}$ choose functions $g_{k,i}$ for $1 \leq i \leq \mathrm{reps}(k)$ from $\mathcal{H}$ independently at random. Then, for each $k \in \{0, \ldots, K\}$, build $\mathrm{reps}(k)$ hash tables $T_{k,i}$ with $1 \leq i \leq \mathrm{reps}(k)$. For a fixed pair $k \in \{0, \ldots, K\}$ and $i \in \{1, \ldots, \mathrm{reps}(k)\}$, and each $x \in X$, concatenate hash values $(g_{1,i}(x), \ldots, g_{k,i}(x)) \in R^k$ to obtain the hash code $h_{k,i}(x)$. Store references to all points in S in table $T_{k,i}$ by applying $h_{k,i}(x)$. For a point $x \in X$, and for integers $0 \leq k \leq K$ and $1 \leq i \leq \mathrm{reps}(k)$, we let $|T_{k,i}(x)|$ be the number of points in bucket $h_{k,i}(x)$ in table $T_{k,i}$. We store this value so it can be retrieved in constant time. In contrast to a standard LSH data structure, we only accept the number of repetitions, i.e., the allowed space to build the data structure, as an additional parameter. The value K is chosen such that the number of repetitions available suffices to obtain a close point at distance r with constant probability, cf. the proof of Theorem 2.1. This is ensured by the repetition count for all levels $0, \ldots, K$. The space usage of our data structure is $O(n \sum_{0 \leq k \leq K} p_1^{-k}) = O(np_1^{-K}) = O(nL)$. Hence multiple levels only add a constant overhead to the space consumption compared to a standard LSH data structure for level K. Figure 3 provides a visualization of the data structure.

We describe an alternative tree-based data structure that trades query time for space consumption in Appendix A.

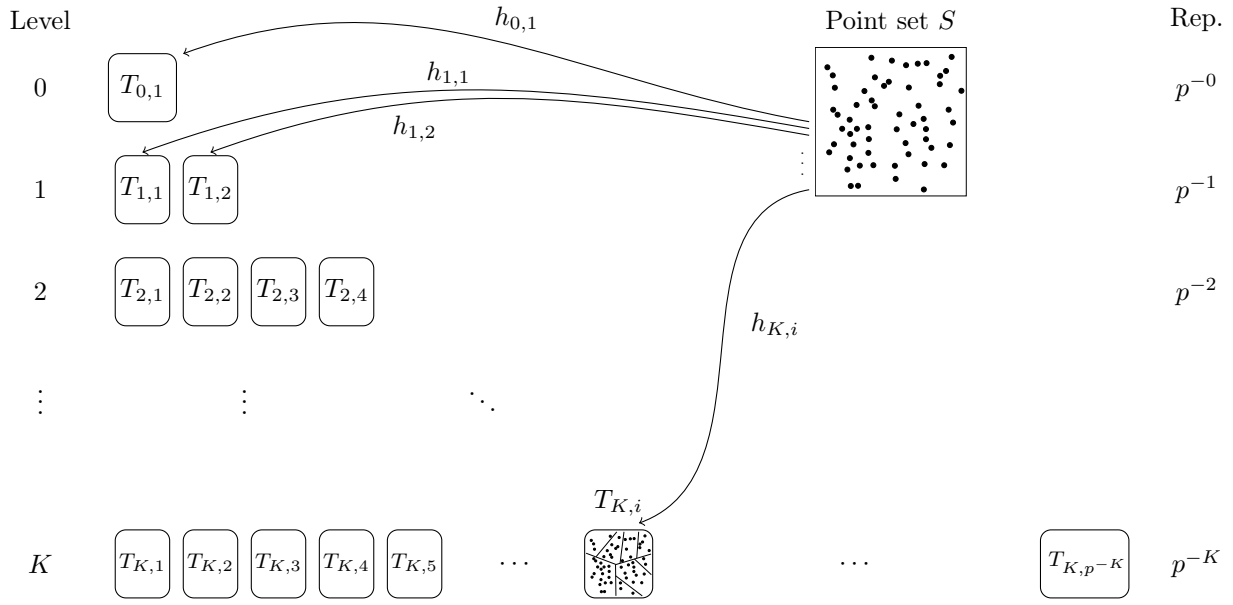

Figure 3: Overview of the multi-level LSH data structure with tables $T_{k,i}$ and hash functions $h_{k,i}$ splitting a data set S. The data structure is set up for levels $0, ..., K$ with repetition count "Rep." for collision probability $p = 1/2$. Example for a space partition of S induced by hash function $h_{K,i}$ is explicitly depicted as the content of table $T_{K,i}$ where each class is a bucket.

4 Standard LSH, Local Expansion, and Probing the Right Level

In this section we show that using a standard LSH approach can yield running time $\Omega(tn^\rho)$ when standard parameter settings such as the ones in the proof of Theorem 2.1 are used to solve SRR. Then, we define a measure for the difficulty of a query. Finally, we show that if the output size and this measure is known, inspecting a certain level in the multi-level LSH data structure gives output- and query-sensitive expected running times.

Suppose we want to solve SRR using an LSH family $\mathcal{H}$. Assume that the query point $q \in X$ is fixed. Given n, t with $1 \leq t \leq n$, and $c > 1$, we generate a data set S by picking

- $t-1$ points at distance ϵ from q, for ϵ small enough that even concatenating $\lceil \frac{\log n}{\log 1/p_2} \rceil$ hash functions from $\mathcal{H}$, we still have collision probability higher than 0.01,
- one point $x \in X$ with $\text{dist}(q, x) = r$,
- the remaining $n - t$ points at distance cr.

We call a set S that is generated by the process described above a *t-heavy input for SRR on q*. By definition, a t-heavy input has expansion c at query point q. We argue that the standard LSH approach is unnecessarily slow on such inputs.

Observation 1. *Fix two values n and $t \leq n$, and let $q \in X$ be a fixed query point. Let S with $|S| = n$ be a t-heavy input generated by the process above. Suppose we want to solve SRR in (X, dist) using the LSH data structure from the proof of Theorem 2.1 with LSH family $\mathcal{H}$ build for S. Then the expected number of points retrieved from the hash tables on query q in the LSH data structure is $\Theta(tn^\rho)$.*

Proof. The standard LSH data structure is set up with $k = \lceil \frac{\log n}{\log 1/p_2} \rceil$ and $L = O(n^\rho)$. L repetitions are necessary to find the close point at distance r with constant probability. By the construction of S, each repetition will contribute at least $\Theta(t)$ very close points in expectation. So, we expect to retrieve $O(tn^\rho)$ close points from the hash tables in total. □

The process described above assumes that the space allows us to pick sufficiently many points at a certain distance. This is for example true in $\mathbb{R}^d$ with Euclidean distance. In Hamming space $\{0,1\}^d$ we would change the above process to enumerate the points from distance $1, 2, \ldots$ and distance $cr + 1, cr + 2, \ldots$. If d and r are sufficiently large, the same observation as above also holds for inputs generated according to this process.

For a set S of points, a point q, and a number $r > 0$, let $N_r(q)$ be the number of points in S at distance at most r from q. We next define the *expansion* at a query point q for a given distance. The expansion measures how far we can increase the radius of an r-sphere around the query point before the number of points before the number of points covered more than doubles. This dimensionality measure is central in the running time analysis of our proposed algorithms.

DEFINITION 3. (EXPANSION) *Let $r > 0$, $q \in X$ and $S \subseteq X$ be a set of points. The* expansion $c^*_{q,r}$ *at point q is the largest number c such that $N_{cr}(q) \leq 2N_r(q)$, where $c^*_{q,r}$ is ∞ if $N_r(q) \geq n/2$.*

We will often simply write c^*_q, when r is known in the context. A visualization for the expansion around a query is shown in Figure 1.

4.1 Query Algorithms If t and c^*_q are Known

In the following, we state the parameter ρ as used in Theorem 2.1 as a function $\rho(r, c)$ such that $\rho(r, c) = \frac{\log(1/p(r))}{\log(1/p(cr))}$, where $p(\Delta)$ is the probability that two points at distance Δ collide. (The probability is over the random choice of the LSH function.) We omit the parameters when their value is clear from the context. The following statements use the Multi-Level LSH data structure introduced in Section 2.

THEOREM 4.1. *Let $r > 0$ and $c \geq 1$. Let S be a set of n points and let DS be the Multi-level LSH data structure for S using $L = \Omega(n^{\rho(r,c)})$. Given a query point q, let $t = N_r(q)$ and c^*_q be the expansion around q in S. Then there exists a query algorithm on DS to solve SRR with the following properties:*

(i) If $c^*_q \geq c$, the algorithm has expected running time $O(t(n/t)^{\rho(r,c^*_q)})$.

(ii) Otherwise, the algorithm has expected running time $O(t(n/t)^{\rho(r,c)} + N_{cr}(q))$.

For $t = 0$, the running time is the same as $t = 1$.

Proof. Let p_1 and p_2 be the probabilities that the query point collides with points at distance r and $c^*_q r$, respectively, given the LSH family used. We consider statement (i) first. Set $k = \lceil \frac{\log(n/t)}{\log(1/p_2)} \rceil$ and note that $p_2^k \leq t/n$ and $p_1^{-k} = \Theta((n/t)^{\rho(r,c^*_q)})$. The algorithm works by inspecting all points in the query buckets in tables $T_{k,1}, \ldots, T_{k,p_1^{-k}}$. This repetition count guarantees constant probability of finding each close point and since $c^*_q \geq c$, we can assume $p_1^{-k} \leq L$. In each repetition, we expect collisions with not more than t close points, $N_{c^*_q r}(q)p_1^k$ points at distance at most $c^*_q r$ and np_2^k far points. By linearity of expectation, the expected number of collisions in all buckets is then not more than $p_1^{-k}(t + N_{c^*_q r}(q)p_1^k + np_2^k)$. By the choice of k, $np_2^k \leq t$ and so this is $O(tp_1^{-k} + N_{c^*_q r}(q))$. By the definition of c^*_q, $N_{c^*_q r}(q) = O(t)$ which means that this term is dominated by the former. Finally looking at every bucket takes time $O(p_1^{-k})$, but this is likewise dominated if $t \geq 1$. Statement (ii) follows by the same line of reasoning, simply using c instead of c^*_q. Since this value of c does not have the expansion property, the term $N_{cr}(q)$ is present in the running time. □

The running time bounds depend on the number of points at distance at most cr. This is inherent to our approach when the expansion around the query is smaller than the c value that can be read off from the number of repetitions and the LSH hash family at hand. The influence of these points is however only linear in their number.

Theorem 4.1 basically shows that there exists a single level of the multi-level LSH data structure that we want to probe when the number of close points t and the expansion around the query are known.

5 Adaptive Query Algorithms

In this section we describe a query algorithm that (with small overhead) obtains the results from Theorem 4.1 without knowing t or the expansion around the query. It turns out that we can get something even better, namely, a running time equivalent to the best over all choices for k and the number of repetitions if we were to know the entire distribution of distances $\text{dist}(q, x)$ from the query point q to data points x in the data set.

We work on the multi-level LSH data structure DS set up for $S \subseteq X$ with tables $T_{k,i}$ as introduced in Section 3. DS is assumed to have been built with L repetitions and K levels.

Suppose a query algorithm inspects the buckets at level k. Let W_k denote the sum of the number of points retrieved from these buckets and the number of buckets inspected. By linearity of expectation, the expected work $\mathrm{E}[W_k]$ is

$$\mathrm{E}[W_k] = p_1^{-k}(1 + \sum_{x \in S} \Pr[h_k(q) = h_k(x)]), \tag{5.1}$$

because the algorithm considers p_1^{-k} repetitions (we omit ceilings for ease of presentation) and the expected number of collision within one repetition is $\sum_{x \in S} \Pr[h_k(q) = h_k(x)]$.

The function $\mathrm{E}[W_k]$ over k will have a minimum in $[0, K]$. We denote this minimum by W_{single}, i.e.,

we set

$$W_{\text{single}} = \min_{0 \le k \le K} \mathrm{E}[W_k]. \tag{5.2}$$

W_{single} denotes the expected work on the "optimal level" of the data structure for the query point, in the sense that we expect to check the fewest points while guaranteeing to find each close point with constant probability. We refer to it with the subscript "single" to emphasize that only a single bucket is checked in each repetition. Our algorithmic goal is to describe an algorithm that finds this level with only small overhead. Since W_{single} describes the minimum expected work knowing the distance distribution *from the query point*, we note that this quantity is always upper bounded by running times stated in Theorem 4.1. However, in many important cases it can be much smaller than that. To make this precise, we calculate W_{single} for different data distributions, including "locally growth-restricted" as considered in [14]. In this case it turns out that $W_{\text{single}} = O(\log n)$, an exponential improvement over the "standard" query time $O(n^\rho)$. Details are provided in Appendix B.

The query algorithm is given as Algorithm 1 and works as follows: For each level $0 \le k \le K$, calculate the work of doing $\text{reps}(k) = \lceil 2p_1^{-k} \log 2k \rceil$ repetitions by summing up all bucket sizes. (The $2 \log 2k$ factor is a technical detail explained in the proof below.) Terminate as soon as the optimal level has been provably found, which may be one that we have considered in the past, and report all close points in the candidate set. The decision whether to consider a new level is based on whether the number of buckets to look at is larger than the smallest candidate set found so far.

THEOREM 5.1. *Let r and $S \subseteq X$ with $|S| = n$ be given. Then Algorithm 1 solves SRR with pointwise constant probability. The expected running time of the **while**-loop in lines (2)–(6) and the expected number of distance computations in line (7) is $O(W_{single} \log\log W_{single})$, where*

$$W_{\text{single}} = \min_{0 \le k \le K} \left[p_1^{-k} (1 + \sum_{x \in S} \Pr[h_k(q) = h_k(x)]) \right].$$

This theorem says that Algorithm 1 finds the best level for the query point at hand with only small overhead compared to an algorithm that would have been told by some oracle which level of the data structure has minimum expected work for the query.

Proof. First we show that the algorithm works correctly, then we argue about its running time.

For correctness, let $y \in S$ be a point with $\text{dist}(y, q) \le r$. We want to show that with constant probability, y is present in one of the buckets the algorithm inspects in line 7. Since y is present in any particular repetition with probability p_1^k, intuitively making $1/p_1^k$ repetitions should suffice for correctness. However, our case is a bit more delicate. The probability of y being present in $\bigcup_i T_{k,i}$ is p_1^k if we do not know anything about the data. However, for the choice of k used by the algorithm, $\bigcup_i T_{k,i}$ is particularly small. The expected size of $\bigcup_i T_{k,i}$ is larger conditioned on y being present, and so equivalently if we choose the k that minimizes $\bigcup_i T_{k,i}$ we also decrease the chance of y finding there. It is seems difficult to handle this correlation directly, so we take a different approach here. We prove that y is present for every k we may choose with sufficiently high probability before the algorithm inspects bucket sizes. This requires us to choose a slightly higher number of repetitions, in particular $\text{reps}(k)$ is such that the probability of finding y is at least $1 - (1 - p_1^k)^{\text{reps}(k)} \ge 1 - 1/(2k)^2$. By a union bound over the K levels of the data structure, y can be found on every level with probability at least $1 - \sum_{k=1}^{\infty} 1/(2k)^2 \ge 1/2$, which shows correctness.

Now we consider the running time. The work inside the loop is dominated by line (3) which takes time $O(\text{reps}(k))$, using constant time access to the size of the buckets. Say the last value k before the loop terminates is k^*, then the loop takes time $\sum_{k=1}^{k^*} O(\log k \cdot p_1^{-k}) \le \log k^* \cdot p_1^{-k^*} \sum_{k=0}^{\infty} O(p_1^k) = O(\log k^* \cdot p_1^{-k^*}) = O(w_{\text{best}})$, where the last equality is by the loop condition, $\text{reps}(k^*) \le w_{\text{best}}$.

In line 7, the algorithm looks at w_{best} points and buckets.

Need p_1 to be a constant here, and not $1 - o(1)$.

Hence the total expected work is

$$\begin{aligned} \mathrm{E}[w_{\text{best}}] &= \mathrm{E}\left[\min_{0 \le k \le K} w_k\right] \\ &\le \min_{0 \le k \le K} \mathrm{E}[w_k] \\ &= O(\min_{0 \le k \le K} \log k \cdot \mathrm{E}[W_k]) \\ &= O(\log k' \cdot \mathrm{E}[W_{k'}]) \\ &= O(W_{\text{single}} \log\log W_{\text{single}}). \end{aligned} \tag{5.3}$$

Here the first inequality follows by Jensen's inequality using the concavity of the min function. We bound the minimum over k by choosing a concrete value $k' = \arg\min_{0 \le k \le K} \mathrm{E}[W_k]$. Finally, we bound k' by $p_1^{-k'} \le W_{\text{single}}$. □

Algorithm 1 Adaptive-Single-Probe(q, p_1, T)

1: $k \leftarrow 1, k_{\text{best}} \leftarrow 0, w_{\text{best}} \leftarrow n$;
2: **while** $\text{reps}(k) \leq \min(L, w_{\text{best}})$ **do**
3: $\quad w_k \leftarrow \sum_{i=1}^{\text{reps}(k)} (1 + |T_{k,i}(q)|)$;
4: $\quad$ **if** $w_k < w_{k_{\text{best}}}$ **then**
5: $\quad\quad k_{\text{best}} \leftarrow k; w_{\text{best}} \leftarrow w_k$;
6: $\quad k \leftarrow k + 1$;
7: **return** $\bigcup_{i=1}^{\text{reps}(k_{\text{best}})} \{x \in T_{k_{\text{best}},i}(q) \mid \text{dist}(x, q) \leq r\}$

Adaptive query algorithm on the Multi-level LSH data structure from Section 3 set up with hash tables $T_{k,i}$ with $0 \leq k \leq K$ and $1 \leq i \leq \text{reps}(k) = \lceil 2p_1^{-k} \log 2k \rceil$. We denote with $|T_{k,i}(q)|$ the size of the bucket the query point q is hashed to in the i-th repetition on level k.

5.1 A Multi-probing Version of Algorithm 1

A common technique when working with LSH is multi-probing [24, 26, 15, 6, 21]. The idea is that often the exact bucket $h_k(q)$ does not have a much higher collision probability with close points than some "nearby" bucket $\sigma(h_k(q))$. To be more precise, for a hash function $h_k \colon X \to R^k$, we define a *probing sequence* $\sigma = (\sigma_{k,\ell})_{\ell \geq 1}$ as a sequence of functions $R^k \to R^k$. Now when we would probe bucket $T_{k,i}(h_k(q))$, we instead probe $T_{k,i}(\sigma_{k,1}(h_k(q))), T_{k,i}(\sigma_{k,2}(h_k(q))), \ldots$. (Where $\sigma_{k,1}$ will usually be the identity function.)

For a point y at distance r from q, we will be interested in the event $[\sigma_{k,\ell}(h_k(q)) = h_k(y)]$. The probability that this event occurs is donated by $p_{k,\ell}$. We say that a probing sequence σ is *reasonable*, if (i) for each k we have $p_{k,1} \geq p_{k,2} \geq \ldots$ and (ii) for each ℓ we have $p_{k,\ell} \geq p_{k+1,\ell}$.[1] The intuition is that we probe buckets in order of probability of collision with the query point. In particular, by disjointness of the events, the probability of a collision within the first ℓ probes is exactly

$$P_{k,\ell} = p_{k,1} + \cdots + p_{k,\ell}. \tag{5.4}$$

This means that with ℓ probes per repetition, it suffices to repeat $1/P_{k,\ell}$ times to obtain constant probability of finding y.

To state the complexity of our algorithm, we generalize the quantities W_k and W_{single} from (5.2) in the natural way to include multi-probing. We let $W_{k,\ell}$ denote the sum of the number points retrieved from the buckets plus the number of buckets inspected for each parameter pair (k, ℓ). By linearity of expectation, the expected work for (k, ℓ) is then

$$\mathrm{E}[W_{k,\ell}] = \frac{1}{P_{k,\ell}} \left(\ell + \sum_{\substack{x \in S \\ 1 \leq i \leq \ell}} \Pr[\sigma_{k,i}(h(q)) = h(x)] \right)$$

Analogously to the single-probing approach, we let W_{multi} denote the minimal work one would expect to need for an LSH based approach that knows the optimal values of k and ℓ:

$$W_{\text{multi}} = \min_{\substack{0 \leq k \leq K \\ \ell \geq 1}} \mathrm{E}[W_{k,\ell}] \tag{5.5}$$

Incorporating multi-probing into Algorithm 1 is done by carefully searching through the now two-dimensional, infinite space $[0, K] \times [1, \infty]$ of parameters. The algorithm is given as Algorithm 2 and works as follows: For parameter pairs (k, ℓ) we define two functions

$$\text{reps}(k, \ell) = \lceil 2 \log(2\ell k)/P_{k,\ell} \rceil \quad \text{and} \tag{5.6}$$

$$\text{cost}(k, \ell) = \ell \cdot \text{reps}(k, \ell), \tag{5.7}$$

where $\text{reps}(k, \ell)$ is the repetition count analogous to the first algorithm, and $\text{cost}(k, \ell)$ represents the number of buckets that need to be inspected for the parameter pair. The parameter space is explored in order of this cost.

For each considered parameter pair (k, ℓ), the actual work $w_{k,\ell}$ of inspecting the candidate set $\bigcup_{i=1}^{\text{reps}(k,\ell)} \bigcup_{j=1}^{\ell} T_{k,i}(\sigma_{k,j}(h_k(q)))$ is calculated by summing up the bucket sizes plus the number of buckets. The algorithm keeps track of the size of the smallest work load seen so far, and it terminates as soon as this size is smaller than the smallest cost value of all parameter pairs not considered so far. We note that to obtain good query time, we have to compute the work loads $w_{k,\ell}$ in line 9 of Algorithm 2 slightly differently. The exact replacement is discussed in the proof below.

[1] As long as the collision probabilities $p_{k,\ell}$ are known or can be estimated accurately enough, an arbitrary probing sequence can be made reasonable by sorting it. All probing sequences we know of from the literature follow this approach, see [6] for an example.

Algorithm 2 Adaptive-Multi-probe(q, σ, T)

$w_{\text{best}} \leftarrow n$; $k_{\text{best}} \leftarrow 0$; $\ell_{\text{best}} \leftarrow 1$
PQ $\leftarrow$ empty priority queue ▷ Manages pairs (k, ℓ) with priority $\text{cost}(k, \ell) = \ell \cdot \text{reps}(k, \ell)$.
PQ.insert($(1, 1)$)
while PQ.min() $< w_{\text{best}}$ **do**
  $(k, \ell) \leftarrow$ PQ.extractMin()
  if $k < K$ and $\ell = 1$ **then**
    PQ.insert($(k + 1, 1)$)
  PQ.insert($(k, \ell + 1)$)
  $w_{k,\ell} \leftarrow \sum_{i=1}^{\text{reps}(k,\ell)} \sum_{j=1}^{\ell} (1 + |T_{k,i,j}(q)|)$
  if $w_{k,\ell} < w_{\text{best}}$ **then**
    $k_{\text{best}} \leftarrow k$; $\ell_{\text{best}} \leftarrow \ell$; $w_{\text{best}} \leftarrow w_{k,\ell}$
return $\bigcup_{i=1}^{\text{reps}(k_{\text{best}},\ell_{\text{best}})} \bigcup_{j=1}^{\ell_{\text{best}}} \{x \in T_{k_{\text{best}},i,j}(q) \mid \text{dist}(x, q) \le r\}$

Adaptive multi-probing query algorithm on the Multi-Level LSH data structure from Section 3 with K levels and $\text{reps}(k, \ell) = \lceil 2 \log(2\ell k)/P_{k,\ell} \rceil$. For clarity, we write $T_{k,i,j}(q) = T_{k,i}(\sigma_{k,j}(h_{k,i}(q)))$ for the jth bucket in the sequence at level k, repeititon i. The algorithm scans the parameter space $[0, K] \times [1, \infty)$ in order of $cost(k, \ell)$ starting at $(0, 1)$. The PQ.min function returns the smallest priority in the priority queue.

THEOREM 5.2. *Let $S \subseteq X$ and r be given. Let (k, ℓ) be a pair that minimizes the right-hand side of (5.5). Given a reasonable probing sequence σ, Algorithm 2 on DS solves SRR with point-wise constant probability. If DS supports at least reps(k, ℓ) repetitions, the expected running time is $O(W_{multi} \log^3 W_{multi})$ and the expected number of distance computations is $O(W_{multi} \log W_{multi})$, where*

$$W_{multi} = \min_{\substack{0 \le k \le K \\ \ell \ge 1}} \left[\frac{1}{P_{k,\ell}} \left(\ell + \sum_{\substack{x \in S \\ 1 \le i \le \ell}} \Pr[\sigma_{k,i}(h(q)) = h(x)] \right) \right]$$

Proof. We first show correctness and then bound the running time of the algorithm.

To show the correctness of the algorithm, let $y \in S$ be an arbitrary point at distance at most r from the query point q. Similarly to the proof of Theorem 5.1, we show that the algorithm is correct for all parameter pairs simultaneously. For each pair (k, ℓ) considered by the algorithm, point y is found within the first ℓ probed buckets in $T_{k,i}$ with probability at least $P_{k,\ell}$ for $1 \le i \le \text{reps}(k, \ell)$ by (5.4). With $\text{reps}(k, \ell)$ repetitions, the probability of finding y is at least $1 - (1 - P_{k,\ell})^{\text{reps}(k,\ell)} \ge 1 - (2\ell k)^{-2}$ using the definition from (5.6). A union bound over the whole parameter space then yields $\sum_{k=1}^{\infty} \sum_{\ell=1}^{\infty} (2\ell k)^{-2} < 7/10$, and so y can be found in one of the probed buckets for every parameter choice with constant probability, which shows correctness.

To analyze the running time, we consider the value $w_{\text{best}} = w_{k_{\text{best}},\ell_{\text{best}}}$ found in the loop. This value bounds the number of distance computations and we will show that the number of operations done in the while-loop can similarly be bounded by $O(w_{\text{best}} \log^2 w_{\text{best}})$. We show that w_{best} is the smallest among all $w_{k,\ell}$ in the parameter space, which will allow us to bound $\text{E}(w_{\text{best}}) = O(W_{\text{multi}} \log W_{\text{multi}})$ as in the theorem.

Starting from the end, consider any pair (k, ℓ) in the parameter space. If the algorithm actually inspected this pair, we must have $w_{k,\ell} \ge w_{\text{best}}$, since we always keep the smallest value seen. If the pair was not inspected by the algorithm, let (k', ℓ') be the pair of smallest cost left in the priority queue when the loop breaks. Then we will show

$$w_{\text{best}} \le \text{cost}(k', \ell') \le \text{cost}(k, \ell) \le w_{k,\ell}$$

proving minimality. Recall that $\text{cost}(k, \ell)$ from (5.7) is used as the priority of a parameter pair in the priority queue. The loop condition thus directly gives us the first inequality. For the third inequality note that $\text{cost}(k, \ell)$ denotes the number of buckets associated with a parameter pair (k, ℓ). Since $w_{k,\ell}$ is the number of these buckets plus the points points inside them, we get the inequality.

Finally for the second inequality, we will show that $\text{cost}(k, \ell)$ is monotone in k as well as in ℓ. Since (k, ℓ) was not considered by the algorithm, it must have either higher k or ℓ than some pair in the priority queue while the other parameter is the same, and so by monotonicity its cost is higher, giving the inequality.

The function $\text{cost}(k, \ell)$ is monotone in k because $p_{k,\ell} \geq p_{k+1,\ell}$ for reasonable probing sequences as defined. This implies $P_{k+1,\ell} \leq P_{k,\ell}$ and so $1/P_{k,\ell}$, $\text{reps}(k, \ell)$ and $\text{cost}(k, \ell)$ are all monotonically increasing in k.

For ℓ we have to be a bit more careful, since $1/P_{k,\ell}$ is decreasing in ℓ. This is of course because doing more multiprobing increases our chances of finding y, and so we have to do fewer independent repetitions. Luckily $\ell/P_{k,\ell}$ is monotonically increasing in ℓ, and so $\text{cost}(k, \ell)$ is as well. The proof for this is given in Appendix C.

Next we bound the running time of the loop. The way line 9 is written in the figure, the loop actually takes far too much time. When computing a new value $w_{k,\ell+1}$, we instead take advantage of the work $w_{k,\ell}$ already discovered, and only consider the number of buckets that are new or no longer needed. Specifically, we may compute

$$w_{k,\ell+1} = w_{k,\ell} + \sum_{i=1}^{\text{reps}(k,\ell+1)} |T_{k,i,\ell+1}(q)| - \sum_{j=1}^{\ell} \sum_{i=1+\text{reps}(k,\ell)}^{\text{reps}(k,\ell+1)} |T_{k,i,j}(q)|. \tag{5.8}$$

Using this calculation, we consider each bucket size at most twice (once when added, and once when subtracted). Thus, computing the values $w_{k,1}, \ldots, w_{k,\ell}$ takes time at most $2(\text{reps}(k, 1) + \cdots + \text{reps}(k, \ell))$.

Let k^* be the largest value such that a parameter pair (k^*, ℓ) was visited. Similarly, for each k let ℓ_k be the largest value such that the pair (k, ℓ_k) was visited. Then the total time spent is no more than

$$\begin{aligned}\sum_{k=1}^{k^*} 2 \sum_{\ell=1}^{\ell_k} \text{reps}(k, \ell) &\leq 4 \sum_{k=1}^{k^*} \log(2k\ell_k) \sum_{\ell=1}^{\ell_k} 1/P_{k,\ell_k} \\ &= \sum_{k=1}^{k^*} (\log k\ell_k)\, O(\ell_k(\log \ell_k)/P_{k,\ell_k}) \\ &= \sum_{k=1}^{k^*} (\log \ell_k)\, O(\text{cost}(k, \ell_k)), \qquad (5.9)\end{aligned}$$

where we used Lemma C.1 to bound the sum over $1/P_{k,\ell_k}$. Now, observe that by the loop condition we know that $\text{cost}(k, \ell_k) \leq w_{\text{best}}$. Moreover, the value k^* is bounded by $k^* = O(\log w_{\text{best}})$ because the algorithm considered $(k^*, 1)$ and we know that $p_1^{-k^*} \leq \text{cost}(k^*, 1) \leq w_{\text{best}}$. Finally, we bound ℓ_k by $\text{cost}(k, \ell_k) \leq w_{\text{best}}$, so $\log \ell_k = O(\log w_{\text{best}})$. This allows us to bound (5.9) by $O(w_{\text{best}} \log^2 w_{\text{best}})$. Having these bounds on k^* and ℓ_k, we can now see that the loop is iterated $O(w_{\text{best}} \log w_{\text{best}})$ times and each priority queue operation takes time $O(\log \log w_{\text{best}})$, because there are at most k^* elements managed at the same time. So, all priority queue operations take time $O(w_{\text{best}} \log w_{\text{best}} \log \log w_{\text{best}})$ and are dominated by the work load computations.

Finally, a calculation analogous to (5.3) shows that $w_{\text{best}} = O(W_{\text{multi}} \log W_{\text{multi}})$ which proves the theorem. □

5.2 Summary of Results

We stress that the proposed algorithms work at least as well as standard LSH for each data set and query, given the space restrictions w.r.t. the number of repetitions provided by the user. In particular, these quantities are always at most as large as the expected running times stated in Theorem 4.1 and Theorem 6.1 (to be presented in the next section), given the number of repetitions is as large as stated in these theorems. This comes at the cost of making slightly more repetitions per level. Specifically, $O(\log \log W_{\text{single}}) = O(\log \log n)$ more repetitions are needed in Theorem 5.1 and $O(\log^3 W_{\text{multi}}) = O(\log^3 n)$ more repetitions are needed in Theorem 5.2.

Most importantly, Theorem 5.1 and Theorem 5.2 show that we are never more than log factors away from the ideal query time of a tuned LSH data structure across all possible parameters given the space constraints of the data structure. These quantities are query specific parameters, so we cannot assume that offline tuning achieves these candidate set sizes for all query points simultaneously.

6 A Probing Sequence in Hamming Space

In this section we analyze bit sampling LSH in Hamming space [17, Section 3.2.1] using a novel, simple probing sequence. We consider the static setting as in Section 4.1, where the number of points to report and the expansion around the query is known. We then show the existence of a certain (optimal) level and probing length parameters, and prove that using those give a good expected running time. The adaptive query algorithm from Section 5 would find parameters at least as good as those, and thus has a running time at least as good as what we show here (asymptotically within logarithmic factors).

Our scheme uses hash functions $h_k : \{0,1\}^d \to \{0,1\}^k$ that sample k positions at random with repetition. For a fixed query point $q \in \{0,1\}^d$ and $k \geq 1$, the probing sequence $\sigma_{k,\ell}$ maps $h_k(q)$ to the ℓ-th closest point in $\{0,1\}^k$, where ties are broken arbitrarily. This sequence can be generated efficiently, see [22].

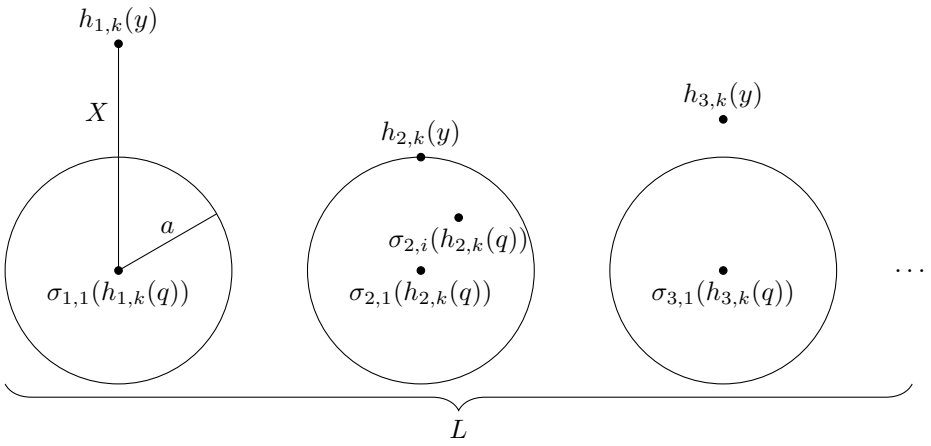

Figure 4: At each of the L repetitions we query the closest ℓ positions. Since the projected distance X to our target point y is distributed as $\mathrm{Bin}(k, \mathrm{dist}(q,y)/d)$, we find y with constant probability by setting $L = O(\Pr[X \le a]^{-1})$.

Fix a target close point $y \in \{0,1\}^d$ at distance r, let p be the probability that q and y collide, and let $p_{k,\ell}$ be the probability that y lands in the ℓ-th bucket that we check. Furthermore, let $V(a) = \sum_{i=0}^{a} \binom{k}{i}$ be the volume of the radius a Hamming ball. If $\sigma_{k,\ell} h_k(q)$ is at distance a to $h(q)$, we have a collision if q and y differ in exactly a out of the k coordinates chosen by h_k. Hence, $p_{k,\ell} = p^{k-a}(1-p)^a$ for the a satisfying $V(a-1) < \ell \le V(a)$. Thus, the sequence is reasonable. Figure 4 illustrates our approach.

The best approximations to sizes of hamming balls are based on the entropy function. Hence, for the purpose of stating the theorem, we introduce the following notation. For $\alpha \in [0,1]$ and $\beta \in [0,1]$, we let

$$\mathrm{H}(\alpha) = \alpha \log 1/\alpha + (1-\alpha)\log 1/(1-\alpha) \text{ and}$$
$$\mathrm{D}(\alpha \parallel \beta) = \alpha \log(\alpha/\beta) + (1-\alpha)\log((1-\alpha)/(1-\beta))$$

denote the binary entropy of α and the relative entropy between α and β, respectively. Moreover, let $\rho(r,c) = \frac{\log t}{\log n}\left(1 + \frac{\mathrm{D}(\alpha \parallel 1-p(r))}{\mathrm{H}(\alpha)}\right)$ where α is defined implicitly from $\frac{\log t}{\log n}\left(1 + \frac{\mathrm{D}(\alpha \parallel 1-p(cr))}{\mathrm{H}(\alpha)}\right) = 1$.

THEOREM 6.1. *Let $r > 0$ and $c \ge 1$. Let $S \subseteq \{0,1\}^d$ be a set of n points and let DS be the Multi-level LSH data structure obtained from preprocessing S with $L = \Omega(n^{\rho(r,c)})$. Given a query point q, let $t = N_r(q)+1$ and let c_q^* be the expansion around q in S. If $c_q^* \ge c$, there exists a query algorithm on DS to solve SRR with running time $O(n^{\rho(r,c_q^*)})$, otherwise the running time is $O(n^{\rho(r,c)} + N_{cr}(q))$.*

We do not know of a simple closed form for $\rho(r,c)$, but Figure 2 on Page 4 shows a numerical evaluation for comparison with the running time obtained for single-probing and other approaches.

The figure suggests that we always get better exponents than the single-probe approach, and that we get optimal query time for large t and asymptotically optimal for $t = n^{o(1)}$. Corollary 6.1 confirms this:

COROLLARY 6.1. *Let $\rho = (\log p_1)/(\log p_2) < 1/c$ be the usual exponent for bit sampling, then:*

If $t \ge n^{1-1/(p_2 \log p_1 + (1-p_2)\log 1/(1-p_1))}$, the expected query time is $O(n^{\rho(r,c)}) = O(n^\rho + t)$.

If $t = n^{o(1)}$, the expected query time is $O(n^{\rho(r,c)}) = n^\rho t^{O\left(1/(\frac{\log n}{\log t})\right)} = n^\rho t^{o(1)}$.

Note that this is the first algorithm to beat $n^\rho t^{\Omega(1)}$, even if only for certain ranges of t. The proof gives the exact asymptotic behavior.

Proof of Theorem 6.1. We will now show that the smaller number of repetitions needed by multi-probing leads to fewer collisions with the t very close points in hard instances of SRR. To see this, we bound the value of W_{multi} from (5.5) as follows:

$$\begin{aligned} W_{\text{multi}} &= \\ &\min_{k,\ell}\left[\frac{\ell + \sum_{x\in S, 1\le\ell} \Pr[\sigma_{k,\ell}(h_k(q)) = h_k(x)]}{\sum_{1\le\ell}\Pr[\sigma_{k,\ell}(h_k(q)) = h_k(y)]}\right] \\ &\le \min_{k,a}\left[\frac{V_k(a) + \sum_{x\in S}\Pr[\mathrm{dist}(h_k(q),h_k(x)) \le a]}{\Pr[\mathrm{dist}(h_k(q),h_k(y)) \le a]}\right] \\ &\le \min_{k,a}\left[\frac{V_k(a) + t + t'\Pr[X_1 \le a] + n\Pr[X_2 \le a]}{\Pr[X_1 \le a]}\right], \end{aligned}$$

where $X_1 \sim \mathrm{Bin}(k, 1-p_1)$ and $X_2 \sim \mathrm{Bin}(k, 1-p_2)$. The first inequality holds by restricting ℓ to only take values that are the volume of a k-dimensional

hamming ball; in the second inequality we upper bounded the collision probabilities for points in ranges $[0, r)$, $[r, cr)$ and $[cr, d]$.

The next step is to minimize this bound over the choice of k and a. We focus on $t' = O(t)$ and so we want

$$V_k(a) = t = n \Pr[X_2 \leq a]. \tag{6.10}$$

For simplicity we write $\alpha = a/k$ for the normalized radius. We use the following tight bound [27] on the tail of the binomial distribution, for $\alpha \in (0, 1/2)$:

$$\Pr[\mathrm{Bin}(k, p) \leq \alpha k] = \exp(-k\,\mathrm{D}(\alpha \parallel p))\Theta(1/\sqrt{k})$$
$$V_k(\alpha k) = \exp(kH(\alpha))\Theta(1/\sqrt{k}).$$

With those, our equation (6.10) can be written as

$$k\,\mathrm{H}(\alpha) = \log t = \log n - k\,\mathrm{D}(\alpha \parallel 1 - p_2) \text{ suggesting}$$
$$k = \frac{\log t}{\mathrm{H}(\alpha)} = \frac{\log n}{\mathrm{H}(\alpha) + \mathrm{D}(\alpha \parallel 1 - p_2)} \text{ and}$$
$$\frac{\log t}{\log n} = \frac{\mathrm{D}(\alpha \parallel 1 - p_2)}{\mathrm{H}(\alpha)} + 1.$$

We can then plug k into the bound on W_{multi}:

$$\begin{aligned} W_{\text{multi}} &\leq \frac{3t + t' \Pr[X_1 \leq a]}{\Pr[X_1 \leq a]} \\ &= \frac{t}{\exp(-k\,\mathrm{D}(\alpha \parallel 1 - p_1))\Theta(1/\sqrt{k})} + t' \\ &= O\left(n^{\frac{\log t}{\log n}\left(\frac{\mathrm{D}(\alpha \parallel 1-p_1)}{\mathrm{H}(\alpha)}+1\right)}\right) + t' \end{aligned} \tag{6.11}$$

which are exactly the values stated in Theorem 6.1.□

Proof sketch of Corollary 6.1. For the first statement observe that if α is as large as $1 - p_1$, then $\Pr[\mathrm{Bin}(k, 1 - p_1) \leq \alpha k]$ is constant. The second factor in the minimization has all terms being within a constant of t, and so the whole thing becomes $O(t)$. We can check that $\alpha \geq 1 - p_1$ happens exactly when $\frac{\log t}{\log n} \geq \frac{\mathrm{H}(1-p_2)}{\mathrm{H}(1-p_2)+\mathrm{D}(1-p_2 \parallel 1-p_1)}$. In this range $t \geq n^\rho$, so $O(t) = O(n^\rho + t)$.

For the second part of the corollary, we solve the equation implied by Theorem 6.1, asymptotically as $\tau = \frac{\log t}{\log n} \to 0$. Details can be found in Appendix D, but the idea is as follows: We first define $f_p(\alpha) = 1 + \frac{\mathrm{D}(\alpha \parallel p)}{\mathrm{H}(\alpha)}$, and show $f_{p_1}(\alpha) = (\rho + \psi\alpha/\log\frac{1}{p_2} + O(\alpha^2))f_{p_2}(\alpha)$ for ψ being the constant defined in Corollary 6.1. Using bootstrapping, we show the inversion $\alpha = f_{p_2}^{-1}(1/\tau) = \frac{\log 1/p_2}{\alpha \log 1/\alpha} + O(1/\log\frac{1}{\alpha})$. Plugging this into (6.11) proves the corollary. □

7 Conclusion

In this article we proposed two adaptive LSH-based algorithms for Spherical Range Reporting that are never worse than a static LSH data structure knowing optimal parameters for the query in advance, and much better on many input distributions where the output is large or the query is easy.

The main open problem remaining is to achieve target time $O(n^\rho + t)$ for all inputs and n and $t \leq n$. One approach might be a data-dependent data structure as described in [8]. In the light of our multi-probing results, this bound might even be obtained using data-independent methods as well. Here, it would be interesting to analyze other probing sequences. It would be interesting to see whether one can describe adaptive query algorithms that make use of the output-sensitive space/time-tradeoff data structures we described in Appendix E. Finally, it would be natural to extend our methods to give better LSH data structures for the approximate k-nearest neighbor problem.

Acknowledgements

The authors would like to thank the anonymous reviewers for their useful suggestions, which helped to improve the presentation of the paper. In addition they want to thank Ninh Pham for early discussions about adaptive LSH and output sensitivity. Finally they thank the entire Scalable Similarity Search group at ITU Copenhagen for reviews and interesting comments on earlier versions of this paper.

References

[1] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Contemporary Mathematics 223*, pages 1–56, 1999.

[2] Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 136–150, 2015.

[3] Alexandr Andoni. *Nearest Neighbor Search: the Old, the New, and the Impossible.* PhD thesis, MIT, 2009.

[4] Alexandr Andoni and Piotr Indyk. E2LSH, user manual. 2005.

[5] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006*, pages 459–468, 2006.

[6] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *Advances in Neural Information Processing Systems 28, NIPS 2015*, pages 1225–1233. Curran Associates, Inc., 2015.

[7] Alexandr Andoni, Thijs Laarhoven, Ilya Razen-

shteyn, and Erik Waingarten. Lower bounds on time-space trade-offs for approximate near neighbors. *arXiv:1605.02701*, 2016. Accepted for publication at SODA'17.

[8] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM Symposium on the Theory of Computing, STOC 2015*, pages 793–801. ACM, 2015.

[9] Sunil Arya, Guilherme D Da Fonseca, and David M Mount. A unified approach to approximate proximity searching. In *European Symposium on Algorithms, ESA 2010*, pages 374–385. Springer, 2010.

[10] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005*, pages 651–660. ACM, 2005.

[11] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, STOC 2002*, pages 380–388, 2002.

[12] Bernard Chazelle, Ding Liu, and Avner Magen. Approximate range searching in higher dimension. *Comput. Geom.*, 39(1):24–29, 2008.

[13] Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. *CoRR*, abs/1605.02687, 2016. Accepted for publication at SODA'17.

[14] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth annual Symposium on Computational Geometry, SOCG 2004*, pages 253–262. ACM, 2004.

[15] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling LSH for performance tuning. In *Proceedings of the 17th ACM conference on Information and Knowledge Management, CIKM 2008*, pages 669–678. ACM, 2008.

[16] Leonhard Euler. De progressionibus harmonicis observationes. *Commentarii academiae scientiarum Petropolitanae*, 7(1734-35):150–156, 1740.

[17] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

[18] Sariel Har-Peled and Sepideh Mahabadi. Proximity in the age of distraction: Robust approximate nearest neighbor search. *CoRR*, abs/1511.07357, 2015. Accepted for publication at SODA'17.

[19] Piotr Indyk. *High-dimensional computational geometry*. PhD thesis, Stanford University, 2000.

[20] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, STOC 1998*, pages 604–613, 1998.

[21] Michael Kapralov. Smooth tradeoffs between insert and query complexity in nearest neighbor search. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015*, pages 329–342, 2015.

[22] Donald E. Knuth. Combinatorial algorithms: Part 2, The Art of Computer Programming, vol. 4a, 2011.

[23] Thijs Laarhoven. Tradeoffs for nearest neighbors on the sphere. *arXiv preprint arXiv:1511.07527*, 2015.

[24] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search, VLDB 2007. pages 950–961. VLDB Endowment, 2007.

[25] Jirí Matousek. Geometric range searching. *ACM Comput. Surv.*, 26(4):421–461, 1994.

[26] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 1186–1195, 2006.

[27] Valentin Petrov. *Sums of independent random variables*, volume 82. Springer Science & Business Media, 2012.

[28] Malcolm Slaney, Yury Lifshits, and Junfeng He. Optimal parameters for locality-sensitive hashing. *Proceedings of the IEEE*, 100(9):2604–2623, 2012.

[29] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

A Trie-based Version of the Data Structure

In this section we discuss an alternative representation of our data structure. This is meant as a replacement for the Multi-level LSH data structure described in the main paper.It offers better space consumption while being slower to query.

As in the LSH forest data structure proposed by Bawa et al. [10], we do not store references to data points in hash tables. Instead we use a sorted array with a trie as a navigation structure on the array. The technical description follows.

First, choose $K \cdot L$ functions $g_{i,j}$ for $1 \leq i \leq L$ and $1 \leq k \leq K$ from $\mathcal{H}$ independently at random. For each $i \in \{1, \ldots, L\}$, we store a sorted array A_i with references to all data points in S ordered lexicographically by there bucket code over R^K. To navigate this array quickly, we build a trie over the bucket codes of all keys in S of depth at most K. Each vertex of the trie has two attributes `leftIndex` and `rightIndex`. If the path from the root of the trie to vertex v is labeled $L(v)$, then `leftIndex` and `rightIndex` point to the left-most and right-most elements in A_i whose bucket code starts with $L(v)$. We fix some more notation. For each point $q \in X$, we let $v_{i,k'}(q)$ be the vertex in trie $\mathcal{T}_i$ that is reached by searching for the bucket code of q on level at most k'. Furthermore, we let $\mathcal{T}_{i,k}(q)$ denote the set of keys that share the same length k prefix with q in trie $\mathcal{T}_i$. We can compute $|\mathcal{T}_{i,k}(q)|$ by subtracting $v_{i,k}(q)$.`leftIndex` from $v_{i,k}(q)$.`rightIndex` $+ 1$.

B Examples For Calculating W_{single} for Certain Input Distributions

In this section we discuss two examples to get a sense for quantity (5.2) defined on Page 7.

Example 1 (Random Points in Hamming Space) Fix a query point $q \in \{0,1\}^d$ and assume that our data set S consists of n uniform random points from $\{0,1\}^d$. Then the distance X from our query point is binomially distributed $\sim \text{Bin}(n, 1/2)$. If we choose the bitsampling hash function as in [20], $\sum_{x \in S} \Pr[h_k(q) = h_k(x)]$ is just

$$n\text{E}[(1 - X/d)^k] = n\text{E}[(X/d)^k]$$

by symmetry. This coresponds to finding the kth moment of a binomial random variable, which we can calculate by writing $X = d/2 + Z_d\sqrt{d/4}$ where Z_d has some distribution with $\text{E}(Z_d) = 0$ and where $Z_d \to Z$ converges to a standard normal. Then

$$\begin{aligned} n\text{E}[(X/d)^k] &= n2^{-k}\text{E}(1 + Z_d/\sqrt{d})^k \\ &= n2^{-k}(1 + k\text{E}(Z_d)/\sqrt{d} + O(k^2/d)) \\ &= n2^{-k}(1 + O(k^2/d)). \end{aligned}$$

For dimension $d = \Omega(\log n)^2$ our algorithm would find the ideal $k \approx \log_2 n$ to get $\sum_{x \in S} \Pr[h_k(q) = h_k(x)] = O(1)$ and $W = n^{\frac{\log 1/p_1}{\log 2}}$.

This work is of course exactly what we would expect for LSH with bitsampling and far points at distance $cr = d/2$. However normally the user would have had to specify this cr value, instead of the algorithm simply finding it for us.

Example 2 (Locally Growth-Restricted Data) Another interesting setting to consider is when the data is locally growth-restricted, as considered by Datar et al. [14, Appendix A]. This means that the number of points within distance r of q, for any $r > 0$, is at most r^c for some small constant c. In [14], the LSH framework is changed by providing the parameter k to the hash function. However, if we fix $r = k$, our algorithm will find a candidate set of size $W = O(\log n)$. So, our algorithm takes advantage of restricted growth and adapts automatically on such inputs.

Formally we can reuse the proof from [14], since they also inspect all colliding points. It is easy to see that theis integral $\int_1^{r/\sqrt{2}} e^{-Bc} c^b \, \text{d}c$ is still bounded by $2^{O(b)}$ when we start at $c = 0$ instead of $c = 1$, since the integrand is less than 1 in this interval.

C Lemma C.1

LEMMA C.1. *Let $x_1 \geq x_2 \geq \ldots$ be a non-increasing series of real numbers, and let $X_n = \sum_{k=1}^n x_k$ be the nth prefix sum. Then it holds:*

$$n/X_n \leq (n+1)/X_{n+1} \tag{C.1}$$

$$\sum_{k=1}^{n} 1/X_k = O(n \log n / X_n). \tag{C.2}$$

Proof. Since the values x_k are non-increasing, we have $X_n \geq nx_n \geq nx_{n+1}$ and so

$$(n+1)X_n \geq nX_n + nx_{n+1} = nX_{n+1}$$

which is what we want for (C.1). For the second inequality, we use (C.1) inductively, we get $a/X_a \leq b/X_b$ whenever $a \leq b$. Hence we can bound (C.2) term-wise as

$$\begin{aligned} \sum_{k=1}^{n} \frac{1}{X_k} &\overset{\text{(C.1)}}{\leq} \sum_{k=1}^{n} \frac{n}{kX_n} \\ &= \frac{n}{X_n} \sum_{k=1}^{n} \frac{1}{k} \\ &= \frac{n}{X_n} H_n \\ &= O(n \log n / X_n). \end{aligned}$$

Here $H_n = 1 + 1/2 + \cdots + 1/n = \log n + O(1)$ is the nth harmonic number with the assymptotics by Euler [16]. □

We may notice that the bound is tight for $x_1 = x_2 = \cdots = x_n$. Say $x_k = 1$ for all k, then $X_k = k$ and $\sum_{k=1}^{n} 1/X_k = H_n = \Omega(n \log n/X_n)$. It is however common for the x_is to be strictly decreasing. In such cases we get closer to the other extreme, in which the $\log n$ factor disappears as $\sum_{k=1}^{n} 1/X_k = n/X_n$, which is sharp when $x_1 = 1$ and $x_k = 0$ for all other $k \geq 2$.

D Proof of Corollary 6.1, second part

Recall that our algorithm runs in time $n^{\rho(r,c)}$ where $\rho(r, c) = \frac{\log t}{\log n} f(\alpha, p_1)$. Here $f(\alpha, p) = 1 + \frac{\mathrm{D}(\alpha \parallel 1-p)}{\mathrm{H}(\alpha)}$, and α is define implicitely from $\frac{\log t}{\log n} f(\alpha, p_2) = 1$.

When t is small compared to n, the multiprobing radius α can be made quite small as well. Working towards results for $t = n^{o(1)}$ we thus consider the regime $\alpha = o(1)$:

$$\begin{aligned} f(\alpha, p_1) &= 1 + \frac{\mathrm{D}(\alpha \parallel 1-p_1)}{\mathrm{H}(\alpha)} \\ &= \frac{\mathrm{H}(\alpha) + \mathrm{D}(\alpha \parallel 1-p_1)}{\mathrm{H}(\alpha) + \mathrm{D}(\alpha \parallel 1-p_2)} f(\alpha, p_2) \\ &= \frac{\log \frac{1}{p_1} + \alpha \log \frac{p_1}{1-p_1}}{\log \frac{1}{p_2} + \alpha \log \frac{p_2}{1-p_2}} f(\alpha, p_2) \\ &= \left(\rho + \frac{\psi}{\log 1/p_2}\alpha + O(\alpha^2)\right) f(\alpha, p_2), \end{aligned} \tag{D.3}$$

for small α and constants

$$\rho = \frac{\log 1/p_1}{\log 1/p_2}$$
$$\psi = \frac{\log \frac{p_1}{1-p_1} \log \frac{1}{p_2} - \log \frac{1}{p_1} \log \frac{p_2}{1-p_2}}{\log 1/p_2} \leq \log \frac{1}{1-p_1}$$

depending on p_1 and p_2. This already shows that we get running time $n^{\frac{\log t}{\log n}\rho f(\alpha,p_2)(1+O(\alpha))} = n^{\rho(1+O(\alpha))}$, which is optimal up to lower order terms, if indeed $\alpha = o(1)$. We thus direct our attention to how fast α goes to 0 as a function of t and n. For that we first expand the following asymptotics:

$$\begin{aligned} \mathrm{H}(\alpha) + \mathrm{D}(\alpha \parallel 1-p) &= \alpha \log \tfrac{1}{1-p} + (1-\alpha) \log \tfrac{1}{p} \\ &= \log \tfrac{1}{p} + O(\alpha) \end{aligned}$$

$$\begin{aligned} \mathrm{H}(\alpha) &= \alpha \log \tfrac{1}{\alpha} + (1-\alpha) \log \tfrac{1}{1-\alpha} \\ &= \alpha \log \tfrac{1}{\alpha} + (1-\alpha)(\alpha - O(\alpha^2)) \\ &= \alpha(\log \tfrac{1}{\alpha} + 1) + O(\alpha^2) \end{aligned}$$

$$\begin{aligned} f(\alpha, p) &= \frac{\mathrm{H}(\alpha) + \mathrm{D}(\alpha \parallel 1-p)}{\mathrm{H}(\alpha)} \\ &= \frac{\log \frac{1}{p} + O(\alpha)}{\alpha(\log \frac{1}{\alpha} + 1) + O(\alpha^2)} \\ &= \frac{\log \frac{1}{p} + O(\alpha)}{\alpha(\log \frac{1}{\alpha} + 1)} \end{aligned} \tag{D.4}$$

We would like to solve (D.4) for α, and plug that into (D.3). To this end, we let $y = f(\alpha, p)/\log \frac{1}{p}$ and note the assymptotic bound $1/y^2 < \alpha < 1/y$. That gives us $\alpha = O(1/y)$ and $\log 1/\alpha = O(\log y)$, and we can use these estimates to "bootstrap" an inversion:

$$\begin{aligned} \alpha &= \frac{1 + O(\alpha)}{y(\log \frac{1}{\alpha} + 1)} \\ &= \frac{1 + O\left(\frac{1+O(\alpha)}{y(\log \frac{1}{\alpha}+1)}\right)}{y\left(\log \frac{1}{\frac{1+O(\alpha)}{y(\log \frac{1}{\alpha}+1)}} + 1\right)} \\ &= \frac{1 + O\left(\frac{1}{y \log y}\right)}{y\left(\log\left[y(\log \frac{1}{\alpha} + 1)\right] + \log\left[\frac{1}{1+O(1/y)}\right] + 1\right)} \\ &= \frac{1 + O\left(\frac{1}{y \log y}\right)}{y \log y + O(y \log \log y)} \\ &= \frac{1 + o(1)}{y \log y} \end{aligned} \tag{D.5}$$

Plugging the result back into (D.3) we finally get:

$$\begin{aligned} \log n^{\rho(r,c)} &= (\log t) f(\alpha, p_1) \\ &= \log t \left(\rho + \frac{\psi}{\log 1/p_2}\alpha + O(\alpha^2)\right) f(\alpha, p_2) \\ &= \log t \left(\rho + \frac{\psi}{\log 1/p_2}\frac{1+o(1)}{y \log y}\right) f(\alpha, p_2) \\ &= \log t \left(\rho f(\alpha, p_2) + \frac{\psi(1+o(1))}{\log f(\alpha, p_2)}\right) \\ &= \rho \log n + \psi \frac{1+o(1)}{\log \frac{\log n}{\log t}} \log t \\ &= \rho \log n + o(\log t), \end{aligned}$$

as $\frac{\log n}{\log t}$ goes to ∞, i.e. when $t = n^{o(1)}$.

We note again that this improves upon all other known methods, which get $\log \mathrm{E}(W_k) = \rho \log n + \Omega(\log t)$ for the same range of parameters.

E A Different Approach to Solving SRR

We reconsider the approach to solve SRR presented in Indyk's Ph.D. thesis [19, Page 12] under the name

"enumerative PLEB". While his method does not yield good running times directly, it is possible to combine a number of very recent results, to get running times similar to the ones achieved by our methods. We give a short overview of this approach next. As in Section 4.1, we assume that the number of points t to report is known. At the end of this section we describe a counting argument that is also contained in Indyk's Ph.D. thesis [19] that allows to solve the c-approximate spherical range counting problem in an output-sensitive way.

Indyk describes a black-box reduction to solve SRR using a standard dynamic data structure for the (c, r)-near neighbor problem. It works by repeatedly querying an (c, r)-near point data structure (time $O(n^{\rho_q})$) and then deleting the point found (time $O(n^{\rho_u})$), where ρ_q and ρ_u are the query- and update-parameters. (For a standard LSH approach, we have $\rho_q = \rho_u$.) This is done until the data structure no longer reports any points within distance r. Due to the guarantees of an (c, r)-near neighbor data structure, in the worst case the algorithm recovers all points within distance cr, giving a total running time of $t'(n^{\rho_q} + n^{\rho_u})$, where t' is the number of points within distance cr which might yield a running time of $\Omega(n^{1+\rho})$ as noticed in Section 4.

Of course, we can never guarantee sublinear query time when t' is large, but we can use a space/time-tradeoff-aware to improve the factor of t, when the number of returned points is large.

We will assume the (c, r)-near neighbor data structure used in the reduction is based on LSH. In [7], Andoni et al. describe a general data structure comprising loosely "all hashing-based frameworks we are aware of":

Definition 4. (List-of-points data structure)

- *Fix sets $A_i \subseteq \mathcal{R}^d$, for $i = 1 \dots m$; with each possible query point $q \in \mathcal{R}^d$, we associate a set of indices $I(q) \subseteq [m]$ such that $i \in I(q) \Leftrightarrow q \in A_i$;*
- *For a given dataset S, the data structure maintains m lists of points $L_1, L_2, \dots, L_m$, where $L_i = S \cap A_i$.*

Having such a data structure, we perform queries as follows: For a query point q, we scan through each list L_i for $i \in I(q)$ and check whether there exists some $p \in L_i$ with $\|p - q\| \leq cr$. If it exists, return p.

Data structures on this form naturally allow insertions of new points, and we notice that if "Lists" are replaced by "Sets" we can also efficiently perform updates.

To solve spherical range reporting, we propose the following query algorithm for a point q:

1. For each $i \in I(q)$ look at every point x in L_i.
2. If $\|x - q\| \leq r$, remove the point from all lists, L_j, where it is present.

This approach allows for a very natural space/time-tradeoff. Assuming that querying the data structure takes expected time $O(n^{\rho_q})$ and updates take expected time $O(n^{\rho_u})$, the expected running time of the query is $O(n^{\rho_q} + tn^{\rho_u})$. This asymmetry can be exploited with a time/space tradeoff. In very recent papers [23, 13, 7] it was shown how to obtain such tradeoffs in Euclidean space for approximation factor $c \geq 1$, for any pair (ρ_q, ρ_u) that satisifies

$$c^2\sqrt{\rho_q} + (c^2 - 1)\sqrt{\rho_u} = \sqrt{2c^2 - 1}.$$

To minimize running time, we may take exponents balancing $T = n^{\rho_q} = tn^{\rho_u}$ and obtain

$$\begin{aligned}\frac{\log T}{\log n} &= \frac{1}{2c^2-1} + \frac{c^2-1}{2c^2-1}\tau \\ &\quad + \frac{c^2\left(c^2-1\right)}{2c^2-1}\left(2 - \tau - 2\sqrt{1-\tau}\right) \\ &\overset{(*)}{\leq} \rho + (1 - c^4\rho^2)\tau,\end{aligned}$$

where $\tau = \frac{\log t}{\log n}$ and $\rho = 1/(2c^2 - 1)$. Here (*) holds for $t \leq \frac{2c^2-1}{c^4}$, and $T = O(t)$ otherwise. Note that this approach requires knowledge of t. A visualization of the running time guarantees of this approach is shown in Figure 5. Note that it requires knowledge of t and does not adapt to the expansion around the query point. It would be interesting to see whether our adaptive methods could be used to obtain a variant that is query-sensitive. Next, we discuss an algorithm for the spherical range counting problem that can be used to obtain an approximation of the value t sufficient for building the data structure presented here.

E.1 Solving c-approximate Spherical Range Counting

In [19, Chapter 3.6], Indyk shows that by performing $O((\log n)^2/\alpha^3)$ queries to independently built (c, r)-near neighbor data structures, there is an algorithm that returns for a query q a number C such that $(1 - \alpha)N_r(q) \leq C \leq (1 + \alpha)N_{cr}(q)$ with constant probability. The running time of the black-box reduction is $O(n^{\rho}(\log n)^2/\alpha^3)$. We show in this section that we can solve the problem in time $O((n/t)^{\rho} \log n/\alpha^3)$.

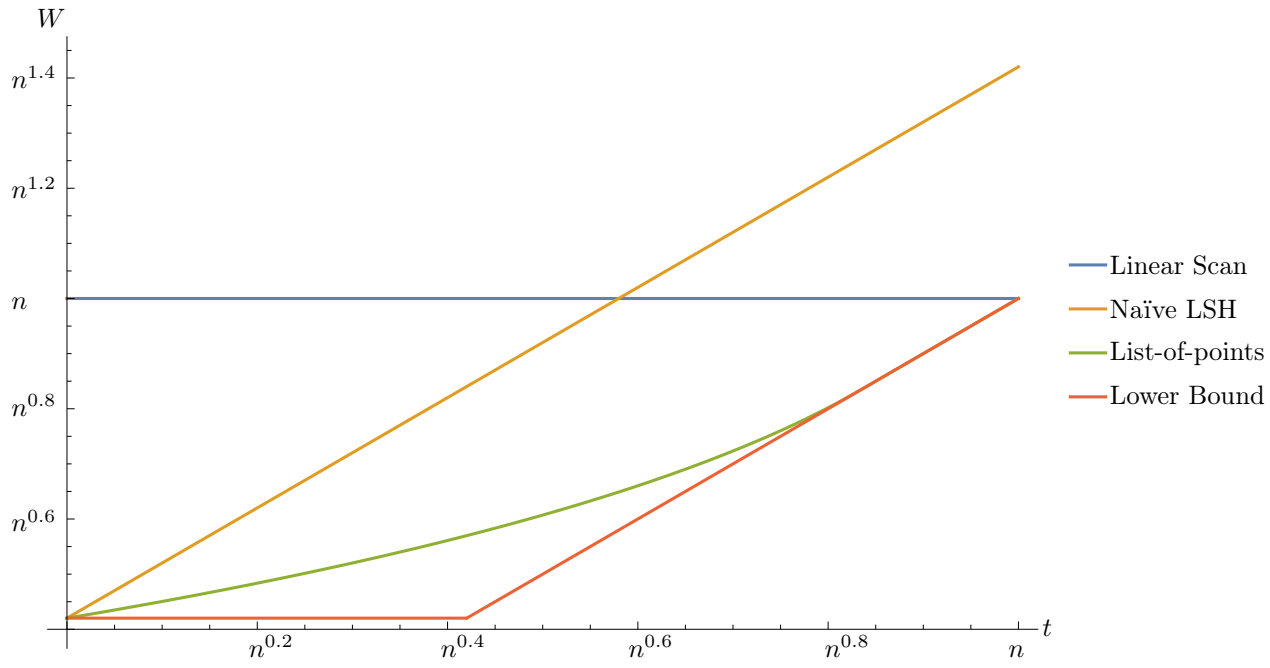

Figure 5: Visualization of the running time guarantees of the space/time-tradeoff list-of-points data structure for $c = 1.3$ in Euclidean space. The x-axis shows the value of t compared to n, the y-axis shows the expected work W. For comparison, we plotted the lower bound of $O(n^\rho + t)$, the running time $O(tn^\rho)$ of the naïve LSH approach, and the running time $O(n)$ of a linear scan.

At the heart of the algorithm of [19] is a subroutine that has the following output behavior for fixed C:

1. If $N_{cr}(q) \leq C(1-\alpha)$, it will answer SMALLER
2. If $N_r(q) \geq C$, it will answer GREATER

The subroutine uses $O(\log n/\alpha^2)$ queries of independently build (c, r)-near neighbor data structures, each built by sampling n/C points from the data set.

We can use the above subroutine to solve the spherical range counting problem in time $O((n/t)^\rho \log n/\alpha^3)$ time as follows. Half the size of α, and perform a geometrical search for the values $t = n, (1-\alpha)n, (1-\alpha)^2 n, \ldots$. Assuming that a query on a data structure that contains n points takes expected time $O(n^\rho)$ and stopping as soon as the algorithm answers "Greater" for the first time, we obtain a running time (without considering the $O(\log n/\alpha^2)$ repetitions for each t value) of

$$\begin{aligned}
&\left(\frac{n}{n}\right)^\rho + \left(\frac{n}{n(1-\alpha)}\right)^\rho + \left(\frac{n}{n(1-\alpha)^2}\right)^\rho + \cdots + \left(\frac{n}{t}\right)^\rho \\
&\leq \left(\frac{n}{t}\right)^\rho + \left(\frac{n(1-\alpha)}{t}\right)^\rho + \left(\frac{n(1-\alpha)^2}{t}\right)^\rho + \cdots \\
&= \left(\frac{n}{t}\right)^\rho \frac{1}{1-(1-\alpha)^\rho} \\
&\leq \left(\frac{n}{t}\right)^\rho \frac{1}{\alpha\rho} \qquad \text{as } (1-\alpha)^\rho \leq 1-\alpha\rho \text{ for } 0 \leq \rho \leq 1,
\end{aligned}$$

which results in a total running time of $O((n/t)^\rho \log n/\alpha^3)$.