Randomized Algorithms for Comparison-Based Search: Supplementary Material

Dominique Tschopp
AWK Group
Bern, Switzerland
dominique.tschopp@gmail.com

Suhas Diggavi
University of California Los Angeles (UCLA)
Los Angeles, CA 90095
suhasdiggavi@ucla.edu

Payam Delgosha
Sharif University of Technology
Tehran, Iran
pdelgosha@ee.sharif.ir

Soheil Mohajer
Princeton University
Princeton, NJ 08544
smohajer@princeton.edu

Abstract

The problem is finding the nearest neighbor (or one of the R-nearest neighbors) for a query object q in a database of n objects, when only similarity comparison questions can be asked is studied in a paper titled "Randomized Algorithms for Comparison-Based Search". Some of the proofs and technical details are skipped in the main paper, due to the page limit. In this appendix, we present all the missing proofs and complementary material.

1 Disorder Constant of the Star Database

Proof of Lemma 1. Take three nodes $x, y, z \in \mathcal{T} \cup \{q\}$ where $\mathcal{T}$ denotes the set of nodes in star shaped graph. We have to show that all four inequalities in Definition 2 hold with $D = \Theta(\alpha)$. However, since all the proofs are similar, we only discuss the first inequality in detail. Our goal is to show that

$$r_x(y) \leq D(r_z(x) + r_z(y)) \tag{1}$$

Let $d = d(x, y)$ denote the distance between x and y in the graph. We first define the distance-ball around point x as $\mathcal{B}_u(r) = \{i \in S | d(u, i) \leq r\}$.

We consider two distinct cases: $d \leq 1$ or $d > 1$. If $d \leq 1$, it means that x and y are in the same SN. In this case we have $r_x(y) \leq \alpha$ because the distance between two nodes in a SN is smaller than the distance between nodes in different SNs. Since $r_z(x) + r_z(y) \geq 1$, simply by substituting $D = \alpha$, the inequality (1) holds.

In the second case where $d > 1$, x and y are in different SNs. We claim that $|\mathcal{B}_x(d)| \leq 4\alpha^2 d$. First focus of the objects whose distance from x over the star graph does not exceed d. Even if x is in the center of the graph, there are at most $d\alpha$ SNs within distance d of x each has α nodes. Furthermore, there could be more objects that could be in the distance-ball $\mathcal{B}_x(d)$ around x due to the additional query point q. If the query point were within distance d of x, it could bring in at most $2d\alpha^2 + \alpha^2 \leq 3d\alpha^2$ additional objects into the distance-ball around x. Indeed, through the query point, a point x could be connected to at most $2d\alpha + \alpha$ objects on each branch (*i.e.*, to d SNs on each side of the SN containing the DN through which x is connected, each with α objects inside, and α objects inside the SN). Therefore $|\mathcal{B}_x(d)| \leq 4\alpha^2 d$.

On the other hand, we claim that $|\mathcal{B}_z(d/4)| \geq d\alpha/4$. Note that $d/4 \leq n/\alpha^2$, because the distance between any two nodes is upper bounded by $4n/\alpha^2$ (this maximum is obtained when two nodes are

at the end of two different branches). Since there are n/α^2 SNs in a branch, each having α nodes, there are at least $d\alpha/4$ nodes within distance less than $d/4$ from z, which shows our claim.

Now, we have,

$$\begin{aligned} r_z(x) + r_z(y) &\geq |\mathcal{B}_z(d(x,z)/2)| + |\mathcal{B}_z(d(y,z)/2)| \\ &\geq |\mathcal{B}_z(d(x,z)/2)| + |\mathcal{B}_z([d(x,z) - d(y,z)]/2)| \\ &\geq |\mathcal{B}_z(\max\{d(x,z)/2, [d(x,z) - d(y,z)]/2\})| \\ &\geq |\mathcal{B}_z(d(x,z)/4)| \geq d\alpha/4 \end{aligned} \tag{2}$$

where we used the fact that $|\mathcal{B}_x(d)|$ is an increasing function w.r.t. d in inequalities above.

Combining (2) with the fact that

$$r_x(y) \leq \mathcal{B}_x(d) \leq 4\alpha^2 d = 16\alpha.\frac{d\alpha}{4} \tag{3}$$

proves our claim in (1) by setting $D = 16\alpha$.

□

2 Lower Bounds for NNS: Details and Proofs

We need a few definitions and tools which will be used in the proof of the theorem. First, we fix an arbitrary *successful* algorithm $\mathcal{A}$, which can find the nearest neighbor for *any* possible query point in the star database. We define the set of questions that algorithm $\mathcal{A}$ asks in response to an input $q \in \mathcal{M}$.

Definition A.1. *In response to an input $q \in \mathcal{M}$, algorithm $\mathcal{A}$ asks a set of questions $\mathcal{Q} = \{\omega_1, \omega_2, \ldots, \omega_m\}$ in which ω_i is a triple (a_i, b_i, ans_i) in which a_i and b_i denote sides of question (we ask $\mathcal{O}(q, a_i, b_i)$), and ans_i is a binary number ("0" for a_i and "1" for b_i) denoting the answer to the question.*

In fact since the answer to a question is crucial in an algorithm. In the following, when we talk about a question we mean both the *question* and its *answer*. Note that since we focus on the deterministic algorithms, questions themselves do not reveal information, more precisely, having the answers up to any step of the algorithm, we can retrieve the next questions. This is true since the ith question only depends on the answers to questions 1 to $i-1$. So we can assign a binary sequence to every query node which is the sequence of answers received from the oracle and contains all the information we need about the attitude of a deterministic algorithm towards that query.

Definition A.2. *For a query $q \in \mathcal{M}$, define Π_q as the binary sequence*[1] $(\pi_1^q, \pi_2^q, \ldots, \pi_{m(q)}^q)$ *which contains the answers of questions algorithm $\mathcal{A}$ asks from the query with q as its input. Note that $m(q)$ is the number of questions algorithm asks from the query to find NN of q.*

Definition A.3. *For a set of questions $\mathcal{Q} = \{(a_i, b_i, ans_i) : i = 1, \ldots, m\}$, we say $\mathcal{Q}$ implies q and denote it by $q \models \mathcal{Q}$, if $\forall 1 \leq i \leq m$, we have $\mathcal{O}(q, a, b) = ans_i$. This basically means that q is consistent with all the questions and answers revealed in set $\mathcal{Q}$.*

Assume $\mathcal{Q}_q$, the set of questions asked by an algorithm $\mathcal{A}$ for an input q, is the same as the set of questions asked when q' is fed to the algorithm as input, *i.e.*, $q' \models \mathcal{Q}_q$. Since all the information of questions is available in the binary sequence of answers, we have $\Pi_q = \Pi_{q'}$. In fact we can define an equivalence relation based on this concept.

Definition A.4 (alike queries)**.** *Two query points q_1 and q_2 are called* alike *and denoted by $q_1 \sim q_2$ if $\Pi_{q_1} = \Pi_{q_2}$.*

In fact our algorithm behaves similarly in response to two alike queries as input, and of course returns the same output for both. Therefore, two alike queries should have the same NN although they may differ in issues which are not important for our algorithm: DNs in other branches and their orders.

The following lemma plays a key role in the proof of Theorem 1.

[1] Note that although all the following notions depend on both the query and algorithm, we can skip their dependency on $\mathcal{A}$ since we have already fixed the algorithm.

Lemma A.1 (Existence of a question about a DN)**.** *Let q be the input query to the algorithm, and assume $\delta_q(i)$ is a DN of q in branch ϕ_i which lies in supernode S. Denote by x th weight of the query edge connecting q to $\delta_q(i)$. If algorithm $\mathcal{A}$ does not ask any question about $\delta_q(i)$ (*i.e.*, $\omega = (a, b, ans)$ with either $a = \delta_q(i)$ or $b = \delta_q(i)$), we can obtain another query q^* with $q \sim q^*$, by changing x to any valid weight x of the form $1 + O(\epsilon)$.*

Proof of Lemma A.1. To prove this lemma, we will choose a proper value for x^*, the weight of the query edge connects q^* to d, and show show that changing x to x^* does not change the answer to any of the questions $\mathcal{A}$ asks when q. This implies $q \sim q^*$. First we present three observations.

(O1) For any node $u \in \mathcal{T}$ on branch ϕ_i, there are α paths connecting it to the query point, each through one branch, and one DN. Since the weight of edges emitting from the center have weight $n/(\alpha^2)$, the shortest path from u to q would be through the DN on the same branch, ϕ_i. Therefore the distance between any node and the query point is the sum of weights of the path connecting it to the query through the same branch.

(O2) If u and v are in the same branch, say ϕ_i, the relative distance $d(q, u) - d(q, v)$ is independent of the weight of the edge connecting q to $\delta_q(i)$, the DN of q on ϕ_i. Hence the answer of the question $\mathcal{O}(q, u, v)$ is independent of this weight.

(O3) For any node u on branch ϕ_i, denote by l_u is the distance between the SN containing u and the DS on ϕ_i (the SN contains $\delta_q(i)$). For two objects u and v on the same branch, $l_v < l_u$ implies $d(q, v) < d(q, u)$. This is due to the fact that the all the object edges in ϕ_i have the same weight. Hence the distance between SNs plays the *first order* role in comparison, and then the object edges and after that query edges determine the answer to the comparison questions.

We want to show that by changing the value of x, the answer of every question in $\mathcal{Q}$ remains the same. Take a question $\omega = (u, v, \text{ans}_{u,v})$ in $\mathcal{Q}$. There are three possibilities:

1. u and v are on the same branch ϕ_j: as a result of **(O2)** above, the answer to ω remains the same, no matter $j = i$ or $j \neq i$.
2. u and v are on two different branches ϕ_j and ϕ_k but $i \notin \{j, k\}$: as a result of **(O1)**, the distances $d(q, u)$ and $d(q, v)$ will remain the same after changing x. So the answer to question ω will not change.
3. u is on ϕ_i and v is on ϕ_j with $i \neq j$: This is the case we discuss the next[2] (See Fig. A.1).

Denote the SN contains v be S_v and the DS on branch ϕ_j by DS_{ϕ_j}. Note that the DN on branch ϕ_i is $\delta_q(i) \in \text{DS}_{\phi_j}$ which is connected to q with weight x. The DN on branch ϕ_j is in DS_{ϕ_j}, say node $\delta_q(j)$ which is connected to q with weight y. If $S_v = \text{DS}_{\phi_j}$, according to **(O3)**, v is closer to query than u regardless of the values of x and y, which means that by changing x, $\text{ans}_{u,v}$ will not change. On the other hand, if the distance between S_v and DS_{ϕ_j} (denoted by l_v) is different from l_u, then again according to **(O3)** $ans_{u,v}$ will not change. Therefore we analyze the case where $l_u = l_v = l$ (see Figure A.1). We define m_1, m_2, m_3 and m_4 to be the weights of edges connecting u to S_u, $\delta_q(i)$ to DS_{ϕ_i}, v to S_v and $\delta_q(j)$ to DS_{ϕ_j}, respectively.

We have to analyze the following two cases, whether $S_u \neq \text{DS}_{\phi_i}$ or $S_u = \text{DS}_{\phi_i}$, separately. Recall that $m_1 = m_2 = i/(4\alpha)$ and $m_3 = m_4 = j/(4\alpha)$.

Case I: $S_u \neq \text{DS}_{\phi_i}$ implies $l \neq 0$, in this case we have,

$$\begin{aligned} d(q, u) &= x + l + \frac{2i}{4\alpha} \\ d(q, v) &= y + l + \frac{2j}{4\alpha} \end{aligned} \tag{4}$$

[2]Clearly, a similar argument holds for the case that u is on ϕ_j and v is on ϕ_i.

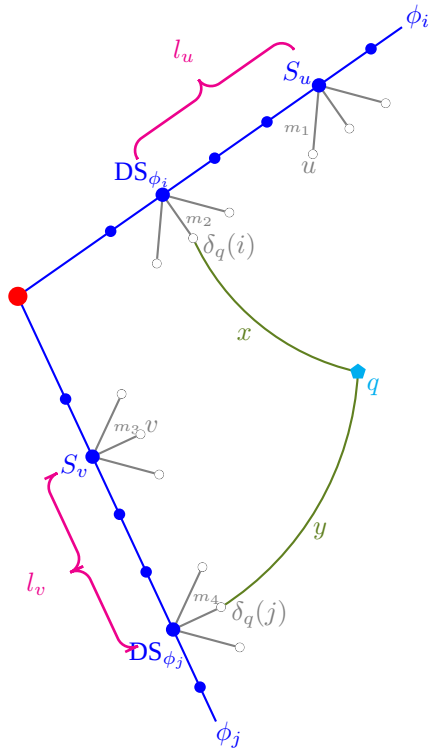

Figure A.1: comparing $d(q, u)$ and $d(q, v)$

and hence

$$|d(q,u) - d(q,v)| \geq \frac{|i-j|}{4\alpha} - |x-y| = \frac{|i-j|}{4\alpha} - O(\epsilon) \tag{5}$$

since $\epsilon \ll 1/(4\alpha)$, by changing x^* so that it remains of the form $1 + O(\epsilon)$, $|x-y|$ would be of $O(\epsilon)$ and the sign of the relative distance will not change which means that the answer will not change.

Case II: $S_u \neq \mathrm{DS}_{\phi_i}$ implies $l = 0$. Note that we assume there is no question about $\delta_q(i)$ but there is a question comparing u and v, therefore $u \neq \delta_q(i)$. If v is the DN in branch ϕ_j, then v is a Direct Node but u is not, therefore v is closer to the query independent of the value of x. Otherwise we have,

$$\begin{aligned} d(q,u) &= x + \frac{2i}{4\alpha} \\ d(q,v) &= y + \frac{2j}{4\alpha} \end{aligned} \tag{6}$$

and hence

$$|d(q,u) - d(q,v)| \geq \frac{|i-j|}{4\alpha} + |x-y| = \frac{|i-j|}{4\alpha} + O(\epsilon). \tag{7}$$

Again, since $\epsilon \ll 1/(4\alpha)$, the answer will not change by changing x.

□

Using this lemma, we have the following corollary.

Corollary A.1 (Existence of a question about a DN)**.** *Let $\mathcal{A}$ be a an algorithm which can find the NNS for a query point q. $\mathcal{A}$ should ask at least one question involving all $\delta_q(i)$ for all $i = 1, \ldots, \alpha$.*

Proof of Corollary A.1. We prove this corollary by contradiction using Lemma A.1. Assume there is no question involving $\delta_q(i)$ for some i, which is connected to q via a query edge of weight x. We can change the value of x and obtain a new query q^* in the following way: if $\delta_q(i)$ is the NN, then increase x to obtain q^* and if it is not, decrease x so that $\delta_q(i)$ becomes the NN of q^*. In both cases, the set of questions and answers of q^* and q are the same but they have different NNs. Therefore since the algorithm is not able to distinguish q and q^*, it will gives a wrong output for at least of them.

Note that since we want q^* to be a valid query, changing x should be in a way that it still remains of the form $1 + O(\epsilon)$. In this case $|x - y| = O(\epsilon)$ and since $\epsilon \ll 1/(4\alpha)$, the sign of the relative distance in two cases and the answer to the question will not change. □

Now, we first show the logarithmic lower bound on the number of questions has to be asked by $\mathcal{A}$ on average.

2.1 Proof of the Logarithmic Bound for NNS Problem

Definition A.5 (co-branch queries)**.** *For a branch $\phi_i, 1 \leq i \leq \alpha$ define the relation $\overset{i}{\equiv}$ in the following way: for two queries $q_1, q_2 \in \mathcal{K}$ we say that $q_1 \overset{i}{\equiv} q_2$ if and only if they have the same DN in branches other than ϕ_i and the ordering of their DNs are the same,* i.e., *(recall Definition 3 in the main paper) $\delta_{q_1}(j) = \delta_{q_2}(j), \forall j \neq i$ and $\Psi_{q_1} = \Psi_{q_2}$.*

It could be easily seen that this is an equivalence relation and divided queries into equivalence classes, each class having n/α queries. We call the class of object which satisfy $\overset{i}{\equiv}$ the *i-co-branch class*.

The i-co-branch queries are exactly the same unless their DNs in branch ϕ_i, which are different. Let $C = \{q_1, q_2, \dots q_{n/\alpha}\}$ be an i-co-branch class and $\delta_{q_j}(i)$ be the DN of q_j in branch ϕ_i. As a result of Lemma A.1, $\mathcal{A}$ should ask at least one question involving $\delta_{q_j}(i)$ when q_j is the input of the algorithm.

Definition A.6. *Define $\mathcal{N}_q^i$ be the number of questions involve at least one node on the branch ϕ_i asked by the $\mathcal{A}$ from the oracle, when q is its input.*

We claim that $\mathcal{A}$ should ask about $\log(n/\alpha)$ nodes in branch ϕ_i on average for i-co-branch queries. More formally, we state and prove the following lemma. This lemma basically states that for co-branch queries the algorithm has to asks at least $\log(n)$ questions per branch on average.

Lemma A.2 (i-co-branch queries on average)**.** *Assume $C = \{q_1, q_2, \dots, q_{n/\alpha}\}$ is an i-co-branch class. We have,*

$$\sum_{k=1}^{n/\alpha} \mathcal{N}_{q_k}^i \geq \frac{1}{2}\frac{n}{\alpha}\log\left(\frac{n}{\alpha}\right) \tag{8}$$

Proof of Lemma A.2. Here we want to build a tree that shows the respond of our algorithm to these co-branch queries. Each node of the tree is a subset of the queries in the class of our interest. At first we put all n/α queries in the top node of the tree. Each node of the tree represents a subset of i-co-branch queries. In each level our algorithm asks a question that according to its answer queries are divided into some disjoint subsets based on the revealed answer. This tree is constructed only when a question involving a node on ϕ_i is asked. Take the first time that there is a question involving a node on ϕ_i. Before this moment, all question are about nodes outside ϕ_i and because co-branch queries are the same outside ϕ_i, the answer to these questions are the same for all i-co-branch queries. Since our algorithm is deterministic, the first question about a node in ϕ_i is the same for all i-co-branch queries. There are two possibilities at each step of the construction of the tree:

1. Assume the question is about two nodes p_s and p_t that both are in ϕ_i. In this situation throw away queries q_s and q_t. The remaining queries are at most divided into two subsets C_1 and C_2 based on the answer of the question.

2. Let this question be about a node p_s on branch ϕ_i and a node outside branch ϕ_i. Throw away query q_s. The remaining queries again are divided into at most two subsets C_1 and C_2 according to the answer of the question.

Note that so far there has not been any question from the DN of queries in C_1 and C_2. This process will be continued recursively. We divide nodes staying on the tree into two parts: *green* nodes which represent the set of queries that so far no question has been asked about their DNs; and *red* nodes which are those we throw away at each step. For instance the top node of the tree is a green node and the one or two node representing queries we throw away at the first level discussed above are red.

Take a green node G at some level in the graph. It represents a set of queries[3] $C_G = \{q_1, q_2, \ldots, q_k\}$ and their DNs $N_G = \{\delta_{q_1}(i), \delta_{q_2}(i), \ldots, \delta_{q_k}(i)\}$, which are not distinguishable for the algorithm based on the answers received so far. Now look at the questions after this point $\mathcal{A}$ asks for them as input. For questions with both sides outside ϕ_i, the answer would be the same for queries in C_G, and so these questions do not divide this set. Therefore up to the point that there is a question with at least one side in ϕ_i, they get the same answer.

Taking the first question that at least one side is in ϕ_i, exactly like the root node, there are two possible cases which will result in children of G in the graph: at most two green nodes C_G^1 and C_G^2 in the first case and four (or three) nodes where at most two of them are green (C_G^1 and C_G^2) and two of them are red nodes which represent the two queries thrown away at this level.

Since green nodes always have children, leafs of our trees are red nodes which represent queries. Since the set of queries for children of a node is disjoint, leafs of the tree are exactly n/α i-co-branch queries. Note that each query has at least as its depth in the tree as questions with at least one side in branch ϕ_i. Since in each node, tree is divided into at most four sub trees, we can represent every leaf of the graph such as q_k with a quaternary[4] sequence of length l_k, or equivalently, a binary sequence of length $2l_k$ for q_k. All such sequences are different for non-identical queries. Moreover, we have $\mathcal{N}_{q_k}^i \geq l_k$. This is exactly a quaternary Huffman tree, in which their average is grater than the entropy of a random variable that takes n/α values uniformly at random. Hence,

$$\frac{1}{\frac{n}{\alpha}} \sum_{k=1}^{n/\alpha} \mathcal{N}_{q_k}^i \geq \frac{1}{\frac{n}{\alpha}} \sum_{k=1}^{n/\alpha} l_k \geq \frac{1}{2\frac{n}{\alpha}} \log\left(\frac{n}{\alpha}\right) \tag{9}$$

which completes the proof. □

Proof of Proposition 1. Since each question is between at most two branches, we have

$$Q_{\mathcal{A}} \geq \frac{1}{|\mathcal{M}|} \frac{1}{2} \sum_{q \in \mathcal{M}} \sum_{i=1}^{\alpha} \mathcal{N}_q^i = \frac{1}{2|\mathcal{M}|} \sum_{i=1}^{\alpha} \sum_{q \in \mathcal{M}} \mathcal{N}_q^i \tag{10}$$

now for a fixed i, we can divide the set of all queries, $\mathcal{M}$, into equivalence classes that in each class C queries are i-co-branch. Since there are n/α queries in each class, there are $\alpha/n|\mathcal{M}|$ classes.

$$\begin{aligned} Q_{\mathcal{A}} &\geq \frac{1}{2|\mathcal{M}|} \sum_{i=1}^{\alpha} \sum_{C} \sum_{q \in C} \mathcal{N}_q^i \geq \frac{1}{2|\mathcal{M}|} \sum_{i=1}^{\alpha} \sum_{C} \frac{1}{2} \frac{n}{\alpha} \log \frac{n}{\alpha} \\ &= \frac{1}{2|\mathcal{M}|} \alpha \frac{\alpha}{n} |\mathcal{M}| \frac{n}{\alpha} \frac{1}{2} \log \frac{n}{\alpha} = \frac{\alpha}{4} \log \frac{n}{\alpha} \end{aligned} \tag{11}$$

□

2.2 Proof of the Quadratic Bound for NNS Problem

Definition A.7 (co-SN queries)**.** *Let S be a SN in branch ϕ_i, define relation $\overset{S}{\equiv}$ in the following way: two queries q_1 and q_2 are called related by $\overset{S}{\equiv}$ if their DNs in branch ϕ_i lie in SN S and their DNs*

[3] Without loss of generality we can re-label the queries so that the first k of them belong to C_G.

[4] A sequence with elements from {0,1,2,3}.

in all other branches are the same, in addition to that the ordering of their DNs are the same; i.e., $q_1 \overset{S}{\equiv} q_2$ *if and only if* $\delta_{q_1}(i) \in S$, $\delta_{q_2}(i) \in S$, $\delta_{q_1}(j) = \delta_{q_2}(j), \forall j \neq i$ *and* $\Psi_{q_1} = \Psi_{q_2}$.

It could be seen easily that this relation is an equivalence relation over queries that have a DN in SN S, hence they are divided into classes with α queries in each class. For a given SN S, we call each of them a S-co-SN class and the queries in each class are called S-co-SN queries.

Definition A.8. *Define $\tilde{\mathcal{N}}_q^S$ to be the number of nodes that are asked in supernode S when with query q as the input to algorithm.*

Lemma A.3 (all nodes should be asked on average)**.** *For a given SN S in branch ϕ_i and a S-co-SN class $C = \{q_1, q_2, \dots q_\alpha\}$, we have*

$$\frac{1}{\alpha}\sum_{k=1}^{\alpha} \tilde{\mathcal{N}}_{q_k}^S \geq \frac{\alpha}{4} \tag{12}$$

Proof of Lemma A.3. Lets denote the nodes in S by $n_1, n_2, \dots, n_\alpha$ where $n_k = \delta_{q_k}(i)$. As a result of Corollary A.1 there should be a question about n_k in the question set of q_k. The array $\mathcal{Q}_k$ contains the questions $\mathcal{A}$ asks when q_k is its input, so $\mathcal{Q}_k[p]$ denotes the pth question asked from the oracle when q_k is input of the algorithm. Let μ_1 be the first place that in arrays $\mathcal{Q}_k$ for at least one $1 \leq k \leq \alpha$ there is a question about a node in S, *i.e.*,

$$\mu_1 = \min\{p | \exists\, 1 \leq k \leq \alpha,\ \mathcal{Q}_k[p] \text{ is about some node in } S\} \tag{13}$$

since questions before index μ_1 are all about nodes outside S and edges in S have the same weight, the answer to them are the same for $q_1, q_2, \dots, q_\alpha$, therefore questions in places 1 to $\mu_1 - 1$ and their answers are exactly the same for $q_1, \dots, q_\alpha$. As a result of this and because our algorithm is deterministic, the nodes compared in $\mathcal{Q}_k[\mu_1]$ are same for all the queries but possibly its answer is different for different k's. There are two possible situations:

1. question μ_1 is between a node n_{k_1} and another node in another SN.
2. this question is between two nodes in S, say n_{k_1} and $n_{k_1'}$.

In the first case, since for $k \neq k_1$,

$$d(n_{k_1}, q_k) = 2\frac{i}{4\alpha} + 1 + \frac{\Psi_{q_k}(i)}{\alpha}\epsilon \tag{14}$$

and the fact that q_ks are equivalent, this distance is the same for $q_k, k \neq k_1$ hence the answer to the μ_1^{th} question is the same for them.

In the second case, using the same reasoning, the answer to the question is the same for queries other than q_{k_1} and $q_{k_1'}$. Therefore

$$\mathcal{Q}_k[p] = \mathcal{Q}_{k'}[p] \quad p \leq \mu_1, k \neq k_1 \text{ (or } k \neq k_1 \text{ and } k \neq k_1' \text{ in the second case)} \tag{15}$$

We want to continue this procedure and exclude more queries. To make it formal, we start with the set of all query indexes, say $T_0 = \{1, 2, \dots, \alpha\}$ and exclude one or two queries from them to obtain T_1, in the first case $T_1 = T_0 \setminus \{k_1\}$ and in the second case $T_1 = T_0 \setminus \{k_1, k_1'\}$. Therefore T_r is the input to the rth level and T_{r+1} is its output, $r \geq 0$.

Continuing this procedure inductively, in the rth level where $T_r \neq \emptyset$ we define

$$\mu_{r+1} = \min\{p > \mu_r | \exists\, k \in T_r,\ \mathcal{Q}_k[p] \text{ is about some node } n_p, p \in T_r\} \tag{16}$$

The key point to continue the procedure is that no query goes out of the play until its direct node is asked in a question (Corollary A.1). Therefore having $T_r \neq \emptyset$, μ_{r+1} is well defined and the procedure comes to an end at level r_f when $T_{r_f} = \emptyset$.

Repeating our reasoning for $r = 0$,

- $\mathcal{Q}_k[p] = \mathcal{Q}_{k'}[p]$ for $k, k' \in T_r$ and $p < \mu_{r+1}$.

- $\mu_{r+1}{}^{\text{th}}$ question (but not necessarily its answer) is the same for queries in T_r.

Again two possible situations could happen, based on which case happens, we define T_{r+1} to be either $T_r \setminus \{k_{r+1}\}$ or $T_r \setminus \{k_{r+1}, k'_{r+1}\}$ where $q_{k_{r+1}}$ and $q_{k'_{r+1}}$ are the queries that go out of the play in level r. Just like (15),

$$\mathcal{Q}_k[p] = \mathcal{Q}_{k'}[p] \quad p \leq \mu_{r+1}, k \neq k_{r+1} \text{ (or } k \neq k_{r+1} \text{ and } k \neq k'_{r+1} \text{ in the second case)} \tag{17}$$

which says that the answer to the μ^{th}_{r+1} is the same for queries in T_{r+1}.

Now we are at the place to give the lower bound. Note that for a query q_k, if it remains in the play up to r^{th} level, i.e. $q_k \in T_{r-1}$, $\tilde{\mathcal{N}}^S_{q_k}$ is at least r, therefore,

$$\begin{aligned} \sum_{k=1}^{\alpha} \tilde{\mathcal{N}}^S_{q_k} &\geq \sum_{k=1}^{\alpha} \sum_{r, q_k \in T_{r-1}} 1 = \sum_{r \geq 0} |T_r| \\ &\geq \sum_{r=0}^{\lfloor \alpha/2 \rfloor} \alpha - 2r \geq \frac{\alpha^2}{4} \end{aligned} \tag{18}$$

hence,

$$\frac{1}{\alpha} \sum_{k=1}^{\alpha} \tilde{\mathcal{N}}^S_{q_k} \geq \frac{\alpha}{4} \tag{19}$$

□

Proof of Proposition 2. If for a query q, $S_{\delta_q(i)}$ denotes the supernode in which the direct node of q in branch ϕ_i is, $\tilde{\mathcal{N}}_q^{S_{\delta_q(i)}}$ is the number of nodes in the direct supernode of q in branch ϕ_i that $\mathcal{A}$ asks a question about them for q as input. Since every question covers at most two nodes, we have,

$$\begin{aligned} Q_{\mathcal{A}} &\geq \frac{1}{2} \frac{1}{|\mathcal{M}|} \sum_{q \in \mathcal{M}} \sum_{i=1}^{\alpha} \tilde{\mathcal{N}}_q^{S_{\delta_q(i)}} \\ &= \frac{1}{2|\mathcal{M}|} \sum_{i=1}^{\alpha} \sum_{q \in \mathcal{M}} \tilde{\mathcal{N}}_q^{S_{\delta_q(i)}} \end{aligned} \tag{20}$$

now we analyze $\sum_{q \in \mathcal{M}} \tilde{\mathcal{N}}_q^{S_{\delta_q(i)}}$ for a fixed $1 \leq i \leq \alpha$. Define $S_k, k = 1, \ldots, n/\alpha^2$ to be supernodes in branch ϕ_i. We know that $\overset{S_k}{\equiv}$ defines an equivalence relation over queries q that $\delta_q(i) \in S_k$. Since the set of queries q that $\delta_q(i) \in S_k$ are distinct for different ks and their union is $\mathcal{M}$, we can break the summation into equivalence classes C for all relations $\overset{S_k}{\equiv}$, for $k = 1, \ldots, n/\alpha^2$. For one equivalence class C, according to Lemma A.3 we know that

$$\sum_{q \in C} \tilde{\mathcal{N}}_q^{S_{\delta_q(i)}} \geq \frac{\alpha^2}{4} \tag{21}$$

since every equivalence class has α elements,

$$\sum_{q \in \mathcal{M}} \tilde{\mathcal{N}}_q^{S_{\delta_q(i)}} = \sum_{C} \sum_{q \in C} \tilde{\mathcal{N}}_q^{S_{\delta_q(i)}} \geq \sum_{C} \frac{\alpha^2}{4} = \frac{|\mathcal{M}|}{\alpha} \frac{\alpha^2}{4} = \frac{\alpha}{4} |\mathcal{M}| \tag{22}$$

combining this with (20) we have:

$$Q_{\mathcal{A}} \geq \frac{1}{2|\mathcal{M}|} \sum_{i=1}^{\alpha} \frac{\alpha}{4} |\mathcal{M}| = \frac{1}{2|\mathcal{M}|} \frac{\alpha^2}{4} |\mathcal{M}| = \frac{1}{8} \alpha^2 \tag{23}$$

which completes the proof. □

2.3 The Algorithm that Achieves the Lower Bound for the Star Graph

In this section, we present an algorithm that achieves the lower bound proved for the star shaped graph.

We can find the DS in each branch by using a binary search method: For a given branch ϕ, denote the supernodes on it by $S_1, S_2, \ldots, S_{n/\alpha^2}$ (S_1 is the closest to the center of the star, and the S_{n/α^2} is the last SN). We can divide these supernodes into two subsets $\mathcal{S}_1 = \{S_1, \ldots, S_{n/(2\alpha^2)}\}$ and $\mathcal{S}_2 = \{S_{1+n/(2\alpha^2)}, \ldots, S_{n/\alpha^2}\}$. Pick a node n_1 from the SN at the middle of $\mathcal{S}$, (*i.e.*, $S_{n/(4\alpha^2)}$) and a node n_2 from the SN at the middle of $\mathcal{S}_2$, (*i.e.*, $S_{3n/(4\alpha^2)}$) and compare them using one question from the oracle. According to **(O3)** in the proof of Lemma A.1, if n_1 is closer to the query, we may conclude that DS of branch ϕ is in segment $\mathcal{S}_1$, otherwise the DS belongs to $\mathcal{S}_2$. This process of partitioning and binary search will be recursively continued in $O(\log n/\alpha^2)$ steps until we find the DS on branch ϕ. The same procedure can be repeated for each branch. Hence we should ask $O(\alpha \log n/\alpha^2)$ questions to find all the α DSs.

There are α nodes in each DS. Therefore using α question we can find the DN in each DS, which yields in a total of α^2 questions to find all the DNs. Finally we can find the NN among the α DNs using α comparisons. Hence, we need $\alpha^2 + \alpha = O(\alpha^2)$ questions for this phase of the algorithm. Therefore this algorithm runs in $O(\alpha^2 + \alpha \log n/\alpha^2)$ questions from the oracle.

3 Additional details for the Hierarchical Algorithm

3.1 Background algorithms

We first present the missing macros which are used in the learning algorithm.

A heap is a binary tree structure. Every node of the tree is stored in an element of an array. A heap is a nearly complete binary tree, meaning that it is completely filled on all levels except possibly the lowest.

The array containing the node values, H, has two attributes length(H) which is the length of the array and heap-size(H) which is the number of tree nodes. Therefore $H[1, \ldots,$ heap-size$(H)]$ contain heap values and the data in $H[$heap-size$(H) + 1, \ldots,$ length$(H)]$ are not valid.

The root of the tree is $H[1]$. Every node i, except the nodes at the lowest level has two children, the left child left$(i) = 2i$ and the right child right$(i) = 2i + 1$. Therefore the parent node of node j is parent$(j) = \lfloor j/2 \rfloor$.

There are two kinds of binary heaps: min-heaps and max-heaps. In a min-heap we have $H[\text{parent}(i)] \leq H[i]$ for all node i, and in a max-heap we have $H[\text{parent}(i)] \geq H[i]$. In this paper we work with min-heaps, therefore from now on we assume every heap is a min-heap. An example of a min-heap is depicted in Fig. A.2.

A heap of n elements is based on a complete binary tree, hence the height of such a heap is of $\Theta(\log n)$. Now we explain three basic algorithms to build, maintain and use a min-heap (the procedure for max-heap is the same, hence all following algorithms will be explained for min-heap).

- `Heapify`: modifies the heap to maintain the heap property and returns in $O(\log n)$ time.
- `BuildHeap`: builds a heap from unordered data.
- `ExtractMin`: extracts the minimum element (which is the root node) and modifies the other elements to make a new heap and runs in $O(\log n)$.

The Heapify algorithm (Algorithm 1) takes an array H and an index i. It assumes that the sub-trees rooted at left(i) and right(i) are heaps but possibly $H[i]$ is bigger than its children and violates the heap property. The algorithm lets the value of node i flow down to maintain the heap property.

We use this algorithm recursively to implement `BuildHeap` (Algorithm 2).

To extract the minimum of the heap (or the root node), we replace the root with the last element in the array and call `Heapify` procedure on root. This is explained in Algorithm 3.

```
input  : An array H and an index i, assuming sub-trees H[left(i)] and H[right(i)] are
         min-heaps.
output: Array H with sub-tree at node i being a min-heap

l ← left(i);
r ← right(i);
if l ≤ heap-size(H) and H[l] < H[i] then
 | smallest ← l;
else
 | smallest ← i;
end
if r ≤ heap-size(H) and H[r] < H[smallest] then
 | smallest ← r;
end
if smallest ≠ i then
 | exchange H[i] ↔ H[smallest] ;
 | Heapify(H, smallest);
end
```

Algorithm 1: Heapify

```
input  : Array H with length length(H)
output: min-heap H

heap-size(H) ← length(H);
for i ← ⌊length(H)/2⌋ downto 1 do
 | Heapify(H, i)
end
```

Algorithm 2: BuildHeap

```
input  : A heap H
output: H with its root extracted
if heap-size(H) < 1 then
 | error heap underflow;
end
min ← H[1];
A[1] ← A[heap-size(H)];
heap-size(A) ← heap-size(A) − 1;
Heapify(A, 1);
return min
```

Algorithm 3: ExtractMin

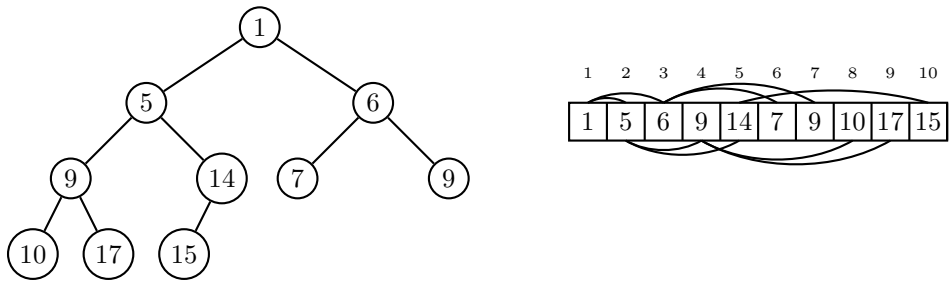

Figure A.2: An example of a min-heap with 10 nodes and its implementation in an array

3.2 Success of the Algorithm

We first prove two technical lemmas that we will need to prove Lemma 2.

Lemma A.4. *If we throw $m = ab \log n$ balls into b bins, each chosen uniformly at random, then the first bin will contain at least one ball with probability more than $1 - \frac{1}{n^a}$*

Proof of Lemma A.4. The probability that a bin contains no ball is

$$\begin{aligned} \mathcal{P}\left[\text{a bin contains no ball}\right] &= (1 - \tfrac{1}{b})^{ab \log n} \\ &\leq e^{-a \log n} \\ &= \tfrac{1}{n^a} \end{aligned}$$

□

Lemma A.5. *We throw m balls into n bins, each chosen uniformly at random. We number the bins from 1 to n. Then, the probability that the total number of balls in bins 1 to $\frac{n}{c}$ being more than $(1+\tau)m/c$ or less than $(1-\tau)m/c$ is at most $2e^{-\tau^2 m/3c}$.*

Proof of Lemma A.5. We throw the balls one after the other into the bins. Let $X_i = 1$ if the ith ball falls in one of the $\frac{n}{c}$ first bins, and 0 else. Let $X = \sum_i X_i$. Clearly, we have $\mathbb{E}[X] = m/c$, as $\mathcal{P}[X_i = 1] = 1/c$ and all X_i's are independent. By the Chernoff Bound (see for instance [1], page 67), we have $\mathcal{P}\left[|X - \mathbb{E}[X]| > \tau m/c\right] < 2e^{-\tau^2 m/3c}$. □

Proof of Lemma 2. Fix an object p and a level i. To visualize the proof, place all objects in the database on a line, such that the object u with rank $r_p(u, \mathcal{T}) = r$ is located at distance r from p (see Fig. A.3). Property 1 tells us that at least one of the samples at level i will be such that its rank w.r.t. p is smaller than λ_{i+1} *i.e.*, $\exists s \in S_i$ s.t. $r_p(s) \leq \lambda_{i+1}$. Clearly, by Lemma A.4, this is true with probability at least $1 - \frac{1}{n^a}$ (set $m = m_i$ and $b = (2D)^i = \frac{n}{\lambda_{i+1}}$ in the lemma). Property 2 tells us that not too many objects can have rank less than λ_i at level i w.r.t. o. Let $c = \frac{n}{\lambda_i} = (2D)^{i-1}$. Now, by Lemma A.5 (set $m = m_i = a(2D)^i \log n$ and $\tau = 1$), the probability that more than $2a(2D)^i \log n/(2D)^{i-1} = 4aD \log n$ samples are among the $\lambda_i = \frac{n}{c}$ closest samples to p is less than $2e^{-2aD \log n/3} = \frac{1}{n^{\Omega(a)}}$. The proof of Property 3 is identical, except that we replace λ_i by λ_{i-1}. Then, we have $c = (2D)^{i-2}$, $\frac{2m_{i+1}}{c} = 16aD^3 \log n$, and the probability that $|S_{i+1} \cap \mathcal{B}_p(\lambda_{i-1})| > 16aD^3 \log n$ is smaller than $\frac{1}{n^{\Omega(a)}}$, as before. For Property 4, we expect $8aD \log n$ objects to be sampled at level i among the $4\lambda_i$ closest objects to p. Again, by setting $\tau = 0.5$ in Lemma A.5, the probability that less than half that many objects get sampled is at most $\frac{1}{n^{\Omega(a)}}$. Finally, the proof of Property 5 is almost identical to the proof of Property 2. By choosing a large enough, we can make sure that the five properties are true for all objects and all levels w.h.p. (take the union bound over the n objects and the $L = \frac{\log n}{\log 2D}$ levels). Roughly speaking, there are of order $n \log n$ inequalities, using union bound we realize that if we take $a \geq 3$, the error probability will be of order $1/n$. As we see, a is a constant independent from n and D.

□

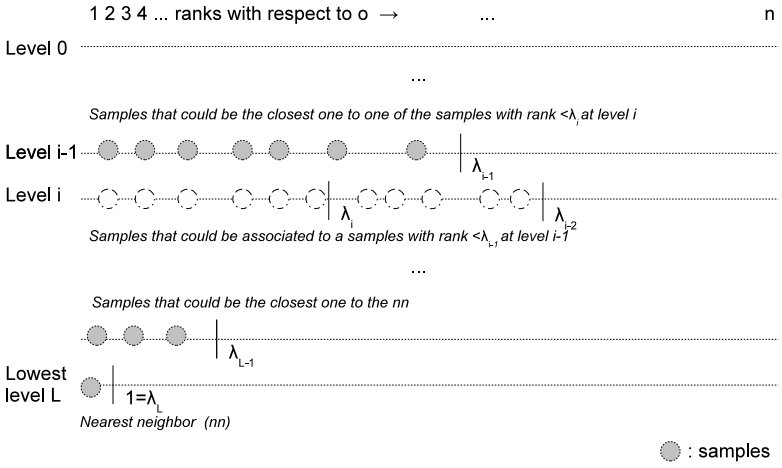

Figure A.3: We place all objects on the line such that the object u with rank $r_p(u, \mathcal{T}) = r$ is located at distance r from p.

References

[1] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.