Mach Learn (2018) 107:1069–1094
https://doi.org/10.1007/s10994-018-5711-7

Improved maximum inner product search with better theoretical guarantee using randomized partition trees

Omid Keivani[1] · **Kaushik Sinha**[1] · **Parikshit Ram**[2]

Received: 12 May 2017 / Accepted: 5 April 2018 / Published online: 23 April 2018

Abstract Recent interest in the problem of maximum inner product search (MIPS) has sparked the development of new solutions. The solutions (usually) reduce MIPS to the well-studied problem of nearest-neighbour search (NNS). To escape the curse of dimensionality, the problem is relaxed to accept approximate solutions (that is, accept anything that is approximately maximum), and locality sensitive hashing is the approximate NNS algorithm of choice. While being extremely resourceful, these existing solutions have a couple of aspects that can be improved upon—(i) MIPS can be reduced to NNS in multiple ways and there is lack of understanding (mostly theoretical but also empirical) when to choose which reduction for best accuracy or efficiency, and (ii) when MIPS is solved via approximate NNS, translating this approximation to the MIPS solution is not straightforward. To overcome these usability issues, we propose the use of randomized partition trees (RPTs) for solving MIPS. We still reduce MIPS to NNS but utilize RPTs to solve the NNS problem. RPTs find the *exact* NNS solution, hence the exact MIPS solution (with high probability), avoiding the need for any translation of approximation. The theoretical properties of RPTs allow us to definitively choose the best MIPS-to-NNS reduction. The empirical properties of RPTs allow us to significantly outperform the state-of-the-art while providing unique fine-grained control over the accuracy–efficiency trade-off. For example, at 80% accuracy, RPTs are 2–5× more efficient than the state-of-the-art. Superiority of RPT comes at the cost of high space complexity overhead which can be a severe limitation of our proposed method. To address this limitation we

Editor: Tapio Elomaa.

✉ Kaushik Sinha
kaushik.sinha@wichita.edu

Omid Keivani
oxkeivani@shockers.wichita.edu

Parikshit Ram
p.ram@gatech.edu

[1] Department of Electrical Engineering and Computer Science, Wichita State University, Wichita, KS 67260, USA

[2] Skytree Inc., Atlanta, GA 30332, USA

introduce two space efficient versions of RPTs which enjoy the same superior performance of RPT while requiring a significantly reduced space complexity overhead.

Keywords Maximum inner product search · Nearest neighbor search · Random projection trees · Locality sensitive hashing

1 Introduction

The problem of maximum inner product search (MIPS) has received considerable attention in recent years due to its wide usage in problem domains such as matrix factorization based recommender systems (Koren et al. 2009; Srebro et al. 2005; Cremonesi et al. 2010), multi-class prediction with large number of classes (Dean et al. 2013; Jain and Kapoor 2009), large scale object detection (Dean et al. 2013; Felzenszwalb et al. 2010) and structural SVMs (Joachims 2006; Joachims et al. 2009). The problem of MIPS is as follows: given a set $S \subset \mathbb{R}^d$ of d-dimensional points and a query point $q \in \mathbb{R}^d$, the task is to find a $p \in S$ such that,

$$p = \arg\max_{x \in S} q^\top x. \tag{1}$$

A naive way to solve this problem is to perform a linear search over S, which often becomes impractical as the size of S increases. The goal is to develop sub-linear algorithms to solve MIPS. Towards this end, recent work (Shrivasatava and Li 2014, 2015; Bachrach et al. 2014; Neyshabur and Srebro 2015) proposed the solution of MIPS with algorithms for nearest-neighbour search (NNS) by presenting transformations to all points $x \in S$ and the query q which reduce the problem of MIPS to a problem of NNS.[1] The sublinear *approximate* NNS algorithm used for MIPS is locality sensitive hashing (LSH) (Andoni and Indyk 2008; Gionis et al. 1999; Datar et al. 2004). While LSH is widely used for approximate NNS (and now for approximate MIPS), it is known to not provide a good control over the accuracy–efficiency trade-off for approximate NNS or approximate MIPS. Moreover, specific to the problem of MIPS, it is not clear which of the multiple proposed reductions (Shrivasatava and Li 2014, 2015; Bachrach et al. 2014; Neyshabur and Srebro 2015) would provide the best accuracy or efficiency (or the best trade-off of the two). We will discuss both these issues in more detail in the following section. These issues raise two questions that we address here:

- Can we develop a MIPS solution which provides fine-grained control over the accuracy–efficiency trade-off?
- Can we definitively choose the best MIPS-to-NNS reduction in terms of the accuracy–efficiency trade-off?

To this end, we propose the use of an ensemble of randomized partition trees (RPTs) (Dasgupta and Sinha 2013, 2015) for MIPS. RPTs have been shown to demonstrate favorable accuracy–efficiency trade-off for NNS relative to LSH while providing fine-grained control on the trade-off spectrum (Sinha 2014). We demonstrate that this usability of RPTs seamlessly transfers over to the problem of MIPS, thereby addressing the first question. Moreover, RPTs theoretically bound the probability of failing to find the *exact* nearest-neighbor as opposed to the failure probability guarantees of LSH which only apply to the approximate nearest-neighbor.[2] We build upon this theoretical property of RPTs to address the second question

[1] More precisely, a true nearest neighbor of transformed q in transformed S is the transformed form of p, where p is a solution of the MIPS problem defined in Eq. 1.

[2] We will further discuss these points in the next section.

by providing a theoretical ordering of the existing MIPS-to-NNS reductions with respect to the performance of RPTs.

Note that, both LSH and RPT performs sub-linear time nearest neighbor search by adopting a simple strategy: given a query point, it restrict its search only to a small subset of the data points that lie in an appropriate hash bucket (in case of LSH) or a leaf node (in case of RPT). As mentioned earlier, ensuring fine-grained control over accuracy efficiency trade-off is a problem in case of LSH as it is not easy to specify number of data points that might lie in any hash bucket, resulting in either inefficient search by retrieving too many data points when a hash bucket is overcrowded or inaccurate search by retrieving too few data points when hash bucket is either empty or contains too few data points. On the other hand, in case of RPT, maximum number of data points that any leaf node of an RPT might contain (and thereby the number of retrieved points) is specified upfront during the tree construction resulting in a fine-grained control on the accuracy efficiency trade-off spectrum and therefore RPT is often preferred over LSH for solving NNS problem (Sinha 2014). However, this fine-grained control over accuracy efficiency trade-off spectrum comes at a cost of expensive storage requirement as each internal node of an RPT needs to store a d dimensional random projection direction and scalar split point pair. For high dimensional data (d is large) this results in a larger space complexity for an RPT data structure as compared to LSH data structure. To address this issue, we present two space saving strategies that results in reduced space complexity for RPT data structure without compromising its favorable accuracy–efficiecy trade-off. Combining this with the best MIPS-to-NNS reduction as per our our theoretical findings, we present an efficient way to solve MIPS problem with better theoretical guarantees and lower space complexity.

Rest of the paper[3] is organized as follows. In Sect. 2, we provide some background on the existing solutions for the MIPS problem and clearly present their shortcomings. We provide the details regarding RPTs and their properties in the same section. In Sect. 3, we present our proposed solution for MIPS along with our main theoretical results. We present two space efficient strategies to reduce space complexity of RPT data structure in Sect. 4. In Sect. 5, we empirically validate our theoretical results and demonstrate the superiority of our solution for MIPS to existing solutions. We reiterate our contributions and conclude with limitations of our proposed work and some open questions in Sect. 6.

2 Background and related work

In this section, we discuss the existing solutions to exact and approximate MIPS and present their limitations. We continue on to introduce RPTs for NNS and motivate their use for MIPS by demonstrating how RPTs overcome the limitations of existing solutions for MIPS.

2.1 Existing solutions for MIPS

MIPS has received a lot of recent attention. Linear search over the set S scales as $O(d|S|)$ per query, becoming prohibitively high for moderately large sets. There was an attempt to solve exact MIPS using a space-partitioning tree and a branch-and-bound algorithm in the original input space (Ram and Gray 2012; Curtin et al. 2013; Curtin and Ram 2014). These branch-and-bound algorithms were shown to have a logarithmic scaling on $|S|$ but the dependence

[3] A shorter version of this paper appeared in the proceedings of the International Joint Conference on Neural Networks (IJCNN), 2017 (Keivani et al. 2017).

on the dimensionality was exponential, limiting their usability to small or moderate number of dimensions. To improve upon this, the problem of MIPS was approximated, reduced to the problem of NNS and solved using an approximate NNS algorithm.

Approximate MIPS. The problem of MIPS is approximated in the following way: given the set $S \subset \mathbb{R}^d$, a query $q \in \mathbb{R}^d$ and an approximation parameter ϵ, the task is to find any $p' \in S$ such that

$$q^\top p' \geq (1-\epsilon) \max_{x \in S} q^\top x. \tag{2}$$

Connecting MIPS to NNS. Recent work (Shrivasatava and Li 2014, 2015; Bachrach et al. 2014; Neyshabur and Srebro 2015) pointed out that MIPS and NNS are closely related and a MIPS problem can be reduced to an NNS problem by applying appropriate transformation to the points in the set S and the query q. We list four such transformations below. Typically these transformations add extra dimensions to the points in S as well to query q so that solving MIPS in original space is equivalent to solving NNS in the transformed higher dimensional space. We use $P_i(\cdot)$ and $Q_i(\cdot)$ to denote the i^{th} transformations applied to data points and query point respectively.

- **Transformation 1 (T1):** $P_1 : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ and $Q_1 : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ are defined as follows:

$$P_1(x) = \left(\frac{x}{\beta}, \sqrt{1 - \frac{\|x\|_2^2}{\beta^2}} \right), \quad Q_1(x) = \left(\frac{x}{\|x\|_2}, 0 \right),$$

  with $\beta = \max_{x \in S} \|x\|_2$ is the maximum norm among all data points in S (Neyshabur and Srebro 2015).
- **Transformation 2 (T2):** $P_2 : \mathbb{R}^d \rightarrow \mathbb{R}^{d+2}$ and $Q_2 : \mathbb{R}^d \rightarrow \mathbb{R}^{d+2}$ are defined as follows:

$$P_2(x) = \left(\frac{x}{\beta_1}, \sqrt{1 - \frac{\|x\|_2^2}{\beta_1^2}}, 0 \right), Q_2(x) = \left(\frac{x}{\beta_1}, 0, \sqrt{1 - \frac{\|x\|_2^2}{\beta_1^2}} \right).$$

  Here $\beta_1 = \max \left\{ \max_{x \in S} \|x\|_2, \max_q \|q\|_2 \right\}$ is maximum norm among all data points in S as well as all possible query points[4] (Neyshabur and Srebro 2015).
- **Transformation 3 (T3):** $P_3 : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ and $Q_3 : \mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$ are defined as follows:

$$P_3(x) = \left(x, \sqrt{\beta^2 - \|x\|_2^2} \right), \quad Q_3(x) = (x, 0),$$

  where $\beta = \max_{x \in S} \|x\|_2$ is maximum norm among all data points in S (Bachrach et al. 2014).
- **Transformation 4 (T4):** $P_4 : \mathbb{R}^d \rightarrow \mathbb{R}^{d+m}$ and $Q_4 : \mathbb{R}^d \rightarrow \mathbb{R}^{d+m}$ are defined as follows:

$$P_4(x) = \left(\frac{x}{\alpha}, \frac{\|x\|_2^2}{\alpha^2}, \ldots, \frac{\|x\|_2^{2^m}}{\alpha^{2^m}} \right), Q_4(x) = \left(\frac{x}{\|x\|_2}, \frac{1}{2}, \ldots, \frac{1}{2} \right).$$

  Here $\beta = \max_{x \in S} \|x\|_2$ is maximum norm among all data points in S and $\alpha = c\beta$ for some $c > 1$ (Shrivasatava and Li 2015).

Each of the above transformations satisfy the following:

[4] Note that for this transformation, the maximum norm over all possible queries is needed in advance.

Theorem 1 *Suppose $S \subset \mathbb{R}^d$ is a set of data points and $q \in \mathbb{R}^d$ is a query point. Then the following holds:*

$$\begin{aligned}\arg\max_{x \in S} q^\top x &= \arg\min_{x \in S} \|P_1(x) - Q_1(q)\|_2 \\ &= \arg\min_{x \in S} \|P_2(x) - Q_2(q)\|_2 \\ &= \arg\min_{x \in S} \|P_3(x) - Q_3(q)\|_2 \\ &= \arg\min_{x \in S} \left(\lim_{m \to \infty} \|P_4(x) - Q_4(q)\|_2\right).\end{aligned}$$

Proof For different transformations, the theorem have already been proven in different literature. For the sake of completeness, we provide the proof below. Note that for any transformation $\mathbf{Tj}, \mathbf{j} = 1, \ldots, 4$, we have,$\|Q_j(q) - P_j(x)\|^2 = \|Q_j(q)\|^2 + \|P_j(x)\|^2 - 2Q(q)^\top P_j(x)$. after doing simple algebraic calculations, for different transformations this yields to:

$$\|Q_1(q) - P_1(x)\|^2 = 2\left(1 - \frac{q^\top x}{\|q\|\beta}\right) \tag{3}$$

$$\|Q_2(q) - P_2(x)\|^2 = 2\left(1 - \frac{q^\top x}{\beta_1^2}\right) \tag{4}$$

$$\|Q_3(q) - P_3(x)\|^2 = \beta^2 + \|q\|^2 - 2q^\top x \tag{5}$$

$$\|Q_4(q) - P_4(x)\|^2 = (1 + m/4) - (2/\alpha)\frac{q^\top x}{\|q\|} + \left(\frac{\|x\|_2}{\alpha}\right)^{2^{m+1}} \tag{6}$$

It is easy to see from Eqs. 3, 4, 5 that maximizing $q^\top x$ is same as minimizing $\|Q_j(q) - P_j(x)\|$ for $j = 1, 2, 3$ which corresponds to transformations **T1** , **T2** , **T3** . Since the last term of Eq. 6 tends to zero as $m \to 0$, the same holds for Transformation **T4** as well. □

The above theorem[5] simply says that each of the four transformations reduces a MIPS problem to an NNS problem and the solution to the **exact** NNS problem in the transformed space is the solution to the **exact** MIPS problem.

2.2 Approximating MIPS via LSH

LSH approximates the NNS problem in the following manner: exact NNS tries to find the $p \in S$ such that $p = \arg\min_{x \in S} \|q - x\|_2$; LSH solves the approximate version of the problem with an approximation parameter ϵ to find any $p' \in S$ such that $\|q - p'\|_2 \leq (1 + \epsilon) \min_{x \in S} \|q - x\|_2$.

LSH is known to scale sub-linearly to $|S|$ and polynomially to the dimensionality d under favorable conditions. However, LSH comes with some known shortcomings for NNS. LSH parameters (hash code length and number of hash tables) do not allow the user to have fine-grained control over the accuracy–efficiency trade-off. For example, specifying a particular hash code length and number of hash tables does not provide any information on the maximum number of points that might be retrieved. This is due to the fact that LSH builds hash tables on a grid irrespective of the data density, and hence can have buckets with no points or with large number of points for the same data set. These issues directly transfer over to the problem of approximate MIPS. However, a lot of research has been done to improve the performance and

[5] We provide the proof in the supplementary material for ease of readability and due to space limitation.

the accuracy–efficiency trade-off of LSH. But many of these improvements either require a deep understanding of LSH which is limited to LSH experts or are based on computationally intensive data-dependent indexing schemes without any theoretical guarantees.

In addition to this, there are some issues with LSH which apply only to the MIPS problem. Firstly, with multiple ways of reducing MIPS to NNS, it is not (theoretically) clear which transformation is best for solving approximate MIPS via LSH.[6] Secondly, after transforming the data with functions P and Q, if LSH is solving the $(1+\epsilon)$-approximate NNS in the transformed space, it translates to a $(1-f(\epsilon, P, Q, \beta))$-approximate MIPS solution where f is some function. For an user to control the approximation in the MIPS problem, they have to carefully translate the approximation desired with LSH, adversely affecting the usability of LSH for approximate MIPS.

We propose the use of randomized partition trees (RPTs) (Dasgupta and Sinha 2013, 2015) for MIPS. We will provide details regarding RPTs in the following subsection. We believe that RPTs avoid the aforementioned shortcomings of LSH for MIPS as follows:

- NNS with RPTs allows the user to have fine-grained control over the accuracy–efficiency tradeoff—the user just needs to set two paramters, the maximum leaf size n_0 (for each tree) and the number of trees L, and the maximum number of retrieved points is upper bounded by $L \cdot n_0$. We will show how this property will seamlessly transfer to the task of MIPS.
- RPTs provide guarantees of the following form for NNS with a given set S and a query q—the probability of finding the *exact* nearest neighbors of q with each RPT is at least some $\rho \in (0, 1)$ where ρ depends on q and S. Using L trees boosts this probability of finding the *exact* neighbors to at least $1-(1-\rho)^L$. With Theorem 1, for a set S, a query q and transformations (P, Q), we know that the *exact* NNS solution in the transformed space **is** the *exact* MIPS solution. The only change for MIPS is that ρ now depends on q, S *and* (P, Q). Unlike LSH, we no longer need to translate the approximation ϵ in the NNS solution to a different approximation $f(\epsilon, P, Q, \beta)$ for the MIPS solution.
- The quantity ρ for RPTs is (theoretically) controlled by an intuitive *potential function* (Dasgupta and Sinha 2013). Since, for MIPS, ρ depends on q, S and (P, Q), we will be able to present a theoretical ordering of the values of ρ for the aforementioned transformations $(P_i, Q_i)\ \forall i = 1, \ldots, 4$, thereby, definitively answering the question "which is the best transformation for solving MIPS via RPTs?", and relinquishing the user from having to choose a transformation (unlike in MIPS with LSH where the user needs to make such a choice and then translate the approximation guarantee).

2.3 Randomized partition trees

A randomized partition tree is a *binary* space partitioning tree data structure, whose root node represents (a subset of) the space containing the complete set S of n objects. The tree is constructed recursively from the root node by splitting each node in a randomized fashion to create left and right child nodes until each node contains at most a pre-specified number of points n_0. At each internal node of the tree, randomized splitting is achieved by choosing a random projection direction and a random split point.

A nearest neighbor (NN) query is answered by routing the query to a leaf node in a RPT, following the path from root to leaf using the aforementioned splitting rules, and returning

[6] Recent work has identified transformations which preserve locality sensitive property better (Shrivasatava and Li 2015; Neyshabur and Srebro 2015). But there is no clear understanding (to the best of our knowledge) of how that translates to the final MIPS performance.

the NN from within that leaf node. A RPT fails to find a query's NN if, at any of the internal nodes along the path from the root to a leaf node, the query and its NN lie on opposite sides of the split. At an internal node containing n' of the n data points from set $S = \{x_1, \ldots, x_n\}$ and a query q, probability of such an event can be bounded by a quantity $\Phi_{n'}(q, S) \log\left(\frac{2}{\Phi_{n'}(q,S)}\right)$, where $\Phi_{n'}(q, S)$ is called the potential function and is defined as

$$\Phi_{n'}(q, S) = \frac{1}{n'} \sum_{i=2}^{n'} \left(\frac{\|x_{(1)} - q\|_2}{\|x_{(i)} - q\|_2} \right), \tag{7}$$

where $x_{(1)}, x_{(2)}, \ldots$ denotes an ordering of the x_i by increasing ℓ_2 distance from q (Dasgupta and Sinha 2013, 2015). By design, $\Phi_{n'}(q, S) \in [0, 1]$. Intuitively, the smaller the the potential function value, the easier the NNS problem and higher the chance that a particular RPT will return the true NN of q.

3 Maximum inner product search with RPTs

Given transformations (P, Q), we can solve MIPS with RPTs by first preprocessing S as follows:

- Choose RPT parameters n_0 and L,
- Generate set $P(S) = \{P(x) \forall x \in S\}$,
- Build L RPTs $\tau_1, \ldots, \tau_L$ on $P(S)$ with leaf size n_0.

For a query q, let $S^l(q) \subset S$ be the points in the leaf of τ_l containing[7] $Q(q)$. The MIPS solution for q is obtained by:

- Generate $Q(q)$ and initialize candidate set $R = \emptyset$,
- For $l = 1, \ldots, L$, $R = R \cup S^l(q)$,
- **return** $\arg\max_{x \in R} q^\top x$

The RPT construction is detailed in Algorithm 1. The routine `ChooseRule` presented in Algorithm 2 generates the randomized splits at each internal node of the tree. A projection direction is chosen uniformly at random from surface of a unit sphere $\mathcal{S}^{d-1}$ and a split point is chosen randomly to create left and right child nodes. Note that to solve MIPS problem, these trees are used in the transformed space after applying appropriate transformation that ensures MIPS-to-NNS reduction.

By construction, RPTs are balanced and require $O\left(n_0 + \log(n/n_0)\right)$ time per tree which is $O(L \log n)$ for a small constant $n_0 \ll n$ and L trees.[8] The probability of success depends on the value of the potential function (Eq. 7). The following corollary of Theorem 1 defines this potential function for the MIPS problem:

Corollary 1 *Given a set $S \subset \mathbb{R}^d$ of n data points and a query $q \in \mathbb{R}^d$, let $(x_{(1)}, x_{(2)}, \ldots, x_{(n)})$ be an ordering, satisfying $q^\top x_{(i)} \geq q^\top x_{(i+1)}$ for $i = 1, \ldots, n-1$. For any $j = 1, \ldots, 4$, suppose transformation (P_j, Q_j) is applied to (S, q). Then the following hold.*

[7] Here we have slightly abused the notation. Note that an RPT does not store the actual d dimensional data points but store only their indices (row numbers of given data matrix, where each row represents a data point) at the leaf nodes. Since after applying transformation (P, Q), transformed data points and query live in a higher dimensional space, subset of transformed data points $P(S^l(q))$ will lie in the leaf node containing $Q(q)$, where indices of $S^l(q)$ and $P(S^l(q))$ are identical.

[8] $O(\log(n/n_0))$ time is required to route a query to its corresponding leaf. Then, at most n_0 points are processed at this leaf.

Algorithm 1 Randomized Partitioning Tree

Input : data $S = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$, maximum number of data points in leaf node n_0, depth of the tree so far d_l
Initialize : tree depth $d_l = 0$
Output : tree data structure
function `MakeTree`(S, n_0, d_l)

1: **if** $|S| \leq n_0$ **then**
2: **return** leaf containing S
3: **else**
4: $d_l = d_l + 1$
5: Rule = **ChooseRule**(S, d_l)
6: LeftTree = `MakeTree`$(\{x \in S : \text{Rule} = \text{true}\}, n_0, d_l)$
7: RightTree = `MakeTree`$(\{x \in S : \text{Rule} = \text{false}\}, n_0, d_l)$
8: **return** (Rule, `LeftTree, RightTree`)
9: **end if**

Algorithm 2 Function `ChooseRule` for RPT

Input : data S, depth of the current node from root d_l
Output : rule
function `ChooseRule`(S, d_l)

1: Pick U uniformly at random from the unit sphere by choosing each of its coordinate independently at random from a standard Normal distribution
2: Pick β uniformly at random from $[1/4, 3/4]$
3: Let v be the β-fractile point on the projection of S onto U
4: Rule$(x) = (x \cdot U \leq v)$
5: **return** (Rule)

(i) $(P_j(x_{(1)}), P_j(x_{(2)}), \ldots, P_j(x_{(n)}))$ *is an ordering satisfying,* $\|P_j(x_{(i)}) - Q_j(q)\|_2 \leq \|P_j(x_{(i+1)}) - Q_j(q)\|_2$, *for* $i = 1, \ldots, n-1$.
(ii) *For any subset* S'_j *of* $S_j = \{P_j(x_{(1)}), \ldots, P_j(x_{(n)})\}$, *the potential function* $\Phi_{|S'_j|}(Q_j(q), S_j)$ *is defined as:*

$$\Phi_{|S'_j|}(Q_j(q), S_j) = \frac{1}{|S'_j|} \sum_{i=2}^{|S'_j|} \left(\frac{\|P_j(x_{(1)}) - Q_j(q)\|_2}{\|P_j(x_{(i)}) - Q_j(q)\|_2} \right) \tag{8}$$

Proof Consider any pair $(x_{(i)}, x_{(i+1)})$ such that $q^\top x_{(i)} \geq q^\top x_{(i+1)}$. From Eqs. 3, 4, 5, it is easy to see that $\|Q_j(q) - P_j(x_{(i+1)})\| - \|Q_j(q) - P_j(x_{(i)})\| \geq 0$ for any $j = 1, 2, 3$. Moreover, as can be seen from Eq. 6 that the same holds for $j = 4$ as $m \to 0$. Considering all the pairs for $i = 1, 2, \ldots, (n-1)$, it is easy to see that $\|Q_j(q) - P_j(x_{(1)})\| \leq \|Q_j(q) - P_j(x_{(2)})\| \leq \cdots \leq \|Q_j(q) - P_j(x_{(n)})\|$. Combining this fact with the definition of potential function from Eqs. 7, 8 follows. □

Lower values of this potential function imply higher probabilities of success in finding the exact MIPS solution. This puts us in a unique position—finding the transformation $\mathbf{T}^* = (P^*, Q^*)$ which achieves the lowest potential value among all transformations **T1** –**T4** is sufficient to pick the best transformation in terms of the MIPS accuracy for fixed efficiency. To this end, we consider each term in the right hand side of Eq. 8, which corresponds to relative placement of any two data points $x, y \in \mathbb{R}^d$ and query point $q \in \mathbb{R}^d$ such that $q^\top y \geq q^\top x$. Applying any of the four transformations (P_j, Q_j) $(j = 1, \ldots, 4)$ ensures that the ratio $\frac{\|P_j(y) - Q_j(q)\|}{\|P_j(x) - Q_j(q)\|} \leq 1$. The following theorem shows that $\mathbf{T1} = (P_1, Q_1)$ achieves the smallest (among the four considered) ratio for any q, x and y:

Theorem 2 *Let $x, y \in \mathbb{R}^d$ be any two data points and let $q \in \mathbb{R}^d$ be a query point such that $q^\top y \geq q^\top x$. Suppose transformation **T4** uses $c > 1$ and m satisfying $m \geq 4\left(\frac{2}{c} - 1\right)$. Then, $\frac{\|Q_1(q)-P_1(y)\|}{\|Q_1(q)-P_1(x)\|} \leq \gamma$, where, $\gamma = \min\left\{\frac{\|Q_2(q)-P_2(y)\|}{\|Q_2(q)-P_2(x)\|}, \frac{\|Q_3(q)-P_3(y)\|}{\|Q_3(q)-P_3(x)\|}, \frac{\|Q_4(q)-P_4(y)\|}{\|Q_4(q)-P_4(x)\|}\right\}$.*

Proof First, we prove a simple Lemma that will be used in our proof.

Lemma 1 *Let a, b two positive scalars such that $\frac{a}{b} \leq 1$. For positive scalars x, y such that $x \leq y$, the following holds:*

$$\frac{a}{b} \leq \frac{a+x}{b+x} \leq \frac{a+y}{b+y}$$

Proof The first inequality follows by observing that $a \leq b \Rightarrow ax \leq bx \Rightarrow ab+ax \leq ab+bx$ and then rearranging terms. Since $(y-x) \geq 0$, the second inequality follows in similar way. □

Now, we will express $\frac{\|Q_1(q)-P_1(y)\|^2}{\|Q_1(q)-P_1(x)\|^2}$ as $\frac{a}{b}$ and then show that same ratio for other transformation functions can be represented as $\left(\frac{a+x}{b+x}\right)$ for some positive x. Invoking Lemma 1 then will yield the desired result. Note that, $\|Q_1(q) - P_1(y)\|^2 = 2\left(1 - \frac{q^\top y}{\beta\|q\|}\right)$. Therefore,

$$\frac{\|Q_1(q) - P_1(y)\|^2}{\|Q_1(q) - P_1(x)\|^2} = \frac{(\beta\|q\| - q^\top y)}{(\beta\|q\| - q^\top x)} \tag{9}$$

Now, $\|Q_2(q) - P_2(y)\|^2 = 2\left(1 - \frac{q^\top y}{\beta^2}\right)$. Therefore,

$$\frac{\|Q_2(q) - P_2(y)\|^2}{\|Q_2(q) - P_2(x)\|^2} = \frac{(\beta_1^2 - q^\top y)}{(\beta_1^2 - q^\top x)} = \frac{(\beta\|q\| - q^\top y) + (\beta_1^2 - \beta\|q\|)}{(\beta\|q\| - q^\top x) + (\beta_1^2 - \beta\|q\|)} \tag{10}$$

Combining Lemma 1, Eq. 9 and the fact that $\beta\|q\| \leq \beta_1 \cdot \beta_1 = \beta_1^2$, we get, $\frac{\|Q_1(q)-P_1(y)\|}{\|Q_1(q)-P_1(x)\|} \leq \frac{\|Q_2(q)-P_2(y)\|}{\|Q_2(q)-P_2(x)\|}$. Next,

$$\begin{aligned}\frac{\|Q_3(q) - P_3(y)\|^2}{\|Q_3(q) - P_3(x)\|^2} &= \frac{(\|q\|^2 + \beta^2 - 2q^\top y)}{(\|q\|^2 + \beta^2 - 2q^\top x)} \\ &= \frac{\left(\frac{\|q\|^2+\beta^2}{2} - q^\top y\right)}{\left(\frac{\|q\|^2+\beta^2}{2} - q^\top x\right)} \\ &= \frac{(\beta\|q\| - q^\top y) + (\frac{\|q\|^2+\beta^2}{2} - \beta\|q\|)}{(\beta\|q\| - q^\top x) + (\frac{\|q\|^2+\beta^2}{2} - \beta\|q\|)}\end{aligned} \tag{11}$$

Note that $(\|q\|^2+\beta^2)/2 \geq \beta\|q\|$. This follows from the fact that $(\|q\|-\beta)^2 \geq 0$ for any value of $\|q\|$ and β. Therefore, combining this with Eq. 9 and Lemma 1 we get, $\frac{\|Q_1(q)-P_1(y)\|}{\|Q_1(q)-P_1(x)\|} \leq \frac{\|Q_3(q)-P_3(y)\|}{\|Q_3(q)-P_3(x)\|}$. Now,

$$\begin{aligned}\|Q_4(q) - P_4(x)\|^2 &= \left(1 + \frac{m}{4}\right) - \frac{2q^\top x}{\alpha\|q\|} + \left(\frac{\|x\|}{\alpha}\right)^{2^{m+1}} \\ &= 2 - \frac{2q^\top x}{\alpha\|q\|} + \left(\frac{m}{4} - 1\right) + \left(\frac{\|x\|}{\alpha}\right)^{2^{m+1}}\end{aligned}$$

$$
\begin{aligned}
&= \frac{2(\alpha\|q\| - q^\top x) + \alpha\|q\|\left(\frac{m}{4} - 1\right) + \alpha\|q\| f(x,m)}{\alpha\|q\|} \\
&= \frac{2\left((\alpha\|q\| - q^\top x) + \frac{\alpha\|q\|}{2}\left(\frac{m}{4} - 1\right) + \frac{\alpha\|q\|}{2} f(x,m)\right)}{\alpha\|q\|} \\
&= \frac{2\left((\beta\|q\| - q^\top x) + \beta\|q\|\left(\frac{c}{2}(1+\frac{m}{4}) - 1\right)\right)}{\alpha\|q\|} \\
&\quad + \frac{2\left(\frac{\alpha\|q\|}{2} f(x,m)\right)}{\alpha\|q\|} \qquad (12)
\end{aligned}
$$

where, $f(x,m) = \left(\frac{\|x\|}{\alpha}\right)^{2^{m+1}}$. Therefore,

$$
\frac{\|Q_4(q) - P_4(y)\|^2}{\|Q_4(q) - P_4(x)\|^2} = \frac{(\beta\|q\| - q^\top y) + \beta\|q\|\left(\frac{c}{2}(1+\frac{m}{4}) - 1\right) + \frac{\alpha\|q\|}{2} f(y,m)}{(\beta\|q\| - q^\top x) + \beta\|q\|\left(\frac{c}{2}(1+\frac{m}{4}) - 1\right) + \frac{\alpha\|q\|}{2} f(x,m)}
$$

For m is large enough (m is at least $4\left(\frac{2}{c} - 1\right)$ so that the second term is non-negative) $f(\cdot, m)$ tends to zero doubly exponentially fast, and therefore, combining this with Eq. 9 and Lemma 1 we get, $\frac{\|Q_1(q)-P_1(y)\|}{\|Q_1(q)-P_1(x)\|} \leq \frac{\|Q_4(q)-P_4(y)\|}{\|Q_4(q)-P_4(x)\|}$. □

The above theorem implies that **T1** will achieve the lowest potential value (as defined in Eq. 8), and will have the highest success probability in finding the exact MIPS solution. Under mild conditions, we provide a total ordering of all the transformations with respect to the ratio $\frac{\|P_j(y)-Q_j(q)\|}{\|P_j(x)-Q_j(q)\|}$ in the following theorem:

Theorem 3 *Let $x, y \in \mathbb{R}^d$ be any two data points and let $q \in \mathbb{R}^d$ be a query point such that $q^\top y \geq q^\top x$. Suppose **T4** uses $c > 1$ and m satisfying $m \geq \frac{8}{c} \max\left\{\frac{\|q\|}{\beta}, \frac{\beta}{\|q\|}\right\}$. Then the following holds: $\frac{\|Q_1(q)-P_1(y)\|}{\|Q_1(q)-P_1(x)\|} \leq \frac{\|Q_3(q)-P_3(y)\|}{\|Q_3(q)-P_3(x)\|} \leq \frac{\|Q_2(q)-P_2(y)\|}{\|Q_2(q)-P_2(x)\|} \leq \frac{\|Q_4(q)-P_4(y)\|}{\|Q_4(q)-P_4(x)\|}$.*

Proof We have shown in Theorem 2 that the following four ratios $\frac{\|Q_1(q)-P_1(y)\|^2}{\|Q_1(q)-P_1(x)\|^2}$, $\frac{\|Q_2(q)-P_2(y)\|^2}{\|Q_2(q)-P_2(x)\|^2}$, $\frac{\|Q_3(q)-P_3(y)\|^2}{\|Q_3(q)-P_3(x)\|^2}$ and $\frac{\|Q_4(q)-P_4(y)\|^2}{\|Q_4(q)-P_4(x)\|^2}$ can be expressed as $\frac{a}{b}$, $\frac{a+\delta_1}{b+\delta_1}$, $\frac{a+\delta_2}{b+\delta_2}$ and $\frac{a+\delta_3(y)}{b+\delta_3(x)}$ respectively (see Eqs. 9, 10, 11, 12). Note that in $\delta_3(\cdot)$ differs in the numerator and denominator as it contains terms $f(y,m)$ and $f(x,m)$ respectively. However, since f decreases doubly exponentially fast in m, for large enough m both $f(x,m)$ and $f(y,m)$ approach zero and $\delta_3(y)$ becomes equal to $\delta_3(x)$. We will show $\delta_1 \geq \delta_2$ and for large enough m, $\delta_3 \geq \delta_1$, then invoking Lemma 1 will yield the desired result. Note that $\delta_1 = \beta_1^2 - \beta\|q\|$ and $\delta_2 = \frac{\|q\|^2+\beta^2}{2} - \beta\|q\|$. Since $\|q\| \leq \beta_1$ and $\beta \leq \beta_1$, we have $\delta_2 = \frac{\|q\|^2+\beta^2}{2} - \beta\|q\| \leq \frac{\beta_1^2+\beta_1^2}{2} - \beta\|q\| = \delta_1$. Next, note that,

$$
\begin{aligned}
\delta_3 &= \beta\|q\|\left(\frac{c}{2}\left(1+\frac{m}{4}\right) - 1\right) + \frac{\alpha\|q\|}{2} f(x,m) \\
&\geq \frac{\beta\|q\|c}{2}\left(1+\frac{m}{4}\right) - \beta\|q\| \overset{a}{\geq} \delta_1
\end{aligned}
$$

Inequality a holds as long as, $\frac{\beta\|q\|c}{2}\left(1+\frac{m}{4}\right) \geq \beta_1^2 \implies m \geq 4\left(\frac{2}{c}\left(\frac{\beta_1^2}{\beta\|q}\right) - 1\right)$ or for any $m \geq \frac{8\beta_1^2}{c\beta\|q\|}$. Note that when $\|q\| \geq \beta$, $\beta_1 = \max\{\beta, \|q\|\} = \|q\|$, and therefore, $\frac{\beta_1^2}{\beta\|q\|} = \frac{\|q\|}{\beta}$.

Alternatively, when $\beta \geq \|q\|, \beta_1 = \max\{\beta, \|q\|\} = \beta$, and therefore, $\frac{\beta_1^2}{\beta\|q\|} = \frac{\beta}{\|q\|}$. Therefore, as long as m is large enough (at least $\frac{8}{c}\max\left\{\frac{\|q\|}{\beta}, \frac{\beta}{\|q\|}\right\}$) the statement of the theorem holds. □

This implies a total ordering among the four transformations in increasing order of the resulting potential values:

Corollary 2 *Given a set $S \subset \mathbb{R}^d$ of data points and a query $q \in \mathbb{R}^d$, let **Ti** $\preceq$ **Tj** indicate that applying **Ti** $= (P_i, Q_i)$ to (S, q) yields a lower potential value as compared to applying **Tj** to the same. Suppose **T4** uses c and m satisfying $m \geq \frac{8}{c}\max\left\{\frac{\|q\|}{\beta}, \frac{\beta}{\|q\|}\right\}$. Then **T1** $\preceq$ **T3** $\preceq$ **T2** $\preceq$ **T4** holds.*

Proof The corollary follows imemdiately from Theorem 3, definition of transformations **T1** , **T2** , **T3** , **T4** and Eq. 8. □

This result suggests that, for RPTs, we can expect the MIPS accuracy of each of the four transformations to follow the same ordering. Note that, **T1** $\preceq$ **T3** $\preceq$ **T2** always holds. If the query norm is either too large or too small relative to the maximum point norm in S, m needs to be large enough to maintain relative position of **T4** in the above ordering. This result theoretically suggests that **T1** is the best (among the existing) transformation for solving MIPS with RPTs.

4 RPT with reduced space complexity

In spite of having nice theoretical guarantee in terms of finding exact nearest neighbors, RPTs are not memory efficient. Note that each internal node of RPT needs to store a pair consisting of a d dimensional random projection vector and a scalar random split point. Space required to store these projections directions is $\sum_{i=0}^{\ln(n/n_0)} 2^i \cdot d = O(dn)$ for constant n_0 for a single RPT. Moreover, if L independent such RPTs are used, total memory requirement for storing all the projection directions is $O(Ldn)$. This leads to total space complexity of L RPTs to be $O(nd + Lnd + Ln)$, where the first term is to store the dataset (nd-dimensioanl data points), the second term is to store the random projection directions and the third term is to store random split points. In comparison, space complexity of LSH, which is a random projection based method, is $O(nd + n^\rho d \log n + n^{1+\rho})$. The first term above is to store n d-dimensional data points. The second terms corresponds to space required to store random projection directions for computing the random hash functions. For a single hash table the random hash function has the form $h : \mathbb{R}^d \to \{0, 1\}^k$ and one needs to store k d-dimensional random projection vectors to store this. To ensure constant failure probability in solving approximate nearest neighbor search, it is recommended to use $k = \log n$ and use $L = n^\rho$ hash tables, where the value of ρ is $1/c$ if LSH needs to return a c-approximate nearest neighbor solution[9] (Indyk and Motwani 1998; Datar et al. 2004; Andoni and Indyk 2008). Finally the third term $n^{1+\rho}$ corresponds to space required to store n^ρ hash tables each of which takes $O(n)$ space. In practice however, practitioners use different values of k and L and the space complexity of LSH reduces to $O(nd + Lkd + Ln)$.

As can be seen from the above discussion, the dominating term appearing in the space complexity expression of RPTs is the term $O(Lnd)$ (compared to the corresponding $O(Lkd)$ term for LSH), i.e. the space required to store the random projection directions. In the following, we discuss two strategies that reduces this term significantly.

[9] For a query $q \in \mathbb{R}^d$, let p^* be its exact nearest neighbor in $S \subset \mathbb{R}^d$, i.e., $p^* = \text{argmin}_{\text{p}\in\text{S}}\|\text{q} - \text{p}\|$. An c-approximate nearest neighbor of q is any $p \in S$, that satisfies $\|q - p\| \leq (1 + c)\|q - p^*\|$.

4.1 Space complexity reduction strategy 1: RPTS

Our first strategy is to reduce the space complexity for individual RPTs. Here, instead of storing separate random projection direction at each internal node, we keep a single common random projection direction for all internal nodes located at any fixed tree depth (level). We call this space reduction strategy as RPTS and provide its pseudo code in Algorithm 3.

Algorithm 3 Function `ChooseRule` for RPTS

Input : data S, depth of the current node from root d_l
Output : rule
function `ChooseRule`(S, d_l)

1: **if** no projection direction have been chosen for this level d_l yet **then**
2: Pick U uniformly at random from the unit sphere by choosing each of its coordinate independently at random from a standard Normal distribution
3: Pick β uniformly at random from $[1/4, 3/4]$
4: **else**
5: Use same U and β already chosen for this level.
6: **end if**
7: Let v be the β-fractile point on the projection of S onto U
8: Rule$(x) = (x \cdot U \leq v)$
9: **return** (Rule)

Since RPTS tree depth is at most $O(\log n)$, each RPTS requires $O(d \log n)$ space to store all the projection directions for that tree. Consequently, if there are L such trees, total space requirement is $O(Ld \log n)$. Performance guarantee of RPTS is immediate as projection directions at different levels are independent to each other and we can simply use an union bound of the failure probabilities over the path that conveys query q from root to leaf node of an RPTS and is given in the following lemma.

Lemma 2 *Given any query point q, probability that a RPTS fails in finding true nearest neighbor of q is same as that of a RPT.*

Proof As projection directions at intermediate nodes at different levels are independent of each other along any path from root to any leaf node, using union bound, the failure probability analysis is essentially the same as in Dasgupta and Sinha (2013, 2015). □

Note that a similar strategy was also introduced in Hyvönen et al. (2016).

4.2 Space complexity reduction strategy 2: RPTB

While RPTS has reduced space complexity as compared to RPT, space required to store the projection directions still increases linearly with L, the number of trees. We now present a second strategy, for which, memory required to store all the projection directions of L trees is independent of L. To achieve this, we keep a fixed number of independent projection directions, chosen uniformly at random from the unit sphere in a bucket. Projection directions from this bucket is used to construct all L randomized partition trees. Using this strategy, while constructing a randomized partition tree, we still use a single projection direction for all nodes located at a fixed level as in RPTS, but the difference now is that projection directions at each level are chosen uniformly at random without replacement from the bucket. Since all projection directions stored in the bucket are independent to each other, this strategy ensures

that projection directions at different level of the tree thus constructed are still independent to each other. We call this space reduction strategy as RPTB and provide its pseudo code in Algorithm 4. Number of projection directions stored in a bucket is typically a constant times $\log n$, as a consequence, space required for storing all the projections directions of L RPTBs is $O(d \log n)$ and is independent of L. As before, it is easy to see that,

Lemma 3 *Given any query point q, probability that a RPTB fails in finding true nearest neighbor of q is same as that of a RPT.*

The reason for keeping the bucket size to be a constant times $\log n$ is appearnat from the following lemma which states that with high probability no two RPTBs have same sequence of projection directions at every level from root to leaf node.

Lemma 4 *For any $c \geq 3$, suppose the bucket in RPTB contains $c \cdot \log n$ distinct projection directions uniformly chosen at random from a unit sphere, where n is the number of data points in S. If the number of RPTBs is limited to at most $\sqrt{n}$, then with probability at least $\left(1 - \frac{1}{2\sqrt{n}}\right)$, no two RPTBs will have same sequence of projection directions at each level along the path from root to leaf node.*

Proof An RPTB is constructed by choosing a projection direction for each level of the RPT uniformly at random without replacement from the bucket. Let m be the depth of such an RPTB. Consider any two instantiations of RPTBs, namely, τ_i and τ_j. Let $\mathcal{A}_{ij}$ be the event that τ_i and τ_j has same sequence of projection directions at every level along the path from root to leaf node. Then $\Pr(\mathcal{A}_{ij}) = \frac{1}{N} \cdot \frac{1}{N-1} \cdots \frac{1}{N-(m-1)} \leq \frac{1}{(N-m+1)^m} \leq \frac{1}{(N-m)^m}$. Note that because of the choice random split, depth m of any RPTB can be at most $\log_{4/3} n = (\log_{4/3} 2) \cdot \log n \leq (5/2) \cdot \log n$, and at least $\log_4 n = \frac{1}{2} \log n$. Suppose the bucket contains $N = cm = c \cdot \log n$ projection directions for some $c \geq 3$. If we have L RPTBs, then the probability that any pair of RPTBs have same sequence of projection directions at every level along the path from root to leaf node is:

$$
\begin{aligned}
\Pr\left(\exists(i, j) \text{ such that } \mathcal{A}_{ij} \text{ happens}\right) &\leq \binom{L}{2} \Pr(\mathcal{A}_{ij}) \\
&\leq \frac{L^2}{2} \left(\frac{1}{(N-m)^m} \right) \\
&\leq \frac{L^2}{2} \left(\frac{1}{((c-1) \cdot m)^{\frac{1}{2} \log n}} \right) \\
&\leq \frac{L^2}{2} \left(\frac{1}{\left(\frac{(c-1)}{2} \log n\right)^{\frac{1}{2} \log n}} \right) \\
&\leq \frac{L^2}{2} \left(\frac{1}{\left(\frac{(c-1)}{2}\right)^{\frac{1}{2} \log n} (\log n)^{\frac{1}{2} \log n}} \right) \\
&\overset{a}{\leq} \frac{L^2}{2} \left(\frac{1}{(\log n)^{\frac{1}{2} \log n}} \right) \\
&\overset{b}{\leq} \frac{L^2}{2} \left(\frac{1}{n^{\frac{1}{2} \log \log n}} \right)
\end{aligned}
$$

Algorithm 4 Function `ChooseRule` for RPTB

Input : data S, depth of the current node from root d_l, constant c
Output : rule
function `ChooseRule`(S, d_l)

1: **if** no projection direction have been chosen for this level d_l yet **then**
2: Pick U uniformly at random without replacement from a bucket containing $c \cdot \log n$ projection directions.
3: Pick β uniformly at random from $[1/4, 3/4]$
4: **else**
5: Use same U and β already chosen for this level.
6: **end if**
7: Let v be the β-fractile point on the projection of S onto U
8: Rule$(x) = (x \cdot U \leq v)$
9: **return** (Rule)

$$\overset{c}{\leq} \frac{L^2}{2}\left(\frac{1}{n^{\frac{3}{2}}}\right) \leq \frac{1}{2\sqrt{n}}\left(\frac{L^2}{n}\right) \overset{d}{\leq} \frac{1}{2\sqrt{n}}$$

Inequality a is due to choice of c while inequality b follows from the following observation. Suppose $\log n = 2^\beta$ for some $\beta > 0$. This implies $\beta = \log\log n$. Clearly, $(\log n)^{\frac{1}{2}\log n} = 2^{\frac{\beta}{2}\log n} = (2^{\log n})^{\frac{\beta}{2}} = n^{\frac{\beta}{2}} = n^{\frac{1}{2}\log\log n}$. Inequality c holds as long as $n \geq 256$ and inequality d follows from the restriction on L. □

Note that, extreme tree depths, i.e., $(5/2) \cdot \log n$ or $\frac{1}{2}\log n$ happens very rarely. For all our experiments tree depth was very close to $\log n$ and a bucket containing $2 \cdot \log n$ distinct projection directions preformed very well.

5 Empirical evaluations

In this section, we present empirical results in the form of three experiments. The first experiment validates the ordering of the transformations **T1** –**T4** presented in Corollary 2. The second experiment demonstrates that the ordering of the potential function values agrees with the actual performance of RPTs for MIPS—that is, the transformation with the lowest potential function value provides the best accuracy–efficiency trade-off for MIPS. The final experiment compares our proposed MIPS solution, including space complexity reduction strategies, with the state-of-the-art approximate MIPS solution with LSH. For all our empirical evaluations we consider eight real world datasets varied dimensionality and size as shown in Table 1. The SIFT dataset contains SIFT image descriptors introduced in Jégou et al. (2011). The original dataset contains 1 million image descriptors. We used 60,000 image descriptors from this dataset for our experiments. The AERIAL dataset contains texture information of large areal photographs (Manjunath and Ma 1996). The COREL dataset is available at the UCI repository (Bache and Lichman 2013). MNIST and USPS are datasets of handwritten digits. REUTERS is a common text dataset used in machine learning and is available in Matlab format in Cai (2009). After removing the missing data, the dataset contained 8293 documents. Netflix and Movielens are datasets usually used in recommender systems. We used exactly the same pre-processing step described in Neyshabur and Srebro (2015) for these two datasets. For SIFT, Aerial, Corel and MNIST dataset we randomly chose 50,000 points as database points that were used to construct appropriate data structure (RPT or hash tables) and 10,000 points as queries. For Movielense and Netflix dataset (Neyshabur and Srebro 2015), number of database points were fixed to 10,677 and 17,770 respectively. We

Table 1 Dataset description

Dataset	# Points in S	# Queries	# Dimensions d
SIFT	50,000	10,000	128
Aerial	50,000	10,000	60
Corel	50,000	10,000	89
MNIST	50,000	10,000	784
Reuters	6000	2293	18,933
Netflix	17,770	10,000	300
Movielens	10,677	10,000	150
USPS	7000	2298	256

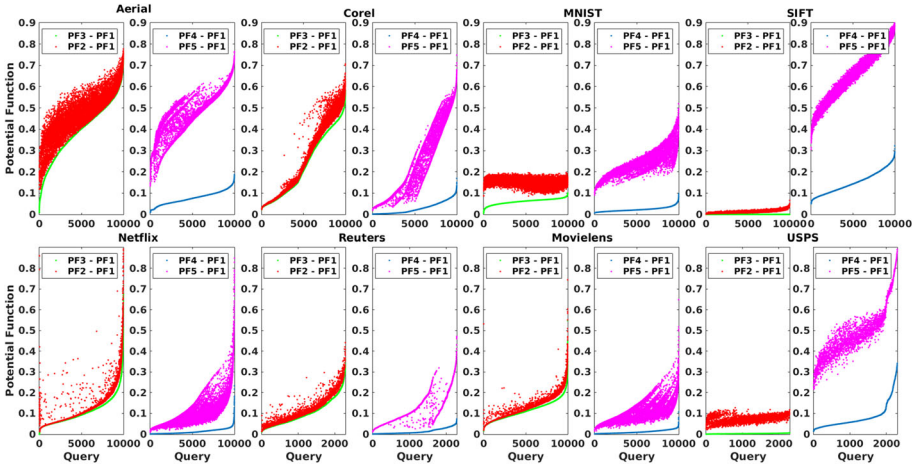

Fig. 1 Potential function differences (y-axis) versus query index (x-axis) plot: (*please view in colour*): the green line indicates the sorted differences PF3 − PF1, while the red dots indicate the differences PF2 − PF1 against the sorted index of PF3 − PF1. The blue line indicates the sorted differences PF4 − PF1, while the purple dots indicate the differences PF5 − PF1 against the sorted index of PF4 − PF1 (Color figure online)

chose the first 10,000 points as query points in each case. For Reuters dataset, we randomly chose 6000 data points as database points and the remaining 2293 points as query points. The USPS dataset contains only 9298 points and we randomly chose 7000 points to construct various data structures and the rest as query points.

5.1 Experiment I: potential function evaluation

In this experiment, we compute the potential function values for each query after applying each of the four transformations. For every dataset, we use PFi, $i = 1, 2$ and 3 to denote the vector of potential function values for all queries upon applying transformation **Ti**. Since transformation **T4** depends on the choice of m, we choose two values—at one extreme, we choose a small $m = 3$ (as suggested in Shrivasatava and Li 2015), and at the other extreme, we choose a large $m = 100$. The corresponding vectors of potential function values are denoted by PF4 and PF5 respectively. We visualize the relative ordering of the transformations for each dataset in Fig. 1. The left panel for each dataset compares **T1** to **T2** and **T3** , while the right panel compares **T1** to **T4** with two different values of m.

To create the visualization in the left panel, we first compute the differences $\texttt{PF3} - \texttt{PF1}$, sort it in the increasing order and generate the green line. We generate the red dots by plotting the differences $\texttt{PF2} - \texttt{PF1}$ against the sorted index of $\texttt{PF3} - \texttt{PF1}$. The green line is always positive for all datasets, indicating that **T1** produces lower potential function values than **T3** . The red dots values are always on or above the green line, indicating that **T3** produces lower potential function values than **T2** (and **T1** produces lower values than both). This demonstrates that $\textbf{T1} \preceq \textbf{T3} \preceq \textbf{T2}$ holds in practice. The visualization in the right panel of each sub-figure in Fig. 1 is generated in a similar manner. We use the sorted differences $\texttt{PF4} - \texttt{PF1}$ to generate the blue line. We generate the purple dots by plotting the differences $\texttt{PF5} - \texttt{PF1}$ against the sorted index of $\texttt{PF4} - \texttt{PF1}$. The results indicate that $\textbf{T1} \preceq \textbf{T4}$ for different values of m.

We do not present a direct comparison of **T2** and **T3** with **T4** because their relative ordering depends on the parameters chosen for **T4** and the dataset characteristics (Corollary 2). However, we would like to note that **T4** ensures that the exact NNS solution in the transformed space is the exact MIPS solution in the original space only as $m \to \infty$ (Theorem 1). Since RPTs provide guarantees on the exact NNS solution (and hence, the exact MIPS solution), **T4** only makes sense for RPTs with large m. However, our empirical results indicate that, as m grows, the potential function value grows, making NNS (and subsequently MIPS) harder, and **T4** undesirable for MIPS with RPTs. Hence, we will not consider **T4** any further in our evaluations.[10]

Note that, Fig. 1 provides only a qualitative view of the relative ordering of various transformation by plotting the potential function values. To present a quantitative view, we first plot histograms (we used 50 bins to plot these histograms) of PF1, PF2 and PF3 for all eight datasets in Fig. 2. To draw these histograms, first the range of potential function, which is [0, 1], is discretized in to 50 disjoint bins, where each bin corresponds to a range of potential function values and then number of queries whose potential function values lie in that range are plotted. Shape of these histograms agree with the ordering of the transformations **T1** , **T2** and **T3** in the sense that histograms of PF1 (that corresponds to **T1**) is concentrated more towards the left as compared to the histograms of PF2 and PF3 for almost all datasets. This indicates that transformation **T1** results in more queries to have lower potential function values as compared to transformation **T2** and **T3** . To quantify this, we convert the histograms into a discrete probability distribution in a straight forward manner by dividing number of query points in each bin by the total number of query points. Let us call these probability distributions PD1, PD2 and PD3, each of which is a vector of size 50. To quantify the different between these distributions, we compute the well known Hellinger distance between these probability distributions using Eq. 13 for each dataset and present in Table 2.

$$H(\texttt{PDi}, \texttt{PDj}) = \frac{1}{\sqrt{2}} \sqrt{\sum_{k=1}^{K} (\sqrt{\texttt{PDi}_k} - \sqrt{\texttt{PDj}_k})^2} \tag{13}$$

Hellinger distance $H(\texttt{PDi}, \texttt{PDj})$ between any two discrete probability distribution PDi and PDj, as given in Eq. 13, is symmetric and is always bounded between 0 and 1. Hellinger distribution 0 implies that two probability distributions are exactly same while Hellinger

[10] **T4** for small m will direct RPTs to find the exact NN in the transformed space which could be significantly different from the exact MIPS solution in the original space. Since the rest of the transformations will direct RPTs to find the exact MIPS solution, the relative comparison for MIPS with **T4** can be unstable and unintuitive. **T4** with large m will direct RPTs to find the exact MIPS solution and produce more intuitive results. However, **T4** with large m is undesirable because the transformed space dimensionality as well as the potential function value will have increased significantly (as observed in the right panels of Fig. 1).

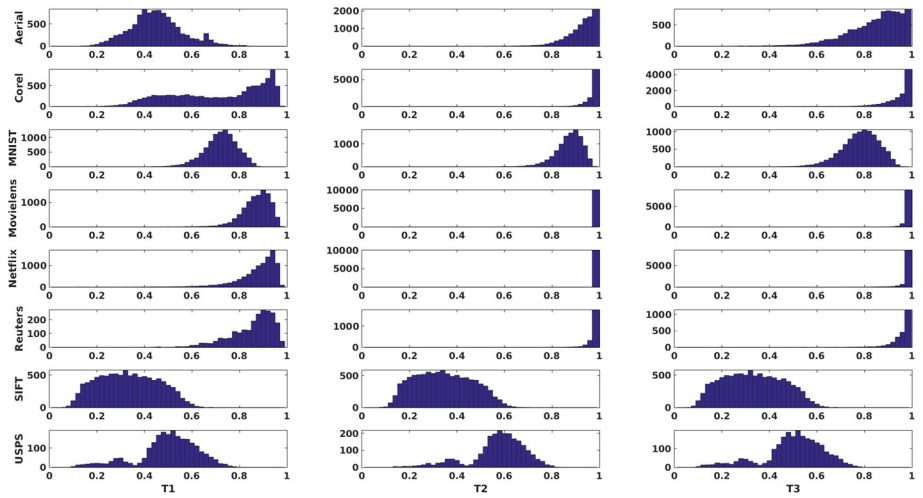

Fig. 2 Histgoram for the potential function with different transformations on eight different datasets. First, second and third columns in the above figure represents histogram of potential function values obtained by applying transformation **T1** , **T2** and **T3** respectively

Table 2 Hellinger distance between PD1, PD2 and PD3

Dataset	$H(\mathrm{PD1}, \mathrm{PD2})$	$H(\mathrm{PD1}, \mathrm{PD3})$	$H(\mathrm{PD2}, \mathrm{PD3})$
Aerial	0.96	0.84	0.31
Corel	0.81	0.66	0.22
MNIST	0.74	0.34	0.45
Movielens	0.98	0.89	0.23
Netflix	0.95	0.75	0.28
Reuters	0.35	0.27	0.11
SIFT	0.08	0.003	0.08
USPS	0.15	0.01	0.15

distance 1 implies they are very very different. In general, higher that Hellinger distance between two probability distributions, more different the two distributions are. Note that a popular measure to compare two probability distribution is Kullback-Leibler divergence or KL-divergence for short. Unlike Hellinger distance, KL-divergence is not symmetric (thus not a distance metric) and is unbounded and therefore we choose Hellinger distance to represent the difference between these probability distributions. Please note that, Helinger distance and KL divergence are related as follows: for any two distributions P and Q, $H(P, Q) \leq \left(\frac{1}{2} D_{KL}(P||Q)\right)^{\frac{1}{4}}$. This follows from the fact that Hellinger distance $H(P, Q)$ and total variation distance $\delta(P, Q)$ are related by the following inequality $H^2(P, Q) \leq \delta(P, Q) \leq \sqrt{2} H(P, Q)$ and Pinsker's inequality relates total variation distance and KL divergence via $\delta(P, Q) \leq \sqrt{\frac{1}{2} D_{KL}(P||Q)}$.

As can be seen from Table 2, $H(\mathrm{PD1}, \mathrm{PD2}) \geq H(\mathrm{PD1}, \mathrm{PD3})$ for all eight datasets, indicating that PD1 is more similar to PD3 than to PD2. This agrees with visual inspection of histogram plots in Fig. 2. In particular, the value of $H(\mathrm{PD1}, \mathrm{PD3})$ sheds some light on the relative position of the green line ($\mathrm{PF3} - \mathrm{PF1}$) in Fig. 1. For example, in case of SIFT and

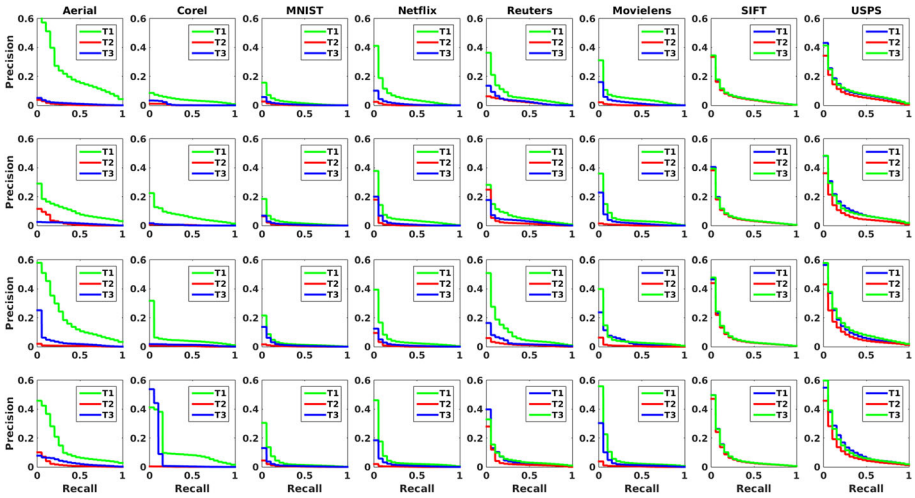

Fig. 3 Precision recall curves for MIPS with RPTs: (*please view in colour*): The first, second, third and fourth row correspond to $n_0 = 10, 20, 40$ and 50 respectively (Color figure online)

and USPS dataset $H(\texttt{PD1}, \texttt{PD3}) \approx 0$, and this explains why the green line representing $(\texttt{PF3} - \texttt{PF1})$ lies very close to zero in Fig. 1, whereas in case of, say Aerial dataset value of $H(\texttt{PD1}, \texttt{PD3})$ is very high and explains why the green line representing $(\texttt{PF3} - \texttt{PF1})$ lies far away from the x-axis in Fig. 1.

5.2 Experiment II: precision-recall curve

In this experiment, we generate precision recall (p–r) curves for finding the 20 highest inner products for each query using RPTs and present the relative performance when using one of **T1** - **T3** as the MIPS-to-NNS reduction. We generate the p–r curves using the technique presented by Shrivasatava and Li (2015), followed by a TREC interpolation (Trec interpolation 2016). We consider four different values of n_0 and present the results in Fig. 3.

The results indicate that the p–r curves for **T1** usually dominate the other p–r curves by a significant margin for most datasets and values of n_0. Moreover, the **T3** performance dominates the **T2** performance. This demonstrates that the MIPS performance of RPTs for the different transformations agrees with the ordering of the potential function values presented in Corollary 2. However, if potential function values obtained from all three transformations (**T1** , **T2** , **T3**) are very similar, as is the case in case of SIFT dataset (see Fig. 1, and corresponding histogram and Hellinger distances in Fig. 2 and Table 2 respectively) then p–r curves for all three transformations are also very similar and there is no clear advantage of one transformation over another. We would also like to point out that the RPT guarantees are probabilistic and there are times when the ordering is violated. For example, **T3** achieves higher precision than **T1** at low recall values for Reuters and Corel data set with $n_0 = 50$ or **T2** performing slightly better than **T3** for Aerial with $n_0 = 50$ and Reuters with $n_0 = 20$ at low recall values.

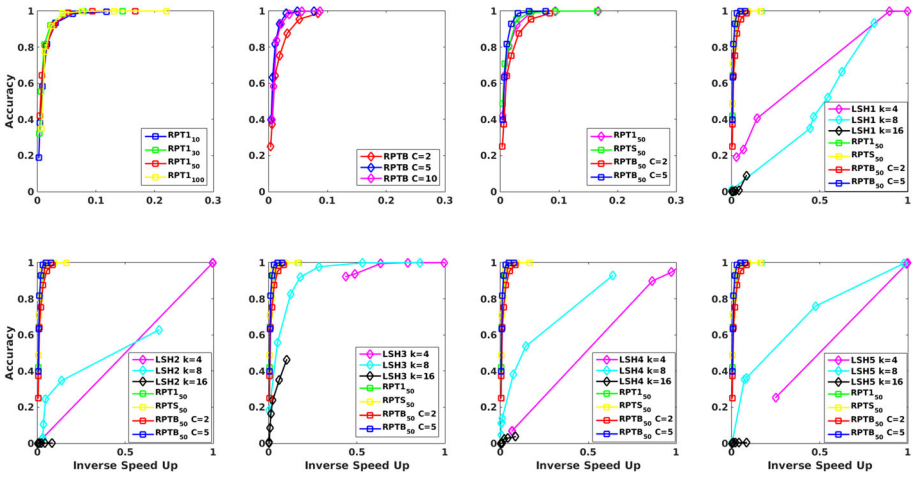

Fig. 4 Accuracy versus inverse Speed Up for Aerial dataset. First plot shows the sensitivity of RPT toward n_0 value. Second plot investigate the sensitivity of RPTB toward parameter C. Third plot compares RPT, RPTS and RPTB. Finally, plot 4–8 compares all RPT versions which has been mentioned in this paper with LSH with different transformations. Plots 1–4 are from left to right, top row and plots 5–8 are from left to right bottom row

5.3 Experiment III: accuracy versus inverse speed-up

In Sects. 5.1 and 5.2 we have demonstrated that **T1** is the best transformation, compared to **T2** and **T3** , to be used with RPT for solving MIPS. In this experiment, we compare the accuracy–efficiency trade-off of RPTs and two of its space complexity reduction strategies, namely RPTS and RPTB, with **T1** for MIPS to existing baselines. In particular, we present our exhaustive experimental results for different choices of n_0 in case of RPT and RPTS, different bucket sizes for RPTB and compare their accuracy–efficiency trade-off with MIPS solution using LSH using all possible transformations (Figs. 4–11).

We consider the task of finding the 10 highest inner products for each query. We choose the hash code lengths of 4, 8, 16 for all LSH variants as they reportedly produce the best performance. We choose leaf sizes n_0 of 10, 30, 50, 100 for RPTs. The accuracy of a method is defined as the recall of the 10 highest inner products averaged over all the queries. The efficiency of a method is defined by the *inverse speed up* over linear search and computed as

$$\text{Inverse speed up} = (\text{Total \# inner products})/|S|$$

with any set S for a MIPS query.[11] The number of RPTs and LSH hash tables are chosen from the set $\{4, 8, 16, 32, 64, 128, 256\}$ to obtain 7 (accuracy, inverse speed up) pairs for

[11] Precision-recall curves make sense when the underlying method/parameters are the same, and imply the same amount of computation to retrieve the candidate set. We consider the total number of inner products μ required for a MIPS query to provide a fair comparison between different kinds of methods/parameters. For each RPT, this corresponds to the number of inner products needed to route a query to a leaf and process the points in that leaf. For each LSH hash table, this corresponds to the number of inner products needed to generate a hash code of a length k and process the points in that hash bucket. Let R be the size of the candidate set retrieved by each method. For RPTs with L trees, $\mu = R + L \log |S|$. For LSH with H hash tables, $\mu = R + Hk$. Note that, in case of RPTB, μ can be at most $R + c \log |S|$, where c specifies the bucket size and can be considerable less compared to L.

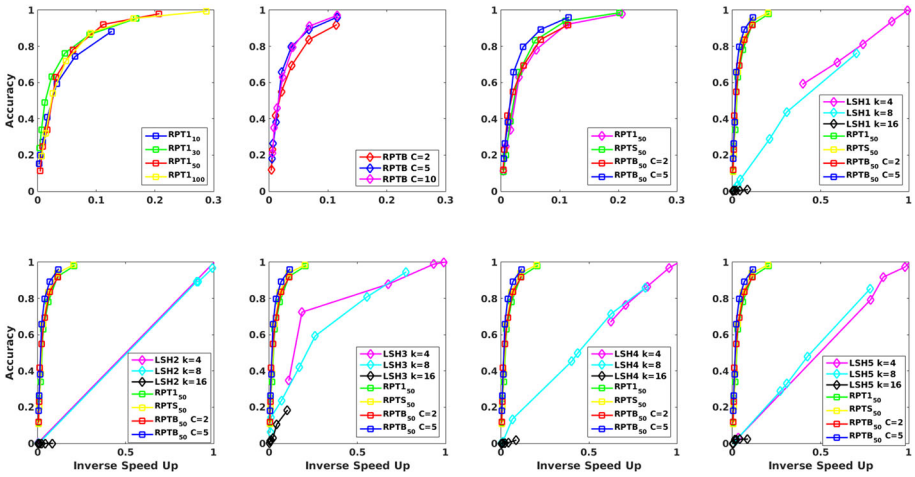

Fig. 5 Accuracy versus inverse Speed Up for Corel dataset. First plot shows the sensitivity of RPT toward n_0 value. Second plot investigate the sensitivity of RPTB toward parameter C. Third plot compares RPT, RPTS and RPTB. Finally, plot 4–8 compares all RPT versions which has been mentioned in this paper with LSH with different transformations. Plots 1–4 are from left to right in top row and plots 5–8 are from left to right in bottom row

each method-parameter combination. These are used to generate the accuracy–efficiency trade-off curves in Figs. 4–11. Moving from left to right on each curve implies increased computation (hence lower efficiency). On each curve, the first marker (from the left) in Figs. 4–11 corresponds to 4 RP-trees (or hash tables), the second marker corresponds to 8 RP-trees (or hash tables) and so on. We continue this way to generate 7 markers for each curve. This way, Figs. 4–11 allows us to compactly represent the interaction between the number of trees (each marker), number of retrieved points by each method (represented by inverse speedup on the x-axis) and the accuracy of the approximate MIPS obtained by LSH and RP-trees (on the y-axis).

In each Figs. 4–11, the left most plot on top row shows the accuracy–efficiency trade-off of RPT for different choices of n_0 values. Observe that unless n_0 is very small, say when $n_0 = 10$, performance of RPT with different n_0 values are very similar. Therefore, in subsequent comparisons we use the choice of $n_0 = 50$. The second plot from the left on top row shows accuracy–efficiency trade-off of RPTB for three different bucket sizes, namely $2 \log n$, $5 \log n$ and $10 \log n$ ($C = 2, 5$ and 10 respectively). Observe that for most datasets their performances are very similar, occasionally $C = 5$ or $C = 10$ performing little better compared to $C = 2$, which provides less randomness. Since the purpose of RPTB is to reduce space complexity and lower the C the better, for subsequent comparisons we use $C = 2$ and $C = 5$ only. The third plot from the left on top row shows the accuracy–efficiency trade-off of RPTS with $n_0 = 50$ along with RPT with $n_0 = 50$ and RPTB with $C = 2$ and $C = 5$. Again, performances of RPT and RPTS are very similar and are not too different from that of the RPTBs except in case of Netflix and Reuters dataset (Figs. 8, 9) where RPTB seems to perform better compared to RPT and RPTS. Note that for all three plots from the left on top row, the x axis ranges from 0 to 0.3 (except Movielens, Netflix, Reuters and USPS dataset i.e., Figs. 7, 8, 9, 11) to show the difference between the accuracy–efficiency trde-offs of RPT, RPTS and RPTB more closely. In subsequent plots, the x-axis ranges from 0 to 1 and thus their accuracy–efficiency trade-off curves look even

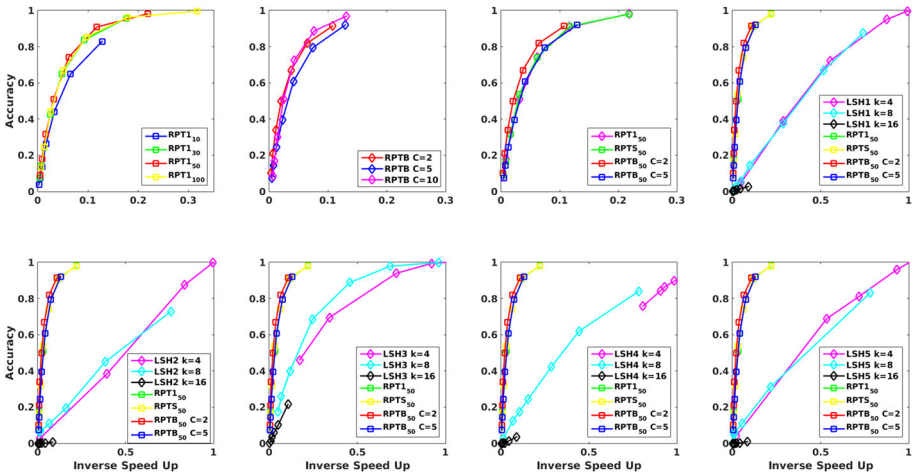

Fig. 6 Accuracy versus inverse Speed Up for MNIST dataset. First plot shows the sensitivity of RPT toward n_0 value. Second plot investigate the sensitivity of RPTB toward parameter C. Third plot compares RPT, RPTS and RPTB. Finally, plot 4–8 compares all RPT versions which has been mentioned in this paper with LSH with different transformations. Plots 1–4 are from left to right in top row and plots 5–8 are from left to right in bottom row

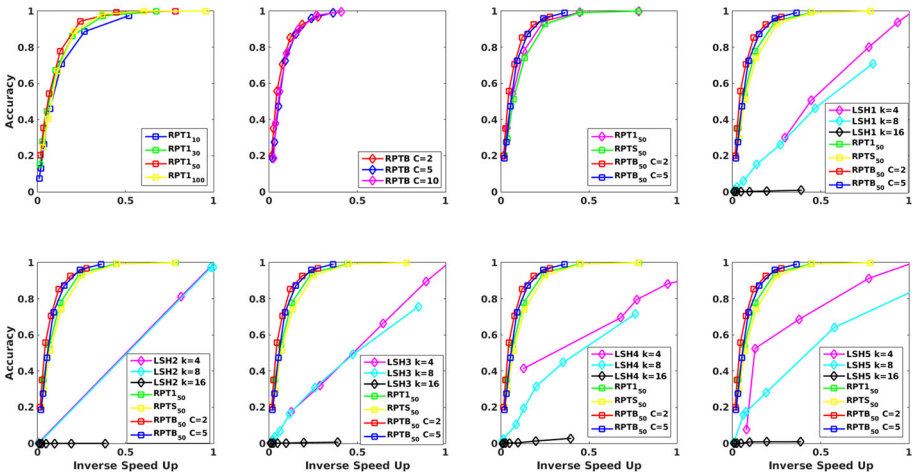

Fig. 7 Accuracy versus inverse Speed Up for Movielens dataset. First plot shows the sensitivity of RPT toward n_0 value. Second plot investigate the sensitivity of RPTB toward parameter C. Third plot compares RPT, RPTS and RPTB. Finally, plot 4–8 compares all RPT versions which has been mentioned in this paper with LSH with different transformations. Plots 1–4 are from left to right in top row and plots 5–8 are from left to right in bottom row

more similar. Also note that in case of Movielens, Netflix and Reuters datasets, histograms of potential function values (after applying transformation **T1**) are concentrated heavily towards right as compared to the remaining 5 datasets (see Fig. 2). This indicates that in the transformed space, the corresponding NNS problem for these three datasets are harder compared to the rest of the datasets and yield higher inverse speed as compared to other datasets to achieve the same level of accuracy. Finally, the remaining five plots in each

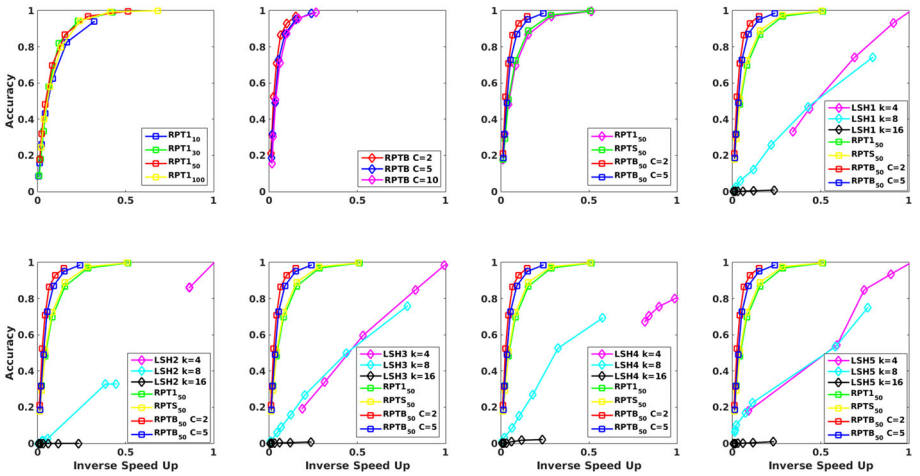

Fig. 8 Accuracy versus inverse Speed Up for Netflix dataset. First plot shows the sensitivity of RPT toward n_0 value. Second plot investigate the sensitivity of RPTB toward parameter C. Third plot compares RPT, RPTS and RPTB. Finally, plot 4–8 compares all RPT versions which has been mentioned in this paper with LSH with different transformations. Plots 1–4 are from left to right in top row and plots 5–8 are from left to right in bottom row

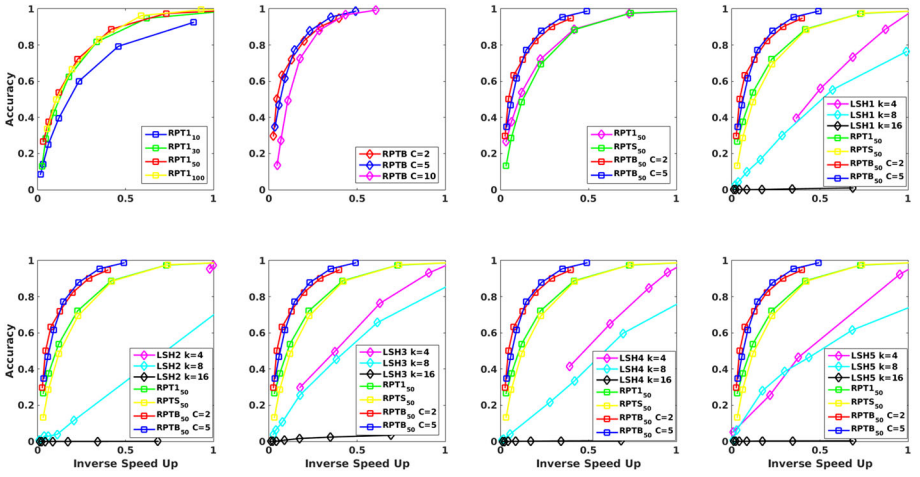

Fig. 9 Accuracy versus inverse Speed Up for Reuters dataset. First plot shows the sensitivity of RPT toward n_0 value. Second plot investigate the sensitivity of RPTB toward parameter C. Third plot compares RPT, RPTS and RPTB. Finally, plot 4–8 compares all RPT versions which has been mentioned in this paper with LSH with different transformations. Plots 1–4 are from left to right in top row and plots 5–8 are from left to right in bottom row

Figs. 4–11 show the accuracy–efficient trade-offs of all possible baseline methods, namely LSH with all transformations(with three different hashcode lengths), for solving MIPS. In these plots, LSH1, LSH2 and LSH3 corresponds to LSH after applying transformations **T1** , **T2** and **T3** respectively, while LSH4 and LSH5 corresponds to LSH after applying transformation **T4** with $m = 3$ and $m = 100$ respectively.

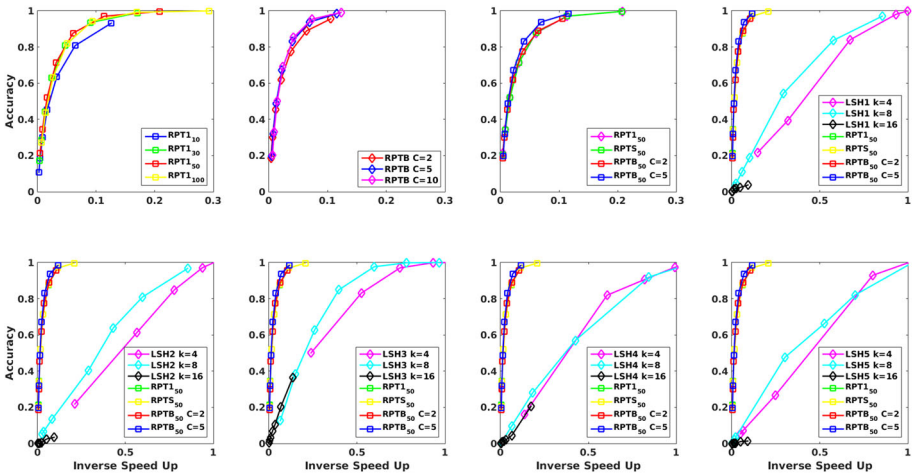

Fig. 10 Accuracy versus inverse Speed Up for SIFT dataset. First plot shows the sensitivity of RPT toward n_0 value. Second plot investigate the sensitivity of RPTB toward parameter C. Third plot compares RPT, RPTS and RPTB. Finally, plot 4–8 compares all RPT versions which has been mentioned in this paper with LSH with different transformations. Plots 1–4 are from left to right in top row and plots 5–8 are from left to right in bottom row

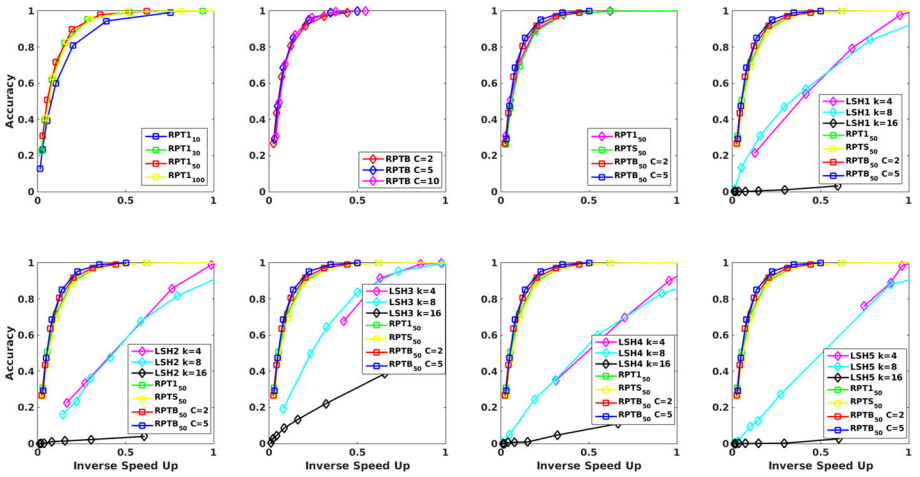

Fig. 11 Accuracy versus inverse Speed Up for USPS dataset. First plot shows the sensitivity of RPT toward n_0 value. Second plot investigate the sensitivity of RPTB toward parameter C. Third plot compares RPT, RPTS and RPTB. Finally, plot 4–8 compares all RPT versions which has been mentioned in this paper with LSH with different transformations. Plots 1–4 are from left to right in top row and plots 5–8 are from left to right in bottom row

As can be seen from Figs. 4–11, as *inverse speedup* $\rightarrow$ 1 accuracy gets close to 100%. When this happens for any method, the corresponding method is no better than a simple linear scan over the entire dataset. It is easy to observe from Figs. 4–11 that RPT/RPTS/RPTB with **T1** requires much less computations (small inverse speed up value) to reach higher accuracy compared to LSH (with any transformation). The results indicate that RPT and two of its space complexity reduction strategies (RPTS and TPTB) achieve a particular level of

accuracy much more efficiently (at smaller values of inverse speed up) than LSH no matter which transformation we use for LSH. Moreover, the performance of RPTs does not appear to be significantly affected by the choice of n_0 (as long as $n_0 \ll |S|$) and RPT/RPTS/RPTB provide easy access to the full accuracy–efficiency trade-off spectrum. On the other hand, regardless of the transformation, performance of LSH is quite sensitive to its parameters. For large hash code lengths (16 or bigger), the probability of generating hash buckets with very low density increases significantly and small (or even empty) candidate sets are generated, leading to low accuracy (recall). For small hash code lengths (say, 4), most hash buckets become very dense, leading to large candidate sets, which produce high accuracy but also high values of inverse speed up (low efficiency). Almost all curves for LSH with $K = 4$ needs linear time (inverse speed up close to 1) to achieve high accuracy (90% or higher). Note that using third transformation for LSH with $k = 8$ has almost the best performance among all other LSH settings (different transformations and different K values).

We end this section noting that various non-LSH approximate nearest neighbor search methods such as randomized kd-tree, k-means tree etc., have superior empirical NNS performance compared to LSH (Muja and Lowe 2014). After applying appropriate transformation that reduces a MIPS problem to an equivalent NNS problem, these methods can be used to solve the reduced NNS problem as well. However, unlike RPT which uses *defeatist* query processing strategy and retrieves data points from a single leaf node of an RPT, these methods retrieve data points from multiple leaf nodes of a single tree by maintaining a priority queue. Also, unlike RPT, these methods provide no theoretical guarantee as far as nearest neighbor search performance is concerned. Therefor, we do not compare these methods in this paper.

6 Conclusions

In this paper we proposed the use of RPTs for the solution of MIPS for two main reasons—(i) to obtain a MIPS solution that allows simple but fine-grained control over the accuracy–efficiency trade-off, and (ii) to theoretically determine the best (among the set of existing) MIPS-to-NNS reduction in terms of the accuracy (for fixed efficiency). Our empirical results on eight real world datasets validate our theoretical claims and also demonstrate our superiority to the current state-of-the-art. For example, at 80% accuracy, our proposed MIPS solution produced results 2-5 times more efficiently than the state-of-the-art. This favorable performance comes at the cost of increased space complexity. Firstly, a single RPT has a memory requirement of $O(dn)$. Hence, the complete ensemble of L RPTs would require a memory overhead of $O(Lnd)$ which can be severely limiting. To address this limitation, we have proposed two space efficient version of RPTs, namely, RPTS and RPTB. Space complexity of RPTS is $O(Ld \log n)$ while space complexity of RPTB is $O(cd \log n)$, where c is typically set to $c = 2$ or $c = 5$ and is independent of L. Empirical evaluations show that RPTS and RPTB enjoy the same level of accuracy–efficiency trade-off as that RPT eliminating the limitations of RPT.

While we are able to achieve state-of-the-art performance for MIPS, we are not taking advantage of the fact that, after using **T1** to reduce MIPS to NNS, the norms $P_1(x) = Q_1(q) = \text{constant } \forall x \in S \ \& \ \forall q$. Is it possible to use this fact to design better algorithms for MIPS? Finally, there is also the unanswered question of whether (and how) we can develop a better, or even optimal, MIPS-to-NNS reduction in terms of the potential function. While we did not address this question here, we presented a precise notion (the potential function in the

transformed space) which can be used to answer questions such as "how is a MIPS-to-NNS reduction better?" or "how is a MIPS-to-NNS reduction optimal?".

References

Andoni, A., & Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, *51*(1), 117–122.

Bache, K., & Lichman, M. (2013). UCI machine learning repository. http://archive.ics.uci.edu/ml.

Bachrach, Y., Finkelstein, Y., Gilad-Bachrach, R., Katzir, L., Koeningstein, N., Nice, N., & Paquet, U. (2014). Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *8th ACM Conference on Recommender Systems*.

Cai, D. (2009). Text datasets in Matlab format. http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html.

Cremonesi, P., Koren, Y., & Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *4th ACM Conference on Recommender Systems*.

Curtin, R. R., & Ram, P. (2014). Dual-tree fast exact max-kernel search. *Statistical Analysis and Data Mining*, *7*(4), 229–253.

Curtin, R. R., Ram, P., & Gray, A. G. (2013). Fast exact max-kernel search. In *Proceedings of SIAM data mining*.

Dasgupta, S., & Sinha, K. (2013). Randomized partition trees for exact nearest neighbor search. In *The 26th Annual Conference on Learning Theory*.

Dasgupta, S., & Sinha, K. (2015). Randomized partition trees for nearest neighbor search. *Algorithmica*, *72*(1), 237–263.

Datar, M., Immorlica, N., Indyk, P., & Mirrokni, C. S. (2004). Locality-sensitive hashing based on p-stable distributions. In *The 20th ACM symposium on computational geometry*.

Dean, T., Ruzon, M., Segal, M., Shlens, J., Vijayanarasimhan, S., & Yagnik, J. (2013). Fast accurate detection of 100,000 object classes on a single machine. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Transactions on pattern Analysis and Machine Intelligence*, *32*(9), 1627–1645.

Gionis, A., Indyk, P., & Motwan, R. (1999). Similarity search in high dimensions via hashing. In *25th International Conference on Very Large Databases*.

Hyvönen, V., Pitkänen, T., Tasoulis, S. K., Jaasaari, E., Tuomainen, R., Wang, L., Corander, J., & Roos, T. (2016). Fast nearest neighbor search through sparse random projections and voting. In *IEEE International Conference on BigData*.

Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *ACM symposium on theory of computing*.

Jain, P., & Kapoor, A. (2009). Active learning for large multi-class problem. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Jégou, H., Douze, M., & Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(1), 117–128.

Joachims, T. (2006). Training linear SVM in linear time. In *12th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Joachims, T., Finley, T., & Yu, C.-N. J. (2009). Cutting plane training of structural SVMs. *Machine Learning*, *77*(1), 27–59.

Keivani, O., Sinha, K., & Ram, P. (2017). Improved inner product search with better theoretical guarantees. In *International Joint Conference on Neural Network*.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37.

Manjunath, B. S., & Ma, W. Y. (1996). Texture features for browsing and retrieving of large image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(1), 117–128.

Muja, M., & Lowe, D. G. (2014). Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(11), 2227–2240.

Neyshabur, B., & Srebro, N. (2015). On symmetric and asymmetric LSHs for inner product search. In *32nd International Conference on Machine Learning*.

Ram, P., & Gray, A. G. (2012). Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Shrivasatava, A., & Li, P. (2014). Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *28th Annual Conference on Neural Information Processing Systems*.

Shrivasatava, A., & Li, P. (2015). Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). In *31st Conference on Uncertainty in Artificial Intelligence*.

Sinha, K. (2014). LSH versus randomized partition trees: Which one to use for nearest neighbor search? In *13th International Conference on Machine Learning and Applications*.

Srebro, N., Rennie, J., & Jaaakkola, T. (2005). Maximum margin matrix factorization. In *19th Annual Conference on Neural Information Processing Systems*.

Trec interpolation. http://trec.nist.gov/pubs/trec16/appendices/measures.pdf. Accessed 14 Sept 2016.