Generalizations of the Theory and Deployment of Triangular Inequality for Compiler-Based Strength Reduction

Yufei Ding, Lin Ning, Hui Guan, Xipeng Shen
North Carolina State University, United States
{yding8, lning, hguan2, xshen5}@ncsu.edu

Abstract

Triangular Inequality (TI) has been used in many manual algorithm designs to achieve good efficiency in solving some distance calculation-based problems. This paper presents our generalization of the idea into a compiler optimization technique, named *TI-based strength reduction*. The generalization consists of three parts. The first is the establishment of the theoretic foundation of this new optimization via the development of a new form of TI named *Angular Triangular Inequality*, along with several fundamental theorems. The second is the revealing of the properties of the new forms of TI and the proposal of *guided TI adaptation*, a systematic method to address the difficulties in effective deployments of TI optimizations. The third is an integration of the new optimization technique in an open-source compiler. Experiments on a set of data mining and machine learning algorithms show that the new technique can speed up the standard implementations by as much as 134X and 46X on average for distance-related problems, outperforming previous TI-based optimizations by 2.35X on average. It also extends the applicability of TI-based optimizations to vector related problems, producing tens of times of speedup.

CCS Concepts • **Software and its engineering → Compilers**

Keywords Machine Learning, Deep Learning, Triangle Inequality, Strength Reduction, Compiler, Optimization

1. Introduction

Strength reduction is a traditional compiler optimization technique. By replacing expensive operations (e.g., $2\times b$) with equivalent but cheaper operations (e.g., $b \ll 1$), it helps improve program performance. Traditional strength reduction is mostly at the level of an individual instruction or statement. Some previous studies (e.g., Finite Differencing [35]) have tried to extend the scope, but they have still primarily focused on replacing multiplications or exponential operations (that involve loop indexing variables) with additions.

This paper concentrates on leveraging triangular inequality (TI) to materialize a type of large-scoped strength reduction.

TI refers to a well-known basic property of triangles: The length of an arbitrary edge of a triangle is less than the sum of the other edges' lengths and is greater than their difference, as illustrated in Figure 1.

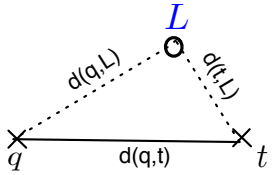


Figure 1. Illustration of traditional triangular inequality, where d(p_1,p_2) is the length between points p_1 and p_2.

TI offers a way to estimate the lower bounds and upper bounds of the distance between two points. Numerous algorithm designs [16, 19, 23, 25, 29, 32, 40] in various domains have manually employed TI for creating fast algorithms. These algorithms are typically for problems that care the distances only in a certain range. The basic idea is that if using bounds can already tell that the distance is impossible to fall into the range of interest, the algorithm can simply avoid computing that distance. Figure 2 shows how the idea helps avoid some distance computations in finding the nearest neighbors of some points—a popular instance-based machine learning method [32]. By comparing the lower bounds of the distance between two points with the currently shortest distance, the optimized code can typically avoid a majority of the distance calculations. Although the bounds calculation needs two other distances, in many situations, those distances are either known or can be reused across the bounds calculations for many points. Numerous previous studies of

some specific data mining algorithms [17, 20, 26, 32] have shown that such optimized algorithms can bring tens or even hundreds of times of speedups.

```
1:  for i = 0 to N do                    for i = 0 to N do
2:      minDist = Int_max;                   minDist = Int_max;
3:      for j = 0 to M do                    for j = 0 to M do
4:          dist = d(a(i), b(j));                lbDist = lb(a(i), b(j));
5:          if minDist > dist                    if minDist <= lbDist
6:              minDist = dist;                      continue;
7:              assign(i)= j;                    …
8:
9:                                           ///lb() function for lower bound of distance
10:                                          ……
11:
                 (a)                                  (b)
```

Figure 2. (a) original code (b) code that avoids some unnecessary distance calculations through the use of distance bounds.

All those prior studies are about manually applying TI to a certain algorithm design. A recent work [15] proposes a compiler-based framework named TOP to ease the process. TOP uses compilers to replace some special API calls with some TI-optimized library functions to get speedups. It is the first work that connects TI with compilers. However, the connection is yet shallow, mainly about using compilers as a tool to help programmers with the TI-related code replacement.

In this paper, we explore some deep connections between TI and compilers. It capitalizes on a key observation that, essentially, what those previous works did was a form of strength reduction: replacing expensive distance computations with cheaper comparisons with distance bounds. Based on that insight, this paper develops TI into a generalized compiler technique, named *TI-based strength reduction*. Compared to the previous TOP work [15], this work makes some major contributions in the underlying theory of the optimization, as well as its deployment and integration in compilers:

First, unlike TOP which bases the optimizations on traditional TI only, this work generalizes the theory of TI by developing a new type of TI, named *Angle Triangular Inequality* (ATI). ATI significantly expands the applicability of TI-based optimizations, and at the same time, enhances the tightness of the bounds. To distinguish them, from now on, we use ETI (Edge Triangular Inequality) for the traditional form of TI, ATI for the newly proposed form of TI, and TI for the union of the two. Unlike ETI, which is based on edges of triangles, ATI is on the relations among angles formed by three vectors[1]. We prove that ATI finds even tighter distance bounds than ETI does. When it is used together with ETI, ATI can help avoid even more distance calculations. Moreover, ATI expands the applicability of TI-based optimizations to include not just distance calculations but also vector-based computations, a scope no prior (manual or automatic) TI work has explored. Vector-based computations widely exist in scientific computing, graphic applications, deep neural networks [27], and similarity quantification in various text mining algorithms [4, 7]. Such a generalization is essential for making TI-based strength reduction into a compiler technology with a broad applicability. (Section 3)

Second, this work generalizes the deployment of TI-based optimizations. Complex code optimizations typically incur costs; TI-based strength reduction is no exception. And different deployments of TI face different cost-benefit tradeoffs. Finding the ways to apply an optimization appropriately is an essential part of the development of a compiler optimization technique. All prior works [15, 32] have been resorting to some ad-hoc thresholds for dealing with the tradeoffs. They are not robust because the tradeoff varies with the attributes of problem instances as our experiments show (Section 6). This work offers a systematic solution. It reveals the main factors and tradeoffs that are related to the deployment of ETI, ATI, and their combination. It then introduces *guided TI adaptation* to help efficiently determine the suitable way to configure the optimizations on the fly. (Section 4)

Finally, this work generalizes the way the optimization can be applied. This generalization eases the process for domain experts to apply the TI-based optimizations. It offers two options. For some C/C++ programs, it can automatically detect the applicable opportunities and transforms the code accordingly. For code not amenable for static analysis, the domain experts can still use a set of predefined APIs to reveal the semantics of the basic algorithm, based on which, the compiler applies the optimizations. (Section 5)

We evaluate the technique on a set of popular data mining, machine learning, and other kinds of applications, including a neural network training algorithm commonly used in deep believe network. The results show that our optimizations speed up standard implementations of those applications by up to tens or hundreds of times. For distance-based computations, it outperforms previous TI-based optimizations by 2.35X on average. It successfully expands TI-based optimizations to cover some vector-based computations that have benefited from no prior TI-based optimizations before, producing tens of times of speedups. (Section 6)

2. TI and Compiler Technique Development

The TI-based optimizations done in previous manual algorithmic designs [17, 20, 26, 32, 40] replace costly distance computations with less expensive bounds estimations, which resembles the high-level concept of the traditional strength reduction in compilers. However, to leverage the conceptual connection and turn TI into a compiler optimization technique requires innovations and substantial efforts in multiple dimensions.

[1] In this article, "vector" carries its mathematical meaning rather than refers to a type of data structure.

In general, to develop a compiler optimization technique, one needs to address questions in three major aspects:

(1) The first is to build up the theoretical foundations for the optimization. The theory could be based on various formalism, from math to logic, depending on the nature of the problem. Take polyhedral analysis-based code parallelization as an example. Its principled problem is how to identify data dependencies in the code, and its solution is based on integer linear programming in loop iteration space. The nature of TI-based strength reduction determines that its development would require some different formalism. Its principled problem is how to compute some tight bounds that can help avoid more expensive computations. Its solution calls for some theoretical developments upon Geometry and Linear Algebra, as we will see in the next section.

(2) The second aspect in developing optimization techniques is revealing the benefits, costs, and applicable conditions of the optimization, and offering ways to reconcile the various concerns in the deployment of the optimization. Most code optimization techniques are double-edged swords. They bring benefits but also incur costs, and are subject to certain applicability limitations. Therefore, an important part of the development of a new compiler optimization technique is to reveal these factors and come up with a solution to guide effective deployment of the optimization. This aspect is particularly important for TI-based strength reduction because it contains multiple variants (some on ETI, some on ATI, some on both) and each variant involves many possible configurations. Understanding their properties and effectively guiding their deployments to tap into their full potential are hence an essential part of the development.

(3) Finally, it is obvious that for an optimization technique to become part of a compiler, it has to be integrated into the compiler infrastructures. Although this part is mostly about engineering efforts, good designs could help make the technique flexible to apply to code with different complexities.

The next three sections explain our development of TI-based strength reduction on each of these three aspects.

3. Generalized Triangular Inequality Theory

This section presents the first dimension of the generalization. It introduces a new form of triangular inequality, ATI. Complementing the traditional triangular inequality (ETI), ATI is based on angles rather than edges. It offers tighter bounds and also extends TI-based strength reduction to some vector operations beyond distance calculations.

We describe some notations first. We use θ_{ab} to represent the angle between two vectors $\vec{a}$ and $\vec{b}$. Throughout this paper, the angle between two vectors is measured by the shortest great circle path between them. For instance, the angle between $\vec{q}$ and $\vec{t}$ is θ_{qt} in Figure 3 rather than its complement as it is greater than θ_{qt}. In another word, all the angles between two vectors are in the range $[0, \pi]$. Although there is no clear physical mapping of such angle when the vectors involved are in high-dimensional space, we could still follow formula $cos(\theta_{qt}) = \frac{\vec{q}\cdot\vec{t}}{|\vec{q}|\cdot|\vec{t}|}$, to compute θ_{qt} and restrict it to $[0, \pi]$. We next present the ATI theorem.

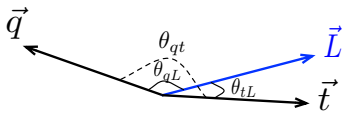


Figure 3. Illustration of angle triangular inequality (ATI).

3.1 ATI Theorem

THEOREM 1. **Angle Triangular Inequality:** *For three arbitrary vectors $\vec{q}$, $\vec{t}$ and $\vec{L}$ in a space, the angles among them, denoted as θ_{qt}, θ_{qL}, θ_{tL}, must meet the following condition:*

$$cos(\theta_{qL} + \theta_{tL}) \leq cos\theta_{qt} \leq cos(\theta_{qL} - \theta_{tL}). \quad (1)$$

This theorem gives the bounds of cosine values among three vectors. Cosine values are commonly used in text mining for similarity comparisons. Therefore, this theorem offers an important foundation for the potential usage of ATI in strength reduction for various text mining algorithms as Section 6 will show.

The three vectors and angles in Figure 3 illustrate the relations stated in the theorem. Note that the vectors can be of arbitrarily large dimensions and don't have to reside on a single 2-D plane. We give the proof as follows.

Proof: Let $\vec{u_q}$, $\vec{u_t}$ and $\vec{u_L}$ represent three unit-length vectors in the direction of $\vec{q}$, $\vec{t}$ and $\vec{L}$ respectively.

We introduce two derived vectors

$$\vec{e_1} = \frac{\vec{u_q} - \vec{u_L} \cdot cos(\theta_{qL})}{sin(\theta_{qL})}$$

$$\vec{e_2} = \frac{\vec{u_t} - \vec{u_L} \cdot cos(\theta_{tL})}{sin(\theta_{tL})}.$$

They are both unit vectors, as $\vec{e_1} \cdot \vec{e_1} = 1$ and $\vec{e_2} \cdot \vec{e_2} = 1$. Moreover, they are both perpendicular to $\vec{u_L}$, because $\vec{e_1} \cdot \vec{u_L} = 0$ and $\vec{e_2} \cdot \vec{u_L} = 0$.

It is easy to see that the following two formulas hold (easily provable by replacing $\vec{e_1}$ and $\vec{e_2}$ with their definitions):

$$\begin{aligned} \vec{u_q} &= \vec{u_L} \cdot cos(\theta_{qL}) + \vec{e_1} \cdot sin(\theta_{qL}) \\ \vec{u_t} &= \vec{u_L} \cdot cos(\theta_{tL}) + \vec{e_2} \cdot sin(\theta_{tL}). \end{aligned} \quad (2)$$

Multiplying the two equations gives (recall $\vec{e_1} \cdot \vec{u_L} = 0$ and $\vec{e_2} \cdot \vec{u_L} = 0$):

$$\vec{u_q} \cdot \vec{u_t} = cos(\theta_{qL})cos(\theta_{tL}) + \vec{e_1} \cdot \vec{e_2} sin(\theta_{qL})sin(\theta_{tL}).$$

As $|\vec{e_1} \cdot \vec{e_2}| \leq 1$ given by the *Cauchy-Schwarz Inequality*, and $sin(\theta) \geq 0$ for all $\theta \in [0, \pi]$, we can get the following relations:

$$\begin{aligned} \vec{u_q} \cdot \vec{u_t} &\geq cos(\theta_{qL})cos(\theta_{tL}) - sin(\theta_{qL})sin(\theta_{tL}) \\ \vec{u_q} \cdot \vec{u_t} &\leq cos(\theta_{qL})cos(\theta_{tL}) + sin(\theta_{qL})sin(\theta_{tL}). \end{aligned} \quad (3)$$

Recall the *Trigonometric Addition Formulas*:

$$\begin{aligned} cos(\theta_1 + \theta_2) &= cos(\theta_1)cos(\theta_2) - sin(\theta_1)sin(\theta_1) \\ cos(\theta_1 - \theta_2) &= cos(\theta_1)cos(\theta_2) + sin(\theta_1)sin(\theta_1). \end{aligned} \tag{4}$$

Therefore, we have

$$cos(\theta_{qL} + \theta_{tL}) \leq \vec{u_q} \cdot \vec{u_t} \leq cos(\theta_{qL} - \theta_{tL}).$$

Because $\vec{u_q} \cdot \vec{u_t} = cos(\theta_{qt})$ as both u_q and u_t are unit vectors, we get

$$cos(\theta_{qL} + \theta_{tL}) \leq cos(\theta_{qt}) \leq cos(\theta_{qL} - \theta_{tL}).$$

The ATI theorem is hence proved. □

Following the ATI theorem, considering the monotonic property of $cos(\theta)$ for $\theta \in [0, \pi]$, it is easy to get the following corollary:

COROLLARY 1. *For three arbitrary vectors $\vec{q}$, $\vec{t}$ and $\vec{L}$ in a space, the angles among them, denoted as θ_{qt}, θ_{qL}, θ_{tL}, must meet the following condition:*

$$|\theta_{qL} - \theta_{tL}| \leq \theta_{qt} \leq \pi - |\pi - (\theta_{qL} + \theta_{tL})|. \tag{5}$$

The far right expression is to convert the sum of the two angles into its counterpart in range [0, π].

Given that $\vec{x} \cdot \vec{y} = |\vec{x}||\vec{y}|cos(\theta_{xy})$, we immediately get the following corollary:

COROLLARY 2. *For three arbitrary vectors in a space $\vec{q}$, $\vec{t}$, $\vec{L}$, the following conditions must hold:*

$$\begin{aligned} \vec{q} \cdot \vec{t} &\geq |\vec{q}| \cdot |\vec{t}| \cdot cos(\theta_{qL} + \theta_{tL}) \\ \vec{q} \cdot \vec{t} &\leq |\vec{q}| \cdot |\vec{t}| \cdot cos(\theta_{qL} - \theta_{tL}). \end{aligned} \tag{6}$$

This corollary gives the bounds of vector dot products, which lead to the usage of ATI in strength reduction for dot product computations as discussed in section 3.2.2.

3.2 Applications for Strength Reduction

As we have mentioned and Figure 2 has illustrated, in many cases, the cost of computing bounds is much lower than that of the original computations, hence the usefulness of TI-based bound estimations for strength reductions. In the previous section, we have briefly mentioned that the cosine bounds from ATI could potentially help optimize text mining algorithms. In this section, we provide a deeper view at the possible applications of ATI for bounds calculations needed for strength reduction.

We first explain that ATI can also be used for estimating distance bounds, and then present an important theorem stating that ATI guarantees to give tighter (or same) bounds than ETI does. We then show that ATI is also useful for bounds calculations for vector-based computations, and exemplify it on a neural network algorithm used in deep learning.

Notation: We use the same letter without a top arrow to represent the corresponding end points of a vector (with the origin as the start). For instance, q is the end point of vector $\vec{q}$.

3.2.1 ATI For Distance Bounds

There is a well known connection between vector dot product and distance calculations. Consider two vectors $\vec{q}$ and $\vec{t}$. The distance between q and t, represented as $d(q, t)$, has the following relation with vector dot product:

$$d^2(q, t) = |\vec{q} - \vec{t}|^2 = |\vec{q}|^2 + |\vec{t}|^2 - 2\vec{q} \cdot \vec{t}. \tag{7}$$

Following Corollary 2, we get the following bounds for $d(q, t)$ (lb for lower bound, ub for upper bound):

$$\begin{aligned} lb(d^2(q,t)) &= |\vec{q}|^2 + |\vec{t}|^2 - 2ub(\vec{q} \cdot \vec{t}) \\ &= |\vec{q}|^2 + |\vec{t}|^2 - 2|\vec{q}| \cdot |\vec{t}| \cdot cos(\theta_{qL} - \theta_{tL}) \\ ub(d^2(q,t)) &= |\vec{q}|^2 + |\vec{t}|^2 - 2lb(\vec{q} \cdot \vec{t}) \\ &= |\vec{q}|^2 + |\vec{t}|^2 - 2|\vec{q}| \cdot |\vec{t}| \cdot cos(\theta_{qL} + \theta_{tL}). \end{aligned} \tag{8}$$

Such bounds are even tighter than the bounds from the traditional ETI. Formally, we have the following theorem:

THEOREM 2. **Tighter ATI-based Distance Bound:** *For three arbitrary vectors $\vec{q}$, $\vec{t}$ and $\vec{L}$ in a space, distance bounds obtained through ATI are never less tight than those obtained through ETI. In another word, the following always holds:*

$$\begin{aligned} |d(q, L) - d(t, L)| &\leq \sqrt{|\vec{q}|^2 + |\vec{t}|^2 - 2|\vec{q}| \cdot |\vec{t}| \cdot cos(\theta_{qL} - \theta_{tL})} \\ d(q, L) + d(\vec{t}, \vec{L}) &\geq \sqrt{|\vec{q}|^2 + |\vec{t}|^2 - 2|\vec{q}| \cdot |\vec{t}| \cdot cos(\theta_{qL} + \theta_{tL})} \end{aligned} \tag{9}$$

where, the left hand sides (LHS) denote the bounds of $d(q, t)$ computed through ETI, and the right hand sides (RHS) are the bounds through ATI. We give the proof as follows.

Proof: As per Formula 7, $d(q, L) = \sqrt{|\vec{q}|^2 + |\vec{L}|^2 - 2\vec{q} \cdot \vec{L}}$; $d(t, L)$ can be rewritten to a similar form. With these rewritings, the two LHS of Formula 2 become the following form respectively:

$$\begin{aligned} lb_{eti} =& |\sqrt{|\vec{q}|^2 + |\vec{L}|^2 - 2|\vec{q}||\vec{L}|cos(\theta_{qL})} \\ & - \sqrt{|\vec{t}|^2 + |\vec{L}|^2 - 2|\vec{t}||\vec{L}|cos(\theta_{tL})}\,| \\ ub_{eti} =& \sqrt{|\vec{q}|^2 + |\vec{L}|^2 - 2|\vec{q}||\vec{L}|cos(\theta_{qL})} \\ & + \sqrt{|\vec{t}|^2 + |\vec{L}|^2 - 2|\vec{t}||\vec{L}|cos(\theta_{tL})}. \end{aligned} \tag{10}$$

We next prove that for arbitrarily given $\vec{q}$ and $\vec{t}$, and a given direction of $\vec{L}$, no matter what the length of $\vec{L}$ is, the largest value of lb_{eti} (the lower bound of d(q,t) from ETI) is no larger than the lower bound of d(q,t) given by ATI (i.e., the top inequality in Formula 2.

The proof goes as follows. The condition for lb_{eti} to reach its maximal value is that its derivative over $|\vec{L}|$ must equal to zero. That is, $\frac{d(lb_{eti})}{d|\vec{L}|} = 0$. Solving that equation, we get $|\vec{L}| = \frac{|\vec{q}| \cdot |\vec{t}| \cdot sin(\theta_{tL} - \theta_{qL})}{|\vec{t}| \cdot sin(\theta_{tL}) - |\vec{q}| \cdot sin(\theta_{qL}))}$, and the value of lb_{eti} at that $|\vec{L}|$ is $\sqrt{|\vec{q}|^2 + |\vec{t}|^2 - 2|\vec{q}| \cdot |\vec{t}| \cdot cos(\theta_{qL} - \theta_{tL})}$, exactly equaling the first RHS in Formula 2. It hence proves that for all the possible values of $\vec{L}$ along the given direction,

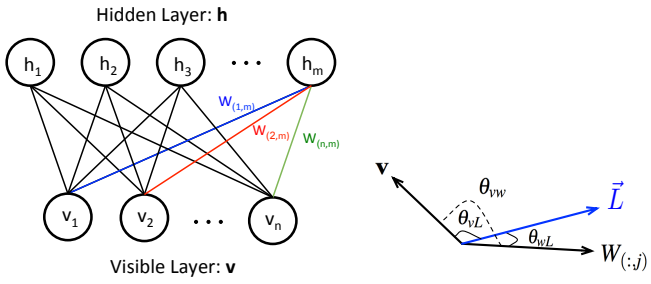


Figure 4. (a) A Binary RBM with n visible units and m hidden units (b) ATI on vector dot product.

the lower bound of $d(q,t)$ computed by ATI is never smaller than the lower bound computed by ETI.

In a similar way (through calculations of the derivative of ub_{eti}), it can be proved that ATI also gives the smallest upper bound that ETI can give. The Tighter ATI-based Distance Bound Theorem is hence proved. □

This theorem is fundamental, concluding on the effectiveness of ATI over ETI in bounding distances. It suggests the potential of using ATI to help avoid more distance calculations than that using ETI. To our best knowledge, this is the first time that the relationship between ATI and ETI on bounding distances is revealed.

3.2.2 ATI For Vector Product

Besides for distance bounds calculations, the bounds that ATI gives for vector dot product can be directly of use for optimizing computations that involve comparisons to vector dot products. Such computations exist in many scientific computing, graphics, data analytics, and machine learning applications. An important example with dot product computations for comparisons is the Restricted Boltzman Machine (RBM), an influential type of artificial neural network used in deep learning [38]. We take it as an example to explain the usage of ATI for optimizing vector computations.

Example RBM is composed of two layers of units as illustrated in Figure 4 (a): a visible layer with n visible units and a hidden layer with m hidden units. The values of the visible nodes together form an n-dim vector $\mathbf{v}$, and those of the hidden-layer nodes form an m-dim vector $\mathbf{h}$. An RBM is characterized by a set of parameters: $\theta = (\mathbf{a}, \mathbf{b}, \mathbf{W})$, where, $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ are the bias vectors for the visible and hidden layers respectively, and $\mathbf{W} \in \mathbb{R}^{n \times m}$ is the weight matrix that contains the weights on the edges between each pair of visible-hidden units.

The standard training algorithm for an RBM is based on Gibbs Sampling, which involves iterative two-way value propagation between the visible and hidden layers. Taking the propagation from the visible layer to the hidden layer as an example, the propagation is based on the following conditional probability calculation:

$$P(h_j = 1|\mathbf{v}) = \sigma(b_j + \mathbf{v}^T W_{(:,j)}), \tag{11}$$

which involves vector dot product $\mathbf{v}^T W_{(:,j)}$. As both $\mathbf{v}$ and $\mathbf{W}$ are high dimensional (hundreds or thousands), the dot product (done on all nodes many times) consumes most of the training time. In Formula 11, function $\sigma(\cdot)$ is the sigmoid activation function, which is a monotonic increasing function.

From the conditional probability, the value of unit h_j is determined as follows:

$$h_j = \begin{cases} 1 & if \;\; r < P(h_j = 1|\mathbf{v}) \\ 0 & otherwise \end{cases} \tag{12}$$

where, r is a random number in the range [0,1].

Let $\mathbf{L}$ be a vector in the space of $\mathbf{v}$. By applying Corollary 2, we can compute the bounds of the conditional probability as follows:

$$\begin{aligned} lb(P(h_j = 1|\mathbf{v})) &= \sigma(b_j + lb(\mathbf{v}^T W_{(:,j)})) \\ &= \sigma(b_j + |\mathbf{v}| \cdot |W_{(:,j)}| cos(\theta_{vL} + \theta_{wL})) \\ ub(P(h_j = 1|\mathbf{v})) &= \sigma(b_j + ub(\mathbf{v}^T W_{(:,j)})) \\ &= \sigma(b_j + |\mathbf{v}| \cdot |W_{(:,j)}| cos(\theta_{vL} - \theta_{wL})). \end{aligned} \tag{13}$$

So according to Formula 12, if $lb(P(h_j = 1|\mathbf{v})) > r$, then h_j can be set to 1, and if $ub(P(h_j = 1|\mathbf{v})) \leq r$, then h_j can be set to 0. In both cases, there is no need for computing $P(h_j = 1|\mathbf{v})$.

These bounds are much cheaper to compute than $P(h_j = 1|\mathbf{v})$. Consider that there are N instances of $\mathbf{v}$ and m hidden nodes (i.e., $1 \leq j \leq m$ in Formula 12). For a given L, the lower or upper bounds need $N + m$ dot products to compute the angles θ_{vL} and θ_{wL}, m cos(), and $2 * N * m$ scalar multiplications. In comparison, the original $P(h_j = 1|\mathbf{v})$ needs $N * m$ vector dot products. As $\mathbf{v}$ is usually in hundreds or thousands of dimensions, saving these dot products with the bounds can be quite beneficial.

Tighter Bounds for Vector Computations We note that just as ATI can be used for bounding distance calculations, ETI can also be used for bounding vector dot products. Equation 7 can easily reformulate to the following:

$$\vec{q} \cdot \vec{t} = 1/2(|\vec{q}|^2 + |\vec{t}|^2 - d^2(q,t)). \tag{14}$$

Replacing $d(q,t)$ with the lower and upper bounds from ETI, we can immediately get the bounds of $\vec{q} \cdot \vec{t}$ respectively: $1/2 \cdot (|\vec{q}|^2 + |\vec{t}|^2 - (d(q,L) + d(t,L))^2)$ and $1/2 \cdot (|\vec{q}|^2 + |\vec{t}|^2 - (d(q,L) - d(t,L))^2)$.

The tighter bounds on distances from ATI over ETI (Theorem 2) directly leads to the following corollary:

COROLLARY 3. **Tighter ATI-based Vector Product Bound:** *For three arbitrary vectors $\vec{q}$, $\vec{t}$ and $\vec{L}$ in a space, vector dot product bounds obtained through ATI are never less tight than those obtained through ETI. In another word, the following always holds:*

$$\begin{aligned} &1/2 \cdot (|\vec{q}|^2 + |\vec{t}|^2 - (d(q,L) + d(t,L))^2) \leq |\vec{q}||\vec{t}| cos(\theta_{qL} + \theta_{tL}) \\ &1/2 \cdot (|\vec{q}|^2 + |\vec{t}|^2 - (d(q,L) - d(t,L))^2) \geq |\vec{q}||\vec{t}| cos(\theta_{qL} - \theta_{tL}). \end{aligned} \tag{15}$$

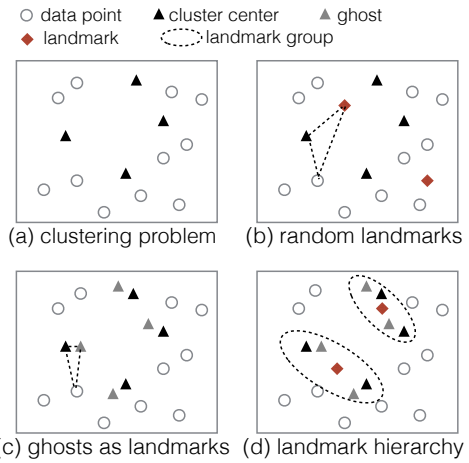


Figure 5. Illustrations of landmarks, ghosts, and landmark hierarchy on a clustering example.

The development of ATI and the related theorems expands the applicability of TI-based optimizations to vector computations besides distance calculations. It also reveals the relative power of ATI and ETI in bounding distances and dot products. These findings, along with the traditional triangular inequality (i.e., ETI), form the theoretical foundation for TI-based strength reduction.

4. Guided Adaptation for Deployment

Although ATI is more powerful than ETI in bounding distances and vector computations, the tightness of bounds is not the only factor relevant to the benefits of TI-based strength reduction. Meanwhile, applying either ETI or ATI faces a number of tradeoffs. The benefits and overhead are sensitive to the attributes (size, dimensions, etc.) of the data sets to operate on, the properties of the program to optimize, and some other factors. Some of these factors (e.g., data attributes) are not known until the execution time of the program.

Therefore, the second essential step in developing the technique of TI-based strength reduction is to find out the various factors that influence the cost and benefits of the deployment of the optimization, and to come up with ways to effectively guide the deployment of the optimization by compilers (and runtime).

This section describes those insights we have obtained, and present our solution, *guided TI adaptation*.

4.1 Terminology

Before getting to the tradeoffs and solutions, we first introduce some terminology that is essential for understanding the rest of the discussion.

Landmarks TI-based strength reduction works in the domain of (often high-dimensional) points or vectors. To optimize the calculation of the distance between two points or the dot product of two vectors, a third point or a third vector would be needed to form a triangle or three angles in order for ETI or ATI to work. Such a point or vector is called a *landmark*.

Landmarks could be created and shared. Consider a clustering problem, in which, there are some data points and some cluster centers, and the goal is to find the cluster center closest to each data point, as Figure 5 (a) illustrates. One may pick a random location in the space as the landmark and use it to form the triangles for optimizing the distance calculations between every data point and every data center, as illustrated in Figure 5 (b).

It is worth noting that as per their definitions, ETI and ATI give tight bounds when the landmark is near one of the two points (or vectors) in question. Having multiple landmarks could offer more choices and hence help get tighter bounds.

Ghosts In many iterative algorithms, the locations of the points or the values of the vectors in question get incremental updates across iterations. We call their locations/values in the previous iteration their *ghosts*. Because a ghost of a point is often close to that point, it can often serve as a good landmark.

Consider the aforementioned clustering example in Figure 5 (a). If the centers move slightly across iterations as illustrated in Figure 5 (c), using the ghosts as landmarks could help ETI or ATI give tighter bounds than using random landmarks. Moreover, the computations already done on those ghosts (e.g., distance to a landmark) may help save some computations in bounds calculations by ETI or ATI.

Landmark Hierarchy and Group Filtering Although using ghosts as landmarks in Figure 5 (c) helps tighten the bounds, there could be too many of them. Using them to get the distance bounds for every point could incur substantial time and space cost.

Landmark hierarchy could help mitigate the issue. Figure 5 (d) illustrates a two-level landmark hierarchy for the clustering example. The low-level landmarks are the ghosts, while each high-level landmark is the center location of a group of low-level landmarks.

Landmark hierarchy enables *group filtering* with ghosts: Suppose $lb(q, G_i)$ is the lower bound between a data point q and all cluster centers within group G_i. If through TI, the optimization shows that $lb(q, G_i)$ is greater than the upper bound of the distance between q and its closest center ($ub(q)$), then no t within G_i can be the closest center to q; distance calculations from those t to q can all be avoided. The low-level landmarks could be used if one wants to make $ub(q)$ tight or when the group filtering fails.

For non-iterative problems (e.g. KNN), there are no iterative searches over some dynamically-updated data sets, and thus no ghosts can be used as low-level landmarks.

Landmark hierarchy uses the high-level landmarks to reduce the cost while using the low-level ones to get tight bounds when necessary. It helps to strike a balance between the bound estimation cost and the estimation quality. It, how-

ever, introduces the complexities in determining the appropriate group size and grouping overhead.

4.2 Existing Insights

As ETI has been used in previous algorithm designs by domain experts, there is already a certain degree of understanding on what landmarks should be used to better take advantage of ETI [16, 19, 23, 25, 29, 32, 40], which has been summarized and extended in the previous TOP work [15]. We review those existing insights as follows. The description assumes that the problem is about distances between a set of query points (Q) and a set of target points (T).

1. If the algorithm is not iterative (i.e., repeatedly update Q or T and recompute their distances), the landmarks can be selected through lightweight clustering (e.g, 5-iteration K-Means clustering) on either T or Q.
2. For iterative algorithm with T or Q getting updated repeatedly, we could use the counterparts of T or Q in the previous iteration as landmarks for this iteration.
3. If the memory space is stringent and the dimension of points is not large, consider to use two levels of landmarks by grouping nearby low-level landmarks into a high-level landmark.

These insights are valuable, but they are insufficient for automatic TI-based strength reduction, for two reasons. First, all these insights are about ETI. The newly proposed ATI differs from ETI in some important ways. There is yet no previous understanding on the proper usage of it. Second, the insights on ETI are qualitative, appearing fuzzy and ambiguous, exemplified by "$|T|$ is much smaller than $|Q|$", "space is stringent", "dimension is not large". Automatic deployment of the optimizations requires quantitative measures for using these insights. For instance, how much space is considered stringent? Is it relative to the size of the problem? If so, how to tell whether the condition is met for a particular problem? Previous work addresses these questions by using some thresholds, which are often fragile, working well on some data sets but poorly on others as Section 6 will show. We next present our solutions to both issues.

4.3 Special Properties Related with ATI

This section describes four special properties of ATI for strength reduction. The first is its most important appealing property that Section 3.2.1 has already proved, we repeat it here for completeness. The second is about what landmarks ATI prefers. The third and fourth are about the relations between ATI and two important optimizations for TI-based strength reduction: the use of group filtering, and the use of early stop.

Property I: Tighter Bounds. For three arbitrary points, the distance, cosine similarity, and vector product bounds from ATI (on the corresponding vectors) are never less tight than those from ETI. This property makes ATI appealing in many usage cases.

Property II: Landmark Preference. Unlike ETI, which prefers landmarks close to either point in the question, ATI prefers landmarks that form small angles from either of the vectors in question. It is easy to see, from the definition of ATI, that such landmarks give tighter bounds than those with large angles from both vectors.

Property III: Group Filtering. Regarding group filtering, we have the following insight:

> For saving distance calculations, ETI is amenable for group filtering but ATI is not; for saving cosine similarity comparisons, ATI is amenable but ETI is not; for saving dot products, neither ETI nor ATI is amenable for group filtering.

We next give a detailed explanation of that insight on distance calculations and a brief explanation on the other two cases.

- Distance:
  For a group of target points G and a given landmark L, according to Formula 8, the distance bounds from a query point to G can be written as follows:

$$\begin{aligned} lb(d(q,G)) &= \sqrt{|\vec{q}|^2 + \min_{\vec{t}\in G}(|\vec{t}|^2 - 2|\vec{q}|\cdot|\vec{t}|\cdot cos(\theta_{qL} - \theta_{tL}))}; \\ ub(d(q,G)) &= \sqrt{|\vec{q}|^2 + \max_{\vec{t}\in G}(|\vec{t}|^2 - 2|\vec{q}|\cdot|\vec{t}|\cdot cos(\theta_{qL} + \theta_{tL}))}. \end{aligned} \tag{16}$$

  While the distance bounds from a query point to G based on ETI can be written as follows:

$$\begin{aligned} lb(d(q,G)) &= d(q,L) - \max_{\vec{t}\in G} d(L,t); \\ ub(d(q,G)) &= d(q,L) + \max_{\vec{t}\in G} d(L,t). \end{aligned} \tag{17}$$

  So, for getting the bounds for a group of target points, ETI needs just the farthest distance from the target points to the landmark (e.g., $max(d(L,t))$). In comparison, as Equation 16 shows, the bounds calculation by ATI would need the bounds on both the angles and the lengths of the target vectors. Efficiently computing and tracking both kinds of bounds add extra complexities. In addition, unlike ETI, in which $max(d(L,t))$ stays the same across different query points, the bounds of the angles $cos(\theta_{qL} - \theta_{tL})$ used in Equation 16 can be different across query points (as θ_{qL} varies with $\vec{q}$), making it even more difficult to compute and track angle bounds efficiently.
- Cosine Similarity:
  On the other hand, for cosine similarity, the group bounds from ATI depend only on the largest angle from the target vectors to the landmark (e.g., $max(\theta_{\vec{L},\vec{t}})$), which stays the same across different query points. But the group bounds from ETI depend on multiple factors and are

hence harder to get. We list the group bounds from ATI and ETI in the Formulae 18 and 19 respectively, with the derivation details omitted.

$$lb(cos(\theta_{\vec{q},G})) = \begin{cases} cos(\theta_{\vec{q},\vec{L}} + \max_{\vec{t}\in G} \theta_{\vec{L},\vec{t}}) & \text{if } \theta_{\vec{q},\vec{L}} + \max_{\vec{t}\in G} \theta_{\vec{L},\vec{t}} \leq \pi; \\ -1 & \text{otherwise.} \end{cases}$$
$$ub(cos(\theta_{\vec{q},G})) = \begin{cases} cos(\theta_{\vec{q},\vec{L}} - \max_{\vec{t}\in G} \theta_{\vec{L},\vec{t}}) & \text{if } \max_{\vec{t}\in G} \theta_{\vec{q},\vec{t}} \leq \theta_{\vec{q},\vec{L}}; \\ 1 & \text{otherwise.} \end{cases} \quad (18)$$

$$lb(cos(\theta_{\vec{q},\vec{G}})) = \min_{\vec{t}\in G}((|\vec{q}|^2 + |\vec{t}|^2 - (d(q,L) + d(L,t))^2)/2|\vec{q}||\vec{t}|);$$
$$ub(cos(\theta_{\vec{q},\vec{G}})) = \max_{\vec{t}\in G}((|\vec{q}|^2 + |\vec{t}|^2 - (d(q,L) - d(L,t))^2))/2|\vec{q}||\vec{t}|). \quad (19)$$

Note that Formula 18 is derived based on the assumption that all angles are in the range of $[0, \pi]$. In the cases (e.g. document clustering) where only positive cosine similarity is interesting, the formulae can be easily modified by replacing -1 with 0 and π with $\pi/2$.

- Vector Product:
  Formulae for computing the group-level vector product bounds can be derived from Formula 18 and Formula 19 by multiplying $|\vec{q}| * |\vec{t}|$ on both sides. Both would be relevant to multiple factors, and are hence difficult to compute. So in general, for vector products, no group filtering is used.

Property IV: Early Stop. Early stop is an optimization in TI-based strength reduction. Consider that a program needs to check whether the distances from q to a group of points $t \in G$ are smaller than a constant C. Suppose all these points share the same landmark L. An efficient way to do the check is to first sort the points in G in a descending order of $d(t, L)$. As a result, the upperbounds of $d(q, t)$ by ETI ($d(q, L) + d(t, L)$) would be in a descending order as well. So as soon as the check encounters a point whose upperbound is smaller than C, no checks would need to do for the remaining points because their upperbounds, and hence $d(q, t)$, must be smaller than C.

Our insight on early stop is similar to that on group filtering:

> For saving distance calculations, ETI is amenable for early stop but ATI is not; for saving cosine similarity comparisons, ATI is amenable but ETI is not; for saving dot products, neither ETI nor ATI is amenable for early stop.

The reasons for the insights are the same as those for the insights on group filtering. For example, consider ATI-based distance saving. Because the lower bounds from ATI are related with both the length of the target vector and the angle between that vector and the landmark, sorting the target points based on their distance lower bounds from ATI is hence difficult to do efficiently.

Problem Type	Tightness		Grouping		Early Stop	
	ETI	ATI	ETI	ATI	ETI	ATI
Distance	✗	✓	✓	✗	✓	✗
Vector Product	✗	✓	✗	✗	✗	✗
Cosine	✗	✓	✗	✓	✗	✓

Table 1. Comparison between ATI and ETI in terms of bound tightness and support of group filtering and early stop over different problem types.

Based on all these analytical results, we use Table 1 to summarize the important properties of ATI and ETI for TI-based strength reduction. ATI is more powerful than ETI in getting tight bounds, but for distance calculations, it is not amenable to group filtering and early stop.

4.4 Insights for Deploying ATI

These properties suggest the following insights for employing ATI for strength reductions.

(1) ATI shall be used without ETI for optimizing vector dot products, and cosine similarities (e.g. top k document retrieval, document clustering, and RBM). In such cases, landmarks shall be created based on the angles of the vectors (e.g., running a lightweight K-Means clustering on the angles of all vectors). Grouping can be applied for algorithms with cosine similarity, but should be avoided for optimizing vector dot products.

(2) When ATI is used for optimizing distance calculations, it is best to be combined with ETI. Such a combination could leverage the best of both worlds: benefiting from the tighter bounds that ATI provides, and at the same time, enjoying the benefits of grouping and early stop that ETI could bring.

We design an algorithm to combined ATI and ETI for distance calculations. It is outlined in Figure 6. It first checks whether *group filtering* shall be applied. If so, ETI is needed for computing the group-level bounds. In the meantime, target points in the same group would be sorted based on their distances to the landmark (for *early stop*). If the group filtering fails on a group of target points, point-level filtering is applied to them. ETI is first used because it allows *early stop* as described in Section 4.3. If ETI-based filtering fails, ATI-based point-level filtering is used. In our experiment, we found that ATI can frequently filter out around half the remaining cases thanks to the tighter bounds it provides. If that filtering also fails, the distance is computed.

4.5 Guided TI Adaptation

We develop a *guided TI adaptation* technique to tackle the second issue in deploying TI-based strength reduction, which is to automatically determine the suitable configurations in the deployment of the optimization.

As mentioned earlier, the suitable way to apply TI-based strength reduction relies on many factors, some of which (e.g., problem size, data dimensions) remain unknown until the execution time of the program. An ideal solution hence

```
//check whether group filtering is applicable
....

if (group filtering is applicable)
   //prepare for group-level filtering with ETI
   for L in Landmarks do
      sort target points in L based on their distances to L

for i = 0 to |Q| do
   //ETI for group-level filtering
   for L in Landmarks do
      if ETI_bound(Q[i], L) passes the comparison
         continue;
      for target point t in L do
         //ETI for point-level filtering
         if ETI_bound(Q[i], t) passes the comparison
            break;
         //ATI for point-level filtering
         if ATI_bound(Q[i], t) passes the comparison
            continue;
         //if all previous filtering fails, run the original code.
         .....
```

Figure 6. Pseudo-code for combined optimization of distance calculations by ETI and ATI-based strength reduction.

must be adaptive to the many runtime factors, and at the same time, incur only minimum overhead.

Guided TI adaptation tries to achieve these goals through a careful combination of qualitative insights, cost-benefit modeling, and runtime sampling. It uses the aforementioned qualitative insights to help narrow down the configuration space of the optimization, employs cost-benefit modeling to characterize some analyzable aspects of the performance and overhead, and uses runtime sampling to treat the aspects that are difficult to model.

4.5.1 Space Cost

A suitable deployment of the TI-based strength reduction should have an acceptable space cost, regarding the memory space budget given either by the user or by the hardware limitation. Space cost includes the space for storing landmarks and the distances (or bounds) between points and landmarks. It is mainly determined by the size of the problem and the number of landmarks. With such information, the space cost can be easily computed analytically. For a given landmark creation scheme, these models help determine the maximum number of landmarks allowed to create to fit in the given space budget. Execution time is more complicated; we give it a more detailed discussion.

4.5.2 Time Cost and Benefit

The time cost and benefit of TI-based strength reduction are hard to model in a static way. Take the ETI optimization as an example, it helps avoid some distance calculations between queries and targets, but also introduces time overhead, including the time for computing bounds between queries and targets, distances (or bounds) from landmarks to queries or targets, and extra comparisons among bounds and distances for avoiding distance calculations. The benefits and costs depend on the size of the problem, the number of landmarks, but also the locations or distributions of the queries and targets. It is more difficult to compute the time cost and benefit analytically to determine the suitable ways to create or select the landmarks for a given problem.

Our method uses the qualitative insights listed in Sections 4.2 and 4.3 to first determine the possible directions to explore, and then uses runtime sampling to precisely determine the solution.

Algorithm Based on the qualitative insights, the algorithm quickly classifies a given program into one of the six categories: non-iterative distance calculations, iterative distance calculations, non-iterative dot product, iterative dot product, non-iterative cosine similarity, iterative cosine similarity. We take the first category as an example to explain our method, which include non-iterative problems on distances between two sets of points (called query and target sets).

The method considers only one-level landmarks as per the qualitative insights. It contains a built-in performance model for the time savings that the TI-based strength reduction can offer, shown as follows:

$$\begin{aligned} T_{save} &= T_{savedDistance} - T_{overhead}; \\ T_{savedDistance} &= (r_d \cdot n \cdot m) \cdot t_{distance}; \\ T_{overhead} &= T_{createLM} + T_{LMdistance} + T_{checks} \\ &\simeq (p \cdot m \cdot k) \cdot t_{distance} \\ &\quad + (n+m) \cdot t_{distance} \\ &\quad + (r_c \cdot n \cdot m + n \cdot k) \cdot t_{checks}; \end{aligned} \tag{20}$$

where, k is the number of landmarks, m and n are the numbers of target points and query points, r_d is the fraction of distance calculations avoided through the TI optimization, $t_{distance}$ is the time taken to calculate the distance of one pair of points, t_{checks} is the time taken to conduct one conditional check on bounds, r_c is the fraction of bound calculations carried out between each pair of query and target points. Thanks to group filtering and early termination, r_c usually is much smaller than one. The formula assumes that the landmarks are created through p iterations of K-Means clustering applied to the target points.

The formula for $T_{savedDistance}$ in the model is the amount of time saved on distance calculations. The three components of $T_{overhead}$ are respectively the time taken for creating landmarks, the time for computing the distances from each target point to its associated landmark, and the time for checking the distance bounds. The optimization introduces some other operations, but they are omitted from the model as the time they take is negligible compared to those three parts.

The cost-benefit tradeoff of TI-based strength reduction is embodied by the model: The larger k is, the tighter the bounds are, and hence the larger r is and the larger $T_{savedDistance}$ is, but at the same time, the larger $T_{overhead}$ is. The goal of our automatic configuration is to determine the value of k to maximize T_{save}.

The challenge is that the relation between k and r_d and r_c is difficult to model because it depends on the distributions

of the data values. That makes it hard to figure out the best k analytically.

We circumvent the difficulty through a runtime sampling-based method. The method consists of the following steps.

(1) *Sampling.* It takes a random small portion (1% in our experiments) of the data sets to form a sample S.

(2) *Hierarchical Clustering.* It runs a quick k-means on S to get k groups, where k is set to $3\sqrt{|S|}$. It then runs hierarchical clustering on the centroids of the groups to build a cluster hierarchy (a tree) with a higher level cluster composed of some smaller clusters. Such a hierarchy offers the flexibility for examining the influence of different numbers of landmarks.

(3) *Trials.* It applies TI-based strength reduction to the computations on S in repeated trials. In each trial, it uses a different number of landmarks by taking the centers of the clusters at a certain level in the cluster hierarchy. Specifically, it starts from the bottom level of the hierarchy containing $3\sqrt{|S|}$ landmarks, and goes up a step on the hierarchy after each trial such that the number of landmarks reduces by about $0.5\sqrt{|S|}$ each time. It records the fraction of distance calculations that are avoided in each trial. As side products of the trials, it attains the average $t_{distance}$ and the average t_{checks}.

(4) *Binary Search.* Based on the data collected from Step 3, for an arbitrary number of landmarks, through scaling (x landmarks for S correspond to $x * \sqrt{|D|/|N|}$ to the whole date set D) and interpolation, the method can estimate the savable fraction of distance calculations of the entire data set, with which, T_{save} can be computed for that number of landmarks through Equation 20. That allows the use of Binary Search to quickly find the best number of landmarks.

For the other cases, our solution works in a similar manner. Details are omitted. It is worth noting that the guided TI adaptation is intended to be used when the input data sets are non-trivial (over 10K points), the sample of which can capture the characteristics of the entire data set. Such datasets also need the optimizations the most. When the data set is small, the need is usually less; prior simpler methods could be used.

5. Integration with Compilers

Based on LLVM, we integrate TI-based strength reduction into a prototype compiler. The compiler supports two modes. It tries to use code pattern matching to automatically detect the opportunities for applying the optimizations, and transforms the code accordingly. At the same time, it offers a set of APIs. Using these APIs, a programmer can express the semantics of the basic algorithms, upon which, the compiler applies the optimizations. This second mode makes the optimization useful even if the original code is not immediately amenable for static analysis.

```
p = a calculated distance or dot product
q = f (p); // some value is derived from p;
          // f() is a monotonic relation;
          // inside f(), x is indep. of p;

if (q op x){ // op is a comparison operator
  body_1;
} // body_1 may read or write p or q
body_2; // no reads of p or q
```

(a) Pattern 1

```
p = a calculated distance or dot product
q = f (p); // some value is derived from p;
          // f() is a monotonic relation;
          // inside f(), x is indep. of p;

if (q op x){ // op is a comparison operator
  body_3;
}
else{
  body_4;
} // only one of body_3 and body_4 can read p or q
body_5; // no reads of p or q
```

(b) Pattern 2

Figure 7. Allowed usage patterns of distances or dot products.

5.1 Through Pattern Matching

The code pattern that the compiler looks for is loops (or BLAS [5] functions) for vector dot product, matrix-matrix multiplication, or distance calculations, with some comparisons over their results following the loops (or function calls).

Specifically, we build a detection gadget with LLVM/-Clang C/C++ compiler frontend. Based on LibTooling and LibASTMatchers supplied by this frontend, it can search for the piece of code having those patterns. Three AST node matchers (forStmt, binaryOperator, FunctionDecl) are used to do the matching. For distance calculations, it currently supports basic patterns for computing Euclidean distances.

Checking the code following the computations of distances and dot products has some intricacies. For TI-based strength reduction to work soundly, the usage of the distances or dot products has to meet some conditions. For instance, if they are the ultimate output of the program, TI-based strength reduction shall not be applied as it avoids the computations of some of the results. Specifically, our compiler module checks whether the usage is one of the two patterns illustrated in Figure 7. In the first pattern, the relevant condition check has only one branch, while the second pattern allows two branches. However, in either case, only one branch reads values of or derived from the distance or dot product. That ensures that the bounds-based filtering by the TI-based strength reduction can work properly. The function "f" in Figure 7 represents monotonic relations. An example is the *sigmoid* function commonly used in artificial neural networks. The monotonicity is necessary for keeping the derived values from the bounds useful.

5.2 Through Assistance of API

Some programs are not amenable for static analysis due to code complexities (e.g., aliases and pointers). To ease the application of TI-based strength reduction in such cases, inspired by some previous work [15], we provide a set of APIs for programmers to use. In these APIs, programmers can express the basic algorithm of their applications that involve distance calculations or dot products. Through them, the compiler can easily capture the semantic of the algorithms and generate the TI-optimized code. Figure 8 lists the core

```
_SR_dotProduct(_SR_vector, _SR_vector);
_SR_vectorMatrixProduct(_SR_vector, _SR_matrix);
_SR_mm(_SR_matrix, _SR_matrix);
_SR_defDistance (enum);
_SR_getLowerBound (_SR_pointSet, _SR_pointSet);
_SR_getUpperBound (_SR_pointSet, _SR_pointSet);
_SR_findClosestTargets (int, _SR_pointSet, _SR_pointSet);
_SR_findFarthestTargets (int, _SR_pointSet, _SR_pointSet);
_SR_findTargetsWithin (float, _SR_pointSet, _SR_pointSet);
_SR_findTargetsBeyond (float, _SR_pointSet, _SR_pointSet);
_SR_update (_SR_pointSet, ...);
```

Figure 8. Core APIs for assisting TI-based Strength Reduction.

APIs. The prefix "_SR_" marks the functions and data structures defined for TI-based strength reduction. The first three functions indicate the type of vector operations, the fourth one indicates the type of distance to compute, and the remaining functions indicate the type of relations between the point sets that are of interest.

6. Evaluation

To demonstrate the efficacy of the proposed TI-based strength reduction, we experiment with eight influential algorithms from various domains, including data mining, deep learning, and graph analytics.

We compare the performance of the implementation optimized by our technique with two other versions: the *standard* and the *optimized*. The *standard* versions are the implementations of the eight classic algorithms [6, 9, 14, 18, 21, 28, 31, 42], on which no triangular optimization is applied. The *optimized* versions are attained by applying TOP [15], a latest work which applies TI-based optimizations. It has two limitations. First, it can apply only ETI and only to distance-based algorithms. Second, its application of the optimization is in an ad hoc manner relying on a set of hardcoded thresholds rather than the systematic adaptive approach this work describes. As no prior work has given methods to apply TI-based optimizations to vector product or cosine similarity calculations, the *standard* and *optimized* versions of the three such algorithms are identical. All implementations are in C++, compiled by GCC with "-O3" optimization flag used.

All the versions of an algorithm have the same semantic; they produce the same outputs. Therefore, our discussion focuses on the performance (running time). The performance data are collected on a workstation equipped with Intel i5-4570 CPU and 16G memory. Each performance number comes from the average of five repeated runs. Besides reporting the speedups, we also analyze the impact of our runtime support for dynamic adaption.

6.1 Benchmarks

In the following, we will give a brief introduction to the eight benchmarks. The first five benchmarks (KNN, KNNjoin, KMeans, ICP, NBody) are the benchmarks the TOP work uses. Including them allows a head-to-head direct comparison with the previous work. The other three benchmarks (DC, KDR, RBM) involve dot products, which allow us to assess the extra applicability enabled by the introduced API. All these algorithms play some important roles in their respective domains. The datasets used for evaluation are commonly used in previous works for performance testing. In particular, they are selected to cover a large range of input sizes, dimensions, and various settings (e.g., K for KNNjoin, KNN, and KMeans).

KNNJoin [6] tries to find K points in set T that are closest to every query point in set Q. KNN [21] is similar to KNNJoin except that it tries to find K target points that are closest to a single query point each time. Such difference would affect the kinds of available TI-based optimizations. For example, grouping on the query points is one type of optimization that is not available for KNN, but is beneficial for KNNJoin. We test these two algorithms on three datasets, *Gassensor*, *Kegg* and *MiniBooNe*, obtained from the UCI Machine Learning Repository [2]. The dataset size N ranges from 13K to 130K, and the dimension ranges from 28 to 129. For each dataset, we test it for K = 10, 50, 100.

KMeans[31] tries to group points in a set into K clusters. It runs iteratively, starting with K initial centers and stopping at convergence. In each iteration, it labels every point with the center that is closest to it, and then uses the average location of the points in the same cluster to update the center of the cluster. We tested KMeans on three datasets: *Kegg*, *USCensus* and *Notredame* to cover a large range of dataset sizes and dimensions. The first two datasets are obtained from the UCI Machine Learning Repository [2] and the last one is a commonly used image dataset [39]. The dataset size N ranges from 65K to 2.5Million, and the dimension ranges from 28 to 128. In particular, we test *Kegg* for K = 16, 64, 256; *USCensus* and *Notredame* for K = 64, 256, 1000.

ICP[9] is an algorithm mapping the pixels (points) in a query image to the pixels in a target image. It is an iterative process. In each iteration, it maps each pixel in a query image to a pixel in the target image that is similar to the query pixel, and then transforms the query image in a certain way. We tested ICP on three datasets, *abalone*, *krkopt* and *letter*, obtained from the UCI Machine Learning Repository [2]. The dataset size N ranges from 4K to 28K, and the dimension ranges from 6 to 16.

Nbody [18] simulates the interplay and movements of particles in set Q in a series of time steps. In each step, it computes the distances between every particle and all particles in its neighborhood. From the distance, it then derives the force the particle is subject to, computes its movement accordingly, and updates its position. The algorithm has some variations. The one used in this work defines the neighborhood of a particle as a sphere of a given radius. We tested it on three datasets used in previous work [15]. The dataset size N ranges from 5K to 440K. The dimension is al-

ways three, representing the position of particles in the three dimensional space.

Document clustering (DC) [28] is the application of clustering techniques to textual documents. DC applys the commonly used term weighting strategy TF-IDF, and dimension reduction method, *Non-negative Matrix Factorization (NMF)* [41], to first get a vector representation of the documents. It then applies Kmeans on the cosine similarity of these vectors to do clustering. The experiment uses three datasets, *enron*, *nytimes*, and *pubmed*, all from the UCI Machine Learning Repository [2]. The dataset size N ranges from 40K to 1M and the original dimension ranges from 2.8K to 141K. The reduced dimensions are 20 for the smallest dataset *enron*, 50 for *nytimes* and 200 for *pubmed*. For each dataset, we tested K = 64, 128, 256.

Top-K Document Retrieval (KDR) [42] is a related problem from information retrieval. It aims to produce K documents that are most similar to a query document [37]. Cosine similarity is used in KDR to quantify the difference between documents. We tested on the same datasets as the ones in DC. The top-K related documents are calculated for each document in the dataset. For each dataset, we tested K = 10, 50, 100.

For Binary RBM [38] (described in Section 3.2.2), we tested it on three datasets (binary images being used): the MNIST handwritten digits dataset [30], the small 20-Newsgroup dataset [33] and the transformed MNIST (f-MNIST) dataset in which each pixel flips its value [10]. The number of images in the datasets ranges from 8.5K to 50K. The number of visible units ranges from 100 to 784 while the number of hidden units is set to 500.

As described in Section 5, our compiler-based strength reduction framework could work on programs through either code pattern matching or the assistance of APIs. Among the eight algorithms we tested, KNN, KNNJoin, KMeans, DC, KDR and RBM are directly transformed from the standard implementations in C++ through code pattern matching, while ICP and NBody are rewritten by us using our APIs due to the complexities (e.g., distance computations mixed with updates, complicated function calls with pointers) in their standard implementations.

6.2 Overall Performance

The graph in Figure 9 gives our speedups on each dataset over the *standard* implementations of the algorithms. Compared with the *standard* version, which does not use TI-based optimizations, our technique achieves as much as 134X (NBody) speedups and 46X on average.

The accelerations come primarily from the savings of distance or dot product computations. Although the mount of savings vary, depending on many factors, we observe over 91% computation savings for all the datasets tested on these benchmarks other than RBM. In particular, we notice that the savings are often more prominent for larger input and problem settings (e.g., dataset size, data dimensions, and the number of clusters). Dataset size is the most influential factor across all benchmarks, regarding the fraction of skipped computations. For example, the most substantial speedup for KMeans is obtained on the largest dataset USCensus, which has dataset size N = 2.5Million, data dimensions D = 64, and number of clusters K = 1K.

The overhead of bound computations is always negligible compared to the original computation cost in the standard version without TI optimization. The reason for this is twofold. First, bound computation itself is a scalar operation, while both distance and dot product computation are vector operations. When the data dimension is high, the cost of bound computation is much smaller than that of direct distance and dot production computation. Second, when grouping and early termination is used as described in section 4.3, the total number of bound computations carried out is much smaller than that of distance and dot product computations required in a standard version.

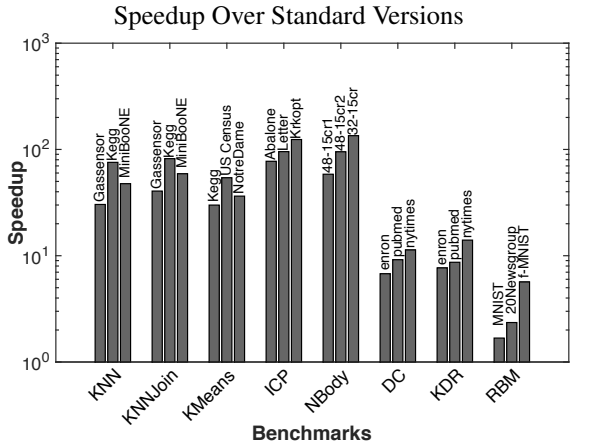
The table at the bottom of Figure 9 reports our speedups (average across all datasets) over the previously *optimized* versions [15]. Compared to the *optimized* versions, our optimization gets 1.14X–1.46X speedups on the distance-based problems, on which, the previous method applies ETI only and does that with some thresholds. On the other programs (DC, KDR, and RBM) that work with cosine similarity or vector dot products, the previous method cannot apply, while our method achieves 3.2X–10.1X speedups thanks to its ATI-based optimizations.

Among these three benchmarks, the computation saving and speedup on RBM are not as large as on the other algorithms, but are still substantial, up to 84% and 4.8X (on f-MNIST). The limited speedup for RBM is a result of three constraints. First, the number of units in the visible layer and hidden layer are limited, much smaller compared to the number of points in other algorithms. Second, the vector dot product results are scaled down through a Sigmoid function, and thus the requirement on the bounds quality is higher. Furthermore, optimization technique such as group-filtering and early termination can not be applied.

For the other two benchmarks DC and KDR, the speedups are more substantial. We find that with ATI we could remove at least 91% and frequently over 94% of the vector product computations on datasets of various dimensions and sizes. These tremendous savings translate into the substantial speedups. The speedups are not as much as the savings of computations because the comparisons with bounds add some overhead and complexities in the control flows.

The leftmost five benchmarks in Figure 9 are distance-related problems. We achieve great speedup over the standard version. The accelerations come primarily from the savings of distance computations enabled by TI optimizations. We found that our method could remove at least 93% and frequently over 99% of the distance computations on the datasets of various dimensions and sizes. As the table

in Figure 9 reports, our framework also outperforms previously highly-optimized versions on testing datasets. The extra speedups come from two aspects. The first is ATI, which expands the applicability of TI-optimizations to dot products, and further improves the quality of the bounds for distance calculations. The second is the guided TI adaptation. With it, the compiler can better select the deployment strategies of TI optimizations that fit each problem and dataset, striking a better cost-benefit tradeoff. We give some detailed analysis of both factors next.



Average Speedup Over the Performance from the Prior Methods

Prog	KNN	KNNjoin	KMeans	ICP	Nbody	DC	KDR	RBM	*geomean*
Speedup	1.35X	1.46X	1.19X	1.17X	1.14X	9.19X	10.13X	3.23X	2.35

Figure 9. The graph shows the speedup over the *standard* version; the table reports the average speedup of our automatic framework compared to the previously *optimized* versions [15].

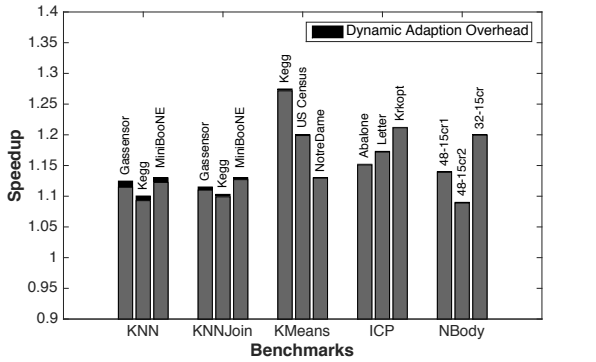


Figure 10. Speedup brought by the guided TI adaptation over using rigid rules [15] for deploying TI-based optimizations. The top black segment on each bar represents the overhead incurred by the runtime sampling and adaptation.

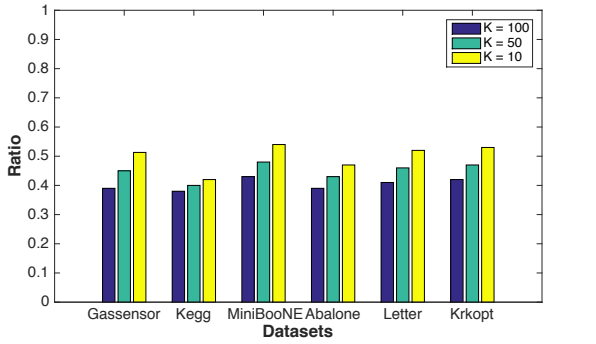


Figure 11. Fraction of extra savings of the distance computations due to the tighter bounds by ATI over those by ETI on KNNJoin.

6.3 Guided TI Adaptation

The guided adaptation in our TI-based strength reduction gives a more systematic way to deploy the optimizations than prior methods. To help isolate the effects of guided TI adaptation, we strip off the usage of ATI from our versions. The algorithms on vector computations are not shown because the ETI-based TOP method cannot apply to them.

Figure 10 reports the benefits brought by the adaptive deployment compared to the previous threshold-based ad-hoc deployment in TOP. The grey bars show the speedups, ranging from 1.09X to 1.27X. The black segment on each bar shows the time overhead incurred by the runtime sampling and adaptation operations, less than 3% in all cases. When the data size gets larger or the problem is iterative, the overhead is smaller as it weights less in those longer runs.

6.4 Tighter Bounds by ATI

ATI generates tighter bounds for distance calculations than ETI does. To give a detailed examination on the benefits, we take KNNJoin as an example and report the fraction of extra savings of distance calculations by ATI in various settings. Recall that KNNJoin is a program trying to find the K nearest points for each query point. Besides the three datasets used in the previous subsections, we add three extra datasets *abalone*, *krkopt* and *letter* to further enrich the datasets. These three datasets are from the UCI Machine Learning Repository [2]; their sizes are 4.1K, 20K, 28K, and dimensions are 8, 16, 6 respectively.

The usage of ATI in KNNJoin is to examine the cases that pass through the ETI checks before conducting distance computations. The examination gets the lower distance bounds through ATI and compares them against the current upperbound of the k nearest neighbors. Distance calculations are done only if the former are smaller than the latter.

Figure 11 reports the fraction of extra savings, defined as the fraction of distance computations that are regarded as necessary to do in ETI but unnecessary to do in ATI thanks to the tighter bounds offered by ATI. In the graph, K is the number of nearest neighbors to find for each query point. We vary its value from 10 to 50 and to 100. Figure 11 shows that more than 39% of the distances can be further avoided by applying ATI. More savings are shown for smaller K values than the larger ones. It is because as the number of nearest neighbors of a point to find decreases, more points

are less likely to be the nearest neighbors and hence more potential for TI-based optimizations. The tighter bounds of ATI turns out to give more benefits. But overall, the savings are substantial in all the tested cases.

7. Related Work

Strength reduction is a classic program optimization technique in compilers. Most prior techniques are about replacing multiplication-like operations with cheaper additions [12]. Finite differencing [35] and some later extensions try to optimize incremental computations hidden in loops. To our best knowledge, this paper is the first that proposes the concept of TI-based strength reduction. By generalizing the technique into a program optimization technique, it enriches the applicable scenarios of strength reduction by compilers.

The main savings from TI-based strength reduction come from the avoidance of unnecessary computations. Removing redundant computations from a program is a classic topic in compiler [12]. Prior efforts have tried to extend the scope of the optimizations [11, 13, 24]. They have all focused on removing common subexpressions or dead code. The distance computations and dot products that TI-based strength reduction helps avoid are not considered as redundant in those methods, because their computing results are all used in the conditional checks in the original program, and they are not the type of repeated computations on the same values that the prior methods address. TI-based optimization is also related to incremental computations [1, 3, 36] or dynamic programming, but it is more flexible in what and how computations can be reused. In incremental computation, the exact result of the computation on a sub-problem is reused, whereas TI optimization reuse previous similar but not exactly the same computation result.

Triangular inequality has been used in the design of many algorithms, including K-Means [16, 17, 20, 26], other data mining and machine learning algorithms [32, 40], graph problems [23], and so on [22]. All these are manual algorithm designs, and exploit only ETI. Framework TOP tries to automate the process, showing even better results than the manual ones [15]. This work was inspired by TOP, but makes some significant extensions in both theory and implementation. First, it explores some deep connections between TI and compilers, and develops the concept of *TI-based strength reduction*. Second, unlike TOP which bases the optimizations on traditional TI only, this work generalizes the theory of TI by developing a new type of TI, named *Angle Triangular Inequality* (ATI), and presents some fundamental properties of ATI and its relations with the traditional TI (e.g., Theorems 1 and 2 and Corollary 3). Third, this work finds out the various factors that influence the cost and benefits of the deployment of ATI-based optimizations, and characterizes the scenarios in which the different types of TI-based strength reductions can work well with group filtering and early stop optimizations. Fourth, this work generalizes the deployment of TI-based optimizations. Instead of replying on APIs only, it exploits the possibility for compilers to automatically transform code to leverage the optimizations through code pattern matching. Moreover, it replaces previous ad hoc thresholds with *guided TI adaptation* to help efficiently determine the appropriate ways to configure the optimizations on the fly. Finally, all the extensions help expands the scope of TI-based optimizations from distance-related problems to problems related with distances, vector dot products, and cosine similarities, and demonstrates the significantly enhanced applicability and effectiveness with a variety of applications.

Recent years witnessed some development of approximation-based program optimizations [8, 34]. TI-based strength reduction keeps the semantic of the original program, uses no approximations, and hence introduces no errors into the computation results. Combining TI with approximation-based optimizations could worth future studies.

8. Conclusion

This paper has proposed TI-based strength reduction. It is inspired by the previous work on applying triangular inequality in algorithmic designs. It generalizes the idea into a compiler optimization technique by making three-fold explorations: building up the theoretic foundation via the development of the ATI-related theorems, revealing the properties of ATI and proposing guided TI adaptation to offer a systematic solution to the difficulties in determining the effective ways to deploy TI-based optimizations, and then integrating the techniques into an open-source compiler through a dual-mode design. Experiments validate the effectiveness of this new technique, showing as much as 134X and 46X on average speedups over the original implementation, outperforming the state of the art optimizations by 2.35X on average. It expands the applicability of TI-optimizations from distances to vector computations and cosine similarity.

Acknowledgment

We thank the feedback from the anonymous reviewers and the help from our shepherd Mayur Naik. This material is based upon work supported by DOE Early Career Award (DE-SC0013700), the National Science Foundation (NSF) under Grant No. 1455404, 1455733 (CAREER), and 1525609. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DOE or NSF.

References

[1] B. Aaron, D. E. Tamir, N. D. Rishe, and A. Kandel. Dynamic incremental k-means clustering. In *Computational Science and Computational Intelligence (CSCI), 2014 International Conference on*, volume 1, pages 308–313. IEEE, 2014.

[2] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[3] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin. Incoop: Mapreduce for incremental computations. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 7. ACM, 2011.

[4] V. Bijalwan, V. Kumar, P. Kumari, and J. Pascual. Knn based machine learning approach for text and document mining. *International Journal of Database Theory and Application*, 7(1):61–70, 2014.

[5] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[6] C. Böhm and F. Krebs. The k-nearest neighbour join: Turbo charging the kdd process. *Knowledge and Information Systems, Springer*, 6(6):728–749, 2004.

[7] D. Cai, X. He, J. Han, and T. S. Huang. Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1548–1560, 2011.

[8] M. Carbin, S. Misailovic, and M. C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *ACM SIGPLAN Notices*, volume 48, pages 33–52. ACM, 2013.

[9] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, IEEE*, pages 2724–2729, 1991.

[10] K. Cho, T. Raiko, and A. Ilin. Enhanced gradient and adaptive learning rate for training restricted boltzmann machines. In *Proceedings of the 28th International Conference Proceedings of the 28 th International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

[11] K. Cooper, J. Eckhardt, and K. Kennedy. Redundancy elimination revisited. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 12–21. ACM, 2008.

[12] K. Cooper and L. Torczon. *Engineering a Compiler*. Morgan Kaufmann, 2003.

[13] S. J. Deitz, B. L. Chamberlain, and L. Snyder. Eliminating redundancies in sum-of-product array computations. In *Proceedings of the 15th international conference on Supercomputing*, pages 65–77. ACM, 2001.

[14] E. W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische mathematik*, volume 1, pages 269–271, 1959.

[15] Y. Ding, X. Shen, M. Musuvathi, and T. Mytkowicz. Top: A framework for enabling algorithmic optimizations for distance-related problems. In *Proceedings of the 41st International Conference on Very Large Data Bases*, 2015.

[16] Y. Ding, X. Shen, M. Musuvathi, and T. Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In *ICML*, 2015.

[17] J. Drake and G. Hamerly. Accelerated k-means with adaptive distance bounds. In *5th NIPS Workshop on Optimization for Machine Learning*, 2012.

[18] V. Eijkhout. *Introduction to High Performance Scientific Computing*. Lulu. com, 2010.

[19] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, volume 3, pages 147–153, 2003.

[20] C. Elkan. Using the triangle inequality to accelerate k-means. In *ICML*, volume 3, pages 147–153, 2003.

[21] E. Fix and J. L. Hodges Jr. Discriminatory analysis-nonparametric discrimination: consistency properties. In *DTIC Document*, 1951.

[22] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM*, pages 156–165, 2005.

[23] M. Greenspan and G. Godin. A nearest neighbor method for efficient ICP. In *3-D Digital Imaging and Modeling, IEEE*, pages 161–168, 2001.

[24] G. Gupta and S. V. Rajopadhye. Simplifying reductions. In *POPL*, volume 6, pages 30–41, 2006.

[25] G. Hamerly. Making k-means even faster. In *SDM, SIAM*, pages 130–140, 2010.

[26] G. Hamerly. Making k-means even faster. In *SDM*, pages 130–140. SIAM, 2010.

[27] G. Hinton., S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[28] A. Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

[29] J. Z. Lai, Y.-C. Liaw, and J. Liu. Fast k-nearest-neighbor search based on projection and triangular inequality. *Pattern Recognition, Elsevier*, 40(2):351–359, 2007.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[31] S. Lloyd. Least squares quantization in pcm. In *Information Theory, IEEE*, volume 28,2, pages 129–137, 1982.

[32] W. Lu, Y. Shen, S. Chen, and B. C. Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proceedings of the VLDB Endowment*, 5(10):1016–1027, 2012.

[33] B. M. Marlin, K. Swersky, B. Chen, and N. Freitas. Inductive principles for restricted boltzmann machine learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 509–516, Chia Laguna Resort, Sardinia, Italy, 2010.

[34] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard. Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels. In *ACM SIGPLAN Notices*, volume 49, pages 309–328. ACM, 2014.

[35] R. Paige and S. Koenig. Finite differencing of computable expressions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):402–454, 1982.

[36] K. Ravichandran, R. Cledat, and S. Pande. Collaborative threads: exposing and leveraging dynamic thread state for efficient computation. In *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, pages 4–4. USENIX Association, 2010.

[37] H. Schütze. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference*, 2008.

[38] T. Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1064–1071, New York, NY, USA, 2008. ACM.

[39] J. Wang, J. Wang, Q. Ke, G. Zeng, and S. Li. Fast approximate k-means via cluster closures. In *Computer Vision and Pattern Recognition (CVPR), IEEE*, pages 3037–3044, 2012.

[40] X. Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. In *Neural Networks (IJCNN), IEEE*, pages 1293–1299, 2011.

[41] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273. ACM, 2003.

[42] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49. ACM, 1999.