

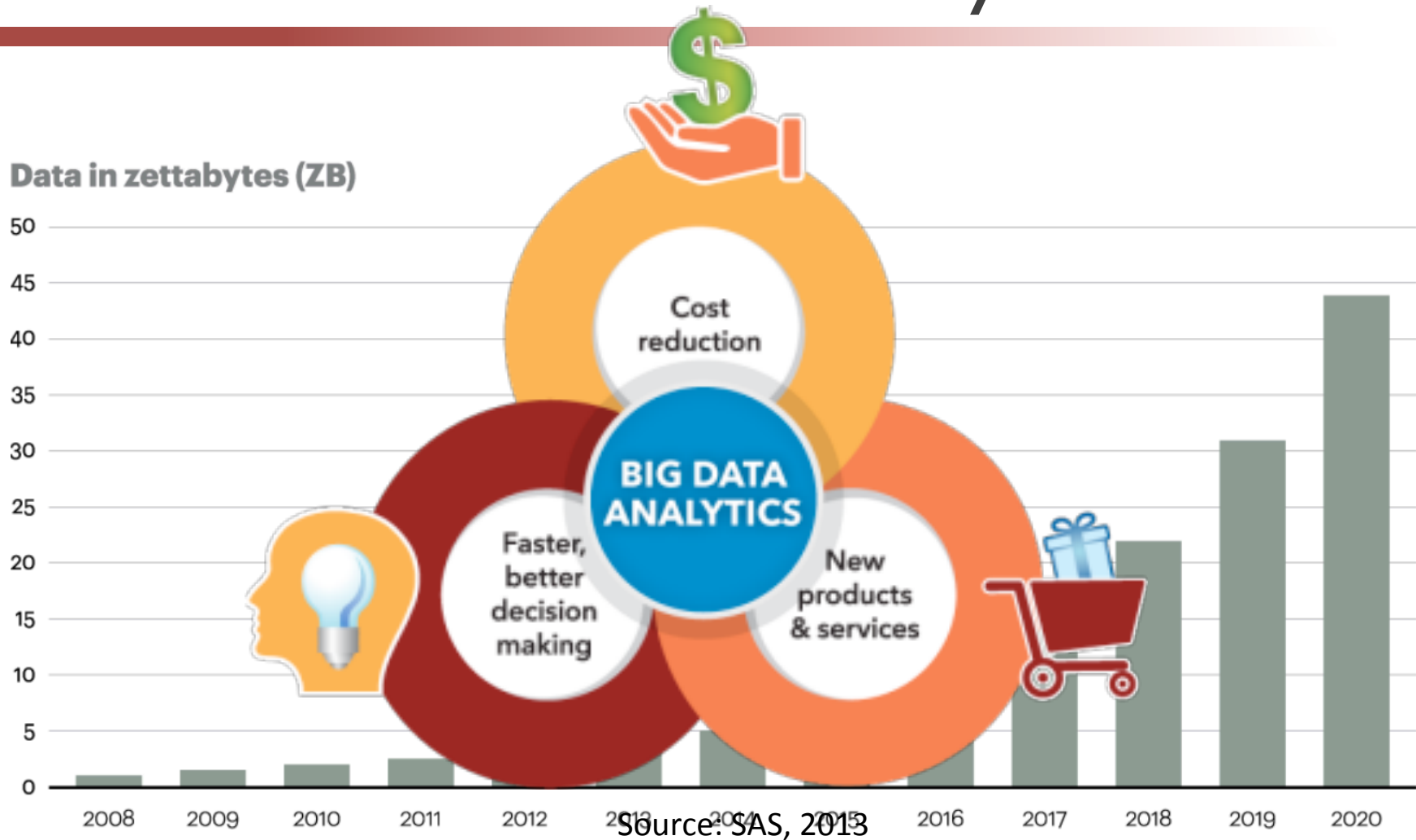


High-level Program Optimization for Data Analytics

Yufei Ding

North Carolina State University

Motivation: Faster Data Analytics



Source: oracle, 2012

Role of My Research



Compiler Technology

- *automatic*, but mostly focus on *instruction-level* inefficiency in program *implementations*.



**Automatic
High-Level
Program Optimization**



(Big) Data Analytics + Other Data-intensive Applications

- *high-level* transformations, which is often more *effective*, but requires a huge amount of *manual* efforts.

My Research



High-level Program Optimization:

- Implementation → **Algorithm**; Instruction → **Formula**

Algorithmic Optimization for Distance-Related Problems

[ICML'15, VLDB'15, ICDE'17, PLDI'17]

Autotuning Algorithmic Choice for Input Sensitivity

[PLDI'15]

Generalizing Loop Redundancy Elimination at a Formula Level *[OOPSLA'17]*

Examining Compilation Scheduling of JIT-Based Runtime System *[ASPLOS'14]*

Parallel Stochastic Gradient Descent (SGD) with Sound Combiners *[applied for patent]*

Focus of this talk

Automatic **Algorithmic** Optimization for **Distance-Related** Problems

magnitudes of speedups.

VLDB'15, ICML'15, ICDE'2017, PLDI'17

Distance-related Problems

- These algorithms are widely used.

Problems	Domain
KMeans	Data Mining (Among Top 10 Most Popular DM Algorithms)
KNN (K Nearest Neighbor)	
KNN Join	
ICP(Iterative Search Problem)	Image Processing
P2P (Point-to-Point Shortest Path)	Graphics
Nbody	Computational Physics

- Distance computations are the performance bottleneck.

KMeans

KNN

P2P: (Point-to-Point Shortest Path)

ICML'2015
Accelerated k -means with adaptive distance bounds

NIPS'2012
Accelerated k -means with adaptive distance bounds

ICML'2003
Using the Triangle Inequality to Accelerate k -Means

Charles Elkan
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093-0114

ELKAN@CS.UCSD.EDU

Abstract
The k -means algorithm is by far the most widely used method for discovering clusters in data. We show how to accelerate it dramatically, while still always computing exactly the same result as the standard algorithm. The accelerated algorithm avoids unnecessary distance calculations by applying the triangle inequality in two different ways, and by keeping track of lower and upper bounds for distances between points and centers. Experiments show that the new algorithm

this center. Conversely, if a point is much closer to one center than to any other, calculating exact distances is not necessary to know that the point should be assigned to the first center. We show below how to make these intuitions concrete.

We want the accelerated k -means algorithm to be usable wherever the standard algorithm is used. Therefore, we need the accelerated algorithm to satisfy three properties. First, it should be able to start with any initial centers, so that all existing initialization methods can continue to be used. Second, given the same initial centers, it should al-

VLDB'12
Optimizing All-Nearest-Neighbor Search with
Trigonometric Pruning

VisionInterface'10
Nearest Neighbour Methods

IJCNN'11
Proceedings of International Joint Conference on Neural Networks, San Jose, CA, USA, 2011

A Fast Exact k -Nearest Neighbors Algorithm for High Dimensional Search Using k -Means Clustering and Triangle Inequality

Xueyi Wang

Abstract—The k -nearest neighbors (k -NN) algorithm is a widely used machine learning method that finds nearest neighbors of a test object in a feature space. We present a new exact k -NN algorithm called AMANN (k -Means for k -Nearest Neighbors) that uses the k -means clustering and the triangle inequality to accelerate the searching for nearest neighbors in a high dimensional space. The AMANN algorithm has two stages. In the building stage, instead of using complex tree structures such as metric trees, k -trees, or ball-trees, AMANN uses a simple k -means clustering method to preprocess the training dataset. In the searching stage, given a query object, AMANN finds nearest training objects starting from the nearest cluster to the query object and uses the triangle inequality to reduce the distance calculations. Experiments show that the performance of AMANN is surprisingly good compared to the traditional k -NN algorithm and tree-based k -NN algorithms such as k -trees and ball-trees. On a collection of 20 datasets with up to 10^7 records and 10^6 dimensions, AMANN shows a 2- to 80-fold reduction of distance calculations and a 2- to 60-fold speedup over the traditional k -NN algorithm for 16 datasets. Furthermore, AMANN performs significant better than a k -tree based k -NN algorithm for most datasets. The results show that AMANN is effective for searching nearest neighbors in high dimensional spaces.

[11] have been proposed to efficiently reduce the distance calculations and find exact nearest neighbors in higher dimensions. These methods iteratively divide training objects and build tree structures using criteria such as absolute coordinates and relative distances, so that a query object needs to check distances with only a limited number of training objects instead of the whole dataset. One problem for these methods is that when the dimensionality of a dataset is high, most of the training objects in the data structures will end up being evaluated and the searching efficiency is no better to or even worse than the traditional k -NN algorithm [10] [18], especially for large k values.

Due to the difficulty of accelerating the k -NN algorithm in high dimensional space, some methods have focused on finding approximate answers. For example, the hashing method from [9] and the priority queue based method from [3] achieved a speedup of several fold over the traditional k -NN by outputting k neighbors within $(1+\epsilon)$ of the true nearest neighbor distances. Hart [13] and Wilson [23] used techniques called condensing and editing to reduce objects from the dataset and accelerate the searching for nearest neighbors.

In this paper, we present a new algorithm called AMANN (k -Means for k -Nearest Neighbors) that efficiently search-

SIAM'05
Reach-based Routing: A New Algorithm Optimized for
Shortest Path

ALENEX'04
Computing the Shortest Path in a Graph with
Euclidean Weights

Andrew V. Goldberg
Chris Harrison

Abstract
We propose shortest path algorithms that use A^* search in combination with a new graph-theoretic lower-bounding technique based on landmarks and the triangle inequality. Our algorithms compute optimal shortest paths and work on any directed graph. We give experimental results showing that the most efficient of our new algorithms outperforms previous algorithms, in particular A^* search with Euclidean bounds, by a wide margin on road networks and on some synthetic problem families.

1 Introduction
The shortest path problem is a fundamental problem with numerous applications. In this paper we study one of the most common variants of the problem, where the goal is to find a point-to-point shortest path in a weighted, directed graph. We refer to this problem as the *P2P problem*. We assume that for the

processing space. The best bound in this context (see [10]) is superlinear in the output path size unless the path is very long. Preprocessing using geometric information and hierarchical decomposition is discussed in [19, 28, 34]. Other related work includes algorithms for the single-source shortest path problem, such as [1, 4, 6, 7, 13, 14, 16, 17, 18, 22, 25, 32, 35], and algorithms for approximate shortest paths that use preprocessing [3, 23, 33].

Usually one can solve the P2P problem while searching only a small portion of the graph; the algorithm's running time then depends only on the number of visited vertices. This motivates an *output-sensitive* complexity measure that we adopt. We measure algorithm performance as a function of the number of vertices on the output path. Note that this measure has the additional benefit of being machine-independent.

In Artificial Intelligence settings, one often needs to find a solution in a search space. The classical A^*

How to build a automatic framework to save all these manual efforts?

Challenges

- What are the beneficial and legal higher-level transformations that lead to better algorithms?
- Can we have an abstraction to unify various problems in different domains?
 - Then we should be able to turn the algorithmic optimization into compiler-based transformations.

Could have saved many years' of manual efforts!

Case Study on KMeans

Yinyang KMeans:
A Drop-In Replacement of the Classic
KMeans with Consistent Speedup

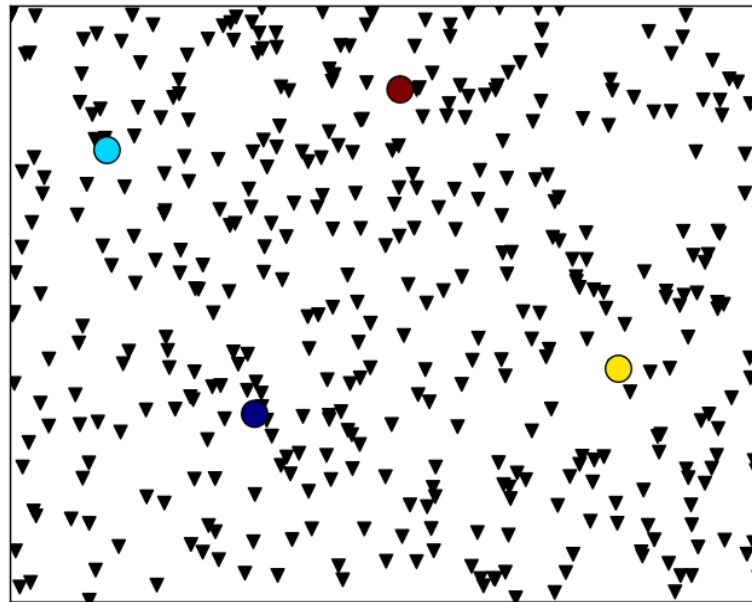
ICML'2015

*Collaborated w/ MSR
(Madan Musuvathi's group)*

Background: KMeans

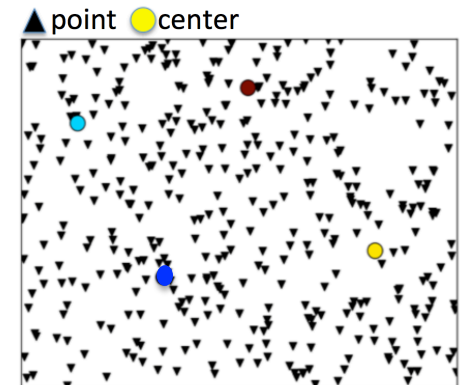
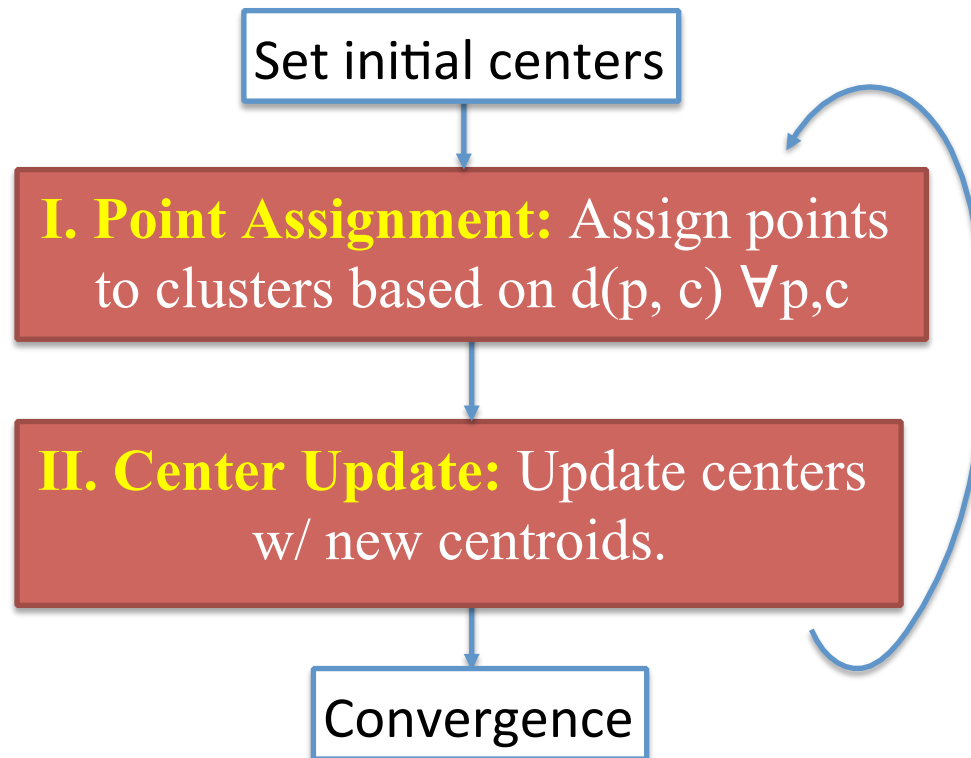
- Usage: group N points (in D dimensions) into K clusters.
- Demo with $N = 600$, $D = 2$, $K = 4$

▲ point ● center



Background: KMeans

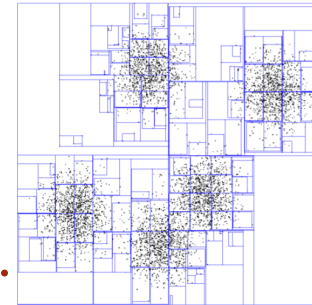
- Usage: group N points (in d dimensions) into K clusters.
- Standard KMeans [by Lloyd in 1957]:



Step I is the performance bottleneck: $N * K$ distances calc. per iteration.

Prior Works

- Grouping:
 - e.g., K-D Tree *[Kanungo et al., 2002]*.
 - **Overhead grows exponentially with dimension.**
- Incremental Computing:
 - e.g., Triangle inequality *[Elkan, 2003; Hamerly, 2010; Drake & Hamerly, 2012]*.
 - **Large Memory Overhead.**
 - **Slowdowns for medium dim, large K & N.**
- Approximation *[Wang et al., 2012]*
 - **Unable to inherit the level of trust.**



Standard KMeans by Lloyd remains
the dominant choice in practice!

Yinyang KMeans

Grouping + Incremental Computing

- On average, **9.36X faster** than classic Means.
- **No slowdown** regardless of N, K, d.
- Guarantee to produce the **same result** as Standard KMeans.



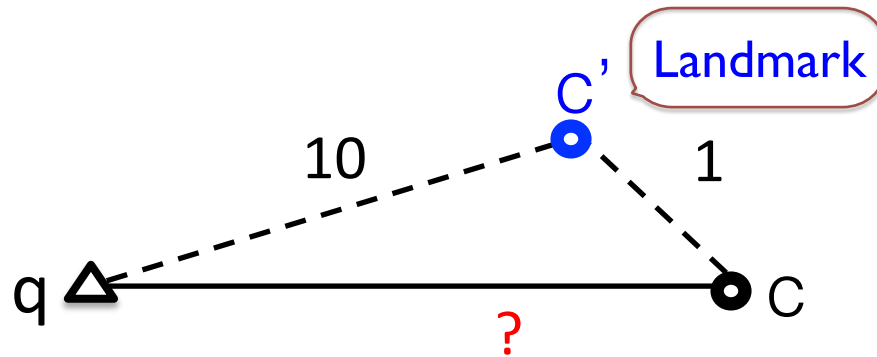
A harmony w/ contrary forces

Yin: upper bound
V.S.

Yang: lower bound
(these bounds comprises the
filters for distance computations)

Triangular Inequality (TI)

- The fundamental tool for getting bounds:



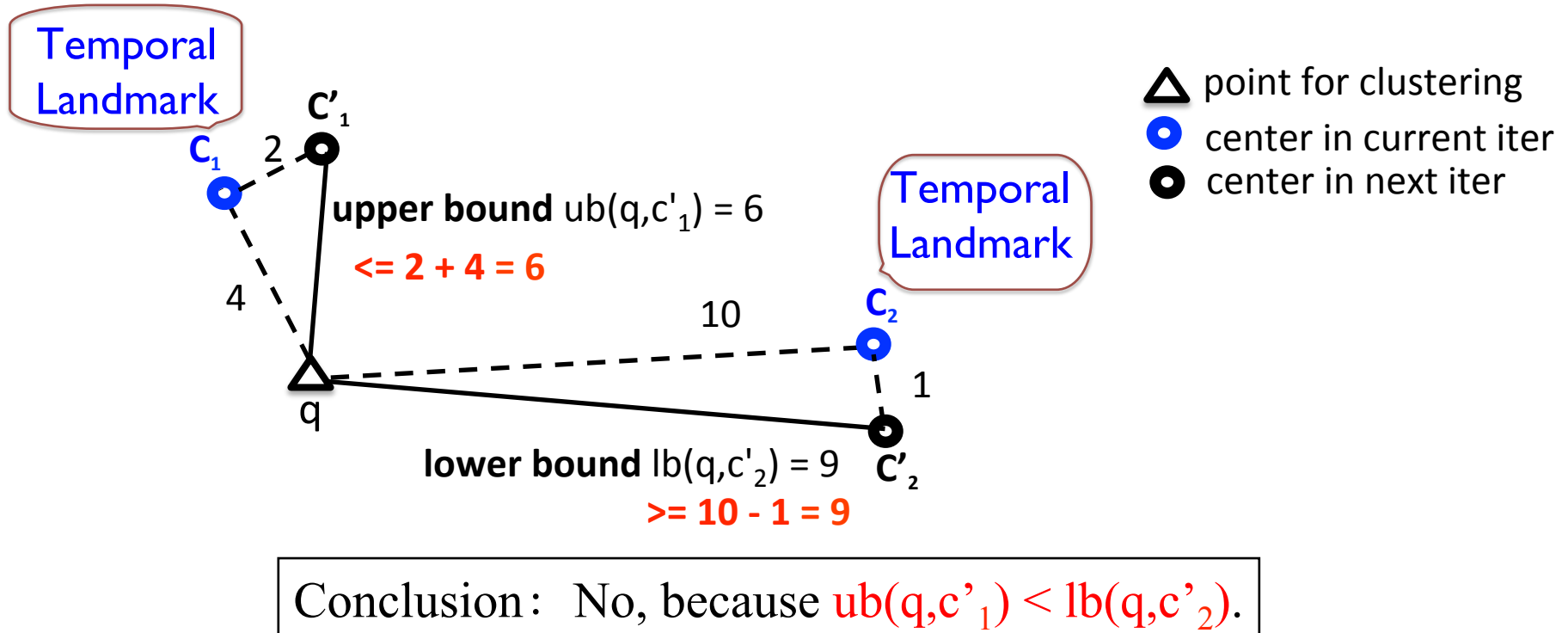
$$\text{TI: } \underline{|d(q, c') - d(c', c)|} \leq d(q, c) \leq \underline{d(q, c') + d(c', c)}$$

$$\text{Lower bound:} \\ \text{lb}(q, c) = |10 - 1| = 9$$

$$\text{Upper bound:} \\ \text{ub}(q, c) = 10 + 1 = 11$$

How are bounds used?

Example: Will q switch its assignment from c'_1 to c'_2 in the next iteration? (q is currently assigned to c_1 .)

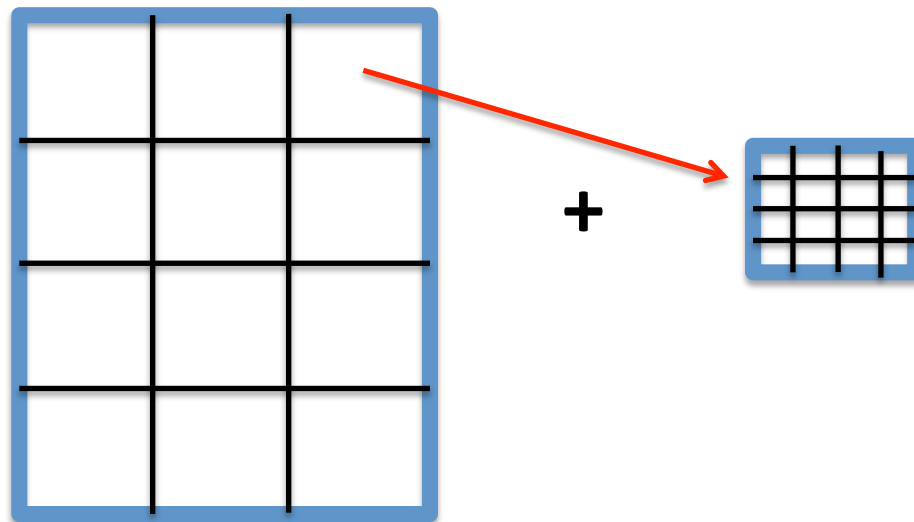


Design of Yinyang Kmeans

- Innovative way of using upper and lower bounds.
 - Joint of filters: Group(Global) filter + Local filter.

Group (Global) filter

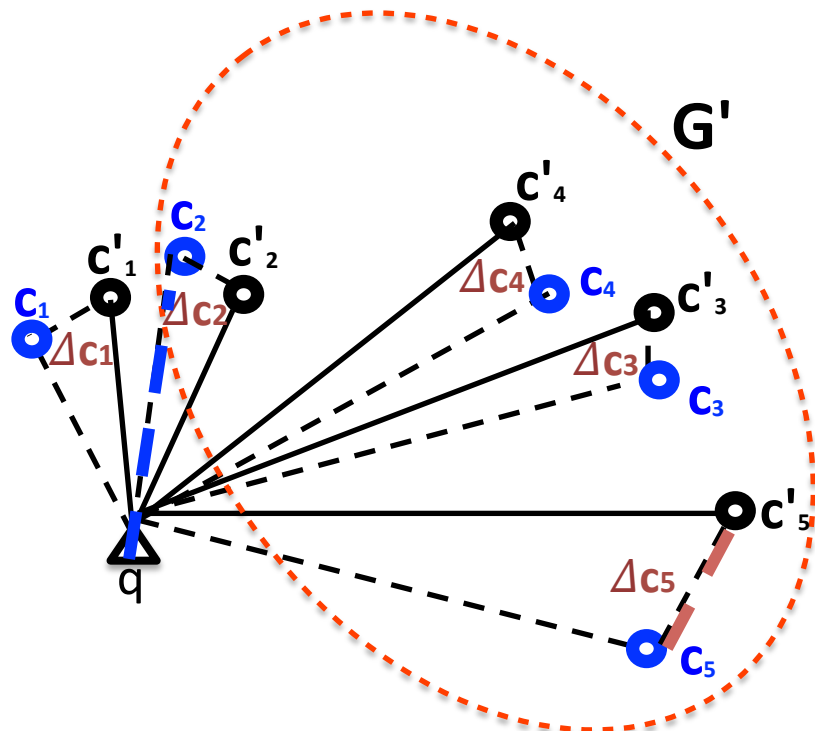
Local filter



Global Filtering

One check

filtering rule: if $ub(q, c'_i) \leq lb(q, G')$, then q **will not change** its assignment.



How to compute these bounds?

$$ub(q, c'_i) = ub(q, c_i) + \Delta c_i$$

$$lb(q, G') = \underline{lb(q, G)} - \underline{\max(\Delta c_i)}, \forall c_i$$

Limiting factors of **1** lower bound:
Benefits: over 60% redundant distance

1. $\underline{lb(q, G)}$: closest center in G' computation can be removed (e.g., c_2).
2. $\underline{\max(\Delta c_i)}$: biggest drifter (how far a center moved across iteration) . (e.g., Δc_5).

current iter: $G' = \{C'_1, C'_2, \dots, C'_k\} - C'_1$

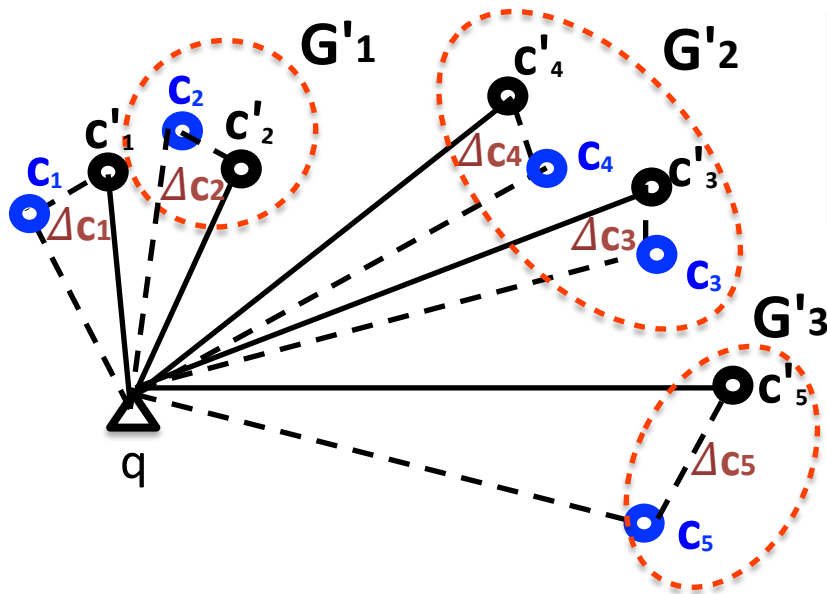
next iter: $G = \{C_1, C_2, \dots, C_k\} - C_1$

(q is currently assigned to C_1)

Group Filtering

m checks

Filtering Rule: if $ub(q, c'_i) \leq lb(q, G'_i)$, then q will not change its assignment to any center in G'_i .



Divide centers into m groups:
 $\{G_1, G_2, \dots, G_m\}$

How to compute these bounds?

$$ub(q, c_1) = ub(q, c'_1) + \Delta c_1$$

$$lb(q, G_i) \leq lb(q, G'_i) - \max(\Delta(c)), \forall c \in G_i$$

Benefits of m lower bounds:

1. $lb(q, G_i)$: local closest center in G_i .
2. $\max(\Delta(c))$: local biggest drifter.

Over 80% redundant distance computations can be removed

Group Filtering

- Overhead Analysis (**m** groups):

$$\begin{aligned} \text{ub}(q, c'_1) &= \text{ub}(q, c_1) + \Delta c_1 \\ \text{lb}(q, G'_i) &\leq \text{lb}(q, G_i) - \max(\Delta c), \forall c \in G_i \end{aligned}$$

Time Cost: K distances (for center drifts) + $N \cdot (m + 1)$ bounds
lightweight compared to standard $N \cdot K$ distances.

Space Cost: $N \cdot (m + 1)$ for maintaining m lower bounds per point.
comparable to $N \cdot D$ for storing N points in D dimension.

Group Filtering

- When/How to group centers?

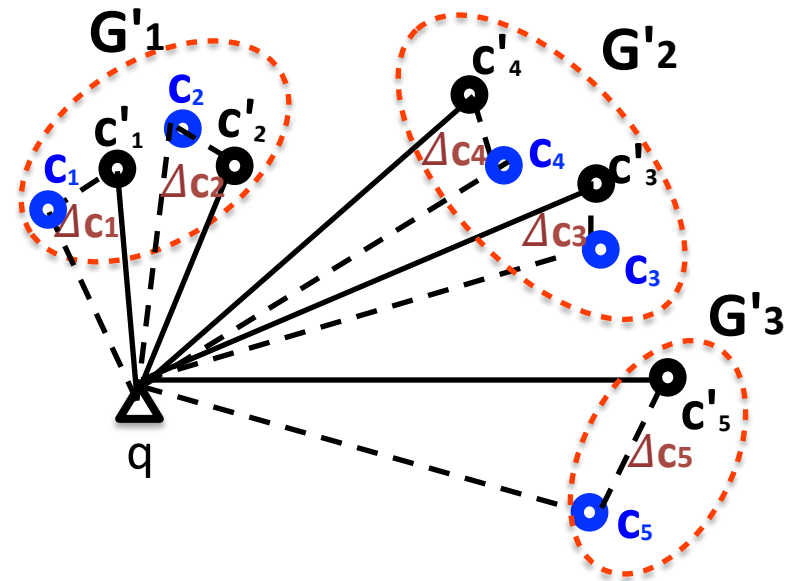
one time grouping
over initial centers
through 5-iter Kmeans.

- How many groups?



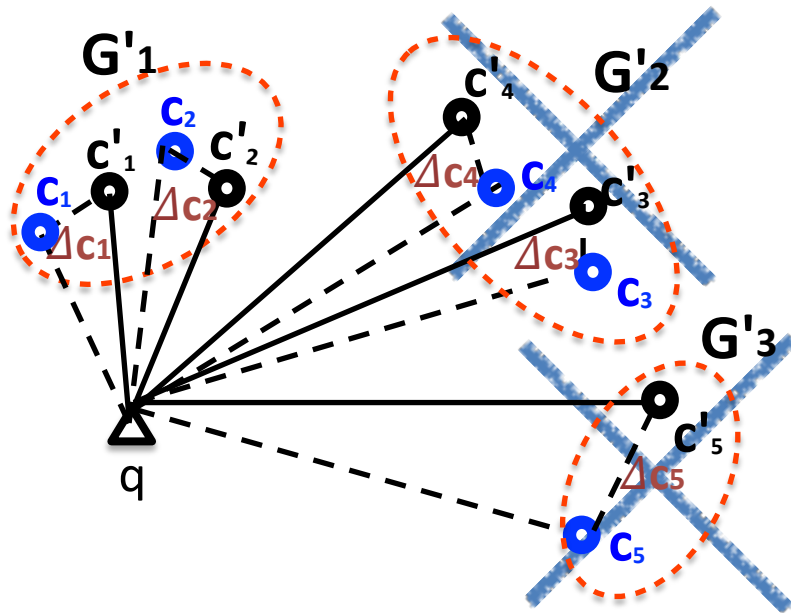
A space-conscious elastic design:

$$m = \begin{cases} K/10 & \text{if space allows} \\ \text{max value} & \text{otherwise} \end{cases}$$



Grouping centers into m
groups: $\{G_1, G_2, \dots, G_m\}$

Local Filtering



Filtering rule: for each center in the remaining group, if $\text{Min}(q, G'i) \leq \text{lb}(q, Gi) - \Delta c_j$, then q will not change its assignment to c_j .

No extra memory cost!

Efficiency:

Over 90% redundant distance computations can be removed.

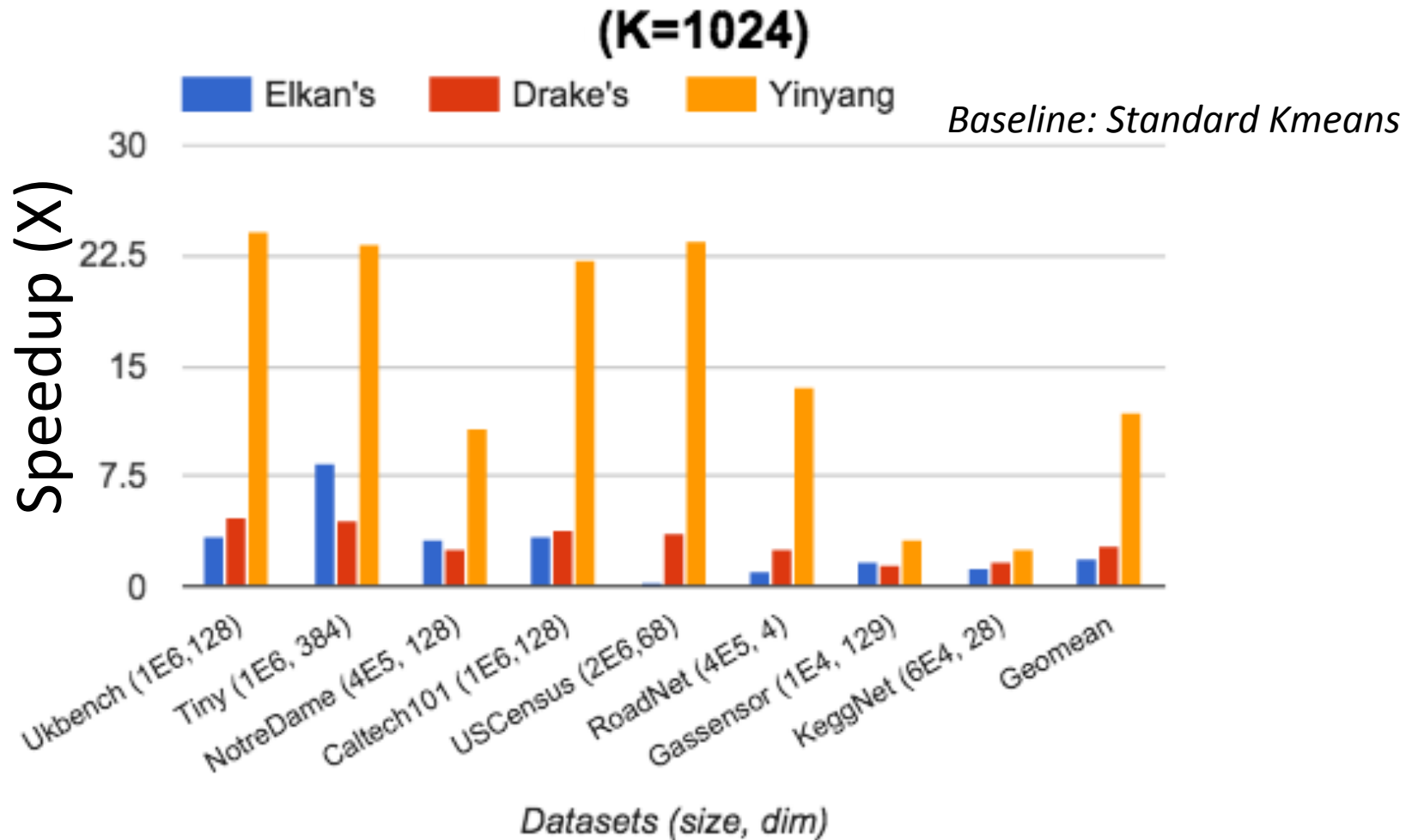
Evaluation

- Compared to three other methods:
 - Standard (Lloyd's) K-Means
 - Elkan's K-Means [2003]
 - Drake's K-Means [2012]
- Input: real-world data sets (with different N, K, Dim)
 - 4 from UCI machine learning repository [Bache Lichman, 2013]
 - 4 other commonly used image data sets [Wang et al., 2012].
- Implemented in GraphLab (a map-reduce framework)
 - <http://research.csc.ncsu.edu/nc-caps/yykmeans.tar.bz2>
- Two machines
 - 16GB memory, 8-core i7-3770K processor
 - 4GB memory, 4-core Core2 CPU

Evaluation

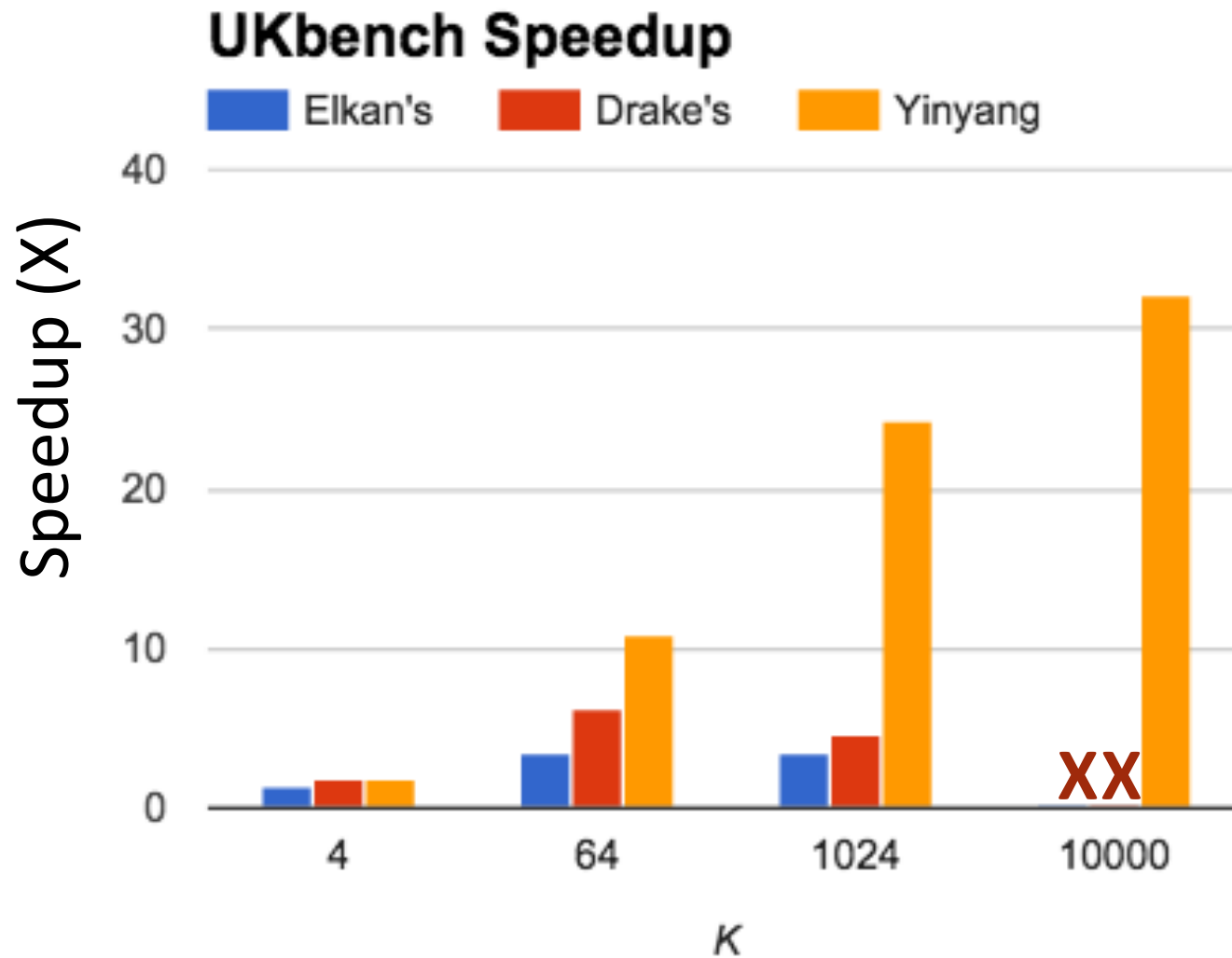
(16GB, 8-core)

Clustering results are the same as those of the standard Kmeans.



Evaluation

Baseline: Classic K-means
(16GB, 8-core)





HPCC / ... / Archive

Implement the Yinyang K-Means Clustering Algorithm

Created and last modified by Lorraine Chapman on Mar 22, 2016

This project is available as an internship opportunity with HPCC Systems this summer.

Towards Yinyang K-means on GPU



by **Vadim Markovtsev** 26 July 2016

The codez: [GitHub](#).

Y **Hacker News** new | comments |

▲ Yinyang K-Means: A Replacement
45 points by jcr 549 days ago | hide | past | web

Y **Hacker News** new | comments | show | ask | jobs | submit

▲ Towards Yinyang K-means on GPU (sourced.tech)
61 points by tanoku 170 days ago | hide | past | web | 14 comments | favorite



Olivier Grisel

@ogrisel

Yinyang K-Means
faster than Stan



datatonic

@teambdatonic

Yinyang K-Means
the Classic K-M



Barney Pell

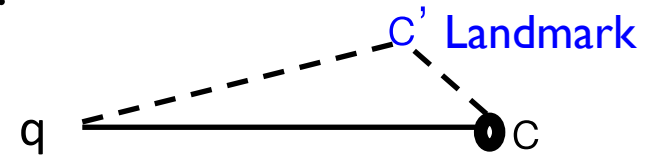
@barneyp

Yinyang K-Means: A Drop-In Replace
the Classic K-Means with Consistent



Algorithmic Optimization Design

- Triangle Inequality Optimization (TOP).



- **landmark definition.**

“temporal landmarks” for iterative problems like KMeans.

- **group filtering.**

of **groups** to strike a good tradeoff between space cost and redundant distance computation elimination.

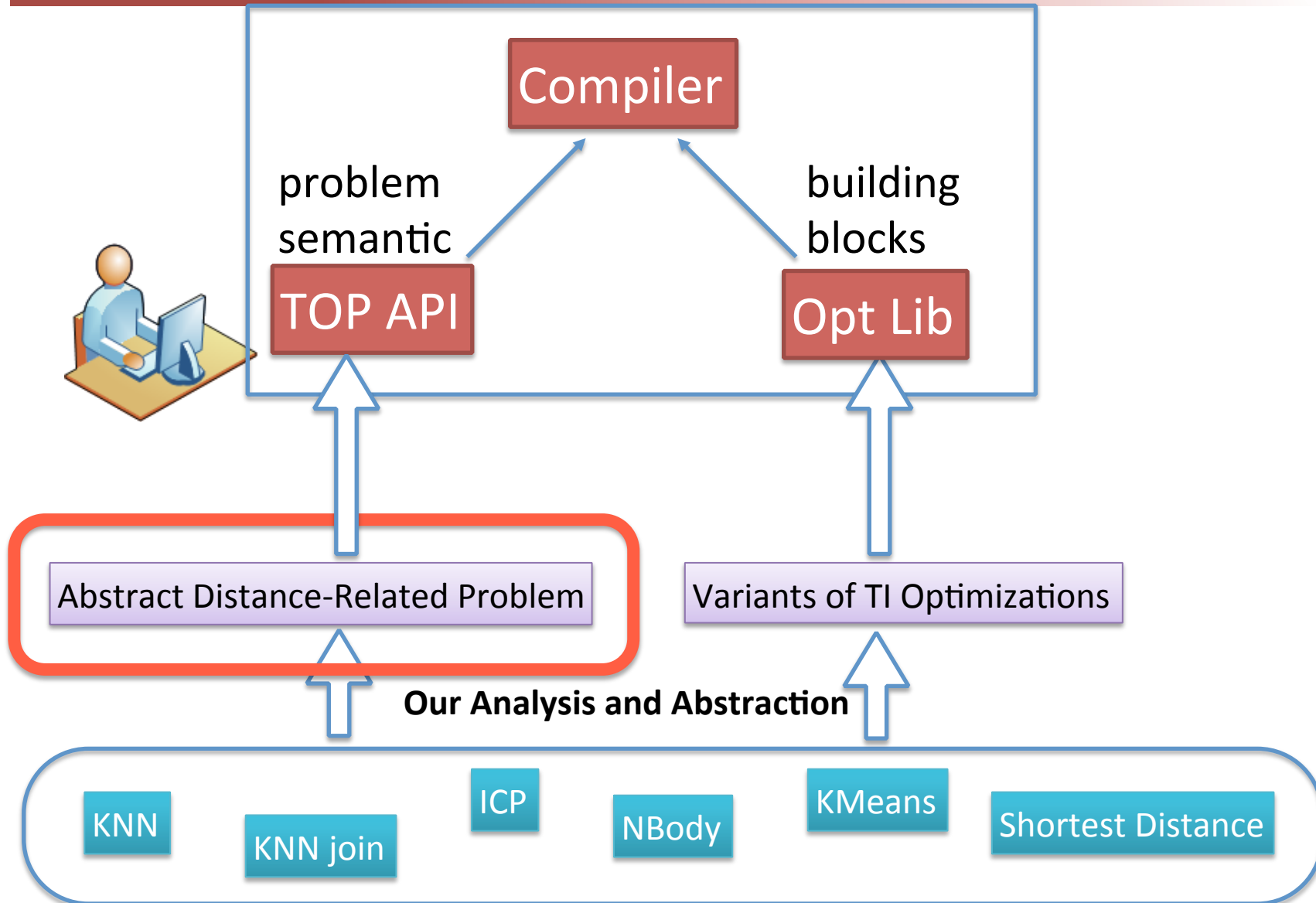
Automatic Framework

TOP: Enabling Algorithmic Optimizations for Distance-related Problems

VLDB'2015

*Collaborated w/ MSR
(Madan Musuvathi's group)*

TOP Framework

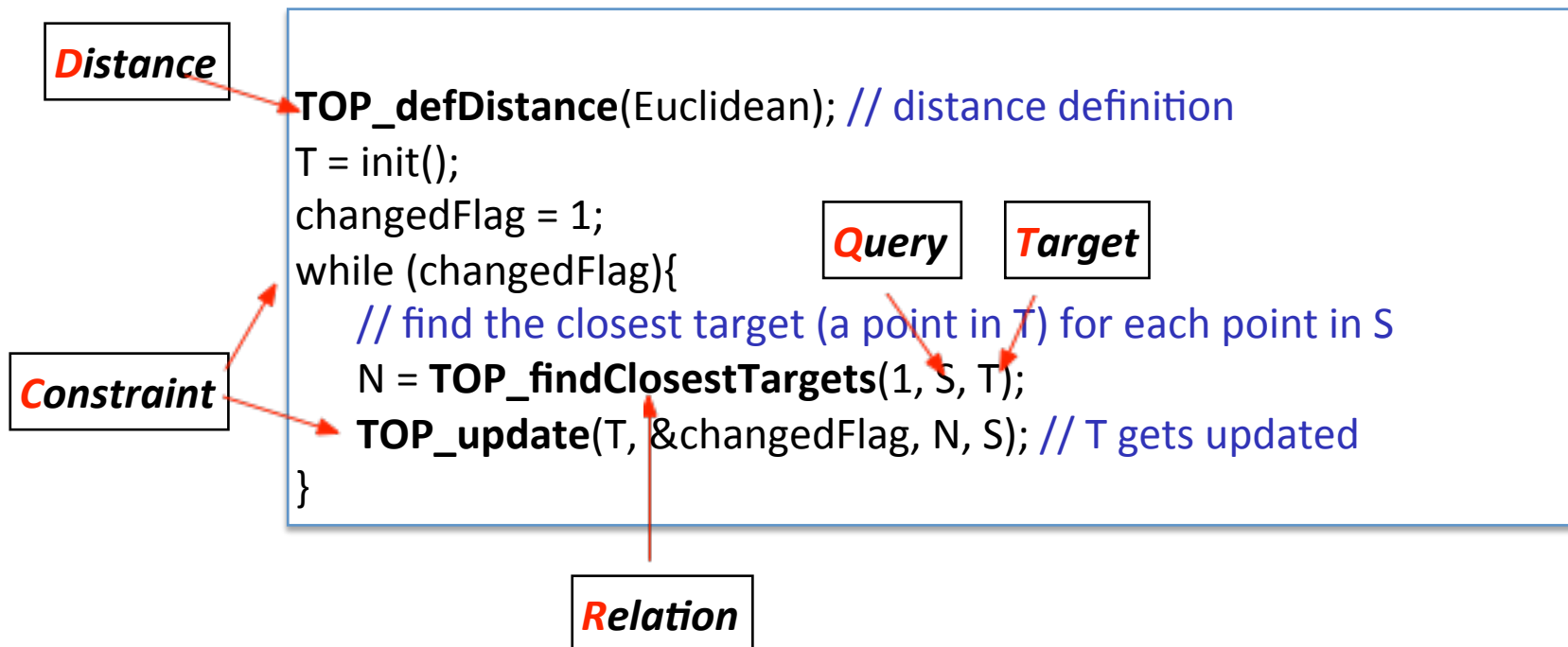


Abstraction for Distance-related Problem

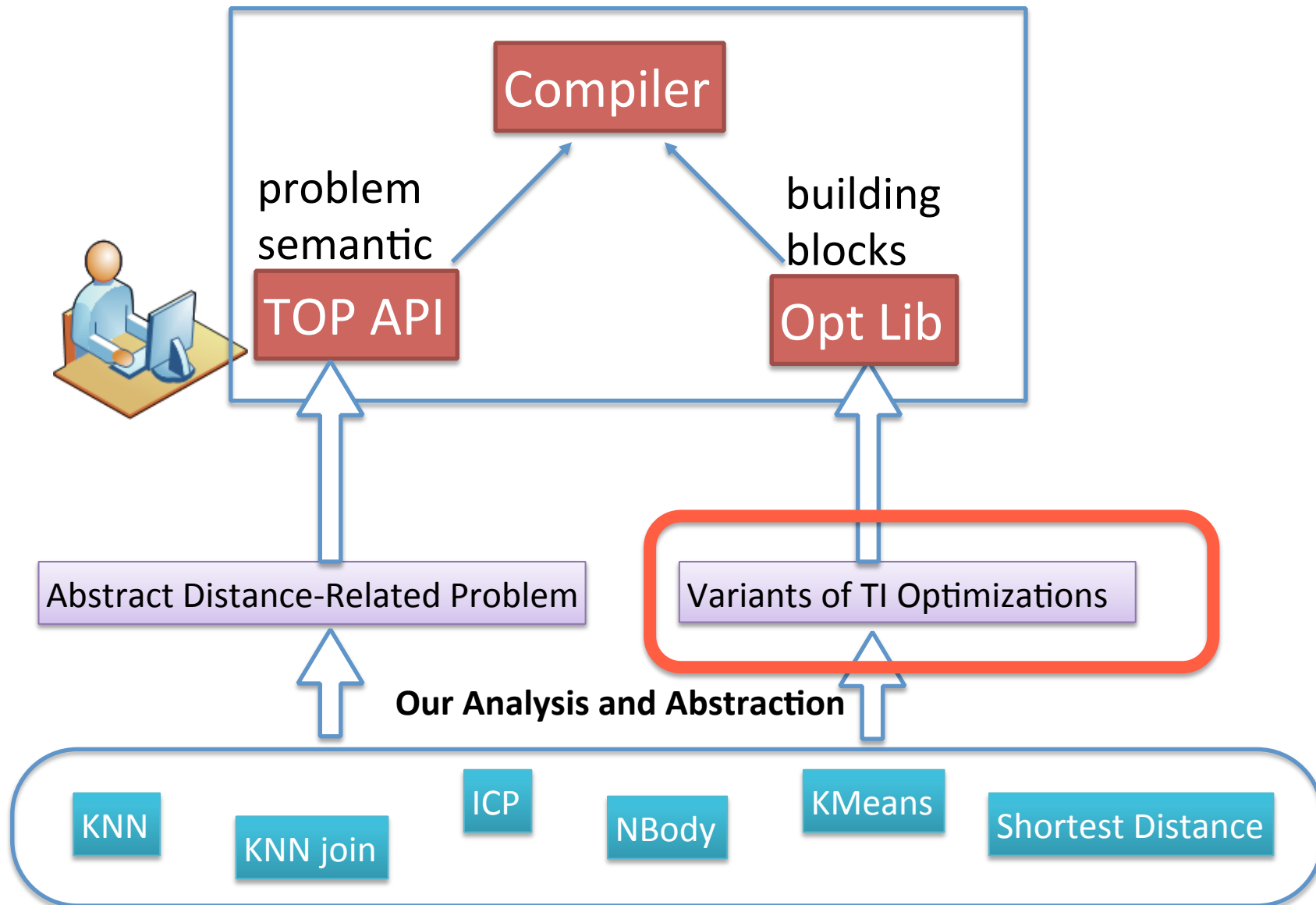
- A 5-element Tuple $\langle Q, T, D, R, C \rangle$
 - finding some kind of **R**elations between two sets of points, a **Q**uery set and a **T**arget set, based on certain type of **D**istance and under some update **C**onstraints.

<i>Problems</i>	<i>Query</i>	<i>Target</i>	<i>Distance</i>	<i>Relation</i>	<i>Constraints</i>
KMeans	Points	Centers	Euclidean	Top 1 Closest	Iterative update to T

KMeans Written with Our APIs

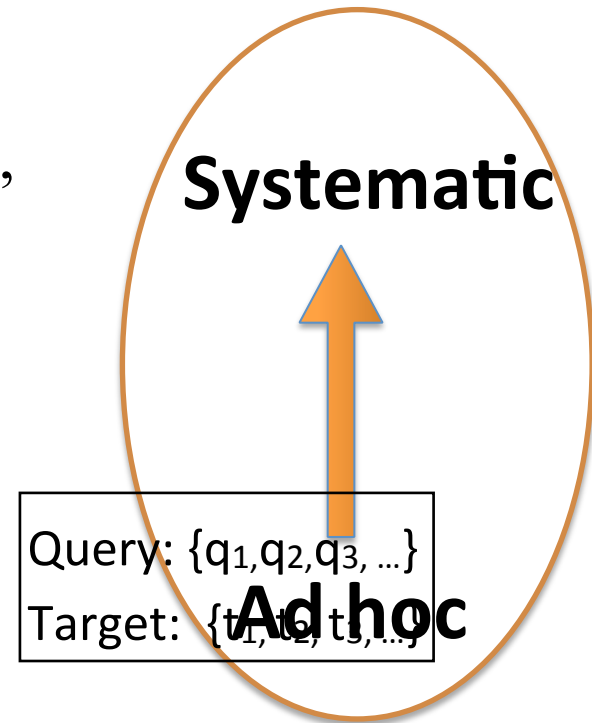
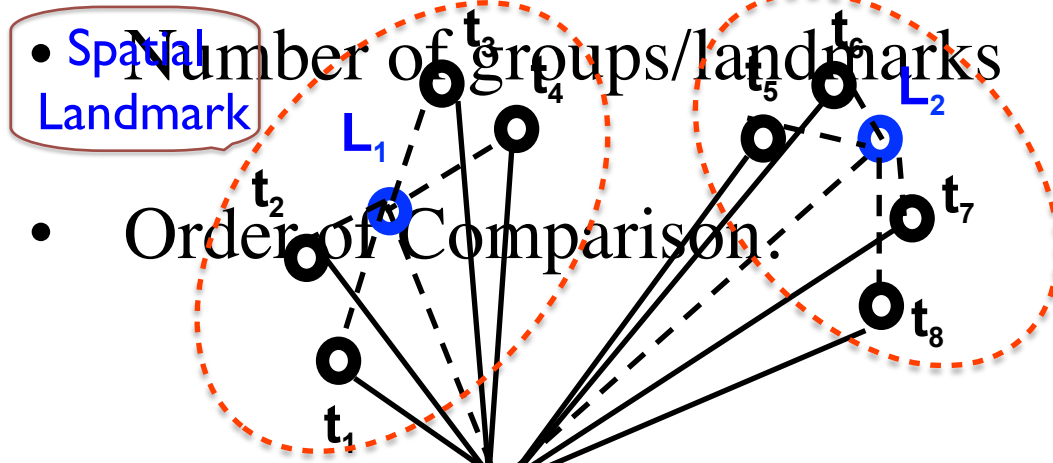


TOP Framework



Key Optimization Knobs

- Landmark definitions.
 - e.g., temporal landmark (e.g., Kmeans),
spatial landmark (e.g., KNN).



Beat the algorithms manually optimized by experts!

Optimization Selection

- 7 principles of applying TI optimization.
- Rule-based Selection Framework (Clang).

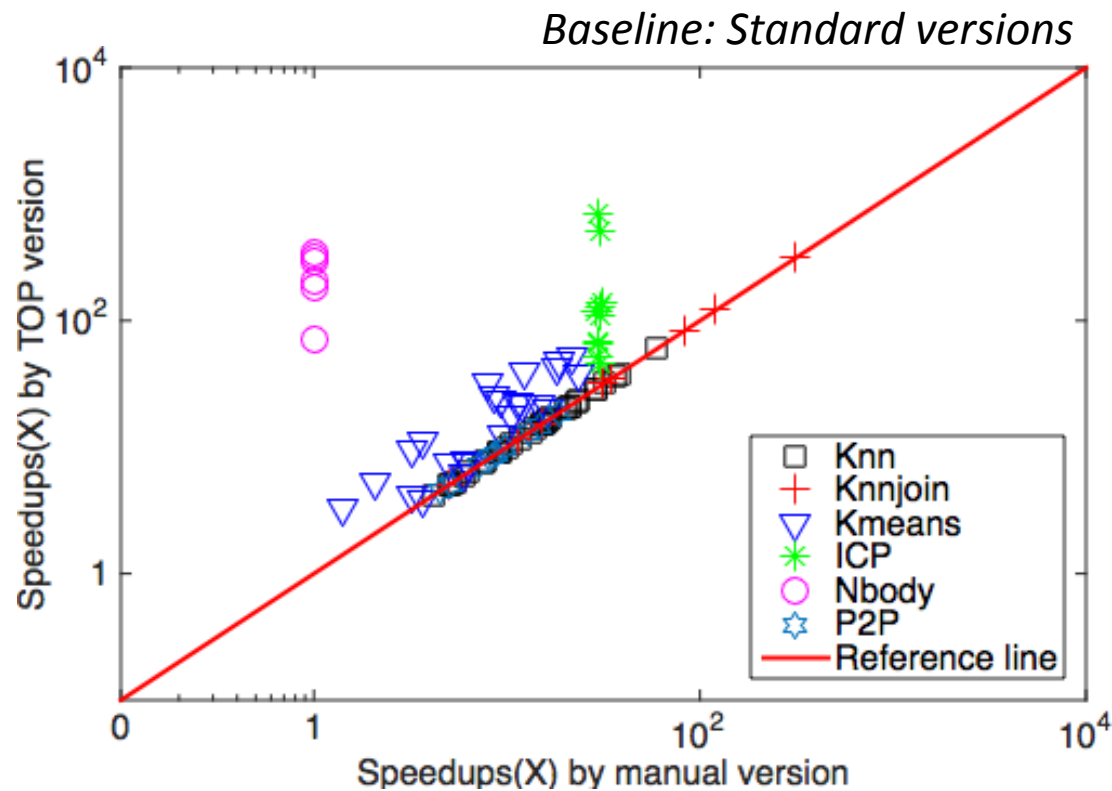
- 1. Decide** the best way of defining landmark and the number of landmarks to use.
- 2. Insert** codes preparing landmarks for optimizations.
- 3. Replace** these TOP APIs (e.g., TOP_findClosestTargets) with optimized codes.

Evaluation

- Tested on six distance-related problems.
- Compared to two other methods for each problem:
 - Standard version without TI optimization.
 - Manual optimization from previous works.
- Input: real-world data sets used in previous papers.
- Machine
 - Intel i5-4570 CPU and 8G memory.

Evaluation — Running time

Each point in the graph stands for one input setting.



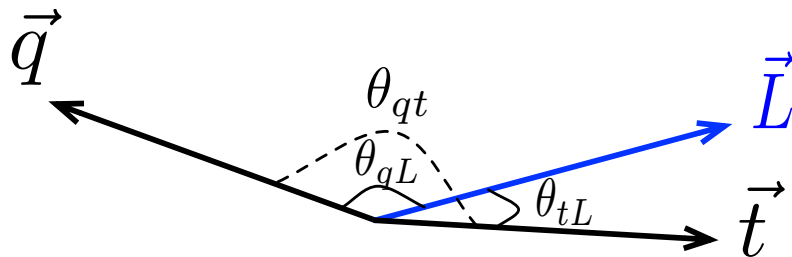
Average speedups: 50X for TOP vs. 20X for manual version from previous works.

Over 93% of the distance computation can be saved by TOP.

TI-based Strength Reduction

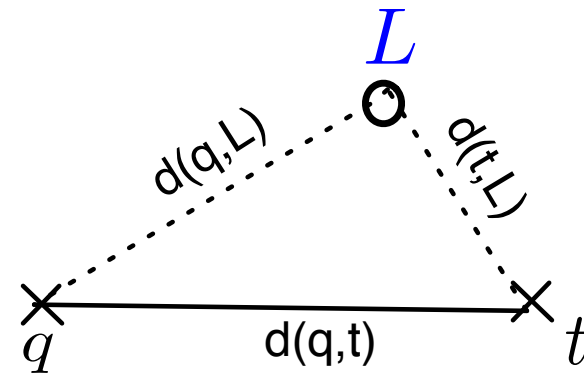
[PLDI'17]

- Theoretic foundation for generalization of TI to compute bounds of “vector dot product”!



$$\vec{q} \cdot \vec{t} \geq |\vec{q}| \cdot |\vec{t}| \cdot \cos(\theta_{qL} + \theta_{tL})$$

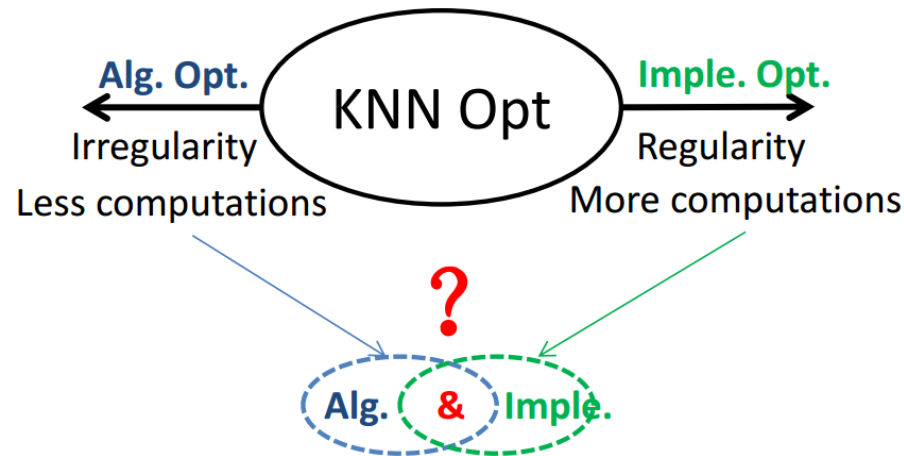
$$\vec{q} \cdot \vec{t} \leq |\vec{q}| \cdot |\vec{t}| \cdot \cos(\theta_{qL} - \theta_{tL})$$



$$|d(q,L) - d(L,t)| \leq d(q,t) \leq d(q,L) + d(L,t)$$

- Deep learning, e.g., *Restricted Boltzmann Machines*.
Text mining which uses cosine similarity as “distances”.
- Computation results is not directly used for comparison.
- Static analysis to detect code patterns for optimization (Clang).

- A fundamental tension: Redundancy and Regularity.
 - critical for performance.



- Our solution:
 - Careful implementations on GPU,
 - Elastic algorithmic design,
 - Up to 12X speedups over the state-of-art version (CUBLAS).

My Research



High-level Program Optimization:

- Implementation → **Algorithm**; Instruction → **Formula**

Algorithmic Optimization for Distance-Related Problems
[ICML'15, VLDB'15, ICDE'17, PLDI'17]

Autotuning Algorithmic Choice for Input Sensitivity
[PLDI'15]

Generalizing Loop Redundancy Elimination at a Formula Level
[OOPSLA'17]

Examining Compilation Scheduling of JIT-Based Runtime System
[ASPLOS'14]

Parallel Stochastic Gradient Descent with Sound Combiners
[applied for patent]

Other work

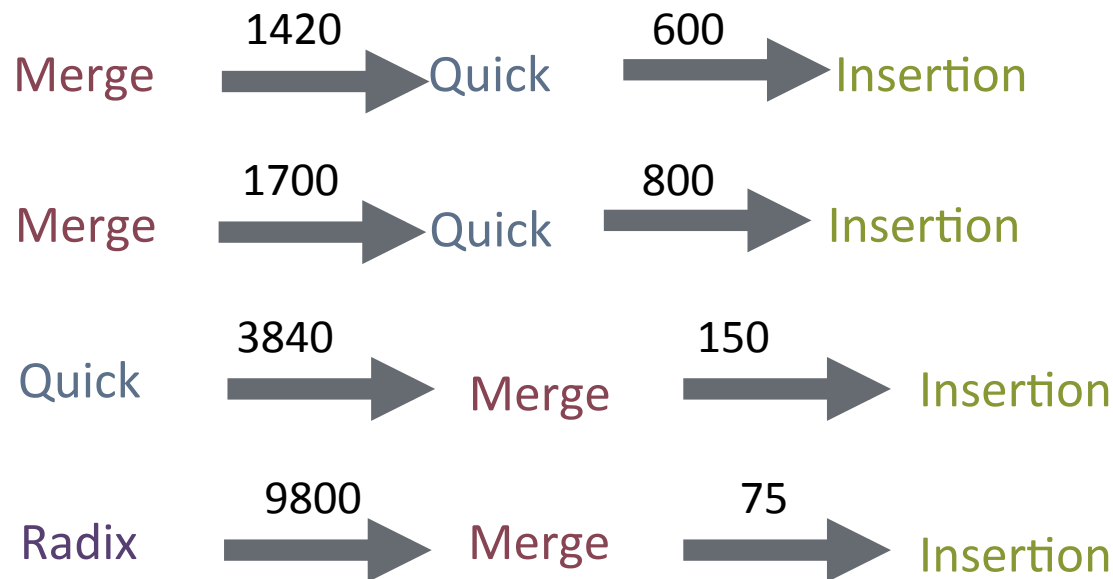
Autotuning Algorithmic Choice for Input Sensitivity

[PLDI'15]

*Collaborated w/ MIT
(Saman Amarasinghe's group)*

Algorithmic Autotuning

- Best optimization: autotuning + alg. choices.
 - E.g., what is best optimization for sorting?



...

- Huge number of potential optimizations by varying the **type** and **order** of algorithm to use.

Our Contribution

- 3X averaged speedup over static optimization
 - on 6 benchmarks (e.g., sorting, clustering, helmholtz).
- *Language and compiler support.*
- A **Two-level input learning framework**
 - the enormous optimization space,
 - variable accuracy of algorithmic choices.

My Research



High-level Program Optimization:

- Implementation → **Algorithm**; Instruction → **Formula**

Algorithmic Optimization for Distance-Related Problems
[ICML'15, VLDB'15, ICDE'17, PLDI'17]

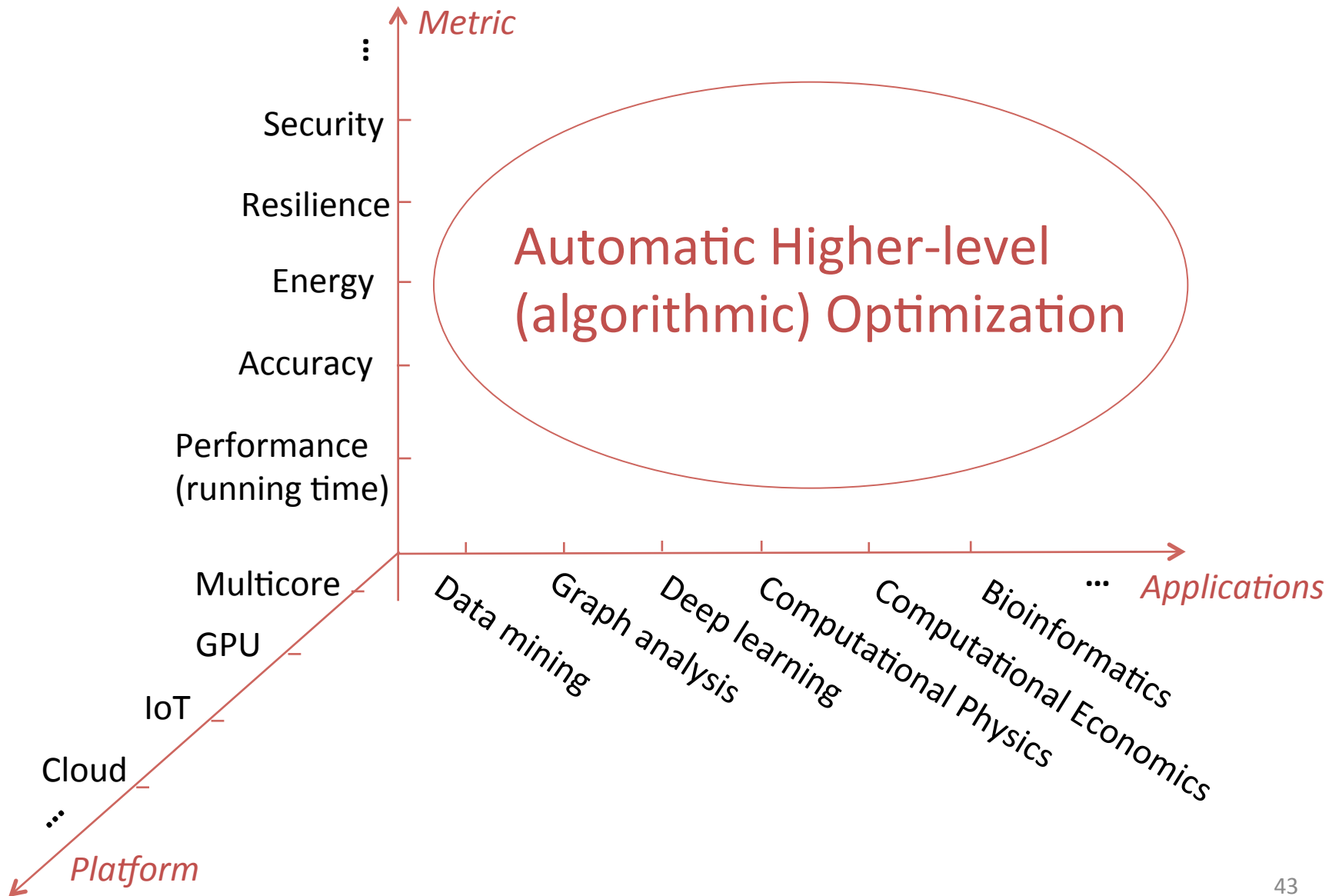
Autotuning Algorithmic Choice for Input Sensitivity
[PLDI'15]

Generalizing Loop Redundancy Elimination at a Formula Level
[OOPSLA'17]

Examining Compilation Scheduling of JIT-Based Runtime System
[ASPLOS'14]

Parallel Stochastic Gradient Descent with Sound Combiners
[applied for patent]

Future Research (Long-Term, High-Level)



Future Research (3~5 Years)

- Combine the higher-level program optimization and lower-level optimizations. (*High-performance Computing*)
- Automate extractions of the domain-specific knowledge for high-level program optimizations. (*NLP, text mining, ontology, ...*)
- Combine algorithmic optimizations with approximation-based computing?
- Deep learning in bioinformatics, astronomy, etc.
 - Hyper-parameter tuning + structural learning
 - Incremental computing for the searching process.
- Cyber-Physical Systems (*CPS*).
 - Program language support for expressing user specifications, e.g., helping resolve dependency in smart homes.
 - embedded intelligence: data analytics in edge computing (*IoT*).

Publications

[OOPSLA'17] "GLORE: Generalized Loop Redundancy Elimination upon LER-Notation", **Yufei Ding**, Xipeng Shen, to appear.

[PLDI'17] "Generalizations of the Theory and Deployment of Triangular Inequality for Compiler-Based Strength Reduction", **Yufei Ding**, Ning Lin, Hui Guan, Xipeng Shen, to appear.

[ICDE'17] "Sweet KNN: An Efficient KNN on GPU through Reconciliation of Redundancy and Regularity", Guoyang Chen, **Yufei Ding**, Xipeng Shen, to appear.

[PLDI'15] "Autotuning algorithmic choice for input sensitivity", **Yufei Ding**, Jason Ansel, Kalyan Veeramachaneni, Xipeng Shen, Una-May O'Reilly, Saman Amarasinghe. ACM SIGPLAN conference on Programming Language Design and Implementation, Portland, Oregon, June 13-17, 2015.

[ICML'15] "Yinyang K-Means: A Fast and Accurate Clustering Algorithm", **Yufei Ding**, Yue Zhao, Xipeng Shen, Max W. Liberman. International Conference on Machine Learning, Lille, France, July 06-11, 2015.

[VLDB'15] "TOP: A Framework for Topological Optimization of Query Plans", **Yufei Ding**, Xipeng Shen, Madan Musuvathi, Yixiao Zhang, Yixiao Zhang, Yixiao Zhang, Kohala Coast, Hawaii, August 24-28, 2015.

[ASPLOS'14] "Finding the Right Balance between JIT and AOT", **Yufei Ding**, Mingzhou Zhou, Zhijia Zhao, Saran Eisenstat, Xipeng Shen. The Nineteenth International Conference on Architectural Support for Programming Languages and Operating Systems, Salt Lake City, 2014.

[OOPSLA'14] "Call Sequence Prediction through Probabilistic Calling Automata", Zhijia Zhao, Bo Wu, Mingzhou Zhou, **Yufei Ding**, Jianhua Sun, Xipeng Shen, Youfeng Wu. Proc. of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems, 2015.

[CGO'13] "ProfMig: A Framework for Flexible Migration of Program Profiles Across Software Versions", Mingzhou Zhou, Bo Wu, **Yufei Ding**, and Xipeng Shen. International Symposium on Code Generation and Optimization Shenzhen, China, 2013.

Questions?