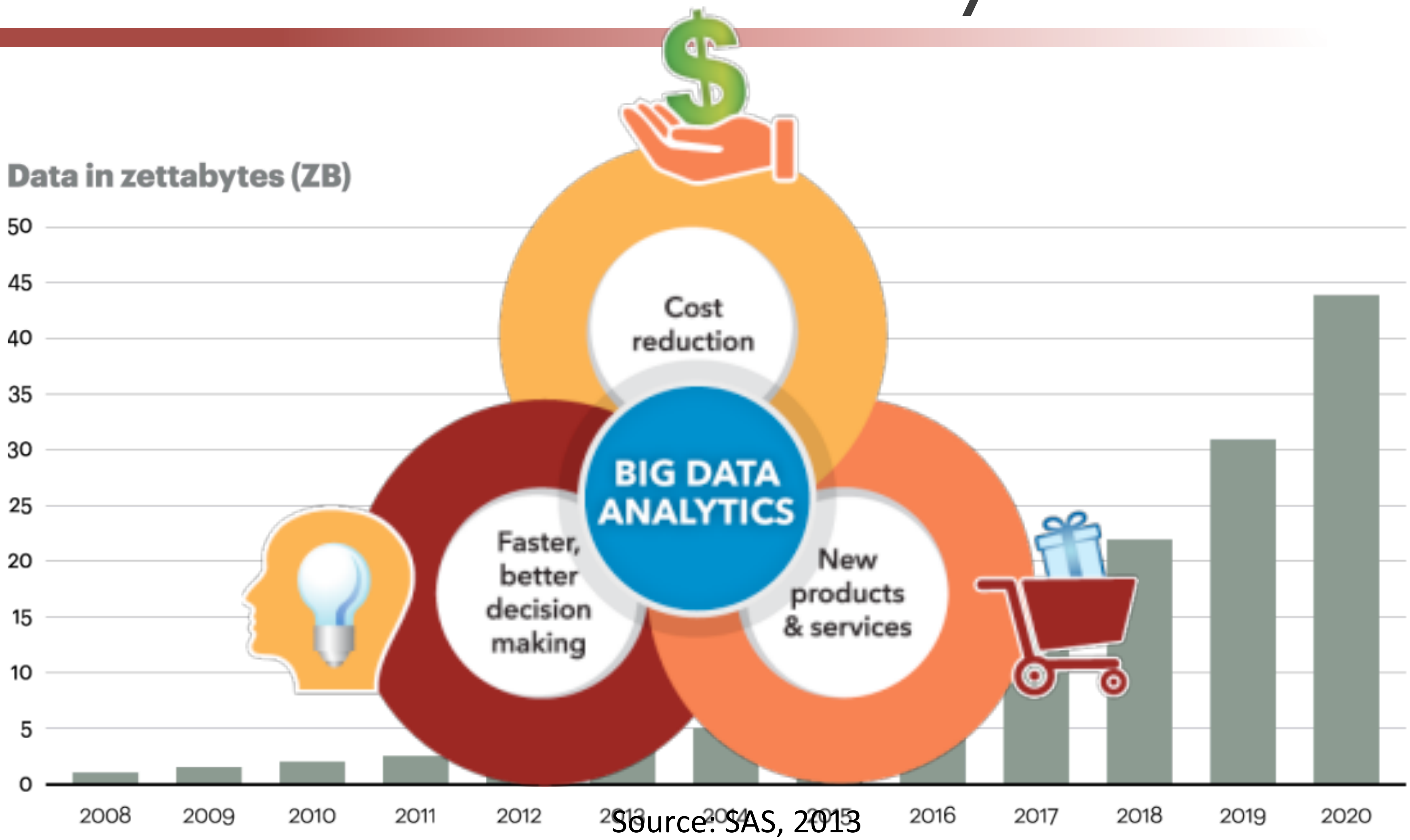
High-level Program Optimization for Data Analytics

Yufei Ding

North Carolina State University

Motivation: Faster Data Analytics



Source: SAS, 2013

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*

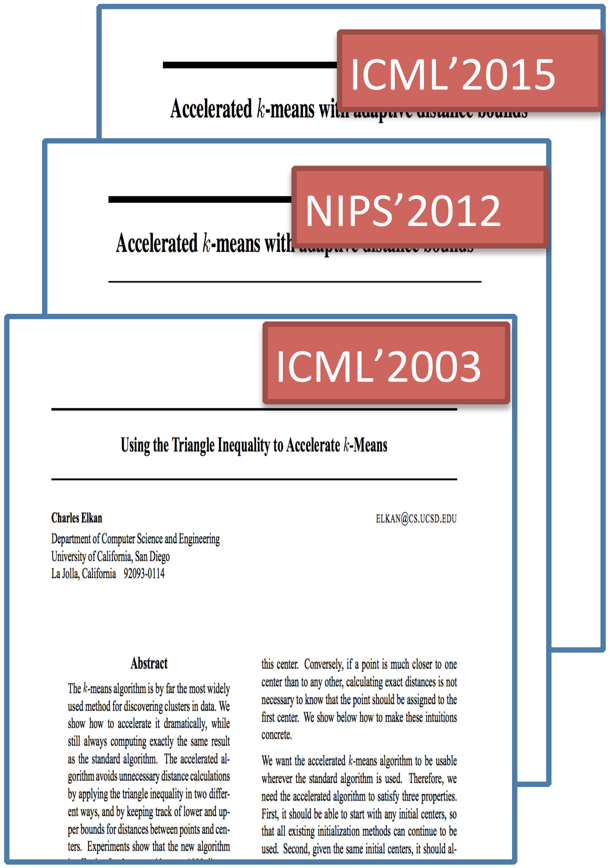
Distance-related Problems

- These algorithms are widely used.

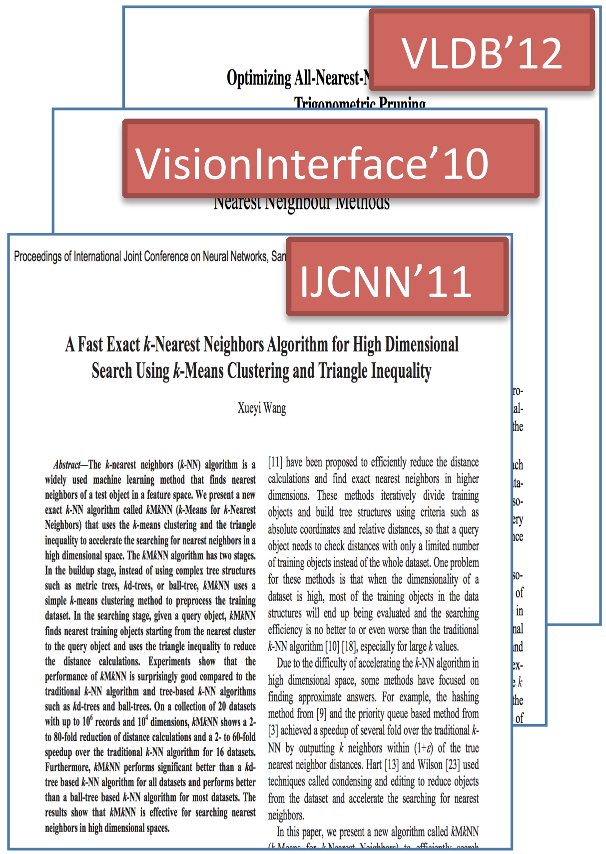
Problems	Domain
KMeans	Data Mining (Among Top 10 Most Popular DM Algorithms)
KNN (K Nearest Neighbor)	
KNN Join	
ICP(Iterative Search Problem)	Image Processing
P2P (Point-to-Point Shortest Path)	Graphics
Nbody	Computational Physics

  – Distance computations are the performance bottleneck.

KMeans

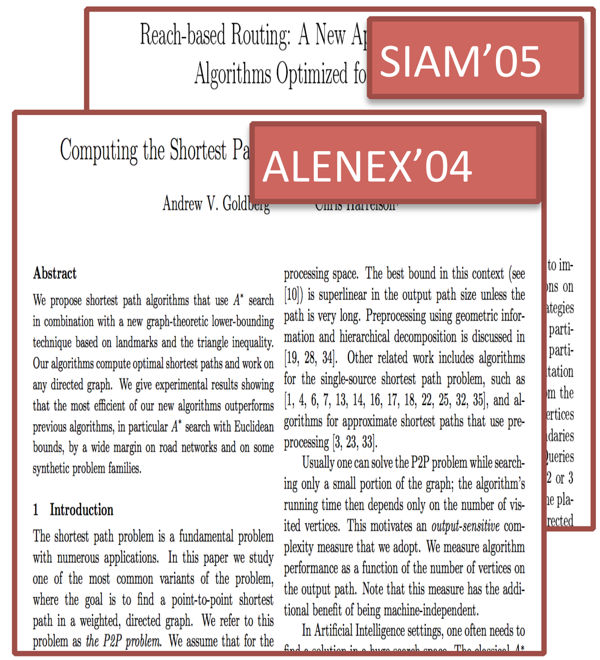


KNN



P2P:

(Point-to-Point Shortest Path)



. . .

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

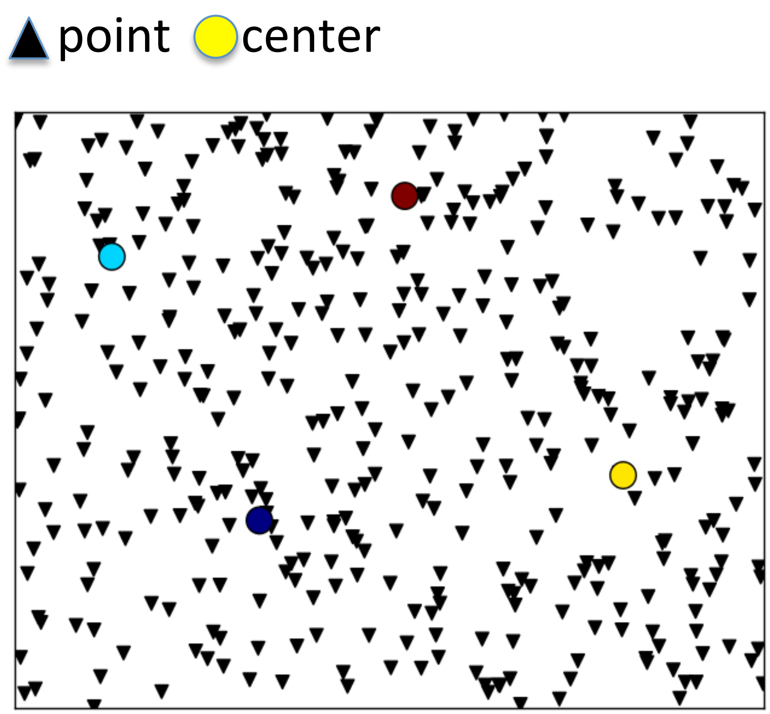
Yinyang KMeans:
A Drop-In Replacement of the Classic KMeans with Consistent Speedup

ICML'2015

Collaborated w/ MSR
(Madan Musuvathi's group)

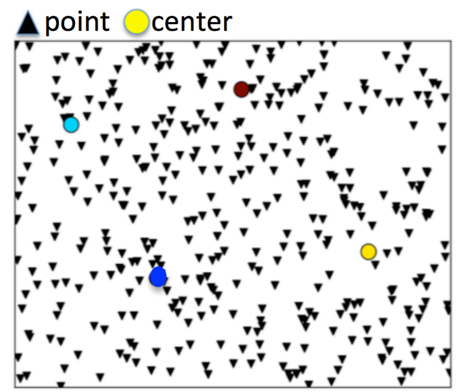
Background: KMeans

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



Background: KMeans

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



Set initial centers

I. Point Assignment: Assign points to clusters based on d(p, c) ∀p,c

II. Center Update: Update centers w/ new centroids.

Convergence

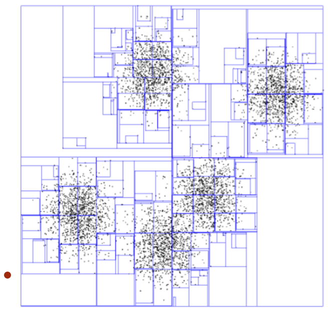
Step I is the performance bottleneck: N*K distances calc. per iteration.

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

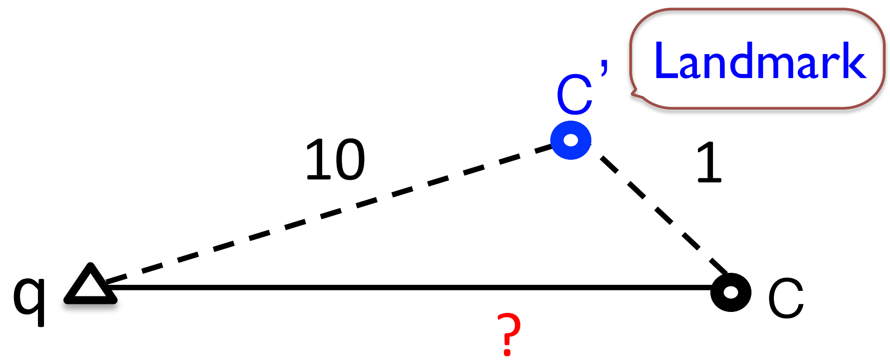
V.S.

Yang: lower bound

(these bounds comprises the filters for distance computations)

Triangular Inequality (TI)

- The fundamental tool for getting bounds:



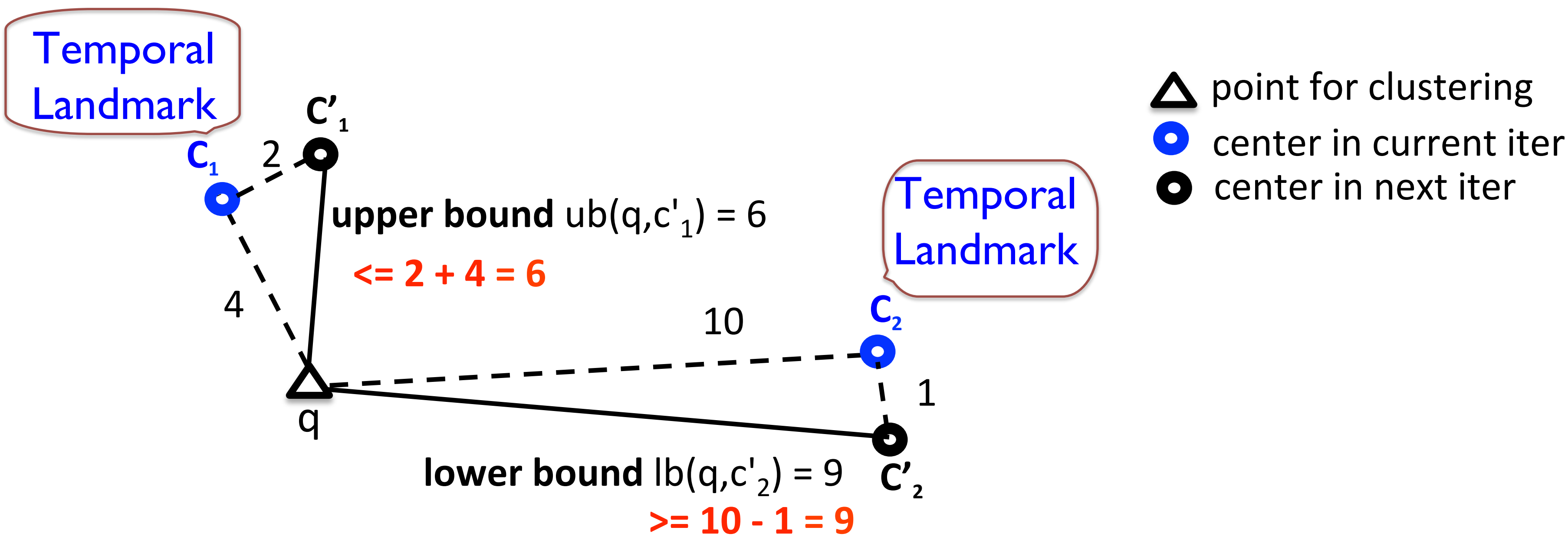
TI: $\underline{|d(q,c') - d(c',c)|} <= d(q,c) <= \underline{d(q,c') + d(c',c)}$

Lower bound:
lb(q, c) = |10 – 1| = 9

Upper bound:
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.)

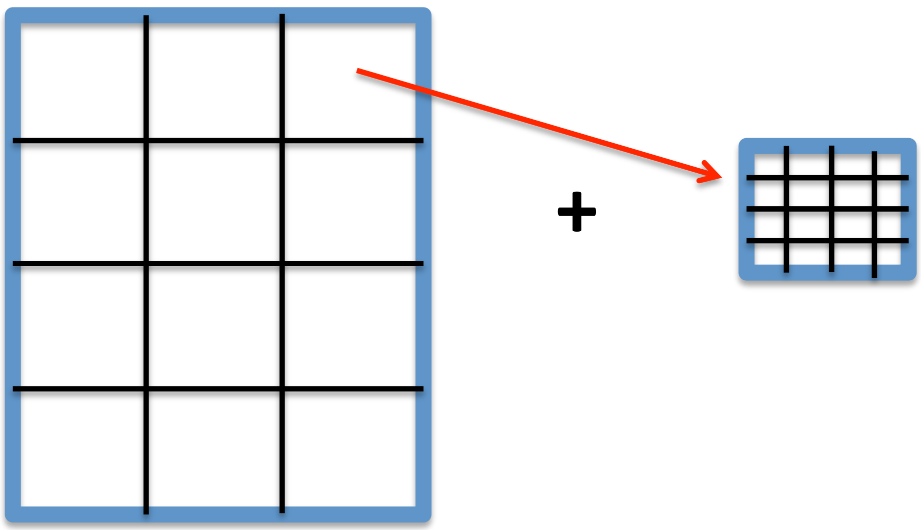


Conclusion: No, because $ub(q,c'_1) < lb(q,c'_2)$.

Design of Yinyang Kmeans

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

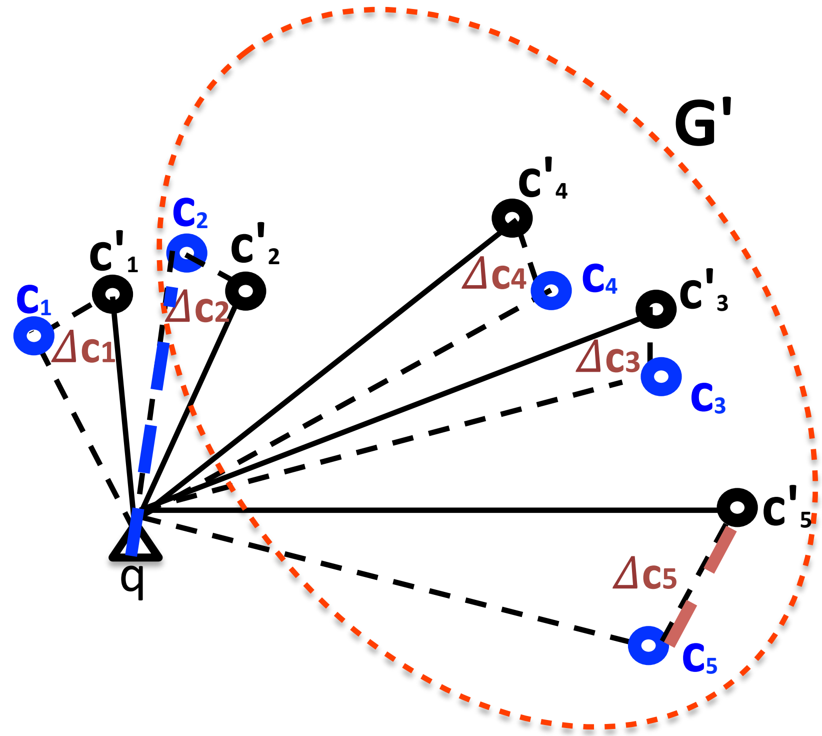
Group (Global) filter **Local filter**



Global Filtering

One check

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



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

next iter: G = {C$_1$,C$_2$, ..., C$_k$} – C$_1$

(q is currently assigned to C$_l$)

How to compute these bounds?

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

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

Benefits: over 60% redundant distance computation can be removed

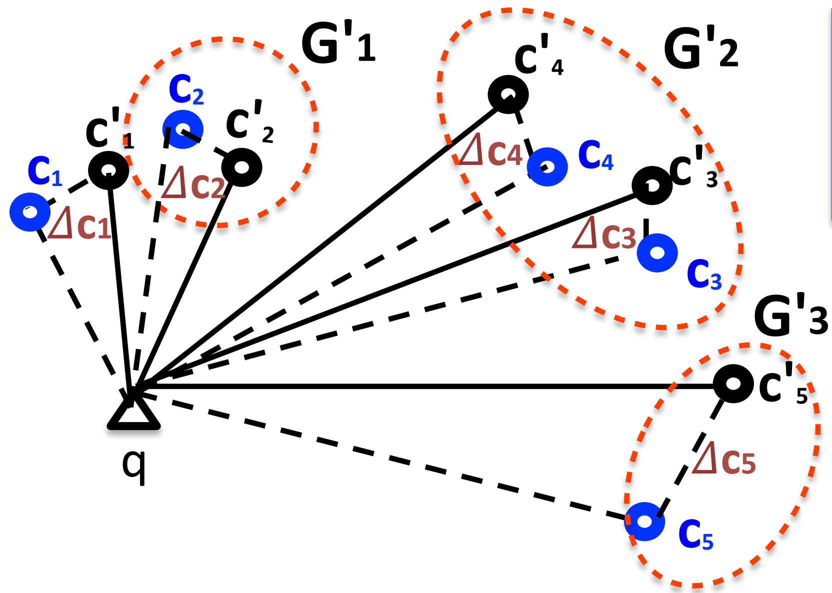
Limiting factors of 1 lower bound:

1. lb(q,G): closest center in G' (e.g., c_2).
2. max (Δc_i): biggest drifter (how far a center moved across iteration) . (e.g., Δc_5).

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: $\{G_1, G_2, \ldots, G_m\}$

How to compute these bounds?

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

Benefits of m lower bounds:

1. lb(q,Gi): local closest center in Gi.
2. max (Δ(c)): local biggest drifter.

Over 80% redundant distance computations can be removed

Group Filtering

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

$$ub(q,c'_1) = ub(q, c_1) + \Delta c_1$$
$$lb(q, G'_i) \leq lb(q,G_i) - \max(\Delta c), \forall c \in G_i$$

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

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

Group Filtering

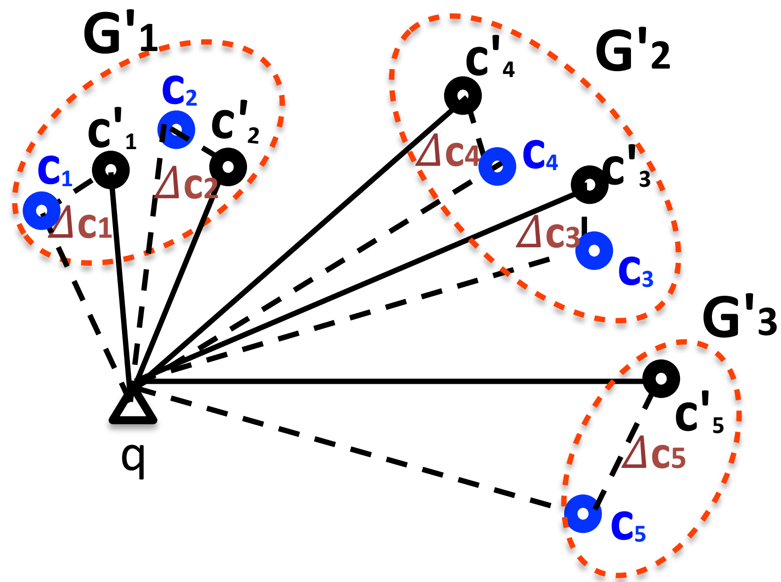
- When/How to group centers?

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

- How many groups? 

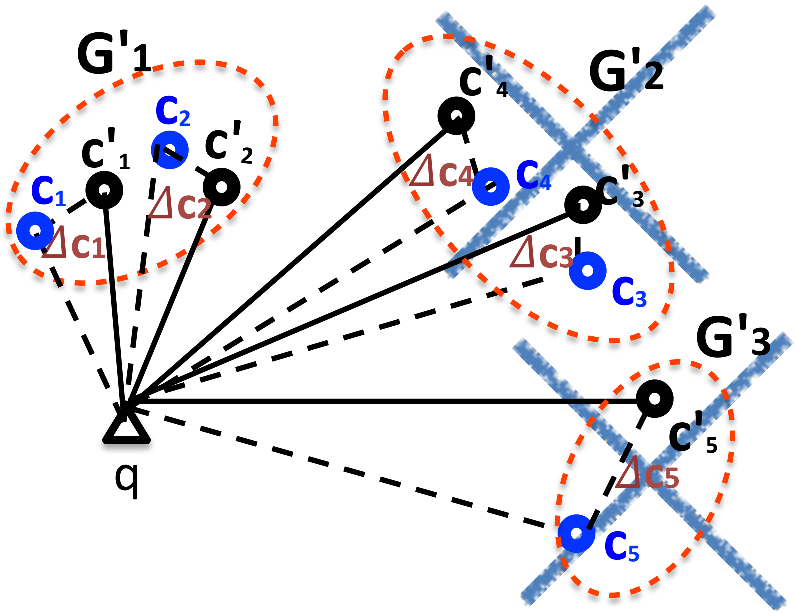
A space-conscious elastic design:

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



Grouping centers into m groups: {G_1,G_2, …, G_m}

Local Filtering



Filtering rule: for each center in the remaining group, if *Min(q,G'i)* $\le$ *lb(q, Gi)* - Δc_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]*.

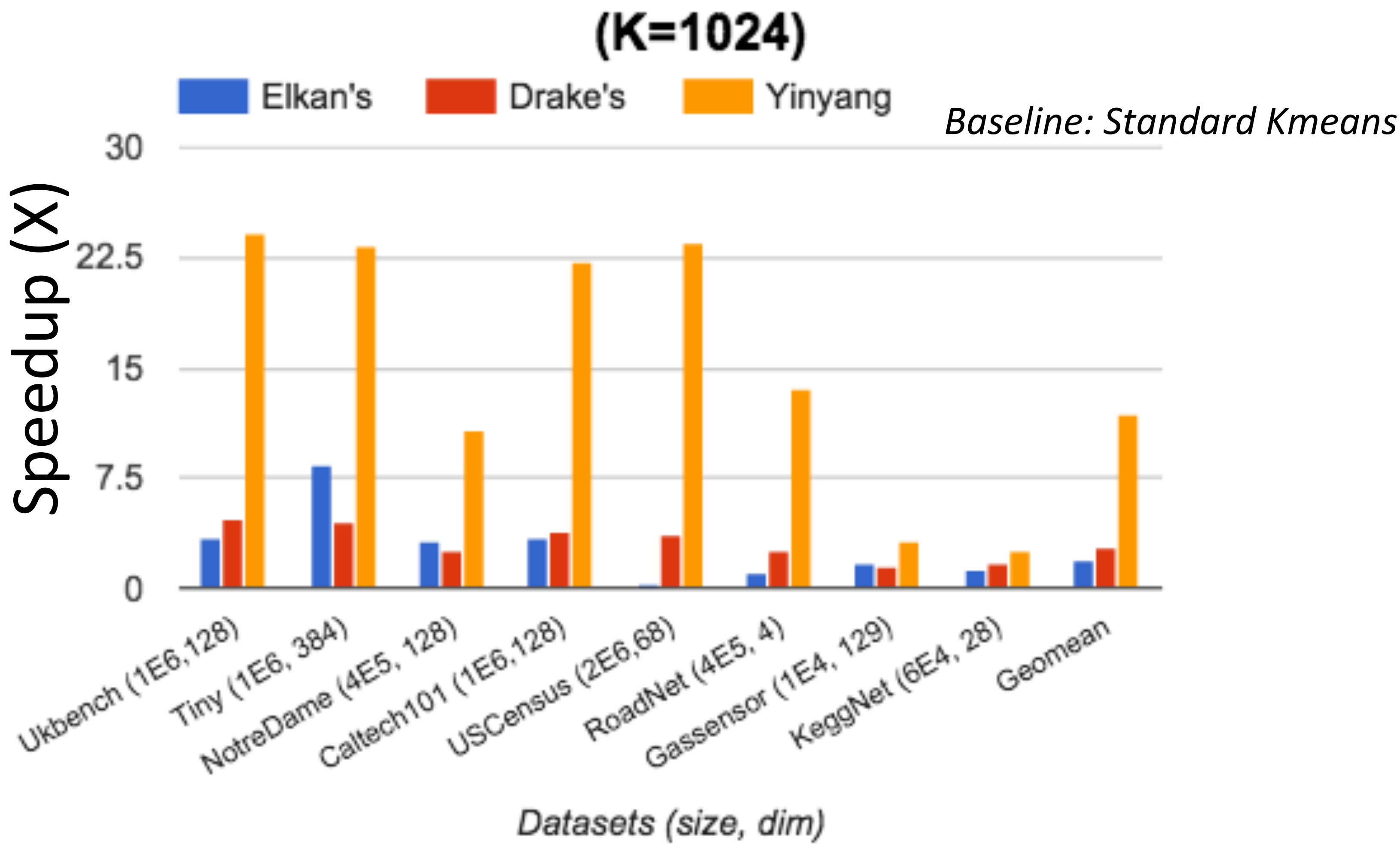
- Implemented in GraphLab (a map-reduce framework)
  - http://research.csc.ncsu.edu/nc-caps/yykmeans.tar.bz2

- Two machines
  - 16GB memory, 8-core i7-3770K processor
  - 4GB memory, 4-core Core2 CPU

Evaluation

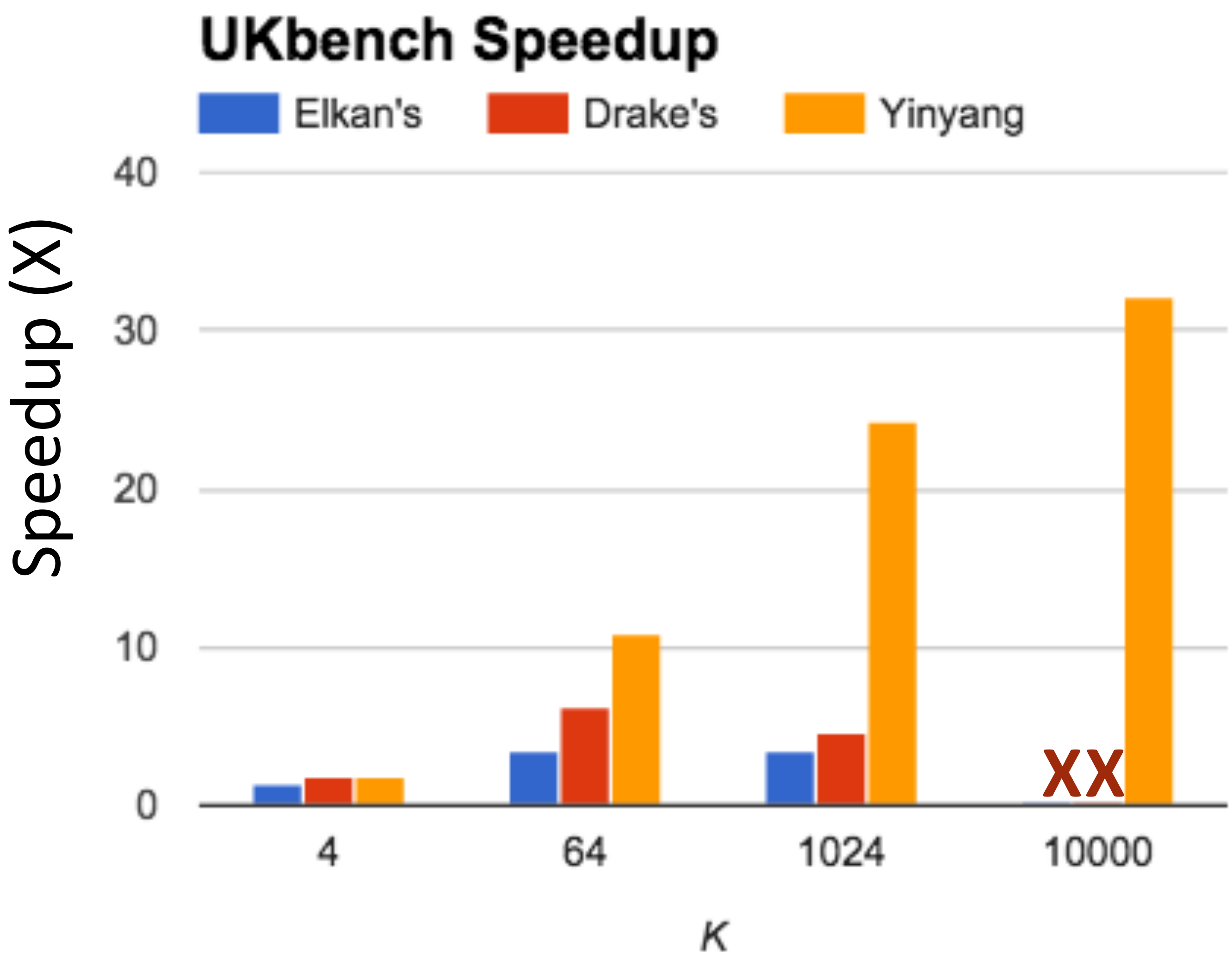
(16GB, 8-core)

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



Evaluation

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



Microsoft Power BI

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**.

Hacker News new | comments |

▲ Yinyang K-Means: A Replacement

45 points by jcr 549 days ago | hide | past | web

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-Mear
faster than Stan

datatonic @teamdatatonic

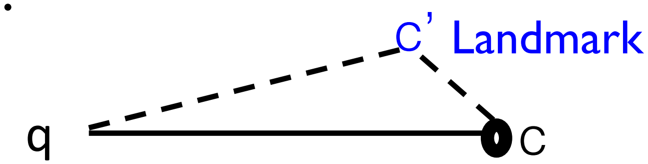
Yinyang K-Mear
the Classic K-M

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.

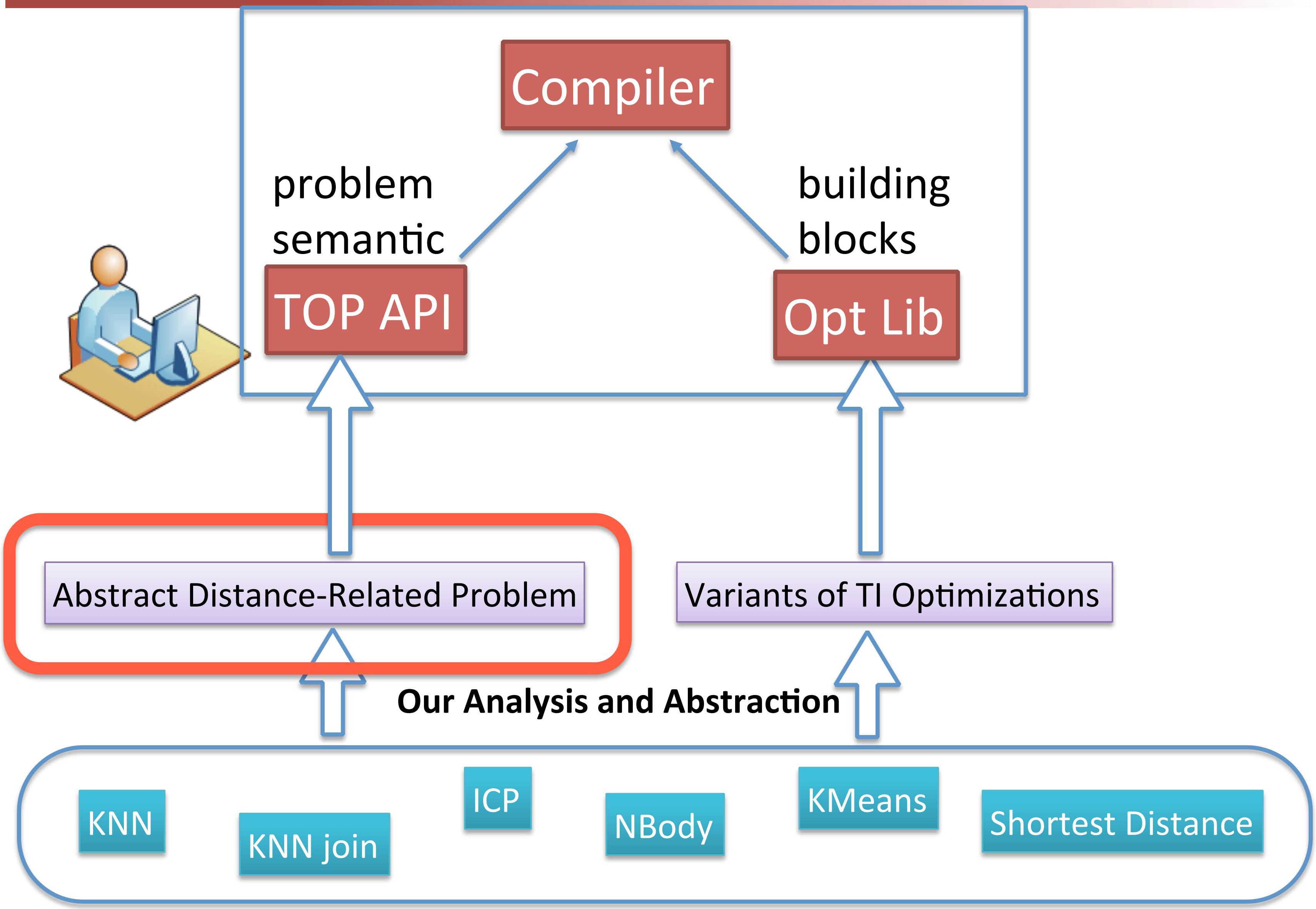
Automatic Framework

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 <Q, T, D, R, C>
  - finding some kind of Relations between two sets of points, a Query set and a Target set, based on certain type of Distance and under some update Constraints.

Problems	*Query*	*Target*	*Distance*	*Relation*	*Constraints*
KMeans	**Points**	**Centers**	**Euclidean**	**Top 1 Closest**	**Iterative update to T**

KMeans Written with Our APIs

Distance

```
TOP_defDistance(Euclidean); // distance definition
T = init();
changedFlag = 1;
while (changedFlag){
    // find the closest target (a point in T) for each point in S
    N = TOP_findClosestTargets(1, S, T);
    TOP_update(T, &changedFlag, N, S); // T gets updated
}
```

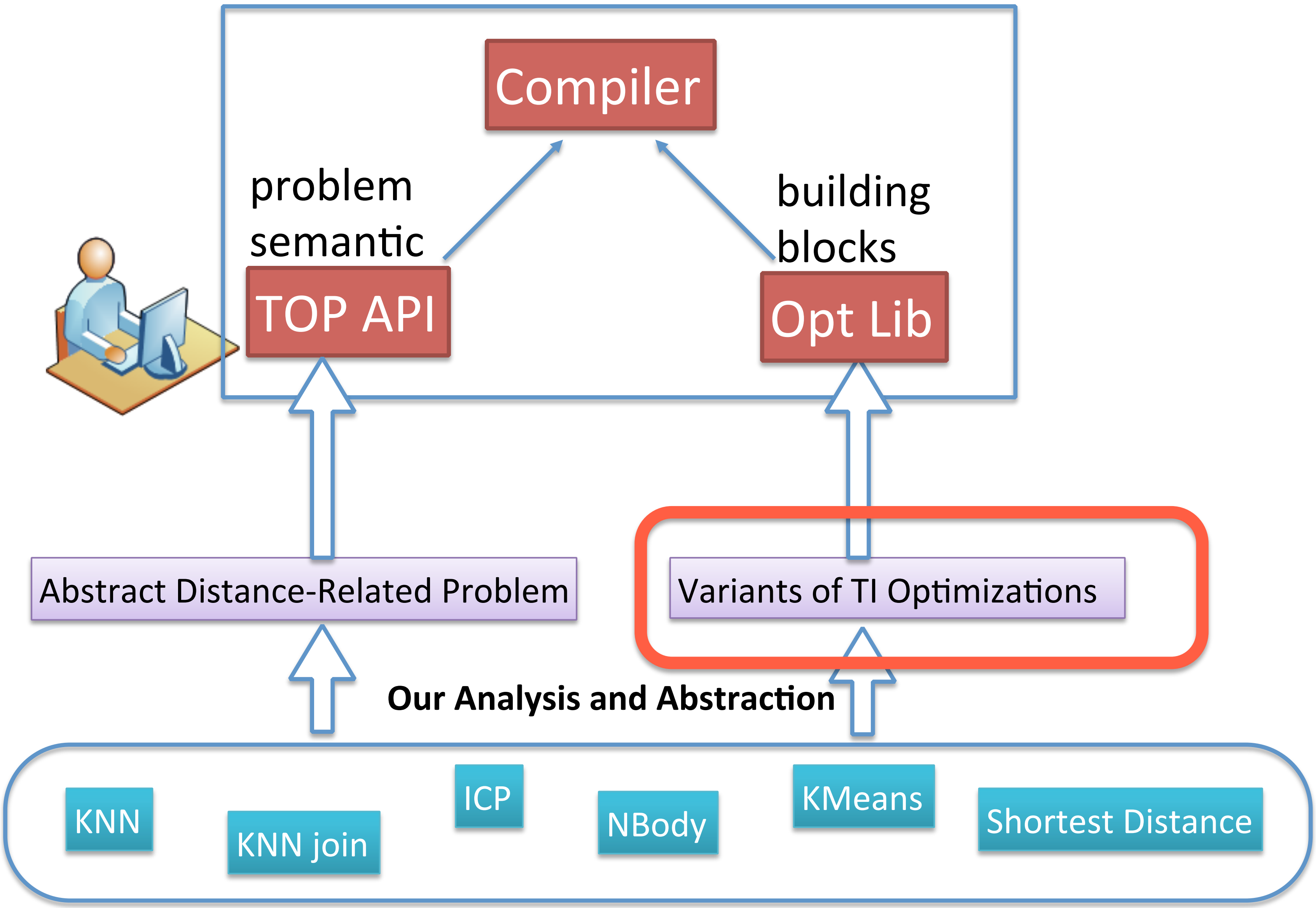
Query

Target

Constraint

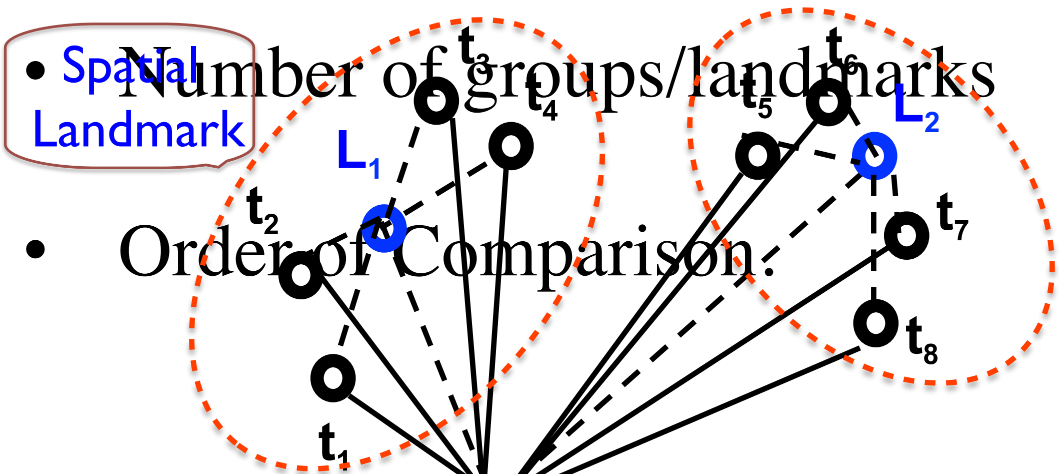
Relation

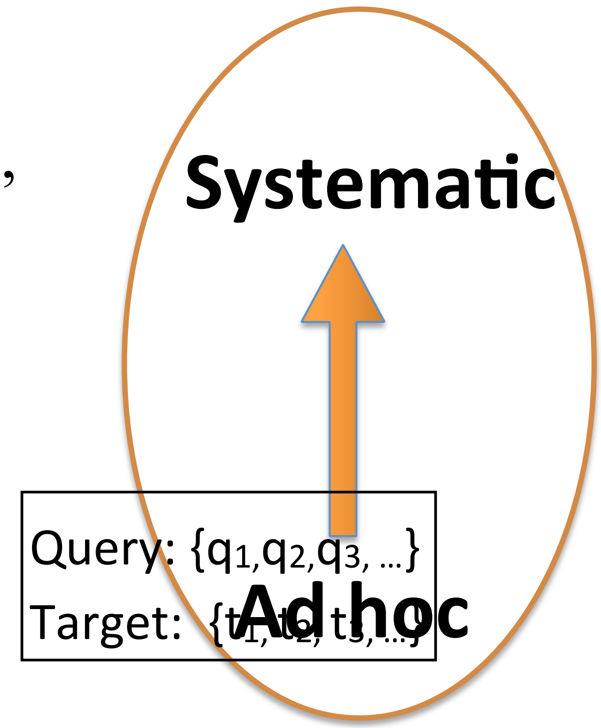
TOP Framework



Key Optimization Knobs

- Landmark definitions.
  - e.g., temporal landmark (e.g., Kmeans), spatial landmark (e.g., KNN).
- Number of groups/landmarks
- Order of Comparison.





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.

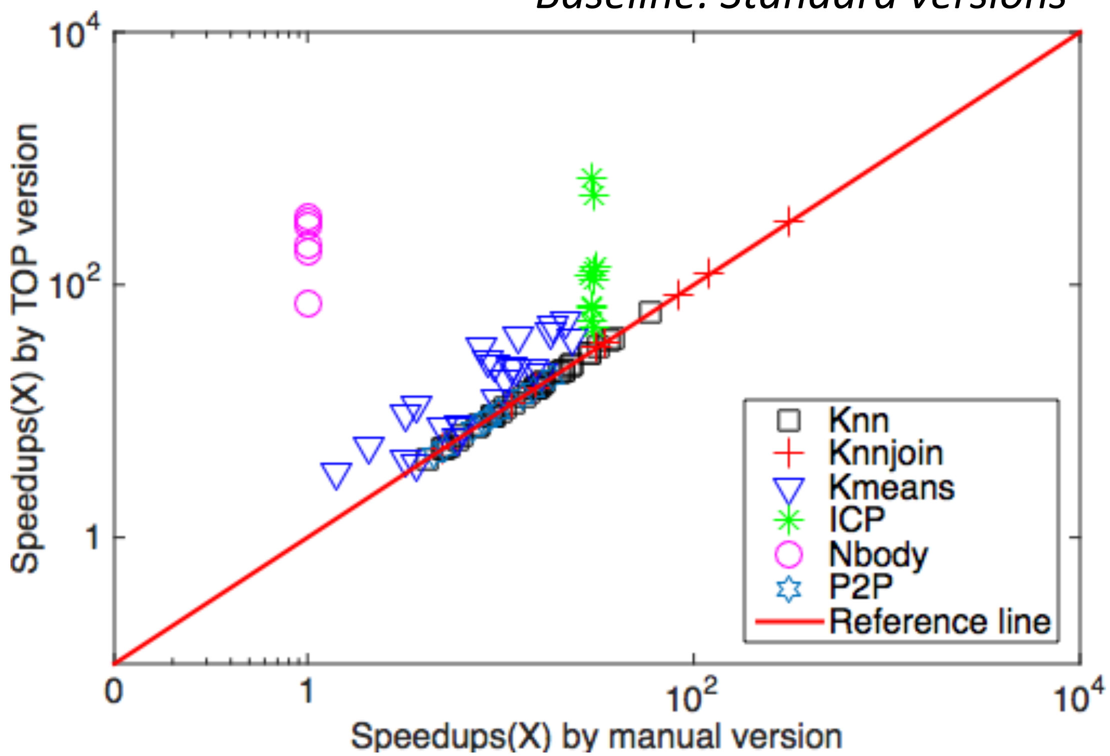
- Input: real-world data sets used in previous papers.

- Machine
  - Intel i5-4570 CPU and 8G memory.

Evaluation — Running time

Each point in the graph stands for one input setting.

Baseline: Standard versions



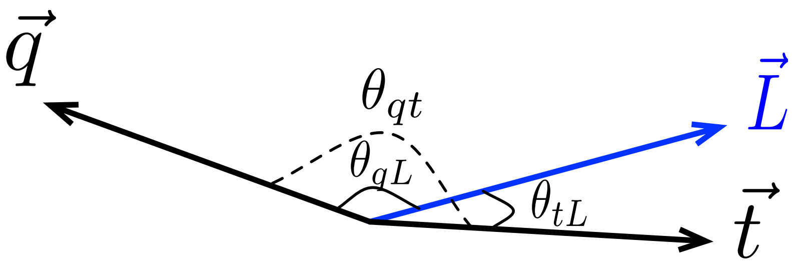
Average speedups: 50X for TOP vs. 20X for manual version from previous works.

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

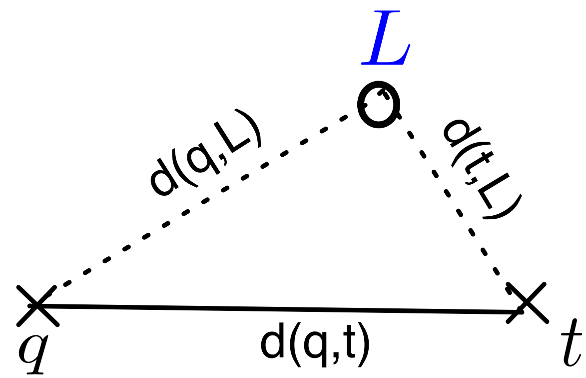
TI-based Strength Reduction

[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)| ≦ d(q,t) ≦ d(q,L) + d(L,t)

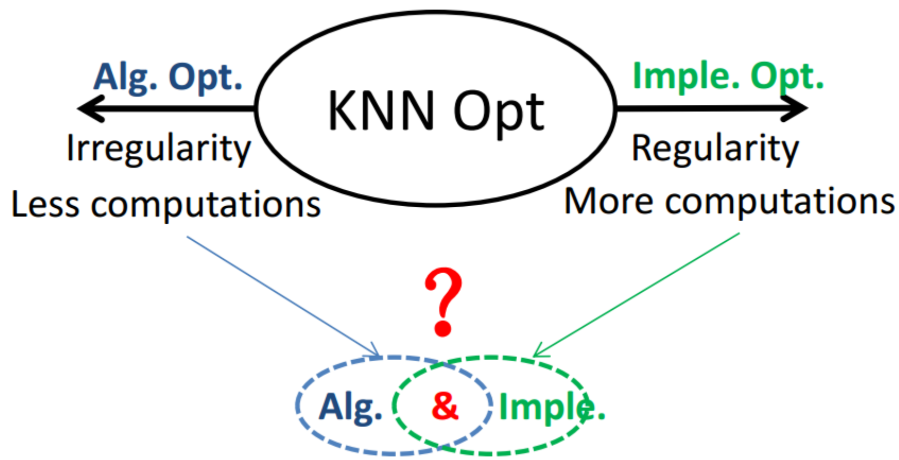
  - Deep learning, e.g., *Restricted Boltzmann Machines.* Text mining which uses cosine similarity as "distances".
  - Computation results is not directly used for comparison.

- Static analysis to detect code patterns for optimization (Clang).

KNN on GPU

ICDE'2017

- 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?

Merge $\xrightarrow{1420}$ Quick $\xrightarrow{600}$ Insertion

Merge $\xrightarrow{1700}$ Quick $\xrightarrow{800}$ Insertion

Quick $\xrightarrow{3840}$ Merge $\xrightarrow{150}$ Insertion

Radix $\xrightarrow{9800}$ Merge $\xrightarrow{75}$ Insertion

. . .

  - 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]

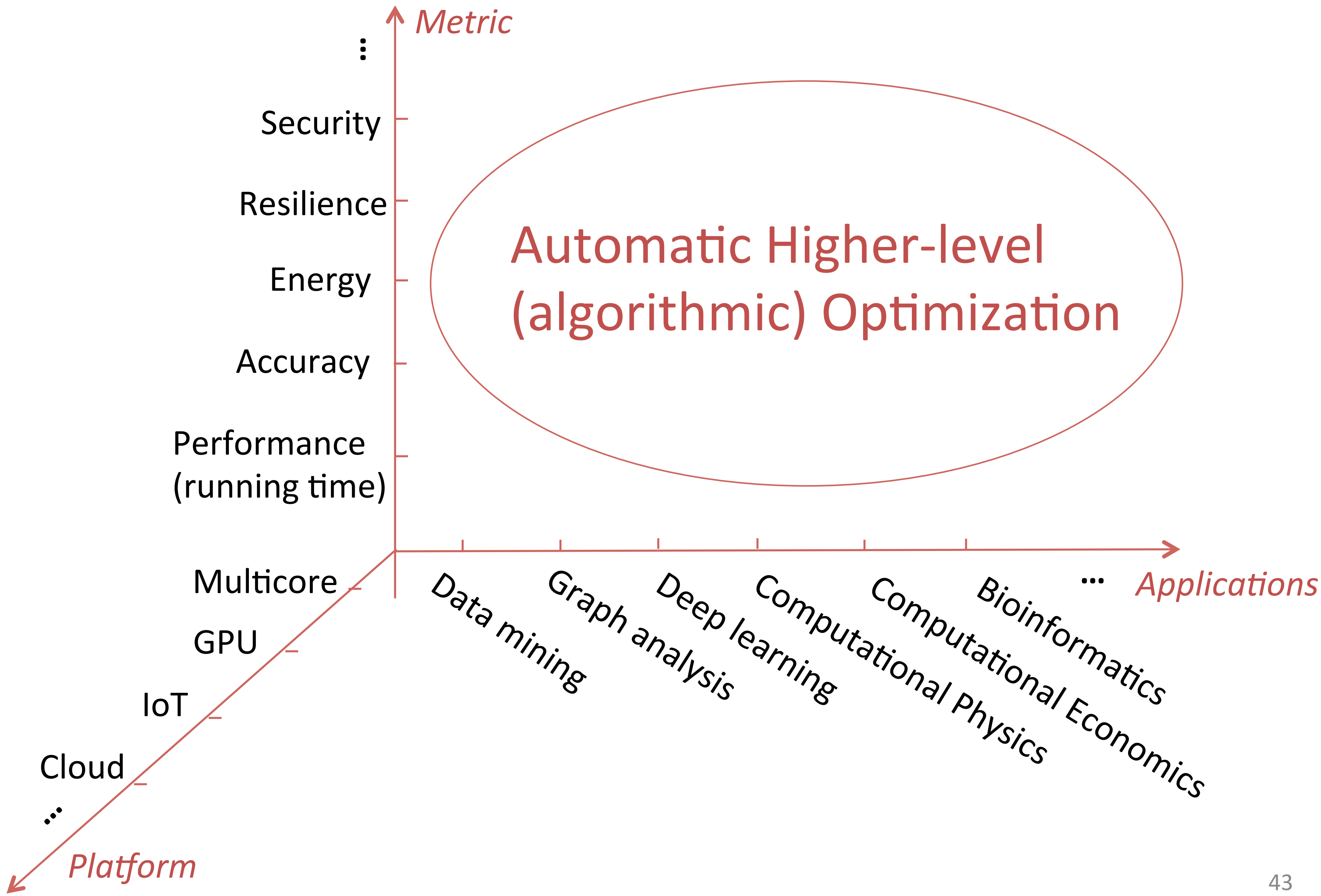
Autotuning Algorithmic Choice for Input Sensitivity
[PLDI'15]

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, Portlan

[ICML’15] “Yinyang K-Me … ent Speedup”, **Yufei Ding**, Yue Zhao, Xipeng Shen, Ma … achine Learning, Lille, France, July 06-11, 2015.

[VLDB’15] “TOP: A Framew … ed Problems”, **Yufei Ding**, Xipeng Shen, Madan Musuv … y Large Data Bases, Kohala Coast, Hawaii, Augu

[ASPLOS’14] “Finding the … Scheduling for JIT-Based Runtime System”, **Yufei Ding**, … The Nineteenth International Conference on Architectural Support for Programming Languages and Operating Systems, Salt Lake City, 2014.

Questions?

[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.