Accelerating nearest neighbor search on manycore systems

Lawrence Cayton
Max Planck Institute for Intelligent Systems

Why study it?

- NN search is a core subroutine in machine learning (and DB, CG, IR, theory ..)

- But it's expensive, especially at test time.

Why study it now?

Why study it now?

Research@Intel: The cloud's future is many-core and GPU accelerated *

Published about 19 hours ago - by Jon Stokes | Posted in: Uptime

FEATURE STORY

And at the annual Research@Intel day, the chipmaker talked up its plans for the future of the datacenter, and Ars was there to find out what Intel was cooking up in its labs.

READ MORE (2 PAGES) COMMENT (37)

* from Ars Technica, 15 June 2011.

Why study it now?

 *

Research@Intel: The cloud's future is many-core and GPU accelerated

Published about 19 hours ago - by Jon Stokes | Posted in: Uptime

FEATURE STORY

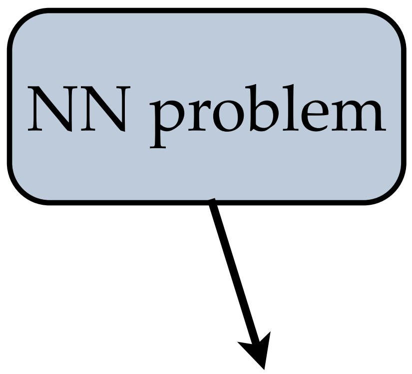
And at the annual Research@Intel day, the chipmaker talked up its plans for the future of the datacenter, and Ars was there to find out what Intel was cooking up in its labs.

READ MORE (2 PAGES)   COMMENT (37)

- GPUs/multicore CPUs = tremendous power for data analysis
- But unleashing this power requires a fundamental rethinking of algorithms and data structures.

* from Ars Technica, 15 June 2011.

Motivation



Motivation

NN problem

(single core) brute force

Motivation

NN problem

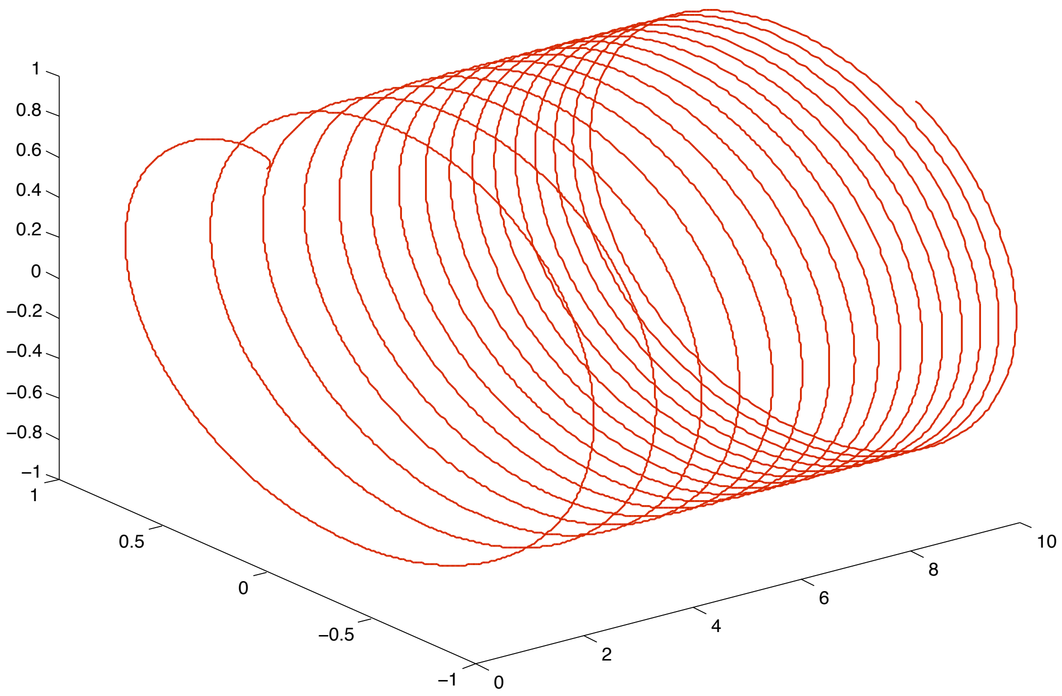
(single core) brute force —speed-up→ metric data struct

Algorithmic

- Sublinear dependence on n
- "constant" dependent on intrinsic dimensionality, not extrinsic dimensionality

Aside: extrinsic / intrinsic

Data often only **appears** high-d,
but is actually **intrinsically** low-d.



Want algs that scale with the intrinsic dim

Motivation

NN problem

(single core) brute force —speed-up→ metric data struct

Algorithmic

- Sublinear dependence on n
- “constant” dependent on intrinsic dimensionality, not extrinsic dimensionality

Motivation

NN problem

(single core) brute force

Motivation

NN problem

(single core) brute force

speed-up

(many core) brute force

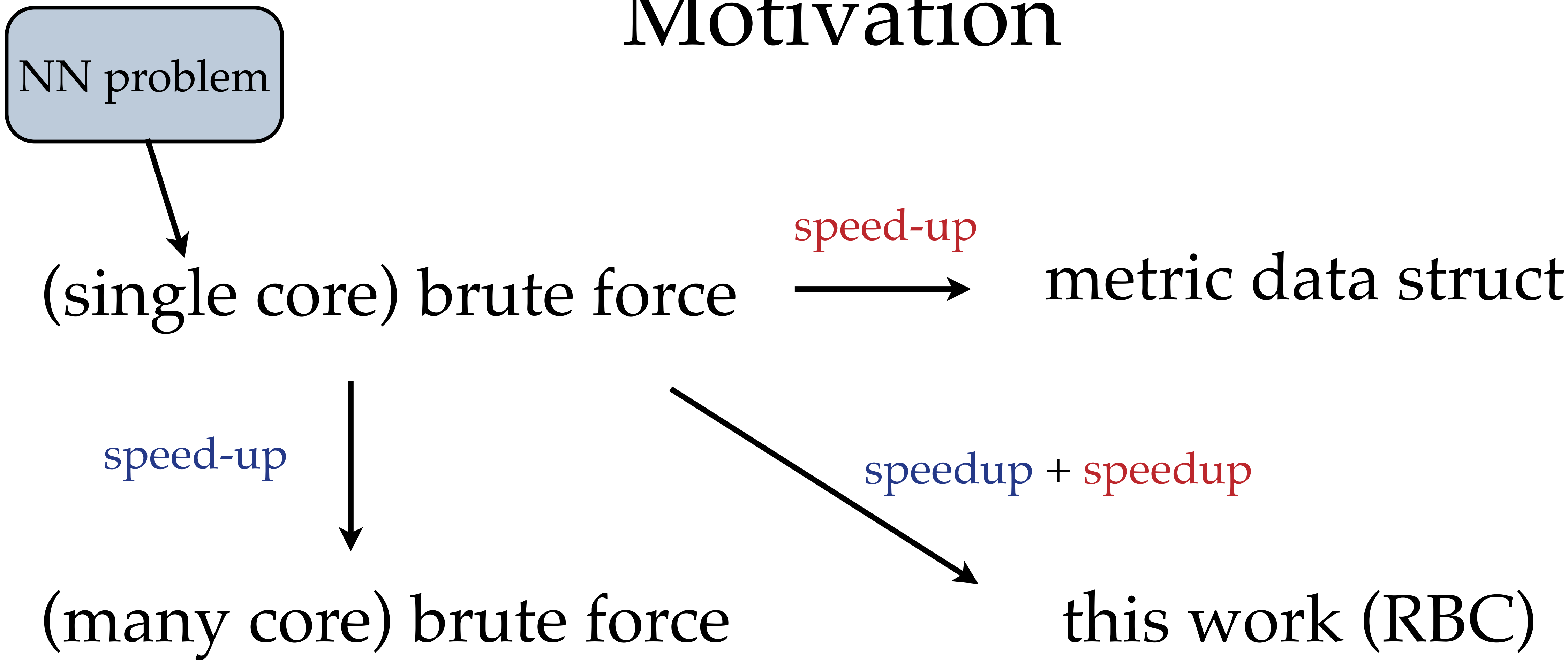
Structural: BF is trivial to parallelize

Motivation

NN problem

(single core) brute force

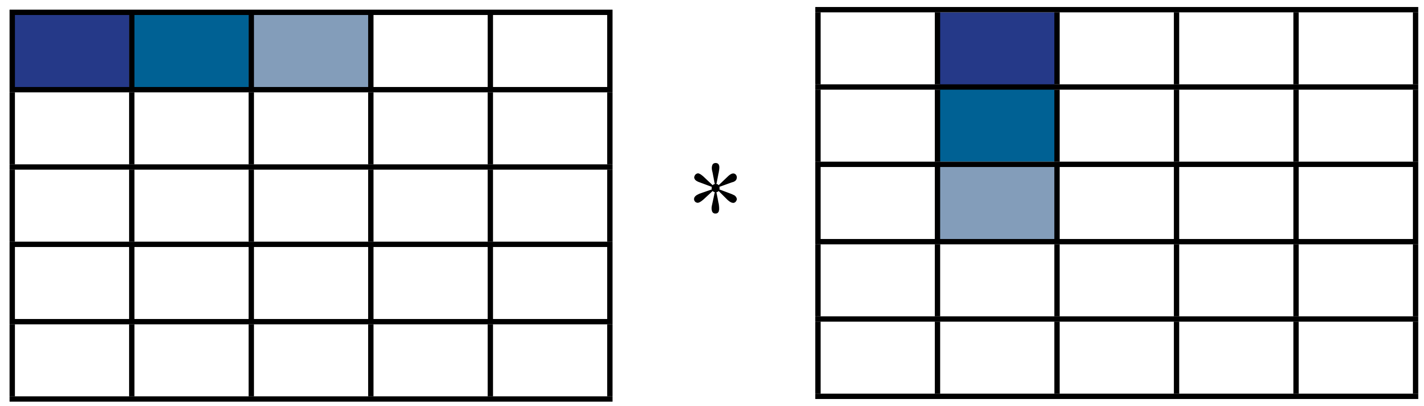
Motivation



want structural + algorithmic benefits.

What works on many core?

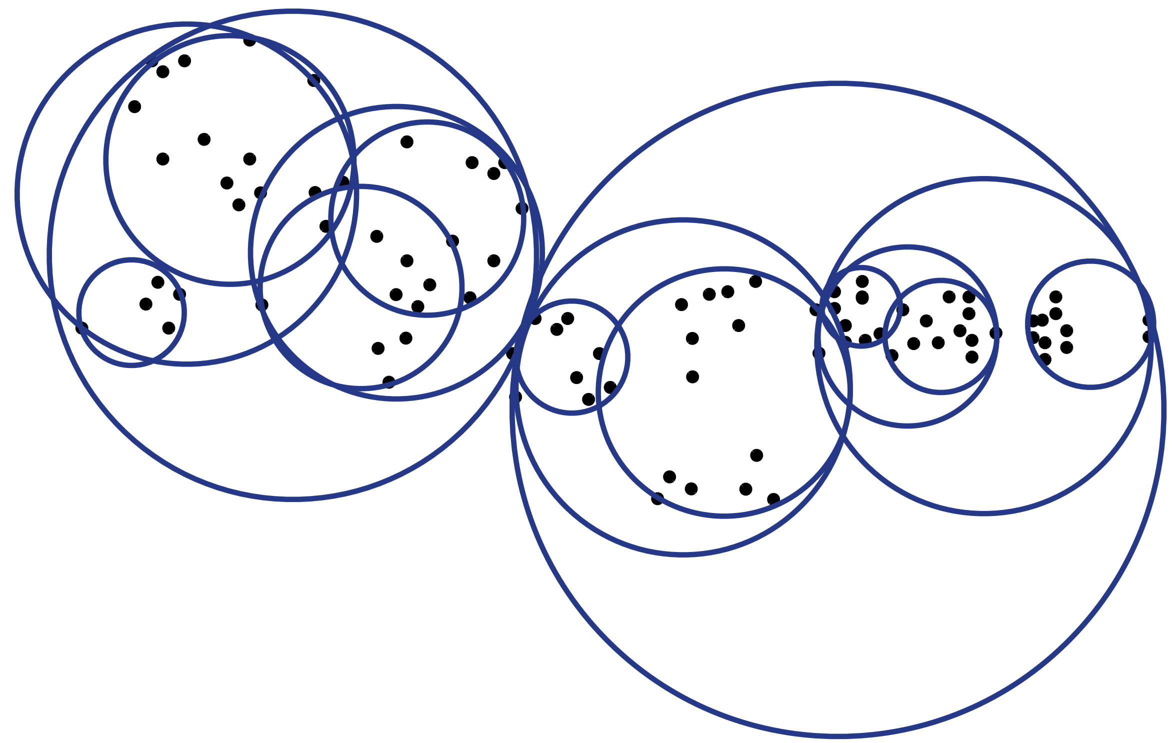
matrix multiplication: it's the operation that gets closest to using all of a processor.



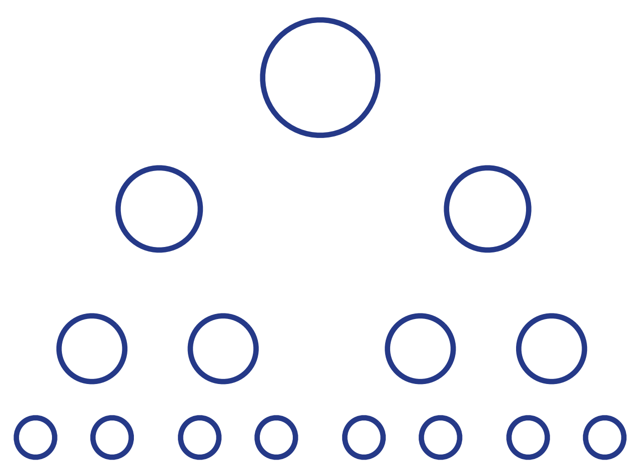
- Many independent operations
- No conditionals
- High memory re-use + regular memory access

NN data structures

Hierarchically decompose space; hopefully will only have to look at a small part

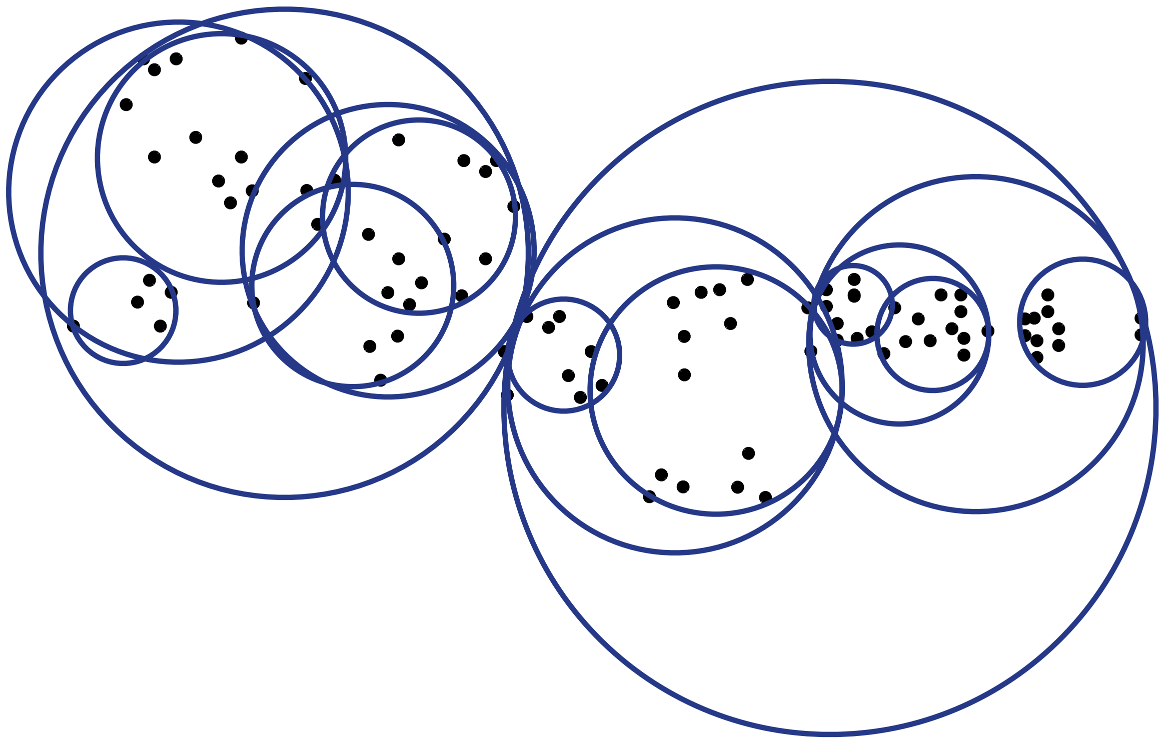
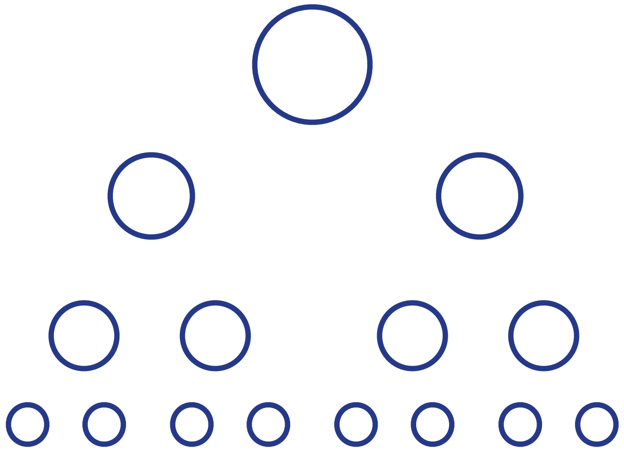


Organize cells into a tree:



Explore using branch-and-bound

On many core?

- Conditional exploration
- Irregular memory accesses/little mem re-use
- Ouch.

Problem setting

Database $X = \{x_1, x_2, \ldots, x_n\}$

Query q (or many queries Q)

Metric $\rho(\cdot, \cdot)$

Goal: return x_i minimizing $\rho(q, x_i)$

$(\forall q \in Q)$

Brute force search

For each query $q \in Q$, perform a linear scan of X;

return the nearest.

Call this procedure $\mathbf{BF}(Q, X)$.

If $I \subset \{1, \ldots, n\}$, $\mathbf{BF}(Q, X[I])$ only considers indices I.

Parallelization of $\mathbf{BF}(Q, X)$

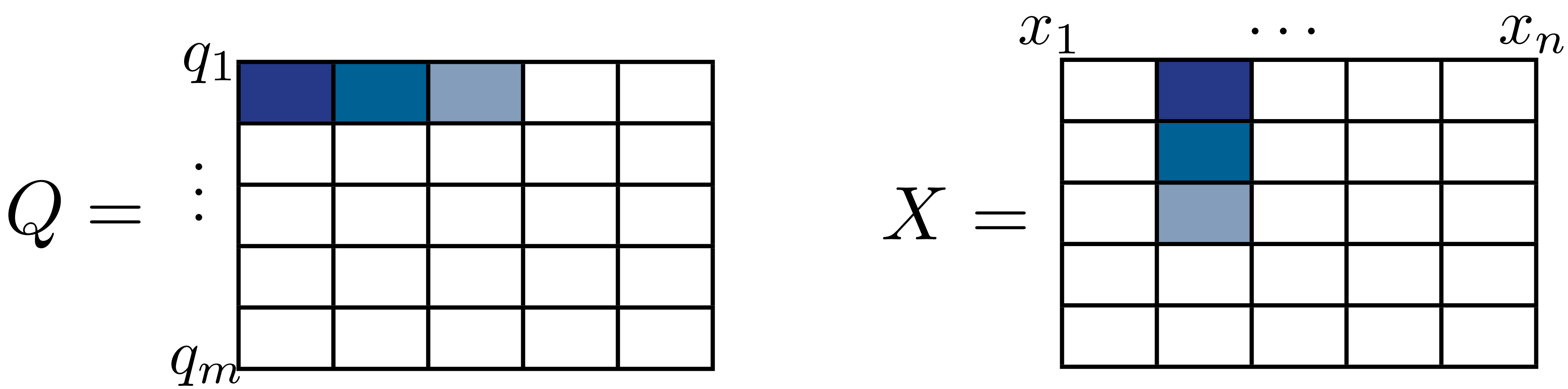
$\mathbf{BF}(Q, X)$ ↕ matrix-matrix multiplication

$\mathbf{BF}(q, X)$ ↕ matrix-vector multiplication

Parallelization of both is incredibly well-studied

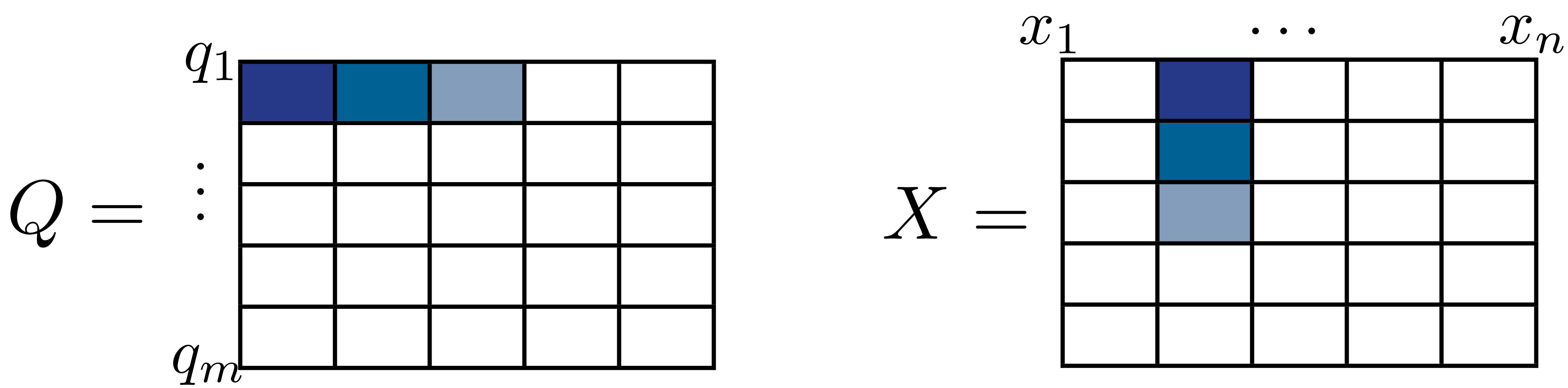
Parallelization of $\mathbf{BF}(Q, X)$

1. Compute distances via block decomposition

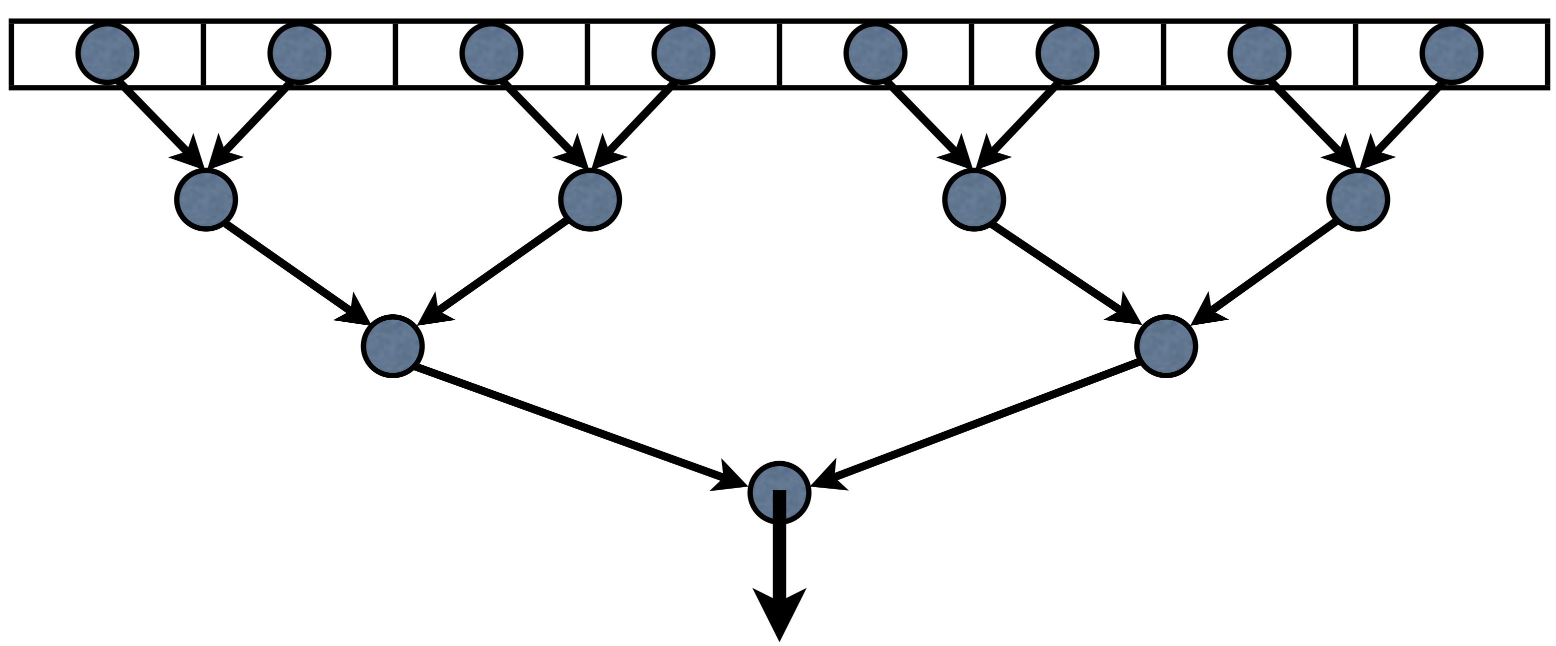


Parallelization of $\mathbf{BF}(Q, X)$

1. Compute distances via block decomposition



2. For each query, do a parallel-reduce on the distances



But..

Work is $O(n)$ per query.

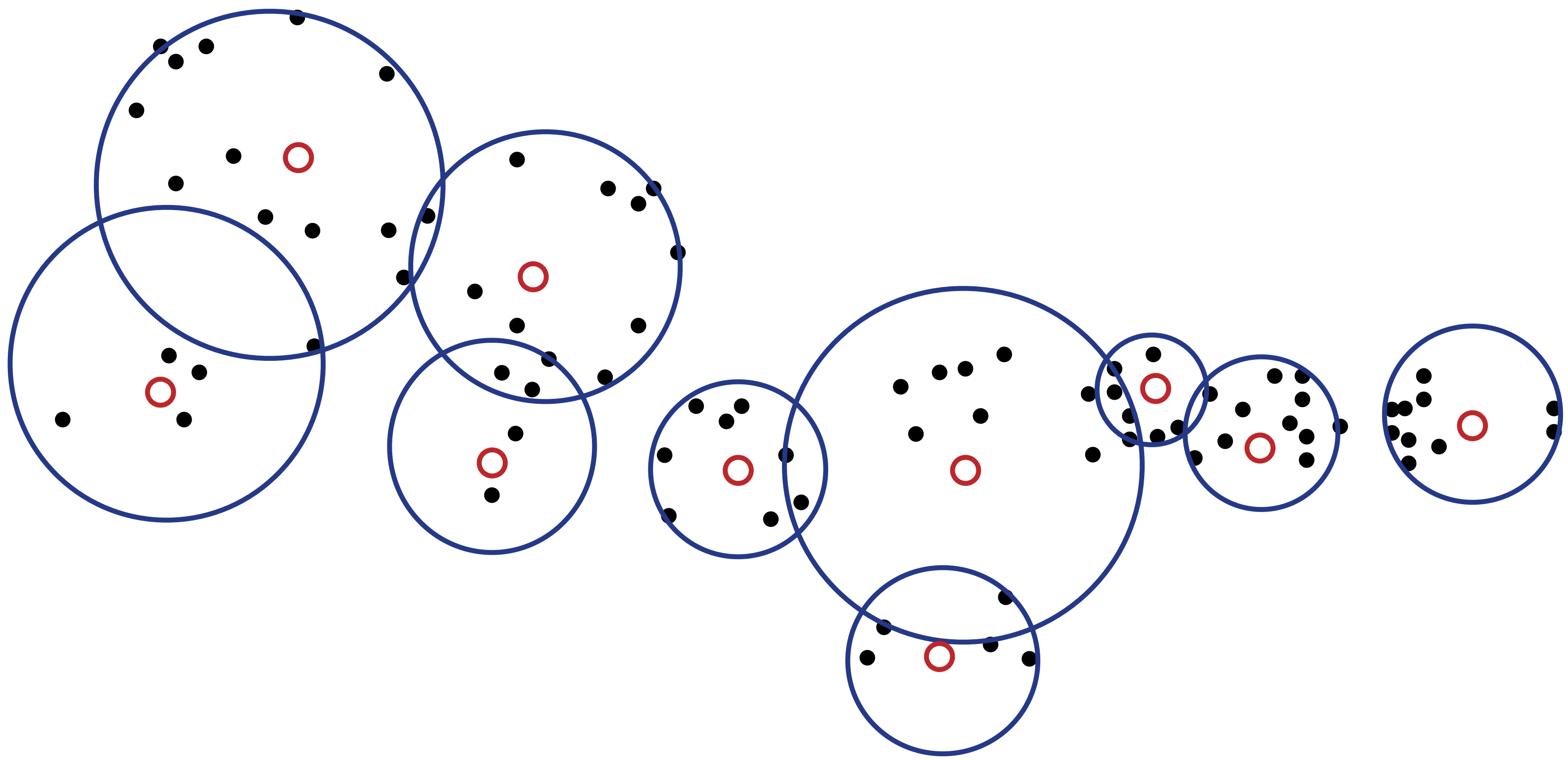
But..

Work is $O(n)$ per query.

This project:

- Reduce the work to roughly $O(\sqrt{n})$ per query
- Maintain the computational structure of $\mathbf{BF}(Q, X)$

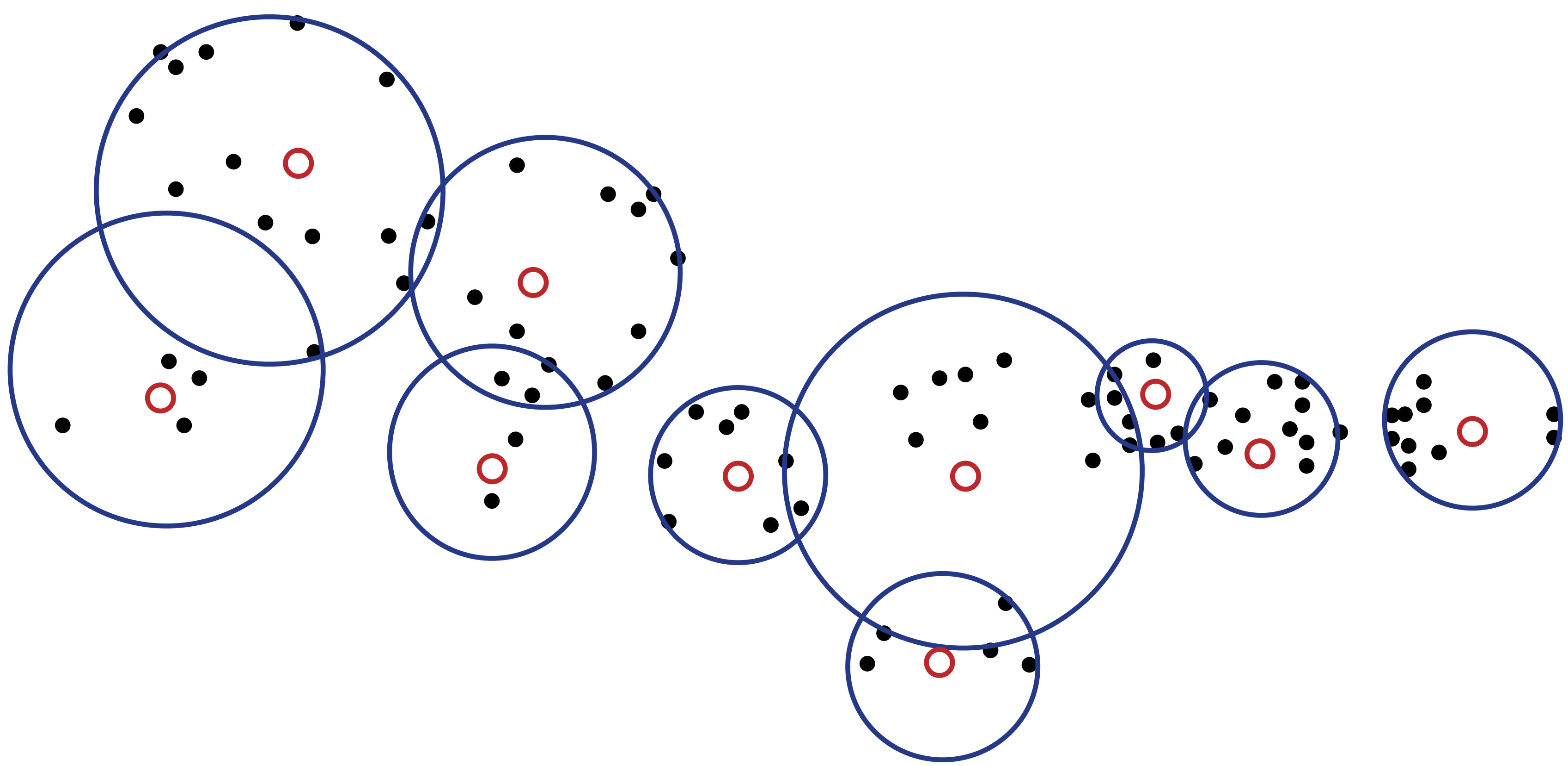
Random ball cover - data structure



r random representatives

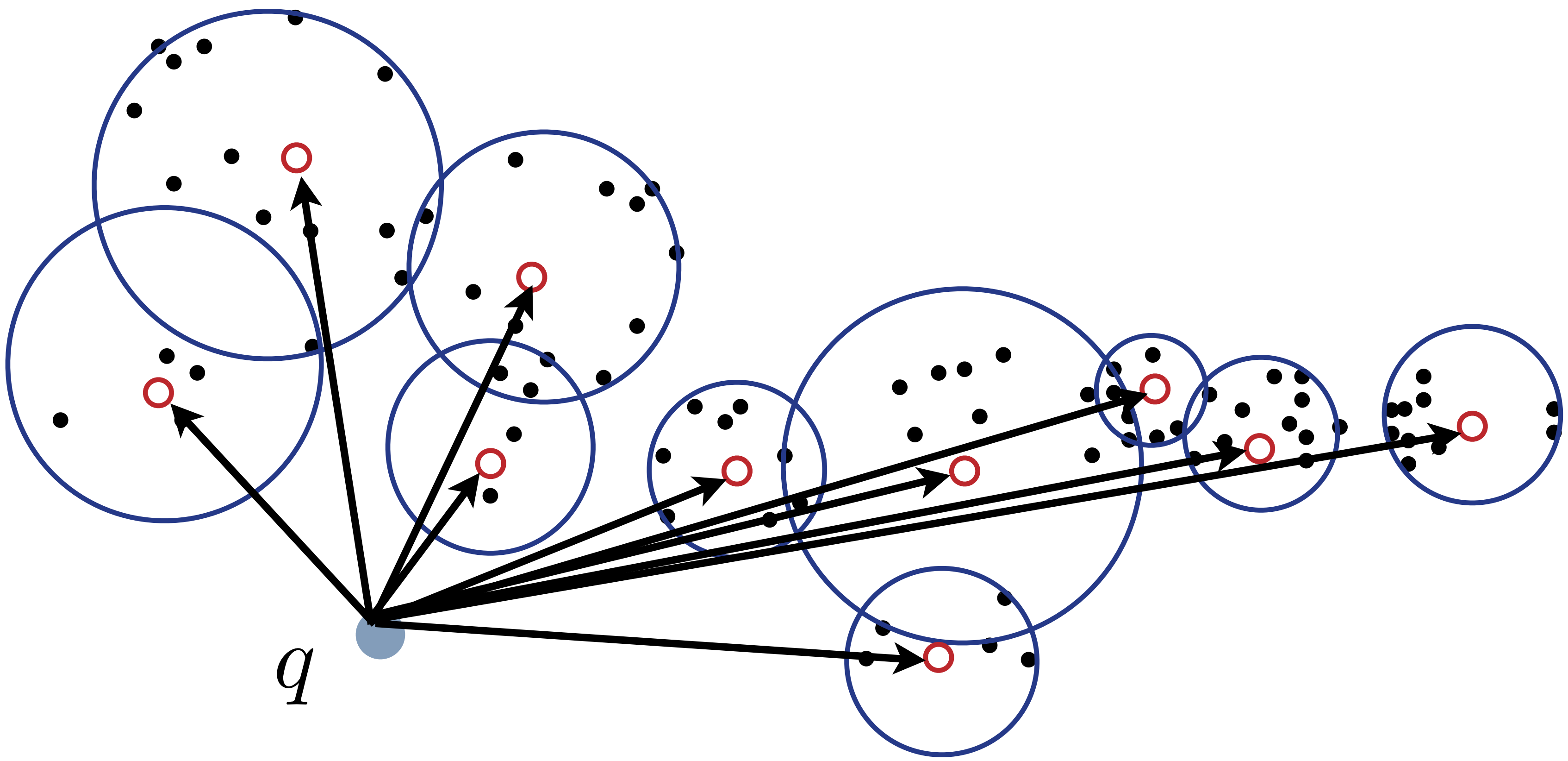
ball around representatives containing s points

Random ball cover - data structure



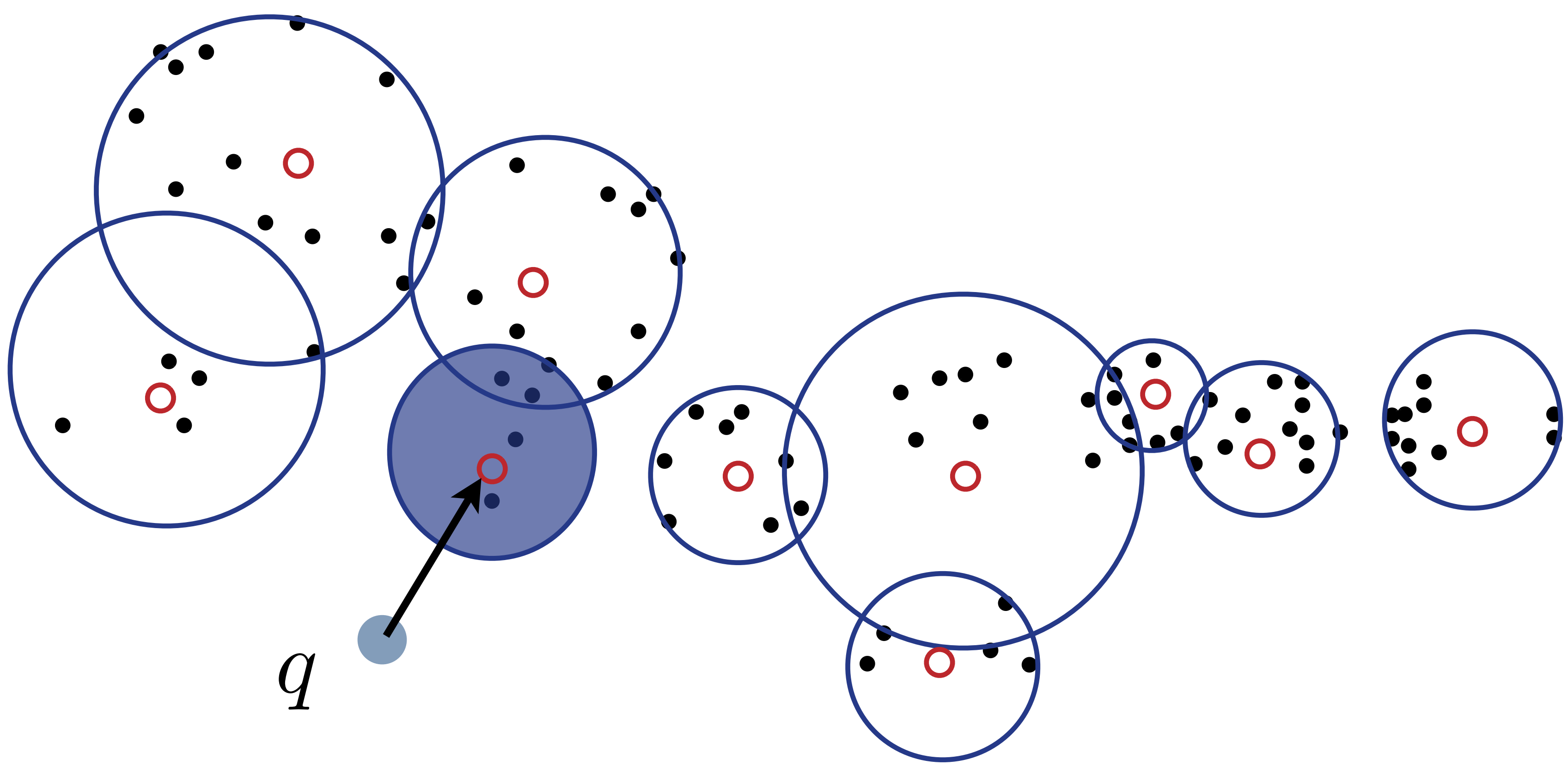
Notation: L_r - indices of points owned by rep r.

One-shot search algorithm



1. compute nearest representative

One-shot search algorithm cont.



2. find nearest point within set covered by nearest representative

One-shot algorithm: restatement

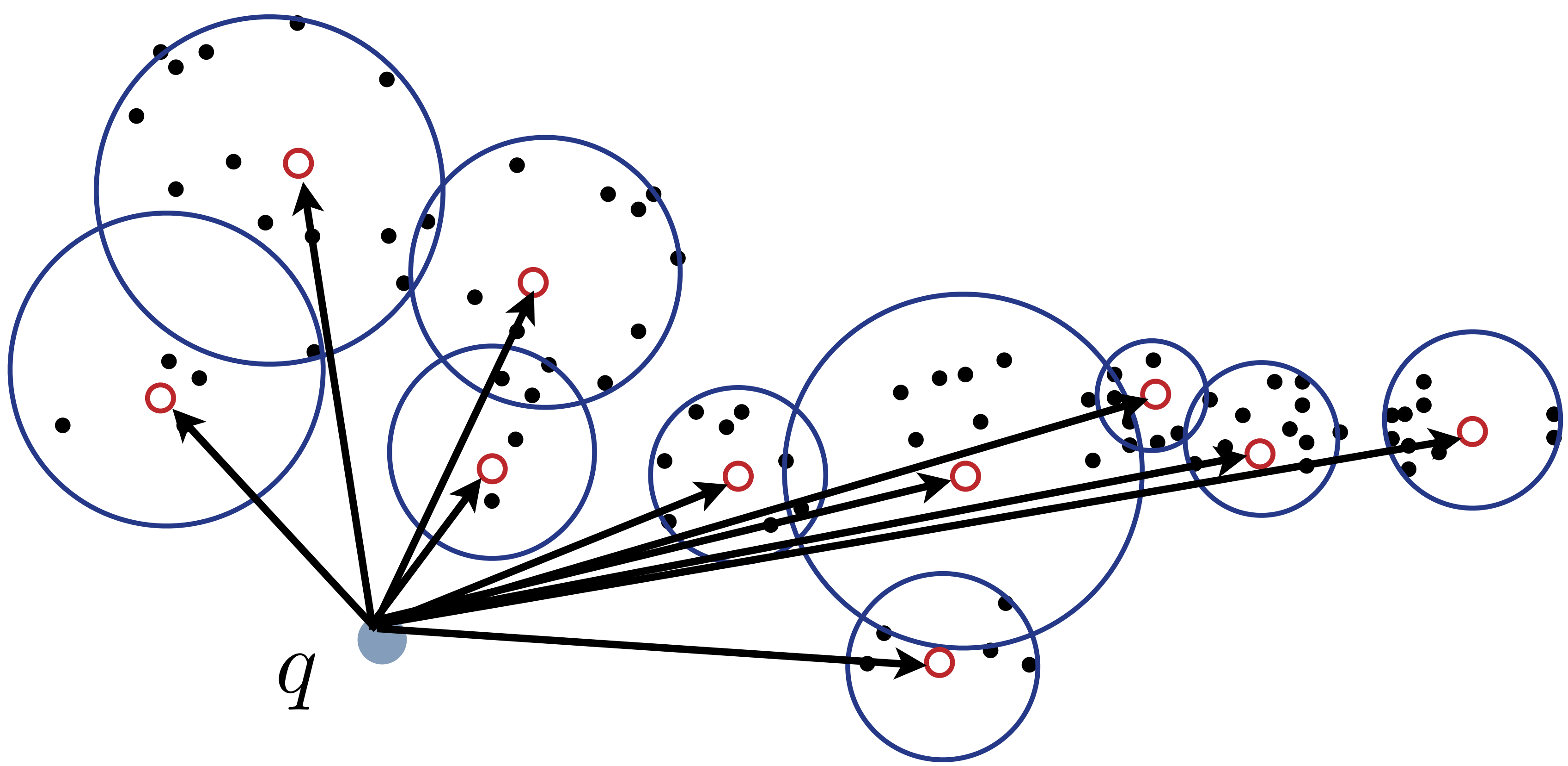
Call **BF**(q, R); get rep r back.

Call **BF**$(q, X[L_r])$.

i.e. two brute force searches

(later, we'll see that each is roughly $O(\sqrt{n})$)

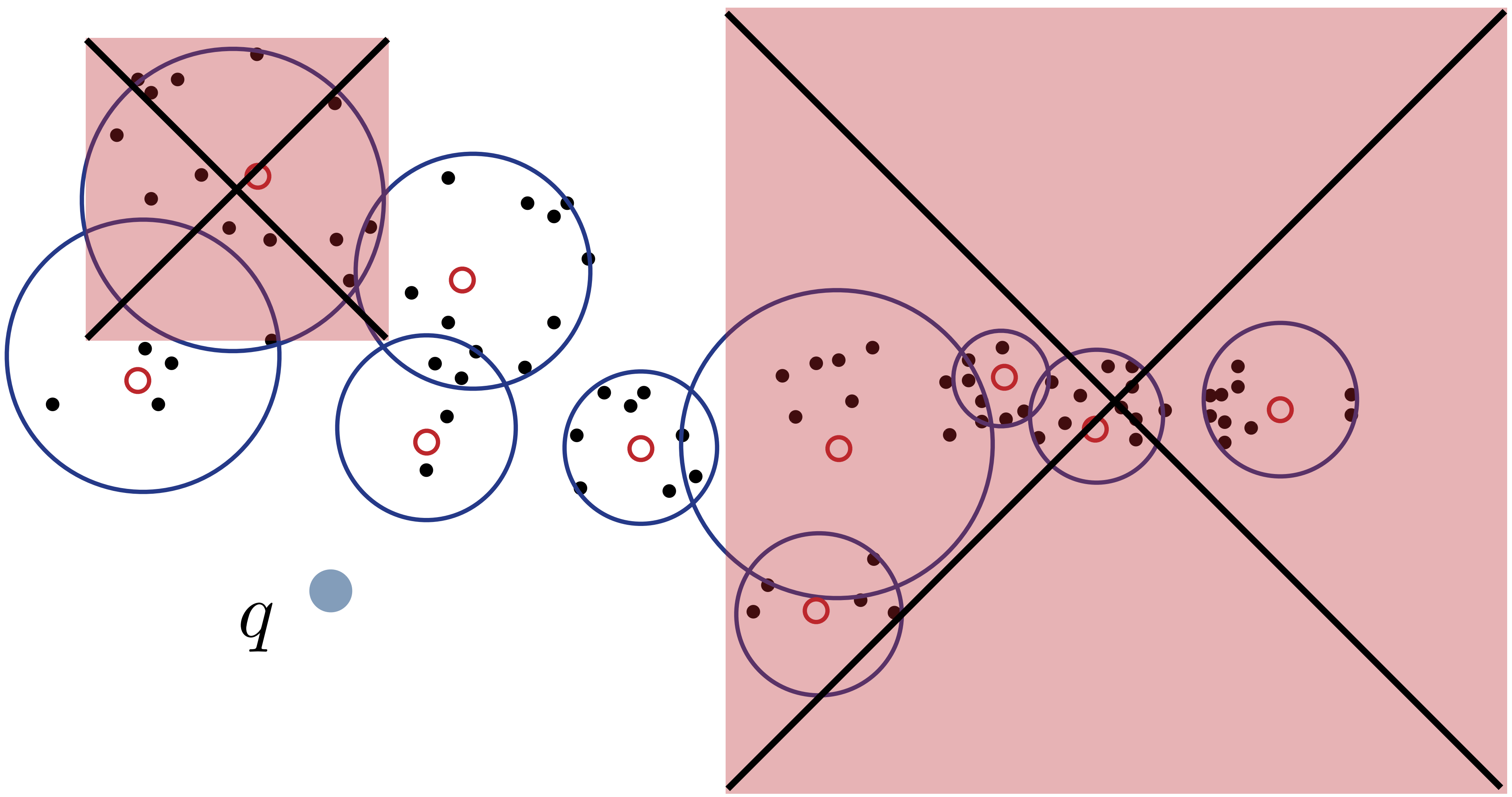
Exact search algorithm



1. compute nearest representative

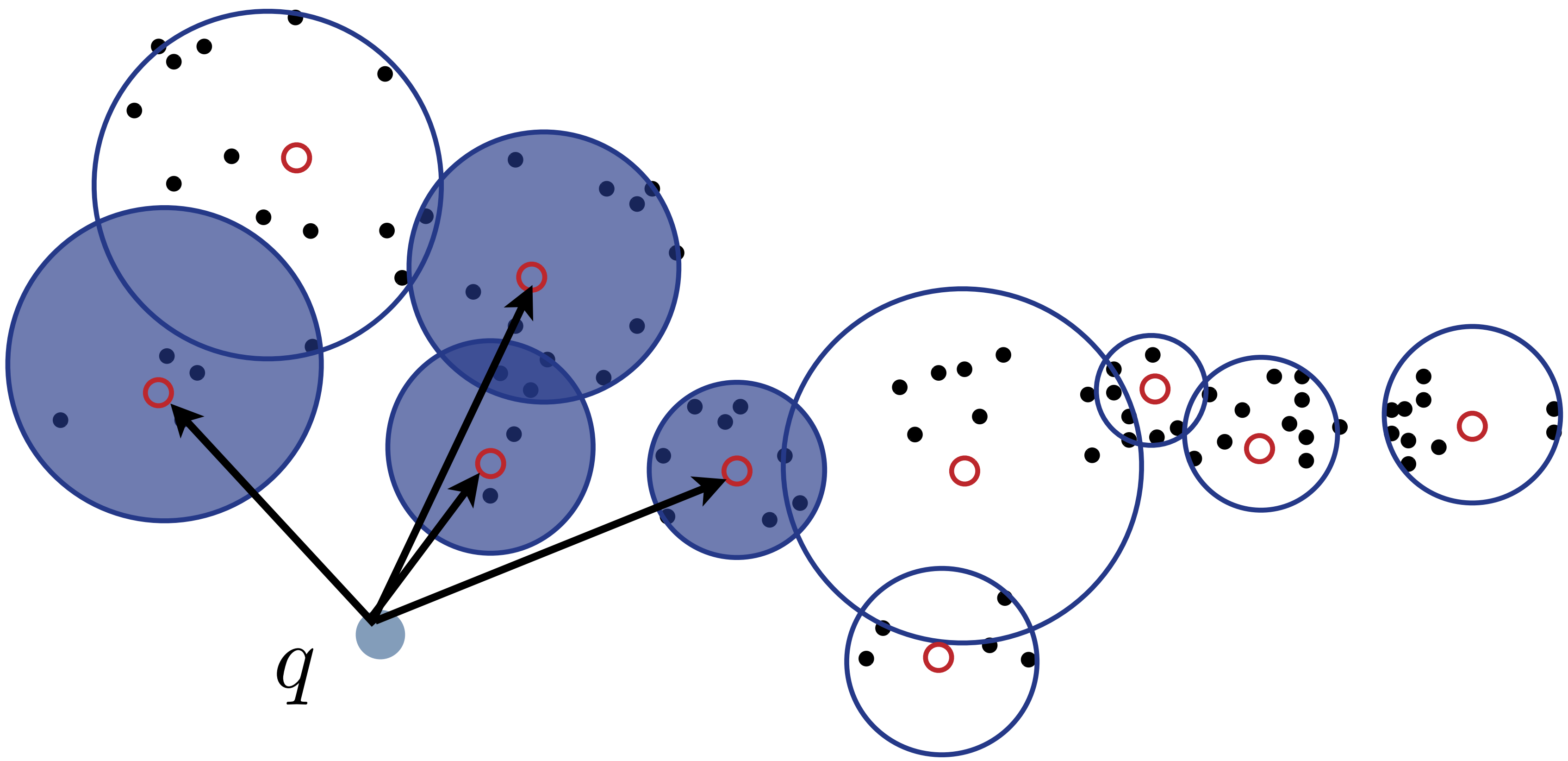
(same as before)

Exact search algorithm



2. prune out as many balls as possible

Exact search algorithm



3. Search the rest and return the nearest.

Exact search restatement

Call **BF**(q, R); get rep r back.

Compute lists $L_1, \ldots, L_t$ that can not be pruned.

Call **BF**$(q, X[L_1 \cup \cdots \cup L_t])$.

Theory

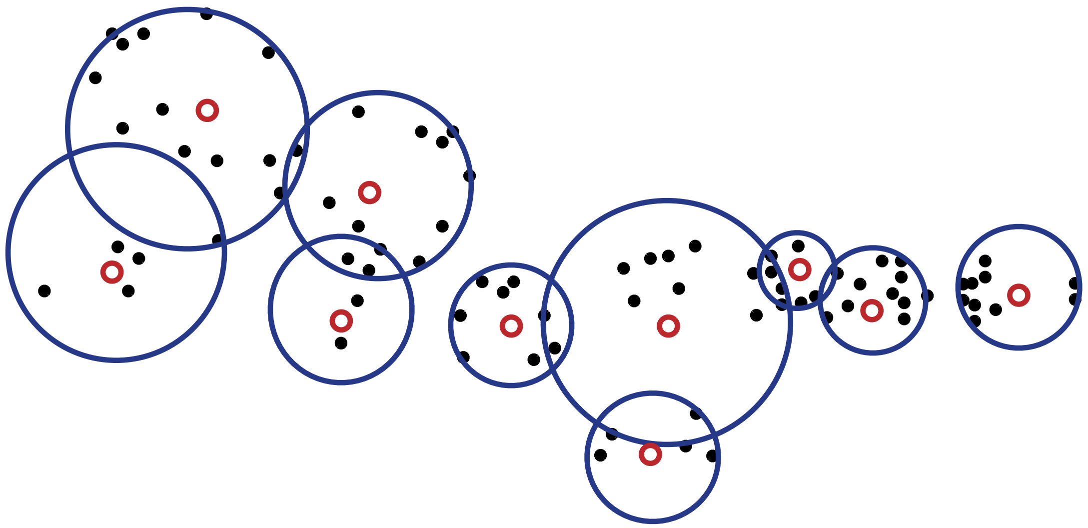
Both algs have

- $O(\sqrt{n})$ dependence on the data
- some dependence on the *growth rate c,*

$$\text{where } c \approx 2^{\text{intrinsic dim}}.$$

Exact search alg

Guaranteed to find the exact NN; but how long does it take?

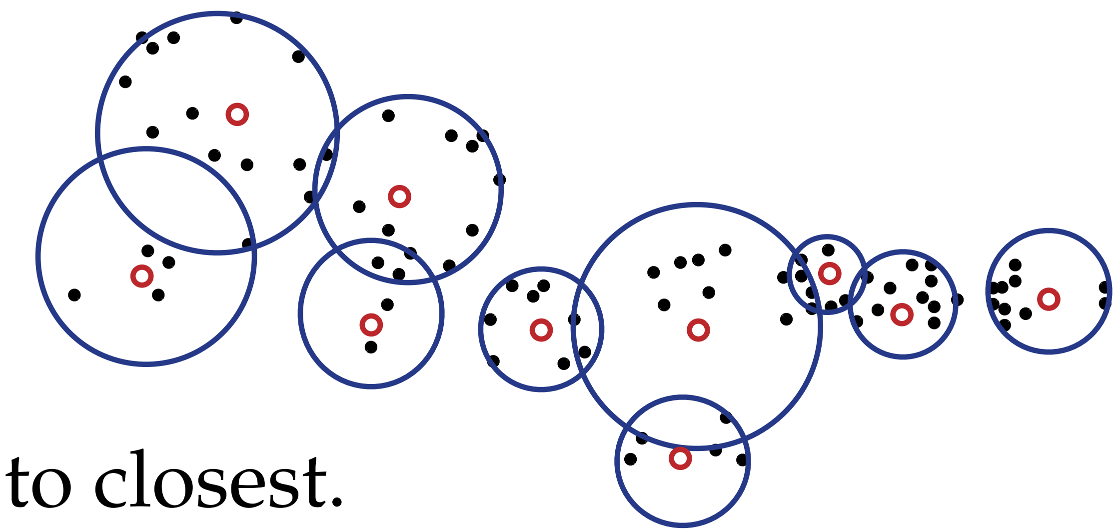


Data structure details:

- Each rep r chosen independently w.p. p.
- Each $x \in X$ assigned to nearest r.

(think of $p \approx \frac{c}{\sqrt{n}}$)

Exact search alg



Alg redux

- Call **BF**(q, R); let γ be dist to closest.
- Let $r_1, \ldots, r_t$ be the reps that sat $\rho(q, r_i) \leq 3\gamma$.
- Call **BF**$(q, X[L_1 \cup \cdots \cup L_t])$.

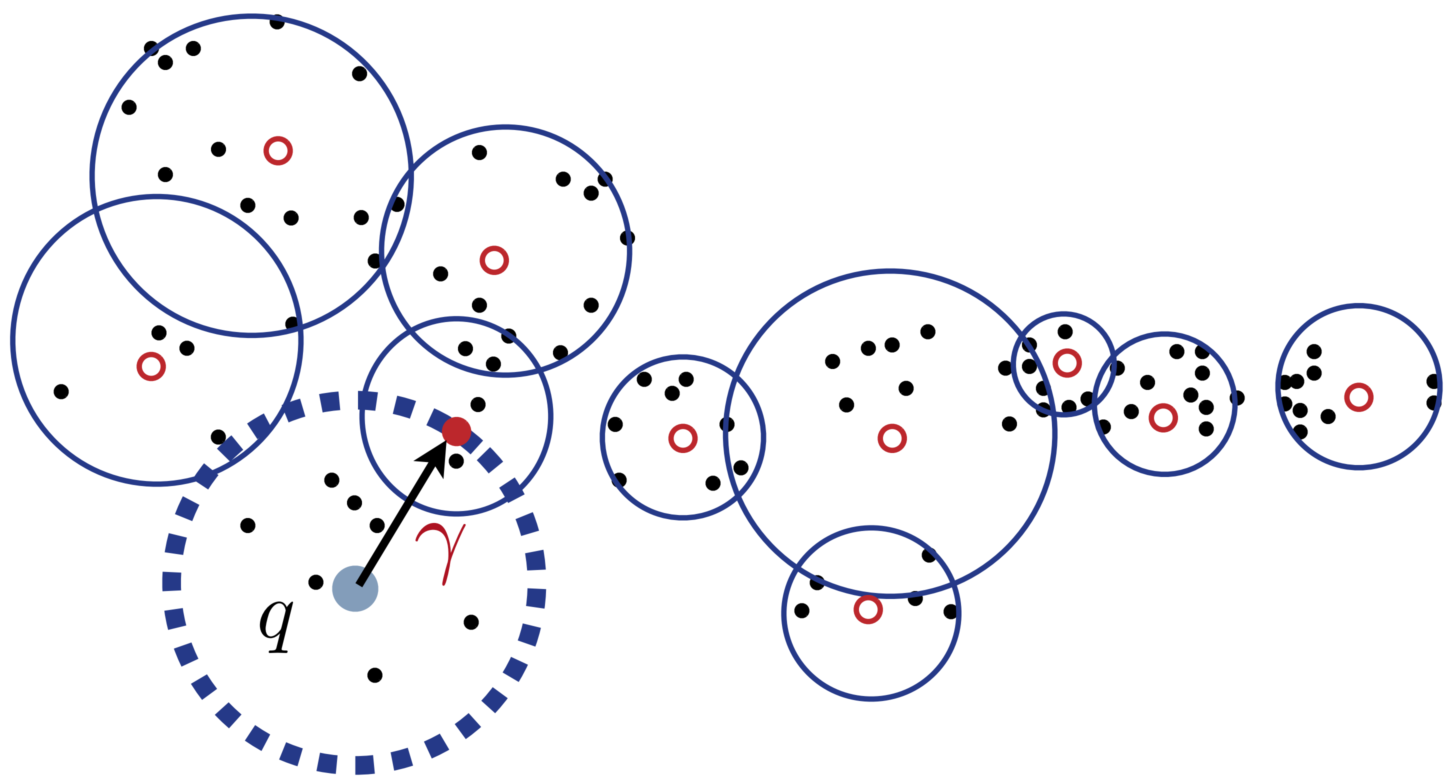
First step has expected complexity $1/p$.

Third step: want to bound $|L_1 \cup \cdots \cup L_t|$

Exact search alg

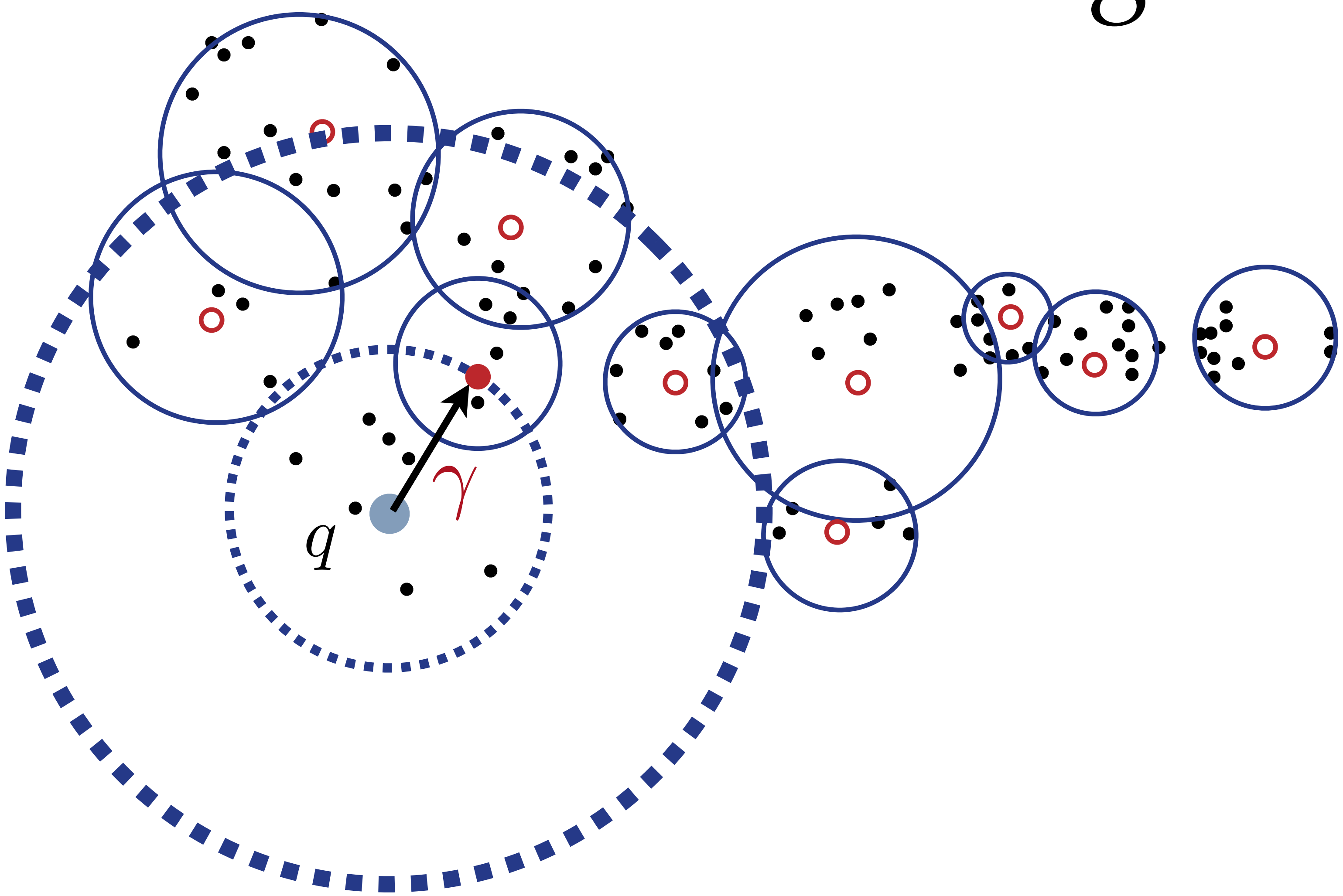
Let $\gamma = \rho(q, r_q)$ (dist to q's NN among R).

How many points are in $B(q, \gamma)$?



In expectation, about $1/p$

Exact search alg



Recall that all relevant reps r sat $\rho(q, r) \leq 3\gamma$.

⋮

Can show that the NN of q must lie in $B(q, 7\gamma)$.

Exact search alg

Setting $p = O(c^{3/2}/\sqrt{n})$,

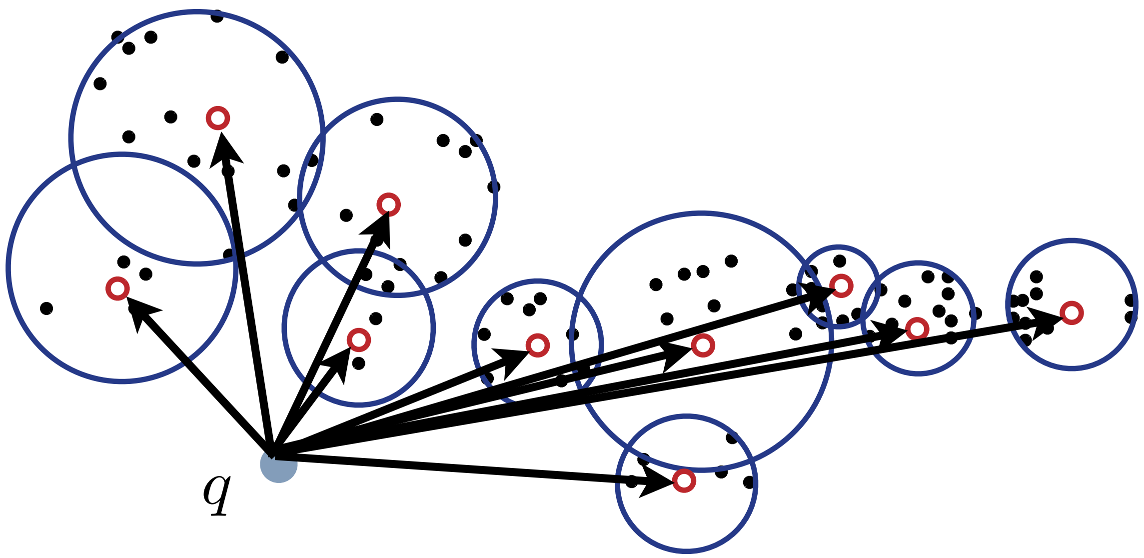
and applying the growth rate condition,

get bound on $|B(q, 7\gamma)|$

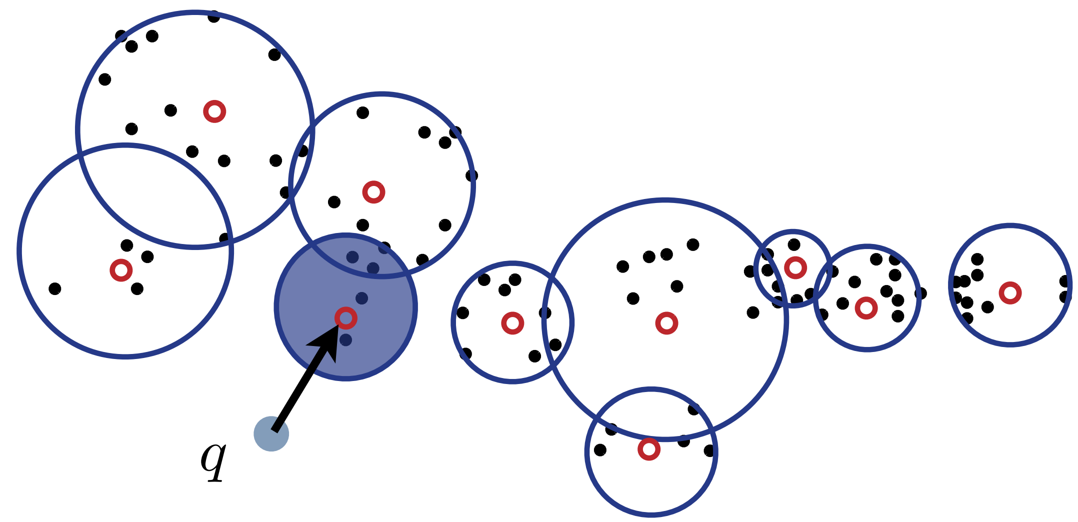
the expected run time is $O(c^{3/2}\sqrt{n})$.

One shot alg

Recall:



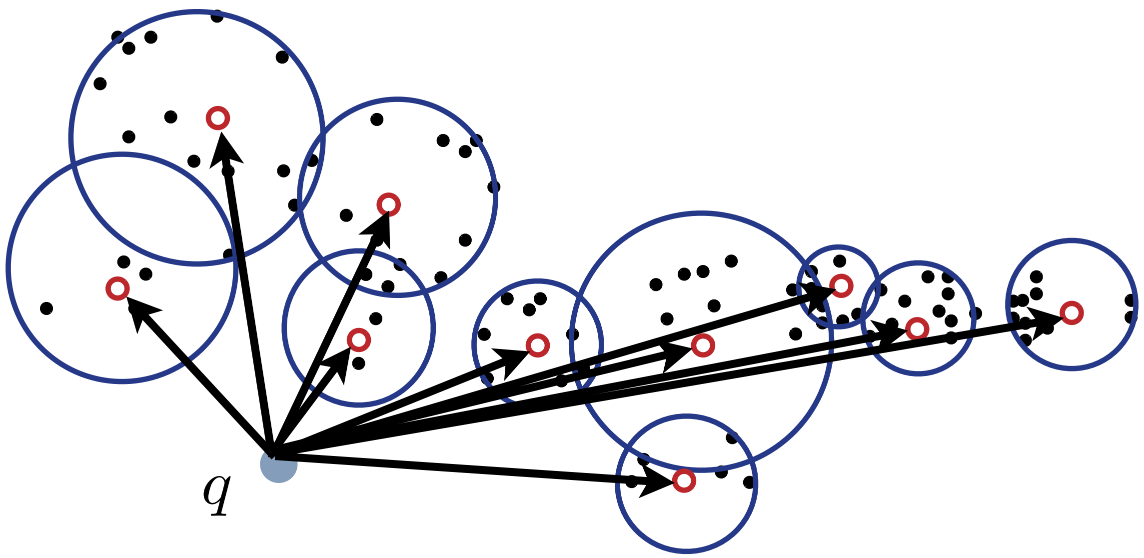
1. Call $\mathbf{BF}(q, R)$; get r_q.



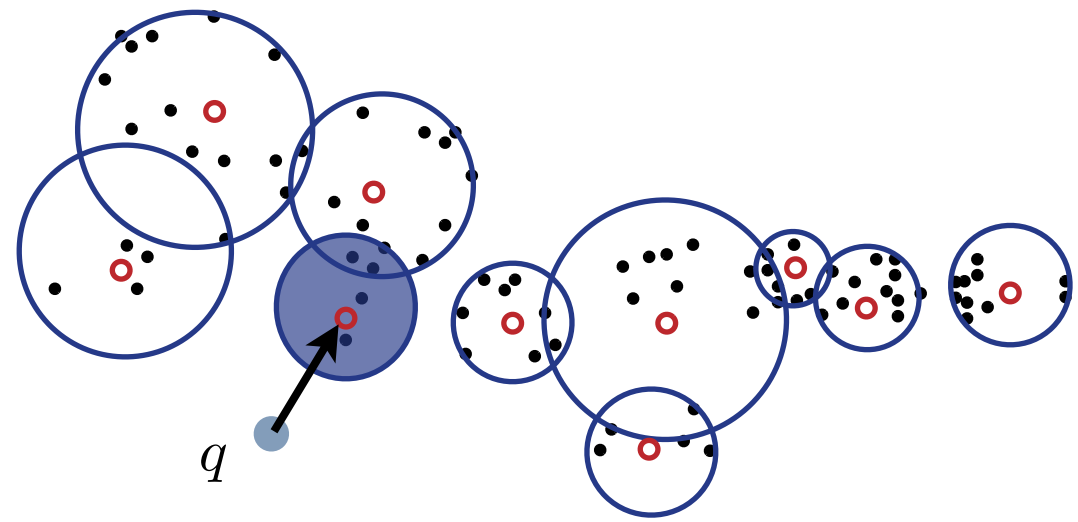
2. Call $\mathbf{BF}(q, X[L])$.

One shot alg

Recall:



1. Call $\mathbf{BF}(q, R)$; get r_q.



2. Call $\mathbf{BF}(q, X[L])$.

Set $n_r = s = c\sqrt{n} \cdot \sqrt{\ln \frac{1}{\delta}}$.

Then the one-shot alg is correct w.p. $\geq 1 - \delta$.

Experiments on 48 cores

Experiments show two things:

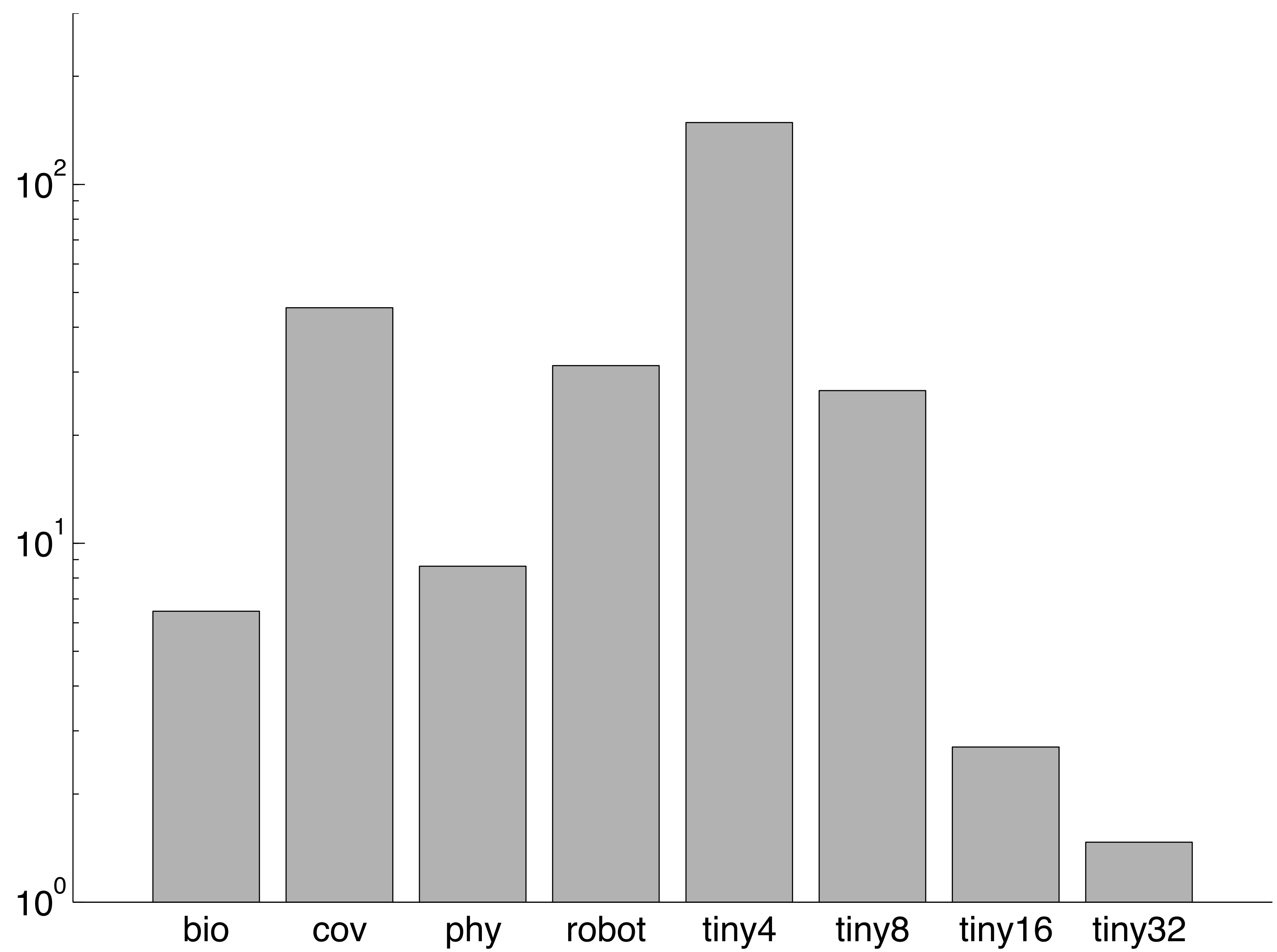
1. The RBC search alg reduces the work for NN (supports the theory)

2. It parallelizes effectively (supports the design choices)

Data

Name	Num pts	Dim
Bio	200k	74
Covertype	500k	54
Physics	100k	78
Robot	2M	21
TinyIm	10M	4-32

Exact search results



Actual times for 10k queries

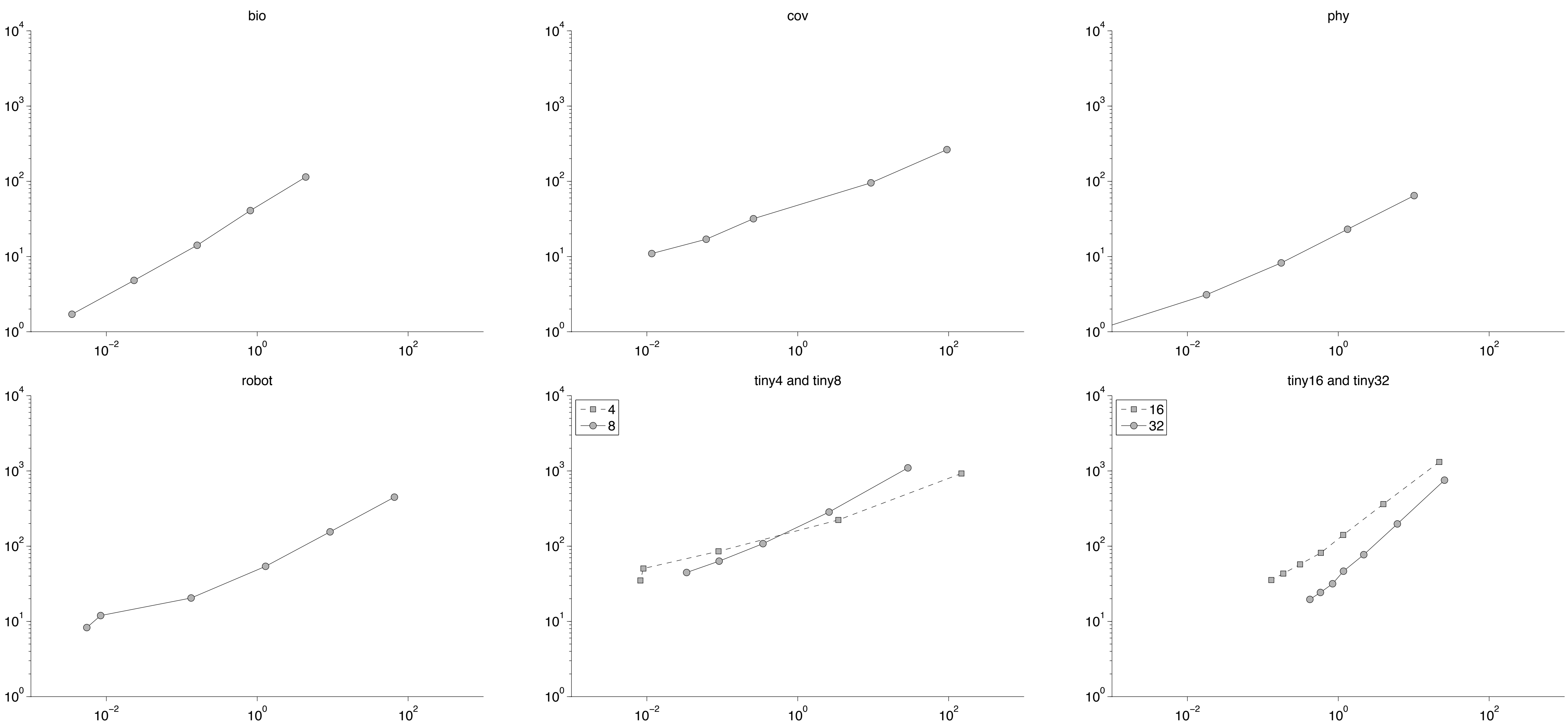
Data	Time in seconds
Bio	.4s
Cov	.4s
Phy	.3s
Robot	1.2s
Tiny4	.7s
Tiny8	.8s
Tiny16	3.0s
Tiny32	7.5s

One-shot search

The parameter allows you to trade-off between speed and quality.

Error measure: rank of returned point.
e.g. rank-0 is exact NN, rank-1 is 2nd NN, ..

One-shot search results



GPU results

Data	Speedup (GPU)
Bio	38.1
Covertype	94.6
Physics	19.0
Robot	53.2
TinyIm4	188.4

Cover tree comparison

Data	Cover Tree	RBC
Bio	18.9	6.4
Covertype	0.4	1.1
Physics	1.9	1.7
Robot	4.6	5.1
Tiny4	0.5	1.2
Tiny8	14.6	3.3
Tiny16	178.9	25.1
Tiny32	387.0	67.9

Conclusion

- Simple, high performance method
- Broadly applicable
- Theoretically sound
- Good implementations available