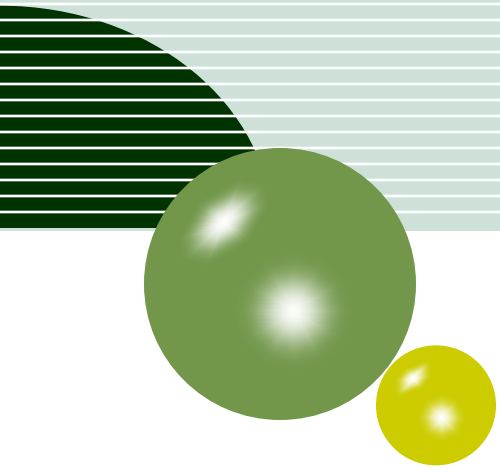
A Fast Indexing Method for Protein 3-D Structure Searching

Tetsuo Shibuya

Human Genome Center, Institute of Medical Science

University of Tokyo



Joint work with **Genki Terashi** and **Mayuko Takeda-Shitaka**

Kitasato University

Today's talk

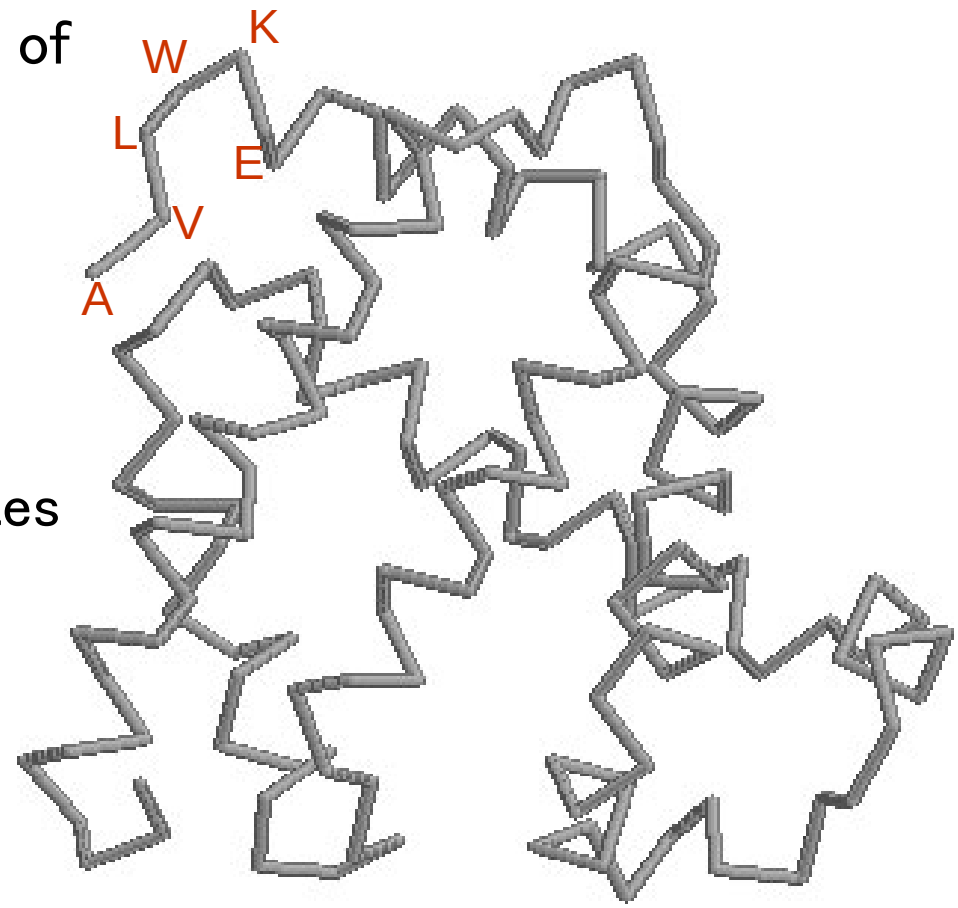
- **LB3D: A Fast Indexing Algorithm for Protein 3-D Structure Searching**
  - $O(n \log n)$ preprocessing
  - $O(n)$ space for indexing
    - Just a sorted array (like the suffix array)
  - Practically very fast query
    - Average-case $O(m + {}^{n}/_{\sqrt{m}})$-time query
      - analyzed based on the ***FJC*** model
    - Practically faster than previous searching algorithms even if we include the preprocessing time
      - though theoretically worse than the best-known $O\left(m + {}^{n}/_{m^{1-\varepsilon}}\right)$ -time searching bound [Shibuya 2010]

n: database size (#bases)
m: query size (#bases)

Genki Terashi, Tetsuo Shibuya, and Mayuko Takeda-Shitaka (2012) "LB3D: A Protein Three-Dimensional Substructure Search Program Based on the Lower Bound of a Root Mean Square Deviation Value," *J. Comput. Biol.*, 19(5).

Protein Structure

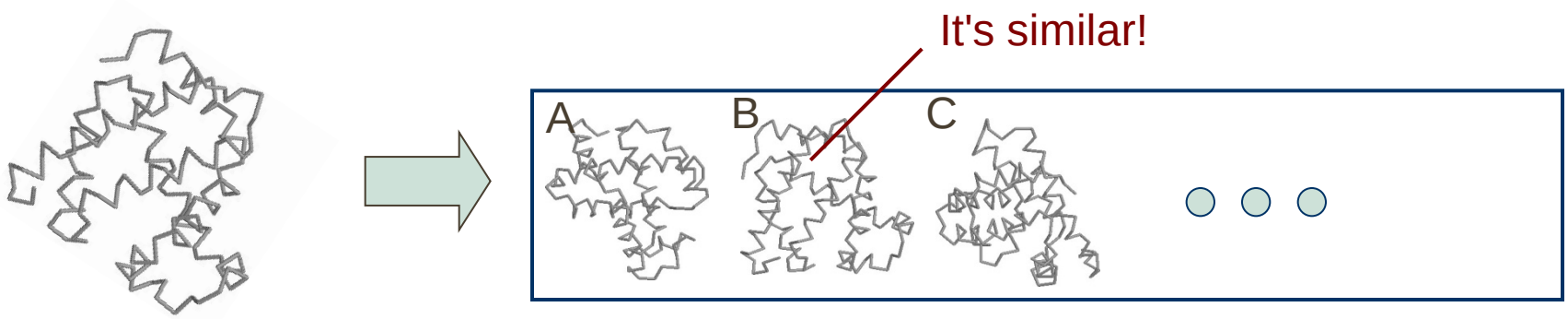
- **A protein**
  - A chain molecule consisting of 20 kinds of amino acids
  - Folded into some structure
- **Primitive 3-D structure Representation**
  - A sequence of 3-D coordinates of the C_α atoms (or the backbone atoms)



Motivation

- **Structurally similar proteins**
  - Tend to have similar functions even if not similar at the residue level
  - Important for functional analysis
- **PDB (Protein Data Bank)**
  - 94,000~ entries (Sep 24, 2013)
    - Increasing rapidly (by 20% per year)

→ **Faster searching algorithms desired!**



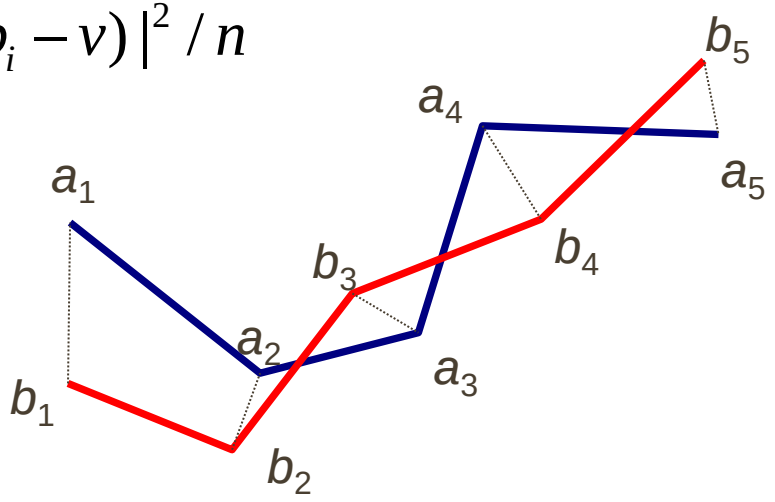
Query: Protein structure

Protein Structure Database

How to Compare Two Structures?

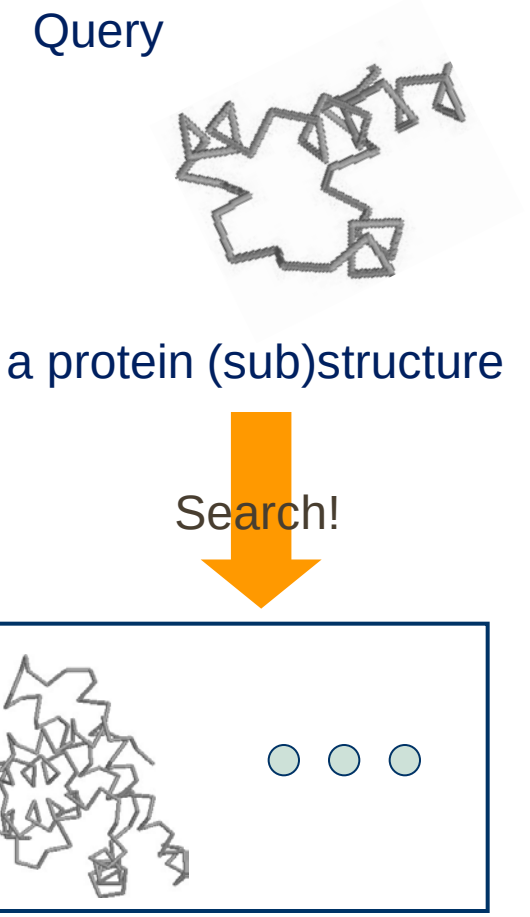
- **RMSD: Root Mean Square Deviation**
  - The most widely-used similarity measure for protein structures
  - Computable in $O(n)$ time using SVD [Kabsch '76][Umeyama '91]
    - n: chain length
    - Correspondence of atoms is given

$$RMSD(A,B) = \min_{R,v} \sqrt{\sum_{i=1}^{n} | a_i - R \cdot (b_i - v) |^2 / n}$$



The Most Fundamental Problem

- **Database**
  - Protein 3-D structures in a database
- **Query**
  - A (sub)structure
- **Output**
  - All the similar substructures in the database
    - *i.e.,* RMSD ≤ some given bound c
    - No insertions/deletions



History of the problem

- **Naive $O(nm)$ algorithm**
  - Compute RMSDs for all the $n - m + 1$ substructures of length m in the database
- **Theoretical worst-case $O(n \log m)$ algorithm** [Schwartz et al. '87]
  - Utilized the FFT-based convolution technique
  - Practically not so faster than the naive algorithm
- **Average-case $O(n)$ algorithm** [Shibuya RECOMB 2009]
  - Worst-case: $O(nm)$
  - Practically 5-100 times faster than the above algorithms
- **Average-case $O\left(m + {}^{n}/_{m^{1-\varepsilon}}\right)$ algorithm** [Shibuya 2010]
  - Average-case complexity analyzed based on the FJC model
  - Worst-case: $O(nm)$
  - Not a practical algorithm, though

n: database size (#bases)
m: query size (#bases)

Previous Indexing Algorithms

- **PSIST** [Gao, Zaki 2005]
  - Utilizing the suffix tree, but cannot deal with the RMSD
- **Geometric suffix tree** [Shibuya, J.ACM 2010]
  - An extension of the suffix tree that supports the 3-D protein structure searching based on the RMSD
  - Too large construction time: $O(n^2)$
- **Some theoretical insights** [Shibuya 2009]
  - Average-case $O(m + {}^{n}/_{\sqrt{m}})$-time query after $O(n \log n)$ preprocessing
    - Not a practical algorithm, though
    - Theoretically worse than the later $O\left(m + {}^{n}/_{m^{1-\varepsilon}}\right)$ algorithm [Shibuya 2010]

the same bound

Today's talk

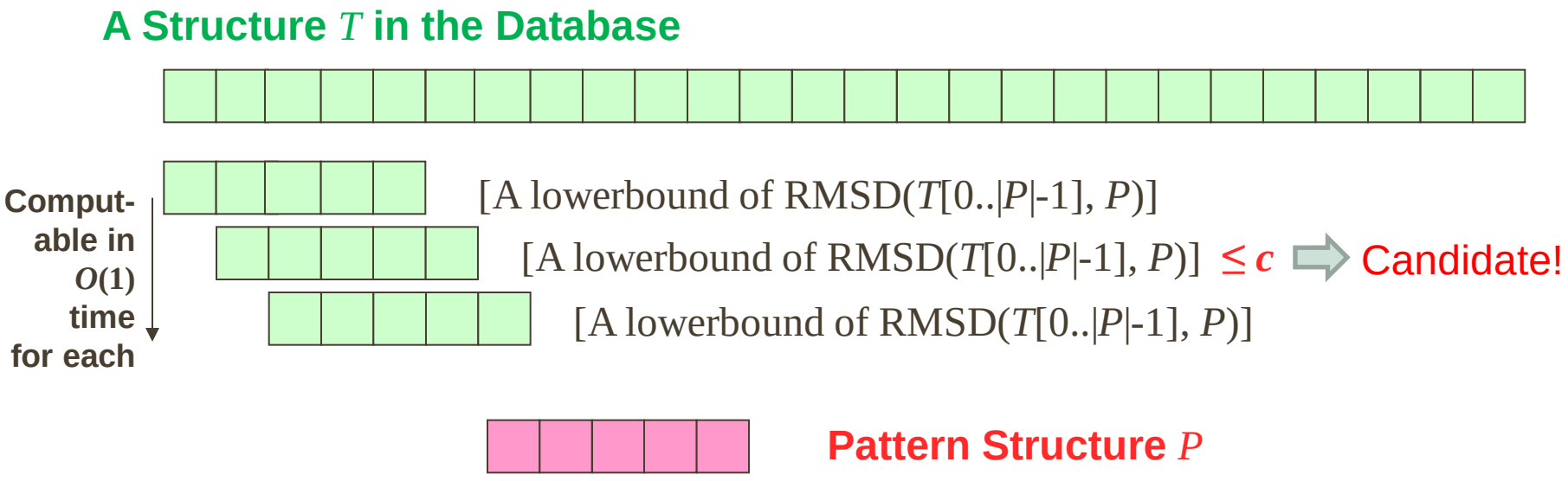
A practical algorithm that supports average-case $O(m + {}^{n}/_{\sqrt{m}})$-time query after $O(n \log n)$ preprocessing

Shibuya's $O(n)$ Algorithm

[Shibuya, RECOMB 2009]

- **Filtering-based expected linear time algorithm**
  - Compute lower bounds of the RMSDs (instead of hash values)
  - Compute the actual RMSD for only the substructures with small enough lower bounds

c.f. Karp-Rabin algorithm for ordinary strings

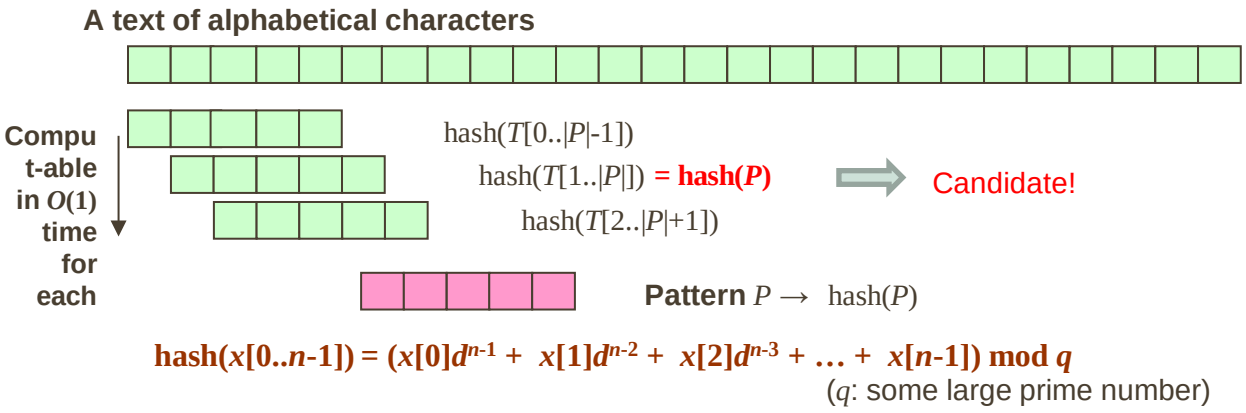


Compute the RMSD only when the lower bound is $\leq c$

Keys to the $O(n)$

- **Lower bound computation**
  - should be done in linear time (in total)
- **Expected number of candidates after filtration**
  - should be less than $O(n/m)$
    - as checking requires $O(m)$ time
  - Model: **FJC Model**

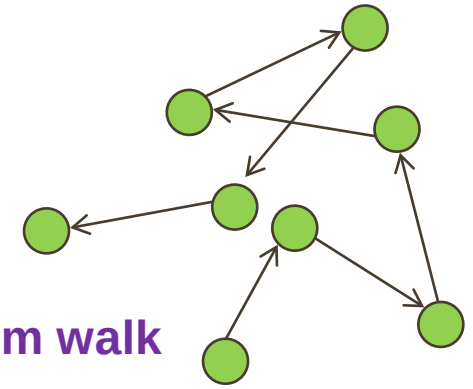
c.f. Karp-Rabin (1981)



- Hash values is computable in linear time in total
- Checking requires only $O(1)$ time
- Text: A random string

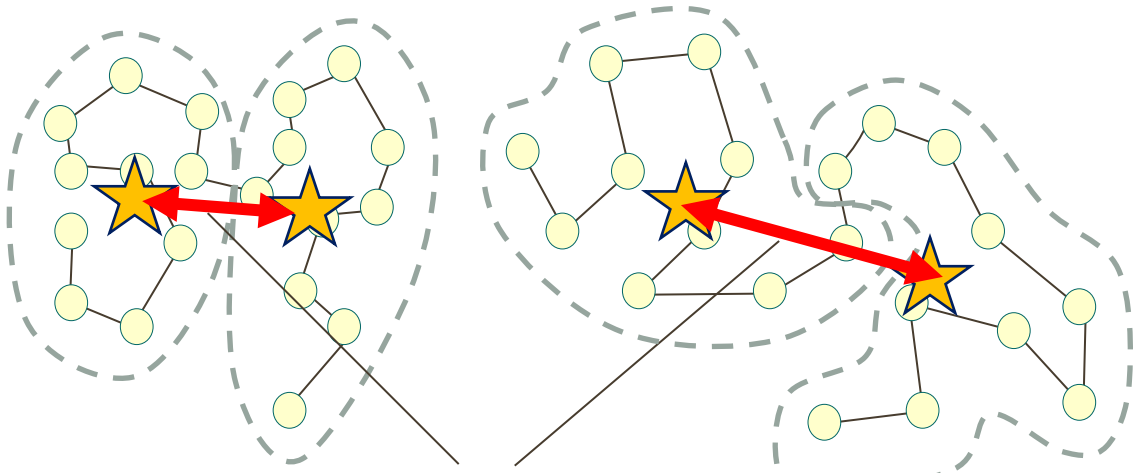
Model of 'Random' Chain-Molecule Structures

- **Freely-jointed chain (FJC) model**
  - The most basic model of chain molecules in molecular physics
    - Also called the '*Ideal chain* model' or the '*Random-walk* model'.
    - It explains behaviors of chain molecules very well, though it ignores many physical/chemical limitations
      - Collisions, edge angle limitation, existence of alpha helix/beta sheet, etc.



D_1: A Lower Bound of the RMSD

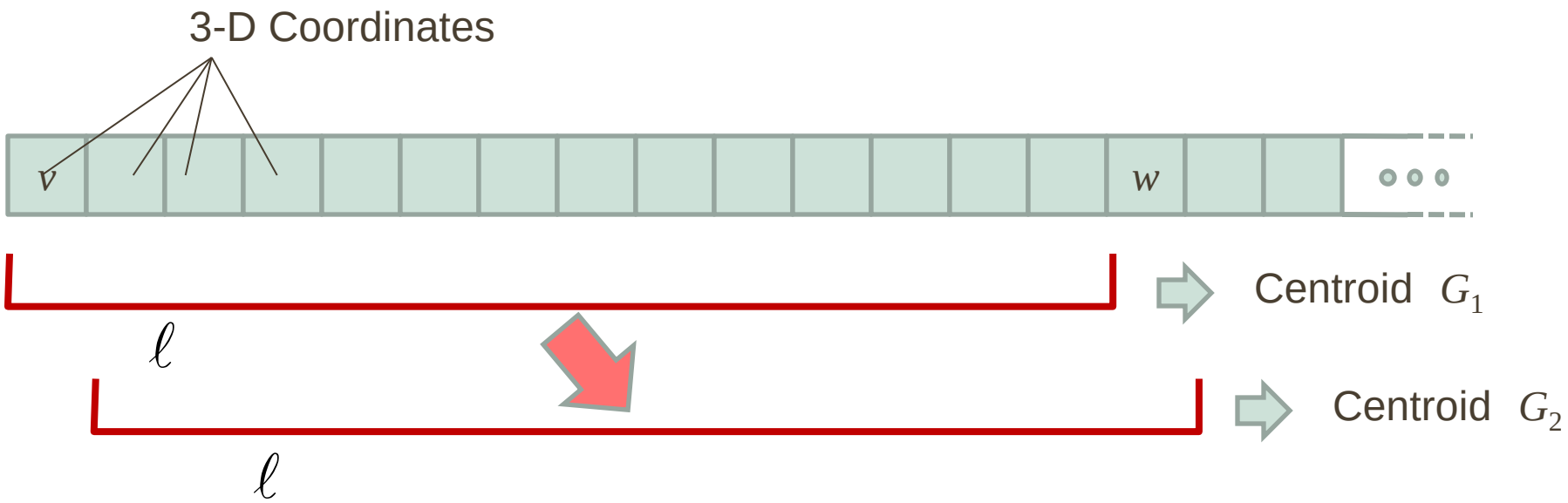
- **$D_1(P,Q)$**
  - $|H(P) - H(Q)|/2$
    - where $H(P) = |G(P[1..m/2]) - G(P[m/2+1..m])|$ (m: even number)
    - $G(S)$ is the centroid (center of mass) of structure S
    - Consider n as an even number (to simplify the discussion)
  - It is always smaller than or equal to $RMSD(P, Q)$



1/2 of the difference of
these two distances (= D_1)

D_1 can be Computed in Linear Time!

- **The centroid of each substructure can be computed in $O(1)$ time!**
  - $O(n)$ in total (n: text length)



$$G_2 = G_1 + (w - v) / \ell$$

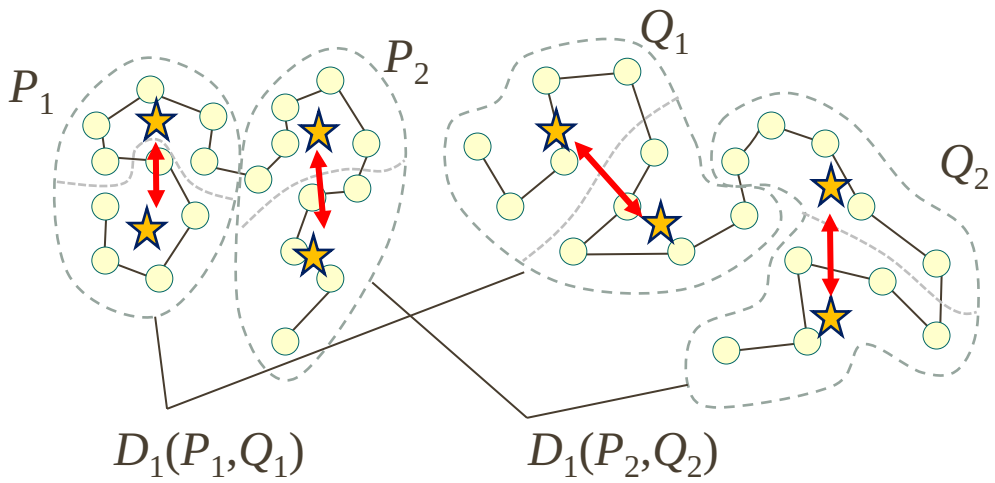
ℓ : substructure length

D_k : Extension of D_1

- $D_2(P,Q) = [\{(D_1(P_1,Q_1)^2+D_1(P_2,Q_2)^2)\}/2]^{1/2}$
  - P_1, P_2 : The first/second half of P
  - Q_1, Q_2 :The first/second half of Q
  - is also a lower bound of RMSD(P, Q)
- **Easily extendable to D_k (k>2)**
  - by dividing each structure into $2k$ parts → D_k

Computable in $O(n)$ time in total (k=const)

D_2



$|P|$, $|Q|$ is assumed to be multiples of 4 (to simplify the discussion)

The Complexity of the D_k-Based Algorithm

- **Lower bound computation**
  - $O(n)$
- **Expected number of candidates**
  - $O(n/m^{k/2})$
    - under the FJC model
- **Total expected time complexity**
  - $O(n)$ for any constant $k \geq 2$
    - $O(n\sqrt{m})$ in case $k = 1$

Experimental Results

[Shibuya 2009]

- **Target database: The whole PDB (September 5th, 2008)**
  - 52,821 entries / 244,719 chains / 38,267,694 a.a.
- **Query**
  - 100 random substructures of each specified length, taken from PDB
  - Threshold: 1 Å
- **Computation Time (sec)**
  - Average computation time of 100 random queries
  - on 1 CPU of 1200MHz UltraSPARC III on SunFire 15K

Query Length	40	80	120	160	200
#Substructures	33,722,208	21,692,707	16,134,096	12,362,509	9,559,056
#Hits	38.1	32.9	27.3	16.0	23.2
D_1	98.9	92.4	75.6	59.4	60.0
D_2	58.9	36.4	32.8	27.3	25.7
D_3	74.5	25.5	17.3	14.2	12.9
Naive	447.0	442.0	415.2	378.9	342.5
FFT	531.9	463.1	399.8	330.6	293.0

(sec)

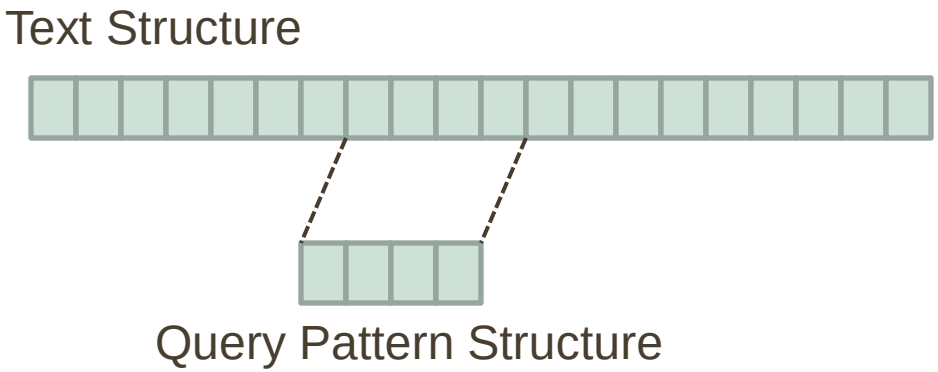
Keys to Faster Query

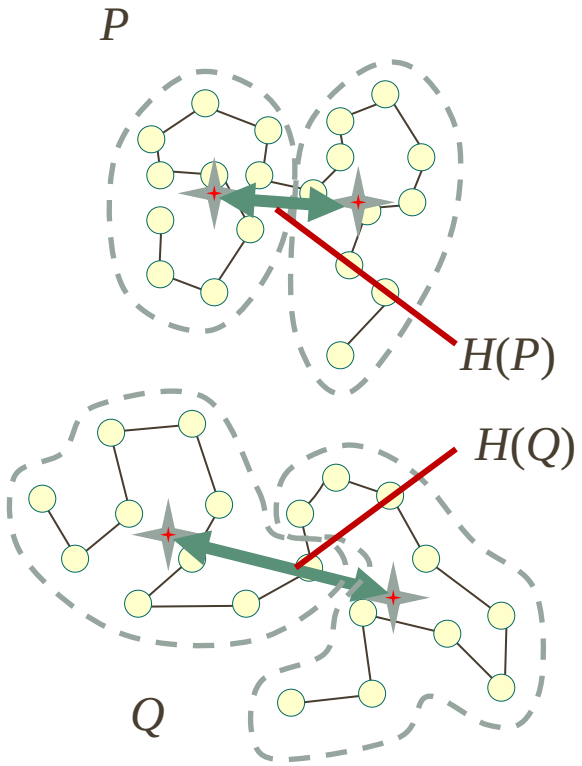
[Terashi, Shibuya, Takeda-Shitaka 2012]

- **Sort structures to enable binary search!**
  - Like the suffix arrays for strings
- **Use better lower bounds to reduce the number of candidates**
  - Still should be computable in linear time, though

Sorting Structures

- **D_1–based candidates can be searched with binary search!**
  - on a sorted array of centroid–centroid distances $H(T[i..i+m-1])$



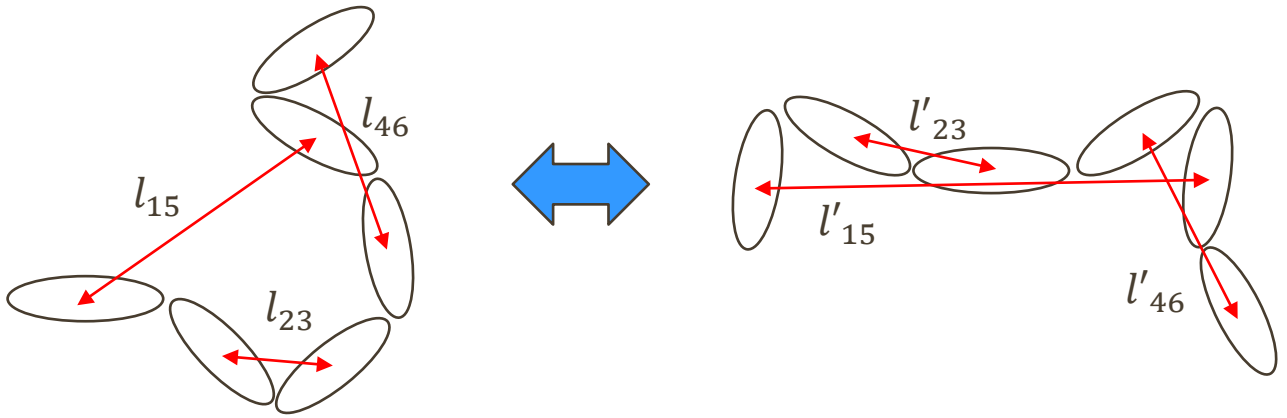


$$D_1 = |H(P) - H(Q)| / 2$$

Lower Bound Variations

- **If you divide each (sub)structures into 6 parts, there are many ways to compute lower bounds**

An example of a lower bound

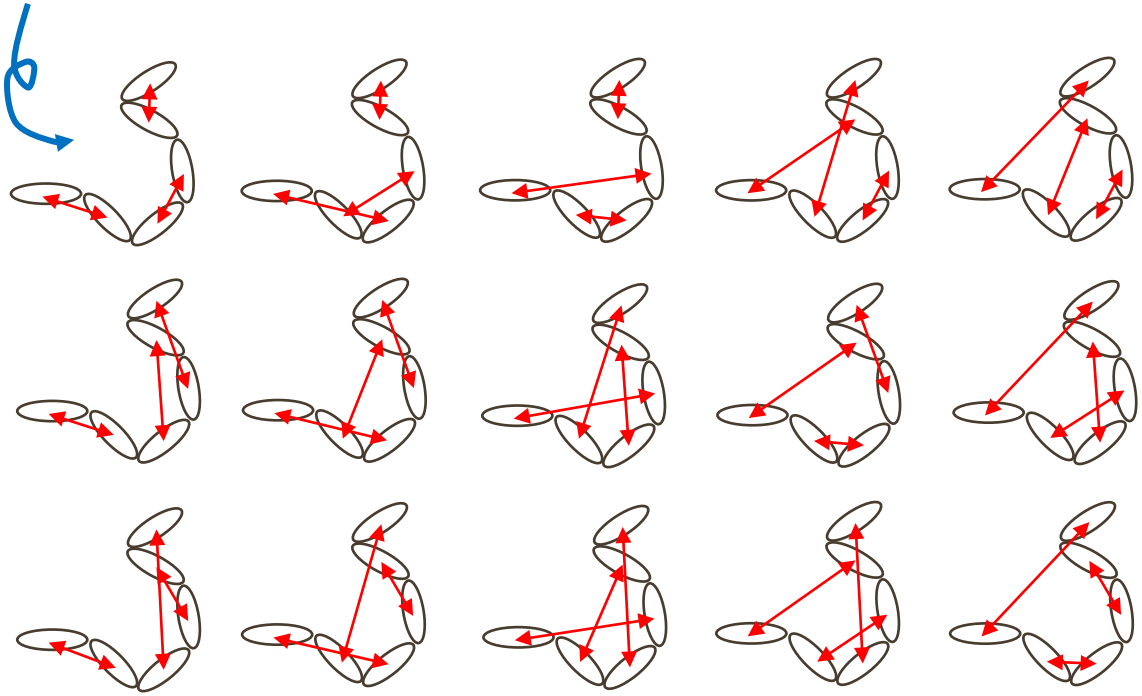


$$LB_{(1,5)(2,3)(4,6)} = \sqrt{\{(l_{15} - l'_{15})^2 + (l_{23} - l'_{23})^2 + (l_{46} - l'_{46})^2\}/3}$$

Various possible combinations, here!

Better Lower bounds

- **We can use the maximum value among all the 15 different lower bounds!**
  - Computable in linear time (very fast!)
  - D_3 is just one of these 15 lower bounds



Nearly tight!

LB3D Algorithm

- **Preprocessing (= Indexing)**
  - Just sort all the substructures by the l_{16} value
    - $O(n \log n)$ time
- **Query**
  - Find candidates whose $\mathrm{LB}_{(1,6)} < c$
    - Binary search using the above index
    - #remaining candidates: $O(\frac{n}{\sqrt{m}})$
  - Compute all the 15 (=constant) lower bounds for the candidates
    - $O(m + \frac{n}{\sqrt{m}})$ time
  - If all the lower bounds are smaller than the threshold, check the RMSD value
    - #remaining candidates: $O(\frac{n}{m^{1.5}}) \rightarrow O(m + \frac{n}{\sqrt{m}})$ **time in total**

Results

[Terashi, Shibuya, Takeda-Shitaka, 2012]

- **Target database: The whole SCOP 1.75 database**
  - 110,799 entries / 20,429,263 a.a.
- **Query**
  - 100 random substructures of each specified length, taken from PDB
  - Threshold: 1 Å
- **Computation Time (sec)**
  - Average computation time of 100 random queries
    - Including the preprocessing time (negligible, in fact)
  - on Intel Xeon E5506 CPU at 2.13 GHz / 12GByte Memory

➢ **2-50 time faster than the D_3-based algorithm**

➢ **20-1,000 times faster than the naive algorithm**

Query Length	40	80	120	160	200
#Substructures	16,139,532	12,009,140	8,606,303	6,179,494	4,433040
#Hits	62.6	44.2	37.6	33.6	29.0
D_3	4.755	1.168	0.620	0.564	0.460
LB3D	0.114	0.058	0.040	0.029	0.020

Summary

- **Linear-time protein 3-D structure searching algorithm** [Shibuya 2009]
  - 5-100 times faster than the naive algorithm
    - which has been the only choice for long years
- **LB3D: Practically even faster 3-D structure searching algorithm** [Terashi, Shibuya, Takeda-Shitaka 2012]
  - 2-50 times faster than the above linear-time algorithm
  - 20-1,000 times faster than the naïve algorithm
    - Including the index construction time

Future Work

- **Improvement**
  - Better lower bounds
  - Practical algorithm with better theoretical bounds
  - Worst-case linear-time algorithm
- **Incorporating indels**
  - A theoretical linear-time algorithm exists [Shibuya, Jansson, Sadakane 2010]
- **Application to other data**
  - Motion data, audio data, music data, stock data, etc.


Thank you!