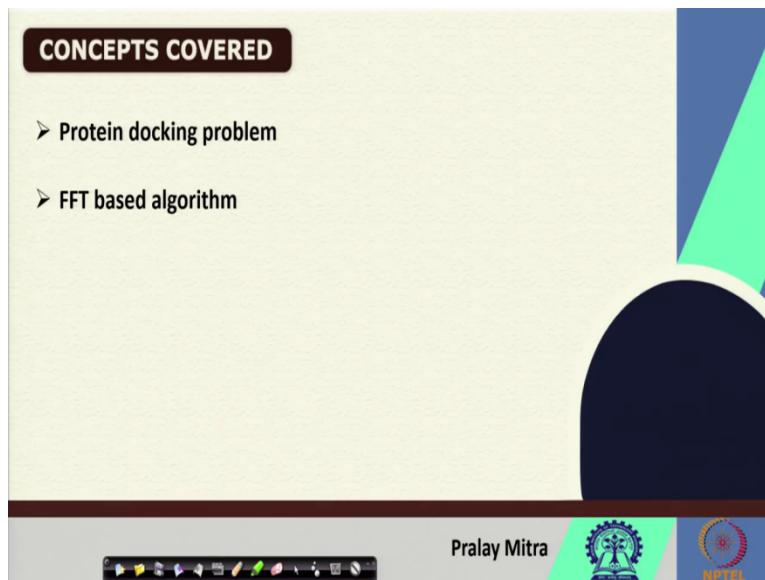


Algorithms for Protein Modelling and Engineering
Professor Pralay Mitra
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 08
Application to Protein Docking, FFT

Welcome back. In this lecture, we shall continue the application of the surface matching that we started in the last lecture for protein docking. Also, we shall highlight the advantage of utilizing the FFT or fast Fourier transform algorithm in this context.

(Refer Slide Time: 00:38)



CONCEPTS COVERED

- Protein docking problem
- FFT based algorithm

Pralay Mitra

The slide features a light beige background with a dark blue and green geometric design on the right side. At the bottom, there is a dark blue footer containing the name 'Pralay Mitra', the IIT Kharagpur logo, and the NPTEL logo. A standard presentation navigation bar is visible at the very bottom of the slide.

KEYWORDS

- Grid
- FFT
- Protein docking

So, we plan to cover the protein docking problem statement and FFT-based algorithm for this purpose. The keyword is the grid, the fast Fourier transform, and protein docking.

(Refer Slide Time: 00:56)

Application to protein docking

Problem Statement:
Given two protein molecules, computationally determine the best orientation for their interaction without the need for experiment.

Sub-problem 1: Generate a number of possible orientations from the two input protein molecules.	Sub-problem 2: Score their orientations and sort (decreasing order) the orientations based on their score function.
---	---

So, let us start with the definition of the protein docking problem. Given two protein molecules computationally you have to determine the most probable orientation or the best orientation of the given two protein molecules that will interact with each other. In other words, the protein docking problem says that given to protein molecules, you have to output a biologically relevant protein complex without the need for experimental verification or validation. But you should

remember that when somebody is developing his algorithm for protein-protein docking, there is a requirement of experimental validation or verification, which is just for benchmarking or validation purposes. Truly, you always do not need experimental validation.

If you look at the left-hand side of the figure, it is the same protein subunits that we have discussed and we took that example of PDB ID: 6BB5 from the last class. The only difference is that I have rotated and translated the red component keeping the yellow component unchanged.

How do you benchmark your algorithm? You start with a known protein complex, separate their component chains, apply some random or arbitrary transformation on both or at least on one subunit, then allow them to dock. Once it will dock then you compare the docking result with the existing protein complex. Usually, it is practice when you are designing a protein docking algorithm.

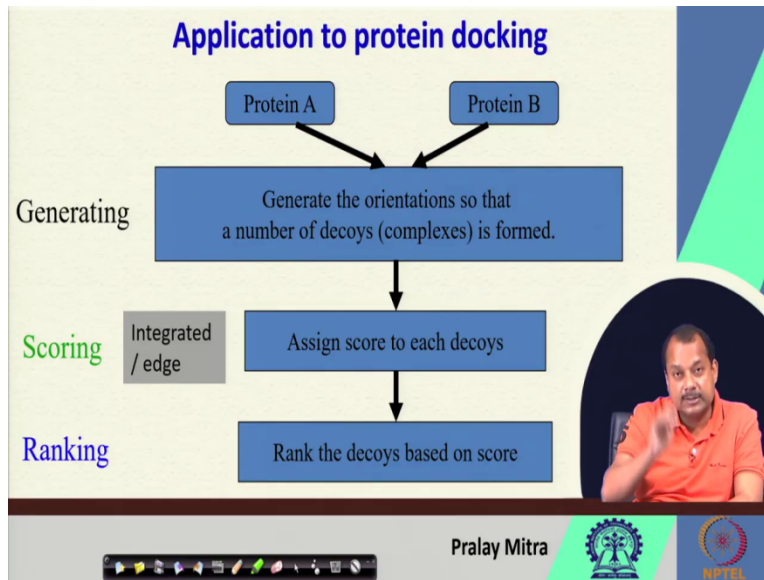
In this context, you understand that there are two subproblems - one is the generation of the protein molecules, another is the scoring of the protein molecules. When I say the generation of the protein molecule that involved one term known as a decoy.

The concept of the word decoy has taken from the war-front, where a soldier put his helmet on a stick and from the trench he used to show to the enemy. The enemy by looking at the helmet will feel that probably that is a soldier but that is not a soldier. This is kind of a decoy.

Similarly, in the context of protein docking, decoy looks like a protein complex, but it may not be the protein complex. First, what do you need to do? If I start with this yellow and red subunit that is given here. I generate all possible orientations.

I know this is complex, I separate chains. Next, I have to generate all possible orientations of one molecule to another by applying random translation and rotations. Given, three axes, I can translate along the X-axis, the Y-axis, the Z-axis, and rotation about the X-axis, the Y-axis, and the Z-axis or a combination of them like about an arbitrary axis. Based upon it I shall generate all possible orientations or decoys. Next, we shall compute the matching score on the decoys using the previous algorithm which includes penetration information. It is required since I am working with the computer program, so there is a possibility that atoms may penetrate that is not allowed.

(Refer Slide Time: 07:24)



I penalize such cases to discard those. After that, I use the score function and go by the rule that the maximum interacting surface is most likely to be correct. Nowadays, we have seen that it will be large enough for the interaction, but it may not be the maximum. Thus, instead of outputting highest or maximum score, we score and output a significant number of cases where the score is good enough. Therefore, two parts – (i) generate all the possible orientations, and (ii) for each orientation score them.

Thus, in the protein docking problem with protein A and protein B as input, we generate several orientations (generating phase) called decoys and score them to rank those decoys based upon the score. The ranking step is a simple step that sorts the decoys based on the score function. If the score function is maximized then I sort by decreasing order or non-increasing order.

Grossly the score function can be either an integrated scoring function or an edge scoring function. In the former case, scoring is done during the generation of the decoys, whereas in the latter case scoring is done after completion of the decoy generation. The advantage of the integrated scoring function is that you have the opportunity to decide whether to retain an orientation or not during the generation phase and hence you output provable decoys. Whereas in the case of edge scoring function, you output all decoys first before the scoring.

(Refer Slide Time: 10:38)

Application to protein docking

- Tagline – “Higher the decoys; better the possibility of having a hit”
- How many is good?
- Move to discrete space

Pralay Mitra

Mostly, we rely on this integrated one where we generate and score. An example is the surface matching score with the penetration option that I demonstrated to you. If we find the score is not good enough, then we discard that orientation immediately. I rely on an integrated one because during the generation phase if there is a possibility of penetration then those are not biologically valid, and will be eliminated (no need to store) because of the penalty due to penetration.

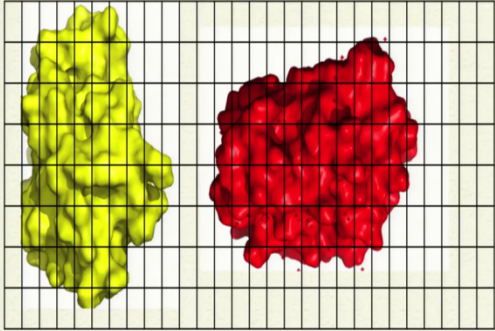
Given two protein molecules red one and a blue one in three-dimension space with their coordinate system (XYZ). They can translate along all three axes and rotate about all three axes and hence there are six degrees of freedom for molecule A for molecule B. Now, the question is how many orientations should you generate? The tagline is higher the decoys better the possibility of having a hit, so, generate as much as possible. How many are good? There is no limit, but well, you need at least a few true positive cases or correct biological orientation.

While I am interested in it, I have to decide my translation steps along the X-, Y-, and Z-axis, and when I rotate then my angular step might be 1-degree, 2-degree, 3-degree, ... Regarding the orientation, remember that 360-degree rotation indicates that I return to the same position. So, whatever the angle step you will decide it must divide 360 to an integer value. For example, if you decide on 9, then $360/9$ is divisible, if you decide 12, $360/12$ is divisible. So, these both are fine and will give you some integer value, but if you say decide on $360/7$ degree then it is not an

integer. So, there will be some mismatch. Usually, this is not accepted for rotation, but 9-degree, 12-degree, which divides 360 degrees to some is accepted.

(Refer Slide Time: 14:03)

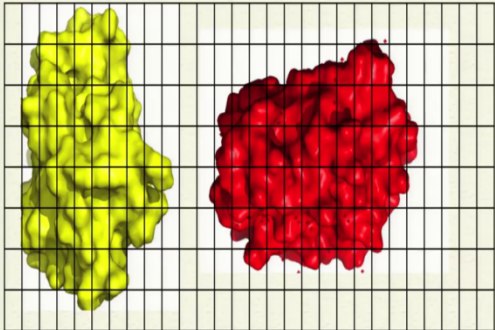
Application to protein docking



Sub-problem 1: Generate a number of possible orientations from the two input protein molecules.

Pralay Mitra

Application to protein docking



Translate and/or Rotate both the subunits?
Or only one is enough?
Which one?

Pralay Mitra

The first subproblem is to generate several possible orientations from the input protein molecules by applying 6 degrees of freedom (3 translations + 3 rotations) on both the protein molecules. Moving to discrete space the molecules are on the grid. Now, you have to translate so that there will be contact between the protein molecules, without which there is no point in doing

orientation calculation. Just think, do I need to translate and/or rotate both the subunits? You think carefully.

One situation is that translating and rotating this one and keeping another one fixed is theoretically the same as translating and rotating both. Thus, the translation rotation of both molecules is not required. Keeping one fixed if I translate and rotate the other one, then you will get the same result that is why there is no need for translation rotation on both which reduces the total computation time. If it is only one then which one? This answer is I believe is simple. Given two protein molecules, since I need to translate and rotate only one and I ensured that translation and rotation of one are good enough for translating and rotating both, I mean, both are same or equivalent. So, for translation and rotation, I choose the small molecule. I select the smaller molecule to be mobile means it will translate and rotate whereas the other one will be static. Hence, I can reduce some computation time. Is not it?

Here are the steps of your second algorithm where the input is two protein molecules and output is several orientations of the protein molecule. The word *several* depends upon two facts, translation and rotation steps. If I decide translation of 1 Å and rotation of 12° degree then I will have a combination of orientations if I decide that translation of 3 Å and rotation of 30°, then there will be less number of orientations. However, I do not know a prior whether all the placements will be retained or not, because if there is penetration that positioning may be excluded.

(Refer Slide Time: 16:40)

Application to protein docking


Algorithm 2:
 Input: Two protein molecules
 Output: Several orientations of the protein molecules

Sub-problem 1:
 Generate several possible orientations from the two input protein molecules.

Steps:

1. Translate along the X-axis
 1. Translate along the Y-axis
 1. Translate along the Z-axis
 1. Rotate about the X-axis
 1. Rotate about the Y-axis
 1. Rotate about the Z-axis
 1. Score the orientation
 2. Output the orientation

Handwritten notes: Translation & rotation steps, O(N^6), 2 sec, (10^3)^6, self score (10^3)



Pralay Mitra

Application to protein docking


Algorithm 2:
 Input: Two protein molecules
 Output: Several orientations of the protein molecules

Sub-problem 1:
 Generate several possible orientations from the two input protein molecules.

Steps:

1. Translate along the X-axis
 1. Translate along the Y-axis
 1. Translate along the Z-axis
 1. Rotate about the X-axis
 1. Rotate about the Y-axis
 1. Rotate about the Z-axis
 1. Score the orientation
 2. Output the orientation

$$MolC_{\alpha, \beta, \gamma} = \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N MolA(i, j, k) \times MolB(i + \alpha, j + \beta, k + \gamma)$$



Pralay Mitra

Anyway, you have two protein molecules. I decided to translate and rotate only the smaller one and keep another protein molecule static. Considering 6 degrees of freedom, 3 translations, and 3 rotations will be nested like this.

After the orientation, we score. We already discussed the scoring equation where Z and Z prime are incorporated for penetration purposes. So, while I output, I mention if the score is greater than some threshold or not. What is this threshold value? Assume that the number of grid cells is

more than 200 or the score value must be greater than 200 or I can decide on the size of the protein molecules.

If both the molecules are large and of comparable size, then 200 overlaps may occur. So, it is always a good idea to give some conditions for outputting the orientation so that penetration cases will be ruled out from here. The next step of analysis will be easy. Mentioning some thresholds by analyzing the size of the protein molecule is very easy to calculate during parsing or reading the protein structure file.

All the algorithms which are available as docking software use this kind of concept. The score must be greater than a threshold to output decoys. If you output all then there is no point in doing an integrated scoring function that can be an edge scoring.

However, one negative point is the computational complexity – $O(N^6)$. That is huge and you should also remember that in a standard protein molecule the number of atoms will be of the order of thousands.

Additionally, if I consider that score computation will take some time then definitely combining all these will take a huge time. When I say $O(N^6)$ and each step is a microsecond then hardly you encounter any problem since high-speed processors are there. But if your score function is not that simple if some other auxiliary physicochemical information in your atom and amino acid record is attached and you are doing that calculation, then it will be highly time-consuming. The size of the protein is 10^3 , to the power 6 for the complexity. That is huge. If I consider that each step will take one second, I cannot afford that one. Hence, we need to adapt the fast Fourier technique.

This is the score function and the first Fourier algorithm. I declare $MolC_{\alpha,\beta,\gamma}$ on the complex $MolA(i, j, k)$ multiplied with $MolB(i+\alpha, j+\beta, k+\gamma)$ and i, j, k all varies from 1 through N . I am assuming that it is a cube. The grid size is identical along all the axes (X, Y, and Z) and without any loss of generality, I am assuming that all the grid steps are the same. Moreover, I am adding α, β, γ along the X-, Y- and Z-direction, respectively so that I can have overlapping information.

(Refer Slide Time: 23:13)

Application to protein docking

$$MolC_{\alpha, \beta, \gamma} = \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N MolA(i, j, k) \times MolB(i + \alpha, j + \beta, k + \gamma)$$

Pralay Mitra

Application to protein docking

Computational Overhead: $O(N^3)$

$$MolC_{\alpha, \beta, \gamma} = \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N MolA(i, j, k) \times MolB(i + \alpha, j + \beta, k + \gamma)$$

Pralay Mitra

I calculated the overlapping information. After that, I computed MolC, and that way I have several values out of which I output the best one for that particular orientation. So, we are moving to the discrete Fourier transform. If you remember, the approach is taken for fast Fourier transform. The best is that you move on to a Fourier space, you do the calculation on there and by inverse Fourier transform come back and then output the result. The steps will surely expedite the process.


(Refer Slide Time: 25:16)

Application to protein docking


Discrete Fourier Transform (DFT) of $X_{i,m,n}$

$$X_{o,p,q} = \sum_{l=1}^N \sum_{m=1}^N \sum_{n=1}^N \exp[-2\pi i(o l + p m + q n) / N] \times X_{i,m,n}$$

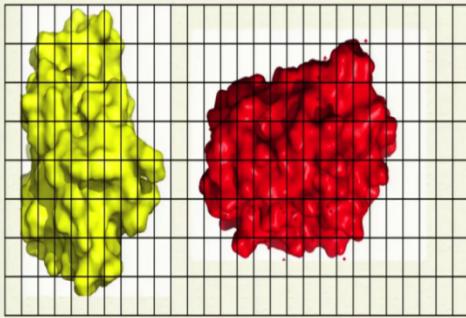
where $o,p,q = \{1, \dots, N\}$ and $i = \sqrt{-1}$




Pralay Mitra




Application to protein docking



Computational Overhead: $O(N^6)$

$$MolC_{\alpha,\beta,\gamma} = \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N MolA(i,j,k) \times MolB(i+\alpha, j+\beta, k+\gamma)$$


Pralay Mitra



Application to protein docking

$$MolC_{\alpha,\beta,\gamma} = \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N MolA(i,j,k) \times MolB(i+\alpha, j+\beta, k+\gamma)$$

Assuming

$$C = \text{DFT}(MolC)$$

$$A = \text{DFT}(MolA)$$

$$B = \text{DFT}(MolB)$$

and A^* is the complex conjugate of A

$$C_{i,j,k} = A_{i,j,k}^* \otimes B_{i,j,k}$$



Pralay Mitra



Application to protein docking

$$MolC_{\alpha,\beta,\gamma} = \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N MolA(i,j,k) \times MolB(i+\alpha, j+\beta, k+\gamma)$$

Taking the inverse Fourier Transform (IFT):

$$MolC_{\alpha,\beta,\gamma} = \frac{1}{N^3} \sum_{x=1}^N \sum_{y=1}^N \sum_{z=1}^N \exp[2\pi i(i\alpha + j\beta + k\gamma) / N] \times C_{i,j,k}$$



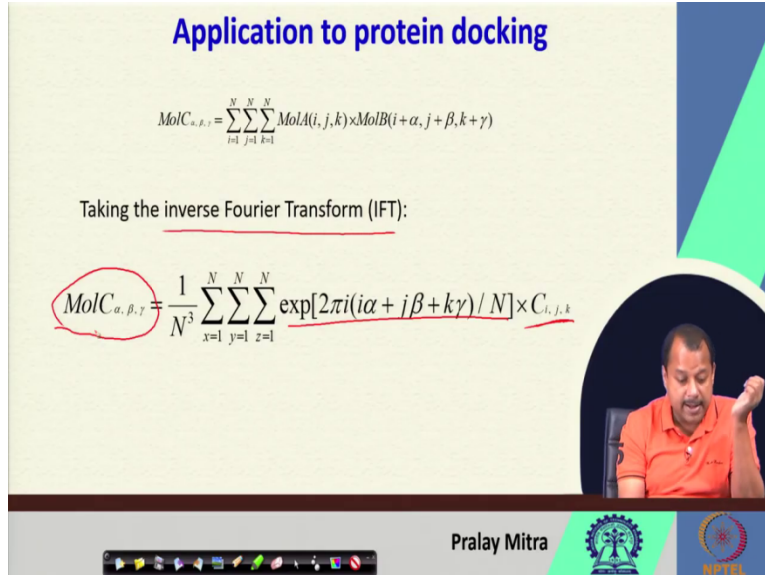
Pralay Mitra



Application to protein docking

$$MolC_{\alpha, \beta, \gamma} = \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N MolA(i, j, k) \times MolB(i + \alpha, j + \beta, k + \gamma)$$

Taking the inverse Fourier Transform (IFT):

$$MolC_{\alpha, \beta, \gamma} = \frac{1}{N^3} \sum_{x=1}^N \sum_{y=1}^N \sum_{z=1}^N \exp[2\pi i(i\alpha + j\beta + k\gamma) / N] \times C_{i, j, k}$$


Pralay Mitra

The Discrete Fourier transform DFT in sort of a function $Xl(m, n)$ can be written as:

$$X_{o, p, q} = \sum_{l=1}^N \sum_{m=1}^N \sum_{n=1}^N \exp[-2\pi i(ol + pm + qn) / N] \times x_{l, m, n}$$

Now, taking the analogy with the previous slide you see α, β, γ here and i, j, k here $i + \alpha, j + \beta, k + \gamma$ now, here o, p, q and l, m, n it varies from 1, to m ; n 1 through N . So, taking this analogy, I can convert this $MolA$ and $MolB$ to DFT the same thing is done here.

As you see $C = \text{DFT}(MolC)$, $A = \text{DFT}(MolA)$, and $B = \text{DFT}(MolB)$ and A^* star is the complex conjugate of A . We can write

$$C_{i, j, k} = A_{i, j, k}^* \times B_{i, j, k}$$

This is the equation that I computed for the surface matching with the penetration option. Now, in the Fourier space, I am getting only the simple multiplication of A conjugate and B where $A = \text{DFT}(MolA)$ and $B = \text{DFT}(MolB)$, and that way I am getting the $\text{DFT}(MolC)$.

Since I am getting the $\text{DFT}(MolC)$, so, to get $MolC$ I have to compute the Inverse Fourier transform or IFT of this C . Now, it is clear to you what I did instead of calculating directly the $MolA$ multiplied with $MolB$ with the penetration option. I converted $MolA$ to Fourier space

using the DFT(MolB). I applied DFT on MolB also. Then, I multiplied the conjugate of A and B to get C and apply inverse Fourier transform on C to get MolC.

(Refer Slide Time: 29:35)

Application to protein docking

Algorithm 3:
Input: Two protein molecules
Output: Several orientations of the protein molecules sorted by surface match

Steps:

1. Digitize Mol_A .
2. $A^*=[DFT(Mol_A)]^*$
3. Digitize Mol_B
4. $B=[DFT(Mol_B)]$
5. $C=A \times B$
6. $c=IFT(C)$
7. Get the score value
8. Change the orientation (rotation/translation)
9. Go to step 3 until required number of orientations changes are not reached.
10. Sort (non-increasing order) by the score value

Taking the help of fast Fourier transform algorithm computational complexity reduces.

Pralay Mitra

Hence, my algorithm will be a simple one. The same input-output but my algorithm digitize MolA, A^* is the complex conjugate of DFT(MolA), digitize MolB then $B=DFT(MolB)$, then multiply A^* with B (small c will be Inverse Fourier Transform of C) and get the score value, change the orientation (rotation and translation), go to step three until the required number of orientation changes are not reaching, sort the decoys in non-increasing order by the score value.

If you take the help of a fast Fourier transform algorithm then the computational complexity will reduce. Usually, if you apply FFT on $O(N^2)$ algorithm then you will get $O(N \log N)$. So, you can calculate what will be the complexity if you apply FFT on $O(N^6)$ algorithm.

(Refer Slide Time: 31:12)

REFERENCES

Katchalski-Katzir, Ephraim, et al. "Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques." *Proceedings of the National Academy of Sciences* 89.6 (1992): 2195-2199.

The slide features a dark blue header with the word 'REFERENCES' in white. The main content area is light green. A video inset in the bottom right shows a man in an orange shirt speaking. At the bottom, there is a navigation bar with icons and logos for IIT Bombay and NPTEL.

This particular algorithm is available in this publication. It is done by Katchalski and Katzir and published in PNAS in 1992, long back. But this is one of the basic works on the protein-protein docking and most of the existing techniques like FTdock, Zdock to some extent uses this particular kind of concept to reduce their computation. Thank you.