Honors Thesis Proposal

for

Coarse Grained Monte Carlo Simulation of the Self-Assembly of the HIV-1 Capsid Protein

Jeffrey Weber

Bo Chen, Ph.D.
Thesis Committee Chair
Department of Physics

Talat Rahman, Ph.D.
Department Chair
Department of Physics

Aniket Bhattacharya, Ph.D.
Committee Member from Major
Department of Physics

Kelly Astro, M.S.Ed.
Director of Research and Civic Engagement
The Burnett Honors College

Ali Orooji, Ph.D.
Committee Member outside Major
Department of Electrical Engineering and Computer Science
INTRODUCTION

The human immunodeficiency virus (HIV) epidemic is estimated to affect about 34.2 million people worldwide[1]. In the United States alone, it was estimated that there were 1,148,200 people living with HIV at the end of 2009, with approximately 50,000 people being infected every year[1, 2]. Worldwide, there were 2.5 million new cases of HIV reported in 2011[1]. Since the HIV epidemic began, about 30 million people around the world have died, with an additional 1.8 million deaths every year[1].

HIV is a virus and by nature, during its lifetime will infect a host cell, copy its genetic material, and leave the cell in order to find a new host[3]. HIV belongs to a class of viruses known as retroviruses, meaning that it carries its genetic material in the form of RNA and uses the process of reverse transcription to replicate[4]. HIV can be further classified as a lentivirus due to the duration of its incubation period. Over long periods, HIV can compromise the human immune system, which allows opportunistic infections to flourish.

Despite the efforts of researchers, there is still no cure for HIV. That being said, HIV is much more treatable now than ever. One possible target for an effective HIV drug could be the viral capsid. The capsid is the part of the virus, typically made up of repeated subunits, that encloses the genetic material[3]. Some studies show that targeting the capsid can affect the infectivity of the virus[5, 6]. The infectivity of HIV-I is dependent on the proper disassembly of the capsid, or “uncoating,” in the host cell[7]. If the structure of the capsid becomes too stable, the genetic material may not be able to escape, resulting in a loss of infectivity[8, 9]. Conversely, if the capsid becomes too destabilized, the virus may not be able to form correctly[7, 8]. Shi, et al. reasoned in the study of how PF74 affects the stability of the capsid and the infectivity of the virus, that the loss of infectivity in destabilized capsids may be due to a flaw in reverse transcription in the host cells that stems from the premature uncoating[7].
Purpose

By studying the HIV-1 capsid, it may be possible to discover new treatments or even cures for those infected with HIV. Although the structure of the subunits involved in the capsid assembly are known, relatively little is known about the assembly process of the HIV-1 capsid. Thus, the goal of this project is to learn as much about the self-assembly of the HIV-1 capsid as possible through Monte Carlo simulations. It has been shown that stabilizing and destabilizing mutations can compromise infectivity, and knowing how the capsid assembles could allow targets for potential antivirals to be determined[10].

Further, a better understanding of the HIV-1 capsid could serve purposes beyond the scope of treating those infected with the virus. It has been shown that viral capsids can be repurposed, including the possibility of targeting other infections[3]. This could make the viral capsid an effective delivery system for drugs targeting that infection. In addition to this, Soto and Ratna note that viral capsids can be useful in bionanotechnology due to the scale and adaptability of viral capsids[11].

Definition of Terms

Terms will be used related to biological elements as well as coding throughout this thesis and will be defined here. The viral capsid is a shell that encloses the genetic material within a virus. The capsid protein may be abbreviated CA. This viral capsid is made up of repeating oligomers or subunits that collect to form the shell. Oligomers are defined as collections of subunits. In the case of HIV-1 the most important oligomers are the dimer, trimer, pentamer and hexamer. These contain two, three, five and six subunit monomers, respectively. The subunit monomer is further divided into two domains, the amino-terminal domain, defined by its termination by an amino acid containing a free amine group, and the carboxyl-terminal domain, which is terminated by a carboxyl group. For simplicity, the amino-terminal domain can be simplified to NTD and the
carboxyl-terminal domain can be represented by CTD. These domains can be further broken down into \( \alpha \) helices and \( \beta \) sheets. The \( \alpha \) helix is a spiral structure of proteins and the \( \beta \) sheets are less common and consist of \( \beta \) strands. \( \beta \) strands are just chains of amino acids that are linearly linked.

Some coding terms that may be used include monte carlo, which is an algorithm that requires random sampling to accomplish some task. Monte carlo may be abbreviated MC, and in this simulation the Metropolis criterion is used. OpenMP is a software package that allows the parallelization of Fortran code, and may be represented by OMP. These and other significant terms may be used and defined throughout this document.
LITERATURE REVIEW

Many scientists have performed a significant amount of research on HIV, the capsid protein and the assembly process. A collection of these studies was analyzed in order to better understand the maturation of HIV and the assembly and structure of the capsid.

In HIV, Gag molecules collect at the plasma membrane of the host cell and creates a spherical structure[12, 13]. This immature shell matures while budding from the plasma membrane of the host cell, where the protease cleaves Gag into component proteins[13, 14]. These proteins include the matrix protein (MA), the capsid protein (CA), and the nuclear capsid protein (NC). About 1,500 of these capsid proteins would assemble to form the capsid shell around the nuclear capsid proteins[15]. Although polymorphisms exist, such as tubular assemblies, HIV-1 cores typically form into the shape of a fullerene cone in the wild type virus[13, 15, 16, 17]. This capsid shell plays a vital role in determining the infectivity of the virus[18].

Structure of HIV-1 Capsid

Viral capsids are typically made up of repeating subunits that together form a fairly symmetric assembly[3]. In the case of the HIV-1 virus, these subunits are the hexamer and pentamer. The hexamer is made up of six dimer subunits, while the pentamer is made up of five. While the pentamer subunit is needed to close the assembled capsid, pentamers have only been observed using mutants in vitro[17, 19]. These oligomers are arranged with the NTD on the inside, with the CTD facing outwards. Adjacent hexamers and pentamers are joined through the CTD dimerization interface as well as a CTD three-fold trimerization interface, which can be seen in Figure 2[16]. Sometimes, dimers must adjust to cover variable distances depending on whether they are connecting hexamers to hexamers and hexamers to pentamers[15].
As stated earlier, the most common HIV-1 capsid shape is considered the fullerene cone when assembled in vivo[17]. This cone is comprised of a lattice of about 250 hexamers and precisely 12 pentamers[7, 15]. Pentamers allow for the sharp angles needed for the lattice to form into a cone[15, 16]. The pentamers are distributed near the ends of the cone, with 5 pentamers at the sharper end and 7 at the wider end[15]. The curvature of the cone can be attributed to additional factors besides these 12 pentamers. One factor contributing to the curvature is the different orientations in the CTD allowed by the flexibility of helix 9[15, 16, 18]. Variations in the NTD-CTD interfaces are also known to contribute to the curvature[18, 20]. The angle of the cone is quantized, and the wild type virus has been known to adopt one of five cone angles[17]. The most common angle is also the narrowest angle, which is much more prominent than other cone angles[12, 17].

Capsid assembly in vitro shows preferred shapes, but with severe polymorphism[12]. HIV-1 has been shown to self-assemble into tubes and cones in vitro[16]. This implies that the required information for capsid formation is contained within the capsid protein[17].

Structure of Capsid Protein

The most basic subunit of the HIV-1 capsid is the CA monomer. Each HIV-1 CA monomer contains an amino-terminal domain (NTD) and a carboxyl-terminal domain (CTD) [21, 22, 23]. The N-terminal domain consists of seven α helices and a β hairpin[21, 22]. A flexible linker region of residues and a $\beta_{10}$ helix connects the N-terminal domain to the C-terminal domain, which consists of four additional $\alpha$ helices[23]. This linker region is important for the correct assembly of the HIV-1 capsid[8]. The monomer subunit can be seen in Figure 1.

In solution, two CA monomers spontaneously form into a dimer at the CTD dimerization interface around helix 9 with a dissociation constant of 18 $\mu$M[23, 24, 25]. Dimerization of subunits supports capsid assembly in vitro[14]. The angle of the dimerization helices is confined and the dimerization interface is much stronger than other interfaces[16].
Assembly *in-vitro*

Assembly of the capsid shell can be contributed to hydrophobic, electrostatic, van der Waals, and hydrogen bonding interactions[26]. All the required information for the subunits to form into the conical capsid is contained within the CA polypeptide[17]. In the case of the HIV-1 capsid protein, interaction sites include a CTD-CTD dimerization interface at helix 9, a three-fold CTD-CTD trimerization interface between at helix, an NTD-CTD intersubunit interface between helices 4 and 8, and an NTD-NTD interface between helices 2 and 3[10, 16, 23, 27]. These interaction sites can be seen in Figure 2.

Assembled structures consist of tubes and cones that resemble actual capsids, but with great uncertainty in diameters[16]. Dimerization of monomers favors the assembly of the capsid *in vitro*, even though dimers may play a different role while assembling *in vivo*[14]. In order for *in vitro* assembly to occur, the capsid protein must be in a solution with a high concentration of salt[16].

![Figure 1](3H47.pdb): The structure of HIV-1 CA as derived from 3H47.pdb.
Simulations

Due to the speed of assembly, it can be difficult to find assembly intermediates and characterize the assembly of viral capsids in detail through experimentation alone[26, 27]. Through the use of simulations based on experimental data, further investigation into assembly pathways can be performed[26, 27]. While a full atomic representation of the entire capsid assembly is not currently practical, coarse grained models can be used to determine assembly pathways[26, 27].

One benefit of coarse grain models is that you can adjust each factor separately in order to see how it affects the assembly of the capsid[26]. This can give insights into how different

![Diagram of viral capsids with lines between cylinders representing interacting interfaces.](image)

Figure 2: Lines between cylinders represent interacting interfaces. Used with permission from Chen and Tycko[27].
interfaces can affect the assembly process. Keeping interaction strengths weak can aid in the assembly process[26]. If the interaction strength is too high, a "kinetic trap" is created, which may stop subunits from being able to form the full capsid[14]. While interaction strengths should be kept weak, they cannot be too weak or no assembly will be observed[14]. Optimal assembly should occur when the subunit associations can be reversed[26].
RESEARCH QUESTION

Methods

The self assembly of the HIV-1 capsid protein will be analyzed through a Monte Carlo simulation written in Fortran 90. Code can be produced to use either the monomer or the dimer as the most basic subunit of assembly. Monte Carlo simulations consist of using random numbers to solve a problem, in this case the Metropolis criterion will be used to determine the pathways of assembly. The simulation maintains a periodic boundary condition while allowing molecules to explore 3D configurations.

The simulation is broken down into several main steps. The system initially sets up an evenly spaced lattice of subunits and allows them to anneal at a high temperature for 10,000 steps per subunit. The annealing process is designed to randomize the system in order to replicate actual systems. Alternatively, a previously defined lattice can be read into the program, and the simulation can be started in this predefined configuration. When one of these processes has been completed, subunit assembly can begin. Assembly consists of looping through each molecule in the system and attempting to randomly translate and rotate the molecule. Maximum steps of 0.05 nm in each Cartesian direction and maximum rotations of π/360 radians can be defined. If the move satisfies all criteria specified in the next paragraph, then the move is accepted and the position and orientation of the molecule is updated. If the move is rejected, the molecule's position and orientation will not be updated. The simulation will then select the next molecule and continue until the simulation is stopped.

Each step can be broken down into several parts. After the molecule is selected, random numbers are used to determine how the molecule should be translated and rotated. It should be noted that the translation must be calculated in all Cartesian directions. In its new position, the subunit must meet several criteria in order to make sure that the move is accepted. First, it must not
collide with any other molecules, so an algorithm will be needed to check for collisions. Additionally, the energy of the system must decrease or pass a random check that is further explained in the Energy Calculation section. If either of these criteria are not met, then the move will be rejected and the next molecule will be selected. If both criteria are satisfied, the move will be accepted, the appropriate values will be updated, and the next molecule will be selected.

Every time the simulation reaches a certain number of steps, the output will be written to file. To save space, only three points from each subunit need to be recorded. From this output, the system can be reconstructed using visualization software. Conclusions about how the system formed can then be derived from images and movies created using this visualization software.

*Translation and Rotation*

Once a molecule has been selected, it must be moved to a new position in order for the system to evolve. Translation could occur in all Cartesian directions and can be determined by producing a random array of three real numbers within an allowed range. This process is described by the equation

$$\text{trsl} = \text{rand} \times 2.0 \times \text{trsl}_{\text{max}} - \text{trsl}_{\text{max}}$$  \hspace{1cm} (1)

where \(\text{rand}\) is a randomly generated number between zero and one. Similarly, an axis of rotation for the molecule can be chosen and a rotation angle be found from the equation

$$\text{angle} = \text{rand} \times 2.0 \times \text{angle}_{\text{max}} - \text{angle}_{\text{max}}$$  \hspace{1cm} (2)
where rand is a random number between zero and one. This angle than be used to derive a rotation matrix

\[
\begin{bmatrix}
    u_x^2 + (1 - u_z^2)cs & u_x \times u_y (1 - cs) + u_z \times sn & u_x \times u_z (1 - cs) - u_y \times sn \\
    u_x \times u_y (1 - cs) - u_z \times sn & u_y^2 + (1 - u_x^2)cs & u_y \times u_z (1 - cs) + u_x \times sn \\
    u_x \times u_z (1 - cs) + u_y \times sn & u_y \times u_x (1 - cs) - u_z \times sn & u_z^2 + (1 - u_y^2)cs
\end{bmatrix}
\]

(3)

where \( u \) is the position vector, \( sn \) is \( \sin(\theta) \), and \( cs \) is \( \cos(\theta) \). This matrix can be applied to rotate the molecule. Assuming that the energy and collision criteria are met, these new coordinates will represent the new location of the molecule.

**Collision Detection**

In order for the system to be considered valid, molecules must not collide with one another. As such, a collision detection algorithm must be implemented each time a molecule is moved or rotated. The collision detection algorithm consists of three main checks, with each subsequent check on a smaller scale than the last. Collision detection must be run over every molecule in the system, or every molecule in the verlet list if implemented. Because of this, the collision detection algorithm is the most costly part of the code.

The first check assumes each molecule is a sphere and calculates the distance between the center of each molecule. The radius of the sphere is assumed to be equal to the maximum reach of the molecule. The distance between the center of the two molecules is then calculated. If it is found that these spheres do not collide, then there is no possibility of collision. If the spheres do collide, then another check must be performed.

The second check must loop over every helix in each of the molecules. The idea again is to find the distances between the center of the two helices and see if the it is greater than the sum of their radii. If the distance is greater that the reach of the helices, then they do not collide and
the next set of helices can be considered. If the reach is greater than the distance, then they may or may not collide and an additional check is needed.

The last level of the collision detection algorithm actually treats two helices as cylinders. If the closest point on their axes is less than the sum of their radii, then the cylinders are said to collide. This is accomplished through an algorithm designed by Dr. Chen[28]. If there is no collision between the cylinders, then the collision detection algorithm is allowed to continue. If a collision between two cylinder is detected, then the algorithm can return immediately that a collision has been found.

Energy Calculation

In order to determine the likelihood of a certain configuration, the energy of each configuration must be calculated. Statistical physics tells us that the system should slowly move towards the state with the least amount of energy associated with it. Because only one molecule is moved at a time, the change in local energy of the displaced molecule is all that needs to be calculated. The local energy between the selected molecule and all interacting molecules must be determined.

The energy calculation algorithm is divided into two parts. Similar to the collision detection algorithm, the algorithm will treat each molecule as spheres and check to see if they are within a particular cutoff limit. The cutoff limit is defined as the distance at which intermolecular interaction strengths are too insignificant to contribute to the total energy, taken to be 10.91037 nm. If the molecules are too far apart, the algorithm can move on to the next molecule. Otherwise, the energy between interaction sites on the molecules must be determined.

The next part determines the interaction energy on a site-by-site basis. The interaction potential was determined using the Lennard-Jones potential as well as and angular restraint. The energy was obtained using
\[ V(\theta, r) = f(\theta) \cdot g(r) \]  

(4)

\[
f(\theta) = \begin{cases} 
\frac{|\theta - \theta_0|}{\theta_{max}} - 1 & |\theta - \theta_0| \leq \theta_{max} \\
0 & |\theta - \theta_0| > \theta_{max}
\end{cases}
\]  

(5)

\[
g(r) = 4\varepsilon_0 \left[ \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^{6} \right].
\]  

(6)

Equation (5) places an angular dependence on the interactions, where \( \theta \) is the angle between interacting cylinders in the simulation, \( \theta_0 \) is the angle derived from experimental data, and \( \theta_{max} \) is the cutoff angle at which the interaction strength becomes zero. Equation (6) is the Lennard-Jones potential, where \( r \) is the distance between interaction cites in the simulation and \( \sigma \) is the optimal distance between interaction sites. This method of calculating energy has been applied to many coarse grained models, such as Chen and Tycko[27].

Once the local energy has been determined, it can be used to find the change in total energy. Metropolis criterion was then used to determine whether the move should be accepted or rejected, assuming it passes the collision criterion.

**Monte-Carlo Considerations**

Monte-Carlo methods use systematic random sampling to evolve the system into a preferred state. An effective Monte-Carlo simulation requires a reliable random number generator. In this simulation, the Fortran intrinsic function for calling random numbers can be used that is seeded using the current date and time. In this simulation, random numbers will be used at three main points in the code. The first is to determine the direction and magnitude of the translation, the second determines the magnitude of rotation, and the third is to check that the energy satisfies the Metropolis criterion.
Visualization

Although Fortran is efficient at simulating the assembly of the HIV-1 capsid, it does not provide a visual output from which conclusions can be drawn. Visualization must be implemented using software such as MATLAB or Visual Molecular Dynamics (VMD). This means that the Fortran simulation must output the state of the system to text files at reasonable intervals. Although the entire system could be written to file, it is much more space efficient if only three points from each molecule are recorded. Since this is a coarse grained model, each molecule is treated as a rigid body and therefore a template can be mapped exactly to the correct position using only these three points.

VMD provides a console from which TCL scripts can be run, making it a very versatile tool in visualizing molecular systems. TCL scripts can be devised to draw systems, highlight features, and make movies. TCL scripts can be used to transform a particular three point representation into a full cylindrical representation, and then implement VMD’s draw commands to draw cylinders. For ease of viewing, the N-terminus and C-terminus can be colored differently in the VMD window.

The idea of drawing the system can be expanded into creating a movie. Because output was recorded at certain intervals, many systems can be drawn and snapshots can be produced at each step. This is best done using a TCL script that implements the drawing script discussed earlier. These snapshots can then be combined into a movie. Unfortunately, this is a static view of the system that can not be probed further once it has been created. A more dynamic way to create a movie is to use VMD’s ability to map trajectories. This means that the three point representation must be converted to cylindrical as before, but instead of drawing cylinders, VMD will draw “bonded atoms.” These bonded atoms can be displayed as uncapped cylinders or using VMD’s “licorice” representation. This allows full functionality of VMD at all steps, including the ability to view the system from different perspectives and focus on remarkable aspects of the
assembly. This trajectory can even be made into a static mpeg movie as before using VMD's Movie Maker extension.

Optimizations

Optimizations on the code are necessary to ensure that results are obtained in the least amount of time possible. These optimizations come from the programmer as well as the compiler. It is necessary to ensure that these optimizations do not compromise the codes integrity, and that the simulation still runs as intended.

Verlet List

An optimization that can be implemented by the programmer is the inclusion of a verlet list. The verlet list is an array that stores a list of all molecules that are within a cutoff distance that can be determined from the interaction cutoff distance. If a molecule is within this range, it will be added to the verlet list. The verlet list must be updated whenever a molecule has moved far enough that it can potentially interact with molecules that were not previously in the verlet list. To maximize the efficiency of the verlet list, the verlet cutoff distance must be small enough to significantly limit the number of molecules, but large enough that the list must not constantly be reset.

By using a verlet list, the number of molecules that must be checked in the collision detection and energy calculation algorithms can be severely reduced. These methods must be modified to loop over molecules in the verlet list instead of all molecules in the system. Although the verlet list takes some time to set up, the speed boost in these two algorithms reduces the overall runtime. This boost can be further improved by sorting the verlet list by distance from the source molecule, allowing the collision detection algorithm a better shot at detecting the collision and exiting early.
OpenMP

Multi-threading is a way to squeeze more performance out of algorithms by parallelizing them. An algorithm is a good candidate for parallelization if it is running a large number of repeated tasks. For this reason, the collision detection, energy calculation, and verlet list algorithms are good candidates for parallelization. The loop over one molecule is not a good candidate for parallelization because if two molecules are moved simultaneously, the collision detection and energy calculation algorithms may return incorrect values, which can lead to collisions of molecules or moves that are accepted or rejected based on incorrect energy values.

OpenMP provides an easy way for programmers to parallelize their Fortran codes. OpenMP uses a system of directives that can be used to create parallel regions in the code. The speed increase that can be obtained for a given region is not directly proportional to the number of threads it implements, as the setup cost must be taken into account. This means that for algorithms that do not require a large number of steps, OpenMP may actually take more time to set up the parallel regions than can be saved by the parallelization. Because of this, algorithms must be analyzed to see if there is actually any benefit to parallelization. The increase in speed is also dependent on which variables are made private and which are public to all threads, so these must be checked to see how the best possible runtime can be obtained.

The collision detection algorithm can benefit from parallelization using an OMP PARALLEL DO loop. The algorithm must loop over a large number of molecules, and must do a significant amount of work for each. In order to make sure each thread knows if there is a collision, an OMP ATOMIC call can be used to synchronize these values without delaying the algorithm. The collision detection can benefit from returning early if a collision is found, but OpenMP requires that all loops finish before the parallel region can be ended. To accommodate for this, a flag can be used to skip the work in a loop if the collision criterion has been triggered.

The energy calculation algorithm does not contain the same runtime complexity as the
collision algorithm, but can still benefit from parallelization using OMP PARALLEL SECTIONS. This will allow the energy of the system before the move and after the move to be calculated in parallel. Because these two separate energy values are calculated in sections, they do not need to call OMP ATOMIC like the collision detection algorithm needed to.

The verlet list algorithm is parallelized in a manner similar to the collision detection algorithm using OMP PARALLEL DO. Unlike the collision detection algorithm, the verlet list algorithm should not try to access the same variable, and will not end early, so it is a straightforward optimization.

Compilers

The compiler used can also have some effect on the speed of the simulation. Two compilers that are readily available are the Intel® Fortran compiler(ifort) and the GNU Fortran compiler(gfortran). Although the compiled codes should perform identical tasks, the compilers may optimize the code in different ways. This means that one of these compilers may produce a faster simulation than the other. Each of these compilers come with their own set of compilation flags that optimize the code in different ways. By choosing the correct compiler and optimizations, the code can be tuned to run more efficiently while maintaining accuracy.

Algorithm Organization

While it may seem trivial, the organization of the collision detection and energy calculation algorithms is vital in determining the speed of the simulation. If the faster of these two algorithms has to wait on the slower one, then the code is not running as efficiently as it could. The collision detection algorithm requires significantly more computational time than the energy calculation algorithm, so the energy criterion should be checked first. As long as all criteria are satisfied, it does not matter in which order they are checked. If the energy criterion is not satisfied, the code can skip the costly collision detection algorithm, providing the simulation with a boost in speed.
Model

The HIV-1 capsid protein consists of 11 $\alpha$ helices and a $3_{10}$ helix. In this coarse grained model, each of these helices will be represented by a cylinder that represents the helix. In the model, the $3_{10}$ helix is labeled as helix 8. Due to the shape of helix 7, it is divided into two cylinders. This gives the model a total of 13 cylinders per monomer subunit. This model can be seen in Figure 3.

Interaction sites were defined along several cylinders in the coarse grained model. An NTD-NTD interface is defined between cylinders 2 and 3. Values used to determine the potential were $\sigma = 0.9702867$ nm and $\theta_0 = 0.690483$. An NTD-CTD interface is defined between cylinders 4 and 10, corresponding to helices 4 and 8, with $\sigma = 0.7257217$ nm and $\theta_0 = 1.218533$. A CTD-CTD trimerization interface is defined between along cylinder 12, corresponding to helix 10, with $\sigma = 0.982183$ nm and $\theta_0 = 0.5061$. For cases where the monomer was used as the most basic subunit, the dimerization interface was defined along cylinder 11, corresponding to helix 9, with $\sigma = 0.9841567$ nm and $\theta_0 = 0.737260761$. In most cases, the $\epsilon_0$ can be taken to equal 4 kcal/mol.

![Figure 3: The actual capsid protein(left) compared to the model(right). This image was used with permission from Chen and Tycko[27].](image-url)
or 6 kcal/mol in the case of the dimerization interface, but this value can be modulated to see how it affects the system.

**Hypotheses**

It is hypothesized that the simulation should allow subunits to assemble into a lattice resembling the authentic HIV-1 capsid. From this, the curvature and arrangement of oligomers can be determined. It is also predicted that as the interface strengths are varied, the curvature of the lattice will change. This can help to identify the significance of interfaces in the capsid assembly. It can also be said that the significance of the dimerization to the assembly process can be determined by using the monomer as the most basic subunit. Dimerization is believed to be important in the correct and efficient assembly of the HIV-1 capsid.
LIST OF REFERENCES


23