TESIS - KM 185401

MOLECULAR DOCKING SENYAWA ALKALOID SA2014 TERHADAP PROTEIN CYCLIN D1 PADA KANKER MENGGUNAKAN FIREFLY ALGORITHM

ZULFA AFIQ FIKRIYA
NRP 06111750010004

DOSEN PEMBIMBING:
Prof. Dr. Mohammad Isa Irawan, M.T.
Dr. Awik Puji Dyah Nurhayati, M.Si

PROGRAM MAGISTER
DEPARTEMEN MATEMATIKA
FAKULTAS MATEMATIKA, KOMPUTASI, DAN SAINS DATA
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2019

THESIS - KM 185401

MOLECULAR DOCKING OF ALKALOID COMPOUND SA2014 TOWARDS CYCLIN D1 PROTEIN IN CANCER USING FIREFLY ALGORITHM

ZULFA AFIQ FIKRIYA
NRP 06111750010004

SUPERVISORS:
Prof. Dr. Mohammad Isa Irawan, M.T.
Dr. Awik Puji Dyah Nurhayati, M.Si

MASTER PROGRAM
DEPARTMENT OF MATHEMATICS
FACULTY OF MATHEMATICS, COMPUTING, AND DATA SCIENCE
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA
2019

LEMBAR PENGESAHAN TESIS

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Matematika (M.Mat.)
di
Institut Teknologi Sepuluh Nopember
Oleh:

ZULFA AFIQ FIKRIYA
NRP: 06111750010004

Tanggal Ujian: 16 Juli 2019
Periode Wisuda: September 2019

Disetujui oleh:
Pembimbing:

1. Prof. Dr. Mohammad Isa Irawan, M.T.
   NIP: 196312251989031001 .....

2. Dr. Awik Puji Dyah Nurhayati, M.Si
   NIP: 197006211998022001 .....

Penguji:

1. Dr. Dwi Ratna S., S.Si, M.T.
   NIP: 196904051994032003 .....

2. Dr. Chairul Imron, MI.Komp.
   NIP: 196111151987031003 .....

Kepala Departemen Matematika
Fakultas Matematika, Komputasi, dan Sains Data

Dr. Imam Mukhlash, S.Si, MT
NIP: 197008311994031003

MOLECULAR DOCKING SENYAWA ALKALOID SA2014 TERHADAP PROTEIN CYCLIN D1 PADA KANKER MENGGUNAKAN FIREFLY ALGORITHM

Nama Mahasiswa	:	ZULFA AFIQ FIKRIYA
NRP	:	06111750010004
Pembimbing	:	1. Prof. Dr. Mohammad Isa Irawan, M.T.
		2. Dr. Awik Puji Dyah Nurhayati, M.Si

ABSTRAK

Molecular docking atau penambatan ligan pada protein merupakan bidang komputasi yang sedang berkembang. Molecular docking dapat digunakan untuk mencari pola interaksi yang paling tepat antara protein reseptor dan ligan serta menjadi dasar penemuan dan desain obat berdasarkan struktur. Perkembangan dari metode dan algoritma docking yang efisien akan sangat berguna dalam penemuan obat secara simulasi. Firefly algorithm merupakan salah satu metode yang bisa digunakan untuk simulasi molecular docking. Firefly algorithm digunakan untuk mencari konformasi protein dan ligan yang optimal sehingga energi ikatan dari sistem secara keseluruhan diminimalkan. Dalam penelitian ini, digunakan dua kompleks protein-ligan dari Protein Data Bank (PDB) yaitu 2cpp dan 3ptb untuk menguji kemampuan algoritma. Didapatkan hasil bahwa firefly algorithm dapat digunakan untuk menyelesaikan molecular docking dengan nilai parameter pergerakan acak (α) sebesar 1.0. Kemudian algoritma ini digunakan untuk menyelesaikan molecular docking senyawa alkaloid SA2014 dari spons laut Cinachyrella anomala terhadap protein cyclin D1 pada kanker. Didapatkan hasil bahwa afinitas ligan SA2014 terhadap protein cyclin D1 lebih tinggi dibandingkan doxorubicin (sejenis obat kemoterapi) sehingga senyawa SA2014 memiliki potensi yang besar sebagai antikanker.

Kata-kunci: *firefly algorithm*, *molecular docking*, penemuan obat, cyclin D1, alkaloid SA2014

MOLECULAR DOCKING OF ALKALOID COMPOUND SA2014 TOWARDS CYCLIN D1 PROTEIN IN CANCER USING FIREFLY ALGORITHM

Name	:	ZULFA AFIQ FIKRIYA
NRP	:	06111750010004
Supervisors	:	1. Prof. Dr. Mohammad Isa Irawan, M.T.
		2. Dr. Awik Puji Dyah Nurhayati, M.Si

ABSTRACT

Molecular docking or ligand binding in proteins is a developing field of computing. Molecular docking can be used to find the most appropriate interaction pattern between protein receptors and ligands and become the basis for the drug discovery and design based structures. The development of efficient docking methods and algorithms will be very useful in drug discovery simulation. Firefly algorithm is one of the method that can be used for molecular docking simulations. Firefly Algorithm is used to find the optimal conformation of proteins and ligands so that the binding energy of the whole system is minimized. In this research, two protein-ligand complexes from the Protein Data Bank (PDB), namely 2cpp and 3ptb were used to test the performance of the algorithm. The results show that the firefly algorithm can be used to solve molecular docking with random movement parameter (α) 1.0. Then this algorithm is used to solve molecular docking of alkaloid compounds SA2014 from Cinachyrella anomala sea sponges towards cyclin D1 protein in cancer. The results show that the SA2014 ligand affinity for cyclin D1 protein was higher than doxorubicin (a type of chemotherapy drug) so that the SA2014 compound had a great potential as an anticancer

Key-words: firefly algorithm, molecular docking, drug discovery, Cyclin D1, alkaloid SA2014

KATA PENGANTAR

Assalamu'alaikum wr. wb.

Segala puji dan syukur bagi Allah SWT yang memiliki apa yang ada di langit dan di bumi atas segala rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan Tesis yang berjudul

"MOLECULAR DOCKING SENYAWA ALKALOID SA2014 TERHADAP PROTEIN CYCLIN D1 PADA KANKER MENGGUNAKAN FIREFLY ALGORITHM"

sebagai salah satu syarat kelulusan Program Magister Departemen Matematika Institut Teknologi Sepuluh Nopember (ITS) Surabaya.

Dalam menyelesaikan Tesis ini, penulis telah banyak mendapat bantuan serta masukan dari berbagai pihak. Oleh karena itu, dalam kesempatan ini penulis menyampaikan terima kasih kepada:

1. Keluarga tercinta yang senantiasa memberikan dukungan dan doa dengan ikhlas.
2. Bapak Prof. Dr. Mohammad Isa Irawan, MT dan Ibu Dr. Awik Puji Dyah Nurhayati, M.Si selaku dosen pembimbing Tesis yang telah banyak memberikan bimbingan, arahan serta motivasi sehingga Tesis ini dapat terselesaikan.
3. Bapak Dr. Mahmud Yunus, M.Si selaku Ketua Program Studi S2 Matematika ITS.
4. Ibu Dr. Dwi Ratna S., S.Si, MT dan Bapak Dr. Chairul Imron, MI.Komp. selaku dosen penguji yang telah memberikan saran dan masukan demi perbaikan Tesis ini.
5. Seluruh jajaran dosen dan staf Departemen Matematika ITS.
6. Teman-teman seperjuangan yang saling mendukung dan memotivasi.
7. Semua pihak yang tidak bisa penulis sebutkan satu-persatu, terima kasih telah membantu sampai terselesaikannya Tesis ini.

Apabila dalam Tesis ini ada kekurangan, penulis mengharapkan kritik dan saran dari pembaca. Semoga Tesis ini dapat bermanfaat bagi semua pihak.

Wassalamu'alaikum wr. wb.

Surabaya, Juli 2019

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

Pada bab ini dijelaskan tentang hal yang melatarbelakangi penelitian ini. Selain itu, dipaparkan juga tentang rumusan dan batasan masalah. Kemudian dijelaskan tentang tujuan dan manfaat yang diperoleh dari penelitian ini.

1.1 Latar Belakang

Salah satu bidang teknologi yang berkembang sekarang ini adalah komputasi biologi. Komputasi biologi adalah bidang ilmu yang berfokus pada penyusunan sebuah model matematika dalam menyelesaikan dan menganalisis masalah sekuen biologi. Komputasi biologi atau dikenal sebagai bioinformatika merupakan kombinasi biologi dan komputasi yang menggunakan aplikasi dari alat komputasi dan analisis untuk menangkap dan menginterpretasikan data-data biologi. Bidang kajian bioinformatika yang sedang berkembang sekarang ini adalah *molecular docking*. *Molecular docking* dapat digunakan untuk mencari pola interaksi yang paling tepat antara dua molekul, yaitu reseptor dan ligan. *Molecular docking* bertujuan meniru peristiwa interaksi suatu molekul ligan dengan protein yang menjadi targetnya pada uji *in-vitro* (Gane dan Dean, 2000).

Peningkatan jumlah struktur biologi molekuler yang ada membuat *molecular docking* menjadi alat yang sangat penting dalam penemuan dan desain obat rasional berdasarkan struktur (Gane dan Dean, 2000). Salah satunya yaitu pada desain obat kanker. Kanker adalah penyakit akibat pertumbuhan tidak normal dari sel-sel jaringan tubuh yang berubah menjadi sel kanker. Sel-sel kanker ini dapat menyebar ke bagian tubuh lainnya sehingga dapat menyebabkan kematian. Terapi pengobatan kanker pada umumnya menggunakan terapi kombinasi (ko-kemoterapi) dengan obat/senyawa yang memiliki efek sinergis untuk mengobati sel kanker dan memiliki efek samping seminimal mungkin. Obat anti kanker yang tidak menimbulkan efek samping umumnya diperoleh dari alam. Riset untuk mendapatkan kandidat potensial obat anti kanker baru masih sangat diperlukan. Desain obat kanker diperlukan sebagai tahap seleksi awal serta untuk mengetahui *treatment* yang baik melalui interaksi antara protein-protein dan protein-ligan.

Molecular docking terbukti sangat efektif dalam studi interaksi protein-ligan. (Hou, dkk., 1999). *Docking* merupakan suatu permasalahan yang sulit dengan melibatkan banyak derajat kebebasan, sehingga perkembangan dari metode dan algoritma *docking* yang efisien akan menjadi sangat berguna dalam desain obat baru (de Magalhães, dkk., 2004). Algoritma optimasi sangat membantu dalam *docking* untuk mendapatkan desain obat secara simulasi. Metode-metode *artificial intelligence* telah banyak diterapkan

untuk permasalahan *molecular docking*, di antaranya yaitu *genetic algorithm, differential evolution, particle swarm optimization* (Camacho, dkk., 2015), *harmony search algorithm* (Reddy dan Reddy, 2013), *dan ant colony optimization* (Korb, dkk., 2007) serta metode *machine learning*, yaitu *extreme learning machine* (Mahdiyah, dkk., 2016).

Saat ini banyak bermunculan algoritma-algoritma *metaheuristic* yang berdasar pada kejadian-kejadian alami, biasa disebut dengan *nature-inspired metaheuristic algorithm*. *Nature-inspired metaheuristic algorithm*, terutama yang berdasar pada *swarm intelligence*, telah banyak dikembangkan dan dipelajari dalam beberapa tahun terakhir, salah satunya yaitu *firefly algorithm* yang dikembangkan pada tahun 2008. *Firefly algorithm* adalah algoritma yang berdasar pada pola dan perilaku cahaya dari kawanan *firefly* (kunang-kunang). Meskipun *firefly algorithm* memiliki banyak kemiripan dengan algoritma-algoritma *swarm intelligence* lainnya, seperti *particle swarm optimization* (PSO), *artificial bee colony optimization* (ABC), dan *bacterial foraging algorithm* (BF), namun *firefly algorithm* lebih sederhana dalam konsep dan implementasi (Apostolopoulos dan Vlachos, 2010). Untuk masalah optimasi, *firefly algorithm* biasanya memberikan hasil yang lebih baik dibandingkan algoritma-algoritma yang telah ada sebelumnya. *Firefly algorithm* telah digunakan untuk menyelesaikan berbagai macam permasalahan, di antaranya yaitu *travelling salesman problem* (TSP), *vehicle routing problem* (VRP), dan pelacakan objek dalam pengolahan citra. *Firefly algorithm* sangat efisien dan dapat mengungguli algoritma-algoritma konvensional seperti *genetic algorithm* (Ali, dkk., 2014).

Berdasarkan latar belakang di atas, maka dalam penelitian ini akan dikaji penggunaan *firefly algorithm* untuk menyelesaikan *molecular docking*. *Firefly algorithm* digunakan untuk mencari konformasi protein dan ligan yang optimal sehingga energi bebas dari sistem secara keseluruhan diminimalkan. Kasus yang akan diambil adalah *molecular docking* senyawa alkaloid SA2014 dari spons laut *Cinachyrella anomala* terhadap protein Cyclin D1 pada kanker.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, didapatkan rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana menyelesaikan *molecular docking* dengan *firefly algorithm*?
2. Bagaimana hasil *molecular docking* yang diperoleh dengan *firefly algorithm*?

1.3 Batasan Penelitian

Batasan masalah yang digunakan dalam penelitian ini adalah sebagai berikut:

1. *Molecular docking* yang digunakan adalah *semi-flexible docking*.
2. Jenis *molecular docking* yang dilakukan adalah *protein-ligand docking*.

3. Uji yang dilakukan adalah uji *in-silico*.

4. *Molecular docking* yang dilakukan menggunakan satu protein dan satu ligan.

1.4 Tujuan Penelitian

Berdasarkan permasalahan yang telah dirumuskan sebelumnya, tujuan dari pengerjaan penelitian ini adalah sebagai berikut:

1. Untuk menyelesaikan *molecular docking* dengan *firefly algorithm*.

2. Untuk menganalisis hasil *molecular docking* yang diperoleh dengan *firefly algorithm*.

1.5 Manfaat Penelitian

Manfaat yang bisa diperoleh dari pengerjaan penelitian ini adalah :

1. Adanya kajian baru tentang *firefly algorithm* untuk menyelesaikan *molecular docking*. Algoritma tersebut nantinya dapat dikembangkan lebih lanjut oleh peneliti lain.

2. Hasil dari penelitian ini diharapkan dapat memberikan kemudahan dalam pembuatan dan desain obat baru. Dalam uji *in-vitro* untuk pembuatan obat dibutuhkan biaya yang mahal. *Molecular docking* dapat mengurangi biaya yang mahal dan perhitungan yang kompleks.

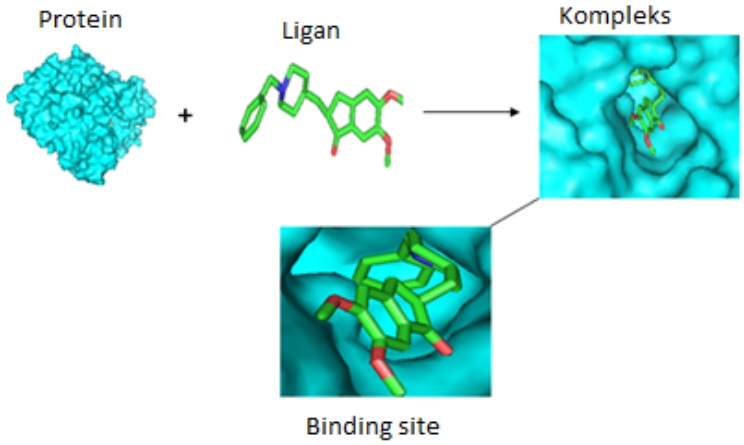
3. Mengetahui performansi dari *firefly algorithm* untuk menyelesaikan *molecular docking*.

BAB 2
KAJIAN PUSTAKA

Dalam bab ini dibahas mengenai kajian pustaka dan dasar teori yang berkaitan dengan topik penelitian tesis.

2.1 *Molecular Docking*

Dalam bidang pemodelan molekul, *docking* adalah metode untuk memprediksi orientasi yang tepat dari suatu molekul ketika terikat satu sama lain untuk membentuk kompleks yang stabil. Informasi tentang orientasi ini dapat digunakan untuk memprediksi kekuatan hubungan atau afinitas ikatan antara dua molekul. *Docking* sering digunakan untuk memprediksi orientasi ikatan kandidat obat bermolekul kecil terhadap target proteinnya untuk memprediksi afinitas dan aktivitas molekul kecil. *Docking* memainkan peran penting dalam desain obat secara rasional (Mukesh dan Rakesh, 2011).



Gambar 2.1: *Molecular Docking*

Molecular docking atau penambatan molekul mensimulasikan secara komputasi proses pengenalan molekul. *Molecular docking* dapat digambarkan seperti pada Gambar 2.1. Tujuan dari *molecular docking* adalah untuk mencapai konformasi yang optimal untuk protein dan ligan serta orientasi relatif antara protein dan ligan sehingga energi bebas dari sistem secara keseluruhan diminimalkan. Proses komputasi mencari ligan yang cocok baik secara geometris dan energi ke *binding site* dari protein ini disebut *molecular docking*. *Molecular docking* membantu dalam mempelajari obat/ligan atau interaksi reseptor/protein dengan mengidentifikasi *active site* yang cocok pada

protein, mendapatkan geometri terbaik dari kompleks ligan-reseptor, dan menghitung energi interaksi dari ligan yang berbeda untuk merancang ligan yang lebih efektif.

Keberhasilan program *docking* bergantung pada dua komponen yaitu algoritma pencarian dan *scoring function* (Mukesh dan Rakesh, 2011). *Scoring function* digunakan untuk menghitung afinitas kompleks ligan-protein yang terbentuk dan untuk mengurutkan peringkat senyawa. Identifikasi ini didasarkan pada beberapa teori seperti teori energi bebas Gibbs. Nilai energi bebas Gibbs yang kecil menunjukkan bahwa konformasi yang terbentuk adalah stabil, sedangkan nilai energi bebas Gibbs yang besar menunjukkan tidak stabilnya kompleks yang terbentuk. Sedangkan penggunaan algoritma berperan dalam penentuan konformasi (*docking pose*) yang paling stabil dari pembentukan kompleks (Funkhouser, 2007). Gugus-gugus fungsional ligan akan berinteraksi dengan residu-residu asam amino protein reseptor sehingga membentuk ikatan intermolekular. Kekuatan ikatan inilah yang dihitung dan diperingkatkan (*ranking*) dengan *scoring function*.

Penelitian tentang metode komputasi untuk membuat kombinasi protein dan ligan sudah banyak dilakukan. Berbagai metode komputasi telah dikembangkan untuk menyelesaikan permasalahan *molecular docking*, di antaranya yaitu:

1. *Flexible Ligand Docking Using Differential Evolution* (Thomsen, 2003)
   Penelitian ini menerapkan *Differential Evolution Algorithm* (DE) pada permasalahan *molecular docking* dengan menggunakan program AutoDock. Algoritma DockDE dibandingkan dengan Lamarckian GA (LGA) yang disediakan oleh AutoDock. Perbandingan dilakukan pada enam masalah *docking* yang umum digunakan. Hasilnya menunjukkan bahwa DockDE mengungguli algoritma yang lain pada semua masalah. DockDE menunjukkan kinerja yang luar biasa dalam kecepatan konvergensi dan ketahanan mengenai solusi yang ditemukan.

2. *A Genetic Algorithm for the Ligand-Protein Docking Problem* (de Magalhães, dkk., 2004)
   Penelitian ini mengembangkan metode *semi-flexible docking* yang menggunakan algoritma genetika yang disebut *"steady-state" genetic algorithm* (SSGA) pada lima kompleks protease-ligan HIV-1 yang memiliki struktur tiga dimensi yang telah diketahui. SSGA diuji untuk *docking* dengan ligan yang *rigid* dan fleksibel. Algoritma genetika yang diimplementasikan mampu melakukan *docking* dengan sukses pada molekul yang *rigid* dan fleksibel.

3. *An Ant Colony Optimization Approach to Flexible Protein-Ligand Docking* (Korb, dkk., 2007)
   Penelitian ini mengenalkan sebuah algoritma *docking* yang disebut PLANTS (*Protein-Ligand ANT System*) yang berdasarkan pada *Ant Colony Optimization*. Dalam penelitian ini dipelajari keefektifan PLANTS dengan mengatur beberapa parameter dan menyajikan

perbandingan kinerja PLANTS dengan GOLD, suatu program yang berdasarkan pada algoritma genetika dan sering digunakan dalam industri farmasi. Hasilnya menunjukkan bahwa PLANTS dapat melakukan docking secara efektif pada protein kinase A.

4. *Solving Molecular Flexible Docking Problems with Metaheurustics: A Comparative Study* (Camacho, dkk., 2015)
   Penelitian ini membandingkan metode metaheuristik dalam menyelesaikan *flexible docking*. Metode metaheuristik yang dipilih adalah *steady state Genetic Algorithm, Differential Evolution*, dan *Particle Swarm Optimization*. Metode-metode ini dievaluasi dan dibandingkan dengan algoritma *docking* yang terkenal yaitu AutoDock 4.2. Hasil pengujian menunjukkan bahwa *Differential Evolution* memperoleh hasil terbaik secara keseluruhan, bahkan mengungguli algoritma lain yang secara khusus dirancang untuk *molecular docking*.

2.1.1 Klasifikasi *Molecular Docking* Berdasarkan Jenis Ligan

Berdasarkan jenis ligannya, metode *molecular docking* dibedakan menjadi beberapa golongan, yaitu:

1. *Protein-protein docking* (protein reseptor dengan protein ligan)
2. *Protein-ligand docking*
3. *Protein-small molecule docking* (protein reseptor dengan molekul kecil)
4. *Protein-DNA/RNA docking* (protein reseptor dengan asam nukleat)

2.1.2 Klasifikasi *Molecular Docking* Berdasarkan Fleksibilitas Molekul

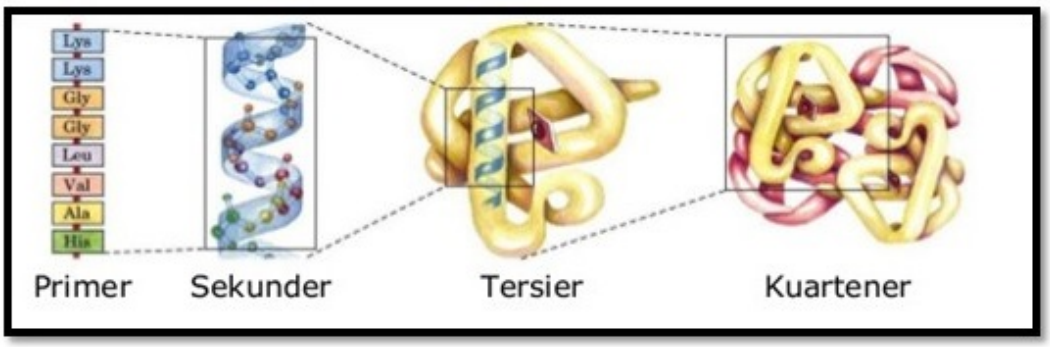
Berdasarkan fleksibilitas molekulnya, motede *molecular docking* dibedakan menjadi beberapa golongan, yaitu:

1. *Rigid docking*
   Rigid docking adalah salah satu cabang dari metode *molecular docking* yang memposisikan molekul yang akan dipasangkan sebagai molekul yang bersifat *rigid*. Definisi *rigid* adalah suatu kondisi di mana kedua molekul yang digunakan dikondisikan untuk tetap (tidak mengalami perubahan konformasi) selama proses *docking* sedang berlangsung.
2. *Semi-flexible docking*
   Semi-flexible docking adalah metode *molecular docking* yang memperlakukan dua molekul yang akan dipasangkan secara berbeda. Salah satu molekul (biasanya molekul yang berukuran lebih kecil; ligan) diperlakukan sebagai molekul yang fleksibel, sedangkan molekul yang berukuran lebih besar (protein reseptor) diperlakukan sebagai molekul yang *rigid*.
3. *Flexible docking*
   Flexible docking adalah metode *molecular docking* yang memperlakukan dua molekul yang akan dipasangkan sebagai molekul yang fleksibel.

2.2 Protein

Protein adalah makromolekul yang paling banyak ditemukan di dalam sel makhluk hidup dan merupakan 50 persen atau lebih dari berat sel kering. Protein dan gen memiliki hubungan yang sangat dekat. Kode genetika berupa DNA dienkripsi dalam bentuk kromosom yang selanjutnya kode genetika tersebut ditranslasikan menjadi protein.

Protein tersusun dari peptida-peptida sehingga membentuk suatu polimer yang disebut polipeptida. Setiap monomernya tersusun atas suatu asam amino. Ada 20 macam asam amino yang dapat ditemui di alam. Asam amino yang berbeda-beda disusun sedemikian hingga, sehinggga terbentuklah struktur utama protein (Shen dan Tuszynki, 2008). Struktur protein terbagi menjadi 4, yaitu struktur utama atau primer, struktur sekunder, struktur tersier, dan struktur kuartener. Struktur utama adalah rantai asam amino dari protein. Struktur sekunder adalah bentuk berlekuk yang diakibatkan oleh ikatan hidrogen pada peptida. Struktur tersier adalah bentuk 3 dimensi protein, yang diakibatkan reaksi antar rantai peptida. Sedangkan struktur kuartener adalah bentuk 3 dimensi untuk protein yang memiliki rantai asam amino. Perbedaan bentuk struktur protein dapat dilihat pada Gambar 2.2.



Gambar 2.2: Struktur Protein

2.3 Ligan

Ligan adalah molekul sederhana yang dalam senyawa kompleks bertindak sebagai donor pasangan elektron (basa Lewis). Ligan akan memberikan pasangan elektronnya kepada atom pusat yang menyediakan orbital kosong. Interaksi antara ligan dan atom pusat menghasilkan ikatan koordinasi. Ligan adalah sebuah molekul sinyal kecil yang terlibat dalam proses anorganik dan biokimia. Dalam kimia koordinasi, ligan memungkinkan pembentukan kompleks koordinasi, atau hubungan antara molekul yang berbeda dalam larutan. Dalam biokimia umumnya mendefinisikan ligan sebagai molekul *messenger*, seperti hormon, substrat, atau aktivasi dan faktor inhibisi (Horton, dkk., 2006).

2.4 *Firefly Algorithm*

Firefly algorithm pertama kali dikembangkan oleh Xin-She Yang pada tahun 2008 di Cambridge University, yang didasarkan pada pola berkedip dan perilaku kunang-kunang. Terdapat sekitar dua ribu spesies kunang-kunang, dan kebanyakan dari mereka mengeluarkan cahaya singkat yang mempunyai ritme tertentu. Pola dari ritme tersebut biasanya berbeda-beda, tergantung dari spesiesnya. Ritme cahaya tersebut mempunyai dua fungsi utama, yaitu untuk menarik perhatian kunang-kunang lain dan untuk menarik perhatian mangsa. Intensitas cahaya yang berjarak r dari pusat cahayanya mengikuti hukum *inverse-square*, yaitu intensitas cahaya mengecil seiring bertambahnya r. Intensitas cahaya juga mengecil karena terserap oleh udara. Hal ini menyebabkan kunang-kunang dapat melihat kawannya dalam jarak maksimal tertentu.

Firefly algorithm bekerja dengan memperhatikan aturan-aturan berikut ini:

1. Semua kunang-kunang adalah *unisex*, yaitu setiap kunang-kunang akan tertarik dengan kunang-kunang lain tanpa memperhatikan jenis kelaminnya.

2. Besarnya ketertarikan sebanding dengan intensitas cahaya yang dikeluarkan oleh kunang-kunang. Kunang-kunang dengan cahaya yang lebih redup akan bergerak kepada yang lebih terang. Intensitas cahaya suatu kunang-kunang akan berkurang seiring dengan bertambahnya jarak. Apabila tidak ada kunang-kunang yang lebih terang dari yang lain, maka kunang-kunang akan bergerak secara acak.

3. Intensitas cahaya dari kunang-kunang ditentukan oleh fungsi objektif dari masalah yang diberikan.

Berikut ini beberapa istilah yang digunakan dalam *Firefly algorithm* dan definisinya (Yang, 2014).

1. Populasi adalah sebuah kumpulan solusi yang direpresentasikan dengan kunang-kunang (*firefly*).

2. *Firefly* adalah individu dalam populasi yang terdiri kumpulan kode yang merepresentasikan solusi dari permasalahan.

3. Intensitas cahaya adalah nilai atau ukuran untuk mengevaluasi *firefly*.

4. *Attractiveness* adalah daya tarik seekor kunang-kunang yang dinilai oleh kunang-kunang lainnya berdasarkan intensitas cahayanya.

5. *Distance* adalah jarak antara dua *firefly*.

6. *Movement* adalah pergerakan yang dilakukan masing-masing *firefly* menuju *firefly* lain yang intensitas cahayanya lebih terang.

2.4.1 Intensitas Cahaya

Dalam *firefly algorithm* terdapat dua masalah penting yaitu variasi intensitas cahaya dan perumusan *attractiveness*. Kecerahan pada kunang-kunang akan ditentukan oleh fungsi tujuan dan *attractiveness* sebanding dengan kecerahan. Dengan demikian untuk setiap dua kunang-kunang yang berkedip, kunang-kunang dengan cahaya yang kurang terang akan bergerak ke arah kunang-kunang yang cahayanya lebih terang.

Attractiveness β bernilai relatif, karena intensitas cahaya harus dilihat dan dinilai oleh kunang-kunang lain. Dengan demikian, hasil penilaian akan berbeda tergantung dari jarak antara kunang-kunang yang satu dengan yang lainnya (r_{ij}). Selain itu, intensitas cahaya akan menurun dari sumbernya dikarenakan terserap oleh media, misalnya udara. Sehingga dapat ditentukan *attractiveness* (β) dengan jarak r sebagai berikut:

$$\beta = \beta_0 e^{-\gamma r^2} \tag{2.1}$$

dengan β_0 adalah daya tarik di saat $r = 0$ dan $\gamma \in [0, \infty)$ adalah koefisien penyerapan cahaya.

2.4.2 *Distance*

Distance adalah jarak antara dua kunang-kunang i dan j pada posisi x_i dan x_j. Jarak kartesian antara dua kunang-kunang tersebut dirumuskan sebagai berikut (Yang, 2014):

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{n} (x_i^k - x_j^k)^2} \tag{2.2}$$

dengan x_i^k adalah komponen ke-k dari x_i pada kunang-kunang i.

2.4.3 *Movement*

Movement adalah pergerakan yang dilakukan kunang-kunang i karena ketertarikan terhadap kunang-kunang lain j, yang intensitas cahayanya lebih terang. Dengan adanya *movement*, maka posisi kunang-kunang atau solusi dari permasalahan akan berubah sesuai rumus berikut (Yang, 2014):

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2} (x_j^t - x_i^t) + \alpha \epsilon_i^t \tag{2.3}$$

dengan suku pertama merupakan posisi lama dari kunang-kunang, suku kedua menyatakan pergerakan ke arah kunang-kunang j dan suku ketiga menyatakan pergerakan acak dengan α adalah parameter, dan ϵ_i adalah vektor bilangan acak yang diambil dari distribusi normal atau distribusi uniform. r_{ij} adalah jarak antara kunang-kunang i dan kunang-kunang j.

2.4.4 Langkah-Langkah *Firefly Algorithm*

Langkah-langkah dalalm *firefly algorithm* adalah sebagai berikut:

1. Inisialisasi parameter *firefly algorithm*.

2. Membangkitkan secara random populasi awal sebanyak m *firefly*. Hitung intensitas cahaya tiap *firefly* $I(x)$ berdasarkan nilai fungsi tujuan $f(x)$.

3. Membandingkan intensitas cahaya tiap *firefly* dengan *firefly* lainnya. Apabila terdapat *firefly* yang intensitas cahayanya lebih besar, akan dilakukan *update* pergerakan *firefly* menggunakan persamaan *movement* (2.3).

4. Menentukan G-best. Untuk iterasi pertama, *firefly* terbaik (*firefly* dengan intensitas cahaya terbesar) adalah G-best.

5. Membandingkan *firefly* terbaik tiap iterasi dengan G-best yang diperoleh. Apabila intensitas cahaya *firefly* terbaik saat itu lebih besar dari pada G-best, maka *firefly* tersebut menjadi G-best.

6. Melakukan *movement* kepada *firefly* terbaik dan menggabungkannya dengan *firefly* yang lain untuk menjadi populasi awal pada iterasi selanjutnya.

7. Melakukan proses di atas sampai batas iterasi dipenuhi.

Langkah-langkah *firefly algorithm* dapat diringkas seperti yang ditunjukkan dalam *pseudo code* dalam Gambar 2.3.

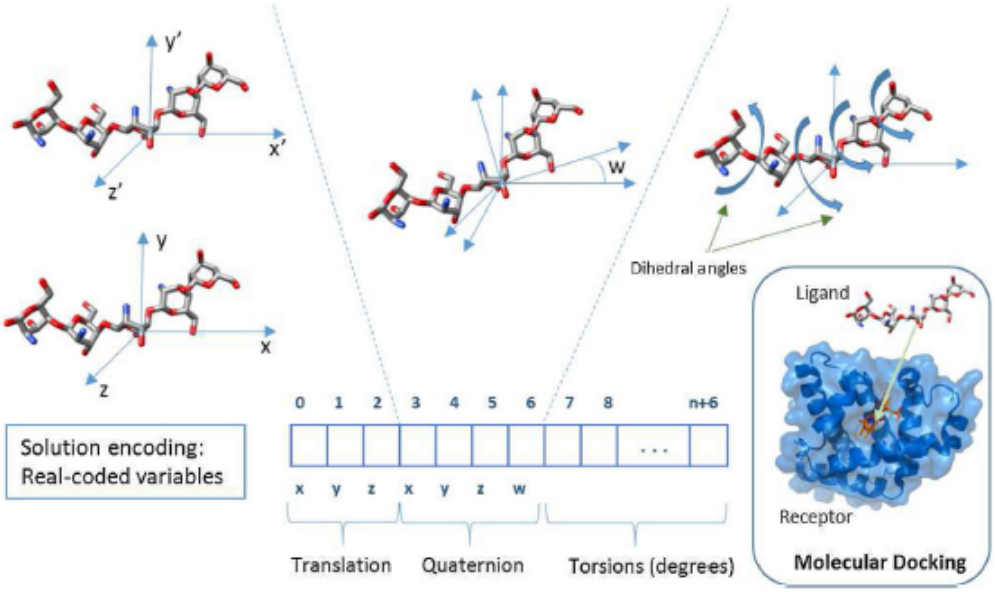
```
Fungsi objektif f(x), x = (x_1, ..., x_d)^T
Inisialisasi populasi firefly x_i (i = 1, 2, ..., n)
Intensitas cahaya I_i pada x_i ditentukan oleh f(x_i)
Tentukan nilai γ, α, dan β_0

while (t <MaxGeneration)
for i = 1 : n
      for j = 1 : n
      if (I_i < I_j), pindahkan firefly i menuju j; end if
            Ubah ketertarikan pada jarak r dengan exp[−γr]
            Evaluasi solusi baru dan perbarui intensitas cahaya
      end for j
end for i
Urutkan peringkat firefly dan temukan solusi terbaik G-best
end while
```

Gambar 2.3: *Pseudo Code* untuk *Firefly Algorithm*

2.5 Fungsi Objektif untuk *Molecular Docking*

Tujuan utama dalam *molecular docking* adalah menemukan konformasi yang optimal antara ligan dan reseptor yang menghasilkan energi ikatan minimum. Interaksi antara ligan dan reseptor dapat dijelaskan oleh fungsi objektif yang ditentukan berdasarkan tiga komponen yang mewakili derajat kebebasan: (1) translasi dari molekul ligan yang melibatkan tiga sumbu (x, y, z) dalam koordiant Kartesius; (2) orientasi ligan yang dimodelkan sebagai empat variabel quaternion termasuk kemiringan sudut (w); dan (3) fleksibilitas, diwakili oleh rotasi bebas dari torsi (sudut dihedral) dari ligan dan rantai samping dari reseptor. Untuk variabel x, y, z dari lokasi translasi dan quaternion, digunakan satuan Å(angstrom) atau 10^{-10} meter. Sedangkan untuk w dan sudut rotasi titik ligan, digunakan radian, dengan besaran $[-\pi, \pi]$. Dengan demikian representasi solusi untuk permasalahan *molecular docking* adalah seperti pada Gambar 2.4.



Gambar 2.4: Representasi Solusi Permasalahan *Molecular Docking*

Fungsi objektif atau *scoring function* dalam *molecular docking* adalah meminimumkan total energi ikatan dari kompleks ligan-makromolekul berdasarkan persamaan berikut (Camacho, dkk., 2015)

$$\begin{aligned} \Delta G &= ({V_{bonded}}^{L-L} - {V_{unbonded}}^{L-L}) + ({V_{bonded}}^{R-R} - {V_{unbonded}}^{R-R}) \\ &\quad +({V_{bonded}}^{R-L} - {V_{unbonded}}^{R-L} + \Delta G_{conf}) \end{aligned} \tag{2.4}$$

$$\begin{aligned} V &= W_{vdw}\Sigma_{i,j}(\frac{A_{ij}}{{r_{ij}}^{12}} - \frac{B_{ij}}{{r_{ij}}^{6}}) + W_{hbond}\Sigma_{i,j}E(t)(\frac{C_{ij}}{{r_{ij}}^{12}} - \frac{D_{ij}}{{r_{ij}}^{6}}) \\ &\quad +W_{elec}\Sigma_{i,j}\frac{q_i q_j}{\varepsilon(r_{ij})r_{ij}} + W_{sol}\Sigma_{i,j}(S_i V_j + S_j V_i)e^{(-\frac{{r_{ij}}^2}{2\sigma^2})} \end{aligned} \tag{2.5}$$

$$\Delta G_{conf} = W_{conf} N_{tors} \tag{2.6}$$

Seperti yang dirumuskan dalam persamaan (2.4), energi bebas dari ikatan dihitung dari perbedaan antara ligan (L) dan protein (R) dalam keadaan terikat dan tak terikat. Setiap pasangan dari evaluasi energik termasuk evaluasi dispersi/tolakan (vdw), ikatan hidrogen (hbond), elektrostatik (elec), dan desolvasi (sol). Bobot W_{vdw},W_{hbond},W_{conf},W_{elec}, dan W_{sol} pada persamaan (2.5) dan (2.6) adalah konstanta untuk van der Waals, ikatan hidrogen, gaya torsional, interaksi elektrostatik, dan desolvasi. Dalam persamaan (2.5), r_ij menyatakan jarak *interatomic*, A_{ij} dan B_{ij} adalah parameter Lennard-Jones. Demikian pula C_{ij} dan D_{ij} adalah parameter Lennard-Jones untuk energi potensial dengan kedalaman maksimum antara dua atom, dan $E(t)$ menyatakan direksionalitas bergantung sudut. Bentuk ketiga pada persamaan (2.5) menggunakan pendekatan Coulomb untuk elektrostatik. Bentuk keempat dihitung dari volume (V) atom yang mengelilingi sebuah atom yang diberikan dengan bobot S, dan suatu bentuk eksponensial yang melibatkan jarak atom.

2.6 Transformasi Tiga Dimensi

Transformasi dapat diartikan sebagai suatu metode yang dapat digunakan untuk memanipulasi lokasi sebuah titik. Apabila transformasi dikenakan terhadap sekumpulan titik yang membentuk sebuah benda (objek) maka benda tersebut akan mengalami perubahan. Perubahan dalam hal ini adalah perubahan dari lokasi awal suatu benda menuju lokasi yang baru dari benda tersebut. Sebuah benda dengan titik P ditransformasikan ke titik Q menggunakan rumus-rumus tertentu, sehingga titik Q merupakan lokasi baru dari titik P (Rawuh, 1993).

Transformasi tiga dimensi menggunakan tiga buah sumbu yaitu sumbu x, sumbu y, dan sumbu z. Sumbu x mewakili ukuran panjang, sumbu y mewakili ukuran lebar, sedangkan sumbu z mewakili kedalaman. Transformasi tiga dimensi dapat berupa:

1. Translasi
   Translasi dapat diartikan dengan perubahan lokasi dari lokasi awal menuju ke lokasi baru. Titik $A(x, y, z)$ digeser sejumlah Trx pada sumbu x, digeser sejumlah Try pada sumbu y, dan digeser sejumlah Trz pada sumbu z. Translasi pada benda (objek) tiga dimensi dapat menggunakan rumus sebagai berikut:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} Trx \\ Try \\ Trz \end{pmatrix} \tag{2.7}$$

2. Skala
   Skala berfungsi untuk memperbesar atau memperkecil benda (objek)

sesuai dengan ukuran yang diigninkan. Skala pada benda (objek) tiga dimensi dapat dilakukan dengan menggunakan matriks sebagai berikut:

$$M_s = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{pmatrix} \tag{2.8}$$

3. Rotasi
   Berbeda dengan rotasi dua dimensi yang menggunakan titik pusat $(0, 0)$ sebagai pusat perputaran, rotasi pada benda (objek) tiga dimensi menggunakan sumbu koordinat sebagai pusat perputaran. Karena pada objek tiga dimensi terdapat tiga buah sumbu koordinat, maka terdapat tiga macam rotasi yang dapat dilakukan, yaitu:

   (a) Rotasi sumbu x
   Rotasi pada sumbu x menggunakan matriks sebagai berikut:

$$M_{rx} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{2.9}$$

   (b) Rotasi sumbu y
   Rotasi pada sumbu y menggunakan matriks sebagai berikut:

$$M_{ry} = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \tag{2.10}$$

   (c) Rotasi sumbu z
   Rotasi pada sumbu z menggunakan matriks sebagai berikut:

$$M_{rz} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.11}$$

2.7 Rotasi Vektor Menggunakan Bilangan Quaternion

Bilangan quaternion mulai diperkenalkan oleh Sir William Rowan Hamilton pada tahun 1843. Bilangan quaternion merupakan jawaban para ilmuwan yang mencoba mencari ekivalen tiga dimensional dari bilangan kompleks. Quaternion didefinisikan dalam bentuk (Arwoko, dkk., 2018):

$$z = a + ib + jc + kd$$

Objek ini disebut dengan 'quaternion', merupakan penjumlahan aljabar dari bagian riil (nilai a) dan bagian imajinernya (nilai b, c, d).

Jika ada dua quaternion q_1 dan q_2 sebagai berikut:

$$q_1 = s_1 + ix_1 + jy_1 + kz_1$$

$$q_2 = s_1 + ix_2 + jy_2 + kz_2$$

maka perkalian kedua quaternion adalah:

$$q_1 q_2 = s_1 s_2 - v_1 \cdot v_2 + s_1 v_2 + s_2 v_1 + v_1 \times v_2$$

Rotasi vektor merupakan salah satu aplikasi dari bilangan quaternion. Suatu vektor p akan dirotasikan sejauh sudut θ pada sumbu yang diberikan oleh unit vektor u. Vektor p yang telah dirotasikan melalui sumbu u menghasilkan vektor baru p' didapat dengan operasi seperti berikut:

$$p' = qpq^{-1}$$

dengan

$$p = xi + yj + zk$$

$$q = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})u$$

$$q^{-1} = \cos(\frac{\theta}{2}) - \sin(\frac{\theta}{2})u$$

$$i^2 = j^2 = k^2 = -1$$

$$ijk = -1, ij = k, jk = i, ki = j$$

dapat dimanipulasi secara aljabar ke dalam bentuk $p' = Rp$, dengan R merupakan suatu matriks rotasi yang diberikan oleh:

$$R = \begin{pmatrix} 1 - 2s(y^2 + z^2) & 2s(xy - z\theta) & 2s(xz + y\theta) \\ 2s(xy + z\theta) & 1 - 2s(x^2 + z^2) & 2s(yz - x\theta) \\ 2s(xz - y\theta) & 2s(yz + x\theta) & 1 - 2s(x^2 + y^2) \end{pmatrix} \tag{2.12}$$

2.8 Interpolasi Trilinier

Interpolasi adalah metode untuk memperkirakan nilai suatu titik jika nilai fungsi hanya diketahui untuk titik-titik di sekitarnya dan bukan nilai fungsi untuk titik tersebut. Interpolasi yang paling umum dan dasar adalah interpolasi linier. Jika fungsi yang diinterpolasi adalah linier, maka nilai yang diperoleh melalui interpolasi linier adalah tepat, namun jika tidak, maka nilai yang diperoleh adalah nilai estimasi. Interpolasi linier banyak digunakan dalam berbagai bidang untuk mengestimasi nilai karena kesederhanaan dan biaya komputasinya yang rendah. Interpolasi linier dapat dihitung sebagai berikut (Kadlubiak, 2015):
Misalkan fungsi linier

$$y = f(x)$$

dan dua titik x_0 dan x_1, maka nilai dari titik x adalah

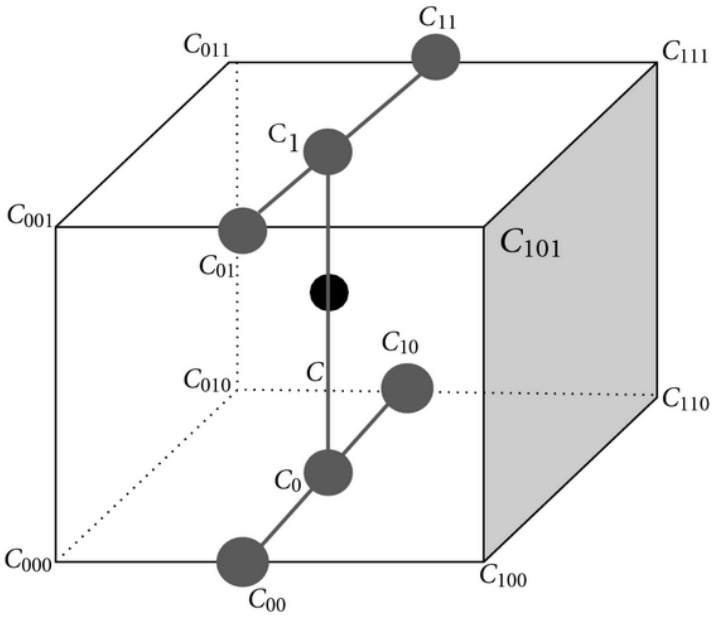
$$f(x) = y_0(1 - m) + y_1 m$$

dengan

$$m = \frac{x - x_0}{x_1 - x_0}$$

Secara umum, interpolasi dari fungsi berdimensi tinggi dapat dinyatakan sebagai sekumpulan interpolasi linier pada masing-masing dimensi. Prinsip-prinsip interpolasi linier yang diterapkan pada bidang tiga dimensi disebut interpolasi trilinier.

Misalkan kita memiliki titik-titik pada kubus: $P_{000}, P_{100}, P_{010}, P_{110}, P_{001}, P_{101}, P_{011}, P_{111}$ dan nilai dari titik-titik ini adalah $C_{000}...C_{111}$ seperti pada Gambar 2.5 berikut.



Gambar 2.5: Interpolasi Trilinier

Nilai interpolasi C dari titik P yang terletak di dalam kubus dapat dihitung sebagai berikut.
Pertama kita intepolasi sepanjang x.

$$C_{00} = C_{000} * (1 - d_x) + C_{100} * d_x$$

$$C_{10} = C_{010} * (1 - d_x) + C_{110} * d_x$$

$$C_{01} = C_{001} * (1 - d_x) + C_{101} * d_x$$

$$C_{11} = C_{011} * (1 - d_x) + C_{111} * d_x$$

dengan

$$d_x = \frac{P_x - P_{000_x}}{P_{100_x} - P_{000_x}}$$

Kemudian kita interpolasi nilai-nilai ini sepanjang sumbu y.

$$C_0 = C_{00} * (1 - d_y) + C_{10} * d_y$$

$$C_1 = C_{01} * (1 - d_y) + C_{11} * d_y$$

dengan

$$d_y = \frac{P_y - P_{000_y}}{P_{010_y} - P_{000_y}}$$

Kemudian kita interpolasi nilai-nilai ini sepanjang sumbu z sehingga nilai C dapat dihitung sebagai berikut.

$$C = C_0 * (1 - d_z) + C_1 * d_z \tag{2.13}$$

dengan

$$d_z = \frac{P_z - P_{000_z}}{P_{001_z} - P_{000_z}}$$

2.9 Kanker

Kanker adalah suatu penyakit yang ditandai dengan pembelahan sel tidak terkendali dan kemampuan sel menyerang jaringan biologis lainnya, baik pertumbuhan langsung di jaringan tetangganya (invasif) maupun migrasi sel ke tempat yang lebih jauh (metastasis) (Diandana, 2009). Perbedaan sel normal dan sel kanker dapat dilihat pada Gambar 2.6. Sel kanker timbul dari sel normal tubuh yang mengalami transformasi menjadi ganas, karena adannya mutasi spontan atau induksi karsinogen. Transformasi sel tersebut terjadi karena mutasi gen yang mengatur pertumbuhan dan diferensiasi sel, yaitu proto-onkogen atau supresor gen (anti onkogen). Sedangkan paparan karsinogen antara lain berbagai jenis virus, bahan kimia dan radiasi, serta ultraviolet (Franks dan Teich, 1998).

Sel kanker memiliki karakteristik sebagai berikut (Hanahan dan Weinberg, 2011):

1. Sel kanker dapat mencukupi persyaratan dalam hal sinyal pertumbuhan untuk dirinya sendiri. Sinyal pertumbuhan dibutuhkan untuk berpoliferasi secara terus menerus. Berbeda dari sel normal, sel-sel kanker bisa tinggal dan terus tumbuh.

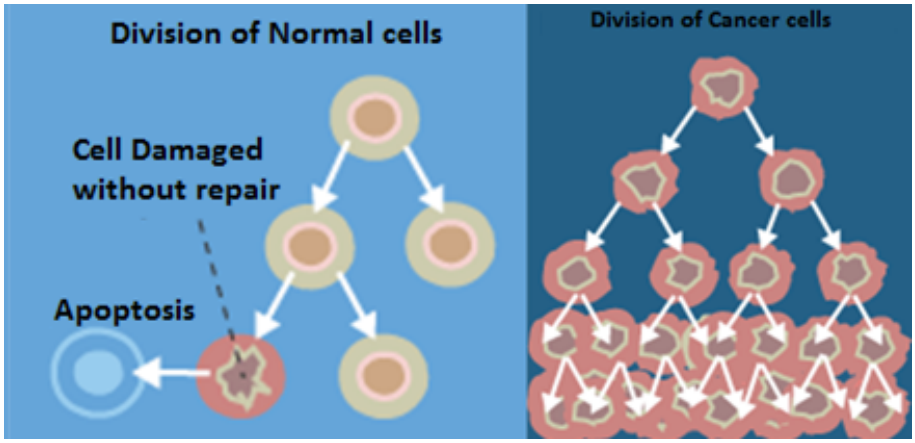
2. Sel tidak sensitif terhadap sinyal anti-pertumbuhan. Sel kanker tidak merespon sinyal sel yang memberhentikan pertumbuhan dan pembelahan sel.

3. Sel kanker dapat menghindar dari mekanisme apoptosis. Apoptosis merupakan program kematian sel ketika sel mengalami kerusakan baik secara struktural maupun secara fungsional yang tidak bisa ditoleransi lagi. Sel kanker dapat menghindar dari kematian sel dengan memblokir jalur apoptosis yang terjadi di dalam sel.

4. Sel kanker memiliki potensi yang tidak terbatas untuk menginduksi replikasi.

5. Sel kanker dapat menginduksi angiogenesis untuk memenuhi kebutuhan oksigen dan nutrisi. Pada tingkat perkembangan tumor, sel-sel tumor hiper-proliferasi akan mengekspresikan protein pro-angiogenik sehingga akan terbentuk cabang baru dari pembuluh darah ke sel-sel kanker yang kemudian akan memasok kebutuhan nutrisi dan oksigen pada sel kanker.

6. Sel kanker dapat menginvasi jaringan sekitar dan membentuk jaringan plot kanker yang baru.



Gambar 2.6: Perbedaan Sel Normal dan Sel Kanker

Kanker secara garis besar dibagi menjadi dua kelompok, yaitu kanker jinak dan kanker ganas. Kanker jinak (benigna) memiliki kecenderungan untuk tumbuh lebih lambat dari kanker ganas dan tidak menyebar ke organ lain. Kanker ganas (maligna) memiliki pertumbuhan sel yang sangat cepat, dapat menginvasi serta menghancurkan jaringan di sekitarnya dan pada tahap selanjutnya akan menyebar ke organ-organ lain (Lumongga, 2008).

2.10 Siklus Sel

Setiap sel baik normal maupun kanker mengalami siklus sel. Siklus sel pada sel eukaryotik merupakan suatu tahapan kompleks meliputi penggandaan materi genetik, pengaturan waktu pembelahan sel, dan interaksi antara protein dan enzim. Siklus sel pada sel eukaryotik dapat dibagi menjadi empat tahap, yaitu: G_1 (Gap 1), S (Sitesis), G_2 (Gap 2), dan M (Mitosis). Tahap G_1 merupakan selang antara tahap M dan S. Pada tahap G_1 sel terus tumbuh dan melakukan persiapan untuk sintesis DNA. Sel akan melakukan sintesis DNA dan terjadi proses replikasi kromosom pada saat berada di tahap S. Pada tahap G_2, sel yang telah mereplikasi kromosom akan menduplikasi keseluruhan komponen seluler lainnya. Selain itu terjadi pula sintesis mRNA dan beberapa

protein tertentu. Sedangkan pembelahan sel atau sering disebut dengan tahap mitosis terdiri dari empat sub tahapan, yaitu profase, metafase, anafase, dan telofase. Pada kondisi tertentu, sel-sel yang tidak membelah karena tidak berdiferensiasi, meninggalkan tahap G_1 dan pindah ke dalam tahap G_0. Sel-sel yang berada dalam tahap G_0 sering disebut sedang beristirahat/diam (Murti, dkk., 2007).

Setiap tahap dalam siklus sel dikontrol secara ketat oleh regulator siklus sel, yaitu (Vermeulen, dkk., 2003):

1. Cyclin Jenis cyclin utama dalam siklus sel adalah cyclin D, E, A, dan B. Cyclin diekspresikan secara periodik sehingga konsentrasi cyclin berubah-ubah pada setiap fase siklus sel. Berbeda dengan cyclin yang lain, cyclin D tidak diekspresikan secara periodik akan tetapi selalu disintesis selama ada stimulasi *growth-factor*.

2. Cyclin-dependent kinases (CDK) CDK utama dalam siklus sel adalah CDK 4, 6, 2, dan 1. CDK merupakan treonin atau serin protein kinase yang harus berikatan dengan cyclin untuk aktivasinya. Konsentrasi CDK relatif konstan selama siklus sel berlangsung. CDK dalam keadaan tak berikatan bersifat tidak aktif karena situs katalitik tempat ATP dan substrat berikatan diblok oleh ujung C-terminal dari CKI. Cyclin akan menghilangkan pengeblokan tersebut. Ketika diaktifkan, CDK akan memacu proses *downstream* dengan cara memfosforilasi protein spesifik.

3. Cyclin-dependent kinase inhibitor (CKI) CKI merupakan protein yang dapat menghambat aktivitas CDK dengan cara mengikat CDK atau kompleks cyclin-CDK. CKI terdiri dari dua kelompok protein yaitu INK4 (p15, p16, p18, dan p19) dan CIP/KIP (p21, p27, p57). INK4 bertugas mencegah progresi fase G_1, sedangkan CIP/KIP meregulasi fase G_1 dan S dengan menghambat kompleks G_1 cyclin-CDK dan cyclin B-CDK1.

2.11 Cyclin D1

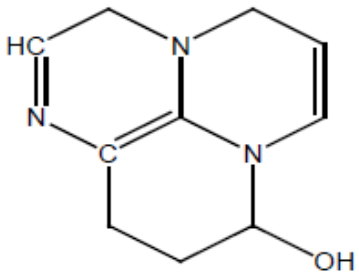
Protein cyclin D merupakan salah satu regulator positif pada siklus sel. Cyclin D terdiri atas tiga tipe yakni cyclin D1, D2, dan D3. Pada siklus sel, cyclin D1 tidak hanya berperan pada saat fase G_1 saja. Pada fase G_2 tingkat cyclin D1 cenderung meningkat, sedangkan pada fase S tingkat cyclin D1 cenderung rendah. Aktivitas cyclin D1 pada pada fase G_2 bergantung pada sinyal *proliferative* yang akan menginduksi cyclin D1. Pada fase ini sel akan menentukan titik poin untuk melanjutkan proliferasinya, ketika tidak terdapat cyclin D1 maka siklus sel akan memasuki fase istirahat. Pada fase S keberadaan cyclin D1 harus ditekan karena cyclin D1 memiliki kemampuan untuk menghambat sintesis DNA dilihat dari kemampuannya yang dapat berikatan dengan regulator penting sintesis DNA, PCNA (Yang, dkk., 2006). Bentuk dari gen Cyclin D1 dapat dilihat pada Gambar 2.7.

Gambar 2.7: Gen CCND1

Cyclin D1 diketahui berkorelasi dengan gejala awal kanker serta resiko perkembangan tumor dan metastasis. Gen cyclin D1, CCND1 mengalami amplifikasi sebesar 20 % dan terdapat ekspresi protein cyclin D1 secara berlebihan pada kanker. Hasil overekspresi cyclin dapat diinduksi dari sinyal onkogenik maupun berasal dari mutasi pada gen cyclin. Hal ini mengakibatkan adanya pertumbuhan yang berlebihan pada sel kanker. Gen cyclin D1 terletak pada kromosom 11q13 di mana pada kromosom ini genom biasanya mengalami amplifikasi pada karsinoma manusia, termasuk pada kanker (Giordano dan Normanno, 2009).

2.12 Senyawa SA2014

Senyawa SA2014 merupakan derivat dari kelompok *cinachyramine* dan termasuk ke dalam golongan alkaloid. Alkaloid merupakan senyawa yang memiliki unsur nitrogen dan biasanya berbentuk siklik. Senyawa ini diisolasi dari spons laut *Cinachyrella anomala*. Spons laut ini termasuk ke dalam kelas Demospongiae dan banyak terdapat di Pantai Kukup, Daerah Istimewa Yogyakarta. Isolasi pada spons ini dilakukan dengan cara memotong tubuh spons menjadi bagian yang lebih kecil (2-3 mm) serta dilakukan maserasi dengan etanol. Selanjutnya dilakukan isolasi dan elusidasi senyawa dengan menggunakan *thin layer chromatograph* (TLC). Hasil isolasi berbentuk kristal panjang dan memiliki titik leleh sebesar 121degC. Isolat senyawa yang telah didapatkan selanjutnya diidentifikasi dengan mengumpulkan data spektroskopik melalui uji spektrum FT-IR, 1H-NMR, spektra 13C-NMR, NMR dua dimensi. Berdasarkan uji tersebut diketahui senyawa tersebut memiliki formula $C_10H_13N_3O$ dengan nama struktur 1,4,9-triazatricyclo [7, 3, 1, 0] trideca-3, 5 (13), 10-trien-8-ol (SA2014) (Nurhayati, dkk., 2014). Struktur dari senyawa SA2014 dapat dilihat pada Gambar 2.8.



Gambar 2.8: Struktur Molekular SA2014

Dalam penelitian yang berjudul *Molecular Docking of Alkaloid Compound SA2014 from Marine Sponges Cinachyrella anomala Towards P53 Protein* (Nurhayati, dkk., 2017) telah dikaji tentang *docking* senyawa alkaloid SA2014 dan *Doxurubicin* terhadap protein P53 pada kanker payudara. Hasilnya menunjukkan bahwa senyawa SA2014 memiliki kemampuan sebagai senyawa antikanker terhadap kanker payudara T47D melalui interaksi asam amino leusin dan fenilalanin.

BAB 3
METODE PENELITIAN

Pada bab ini diuraikan tentang tahapan sistematis dalam penyusunan Tesis. Di samping itu, dijelaskan pula prosedur dan proses pelaksanaan tiap tahap yang dilakukan dalam menyelesaikan Tesis.

3.1 Tahapan Penelitian

Beberapa tahapan penelitian yang dilakukan dalam penelitian ini adalah sebagai berikut:

1. Studi literatur
   Studi literatur merupakan tahap untuk mengidentifikasi permasalahan dan mencari referensi yang menunjang penelitian. Referensi yang dipakai dapat berupa buku-buku literatur, tugas akhir, atau tesis yang berkaitan dengan permasalahan, maupun artikel dari internet. Tahap ini diperlukan sebagai landasan dalam melakukan penelitian.

2. Pengumpulan data
   Data yang digunakan dalam penelitian ini berupa data kompleks protein-ligan dalam format .pdb, data struktur protein Cyclin D1, data struktur senyawa alkaloid SA2014, dan data struktur doxorubicin. Data kompleks protein-ligan yang digunakan sebanyak 2 buah dengan masing-masing kompleks terdiri dari protein dan ligan yang telah ter-*docking*. Data kompleks ini merupakan hasil dari kristalografi sinar-X yang dapat diunduh pada RCSB Protein Data Bank melalui situs web https://www.rcsb.org. Data kompleks ini digunakan untuk mengevaluasi metode *docking*. Data struktur protein cyclin D1 dengan kode 2W96 diambil melalui RCSB Protein Data Bank, sedangkan data struktur doxorubicin diambil melalui *drug bank*. Struktur senyawa alkaloid SA2014 dapat diperoleh dari jurnal. Protein target yang digunakan adalah cyclin D1 di-*docking*-kan dengan ligan (target obat) yaitu senyawa alkaloid SA2014 dan doxorubicin.

3. Persiapan data
   Persiapan data kompleks protein-ligan dilakukan dengan menggunakan *software* Chimera untuk memisahkan protein dan ligan yang telah ter-*docking*. Hasil dari Chimera ini diperoleh dua file untuk setiap kompleks, yang masing-masing berisi struktur tiga dimensi dari ligan dan protein. Protein dan ligan tersebut kemudian di-*redocking* untuk mengevaluasi algoritma yang digunakan. Data struktur senyawa alkaloid SA2014 yang diperoleh dari jurnal digambar dengan menggunakan MarvinSketch. Kemudian protonasi senyawa dicek pada pH = 7.4.

4. Pembuatan *grid map*
   Pembuatan *grid map* dilakukan dengan menggunakan *software* AutoDock. *Grid map* yang dibuat memiliki ukuran $40 \times 40 \times 40$ dengan jarak 0.375Å Dari *grid map* ini diperoleh data berupa koordinat, jenis, dan muatan dari masing-masing atom pada ligan. Kemudian diperoleh juga nilai desolvasi, elektrostatik, dan energi ikatan van der waals serta ikatan hidrogen dari masing-masing jenis atom untuk setiap titik pada *grid map*.

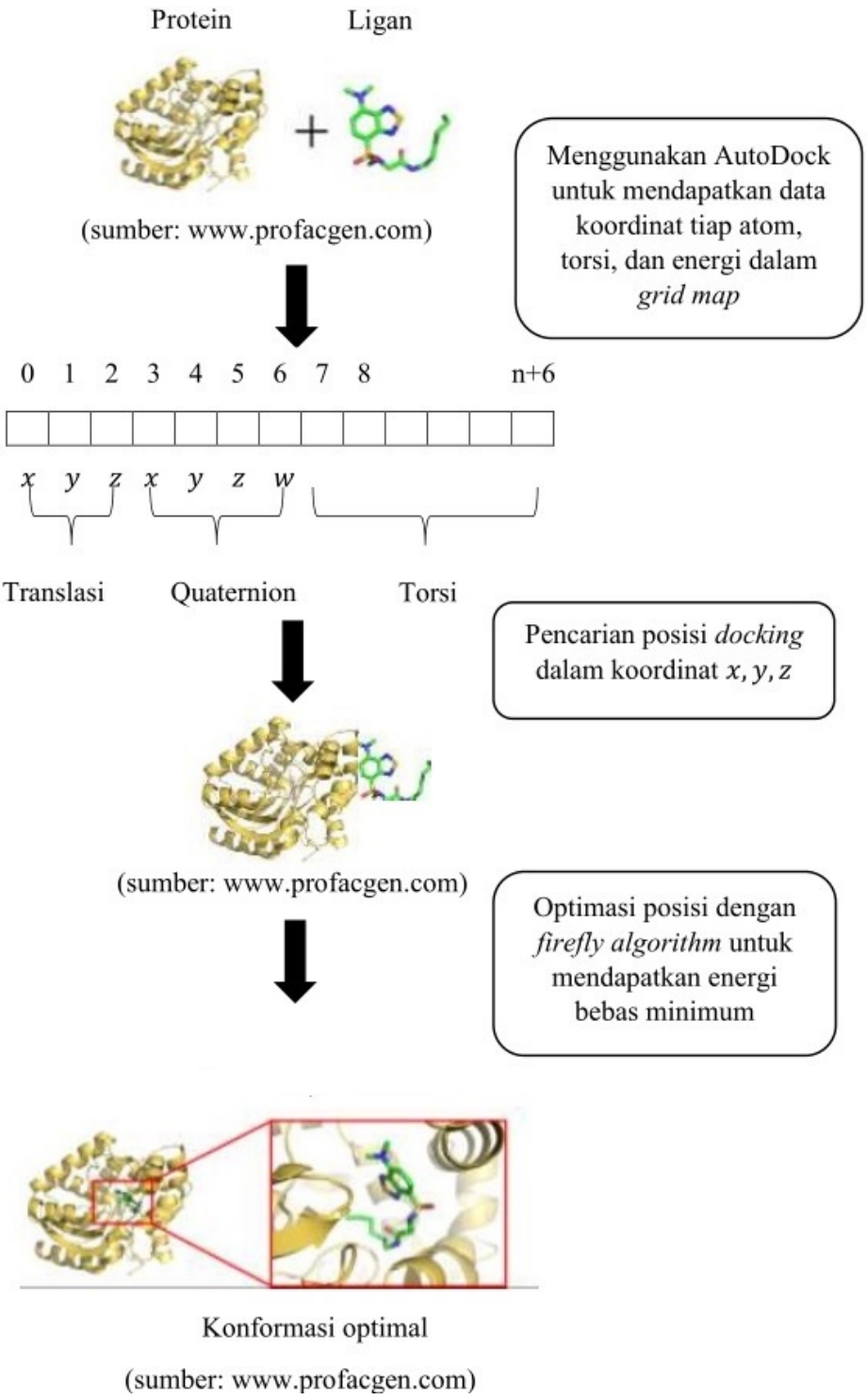
5. Pembuatan *database*
   Database dibuat dengan menggunakan MySQL. *Database* ini berguna untuk menyimpan data yang diperoleh dari *grid map* yang telah dibuat pada tahap sebelumnya.

6. Perancangan *firefly algorithm* untuk menyelesaikan *molecular docking*
   Pada tahap perancangan algoritma ini dirumuskan representasi solusi, cara membentuk solusi awal, dan cara membentuk solusi baru yang berdasarkan pada *firefly algorithm* untuk menyelesaikan *molecular docking*, serta penentuan beberapa parameter yang dibutuhkan untuk perhitungan.

7. Implementasi
   Firefly algorithm yang telah dirumuskan untuk *molecular docking* diimplementasikan dalam bahasa pemrograman Java dalam NetBeans IDE 8.0. *Source code* dari implementasi ini ditampilkan di Lampiran. Pada tahap ini, dari calon solusi yang diperoleh, dihitung koordinat baru untuk masing-masing atom pada ligan. Dari koordinat baru ini kemudian dihitung fungsi objektifnya yaitu meminimalkan energi ikatan dari kompleks yang dihasilkan. Nilai dari fungsi objektif ini diperoleh menggunakan interpolasi trilinier dari data energi ikatan van der waals dan ikatan hidrogen setiap jenis atom, desolvasi, dan elektrostatik yang telah disimpan di *database*. Selanjutnnya dilakukan proses optimasi dengan *firefly algorithm*. Proses tersebut akan dilakukan terus menerus hingga batas iterasi atau syarat berhenti terpenuhi. Skema proses *molecular docking* ini dapat dilihat pada Gambar 3.1

8. Evaluasi Hasil *Docking*
   Setelah algoritma selesai diimplementasikan, maka tahap selanjutnya adalah melakukan evaluasi. Dalam tahap ini, dilakukan *redocking* menggunakan *firefly algorithm* pada struktur tiga dimensi protein dan ligan dari data kompleks yang telah dipisahkan dengan *software* Chimera. Evaluasi dilakukan dengan melakukan perhitungan nilai RMSD (*Root Mean Square Deviation*). RMSD merupakan pengukuran dua pose dengan membandingkan konformasi hasil *docking* dan konformasi hasil kristalografi (dari file PDB). Nilai RMSD < 2Å akan digunakan sebagai kriteria kesuksesan metode *docking*. Semakin kecil nilai RMSD menunjukkan posisi/ikatan ligan yang diprediksi semakin

baik karena semakin mendekati konformasi yang sebenarnya (Thomsen, 2003).



Gambar 3.1: Skema Proses *Molecular Docking*

Formula untuk menghitung RMSD dari dua buah struktur adalah sebagai berikut (Carugo dan Pongor, 2001):

$$rmsd = \sqrt{\frac{\sum_i d_i^2}{n}} \tag{3.1}$$

dengan
n : jumlah atom
d_i : jarak antara dua atom i berdasarkan dua struktur

9. *Molecular docking* senyawa uji
   Apabila nilai RMSD yang diperoleh pada tahap evaluasi kurang dari 2Å maka dapat digunakan sebagai protokol *docking* untuk sampel berikutnya. Dalam penelitian ini sampel yang digunakan yaitu protein cyclin D1 dengan ligan senyawa alkaloid SA2014 dan doxorubicin sebagai pembanding. *Molecular docking* menghasilkan *output* berupa skor yang menggambarkan energi total ikatan protein-ligan. Dalam penelitian ini akan dibandingkan skor *docking* dari senyawa alkaloid SA2014 dan doxorubicin untuk mengetahui senyawa yang lebih poten di antara keduanya.

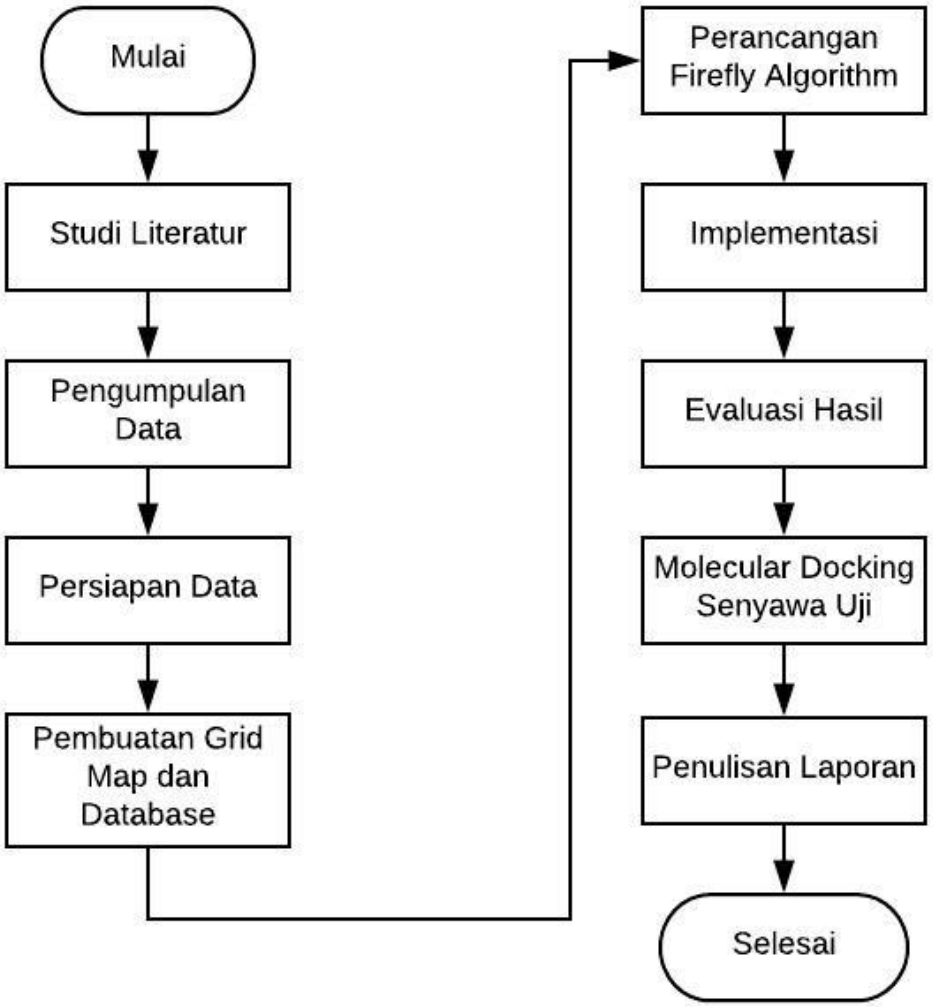
10. Publikasi penelitian
    Pada tahapan ini, dilakukan publikasi penelitian dengan berpartisipasi pada seminar internasional. Kegiatan ini bertujuan untuk menunjukkan hasil penelitian yang telah dilakukan yang dapat dijadikan sebagai bahan kajian untuk melakukan penelitian selanjutnya.

11. Penarikan kesimpulan
    Penarikan kesimpulan dilakukan dengan memperhatikan hasil dan pembahasan yang telah diselesaikan pada tahap-tahap sebelumnya. Kesimpulan yang ditarik adalah mengenai apakah *firefly algorithm* dapat digunakan untuk menyelesaikan *molecular docking* dengan baik dan bagaimana hasil *docking* dari protein Cyclin D1 dan senyawa alkaloid SA2014 yang diperoleh dengan metode tersebut.

12. Penyusunan Laporan Penelitian
    Pada tahap ini, dilakukan penyusunan laporan yang sistematis. Hal ini dilakukan agar pembaca dapat dengan mudah memahami penelitian yang dilakukan.

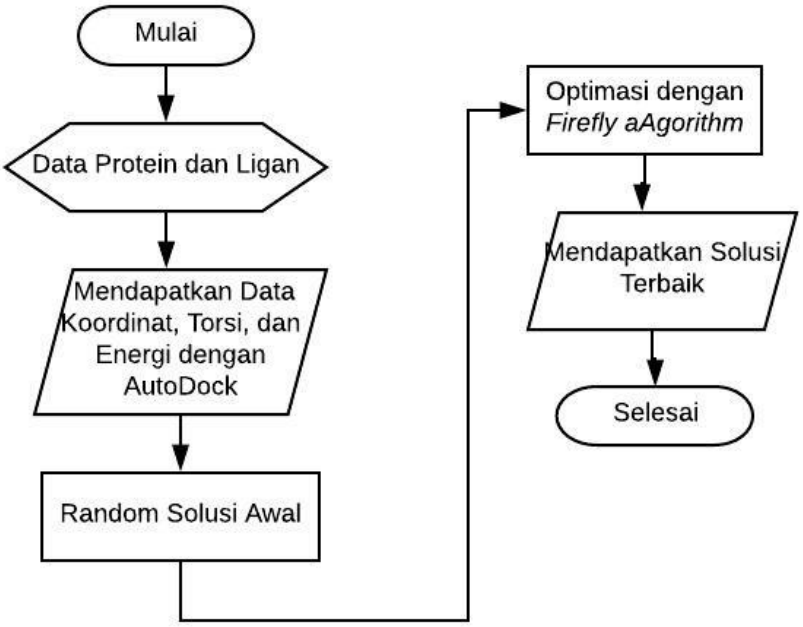
3.2 Diagram Alir Penelitian

Berdasarkan uraian di atas, penelitian Tesis ini dapat dinyatakan dalam diagram alir sebagai berikut.



Gambar 3.2: Diagram Alir Penelitian

Sedangkan diagram alir untuk proses *molecular docking* dapat dilihat pada gambar berikut.



Gambar 3.3: Diagram Alir Proses *Molecular Docking*

BAB 4
ANALISIS DAN PEMBAHASAN

Pada bab ini berisi tentang preparasi data, perancangan algoritma,validasi algoritma, dan analisis hasil yang diperoleh dengan algoritma tersebut.

4.1 Langkah-Langkah *Molecular Docking*

Sebelum melakukan *molecular docking*, dilakukan pengumpulan data telebih dahulu. Data yang dikumpulkan berasal dari RCSB Protein Data Bank. Setelah data terkumpul, kemudian dilakukan persiapan data. Lalu dilakukan *molecular docking* dengan algoritma optimasi yang digunakan yaitu *firefly algorithm.* Langkah-langkah dalam menyelesaikan *molecular docking* secara garis besar dapat dirangkum sebagai berikut:

1. Mengunduh data kompleks protein-ligan dari Protein Data Bank.
2. Pemrosesan data dengan Chimera untuk memisahkan struktur tiga dimensi ligan dan protein.
3. Pemrosesan dengan AutoDock untuk membuat *grid map* tiga dimensi.
4. Membuat *database* untuk menyimpan data koordinat dan energi.
5. Menentukan nilai parameter *firefly algorithm.*
6. Melakukan optimasi menggunakan *firefly algorithm.*
7. Visualisasi hasil *docking* yang diperoleh menggunakan Chimera.

4.2 Persiapan Data

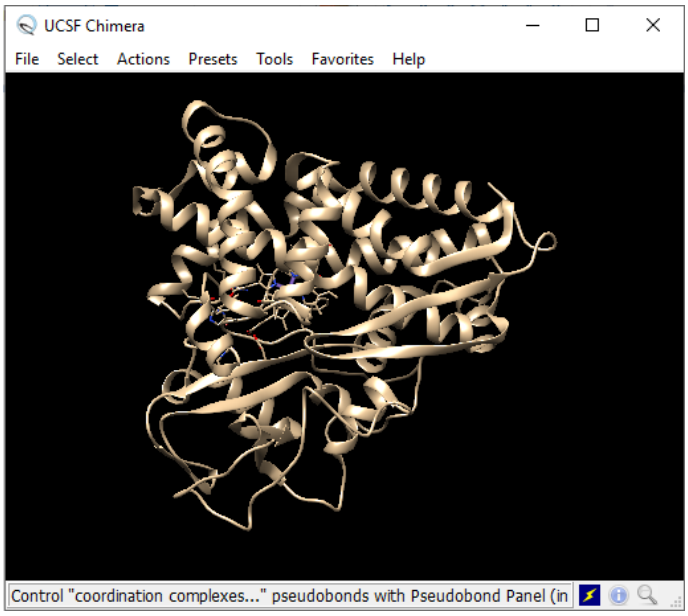
Tahap ini bertujuan untuk mempersiapkan data agar data awal yang telah diperoleh dapat diproses oleh algoritma. Data yang digunakan dibagi menjadi dua, yaitu data evaluasi dan data uji. Data evaluasi terdiri dari 2 data kompleks protein-ligan dalam format .pdb. Data evaluasi ini diunduh dari RCSB Protein Data Bank dengan kode 2cpp dan 3ptb. Data kompleks ini merupakan struktur tiga dimensi dari protein dan ligan yang telah ter-*docking* yang diperoleh dengan kristalografi sinar-X dari hasil eksperimen. Sedangkan untuk data uji terdiri dari data struktur protein target (cyclin D1), struktur senyawa alkaloid SA2014, dan data struktur doxorubicin. Pada penelitian ini protein cyclin D1 didapatkan dari RCSB Protein Data Bank dengan kode 2W96, data struktur doxorubicin diambil melalui *drug bank*, sedangkan struktur senyawa alkaloid SA2014 diperoleh dari jurnal. Rincian dari data yang digunakan dalam penelitian ini dapat dilihat pada Tabel 4.1.

Tabel 4.1: Rincian Data

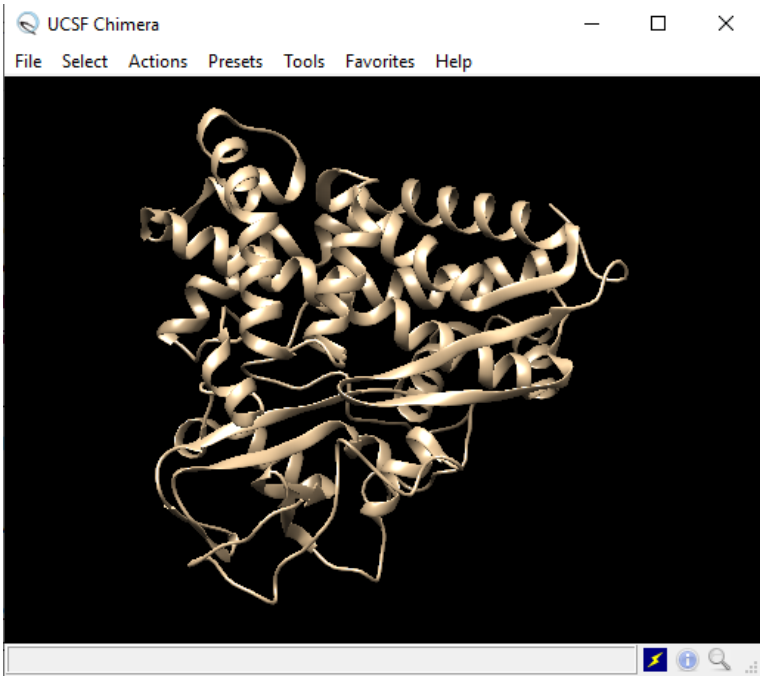
Nama Ligan	Jumlah Atom	Jumlah Torsi
2cpp	11	0
3ptb	9	1
SA2014	14	0
Doxorubicin	39	3

Tahap-tahap persiapan data dalam penelitian ini adalah sebagai berikut.

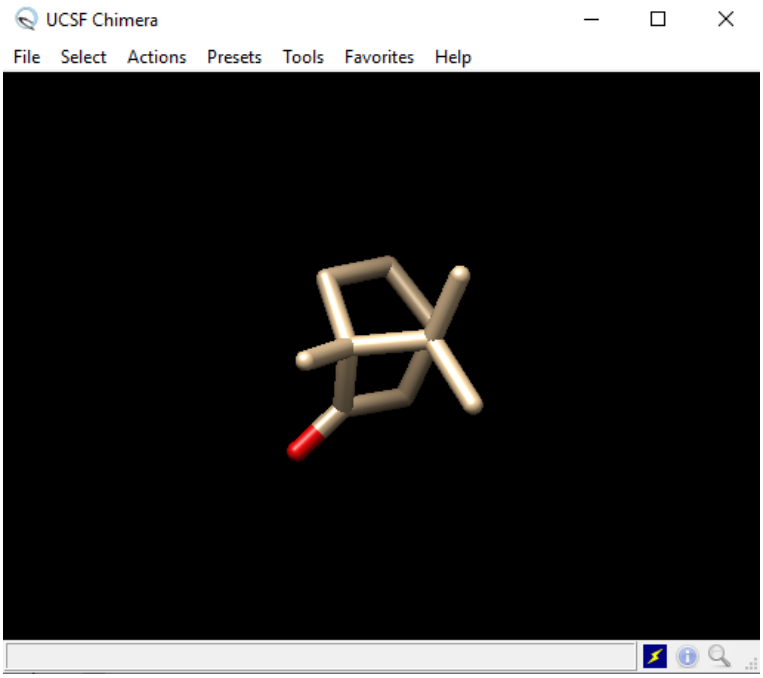
1. Pemisahan protein dan ligan pada data evaluasi
   Data evaluasi yang merupakan data kompleks yang terdiri dari protein dan ligan diproses menggunakan bantuan *software* Chimera untuk memisahkan antara protein dan ligannya. Contoh struktur tiga dimensi dari data kompleks dapat dilihat pada Gambar 4.1. Dari gambar di atas terlihat bahwa terdapat ligan yang ter-*docking* pada protein. Dengan menggunakan bantuan Chimera, protein dan ligan tersebut dipisahkan sehingga didapatkan dua struktur tiga dimensi yang terdiri dari protein saja dan ligan saja, seperti pada Gambar 4.2 dan Gambar 4.3. Protein dan ligan yang telah dipisahkan ini kemudian di-*redocking* dengan menggunakan *firefly algorithm*. Hasil dari *redocking* ini menentukan kesuksesan dari metode yang digunakan. Jika hasil yang diperoleh memiliki RMSD < 2Å maka dapat digunakan untuk melakukan *docking* pada data uji.



Gambar 4.1: Struktur Tiga Dimensi Kompleks '2cpp'



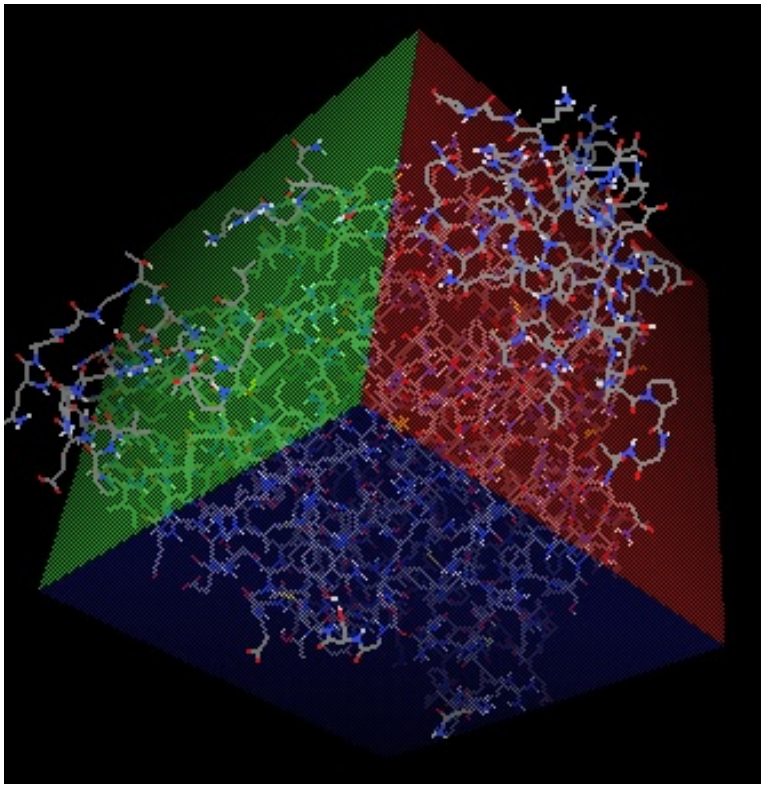
Gambar 4.2: Struktur Tiga Dimensi Protein



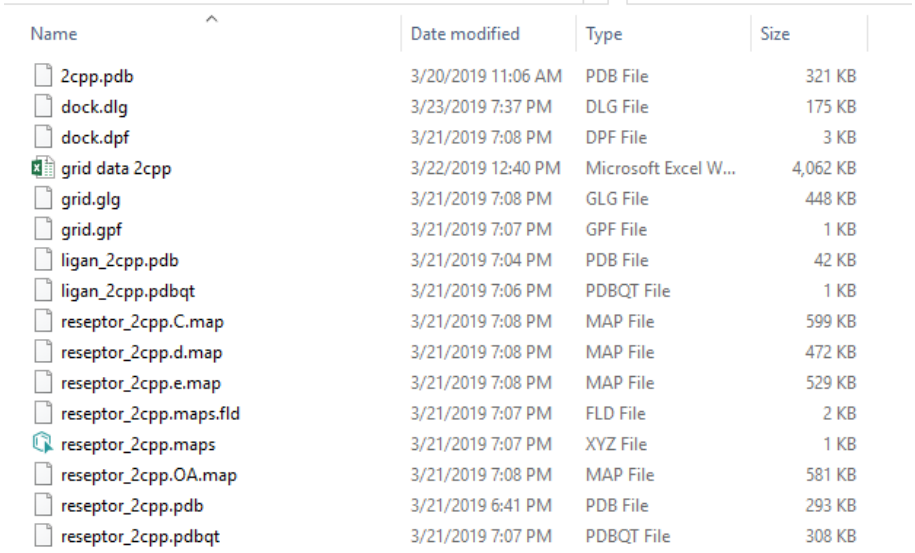
Gambar 4.3: Struktur Tiga Dimensi Ligan

2. Pembuatan *grid map* data evaluasi
   Data protein dan ligan ini kemudian diproses dengan Autodock untuk

membuat *grid map* tiga dimensi. *Grid map* yang dibuat dalam penelitian ini memiliki ukuran $40 \times 40 \times 40$ dengan jarak 0.375Å. Contoh dari *grid map* dapat dilihat pada Gambar 4.4. Dari *grid map* ini bisa didapatkan data koordinat masing-masing atom, jumlah torsi pada ligan dan posisi masing-masing torsi, serta data energi ikatan van der waals dan ikatan hidrogen untuk masing-masing jenis atom, desolvasi, dan elektrostatik pada setiap titik dalam *grid map* yang tersimpan dalam file-file seperti pada Gambar 4.5.



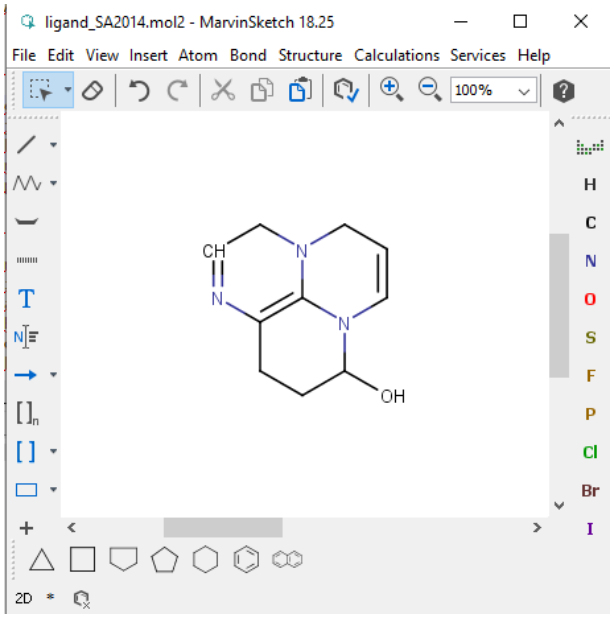
Gambar 4.4: Contoh *Grid Map* pada Autodock



Name	Date modified	Type	Size
2cpp.pdb	3/20/2019 11:06 AM	PDB File	321 KB
dock.dlg	3/23/2019 7:37 PM	DLG File	175 KB
dock.dpf	3/21/2019 7:08 PM	DPF File	3 KB
grid data 2cpp	3/22/2019 12:40 PM	Microsoft Excel W...	4,062 KB
grid.glg	3/21/2019 7:08 PM	GLG File	448 KB
grid.gpf	3/21/2019 7:07 PM	GPF File	1 KB
ligan_2cpp.pdb	3/21/2019 7:04 PM	PDB File	42 KB
ligan_2cpp.pdbqt	3/21/2019 7:06 PM	PDBQT File	1 KB
reseptor_2cpp.C.map	3/21/2019 7:08 PM	MAP File	599 KB
reseptor_2cpp.d.map	3/21/2019 7:08 PM	MAP File	472 KB
reseptor_2cpp.e.map	3/21/2019 7:08 PM	MAP File	529 KB
reseptor_2cpp.maps.fld	3/21/2019 7:07 PM	FLD File	2 KB
reseptor_2cpp.maps	3/21/2019 7:07 PM	XYZ File	1 KB
reseptor_2cpp.OA.map	3/21/2019 7:08 PM	MAP File	581 KB
reseptor_2cpp.pdb	3/21/2019 6:41 PM	PDB File	293 KB
reseptor_2cpp.pdbqt	3/21/2019 7:07 PM	PDBQT File	308 KB

Gambar 4.5: Contoh *File* yang Dihasilkan dari Pembuatan *Grid Map* pada Autodock

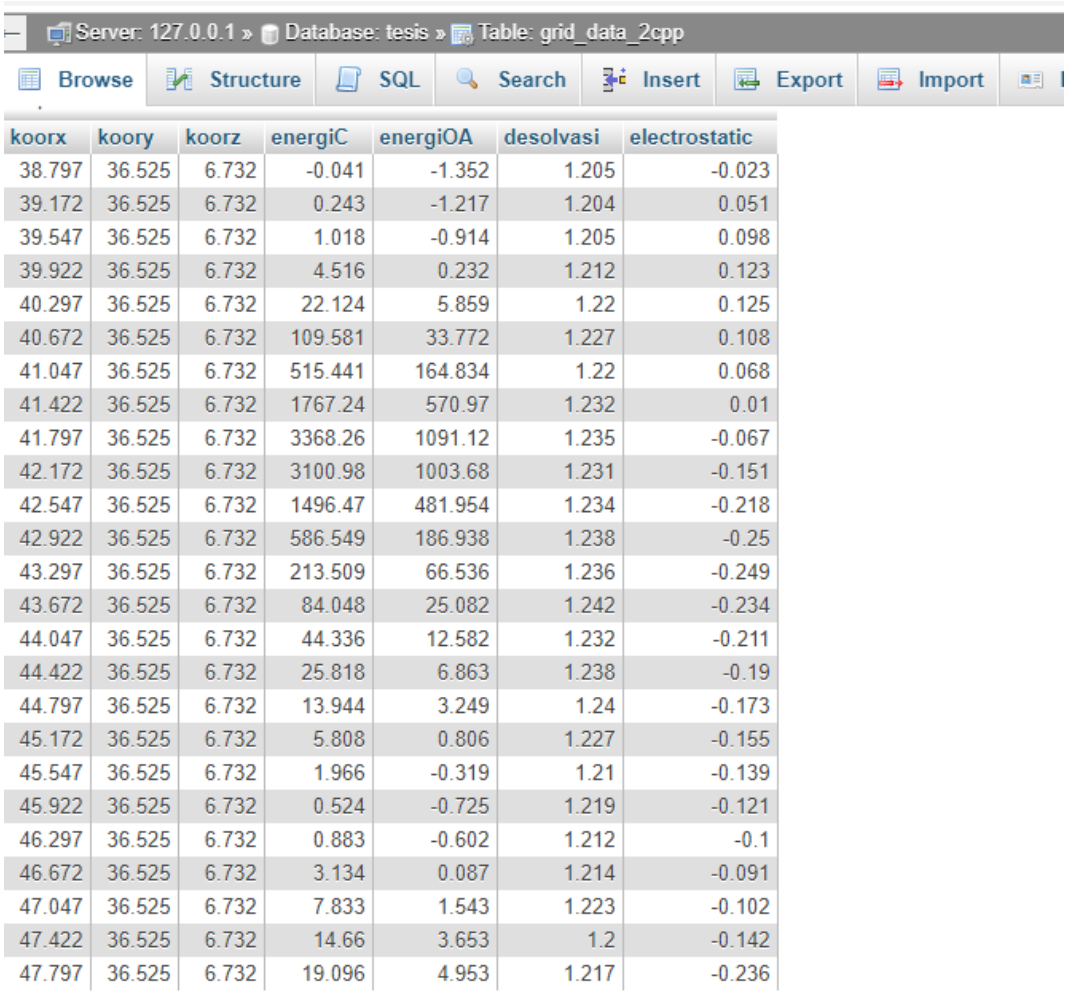
3. Pembuatan Ligan SA2014 dengan MarvinSketch
   Struktur dari senyawa alkaloid SA2014 yang diperoleh dari jurnal kemudian digambar dengan menggunakan *software* MarvinSketch dan disimpan dengan format .mol2 seperti pada Gambar 4.6.



Gambar 4.6: Ligan SA2014 yang Digambar dengan MarvinSketch

4. Pembuatan *grid map* data uji
   Untuk data uji, digunakan senyawa antikanker bahan alam golongan alkaloid yakni SA2014 untuk mengetahui interaksinya dengan protein cyclin D1. Senyawa ini telah diteliti aktivitasnya sebagai antikanker secara *in vitro*. Pada penelitian ini digunakan pula kontrol positif berupa obat komersil yaitu doxorubicin. Data uji yang diperoleh ini sudah terpisah antara protein dan ligannya. Dari data uji ini dibuat *grid map* untuk mendapatkan data koordinat atom, torsi, dan energi.

5. Pembuatan *database*
   Data-data yang diperoleh dari *grid map* ini kemudian disimpan dalam *database* MySQL. Tampilan dari salah satu tabel pada database ini dapat dilihat pada Gambar 4.7.



koorx	koory	koorz	energiC	energiOA	desolvasi	electrostatic
38.797	36.525	6.732	-0.041	-1.352	1.205	-0.023
39.172	36.525	6.732	0.243	-1.217	1.204	0.051
39.547	36.525	6.732	1.018	-0.914	1.205	0.098
39.922	36.525	6.732	4.516	0.232	1.212	0.123
40.297	36.525	6.732	22.124	5.859	1.22	0.125
40.672	36.525	6.732	109.581	33.772	1.227	0.108
41.047	36.525	6.732	515.441	164.834	1.22	0.068
41.422	36.525	6.732	1767.24	570.97	1.232	0.01
41.797	36.525	6.732	3368.26	1091.12	1.235	-0.067
42.172	36.525	6.732	3100.98	1003.68	1.231	-0.151
42.547	36.525	6.732	1496.47	481.954	1.234	-0.218
42.922	36.525	6.732	586.549	186.938	1.238	-0.25
43.297	36.525	6.732	213.509	66.536	1.236	-0.249
43.672	36.525	6.732	84.048	25.082	1.242	-0.234
44.047	36.525	6.732	44.336	12.582	1.232	-0.211
44.422	36.525	6.732	25.818	6.863	1.238	-0.19
44.797	36.525	6.732	13.944	3.249	1.24	-0.173
45.172	36.525	6.732	5.808	0.806	1.227	-0.155
45.547	36.525	6.732	1.966	-0.319	1.21	-0.139
45.922	36.525	6.732	0.524	-0.725	1.219	-0.121
46.297	36.525	6.732	0.883	-0.602	1.212	-0.1
46.672	36.525	6.732	3.134	0.087	1.214	-0.091
47.047	36.525	6.732	7.833	1.543	1.223	-0.102
47.422	36.525	6.732	14.66	3.653	1.2	-0.142
47.797	36.525	6.732	19.096	4.953	1.217	-0.236

Gambar 4.7: Tampilan Salah Satu Tabel pada *Database*

4.3 Perancangan *Firefly Algorithm* untuk Menyelesaikan *Molecular Docking*

Prosedur *firefly algorithm* dimulai dari inisialisasi parameter yang digunakan dalam algoritma. Kemudian membangkitkan populasi awal *firefly* $x_i(i = 1, 2, 3, ..., n)$ berdasarkan data yang telah disimpan di dalam *database*. Dari populasi *firefly* ini kemudian dihitung koordinat baru untuk masing-masing atom pada ligan. Selanjutnya menghitung fungsi objektif yaitu jumlah energi ikatan dari kompleks protein dan ligan dengan menggunakan interpolasi trilinier berdasarkan data *grid map* pada *database*. Dari fungsi objektif ini dapat diperoleh intensitas cahaya yang digunakan untuk membandingkan antar *firefly*. *Firefly* yang memiliki intensitas rendah akan mendekati *firefly* yang mempunyai intensitas lebih besar dengan cara *movement*. Setelah semua *firefly* dibandingkan, akan dicari *firefly* terbaik di setiap iterasi yang akan dibandingkan intensitasnya dengan intensitas cahaya *g-best* pada iterasi sebelumnya. Jika intensitasnya lebih besar dari intensitas *g-best*, maka *firefly* tersebut akan menjadi *g-best*. Kemudian *firefly* terbaik akan melakukan perpindahan secara acak. Proses ini akan berulang hingga batas iterasi atau syarat berhenti terpenuhi.

Perancangan *Firefly Algorithm* pada masing-masing tahap selengkapnya akan dijelaskan pada sub bab berikut ini.

4.3.1 Inisialisasi Parameter

Langkah pertama dalam *firefly algorithm* adalah inisialisasi parameter. Parameter yang digunakan dalam *firefly algorithm* yaitu banyaknya *firefly* (m), koefisien parameter random α, keatraktifan pada saat awal β_0, koefisien penyerapan cahaya pada medium γ, dan maksimum iterasi yang diinginkan. Pada sebagian besar implementasi *firefly algorithm* menggunakan $\beta_0 = 1, \alpha \in [0, 1]$ dan $\gamma \in [0, \infty)$.

4.3.2 Membangkitkan Populasi Awal Firefly

Fungsi objektif dalam permasalahan *molecular docking* ditentukan berdasarkan tiga komponen yang mewakili derajat kebebasan: (1) translasi dari molekul ligan yang melibatkan tiga sumbu (x, y, z) dalam koordiant Kartesius; (2) orientasi ligan yang dimodelkan sebagai empat variabel quaternion termasuk kemiringan sudut (w); dan (3) fleksibilitas, diwakili oleh rotasi bebas dari torsi (sudut dihedral) dari ligan. Dengan demikian, solusi dari permasalahan *molelcular docking* merupakan suatu vektor yang terdiri dari $n + 7$ variabel. Tiga nilai pertama mewakili translasi ligan, empat nilai berikutnya mewakili orientasi ligan, dan n nilai yang tersisa merupakan sudut torsi ligan.

Pembangkitan populasi awal *firefly* dilakukan sebanyak m kali. Setiap *firefly* mempunyai banyak elemen sesuai dengan banyaknya variabel dalam vektor solusi, yaitu $n + 7$ dengan n merupakan banyaknya torsi yang terdapat pada ligan. Nilai untuk koordinat translasi berada di antara nilai minimum dan maksimum dari masing-masing sumbu koordinat pada *grid map* yang telah dibuat dengan AutoDock. Parameter quaternion berada pada $[-1, 1]^3 \times$

$[-\pi, \pi]$, dan nilai untuk masing-masing sudut torsi adalah $[-\pi, \pi]$. Untuk variabel x, y, z dari translasi dan quaternion, digunakan satuan Å(angstrom) atau 10^{-10} meter. Sedangkan untuk w dan sudut torsi ligan, digunakan radian.

4.3.3 Menghitung Fungsi Objektif dan Intensitas Cahaya *Firefly*

Fungsi energi untuk permasalahan *molecular docking* umumnya menggunakan Persamaan (2.4), Persamaan (2.5), dan Persamaan (2.6). Namun karena ada beberapa variabel yang tidak diketahui nilainya dan untuk mengurangi biaya komputasi, maka digunakan metode berbasis *grid* untuk menghitung fungsi objektifnya. Pada metode ini, situs aktif protein ditanamkan dalam *grid map* 3D dan pada setiap titik di *grid map* telah dihitung dan disimpan energi interaksi elektrostatik, desolvasi, ikatan hidrogen dan van der waals berdasarkan seluruh atom protein untuk setiap tipe atom ligan. Dengan demikian, nilai energi untuk semua titik yang berada di dalam *grid map* dapat dihitung menggunakan intepolasi trilinier (Camacho, dkk., 2015).

Dalam *database* telah disimpan data koordinat masing-masing atom dalam ligan dan data energi untuk setiap titik dalam *grid map*. Data-data inilah yang digunakan untuk menghitung fungsi objektif. Data koordinat atom dalam *database* akan menjadi koordinat awal. Kemudian dengan setiap *firefly* akan dilakukan transformasi yaitu translasi dan rotasi sehingga didapatkan koordinat baru untuk setiap atom ligan. Dari koordinat baru ini kemudian dihitung energi untuk setiap atom ligan menggunakan interpolasi trilinier dengan data energi *grid map*. Dengan demikian bisa didapatkan fungsi objektif dengan menjumlahkan energi untuk semua atom ligan.

Fungsi objektif dalam *molecular docking* adalah meminimalkan jumlah energi ikatan dari kompleks protein-ligan yang terbentuk. Intensitas cahaya yang baik adalah nilai intensitas cahaya yang besar, sehingga nilai intensitas cahaya berbanding terbalik dengan nilai dari fungsi tujuan. Intensitas cahaya dalam kasus ini diperoleh dengan rumus $I(x) = -f(x)$. Intensitas cahaya tersebut merupakan indikator baik buruknya suatu *firefly*. Jika suatu *firefly* memiliki intensitas cahaya paling tinggi dibandingkan *firefly* lainnya, maka dapat dikatakan bahwa *firefly* tersebut adalah solusi terbaik.

4.3.4 Membandingkan *Firefly*

Pada tahap ini nilai intensitas cahaya pada *firefly* dibandingkan untuk memperoleh *firefly* terbaik. Jika nilai intensitas cahaya *firefly* p (I_p) kurang dari nilai intensitas cahaya *firefly* q (I_q) maka akan terjadi *movement firefly* p menuju *firefly* q yang memerlukan perhitungan jarak dan *attractiveness*. Setelah menghitung jarak dan *attractiveness*, selanjutnya adalah melakukan *movement* ke *firefly* q. Prosedur *movement* akan dilakukan pada *firefly* yang memiliki intensitas cahaya lebih kecil dibandingkan *firefly* lainnya sehingga diperoleh solusi baru untuk setiap *firefly*.

4.3.5 Menentukan *Firefly* Terbaik

Setelah membandingkan intensitas cahaya tiap *firefly*, langkah selanjutnya adalah menentukan *firefly* dengan intensitas cahaya tertinggi. *Firefly* dengan

intensitas tertinggi ini dianggap sebagai *firefly* terbaik.

4.3.6 Menentukan *Global Best* Sementara

Global best (g-best) adalah *firefly* yang mempunyai intensitas terbesar dari semua iterasi yang telah dilakukan. Pada iterasi pertama, *g-best* merupakan *firefly* terbaik. Sedangkan untuk iterasi selanjutnya, *firefly* terbaik pada iterasi tersebut akan dibandingkan dengan *g-best* pada iterasi sebelumnya. Apabila *firefly* terbaik memiliki intensitas cahaya lebih besar dibandingkan dengan *g-best* pada iterasi sebelumnya, maka *firefly* tersebut akan menjadi *g-best*. Sebaliknya, jika *firefly* terbaik memiliki intensitas cahaya lebih kecil dibandingkan dengan *g-best* pada iterasi sebelumnya, maka *g-best* pada iterasi tersebut sama dengan *g-best* pada iterasi sebelumnya.

4.3.7 Melakukan *Movement* pada *Firefly* Terbaik

Setelah mendapatkan *g-best*, langkah selanjutnya adalah melakukan *movement* pada *firefly* terbaik. *Firefly* terbaik pada setiap iterasi akan melakukan pergerakan secara random. Hal ini dilakukan untuk mencari solusi lain di sekitar *firefly* terbaik.

4.4 Contoh Perhitungan *Firefly Algorithm* untuk Menyelesaikan *Molecular Docking*

Berikut ini adalah contoh kasus *molecular docking* yang diselesaikan dengan *firefly algorithm*. Misal ligan yang akan di-*docking* terdiri dari tiga atom dengan satu buah torsi yang terletak di antara atom kedua dan atom ketiga. Atom kedua merupakan pusat dari ligan. Dari data protein dan ligan ini dibuat *grid map* berukuran $3 \times 3 \times 3$ dengan jarak 1Å. Data atom ligan dan data energi yang diperoleh dari pembuatan *grid map* disajikan dalam Tabel 4.2 dan Tabel 4.3 berikut.

Tabel 4.2: Data Atom Ligan

No. Atom	Koordinat x	Koordinat y	Koordinat z	Tipe Atom	Muatan
1	1.4	2	2.3	A	-0.41
2	1.5	1.5	1.5	C	-0.02
3	2.1	1	1.8	A	1.52

Tabel 4.3: Data Hasil Pembuatan *Grid Map*

Koordinat x	Koordinat y	Koordinat z	Energi Atom A	Energi Atom C	Desolvasi	Elektrostatik
1	1	1	-0.327	-0.225	0.005	-0.148
2	1	1	-0.125	-0.013	0.015	-0.165
3	1	1	-0.141	-0.415	0.034	-0.186
1	2	1	0.252	1.251	0.06	-0.209
2	2	1	28.969	20.045	0.1	-0.233
3	2	1	997.394	801.45	0.146	-0.256
1	3	1	73724.6	25565	0.195	-0.274
2	3	1	42.003	51.415	0.252	-0.286
3	3	1	-0.002	-0.215	0.306	-0.294
1	1	2	-0.021	-0.045	0.358	-0.303
2	1	2	-0.036	-0.132	0.409	-0.319
3	1	2	-0.105	-0.441	0.454	-0.338
1	2	2	1.407	2.451	0.505	-0.343
2	2	2	50.115	61.251	0.536	-0.307
3	2	2	-0.332	-0.113	0.606	-0.212
1	3	2	1.373	2.121	0.667	-0.058
2	3	2	-0.2	-0.315	0.758	0.153
3	3	2	-0.283	-0.2	0.841	0.433
1	1	3	-0.311	-0.25	0.952	0.771
2	1	3	11.477	21.151	1.102	1.141
3	1	3	3874.93	4151.68	1.209	1.512
1	2	3	105.167	225.465	1.3	2.245
2	2	3	-0.025	-0.2	1.361	2.814
3	2	3	-0.048	-0.358	1.383	2.468
1	3	3	-0.085	-0.125	1.392	0.32
2	3	3	2.413	3.441	0.005	-0.168
3	3	3	14.443	15.552	0.023	-0.192

Dalam Tabel 4.2 terdapat koordinat awal, tipe atom, dan muatan untuk masing-masing atom dalam ligan, sedangkan pada Tabel 4.3 terdapat koordinat setiap titik dalam *grid map*, energi untuk masing-masing jenis atom, desolvasi, dan elektrostatik. Karena atom-atom penyusun ligan terdiri dari dua jenis, yaitu A dan C, maka dalam data *grid map* terdapat nilai energi untuk atom A dan atom C. Nilai energi ini merupakan jumlah dari energi ikatan van der waals dan ikatan hidrogen untuk masing-masing jenis atom.

Langkah-langkah yang digunakan untuk menyelesaikan kasus *molecular docking* di atas dengan menggunakan *firefly algorithtm* adalah sebagai berikut.

1. Inisialisasi parameter
   Parameter-parameter *firefly algorithm* yang digunakan dalam contoh

kasus ini yaitu banyaknya *firefly* $(m) = 3$, MaxIterasi $= 5$, $\beta_0 = 1$, $\gamma = 1$, dan $\alpha = 0.1$.

2. Membangkitkan populasi awal *firefly*
   Membangkitkan populasi awal yang terdiri dari 3 *firefly* dengan masing-masing *firefly* terdiri dari 8 elemen. Hasil membangkitkan populasi awal *firefly* disajikan pada Tabel 4.4 berikut.

Tabel 4.4: Populasi Awal *Firefly*

Firefly	Elemen							
	1	2	3	4	5	6	7	8
x_1	1.7	2	1	0.8	-0.25	0.3	30	30
x_2	2	1.3	1.1	-0.7	0.35	0.81	60	45
x_3	1.5	1.4	1	0.31	-0.75	0.9	90	30

3. Menghitung koordinat atom ligan
   Setelah membangkitkan populasi awal *firefly*, langkah selanjutnya adalah melakukan transformasi pada atom ligan sehingga diperoleh koordinat baru untuk masing-masing atom. Proses perhitungan ini akan diuraikan sebagai berikut.
   Misalnya pada *firefly* 1 $x_1 = [1.7, 2, 1, 0.8, -0.25, 0.3, 30, 30]$.

   (a) Titik pusat ligan yang awalnya berada pada koordinat $(1.5, 1.5, 1.5)$ ditranslasi ke koordinat $(1.7, 2, 1)$. Dengan demikian diperoleh $\Delta x = 1.7 - 1.5 = 0.2, \Delta y = 2 - 1.5 = 0.5, \Delta z = 1 - 1.5 = -0.5$. Koordinat baru untuk atom ligan setelah translasi dapat dilihat pada Tabel 4.5

Tabel 4.5: Koordinat Atom Ligan Setelah Translasi

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	1.6	2.5	1.8
2	1.7	2	1
3	2.3	1.5	1.3

   (b) Setelah translasi, kemudian dilakukan rotasi dengan unit quaternion. Parameter quaternion $(0.8, -0.25, 0.3)$ harus dinormalisasi dulu agar menjadi vektor unit dan sudut quaternion 30° harus diubah ke dalam radian, sehingga diperoleh unit quaternion $q = 0.97 + 0.23i - 0.07j + 0.09k$. Kemudian dengan menggunakan matriks rotasi pada Persamaan (2.12) diperoleh koordinat baru sebagai berikut.

Tabel 4.6: Kordinat Atom Ligan Setelah Rotasi dengan Unit Quaternion

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	1.42	2.06	1.91
2	1.7	2	1
3	2.36	1.5	1.15

(c) Karena ligan memiliki satu buah sudut torsi yang berada di antara atom kedua dan ketiga, maka selanjutnya dilakukan rotasi terhadap sumbu x, y, dan z pada atom ketiga. Dengan menggunakan matriks rotasi pada Persamaan (2.9), (2.10), dan (2.11) diperoleh koordinat hasil rotasi sebagai berikut.

Tabel 4.7: Koordinat Atom Ligan Setelah Rotasi

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	1.42	2.06	1.91
2	1.7	2	1
3	1.85	1.5	1.65

Dengan cara yang sama, diperoleh koordinat atom baru untuk *firefly* 2 dan *firefly* 3 yang disajikan pada Tabel 4.8 dan Tabel 4.9.

Tabel 4.8: Koordinat Atom Baru untuk *Firefly* 2

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	1.61	2.04	1.54
2	2	1.3	1.1
3	2.52	1.16	1.67

Tabel 4.9: Koordinat Atom Baru untuk *Firefly* 3

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	0.7	0.96	1.26
2	1.5	1.4	1
3	1.94	1.53	1.57

4. Menghitung fungsi objektif dan intensitas cahaya *firefly*
Dari koordinat baru yang telah diperoleh, kemudian dihitung nilai energi ikatan van der waals dan ikatan hidrogen, desolvasi, dan elektrostatis dari setiap atom dalam ligan. Nilai-nilai ini diperoleh menggunakan

interpolasi trilinier dari data *grid map* yang telah dibuat. Misalnya untuk atom pertama pada *firefly* 1, yaitu dengan koordinat $(1.42, 2.06, 1.91)$. Karena 1.42 berada di antara 1 dan 2, 2.06 berada di antara 2 dan 3, serta 1.91 berada di antara 1 dan 2, maka pada data *grid map* dipilih delapan titik yaitu $P_{000} = (1, 2, 1), P_{100} = (2, 2, 1), P_{010} = (1, 3, 1), P_{110} = (2, 3, 1), P_{001} = (1, 2, 2), P_{101} = (2, 2, 2), P_{011} = (1, 3, 2)$, dan $P_{111} = (2, 3, 2)$. Dengan menggunakan Persamaan (2.13) diperoleh nilai energi atom, desolvasi, dan elektrostatik untuk atom pertama yaitu 36488.31, 0.23, dan -0.26. Karena atom pertama memiliki muatan sebesar -0.41, maka total energi untuk atom ini adalah $36488.31 + |-0.42|\,(1.23) + (-0.41)(-0.26) = 36488.92$. Dengan cara yang sama, dihitung nilai energi untuk atom kedua dan ketiga, sehingga didapatkan nilai fungsi objektif untuk *firefly* pertama yaitu dengan menjumlahkan nilai energi untuk semua atom. Nilai fungsi objektif untuk masing-masing *firefly* disajikan dalam Tabel 4.10 berikut ini.

Tabel 4.10: Nilai Fungsi Objektif Setiap *Firefly*

Firefly	x_1	x_2	x_3
$f(x_i)$	36528.12	15385.39	33.31

Karena fungsi objektif dari permasalahan *molecular docking* adalah diminimalkan dan intensitas cahaya *firefly* akan dicari yang paling maksimal, maka intensitas cahaya untuk masing-masing *firefly* dicari dengan rumus $I(x) = -f(x)$. Hasil perhitungan intensitas cahaya *firefly* selengkapnya disajikan dalam Tabel 4.11.

Tabel 4.11: Intensitas Cahaya *Firefly*

Firefly	x_1	x_2	x_3
$I(x_i)$	-36528.12	-15385.39	-33.31

Firefly terbaik adalah *firefly* dengan intensitas cahaya terbesar. Dengan demikian, *firefly* ke-3 merupakan *firefly* terbaik dengan intensitas cahaya -33.31.

5. Membandingkan *firefly*
   Pada langkah ini, masing-masing *firefly* akan dibandingkan dengan *firefly* lainnya berdasarkan intensitas cahayanya. Jika intensitas cahaya suatu *firefly* lebih kecil dari pada intensitas cahaya *firefly* lainnya, maka *firefly* tersebut akan melakukan perpindahan menuju *firefly* yang mempunyai intensitas lebih baik. Terdapat beberapa perhitungan saat melakukan

perpindahan, yaitu *attractiveness* β dan jarak antar *firefly*. Misalkan *firefly* 1 yang akan dibandingkan dengan *firefly* lainnya. Karena $I(x_1) < I(x_2)$ maka *firefly* 1 akan bergerak menuju *firefly* 2 dengan langkah sebagai berikut:

(a) Menghitung *distance* (jarak) antara *firefly* 1 dan *firefly* 2 dengan menggunanakan Persamaan (2.2).
$r_{12} = \sqrt{\sum_{k=1}^{8}(x_1^k - x_2^k)^2}$
$r_{12} = \sqrt{(1.7-2)^2 + (2-1.3)^2 + ... + (30-45)^2}$
$r_{12} = 33.6$

(b) Menghitung *attractiveness* dengan menggunakan Persamaan (2.1).
$\beta = 1 \times e^{-1 \times 33.6^2} = 0$

(c) Menghitung perpindahan *firefly* per elemen dengan menggunakan Persamaan (2.3).
$x_1^1 = x_1^0 + 0 \times (x_2^0 - x_1^0) + 0.1 \times 0.5$
Sehingga diperoleh *firefly* 1 yang telah diperbarui yaitu $x_1 = (1.75, 2.05, 1.05, 0.85, -0.2, 0.35, 30.05, 30.05)$.

Dengan cara yang sama, dilakukan perbandingan untuk *firefly* lainnya sehingga diperoleh populasi baru *firefly* setelah melakukan *movement* sebagai berikut:

Tabel 4.12: Populasi Baru *Firefly* Setelah Melakukan *Movement*

Firefly	Elemen							
	1	2	3	4	5	6	7	8
x_1	1.75	2.05	1.05	0.85	-0.2	0.53	3.05	30.05
x_2	2.01	1.31	1.11	-0.69	0.36	0.82	60.01	45.01
x_3	1.5	1.4	1	0.31	-0.75	0.9	90	30

Kemudian menghitung fungsi objektif baru untuk semua *firefly* dan menghitung intensitas cahayanya. Fungsi objektif dan intensitas cahaya dapat dihitung dengan cara yang sama seperti pada langkah 4. Intensitas cahaya dari *firefly* hasil *movement* disajikan pada Tabel 4.13 berikut ini.

Tabel 4.13: Intensitas Cahaya *Firefly* Setelah *Movement*

Firefly	x_1	x_2	x_3
$I(x_i)$	-32510.3	-15287.023	-33.31

6. Menentukan *firefly* terbaik
Firefly terbaik adalah *firefly* dengan intensitas cahaya tertinggi pada

setiap iterasi. Dari hasil *movement* diperoleh bahwa intensitas cahaya tertinggi berada pada *firefly* ke-3 dengan nilai intensitas cahayanya adalah -33.31.

7. Menentukan *global gest* sementara
   Pada iterasi pertama, *global best (g-best)* adalah *firefly* terbaik pada iterasi tersebut. Sedangkan untuk iterasi selanjutnya, *g-best* dapat dipilih berdasarkan intensitas cahaya yang paling besar antara *g-best* dengan *firefly* terbaik pada iterasi tersebut. Dari hasil *movement* pada iterasi pertama, *firefly* 3 memiliki intensitas cahaya tertinggi. Oleh sebab itu dipilih *firefly* 3 sebagai *g-best*.

8. Melakukan *movement* pada *firefly* terbaik
   Movement pada *firefly* terbaik dilakukan agar tidak terjebak pada optimum lokal dan untuk membentuk populasi *firefly* baru yang akan digunakan pada iterasi selanjutnya. *Movement* ini dilakukan dengan rumus sebagai berikut.
   $$x_i^{t+1} = x_i^t + +\alpha\epsilon_i^t$$
   Sehingga diperoleh x_3 yang baru yaitu $x_3 = (1.59, 1.49, 1.09, 0.4, -0.66, 0.99, 90.09, 30.09)$

Populasi baru ini akan digunakan sebagai populasi awal pada iterasi selanjutnya. Proses ini akan terus berulang sampai maksimum iterasi telah terpenuhi.

4.5 Evaluasi Algoritma

Hasil dari *molecular docking* dievaluasi berdasarkan nilai energi yang dihasilkan. Semakin kecil nilai energi yang diperoleh menandakan bahwa semakin stabil ikatan yang terjadi antara protein dan ligan. Evaluasi keberhasilan metode *docking* juga dapat dilakukan dengan mencari nilai *root mean square deviation* (RMSD) , yakni dengan membandingkan referensi ligan dengan hasil dari *docking* yang telah dilakukan. Hasil *molecular docking* dapat diterima apabila RMSD hasil *docking* yang dibandingkan dengan referensinya memiliki nilai kurang dari 2Å (Thomsen, 2003).

Dalam penelitian ini, *molecular docking* yang dilakukan adalah *semi-flexible docking*, yaitu protein yang digunakan bersifat *rigid* (kaku), sedangkan ligannya bersifat fleksibel. Fleksibilitas dari protein memiliki efek yang sangat besar terhadap keakuratan dari *molecular docking*. Pergerakan dari protein dapat menyebabkan penurunan yang drastis dalam hal akurasi *docking*. Perlu dilakukan penelitian lebih lanjut untuk mengeksplorasi efek fleksibilitas protein terhadap *molecular docking* (Erickson, dkk., 2004). Oleh karena itu, protein yang digunakan dalam penelitian ini bersifat kaku.

Untuk evaluasi algoritma, digunakan dua buah kompleks yaitu 2cpp dan 3ptb. Kompleks yang diperoleh dari Protein Data Bank ini di-*redocking* menggunakan *firefly algorithm*. Koordinat awal untuk atom ligan 2cpp dan 3ptb ini ditunjukkan pada Tabel 4.14 dan Tabel 4.15

Tabel 4.14: Kordinat Awal untuk Atom Ligan 2cpp

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	46.65	43.05	9.35
2	46.30	41.98	8.21
3	45.64	42.25	7.22
4	46.95	40.66	8.65
5	47.70	41.12	9.96
6	48.91	42.04	9.60
7	48.20	43.35	9.14
8	46.65	42.09	10.62
9	45.27	41.45	11.00
10	47.13	42.75	11.96
11	45.75	44.28	9.33

Tabel 4.15: Kordinat Awal untuk Atom Ligan 3ptb

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	0.77	17.78	18.96
2	-0.02	18.90	18.85
3	-1.37	18.77	18.52
4	-1.93	17.50	18.30
5	-1.12	16.37	18.43
6	0.22	16.52	18.76
7	1.59	18.13	19.81
8	1.57	18.13	20.64
9	2.71	18.50	19.60

Parameter *firefly algorithm* yang digunakan yaitu $\gamma = 1, \beta_0 = 1$, jumlah *firefly* sebanyak 10, nilai α bervariasi, dan jumlah iterasi sebanyak 1000, 1500, dan 2000. Nilai energi yang diperoleh dari masing-masing percobaan dapat dilihat pada Tabel 4.16 dan Tabel 4.17, sedangkan untuk nilai RMSD-nya dapat dilihat pada Tabel 4.18 dan Tabel 4.19.

Tabel 4.16: Nilai Energi untuk 2cpp

Nilai α	Energi		
	1000 iterasi	1500 iterasi	2000 iterasi
0.1	-2.93	-2.96	-3.04
0.3	-4.05	-4.32	-4.36
0.5	-4.26	-4.38	-4.48
0.7	-3.88	-3.92	-4.18
0.9	-4.25	-4.34	-4.46
1.0	-4.44	-4.47	-4.49

Tabel 4.17: Nilai Energi untuk 3ptb

Nilai α	Energi		
	1000 iterasi	1500 iterasi	2000 iterasi
0.1	-1.94	-2.32	-3.42
0.3	-2.72	-3.07	-3.74
0.5	-4.01	-4.03	-4.06
0.7	-3.17	-3.94	-4.05
0.9	-4.06	-4.10	-4.20
1.0	-4.24	-4.32	-4.37

Dari Tabel 4.16 dan Tabel 4.17 terlihat bahwa semakin banyak jumlah iterasi, maka semakin negatif nilai energi yang dihasilkan. Artinya semakin banyak jumlah iterasi, maka semakin bagus solusi yang dihasilkan. Nilai energi untuk kedua kompleks memiliki pola yang sama yaitu dari nilai $\alpha = 0.1$ menurun pada $\alpha = 0.3$ dan $\alpha = 0.5$. Kemudian mengalami kenaikan pada $\alpha = 0.7$ dan menurun kembali pada $\alpha = 0.9$ dan $\alpha = 1.0$. Solusi terbaik yaitu yang menghasilkan nilai energi terendah diperoleh dengan nilai $\alpha = 1.0$.

Tabel 4.18: Nilai RMSD untuk 2cpp

Nilai α	RMSD		
	1000 iterasi	1500 iterasi	2000 iterasi
0.1	3.33	5.34	5.06
0.3	0.95	6.58	7.52
0.5	0.68	0.88	7.48
0.7	5.61	7.19	6.63
0.9	0.74	0.88	0.55
1.0	0.98	0.65	0.59

Tabel 4.19: Nilai RMSD untuk 3ptb

Nilai α	RMSD		
	1000 iterasi	1500 iterasi	2000 iterasi
0.1	5.20	5.49	3.92
0.3	3.49	3.74	3.01
0.5	1.34	1.60	2.11
0.7	3.63	3.22	1.11
0.9	2.56	1.60	1.86
1.0	1.08	1.12	1.53

Pada Tabel 4.18 dan Tabel 4.19 dapat dilihat bahwa untuk kompleks 2cpp, nilai RMSD kurang dari 2Å diperoleh dengan nilai $\alpha = 0.9$ dan $\alpha = 1.0$. Sedangkan untuk kompleks 3ptb, nilai RMSD kurang dari 2Å diperoleh dengan $\alpha = 1.0$. Oleh karena itu, dapat disimpulkan bahwa solusi terbaik diperoleh dengan menggunakan nilai $\alpha = 1.0$.

Semakin kecil nilai RMSD yang didapatkan, maka hasil prediksi yang diperoleh semakin dekat dengan kejadian aslinya. Namun semakin kecil nilai energi yang diperoleh bukan berarti semakin kecil pula nilai RMSD-nya. Hal ini terjadi karena terkadang *firefly algorithm* dapat terjebak pada optimum lokal yang memiliki nilai energi yang rendah namun menghasilkan posisi ligan yang jauh dari ligan referensinya sehingga nilai RMSD yang dihasilkan cukup besar. Dengan menggunakan nilai $\alpha = 1.0$ bisa didapatkan nilai energi terendah dan nilai RMSD yang kurang dari 2Å.

Pada Gambar 4.8 dan Gambar 4.9 dapat dilihat perbandingan posisi ligan hasil *docking* secara simulasi dan referensinya. Ligan hasil simulasi ditandai dengan garis pinggir berwarna hijau, sedangkan ligan referensinya berwarna biru. Hasil *docking* ini memiliki RMSD sebesar 0.98Å untuk kompleks 2cpp dan sebesar 1.08Å untuk kompleks 3ptb. Dapat dilihat bahwa posisi ligan hasil *docking* sangat berdekatan dengan ligan referensinya. Hal ini menunjukkan bahwa algoritma yang digunakan dapat menyelesaikan *molecular docking* dengan baik. Sedangkan koordinat akhir untuk atom ligan 2cpp dan 3ptb yang diperoleh dari hasil simulasi dapat dilihat pada Tabel 4.20 dan Tabel 4.21.

Gambar 4.8: Perbandingan Hasil *Docking* 2cpp dengan Referensinya

Gambar 4.9: Perbandingan Hasil *Docking* 3ptb dengan Referensinya

Tabel 4.20: Kordinat Akhir untuk Atom Ligan 2cpp

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	46.84	44.53	14.95
2	45.29	44.81	14.69
3	44.42	44.58	15.51
4	45.21	45.39	13.27
5	46.73	45.48	12.88
6	47.44	46.60	13.71
7	47.48	45.97	15.15
8	47.30	44.14	13.47
9	46.69	42.82	12.89
10	48.84	43.94	13.27
11	47.10	43.56	16.10

Tabel 4.21: Kordinat Akhir untuk Atom Ligan 3ptb

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	-1.79	13.56	16.59
2	-2.39	14.78	16.43
3	-1.98	15.87	17.22
4	-0.95	15.71	18.16
5	-0.34	14.46	18.30
6	0.77	13.40	17.51
7	-1.26	13.12	15.83
8	-1.56	13.71	15.54
9	-0.29	12.22	15.45

4.6 Hasil Simulasi *Docking* Ligan dan Protein Target

Pada penelitian ini dilakukan pengujian dengan senyawa antikanker bahan alam golongan alkaloid yakni SA2014 untuk mengetahui interaksinya terhadap protein Cyclin D1. Senyawa SA2014 ini telah diteliti aktivitasnya sebagai antikanker secara *in vitro*. Pada penelitian ini digunakan pula kontrol positif berupa obat komersil yaitu doxorubicin. Suatu prosedur *molecular docking* digunakan sebagai acuan untuk menentukan orientasi terbaik dari suatu senyawa terhadap senyawa lainnya. Uji simulasi *molecular docking* menghasilkan suatu konformasi nilai/skor pada senyawa SA2014 maupun senyawa doxorubicin. Pada *molecular docking* hasil yang didapatkan sangat bergantung pada sruktur protein dan ligan. Koordinat awal untuk atom ligan SA014 dan doxorubicin dapat dilihat pada Tabel 4.22 dan Tabel 4.23.

Tabel 4.22: Kordinat Awal untuk Atom Ligan SA2014

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	0	-11.75	3.54
2	0	-13.08	2.77
3	0	-13.08	1.23
4	0	-11.75	0.46
5	0	-10.42	1.23
6	0	-10.42	2.77
7	0	-9.08	0.46
8	0	-7.75	1.23
9	0	-7.75	2.77
10	0	-9.08	3.54
11	0	-11.75	-1.08
12	0	-10.42	-1.85
13	0	-9.08	-1.08
14	0	-7.75	-1.85

Tabel 4.23: Kordinat Awal untuk Atom Ligan Doxorubicin

No. Atom	Koordinat x	Koordinat y	Koordinat z
1	0	7.03	-4.50
2	0	11.90	0.72
3	0	11.90	-5.44
4	0	5.48	-1.83
5	0	3.95	-4.51
6	0	14.60	0.78
7	0	14.60	-5.49
8	0	9.19	-0.76
9	0	7.80	-3.16
10	0	7.80	-1.56
11	0	9.19	-3.95
12	0	10.57	-1.59
13	0	10.57	-3.13
14	0	11.90	-0.82
15	0	6.26	-3.17
16	0	11.90	-3.90
17	0	13.24	-1.59
18	0	13.24	-3.13
19	0	5.49	-4.50
20	0	14.61	-0.76
21	0	14.61	-3.95
22	0	16.01	-1.56
23	0	16.01	-3.16
24	0	17.44	-0.70
25	0	17.44	-4.02
26	0	18.90	-1.52
27	0	18.90	-3.19
28	0	9.21	0.78
29	0	7.89	1.56
30	0	6.54	0.81
31	0	5.22	1.59
32	0	5.24	3.13
33	0	3.88	0.84
34	0	3.91	3.92
35	0	6.58	3.89
36	0	6.60	5.43
37	0	7.90	3.10
38	0	17.43	0.84
39	0	18.75	1.62

Tabel 4.24: Hasil *Docking* Senyawa SA2014 dan *Doxorubicin* terhadap Protein Cyclin D1

Ligan	Skor *Docking* (Nilai Energi)
SA2014	-2.98
Doxorubicin	-1.66

Molecular docking menghasilkan *output* berupa skor yang menggambarkan energi total ikatan protein-ligan. Semakin rendah skor suatu hasil *docking* berarti semakin rendah energi yang digunakan untuk berikatan dan menunjukkan interaksi antara kompleks protein-ligan yang semkain stabil sehingga semakin poten. Suatu senyawa dapat dikatakan lebih poten dari senyawa lainnya dengan membandingkan hasil skor *docking* keduanya. Hasil skor *docking* antara senyawa alkaloid SA2014 dan doxorubicin dengan protein cyclin D1 dapat dilihat pada Tabel 4.24. Hasil skor *docking* dari kedua senyawa tersebut memiliki hasil yang berbeda. Skor antara ligan SA2014 dengan protein cyclin D1 lebih rendah dari skor antara doxorubicin dengan protein cyclin D1. Hal ini dapat diartikan bahwa afinitas ligan SA2014 terhadap protein cyclin D1 lebih tinggi dibandingkan doxorubicin.

BAB 5
KESIMPULAN DAN SARAN

Dari analisis dan pembahasan yang sudah dilakukan, dapat ditarik kesimpulan serta saran untuk pengembangan dan perbaikan penelitian selanjutnya.

5.1 Kesimpulan

Berdasarkan pada pembahasan yang telah dipaparkan, dapat diambil beberapa kesimpulan yaitu:

1. *Firefly algorithm* dapat diterapkan untuk menyelesaikan *molecular docking*. Terdapat beberapa langkah untuk menyelesaikan *molecular docking* dengan *firefly algorithm*. Langkah pertama adalah persiapan data menggunakan Chimera dan Autodock sehingga diperoleh data yang menjadi input untuk *firefly algorithm*. Kemudian menentukan nilai parameter yang diperlukan oleh *firefly algorithm*. Setelah itu, dilakukan optimasi untuk memperoleh konformasi ligan-protein yang menghasilkan energi minimum. Dalam optimasi ini, solusi yang digunakan merupakan suatu vektor yang terdiri dari $n+7$ variabel. Tiga nilai pertama mewakili translasi ligan, empat nilai berikutnya mewakili orientasi ligan, dan n nilai yang tersisa merupakan sudut torsi ligan. Fungsi objektif untuk permasalahan ini adalah meminimumkan nilai energi yang dihitung menggunakan interpolasi trilinier. Proses ini akan dilakukan terus menerus hingga batas iterasi terpenuhi sehingga diperoleh konformasi ligan-protein dengan nilai energi minimum.

2. Berdasarkan hasil evaluasi untuk kompleks 2cpp dan 3ptb, solusi terbaik yaitu solusi yang menghasilkan nilai energi terendah dan RMSD kurang dari 2Å diperoleh dengan menggunakan nilai $\alpha = 1.0$. Kemudian untuk data uji, hasil *docking* senyawa SA2014 memiliki skor -2.98, sedangkan hasil *docking* doxorubicin memiliki skor -1.66. Hal ini dapat diartikan bahwa afinitas ligan SA2014 terhadap protein cyclin D1 lebih tinggi dibandingkan doxorubicin, sehingga senyawa SA2014 memiliki potensi yang besar sebagai antikanker.

5.2 Saran

Saran yang diberikan oleh penulis untuk penelitian selanjutnya antara lain adalah:

1. Dalam penelitian ini, protein yang digunakan bersifat kaku. Akan lebih baik apabila protein yang digunakan bersifat fleksibel karena akan lebih mendekati keadaan riilnya.

2. Mengembangkan algoritma-algoritma baru untuk menyelesaikan *molecular docking*.

3. Mengembangkan fungsi objektif yang dapat digunakan untuk menyelesaikan *molecular docking*.

DAFTAR PUSTAKA

Ali, N., Othman, M. A., Husain, M. N., dan Misran, M. H. (2014), "A Review of Firefly Algorithm", *ARPN Journal of Engineering and Applied Sciences*, Vol.9, pp. 1732-736.

Apostolopoulos, T. dan Vlachos, A. (2010), "Application of the Firefly Algorithm for Solving the Economic Emissions Load Dispatch Problem", *Intenational Journal of Combinatorics*, Vol.2011.

Arwoko, H., Asmawati, E., dan Limanto, S. (2018), "Interpolasi Gerakan Rotasi Animasi 3D Menggunakan Bilangan Quaternion dan Implementasinya pada Fungsi SLERP (Spherical Linear Interpolation)", *Seminar Nasional Inovasi dan Aplikasi Teknologi di Industri*.

Camacho, E. L., Godoy, M. J. G., Nieto, J. G., Nebro, A. J., dan Montes, J. F. A. (2015), "Solving Molecular Flexible Docking Problems with Metaheuristics: A Comparative Study", *Applied Soft Computing*, Vol.28, pp. 379-393.

Carugo, O. dan Pongor, S. (2001), "A Normalized Root-Mean-Square Distance for Comparing Protein Three-Dimensional Structures", *Protein Science*, Vol. 10, pp. 1470-1473.

de Magalhães, C.S., Barbosa, H. J. C., dan Darddenne, L. E. (2004), "A Genetic Algorithm for the Ligand-Protein Docking Problem", *Genetics and Molecular Biology*, Vol. 27, pp. 605-610.

Diandana, R. (2009), *Mengenal Seluk Beluk Kanker*, Yogyakarta: Katahati.

Erickson, J. A., Jalaie, M., Robertson, D. H., Lewis, R. A., dan Vieth, M. (2004), "Lesson in Molecular Recognition: The Effect of Ligand and Protein Flexibility on Molecular Docking Accuracy", *J. Med. Chem.*, Vol. 47, pp. 45-55.

Franks, L. M dan Teich, N. M. (1998), *Cellular and Molecular Biology of Cancer, Third Edition*, New York: Oxford University Press Inc.

Funkhouser, T. (2007), *Protein-Ligand Docking Methods*, New Jersey, USA: Princeton University.

Gane, P. J. dan Dean, P. M. (2000), "Recent Advances in Structure-Based Rational Drug Design", *Current Opinion in Structural Biology*, Vol. 10, pp. 401-404.

Giordano, A. dan Normanno, N. (2009), *Breast Cancer in The Post-Genomic Era*, USA: Humana Press.

Hanahan, D. dan Weinberg, R. A. (2011), "Hallmarks of Cancer: The Next Generation", *Cell*, Vol. 144, pp. 646-674.

Horton, H. R., dkk. (2006), *Principles of Biochemistry, Fourth Edition*, NJ: Pearson/Prentice Hall.

Hou, T., Wang, J., Chen, L. dan Xu, X. (1999), "Automated Docking of Peptides and Proteins by Using a Genetic Algorithm Combined with a Tabu Search", *Protein Engineering*, Vol. 12, pp. 639-647.

Kadlubiak, Kristian. (2015), *Fast Tissue Image Reconstruction Using a Graphics Card*, Thesis, Brno University of Technology.

Korb, O., Stutzle, T., dan Exner, T. E. (2007), "An Ant Colony Optimization Approach to Flexible Protein-Ligand Docking", *Swarm Intell*, Vol. 1, pp. 115-134.

Lumongga, F. (2008), *Invasi Sel Kanker*, Departemen Patologi Anatomi UMS.

Mahdiyah, U., Imah, E. M. dan Irawan, M. I. (2016), "Integrating Data Selection and Extreme Learning Machine to Predict Protein-Ligand Binding Site", *Contemporary Engineering Sciences*, Vol. 9, pp. 791-797.

Mukesh, B. dan Rakesh, K. (2011), "Molecular Docking: A Review", *Int J Res Ayurv Pharm*, Vol. 2, pp. 1746-1751.

Murti, H., Boediono, A., Setiawan, B. dan Sandra, F. (2007), "Regulaasi Siklus Sel: Kunci Sukses (Somatic Cell Nuclear Transfer)", *CDK*, Vol. 34, pp. 312-316.

Nurhayati, A. P. D., Rarastoeti, P., Subagus, W., Istriyati dan Voogt, N. J. D. (2014), "The Anticancer Activity of Marine Sponge Cinachyrella sp. (Family Tetillidae)", *The Journal for Technology and Science*, Vol. 25.

Nurhayati, A. P. D., Santoso, N., Setiawan, E. dan Lianingsih, F. (2017), "Molecular Docking of Alkaloid Compound SA2014 from Marine Sponges Cinachyrella anomala Towards P53 Protein", *International Journal of Drug Discovery*, Vol. 8, pp. 247-249.

Rawuh (1993), *Geometri Transformasi*, Jakarta: Proyek Pembinaan Kependidikan Pendidikan Tinggi.

Reddy, B. R. dan Reddy, L. P. (2013), "Harmony Search Optimization for Flexible Docking", *International Journal of Innovetive Technology and Research*, Vol. 1, pp. 615-616.

Shen, S. N dan Tuszynki, J. A. (2008), *Theory and Mathematical Methods in Bioinformatics*.

Thomsen, R. (2003), "Flexible Ligand Docking Using Differential Evolution", *Conference: Evolutionary Computation*, Vol. 4.

Vermeulen, K., Berneman, Z. N. dan Bockstaele, D. R. (2003), "Cell Cycle and Apoptosis", *Cell Prolif*, Vol. 36, pp. 165-175.

Yang, K., Hitomi, M. dan Stacey, D. W. (2006), "Variations in Cyclin D1 Levels Through The Cell Cycle Determine The Proliferative Fate of a Cell", *Cell Division*, Vol. 1.

Yang, X. S. (2014), *Nature-Inspired Optimization Algorithms*, Elsevier, London.

LAMPIRAN A
Source Code dari Tesis-docking.java dalam *Package* Tesis

```
package tesis;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Arrays;
import java.util.Scanner;

public class Tesis_docking {

public static Connection con;
public static Statement stm;
public static String kompleks;
public static float Xmin, Ymin, Zmin, Xmax, Ymax, Zmax, centerx, centery,
centerz, sisax, sisay, sisaz, sx, sy, sz, rmsd, dis, xligan, yligan, zligan;
public static int Ntor, Natom_ligan, Natom_reseptor, jml_individu;
public static double [][] solution, hasil, solut;
public static float [] fitness, orix, oriy, oriz, ori_x, ori_y,ori_z,
oriX, oriY, oriZ, xrandom, yrandom, zrandom;
public static float [] xawal, yawal, zawal, xAwal, yAwal, zAwal, muatan,
vdw, desol, elec, x_akhir, y_akhir, z_akhir, xtmp, ytmp,ztmp;
public static float [][] x_temp, y_temp, z_temp, xakhir, yakhir, zakhir,
x0, x1, y0, y1, z0, z1, x, y, z, q000, q100, q010, q001, q110, q101, q011,
q111,d000, d100, d010, d001, d110, d101, d011, d111,e000, e100, e010, e001,
e110, e101, e011, e111 ;
public static double [] intensitas, solusi_baru, fterbaik, gbest, sol;
public static double alpha = 0.1, gamma = 1, beta_awal=1,
i_gbest = -100000000, fit_gbest;
public static double r, beta, i_fterbaik, best_fitness;
public static int iterasi, indeks_terbaik, it;
public static String [] tipe_atom;
public static int [] torsi1, torsi2, jml_atomtorsi;
public static int[][] atomtorsi;

public static void main(String[] args) throws SQLException {
Scanner input = new Scanner(System.in);
```

```
System.out.print("Masukkan nama kompleks : ");
kompleks = input.nextLine();
System.out.print("Masukkan jumlah individu : ");
jml_individu = input.nextInt();
System.out.print("Masukkan jumlah iterasi : ");
iterasi = input.nextInt();

try{
String url = "jdbc:mysql://localhost/tesis";
String user = "root";
String pass = "";
Class.forName("com.mysql.jdbc.Driver");
con = DriverManager.getConnection(url, user, pass);
stm = con.createStatement();
System.out.println("Koneksi berhasil");
ResultSet query = stm.executeQuery("SELECT * FROM `kompleks` WHERE
`nama_kompleks` = '"+kompleks+"'");
while(query.next()){
Ntor = query.getInt("ntor");
Natom_ligan = query.getInt("natom_ligan");
Natom_reseptor = query.getInt("natom_reseptor");
Xmin = query.getFloat("xmin");
Ymin = query.getFloat("ymin");
Zmin = query.getFloat("zmin");
Xmax = query.getFloat("xmax");
Ymax = query.getFloat("ymax");
Zmax = query.getFloat("zmax");
centerx = query.getFloat("centerx");
centery = query.getFloat("centery");
centerz = query.getFloat("centerz");
}

for(int i = 0; i<Natom_ligan; i++){
ResultSet query1 = stm.executeQuery("SELECT `tipe_atom`, `x_atom`, `y_atom`,
`z_atom`, `charge` FROM `atom_ligan` WHERE `n_kompleks` = '"+kompleks+"'
AND `no_atom` = "+(i+1));
while(query1.next()){
tipe_atom[i] = query1.getString("tipe_atom");
xAwal[i] = query1.getFloat("x_atom");
yAwal[i] = query1.getFloat("y_atom");
zAwal[i] = query1.getFloat("z_atom");
muatan[i] = query1.getFloat("charge");
}
}
```

```
ResultSet quer = stm.executeQuery("SELECT MAX(‘x_atom‘), MAX(‘y_atom‘),
MAX(‘z_atom‘) FROM ‘atom_ligan‘ WHERE ‘n_kompleks‘=’"+kompleks+"’");
while(quer.next()){
xligan = quer.getFloat("MAX(‘x_atom‘)");
yligan = quer.getFloat("MAX(‘y_atom‘)");
zligan = quer.getFloat("MAX(‘z_atom‘)");
}

for(int i = 0; i<Ntor; i++){
ResultSet query1 = stm.executeQuery("SELECT ‘atom1‘, ‘atom2‘ FROM
‘torsi‘ WHERE ‘n_kompleks‘ = ’"+kompleks+"’ AND ‘no_torsi‘ = "+(i+1));
while(query1.next()){
torsi1[i] = query1.getInt("atom1");
torsi2[i] = query1.getInt("atom2");
}
}

for(int i = 0; i<Ntor; i++){
ResultSet query1 = stm.executeQuery("SELECT COUNT(*) FROM
‘rotasi_torsi‘ WHERE ‘nama_kompleks‘ = ’"+kompleks+"’ AND
‘torsi‘ = "+(i+1));
while(query1.next()){
jml_atomtorsi[i] = query1.getInt("COUNT(*)");
}
int j = 0;
ResultSet query2 = stm.executeQuery("SELECT ‘no_atom‘ FROM
‘rotasi_torsi‘ WHERE ‘nama_kompleks‘ = ’"+kompleks+"’ AND
‘torsi‘ = "+(i+1));
while(query2.next()){
atomtorsi[i][j] = query2.getInt("no_atom");
j=j+1;
}

ResultSet query3 = stm.executeQuery("SELECT ‘x_atom‘, ‘y_atom‘, ‘z_atom‘
FROM ‘atom_ligan‘ WHERE ‘n_kompleks‘ = ’"+kompleks+"’ AND
‘no_atom‘ = "+torsi1[i]);
while(query3.next()){
oriX[i] = query3.getFloat("x_atom");
oriY[i] = query3.getFloat("y_atom");
oriZ[i] = query3.getFloat("z_atom");
}
}
}
catch(Exception e){
System.err.println("Koneksi gagal " + e.getMessage());
```

```
}
finally{
stm.close();
con.close();

}

sisax = (float) (centerx-Math.floor(centerx));
sisay = (float) (centery-Math.floor(centery));
sisaz = (float) (centerz-Math.floor(centerz));
solut = new double[jml_individu][7+Ntor];

solution = inisialisasi(jml_individu, Ntor, Xmin, Xmax, Ymin, Ymax,
Zmin, Zmax);

rmsd = 1000;
hasil = new double[2][7+Ntor];
hasil[1][0]=100000;
it = 0;
Posisi pos = new Posisi(Ntor, torsi1, Xmin, Xmax, Ymin, Ymax, Zmin,
Zmax, Natom_ligan, jml_atomtorsi, atomtorsi, solution, centerx,
centery, centerz, xAwal, yAwal, zAwal, oriX, oriY, oriZ);
xrandom = pos.get_xrandom();
yrandom = pos.get_yrandom();
zrandom = pos.get_zrandom();
oriX = pos.get_oriX();
oriY = pos.get_oriY();
oriZ = pos.get_oriZ();
centerx = pos.get_centerx();
centery = pos.get_centery();
centerz = pos.get_centerz();
Posisi posisi = new Posisi(jml_individu, Ntor, Natom_ligan, jml_atomtorsi,
atomtorsi, solution, centerx, centery, centerz, xrandom, yrandom,
zrandom, oriX, oriY, oriZ);
xakhir = posisi.get_xakhir();
yakhir = posisi.get_yakhir();
zakhir = posisi.get_zakhir();
fitness = hitung_fitness();
System.out.println("Fitness:");
System.out.println(Arrays.toString(fitness));
intensitas = hitung_intensitas();

while(it<iterasi){
System.out.println("Iterasi "+(it+1));
solution = bandingkan_firefly();
```

```
Posisi position = new Posisi(jml_individu, Ntor, Natom_ligan, jml_atomtorsi,
atomtorsi, solution, centerx, centery, centerz, xrandom, yrandom,
zrandom, oriX, oriY, oriZ);
xakhir = position.get_xakhir();
yakhir = position.get_yakhir();
zakhir = position.get_zakhir();
fitness = hitung_fitness();
intensitas = hitung_intensitas();
hasil = firefly_terbaik();
rmsd = hitung_rmsd();
it++;
}
}

public static double[][] inisialisasi(int jml_individu, int Ntor,
float Xmin, float Xmax, float Ymin, float Ymax, float Zmin, float Zmax){
double[][] Solution = new double [jml_individu][7+Ntor];
for(int k = 0; k<jml_individu; k++){
double[] solusi = new double [7+Ntor];
solusi[0] = (double)((Xmax-Xmin)*Math.random()+(Xmin));
solusi[1] = (double)((Ymax-Ymin)*Math.random()+(Ymin));
solusi[2] = (double)((Zmax-Zmin)*Math.random()+(Zmin));
solusi[3] = (double)((2)*Math.random()+(-1));
solusi[4] = (double)((2)*Math.random()+(-1));
solusi[5] = (double)((2)*Math.random()+(-1));
System.arraycopy(solusi, 0, Solution[k], 0, 6);
for(int i = 6; i<7+Ntor; i++){
solusi[i] = (double)((360)*Math.random()+(-180));
Solution[k][i] = solusi[i];
}
}
return Solution;
}

public static float[] hitung_fitness() throws SQLException{
try{
String url = "jdbc:mysql://localhost/tesis";
String user = "root";
String pass = "";
Class.forName("com.mysql.jdbc.Driver");
con = DriverManager.getConnection(url, user, pass);
stm = con.createStatement();
//System.out.println("Koneksi berhasil");
for(int i = 0;i<jml_individu; i++){
for(int j = 0;j<Natom_ligan; j++){
```

```
if(xakhir[i][j]<Xmin || xakhir[i][j]>Xmax || yakhir[i][j]<Ymin ||
yakhir[i][j]>Ymax || zakhir[i][j]<Zmin || zakhir[i][j]>Zmax){
sx = (float)((xakhir[i][j]-Xmin)/0.375);
if(sx<0){
x0[i][j] = (float)(Xmin+(Math.ceil(sx)*0.375));
}
else{
x0[i][j] = (float)(Xmin+(Math.floor(sx)*0.375));
}
x1[i][j] = (float) (x0[i][j]+0.375);
sy = (float)((yakhir[i][j]-Ymin)/0.375);
if(sy<0){
y0[i][j] = (float)(Ymin+(Math.ceil(sy)*0.375));
}
else{
y0[i][j] = (float)(Ymin+(Math.floor(sy)*0.375));
}
y1[i][j] = (float) (y0[i][j]+0.375);
sz = (float)((zakhir[i][j]-Zmin)/0.375);
if(sz<0){
z0[i][j] = (float)(Zmin+(Math.ceil(sz)*0.375));
}
else{
z0[i][j] = (float)(Zmin+(Math.floor(sz)*0.375));
}
z1[i][j] = (float) (z0[i][j]+0.375);

d000[i][j] = 1;
e000[i][j] = 1;
q000[i][j] = 1;
d100[i][j] = 1;
e100[i][j] = 1;
q100[i][j] = 1;
d010[i][j] = 1;
e010[i][j] = 1;
q010[i][j] = 1;
d001[i][j] = 1;
e001[i][j] = 1;
q001[i][j] = 1;
d110[i][j] = 1;
e110[i][j] = 1;
q110[i][j] = 1;
d101[i][j] = 1;
e101[i][j] = 1;
q101[i][j] = 1;
```

```
d011[i][j] = 1;
e011[i][j] = 1;
q011[i][j] = 1;
d111[i][j] = 1;
e111[i][j] = 1;
q111[i][j] = 1;
}

else{
ResultSet query30 = stm.executeQuery("SELECT `koorx`, `koory`, `koorz`
FROM `grid_data_"+kompleks+"` USE INDEX (`koorx`, `koory`, `koorz`)
WHERE (`koorx`>="+xakhir[i][j]+"-0.375 AND `koorx`<"+xakhir[i][j]+")
AND (`koory`>="+yakhir[i][j]+"-0.375 AND `koory`<"+yakhir[i][j]+")
AND (`koorz`>="+zakhir[i][j]+"-0.375 AND `koorz`<"+zakhir[i][j]+")") ;
while(query30.next()){
x0[i][j] = query30.getFloat("koorx");
y0[i][j] = query30.getFloat("koory");
z0[i][j] = query30.getFloat("koorz");
}

ResultSet query31 = stm.executeQuery("SELECT `koorx`, `koory`, `koorz`
FROM `grid_data_"+kompleks+"` USE INDEX (`koorx`, `koory`, `koorz`)
WHERE (`koorx`>="+xakhir[i][j]+" AND `koorx`<"+xakhir[i][j]+"+0.375)
AND (`koory`>="+yakhir[i][j]+" AND `koory`<"+yakhir[i][j]+"+0.375)
AND (`koorz`>="+zakhir[i][j]+" AND `koorz`<"+zakhir[i][j]+"+0.375)") ;
while(query31.next()){
x1[i][j] = query31.getFloat("koorx");
y1[i][j] = query31.getFloat("koory");
z1[i][j] = query31.getFloat("koorz");
}

ResultSet query22 = stm.executeQuery("SELECT `desolvasi`, `electrostatic`,
`energi"+tipe_atom[j]+"` FROM `grid_data_"+kompleks+"` USE INDEX
(`koorx`, `koory`, `koorz`) WHERE (`koorx`>="+x0[i][j]+"-0.05 AND
`koorx`<"+x0[i][j]+"+0.05) AND (`koory`>="+y0[i][j]+"-0.05 AND
`koory`<"+y0[i][j]+"+0.05) AND (`koorz`>="+z0[i][j]+"-0.05 AND
`koorz`<"+z0[i][j]+"+0.05)") ;
while(query22.next()){
d000[i][j] = query22.getFloat("desolvasi");
e000[i][j] = query22.getFloat("electrostatic");
q000[i][j] = query22.getFloat("energi"+tipe_atom[j]);
}

ResultSet query23 = stm.executeQuery("SELECT `desolvasi`, `electrostatic`,
`energi"+tipe_atom[j]+"` FROM `grid_data_"+kompleks+"` USE INDEX
```

```
(‘koorx‘, ‘koory‘, ‘koorz‘) WHERE (‘koorx‘>="+x1[i][j]+"-0.05 AND
‘koorx‘<"+x1[i][j]+"+0.05) AND (‘koory‘>="+y0[i][j]+"-0.05 AND
‘koory‘<"+y0[i][j]+"+0.05) AND (‘koorz‘>="+z0[i][j]+"-0.05 AND
‘koorz‘<"+z0[i][j]+"+0.05)") ;
while(query23.next()){
d100[i][j] = query23.getFloat("desolvasi");
e100[i][j] = query23.getFloat("electrostatic");
q100[i][j] = query23.getFloat("energi"+tipe_atom[j]);
}

ResultSet query24 = stm.executeQuery("SELECT ‘desolvasi‘, ‘electrostatic‘,
‘energi"+tipe_atom[j]+"‘ FROM ‘grid_data_"+kompleks+"‘ USE INDEX
(‘koorx‘, ‘koory‘, ‘koorz‘) WHERE (‘koorx‘>="+x0[i][j]+"-0.05 AND
‘koorx‘<"+x0[i][j]+"+0.05) AND (‘koory‘>="+y1[i][j]+"-0.05 AND
‘koory‘<"+y1[i][j]+"+0.05) AND (‘koorz‘>="+z0[i][j]+"-0.05 AND
‘koorz‘<"+z0[i][j]+"+0.05)") ;
while(query24.next()){
d010[i][j] = query24.getFloat("desolvasi");
e010[i][j] = query24.getFloat("electrostatic");
q010[i][j] = query24.getFloat("energi"+tipe_atom[j]);
}

ResultSet query25 = stm.executeQuery("SELECT ‘desolvasi‘, ‘electrostatic‘,
‘energi"+tipe_atom[j]+"‘ FROM ‘grid_data_"+kompleks+"‘ USE INDEX
(‘koorx‘, ‘koory‘, ‘koorz‘) WHERE (‘koorx‘>="+x0[i][j]+"-0.05 AND
‘koorx‘<"+x0[i][j]+"+0.05) AND (‘koory‘>="+y0[i][j]+"-0.05 AND
‘koory‘<"+y0[i][j]+"+0.05) AND (‘koorz‘>="+z1[i][j]+"-0.05 AND
‘koorz‘<"+z1[i][j]+"+0.05)") ;
while(query25.next()){
d001[i][j] = query25.getFloat("desolvasi");
e001[i][j] = query25.getFloat("electrostatic");
q001[i][j] = query25.getFloat("energi"+tipe_atom[j]);
}

ResultSet query26 = stm.executeQuery("SELECT ‘desolvasi‘, ‘electrostatic‘,
‘energi"+tipe_atom[j]+"‘ FROM ‘grid_data_"+kompleks+"‘ USE INDEX
(‘koorx‘, ‘koory‘, ‘koorz‘) WHERE (‘koorx‘>="+x1[i][j]+"-0.05 AND
‘koorx‘<"+x1[i][j]+"+0.05) AND
(‘koory‘>="+y1[i][j]+"-0.05 AND
‘koory‘<"+y1[i][j]+"+0.05) AND (‘koorz‘>="+z0[i][j]+"-0.05 AND
‘koorz‘<"+z0[i][j]+"+0.05)") ;
while(query26.next()){
d110[i][j] = query26.getFloat("desolvasi");
e110[i][j] = query26.getFloat("electrostatic");
q110[i][j] = query26.getFloat("energi"+tipe_atom[j]);
```

```
}

ResultSet query27 = stm.executeQuery("SELECT ‘desolvasi‘, ‘electrostatic‘,
‘energi"+tipe_atom[j]+"‘ FROM ‘grid_data_"+kompleks+"‘ USE INDEX
(‘koorx‘, ‘koory‘, ‘koorz‘) WHERE (‘koorx‘>="+x1[i][j]+"-0.05 AND
‘koorx‘<"+x1[i][j]+"+0.05) AND (‘koory‘>="+y0[i][j]+"-0.05 AND
‘koory‘<"+y0[i][j]+"+0.05) AND (‘koorz‘>="+z1[i][j]+"-0.05 AND
‘koorz‘<"+z1[i][j]+"+0.05)") ;
while(query27.next()){
d101[i][j] = query27.getFloat("desolvasi");
e101[i][j] = query27.getFloat("electrostatic");
q101[i][j] = query27.getFloat("energi"+tipe_atom[j]);
}

ResultSet query28 = stm.executeQuery("SELECT ‘desolvasi‘, ‘electrostatic‘,
‘energi"+tipe_atom[j]+"‘ FROM ‘grid_data_"+kompleks+"‘ USE INDEX
(‘koorx‘, ‘koory‘, ‘koorz‘) WHERE (‘koorx‘>="+x0[i][j]+"-0.05 AND
‘koorx‘<"+x0[i][j]+"+0.05) AND (‘koory‘>="+y1[i][j]+"-0.05 AND
‘koory‘<"+y1[i][j]+"+0.05) AND (‘koorz‘>="+z1[i][j]+"-0.05 AND
‘koorz‘<"+z1[i][j]+"+0.05)") ;
while(query28.next()){
d011[i][j] = query28.getFloat("desolvasi");
e011[i][j] = query28.getFloat("electrostatic");
q011[i][j] = query28.getFloat("energi"+tipe_atom[j]);
}

ResultSet query29 = stm.executeQuery("SELECT ‘desolvasi‘, ‘electrostatic‘,
‘energi"+tipe_atom[j]+"‘ FROM ‘grid_data_"+kompleks+"‘ USE INDEX
(‘koorx‘, ‘koory‘, ‘koorz‘) WHERE (‘koorx‘>="+x1[i][j]+"-0.05 AND
‘koorx‘<"+x1[i][j]+"+0.05) AND (‘koory‘>="+y1[i][j]+"-0.05 AND
‘koory‘<"+y1[i][j]+"+0.05) AND (‘koorz‘>="+z1[i][j]+"-0.05 AND
‘koorz‘<"+z1[i][j]+"+0.05)") ;
while(query29.next()){
d111[i][j] = query29.getFloat("desolvasi");
e111[i][j] = query29.getFloat("electrostatic");
q111[i][j] = query29.getFloat("energi"+tipe_atom[j]);
}
}
}
}
}

catch(Exception e){
System.err.println("Koneksi gagal " + e.getMessage());
}
```

```
finally{
stm.close();
con.close();
}

for (int i = 0; i < jml_individu; i++){
fitness[i] = 0;
vdw[i] = 0;
desol[i] = 0;
elec[i] = 0;
for(int k = 0; k < Natom_ligan;k++){
vdw[i] = vdw[i] + triLerp(xakhir[i][k], yakhir[i][k], zakhir[i][k],
q000[i][k], q001[i][k], q010[i][k], q011[i][k], q100[i][k], q101[i][k],
q110[i][k], q111[i][k], x0[i][k], x1[i][k], y0[i][k], y1[i][k], z0[i][k],
z1[i][k]);
desol[i] = desol[i] + Math.abs(muatan[k])*triLerp(xakhir[i][k], yakhir[i][k],
zakhir[i][k], d000[i][k], d001[i][k], d010[i][k], d011[i][k], d100[i][k],
d101[i][k], d110[i][k], d111[i][k], x0[i][k], x1[i][k], y0[i][k], y1[i][k],
z0[i][k], z1[i][k]);
elec[i] = elec[i] + muatan[k]*triLerp(xakhir[i][k], yakhir[i][k], zakhir[i][k],
e000[i][k], e001[i][k], e010[i][k], e011[i][k], e100[i][k], e101[i][k],
e110[i][k], e111[i][k], x0[i][k], x1[i][k], y0[i][k], y1[i][k], z0[i][k],
z1[i][k]);

}
fitness[i] = (float) (vdw[i]+desol[i]+elec[i]+ 0.2983*Ntor);
}
return fitness;
}

public static float lerp(float x, float x1, float x2, float q00, float q01) {
if(x2 != x1){
return ((x2 - x) / (x2 - x1)) * q00 + ((x - x1) / (x2 - x1)) * q01;}
else{
return 0;
}
}

public static float biLerp(float x, float y, float q11, float q12, float q21,
float q22, float x1, float x2, float y1, float y2) {
float r1 = lerp(x, x1, x2, q11, q21);
float r2 = lerp(x, x1, x2, q12, q22);
return lerp(y, y1, y2, r1, r2);
}
```

```
public static float triLerp(float x, float y, float z, float q000, float q001,
float q010, float q011, float q100, float q101, float q110, float q111,
float x1, float x2, float y1, float y2, float z1, float z2) {
float x00 = lerp(x, x1, x2, q000, q100);
float x10 = lerp(x, x1, x2, q010, q110);
float x01 = lerp(x, x1, x2, q001, q101);
float x11 = lerp(x, x1, x2, q011, q111);
float r0 = lerp(y, y1, y2, x00, x01);
float r1 = lerp(y, y1, y2, x10, x11);
return lerp(z, z1, z2, r0, r1);
}

public static double[] hitung_intensitas(){
for (int i = 0; i < jml_individu; i++){
intensitas[i]=-1*fitness[i];
}
return intensitas;
}

public static double[][] bandingkan_firefly() {
solut = solution;
for(int i =0; i<jml_individu; i++){
for(int o = 0; o<jml_individu; o++){

if(intensitas[i]<intensitas[o]){
double tmp = 0;
for(int k =0;k < 7+Ntor; k++){
tmp = tmp+Math.pow((solution[i][k]-solution[o][k]),2);
}
r = Math.sqrt(tmp);
beta = beta_awal*Math.exp(-gamma*Math.pow(r, 2));
solusi_baru = new double[7+Ntor];
for(int k =0;k < 7+Ntor; k++){
solusi_baru[k] = solution[i][k] + beta*(solution[o][k]-solution[i][k])+
alpha*(((1)*Math.random()+(0))-0.5);
}
if(solusi_baru[0]>Xmax || solusi_baru[0]<Xmin){
solusi_baru[0]= (Xmax-Xmin)*Math.random()+(Xmin);
}
if(solusi_baru[1]>Ymax || solusi_baru[1]<Ymin){
solusi_baru[1]= (Ymax-Ymin)*Math.random()+(Ymin);
}
if(solusi_baru[2]>Zmax || solusi_baru[2]<Zmin){
solusi_baru[2]= (Zmax-Zmin)*Math.random()+(Zmin);
}
```

```
for(int n = 3; n<6; n++){
if(solusi_baru[n]>1 || solusi_baru[n]<-1){
solusi_baru[n]= (2)*Math.random()+(-1);
}
}
for(int n = 6; n<7+Ntor; n++){
if(solusi_baru[n]>180 || solusi_baru[n]<(-180)){
solusi_baru[n] = (double)((360)*Math.random()+(-180));
}
}

System.arraycopy(solusi_baru, 0, solut[i], 0, 7+Ntor);
}
}
}

solution = solut;
return solution;
}

public static double[][] firefly_terbaik(){
double[][] Hasil = new double[2][7+Ntor];
i_fterbaik = -100000000;
indeks_terbaik = -10;
best_fitness = 10000;
for(int i = 0; i<jml_individu; i++){
if(intensitas[i]>i_fterbaik){
i_fterbaik = intensitas[i];
indeks_terbaik = i;
best_fitness = fitness[i];
}
}
fterbaik = new double[7+Ntor];
System.arraycopy(solution[indeks_terbaik], 0, fterbaik, 0, 7+Ntor);
for(int k =0;k < 7+Ntor; k++){
solution[indeks_terbaik][k] = solution[indeks_terbaik][k]+alpha*(((1)*
Math.random()+(0))-0.5);
}
if(i_gbest  <i_fterbaik){
i_gbest = i_fterbaik;
gbest = fterbaik;
fit_gbest = best_fitness;
}
Hasil[0] = gbest;
Hasil[1][0] = fit_gbest;
```

```
return Hasil;
}

public static float hitung_rmsd(){
rmsd = 0;
x_akhir = new float[Natom_ligan];
y_akhir = new float[Natom_ligan];
z_akhir= new float[Natom_ligan];
xtmp = new float[Natom_ligan];
ytmp = new float[Natom_ligan];
ztmp = new float[Natom_ligan];
double deltax = gbest[0]-centerx;
double deltay = gbest[1]-centery;
double deltaz = gbest[2]-centerz;
double deg = Math.toRadians(gbest[6]);
double norm = Math.sqrt(Math.pow(gbest[3],2)+Math.pow(gbest[4], 2)+
Math.pow(gbest[5],2));
double w = Math.cos(deg/2);
double xi = (gbest[3]*Math.sin(deg/2))/norm;
double yi = (gbest[4]*Math.sin(deg/2))/norm;
double zi = (gbest[5]*Math.sin(deg/2))/norm;

for(int n = 0; n<Natom_ligan; n++){

xawal[n] = (float) (xrandom[n]-centerx);
yawal[n] = (float) (yrandom[n]-centery);
zawal[n] = (float) (zrandom[n]-centerz);
}

for(int m = 0; m<Natom_ligan; m++){
x_akhir[m] = (float)(((1-2*yi*yi-2*zi*zi)*xawal[m]+(2*xi*yi-2*w*zi)*
yawal[m]+(2*xi*zi+2*w*yi)*zawal[m])+gbest[0]);
y_akhir[m] = (float)(((2*xi*yi+2*w*zi)*xawal[m]+(1-2*xi*xi-2*zi*zi)*
yawal[m]+(2*yi*zi-2*w*xi)*zawal[m])+gbest[1]);
z_akhir[m] = (float)(((2*xi*zi-2*w*yi)*xawal[m]+(2*yi*zi+2*w*xi)*
yawal[m]+(1-2*xi*xi-2*yi*yi)*zawal[m])+gbest[2]);
}
if(Ntor!=0){
for(int t = 0; t<Ntor; t++){
orix[t] = (float) (oriX[t]-centerx);
oriy[t] = (float) (oriY[t]-centery);
oriz[t] = (float) (oriZ[t]-centerz);
}
for(int j = 0; j<Ntor; j++){
ori_x[j] = (float)(((1-2*yi*yi-2*zi*zi)*orix[j]+(2*xi*yi-2*w*zi)*
```

```
oriy[j]+(2*xi*zi+2*w*yi)*oriz[j])+gbest[0]);
ori_y[j] = (float)(((2*xi*yi+2*w*zi)*orix[j]+(1-2*xi*xi-2*zi*zi)*
oriy[j]+(2*yi*zi-2*w*xi*oriz[j]))+gbest[1]);
ori_z[j] = (float)(((2*xi*zi-2*w*yi)*orix[j]+(2*yi*zi+2*w*xi)*
oriy[j]+(1-2*xi*xi-2*yi*yi)*oriz[j])+gbest[2]);

for(int k = 0; k<jml_atomtorsi[j]; k++){
xtmp[atomtorsi[j][k]-1]=x_akhir[atomtorsi[j][k]-1]-ori_x[j];
ytmp[atomtorsi[j][k]-1]=y_akhir[atomtorsi[j][k]-1]-ori_y[j];
ztmp[atomtorsi[j][k]-1]=z_akhir[atomtorsi[j][k]-1]-ori_z[j];
x_akhir[atomtorsi[j][k]-1]=(float) (ori_x[j]+xtmp[atomtorsi[j][k]-1]
*(Math.pow(Math.cos(Math.toRadians(gbest[7+j])),2))+
ytmp[atomtorsi[j][k]-1]*(Math.cos(Math.toRadians(gbest[7+j]))*
Math.sin(Math.toRadians(gbest[7+j]))+Math.cos(Math.toRadians
(gbest[7+j]))*Math.pow(Math.sin(Math.toRadians(gbest[7+j])),2))+
ztmp[atomtorsi[j][k]-1]*(Math.pow(Math.sin(Math.toRadians(gbest[7+j])),2)
-Math.pow(Math.cos(Math.toRadians(gbest[7+j])),2)*Math.sin(Math.toRadians
(gbest[7+j]))));
y_akhir[atomtorsi[j][k]-1] = (float) (ori_y[j]+xtmp[atomtorsi[j][k]-1]*
(-Math.cos(Math.toRadians(gbest[7+j]))*Math.sin(Math.toRadians(gbest[7+j])))+
ytmp[atomtorsi[j][k]-1]*(Math.pow(Math.cos(Math.toRadians(gbest[7+j])),2)
-Math.pow(Math.sin(Math.toRadians(gbest[7+j])),3))+
ztmp[atomtorsi[j][k]-1]*(Math.cos(Math.toRadians(gbest[7+j]))*
Math.sin(Math.toRadians(gbest[7+j]))+Math.cos(Math.toRadians(gbest[7+j]))
*Math.pow(Math.sin(Math.toRadians(gbest[7+j])),2)));
z_akhir[atomtorsi[j][k]-1] = (float) (ori_z[j]+xtmp[atomtorsi[j][k]-1]*
Math.sin(Math.toRadians(gbest[7+j]))-
ytmp[atomtorsi[j][k]-1]*Math.sin(Math.toRadians(gbest[7+j]))*
Math.cos(Math.toRadians(gbest[7+j]))+
ztmp[atomtorsi[j][k]-1]*Math.pow(Math.toRadians(Math.cos(gbest[7+j])),2));
}
}}
dis = 0;
for(int i = 0; i<Natom_ligan; i++){
dis = (float) (dis+Math.pow(x_akhir[i]-xAwal[i], 2)+Math.pow(y_akhir[i]-
yAwal[i],2)+Math.pow(z_akhir[i]-zAwal[i], 2));
rmsd = (float) (Math.sqrt(dis/Natom_ligan));
}
return rmsd;
}
}
```

LAMPIRAN B
Source Code dari Posisi.java dalam *Package* Tesis

```
package tesis;

import java.util.Arrays;

public class Posisi {
public static float Xmin, Ymin, Zmin, Xmax, Ymax, Zmax, centerx,
centery, centerz;
public static int Ntor, Natom_ligan, Natom_reseptor, jml_individu;
public static double [][] solution;
public static float [] orix, oriy, oriz, xrandom, yrandom, zrandom,
xt, yt, zt;
public static float [] xawal, yawal, zawal, ori_x, ori_y,ori_z, ox, oy, oz;
public static float [][] x_temp, y_temp, z_temp, xakhir, yakhir, zakhir;
public static int [] jml_atomtorsi;
public static int[][] atomtorsi;
public static double[] sol;

public Posisi(int jml_individu, int Ntor, int Natom_ligan, int[]
jml_atomtorsi, int[][] atomtorsi, double[][] solution, float centerx,
float centery, float centerz, float[] xAwal, float[] yAwal, float[] zAwal,
float[] oriX, float[] oriY, float[] oriZ){

for(int i = 0; i<jml_individu; i++){
double deltax = solution[i][0]-centerx;
double deltay = solution[i][1]-centery;
double deltaz = solution[i][2]-centerz;
double deg = Math.toRadians(solution[i][6]);
double norm = Math.sqrt(Math.pow(solution[i][3],2)+Math.pow(solution[i][4],
2)+Math.pow(solution[i][5],2));
double w = Math.cos(deg/2);
double xi = (solution[i][3]*Math.sin(deg/2))/norm;
double yi = (solution[i][4]*Math.sin(deg/2))/norm;
double zi = (solution[i][5]*Math.sin(deg/2))/norm;

for(int n = 0; n<Natom_ligan; n++){

xawal[n] = (float) (xAwal[n]-centerx);
yawal[n] = (float) (yAwal[n]-centery);
zawal[n] = (float) (zAwal[n]-centerz);
```

```
}
for(int m = 0; m<Natom_ligan; m++){
xakhir[i][m] = (float)(((1-2*yi*yi-2*zi*zi)*xawal[m]+(2*xi*yi-2*w*zi)*
yawal[m]+(2*xi*zi+2*w*yi)*zawal[m])+solution[i][0]);
yakhir[i][m] = (float)(((2*xi*yi+2*w*zi)*xawal[m]+(1-2*xi*xi-2*zi*zi)*
yawal[m]+(2*yi*zi-2*w*xi)*zawal[m])+solution[i][1]);
zakhir[i][m] = (float)(((2*xi*zi-2*w*yi)*xawal[m]+(2*yi*zi+2*w*xi)*
yawal[m]+(1-2*xi*xi-2*yi*yi)*zawal[m])+solution[i][2]);
}
if(Ntor!=0){
for(int t = 0; t<Ntor; t++){
orix[t] = (float) (oriX[t]-centerx);
oriy[t] = (float) (oriY[t]-centery);
oriz[t] = (float) (oriZ[t]-centerz);
}
for(int j = 0; j<Ntor; j++){
ori_x[j] = (float)(((1-2*yi*yi-2*zi*zi)*orix[j]+(2*xi*yi-2*w*zi)*
oriy[j]+(2*xi*zi+2*w*yi)*oriz[j])+solution[i][0]);
ori_y[j] = (float)(((2*xi*yi+2*w*zi)*orix[j]+(1-2*xi*xi-2*zi*zi)*
oriy[j]+(2*yi*zi-2*w*xi*oriz[j]))+solution[i][1]);
ori_z[j] = (float)(((2*xi*zi-2*w*yi)*orix[j]+(2*yi*zi+2*w*xi)*
oriy[j]+(1-2*xi*xi-2*yi*yi)*oriz[j])+solution[i][2]);

for(int k = 0; k<jml_atomtorsi[j]; k++){
x_temp[i][atomtorsi[j][k]-1]=xakhir[i][atomtorsi[j][k]-1]-ori_x[j];
y_temp[i][atomtorsi[j][k]-1]=yakhir[i][atomtorsi[j][k]-1]-ori_y[j];
z_temp[i][atomtorsi[j][k]-1]=zakhir[i][atomtorsi[j][k]-1]-ori_z[j];
xakhir[i][atomtorsi[j][k]-1]=(float)
(ori_x[j]+x_temp[i][atomtorsi[j][k]-1]
*(Math.pow(Math.cos(Math.toRadians(solution[i][7+j])),2))+
y_temp[i][atomtorsi[j][k]-1]*(Math.cos(Ma
th.toRadians(solution[i][7+j]))
*Math.sin(Math.toRadians(solution[i][7+j]
))+Math.cos(Math.toRadians(solution[i][7+
j]))*Math.pow(Math.sin(Math.toRadians(solution[i][7+j])),2))+
z_temp[i][atomtorsi[j][k]-1]*(Math.pow(Math.\
sin(Math.toRadians(solution[i][7+j])),
2)-Math.pow(Math.cos(Math.toRadians(solution[i][7+j])),2)*
Math.sin(Math.toRadians(solution[i][7+j]))));
yakhir[i][atomtorsi[j][k]-1] = (float)
(ori_y[j]+x_temp[i][atomtorsi[j][k]-1]*
(-Math.cos(Math.toRadians(solution[i][7+j
]))*Math.sin(Math.toRadians(solution[i][7+j])))+
y_temp[i][atomtorsi[j][k]-1]*(Math.pow(Math.
cos(Math.toRadians(solution[i][7+j])),
```

```
2)-
Math.pow(Math.sin(Math.toRadians(solution[i][7+j])),3))+
z_temp[i][atomtorsi[j][k]-1]*(Math.cos(Math.toRadians(solution[i][7+j]))*
Math.sin(Math.toRadians(solution[i][7+j]))+
Math.cos(Math.toRadians(solution[i][7+
j]))*
Math.pow(Math.sin(Math.toRadians(solution[i][7+j])),2)));
zakhir[i][atomtorsi[j][k]-1] = (float)
(ori_z[j]+x_temp[i][atomtorsi[j][k]-1]*
Math.sin(Math.toRadians(solution[i][7+j]))-
y_temp[i][atomtorsi[j][k]-1]*Math.sin(Math.
toRadians(solution[i][7+j]))*
Math.cos(Math.toRadians(solution[i][7+j]))+
z_temp[i][atomtorsi[j][k]-1]*Math.pow(Math.
cos(Math.toRadians(solution[i][7+j])),2));
}}
}
}
}
public float[][] get_xakhir(){
return xakhir;
}

public float[][] get_yakhir(){
return yakhir;
}

public float[][] get_zakhir(){
return zakhir;
}

public Posisi(int Ntor, int [] torsi1, float Xmin, float Xmax, float
Ymin, float Ymax, float Zmin, float Zmax, int Natom_ligan, int[]
jml_atomtorsi, int[][] atomtorsi, double[][] solution, float centerx,
float centery, float centerz, float[] xAwal, float[] yAwal, float[] zAwal,
float[] oriX, float[] oriY, float[] oriZ){
sol[0] = (double)((Xmax-Xmin-12)*Math.random()+(Xmin+6));
sol[1] = (double)((Ymax-Ymin-12)*Math.random()+(Ymin+6));
sol[2] = (double)((Zmax-Zmin-12)*Math.random()+(Zmin+6));
sol[3] = (double)((2)*Math.random()+(-1));
sol[4] = (double)((2)*Math.random()+(-1));
sol[5] = (double)((2)*Math.random()+(-1));
sol[6] = (double)((60)*Math.random()+(-30));
for(int i = 7; i<7+Ntor; i++){
sol[i] = (double)((60)*Math.random()+(-30));
```

```
}
double deltax = sol[0]-centerx;
double deltay = sol[1]-centery;
double deltaz = sol[2]-centerz;
double deg = Math.toRadians(sol[6]);
double norm = Math.sqrt(Math.pow(sol[3],2)+Math.pow(sol[4], 2)+
Math.pow(sol[5],2));
double w = Math.cos(deg/2);
double xi = (sol[3]*Math.sin(deg/2))/norm;
double yi = (sol[4]*Math.sin(deg/2))/norm;
double zi = (sol[5]*Math.sin(deg/2))/norm;

for(int n = 0; n<Natom_ligan; n++){
xawal[n] = (float) (xAwal[n]-centerx);
yawal[n] = (float) (yAwal[n]-centery);
zawal[n] = (float) (zAwal[n]-centerz);
}

for(int m = 0; m<Natom_ligan; m++){
xrandom[m] = (float)(((1-2*yi*yi-2*zi*zi)*xawal[m]+(2*xi*yi-2*w*zi)*
yawal[m]+(2*xi*zi+2*w*yi)*zawal[m])+sol[0]);
yrandom[m] = (float)(((2*xi*yi+2*w*zi)*xawal[m]+(1-2*xi*xi-2*zi*zi)*
yawal[m]+(2*yi*zi-2*w*xi)*zawal[m])+sol[1]);
zrandom[m] = (float)(((2*xi*zi-2*w*yi)*xawal[m]+(2*yi*zi+2*w*xi)*
yawal[m]+(1-2*xi*xi-2*yi*yi)*zawal[m])+sol[2]);
}
if(Ntor!=0){
for(int t = 0; t<Ntor; t++){
orix[t] = (float) (oriX[t]-centerx);
oriy[t] = (float) (oriY[t]-centery);
oriz[t] = (float) (oriZ[t]-centerz);
}
for(int j = 0; j<Ntor; j++){
ori_x[j] = (float)(((1-2*yi*yi-2*zi*zi)*orix[j]+(2*xi*yi-2*w*zi)*
oriy[j]+(2*xi*zi+2*w*yi)*oriz[j])+sol[0]);
ori_y[j] = (float)(((2*xi*yi+2*w*zi)*orix[j]+(1-2*xi*xi-2*zi*zi)*
oriy[j]+(2*yi*zi-2*w*xi*oriz[j]))+sol[1]);
ori_z[j] = (float)(((2*xi*zi-2*w*yi)*orix[j]+(2*yi*zi+2*w*xi)*
oriy[j]+(1-2*xi*xi-2*yi*yi)*oriz[j])+sol[2]);

for(int k = 0; k<jml_atomtorsi[j]; k++){
xt[atomtorsi[j][k]-1]=xrandom[atomtorsi[j][k]-1]-ori_x[j];
yt[atomtorsi[j][k]-1]=yrandom[atomtorsi[j][k]-1]-ori_y[j];
zt[atomtorsi[j][k]-1]=zrandom[atomtorsi[j][k]-1]-ori_z[j];
xrandom[atomtorsi[j][k]-1]=(float) (ori_x[j]+xt[atomtorsi[j][k]-1]*
```

```
(Math.pow(Math.cos(Math.toRadians(sol[7+j])),2))+
yt[atomtorsi[j][k]-1]*(Math.cos(Math.toRadians(sol[7+j]))*Math.sin
(Math.toRadians(sol[7+j]))+Math.cos(Math.toRadians(sol[7+j]))*
Math.pow(Math.sin(Math.toRadians(sol[7+j])),2))+
zt[atomtorsi[j][k]-1]*(Math.pow(Math.sin(Math.toRadians
(sol[7+j])),2)-Math.pow(Math.cos(Math.toRadians(sol[7+j])),2)*
Math.sin(Math.toRadians(sol[7+j]))));
yrandom[atomtorsi[j][k]-1] = (float) (ori_y[j]+xt[atomtorsi[j][k]-1]*
(-Math.cos(Math.toRadians(sol[7+j]))*Math.sin(Math.toRadians
(sol[7+j])))+yt[atomtorsi[j][k]-1]*(Math.pow(Math.cos(Math.toRadians
(sol[7+j])),2)-Math.pow(Math.sin(Math.toRadians(sol[7+j])),3))+
zt[atomtorsi[j][k]-1]*(Math.cos(Math.toRadians(sol[7+j]))*Math.sin(Math.
toRadians(sol[7+j]))+Math.cos(Math.toRadians(sol[7+j]))*Math.pow
(Math.sin(Math.toRadians(sol[7+j])),2)));
zrandom[atomtorsi[j][k]-1] = (float) (ori_z[j]+xt[atomtorsi[j][k]-1]*
Math.sin(Math.toRadians(sol[7+j]))-
yt[atomtorsi[j][k]-1]*Math.
sin(Math.toRadians(sol[7+j]))*
Math.cos(Math.toRadians(sol[7+j]))+
zt[atomtorsi[j][k]-1]*Math.pow
(Math.cos(Math.toRadians(sol[7+j])),2));
}}
for(int j=0; j<Ntor; j++){
ox[j]=xrandom[torsi1[j]-1];
oy[j]=yrandom[torsi1[j]-1];
oz[j]=zrandom[torsi1[j]-1];
}
}
}

public float[] get_xrandom(){
return xrandom;
}

public float[] get_yrandom(){
return yrandom;
}

public float[] get_zrandom(){
return zrandom;
}
public float[] get_oriX(){
return ox;
}
```

```
public float[] get_oriY(){
return oy;
}

public float[] get_oriZ(){
return oz;
}
public float get_centerx(){
return (float) sol[0];
}
public float get_centery(){
return (float) sol[1];
}
public float get_centerz(){
return (float) sol[2];
}
}
```

BIODATA PENULIS

Penulis bernama lengkap Zulfa Afiq Fikriya, lahir di Tabanan, 25 Agustus 1995. Anak kedua dari pasangan Shonhaji dan Heriyati, serta memiliki kakak perempuan bernama Ana Mas 'Udah. Jenjang pendidikan formal yang ditempuh yaitu SDN 1 Kediri (2001-2007), SMPN 1 Tabanan (2007-2010), MAN Negara (2010-2013). Pada tahun 2013 penulis diterima di Jurusan Matematika ITS melalui jalur tulis untuk menempuh pendidikan S1 selama empat tahun. Di Jurusan Matematika ITS penulis mengambil bidang minat ilmu komputer (Computer Science). Penulis juga aktif berorganisasi di KM ITS, yaitu sebagai staff Departemen Sains Terapan dan Keprofesian (Sainstek) di Himpunan Mahasiswa Matematika ITS (Himatika ITS) (2014-2015), sekretaris Departemen Syiar di Lembaga Dakwah Jurusan Matematika "Ibnu Muqlah" (2015-2016), dan penyusun juga aktif dalam kepanitian acara tingkat Nasional yaitu Olimpiade Matematika ITS sebagai Question Maker di dalam kampus. Penulis juga melaksanakan Kerja Praktek di PT. Megatama Innotek Nusantara. Penulis lulus S1 dari ITS pada tahun 2017 dan melanjutkan pendidikan ke jenjang S2 pada perguruan tinggi yang sama. Jika ingin memberikan saran, kritik, dan pertanyaan mengenai Tesis ini, bisa menghubungi melalui email zulfa.afiq@gmail.com. Semoga bermanfaat.