BAB II
KAJIAN TEORI

2.1 Kajian Penelitian Sebelumnya

Travelling Salesman Problem (TSP) tidak diragukan lagi adalah masalah yang secara luas dipelajari dalam optimasi kombinatorial. Dalam bahasa yang populer, TSP dapat dideskripsikan sebagai permasalahan untuk mencari jarak perjalanan minimum *n* kota, mulai dan berakhir di kota yang sama dan mengunjungi setiap kota tepat satu kali. Meskipun pernyataan dari permasalahannya terdengar sederhana, TSP sangatlah menantang dan telah menginspirasi lebih dari ribuan publikasi dikhususkan untuk algoritma dan analisis yang berupaya untuk menyelesaikannya secara lebih efektif (Rego, Gamboa, FredGlover, & ColinOsterman, 2011). Traveling Salesman Problem sering dianggap sebagai persoalan yang sangat penting (Rashid & Mosteiro, 2017). Pada penelitian sebelumnya banyak ragam algoritma telah diusulkan untuk menyelesaikan persoalan TSP, berikut pemaparannya.

Penelitian pertama adalah pembuatan itinerary wisata menggunakan TSP dan K-Means klastering. Fokus dari penelitian ini adalah mengusulkan sebuah aplikasi berbasis website untuk membatu wisatawan dalam membuat rencana peralanan wisata menggunakan metode TSP dan k-means klastering (Kholidah, 2007). Khusus untuk penyelesaian TSP, penelitian ini menggunakan Metode *Brute Froce* yaitu *hamilton circuit.* Ide dari algoritma ini adalah mencari mana jarak yang paling kecil dari *n!* dimana n adalah jumlah kota yang ingin dikunjungi. Algoritma ini akan menghasilkan solusi dari TSP yang paling optimal. Namun dengan semakin banyaknya jumlah kota, waktu proses komputasinya akan semakin lama.

Penelitian kedua yaitu megenai permasalahan rencana perjalanan wisata menggunakan *Interactive Genetic Algorithm* (IGA). Untuk menyelesaiakan masalah perencanaan perjalanan wisata, kita perlu batasan waktu, berbagai kegiatan rekreasi, layanan transit, dan prefrensi wisatawan. Namun malasah perencanaan tersebut bersifat tidak terstruktur, dimana tujuan, kriteria keputusan, dan batasan atau *constraint* lain tidak dapat sepenuhnya ditentukan sebelumnya. Penelitian ini mengusulkan sebuah model untuk menyelesaikan masalah perencanaan perjalanan tersebut menggunakan IGA atau *Interactive Genetic Algorithm.* IGA pada dasarnya adalah Algoritma Genetika kecuali fungsi fitnesnya digantikan dengan evaluasi

manusia (Hsu & Chen, 2000). Prosedur IGA pada model yang diusulkan di penelitian ini adalah sebagai berikut :

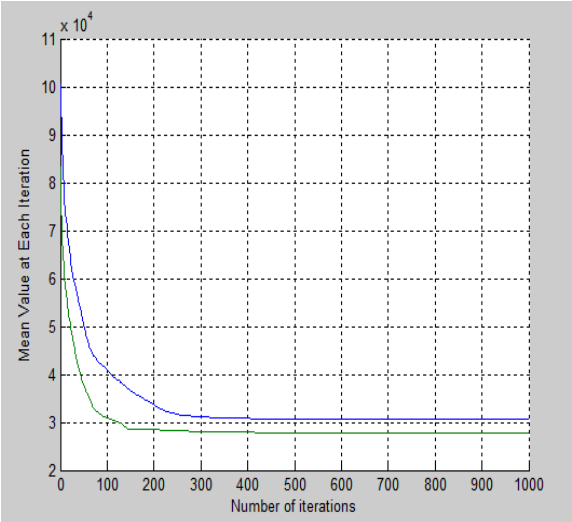
a. Sistem menghasilkan set kromosom sebagai populasi awal.
b. Kemudian fenotip dari kromosom tersebut disajikan kepada pengguna.
c. Pengguna mengevaluasi setiap kromosom yang disajikan sistem.
d. Berdasarkan evaluasi dari pengguna, IGA memilih kromosom, dan melakukan crossover atau mutasi untuk menghasilkan kromosom baru.
e. Kembali ke langkah 2 sampai prefrensi pengguna terpuaskan.

Model yang diusulkan dari penelitian kedua dipercaya dapat membantu para wisatawan dalam membuat rencana perjalanan. Serta menunjukan arah baru untuk mendukung perencanaan perjalanan wisata.

Penelitian berikutnya adalah pemecahan solusi TSP menggunakan Algoritma *Greedy*. Proses penelitian yang dilakukan yaitu dengan menggunakan data input berupa kota dan jarak antar kota kemudian Algoritma *Greedy* dengan penanda akan mengolah data input tersebut sehingga akan dihasilkan jalur terpendek dan total biayannya. Algortma *Greedy* adalah Algoritma yang memecahkan masalah dengan mencari *local optima* langkah demi langkah dimana *local optima* dari setiap langkah akan menjadi *global optima*, artinya ia akan mencari pilihan terbaik tanpa balik mengecek kembali ke belakang. Algoritma *Greedy* biasanya digunakan untuk memecahkan masalah lintasan linier dimana asal dan tujuannya adalah node yang berbeda. Dalam kasus TSP hal ini menjadi suatu kendala dimana setiap node dimungkinkan memiliki 2 arah. Oleh karena itu Algoritma *Greedy* pada penelitian ini harus dilengkapi dengan penanda untuk mengenali kota yang telah dilewati. Tanda tersebut berupa angka 0 dan 1. 0 sebagai tanda kota yang belum dikunjungi dan 1 sebagai tanda kota yang telah dikunjungi. Algoritma *Greedy* tidak selalu memberikan hasil paling optimum. Namun kelebihannya adalah dapat menemukan solusi dengan jumlah node yang banyak dengan waktu yang cepat. Setelah dibandingkan dengan metode *Brute Force*, pada penelitian ini, Algoritma *Greedy* terbukti jauh lebih efektif saat memecahkan TSP dengan jumlah kota lebih dari 20.

Berikutnya adalah penelitian tentang penyelesaian TSP menggunakan Algoritma Atom dan Algoritma Genetika. Pada studi ini Algoritma Genetika dan Algoritma Atom digunakan untuk menyelesaikan permasalahan TSP dan kemudian kedua Algoritma tersebut dibandingkan. Algoritma Genetika merupakan algoritma yang diinspirasi dari perubahan biologis dan algoritma ini menggunakan operator seperti *natural selection*, reproduksi, *crossover* dan mutasi. Sedangkan Algoritma Atom merupakan Algoritma *meta-heuristic* baru

berdasarkan proses pembentukan senyawa (Yildirim & Karci, 2013). Dengan membandingkan Algoritma Genetika dan Algoritma Atom didapati hasil sebagai berikut:



Gambar 2. 1 Perbandingan Algoritma Atom (Garis hijau) dengan Algoritma Genetika (Garis biru) untuk TSP 29 kota (Yildirim & Karci, 2013).

Dari data tersebut dapat disimpulkan bahwa Algoritma Atom memberikan hasil yang lebih baik dari Algoritma Genetika. Dimana perjalanan antar kota diselesaikan dengan jarak atau dalam Gambar 2. 1 tersebut disebut dengan *mean value of each iteration* yang lebih kecil dengan Algoritma Atom, namun kelemahan dari Algoritma Atom adalah algoritma tersebut lebih lambat waktu komputasinya daripada Algoritma Genetika (Yildirim & Karci, 2013).

Berikut rangkuman kajian yang pernah dilakukan terhadap penelitian sebelumnya:

Tabel 2. 1 Rangkuman Penelitian Sebelumnya

Peneliti	Judul	Metode	Studi Kasus
Kholidah, Kartika Nur 2017	Aplikasi Pembuat Itinerary Wisata di Provinsi Daerah Istimewa Yogyakarta dengan Pendekatan Solusi Traveling Salesman Problem dan *K-Means Clustering*	*Brute force* dan *K-Means Clustering*	Pembuatan Itinerary Wisata di Provinsi Daerah Istimewa Yogyakarta

Hsu, Fang-Cheng, Chen, Jiah-Shing, Chen, Poren	*Interactive Genetic Algorithm for Travel Itinerary Planning Problem*	*Interative Genetic Algorithm*	Pemecahan permasalahan perencanaan rencana perjalanan wisata
Lukman, Andi, AR, Rubinah, Nurhayati	Penyelesaian *Travelling Salesman Problem* dengan Algoritma *Greedy*	Algoritma *Greedy*	Penyelesaian Traveling Salesman Problem (TSP)
Yildirim, A. E., Karci, A.	*Solutions of travelling salesman problem using genetic algorithm and atom algorithm.*	*Genetic Algorithm* dan *Atom algorithm*	Penyelesaian Traveling Salesman Problem (TSP)

Studi kasus yang diambil dalam penelitian ini menggunakan parameter berupa jarak kota untuk membuat rencana perjalanan wisata. Oleh karena itu penelitian ini menggunakan pendekatan *Traveling Salesman Problem*. Metode *constrained k-means* digunakan untuk mendapatkan jumlah anggota klaster yang seimbang. Kemudian untuk menambah efisiensi waktu proses komputasi, penelitian ini menerapkan Algoritma Genetika dalam mencari solusi TSP dan mengimplementasikan teknik *paralel programming* atau *multithreading.*

2.2 Traveling Salesman Problem

Traveling Salesman Problem pertama dibahas pada abad ke-18 oleh matematikawan Inggris, Thomas Penyngton Kirkman dan matematikawan Irlandia, Sir William Rowam Hamilton (William, 2007). Dalam bahasa yang populer, TSP dapat dideskripsikan sebagai permasalahan untuk mencari jarak perjalanan minimum *n* kota, mulai dan berakhir di kota yang sama dan mengunjungi setiap kota tepat satu kali. Penyelesaian eksak terhadap masalah TSP mengharuskan untuk menghitung semua kemungkinan rute yang dapat diperoleh berdasarkan jumlah kota yang ingin dikunjungi, kemudian memilih rute mana yang memiliki jarak terpendek. Jika ada *n* kota maka kombinasi rutenya adalah *n!*. Semakin banyak jumlah kota maka kombinasi rutenya tentu juga akan semakin banyak. Dengan cara ini waktu proses komputasi akan semakin lama seiring bertambahnya jumlah kota. Dapat diilustrasikan untuk 20 kota, akan ada sebanyak $2{,}4329 \times 10^{18}$ rute. Lalu dengan mengeliminasi rute yang sama

menggunakan rumus *(n-1)!/2* didapati sebanyak 6,08226x10^{16} rute (Lukman, AR, & Nurhayati, 2011).

2.3 Algoritma Genetika

Algoritma Genetikaa sebagai cabang dari metode heuristic merupakan algoritma yang didasarkan pada proses genetika yang ada pada makhluk hidup yaitu perkembangan generasi dalam sebuah populasi mengikuti prinsip seleksi alam. Hal ini mengacu pada teori evolusi dimana generasi yang paling kuat yang akan bertahan. Dengan meniru teori evolusi ini, Algoritma Genetika dapat digunakan untuk mencari solusi permasalahan-permasalahan dalam kehidupan nyata. Algoritma ini bekerja dengan sebuah populasi dimana populasi tersebut terdiri dari kumpulan individu atau kromosom. Setiap individu merepresentasikan solusi yang mungkin dari persmasalahan yang ada. Serta masing-masing individu mempunyai nilai fitness yang akan digunakan untuk mencari solusi terbaik. Pada kasus TSP kromosom atau individu dapat direpresentasikan sebagai suatu rute perjalanan atau kumpulan kota yang ingin dikunjungi. Masing-masing kromosom akan memiliki fungsi fitnes dimana semakin kecil total jarak maka akan semakin besar nilai fitnesnya. Berdasarkan nilai fitnesnya, kromosom akan dipilih lalu dilakukan proses *crossover* dan mutasi untuk membuat generasi berikutnya. Proses ini akan diulang hingga mendapatkan generasi yang terbaik. Untuk lebih jelasnya Algoritma Genetika terdiri dari beberapa langkah sebagai berikut :

a. Langkah pertama adalah membuat populasi

   Populasi dapat didefinisikan sebagai kumpulan dari rute. Satu rute merepresentasikan sebuah kromosom yang terdiri dari kumpulan kota yang ingin dikunjungi, dan setiap kota dinamakan gen.

b. Kedua, menentukan fitness

   Setelah membuat populasi, kita akan menentukan fitnes pada setiap kromosom. Dalam kasus TSP, nilai fitnes dapat direpresentasikan sebagai jarak dimana semakin kecil jarak maka akan semakin besar nilai fitnessnya. Dengan prinsip tersebut didapati sebuah fungsi fitnes yaitu :

$$fitness = \frac{1}{\text{jarak}} \qquad (2.1)$$

c. Ketiga, memilih *parents* atau orangtua yang yang akan digunakan untuk membuat populasi selanjutnya.

Orangtua merupakan kromosom yang dipilih bedasrkan nilai fitnes mereka. Ada beberapa metode untuk memilih orangtua. Metode yang digunakan pada penelitian ini adalah metode *roulette wheel selection*. *Roulette wheel selection* merupakan metode yag paling umum dan sering digunakan dalam Algoritma Genetikaa. Metode ini bekerja dengan cara menghitung bobot relatif dari setiap fitnes kemudian membandingkan bobot tersebut dengan bilangan acak untuk memilih orangtua.

d. Keempat, melakukan crossover

   Crossover atau penyilangan, prinsipnya adalah menyilangkan gen pada kromosom orangtua pertama dan kedua untuk mendapatkan anak yang memiliki informasi gen dari kedua orangtua. Pada kasus TSP, kromosom tidak diperbolehkan memiliki gen atau kota yang sama. Untuk mengatasi hal ini kita dapat menggunakan *ordered crossover*. Dalam *ordered crossover* kita memilih subset dari orangtua pertama secara acak, kemudian mengisi sisa dari kromosom dengan gen dari orangtua kedua.

e. Kelima, melakukan mutasi

   Mutasi dapat didefinisikan sebagai perubahan acak untuk mempertahankan keragaman Genetika di dalam populasi, yang mana biasanya diterapkan dalam probabilitas rendah. Dikarenakan kasus TSP yang unik, pada proses mutasi kita menggunakan *swap mutation*. Hal ini berarti gen atau kota pada kromosom akan ditukar satu sama lain.

f. Keenam, ulangi prosesnya dalam jumlah generasi yang ditentukan.

Algoritma Genetika tidak selalu dapat memberikan hasil paling optimum. Namun solusi tersebut akan mendekati optimum. Walaupun begitu, algoritma ini dapat menjadi pilihan yang layak untuk menyelesaikan masalah optimasi termasuk TSP ketika metode eksak terlalu lama untuk menyelesaikannya.

2.4 K-Means Clustering

K-means adalah motede sederhana untuk mengklaster data ke dalam kelompok atau klaster *k* berdasarkan kerimipan data. K-means menetapkan anggota klaster dengan cara menghitung jarak *Euclidean* antara titik data dan centroid. Centroid sendiri merupakan titik tengah klaster dimana titik tersebut menjadi acuan untuk menentukan anggota klaster. Tujuan dari k-means adalah memininalkan jarak *Euclidean* titik data terhadap centroid. Proses dari k-means adalah sebagai berikut :

a. Menentukan jumlah klaster *k.*

b. Menentukan centroid dengan cara memilih *k* titik data dari data set secara acak.
c. Menghitung jarak *Euclidean* antara titik terhadap semua centroid, kemudian menetapkan setiap data ke centroid yang terdekat.
d. Memperbarui centroid dengan cara menghitung rata-rata atau *mean* dari semua titik data yang tergabung dalam setiap klaster.
e. Ulangi langkah 1-4 hingga convergen.

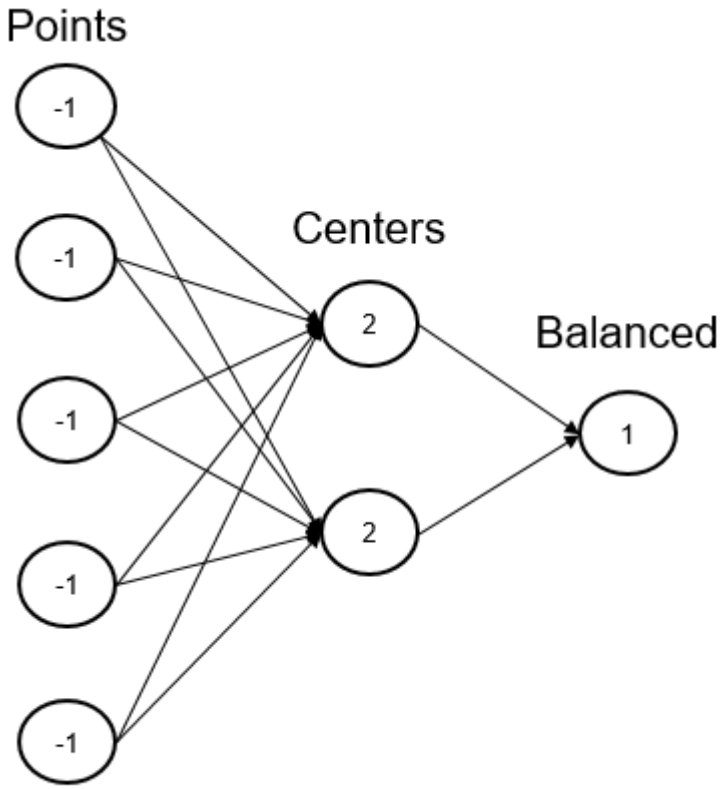
Keuntungan dari menggunakan k-means untuk mengelompokkan atau mengklaster data adalah metode ini menyediakan algortma yang sederhana, mudah untuk digunakan, dan cepat. Namun akan ada kemungkinan dimana klaster tidak memiliki anggota atau klaster memiliki sangat sedikit titik data. Untuk mengatasi hal ini maka dapat dilakukan teknik *balncing*, dimana anggota pada setiap klaster bisa seimbang. Salah satu teknik *balancing* yang dapat digunakan adalah dengan menggunakan *constraint*.

2.5 Constrained K-Means

Constrained k-means merupakan algoritma yang pada dasarnya terdiri dari k-means biasa yang dapat menjalankan k-means dengan jumlah minimum anggota klaster yang ditentukan sebelumnya (Bradley, Bennett, & Demiriz, 2000). Pada algoritma ini, proses menetapkan anggota menggunakan metode *Minimum Cost Flow* (MCF). Titik data adalah node dengan *supply*, centroid adalah node dengan *demand*, dimana *demand* merupakan jumlah minimum anggota pada setiap klaster, dan node tambahan untuk menyeimbangkan global sum dari *supply* dan *demand*. *Demand* pada centroid ditentukan dengan rumus sebagai berikut :

$$demand = \frac{\text{jumlah destinasi}}{\text{jumlah hari}} \qquad (2.2)$$

Model dari algoritma ini dapat dilihat pada Gambar 2. 2.



Gambar 2. 2 Model MCF Constrained K-Means

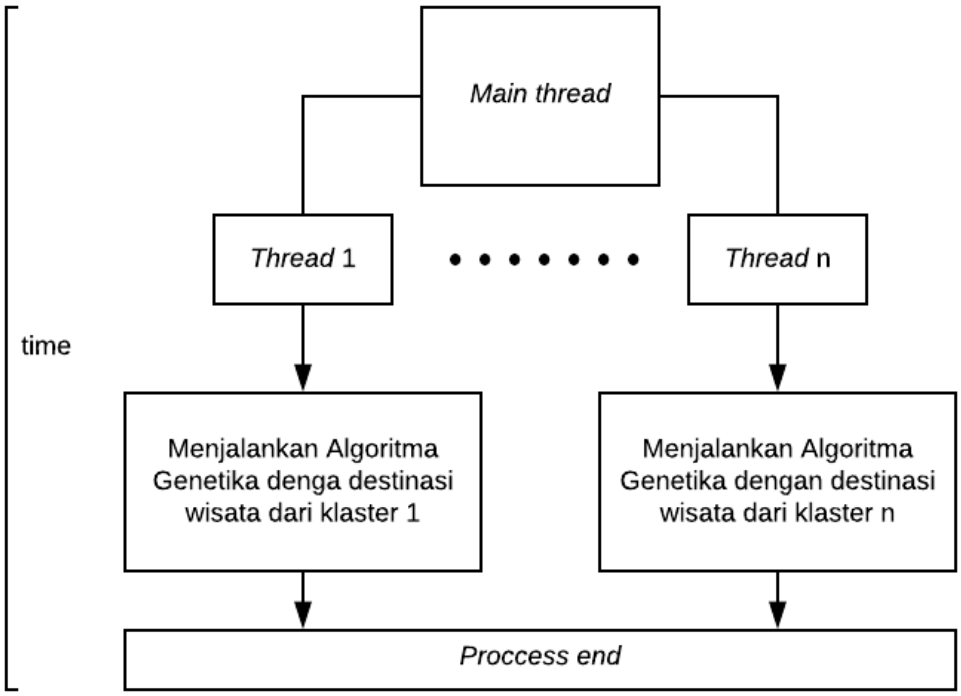
Points merupakan titik data, *centers* adalah centroid, angka di dalam lingkaran adalah *supply* (angka negatif) dan *demand* (angka positif), node *balanced* adalah node tambahan untuk menyeimbangkan *global sum* dari *supply* dan *demand*, dan garis yang menghubungkan antar node merupakan *arch*, dimana setiap *arch* memiliki nilai *cost.* Dengan asumsi jumlah data ≥ jumlah centroid, setiap titik data akan memiliki nilai *supply* -1, node centroid akan memiliki *demand* sesuai dengan jumlah minimum anggota pada setiap klaster, dan node tambahan akan memiliki *demand* sama dengan jumlah titik data dikurangi jumlah centroid. Anggota klaster akan ditetapkan berdasarkan nilai *cost* pada setiap *arch* yang menghubungkan node data dengan node controid dimana nilai *cost* adalah jarak *Euclidean* setiap titik data terhadap semua centroid. Sedangkan untuk *cost* pada *arch* yang menghubungkan node centroid dengan node tambahan atau node *balanced* bernilai 0.

2.6 Multithreading

Multithreading adalah kemampuan komputer atau *Central Proccessing Unit (CPU)* untuk dapat menjalankan program atau aplikasi dalam beberapa *thread* secara bersamaan atau

paralel. Keuntungan dari *multithreading* adalah dapat mengurangi waktu proses komputasi. Teknik ini akan sangat membantu untuk permasalahan optimasi tak terkecuali *Traveling Salesman Problem.* Seperti yang diketahui bahwa dengan semakin banyaknnya kota yang harus dikunjungi pada permasalahan TSP maka waktu proses komputasinya akan semakin lama. Dengan menjalankan TSP secara paralel, komputer akan lebih cepat untuk mendapatkan solusi dari TSP tersebut yaitu mana rute yang mempunyai jarak paling kecil. Implentasi multi*thread*ing pada penelitian ini dilakukan dengan cara membuat *thread* dengan jumlah yang sama dengan jumlah klaster, kemudian menjalankan program dalam setiap *thread*.

Pada Gambar 2. 3 terlihat bahwa program memiliki satu *thread* utama yaitu *main thread* dan ada 2 *thread* baru yaitu *thread* 1 dan *thread* 2. *Thread* 1 dan *thread* 2 akan menjalankan program dengan data yang berbeda secara bersamaan atau paralel.



Gambar 2. 3 Penerapan *Multithreading*

2.7 Usability Testing

Usability Testing adalah suatu analisa yang menentukan seberapa mudah pengguna dalam mengakses atau menggunakan suatu aplikasi. Suatu aplikasi dikatakan *usable* jika fungsi - fungsinya dapat dijalankan secara efektif, efisien, dan memuaskan (Nurhadryani, Sianturi, Hermadi, & Khotimah, 2013). Sesuai dengan standar ISO 9241-11, efektifitas adalah

keberhasilan pengguna dalam mencapai tujuan menggunakan suatu aplikasi atau perangkat lunak, efisiensi adalah kelancaran pengguna dalam mencapai tujuan tersebut, dan kepuasan berkaitan dengan sikap penerimaan pengguna terhadap aplikasi atau perangkat lunak. Berikut merupakan formula untuk mengukur index keberhasilan dari *Usability Testing* (Kurnia, 2019).

$$Index\ Keberhasilan = \frac{\text{Nilai Total Kuisioner}}{\text{Nilai Maksimum Kuisioner}} . 100\% \qquad (2.3)$$

Kategori keberhasilan :

Index Keberhasilan < 40% = Gagal

< 60% Index Keberhasilan ≥ 40% = Kurang Baik

< 80% Index Keberhasilan ≥ 60% = Cukup Baik

Index Keberhasilan > 80% = Baik

Hasil Bagi nilai total kuisioner dengan nilai maksimum kuisioner akan dikalikan dengan 100% untuk menghasilkan nilai presentasae. Nilai tersebut menjadi acuan untuk pengujian ini.

2.8 Black Box Testing

Black Box Testing merupakan pengujian pada perangkat lunak dimana struktur internal / desain / impelementasi dari subyek yang diuji tidak diketahui oleh *tester*. Pengujian ini dapat bersifat fungsional dan non-fungsional (Black Box Testing, 2019). Pengujian *Black Box* hanya berfokus pada masukan dan keluaran pada aplikasi yang diuji tanpa menghiraukan struktur internal aplikasi.

2.9 Pengujian Performa

Pengujian performa yang diakukan pada penelitian ini hanya berfokus kepada waktu pemrosesan sistem atau *Time Behavior*. *Time Behavior* merupakan kemampuan yang dimiliki perangkat lunak dala memberikan respon waktu pengolahan yang sesuai saat melakukan fungsinya (Nilamsari, 2014).

Analisis faktor kualitas performa sistem diukur dengan menghitung rata – rata *Response Time* yang digunakan untuk melakukan beberapa tugas pada sistem. *Response Time* merupakan ukuran perkiraan waktu untuk melakukan tugas yang diberikan pengguna ke sistem (Nilamsari,

2014). Tabel 2. 2 menunjukan *Response Time* dan *user rating* menurut Anna Bouch pada jurnal Guangzhu (2009).

Tabel 2. 2 *Response Time* dan *user rating* menurut Anna Bouch

Response Time	**Rating**
Lebih dari 2 detik	Sangat Bagus
2 – 5 detik	Bagus
6 – 10 detik	Biasa / rata – rata
Lebih dari 10 detik	Buruk