Algoritma *Branch and Bound* untuk *Driver Assignment* pada Aplikasi Simulasi Taksi *Online*

Arlynston Liadi, Andreas Handojo, Tanti Octavia
Program Studi Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
E-Mail: arlynston@gmail.com, handojo@petra.ac.id, tanti@petra.ac.id

ABSTRAK

Taksi *online* adalah salah satu perkembangan dalam teknologi yang saat ini tengah marak digunakan oleh masyarakat luas. Taksi *online* juga memberikan dampak yang positif seperti, *driver* dan *passenger* dapat saling mengetahui lokasi masing-masing dengan akurat, *passenger* dapat melihat *driver* dan informasi kendaraan secara akurat, dan *passenger* dapat dengan mudah mencari transportasi untuk bepergian ke tempat lain. Namun, dalam kenyataannya permintaan dari *passenger* muncul secara *random* pada lokasi yang berbeda-beda, sehingga diperlukan aplikasi yang dapat mengoptimalkan *assignment* antara *driver* dan *passenger*.

Aplikasi dibuat dengan bahasa pemrograman python, html, dan javascript , dengan django sebagai *framework*, bootstrap 3 untuk *front end menu*, google maps API untuk mendapatkan waktu dan jarak tempuh, serta sebagai *front end* pada bagian peta *online*, dan mySQL sebagai *database* yang digunakan.

Hasil akhir dari aplikasi ini adalah sebuah *assignment* untuk *driver* dan *passenger* menggunakan algoritma *branch and bound* berbasis *web* yang dapat meminimalkan biaya penjemputan dan waktu tunggu dengan memperhitungkan total kombinasi jarak tempuh, waktu tempuh, dan jumlah *trip driver* pada taksi *online*.

Kata Kunci: *Branch and bound, google maps api, static, dynamic,* taksi *online*

ABSTRACT

Online taxi is one of the developments in technology that is currently being used by the wider community. Online taxis also have positive impacts such as drivers and passengers can know each other's location accurately, passengers can see drivers and vehicle information accurately, and passengers can easily find transportation to travel to other places. However, in reality requests from passengers appear randomly in different locations, so we need an application that can optimize the assignment between drivers and passengers..

Applications are made with the python, html, and javascript programming languages, with django as a framework, bootstrap 3 for the front end menu, the Google maps API to get time and distance, as well as the front end in the online map section, and mySQL as the database used.

The end result of this application is an assignment for drivers and passengers using a web-based branch and bound algorithm that can minimize pickup costs and waiting times by calculating the total combination of mileage, travel time, and the number of trip drivers on an online taxi.

Keywords: *Branch and bound, google maps api, static, dynamic, online taxi*

1. PENDAHULUAN

Taksi *online* adalah salah satu perkembangan dalam teknologi yang saat ini tengah marak digunakan oleh masyarakat luas. Taksi *online* juga memberikan dampak yang positif seperti, *driver* dan *passenger* dapat saling mengetahui lokasi masing-masing dengan akurat, *passenger* dapat melihat *driver* dan informasi kendaraan secara akurat, dan *passenger* dapat dengan mudah mencari transportasi untuk bepergian ke tempat lain [5]. Namun, dalam kenyataannya permintaan dari *passenger* muncul secara *random* pada lokasi yang berbeda-beda, sehingga diperlukan proses *assignment* yang dapat mengoptimalkan *assignment* antara *driver* dan *passenger*. Simulasi pada taksi *online* bertujuan untuk meminimalkan jarak jemput *driver* dan waktu tunggu *passenger*. Minimalisasi ini dilakukan dengan memperhatikan faktor jarak tempuh, dan waktu tempuh dengan memperhitungkan kondisi lalu-lintas. Selain itu pada simulasi ini juga memperhitungkan faktor *trip* yang telah dilakukan *driver*, untuk membuat sistem semacam antrian, supaya jumlah *passenger* yang didapat oleh setiap *driver* tidak terpaut jauh. Dengan simulasi *assignment* semacam ini maka akan memberikan beberapa keuntungan seperti waktu tunggu *passenger* ketika penjemputan yang relatif lebih cepat, bensin yang dipakai *driver* untuk penjemputan lebih sedikit, dan membuat *trip* yang didapatkan oleh setiap *driver* menjadi tidak terpaut jauh.

Metode seperti *greedy heuristic* telah diusulkan untuk minimalisasi *cost* dalam *assignment*, seperti contohnya *TSP problem* [1]. *Metaheuristic algorithm* seperti *Tabu Search* (TS) dan *Simulated Annealing* (SA) juga diusulkan untuk *assignment* pada *Bus Driver* [2]. *Greedy heuristic* memiliki kelebihan pada waktu komputasi yang sangat cepat namun memiliki kelemahan pada *cost* yang tidak optimal. *Tabu search* dan *simulated annealinng* memiliki hasil *cost* yang optimal namun memiliki masalah pada waktu komputasi yang cenderung lama.

Algoritma *branch and bound* memiliki karakteristik dimana memunculkan hasil akhir yang memiliki *cost* minimal. Dalam simulasi ini *cost* yang dimaksud merupakan kombinasi dari variabel jarak tempuh, waktu tempuh, dan *trip driver*. Dalam menyelesaikan masalah yang serupa seperti misalnya *schedulling problem*, algoritma ini mampu menyelesaikannya secara efektif dan dalam waktu yang wajar [6]. Sehingga dengan *branch and bound* maka dapat ditemukan kombinasi antara *driver* dan *passenger* yang memiliki *cost* terkecil, dan akan mengelimininasi kombinasi dengan *cost* yang besar, sehingga minimalisasi jarak jemput dan waktu tunggu, serta sistem antrian pada *driver* dapat dilakukan.

2. DASAR TEORI

2.1. *Branch and Bound Algorithm*

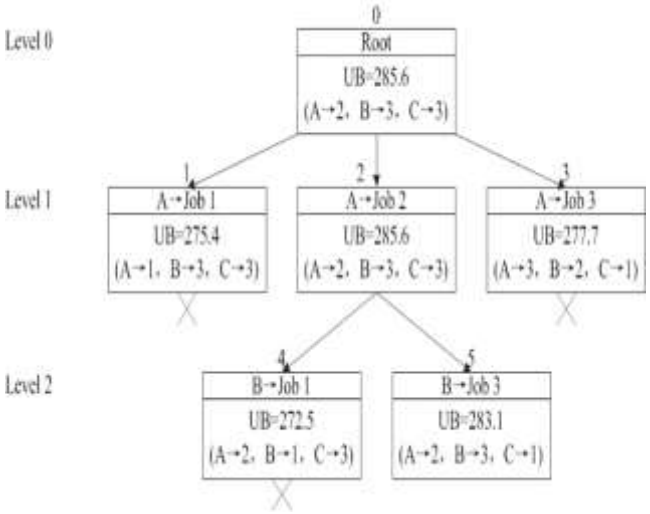
Branch and bound algorithm merupakan salah satu metode yang bertujuan untuk memecahkan masalah *combinatorial optimization*

yang pada umumnya seperti masalah pemasangan dari 2 objek dengan mencari *optimal cost*. *Branch and bound algorithm* dapat digunakan untuk mendapatkan solusi *optimal global* dengan waktu komputasi yang masih dapat diterima [3].

Contoh penyelesaian *assignment* dengan *Branch and Bound algorithm* adalah sebagai berikut, pertama-tama dibuat matrix baru yaitu matrix C [4]:

$$C = \begin{pmatrix} 84.41 & 93 & 88.4 \\ 86.05 & 96.8 & 97.6 \\ 94.08 & 86.4 & 95 \end{pmatrix}.$$

Pertama-tama, angka terkecil pada baris pertama akan diambil untuk menghitung *lower bound*. Kemudian angka itu dijumlahkan dengan angka terkecil dari semua baris di bawahnya untuk mendapat *lower bound*. Perhitungan ini dilakukan pada setiap *level* dan jika ada *lower bound* yang sama atau lebih besar dari sebelumnya, maka akan dieliminasi. Gambar 1 menunjukan visualisasi dari *state space tree* algoritma *branch and bound*.

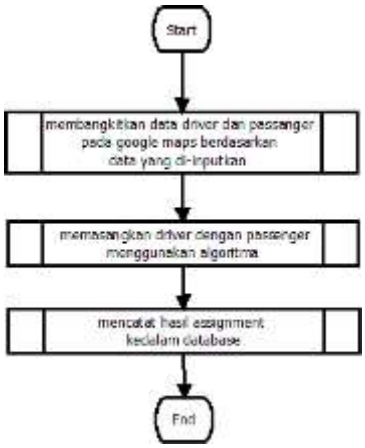


Gambar 1. *State space tree* [4]

3. DESAIN SISTEM

3.1 Desain *Flowchart*

Bagian ini berisi penjelasan tentang proses yang harus dilewati untuk mendapatkan hasil *assignment* antara *driver* dengan *passenger*. Setiap *predefined process* yang digambarkan pada gambar 2, akan dijabarkan dengan lebih terperinci di dalam sub-bab ini. *Predefined process* yang harus dijabarkan ada tiga, yang pertama adalah membangkitkan *data driver* dan *passanger* pada *google maps* berdasarkan *data* yang dimasukkan, yang kedua adalah memasangkan *driver* dengan *passenger* menggunakan algoritma, dan yang ketiga adalah mencatat hasil *assignment* ke dalam *database*. *Flowchart* proses digambarkan seperti pada Gambar 2.



Gambar 2. *Flowchart assignment driver* dengan *passenger*

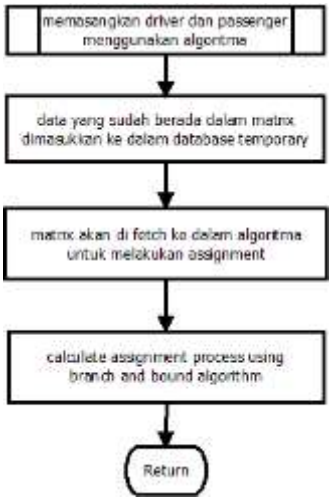
3.1.1 Proses Membangkitkan Data Driver dan Passanger pada Google Maps Berdasarkan Data yang dimasukkan

Bagian ini merupakan penjabaran dari *predefined process* pertama, yaitu membangkitkan *data driver* dan *passanger* pada *google maps* berdasarkan data yang dimasukkan. Proses ini dimulai dengan *user* yang melakukan *input data*. *Data* yang sudah dimasukkan oleh *user* akan dijalankan oleh *google maps api* untuk membangkitkan *data driver* dan *passenger*. Data yang sudah dibangkitkan akan diambil nilai jarak dan waktu dari setiap kombinasi *driver* dan *passenger*. Nilai jarak dan waktu yang telah diambil akan dilakukan normalisasi dengan *range* 1-10. Nilai jarak dan waktu yang telah di normalisasi kemudian akan dijumlahkan untuk setiap kombinasi *driver* dengan *passenger*. Penjumlahan nilai ini akan menghasilkan matriks yang akan dipakai untuk melakukan proses *assignment*. *Flowchart* proses digambarkan seperti pada Gambar 3.

Gambar 3. *Flowchart* membangkitkan *data driver* dan *passenger* pada *google maps* berdasarkan *data* yang dimasukkan

3.1.2 Proses Memasangkan Driver dengan Passenger Menggunakan Algoritma Branch and Bound

Bagian ini merupakan penjabaran dari *predefined process* kedua, yaitu memasangkan *driver* dengan *passenger* menggunakan algoritma *branch and bound* yang sudah dibahas pada bab sebelumnya. Matriks yang telah dihasilkan pada proses sebelumnya akan disimpan sementara dalam *database*. Matriks yang telah tersimpan dalam *database* akan diambil oleh algoritma *branch and bound* untuk melakukan *assignment*. Matriks akan dieksekusi oleh algoritma *branch and bound*, maka akan menghasilkan solusi berupa pemasangan *driver* dengan *passenger*. *Flowchart* proses digambarkan seperti pada Gambar 4.



Gambar 4. *Flowchart* memasangkan *driver* dengan *passenger* menggunakan algoritma *branch and bound*

3.1.3 Proses Mencatat Hasil Assignment ke Dalam Database

Bagian ini merupakan penjabaran dari *predefined process* terakhir, yaitu mencatat hasil *assignment* ke dalam *database*. *Flowchart* proses digambarkan seperti pada Gambar 5.

Pada proses ini, solusi yang dihasilkan sebelumnya oleh algoritma *branch and bound* akan disimpan dalam tabel *assignment* pada *database*. *Status* dari *driver* yang pada tabel *driver*, dan status *passenger* yang berada pada tabel *passenger* akan diganti dari yang sebelumnya bernilai nol menjadi satu.

Gambar 5. *Flowchart* mencatat hasil *assignment* ke dalam *database*

4. IMPLEMENTASI SISTEM

Implementasi program dibuat berdasarkan rancangan desain sistem yang sudah dilakukan sebelumnya. Implementasi program dilakukan dengan menggunakan bahasa pemrograman *html*, *javascript*, dan *python* dan *framework django*. Simulasi ini menggunakan 2 *folder* yaitu: *folder templates* dan *mywebiste*. *Folder templates* menampung semua *file html* yang digunakan dalam simulasi ini. *File html* ini digunakan untuk melakukan semua proses yang sudah dijabarkan pada *usecase* di bab sebelumnya. Sedangkan *Folder mywebsite* menampung semua *file python* dan juga url untuk menjalankan *file python*. *File python* juga digunakan untuk memproses *data* dari *database*, melakukan kalkulasi dan menjalankan *file html*.

5. PENGUJIAN

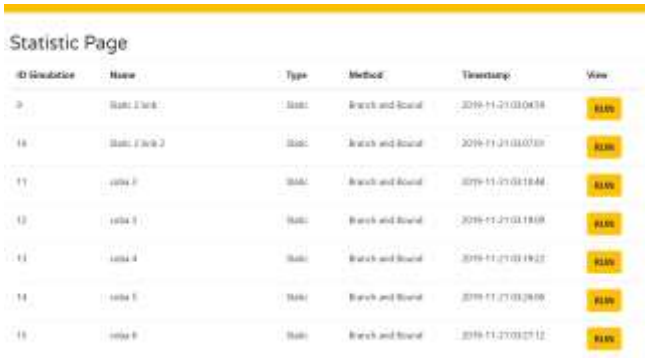
5.1 Pengujian Sistem pada Halaman *Home*

Pada saat pertama kali *user* membuka aplikasi, maka akan dihadapkan dengan halaman *home*. Pada halaman ini, *user* diminta untuk memilih tipe simulasi yang diingikan untuk dijalankan sebelum masuk ke halaman *insert data* seperti pada Gambar 6. Setelah masuk ke halaman *insert data*, maka simulasi dapat dijalankan. Simulasi yang bisa dijalankan adalah *static*, *dynamic* berdasarkan jumlah *batch*, dan *dynamic* berdasarkan waktu.

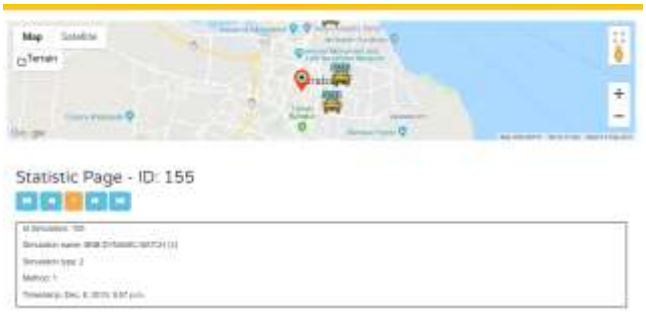
Gambar 6. Tampilan halaman *home*

5.2 Pengujian Sistem pada Halaman *Statistic*

Statistik dari simulasi diperlihatkan setelah *user* memilih simulasi yang ingin ditampilkan seperti pada Gambar 7. Hasil statistik dari simulasi akan ditampilkan seperti pada Gambar 8, Gambar 9, dan Gambar 10.



Gambar 7. Tampilan halaman *statistic*



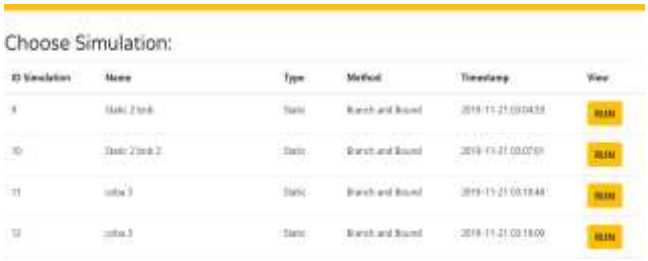
Gambar 8. Tampilan data *statistic* (1)

Gambar 9. Tampilan data *statistic* (2)

Gambar 10. Tampilan data *statistic* (3)

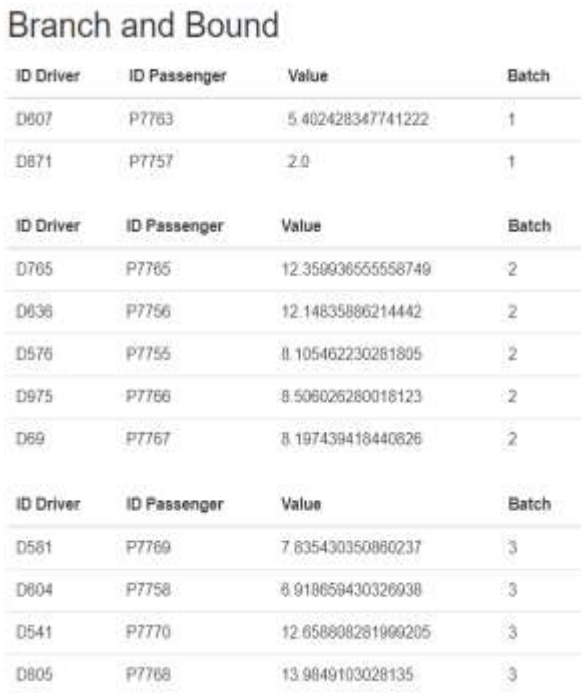
5.3 Pengujian Sistem pada Halaman *Comparison All*

Halaman *comparison all* bertujuan untuk membandingkan hasil dari beberapa algoritma dengan *data* yang sama. user harus memilih simulasi yang ingin dibandingkan datanya seperti pada Gambar 11. *Data* perbandingan akan ditampilkan dalam bentuk tabel seperti pada Gambar 12 dan Gambar 13. data *assignment* akan ditampilkan oleh setiap algoritma secara urut berdasarkan *batch*.



Gambar 11. Tampilan halaman *comparison all* (1)

Gambar 12. Tampilan halaman *comparison all* (2)



Gambar 13. Tampilan data algoritma *branch and bound*

6. KESIMPULAN

Dari hasil perancangan dan pembuatan aplikasi, dapat diambil kesimpulan antara lain:

- Aplikasi dapat menjalankan simulasi *static* dan *dynamic* dengan inputan dari *user*.
- Aplikasi memberikan informasi *driver* yang muncul, *passenger* yang muncul, nilai normalisasinya, dan hasil *assignment* baik *static* atau *dynamic*.
- Aplikasi memberikan informasi total jarak dan waktu, rata- rata jarak dan waktu, dan waktu kalkulasi
- Hasil perhitungan yang didapatkan oleh algoritma *branch and bound* lebih baik jika dibandingkan dengan algoritma *tabu search*.
- Waktu komputasi dari algoritma *branch and bound* mengikuti dari besarnya *data* yang di inputkan, semakin besar *data* maka semakin lama proses perhitungannya berjalan.
- Algoritma *branch and bound* memiliki waktu kalkulasi yang terbesar dibandingkan dengan algoritma lain seperti *tabu search* dan *hungarian algorithm*.

7. DAFTAR PUSTAKA

[1] Çal, M., & Ekici, A. 2018. Solving a Modified TSP Problem by a Greedy Heuristic for Cost Minimization. *International Journal of Modeling and Optimization*, 8(3), 138–144. https://doi.org/10.7763/ijmo.2018.v8.638

[2] Constantino, A. A., de Mendonça Neto, C. F. X., de Araujo, S. A., Landa-Silva, D., Calvi, R., & dos Santos, A. F. 2017. Solving a large real-world bus driver scheduling problem with a multi-assignment based heuristic algorithm. *Journal of Universal Computer Science*, 23(5), 479–504.

[3] Costa, L. H. M., de Athayde Prata, B., Ramos, H. M., & de Castro, M. A. H. 2016. A Branch-and-Bound Algorithm for Optimal Pump Scheduling in Water Distribution Networks. *Water Resources Management*, 30(3), 1037–1052. https://doi.org/10.1007/s11269-015-1209-2

[4] Ding, S., & Zeng, X. J. 2018. Uncertain random assignment problem. *Applied Mathematical Modelling*, 56, 96–104. https://doi.org/10.1016/j.apm.2017.11.026

[5] Silalahi, S. L. B., Handayani, P. W., & Munajat, Q. 2017. Service Quality Analysis for Online Transportation Services: Case Study of GO-JEK. In Procedia Computer Science (Vol. 124, pp. 487–495). Elsevier B.V. https://doi.org/10.1016/j.procs.2017.12.181

[6] Toumi, S., Jarboui, B., Eddaly, M., & Rebaï, A. 2017. Branch-and-bound algorithm for solving blocking flowshop scheduling problems with makespan criterion. *In International Journal of Mathematics in Operational Research (Vol. 10, pp. 34–48)*. Inderscience Enterprises Ltd. https://doi.org/10.1504/IJMOR.2017.080743