Analisa Kompleksitas Algoritma A*

Tanur Rizaldi Rahardjo / 13519214
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13519214@stei.itb.ac.id

***Abstract*—A* Algorithm are pathfinding algorithm. Pathfinding algorithm are used for searching shortest path from given a graph and two nodes which act as starting and endpoint. A* is one of commonly used pathfinding algorithm. A* Algorithm itself having many variation on implementation, some of them are targeted to cut time complexity and other are optimizing for others aspect. As pathfinding is having generally high time complexity, optimization for lower time complexity is needed.**

***Keywords*—Algorithm, Astar, Complexity.**

I. PENDAHULUAN

Travelling salesman problem adalah salah satu masalah yang sering dibahas dalam *computer science.* Sederhananya *travelling salesman problem* adalah permasalahan untuk mencari jalur terpendek dalam suatu graf. Graf ini dapat menjadi suatu abstraksi mewakili suatu kota dan jalan misalnya, namun juga dapat mewakili jaringan listrik, *networking* komputer dan lain-lain.

Terdapat beberapa algoritma yang umum digunakan pada permasalahan ini contohnya algoritma *Dijkstra* dan A*. Ada 2 teknik pencarian yang sering digunakan yaitu *blind search* dan *heuristic search*. Teknik *heuristik* umumnya memberikan waktu yang lebih cepat dibandingkan *blind.* Namun teknik *heuristik* cenderung lebih sulit dipahami dibandingkan teknik *blind*.

Algoritma A* mencari rute yang optimal dan salah satu algoritma *pathfinding* yang komplit. Rute optimal berarti rute tersebut memiliki jarak atau beban terpendek dibandingkan rute lain tetapi tidak dijamin rute yang paling baik dan komplit memiliki arti rute tersebut menghubungkan simpul *starting* dan simpul *endpoint*. Algoritma ini didasarkan pada algoritma A yang dijelaskan oleh Peter Hart, Nils Nilsson, dan Bertram Raphael pada tahun 1968. Algoritma A dioptimisasi *heuristik* dan hasil optimisasi tersebut dinamai algoritma A*. Dalam penerapan, algoritma A* menggunakan *weight* pada *weighted graph* untuk proses menghitung jalur yang optimal. Algoritma A* umumnya lebih cepat daripada algoritma *Dijkstra* dengan selisih waktu rata-rata 40 ms.

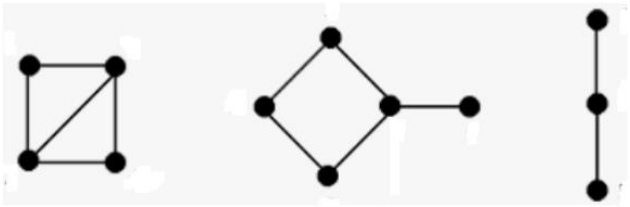
II. TEORI DASAR

2.1 Graf

Graf didefinisikan sebagai suatu pasang himpunan (*V, E*), dengan *V* adalah himpunan tidak kosong dari simpul dan *E* adalah himpunan dari sisi yang menghubungkan dua simpul. Atau secara non-formal dapat disebutkan sebagai kumpulan suatu simpul (*node*) dan sisi (*edge*) yang menghubungkan antara dua buah simpul. Berdasarkan definisi yang umum, graf dapat digolongkan menjadi beberapa golongan untuk mempersempit graf mana yang dimaksud. Pengolongan tersebut menjadi terminologi yang sering digunakan ketika bekerja dengan graf, berikut adalah beberapa terminologi pada graf :

1. Graf sederhana (*Simple graph*)

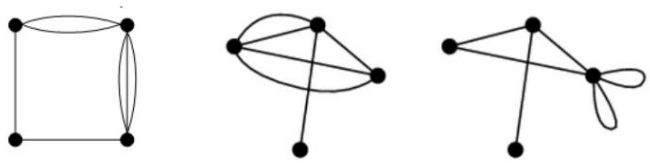
Graf yang tidak mengandung gelang atau *loop* dan tidak ada dua simpul terhubung dengan sisi yang lebih dari satu.



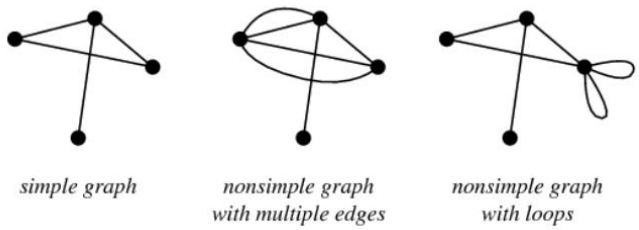
Gambar 2.1 Graf sederhana

2. Graf tak-sederhana (*Unsimple-graph*)

Graf yang mengandung gelang atau *loop* atau terdapat dua simpul yang terhubung dengan sisi yang lebih dari satu. Graf tak-sederhana masih dapat dibedakan lagi menjadi beberapa golongan.



Gambar 2.2 Graf tak-sederhana

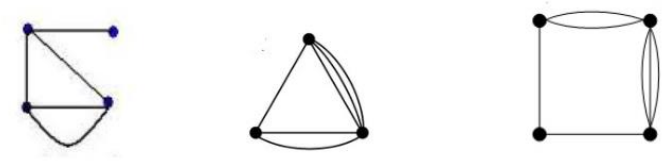


Gambar 2.3 Penggolongan graf tak-sederhana

3. Graf ganda (*Multi-graph*)

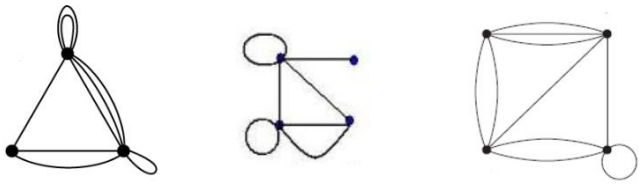
Graf tak-sederhana yang mengandung dua simpul yang

terhubung dengan sisi yang lebih dari satu.



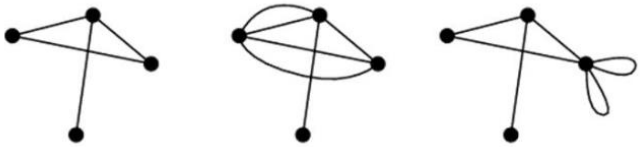
Gambar 2.4 Graf ganda

4. Graf semu (*Pseudo-graph*)
Graf tak-sederhana yang memiliki sisi gelang atau *loop*.



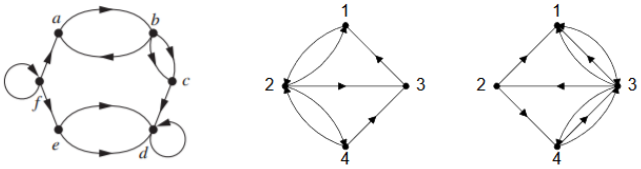
Gambar 2.5 Graf semu

5. Graf tak-berarah (*Undirected graph*)
Graf yang sisinya tidak memiliki orientasi arah.



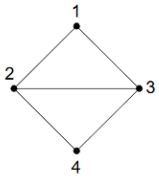
Gambar 2.6 Graf tak-berarah

6. Graf berarah (*Directed graph* atau *Digraph*)
Graf yang sisinya memiliki orientasi arah.



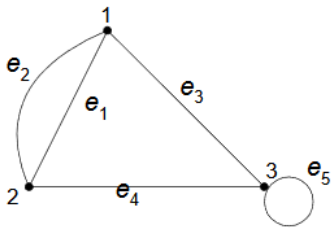
Gambar 2.7 Graf berarah

7. Ketetanggan (*Adjacent*)
Dua simpul disebut bertetangga jika kedua simpul terhubung dengan suatu sisi secara langsung. Pada gambar 2.8, simpul 1 dan 4 tidak bertetangga sedangkan 2 dan 3 bertetangga.



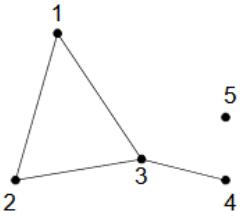
Gambar 2.8 Ketetanggan

8. Bersisian (*Incidency*)
Sisi disebut bersisian dengan suatu simpul jika salah satu ujung dari sisi merupakan simpul tersebut. Pada gambar 2.9, sisi e_1 bersisian dengan simpul 1 atau simpul 2 tetapi tidak dengan simpul 3.



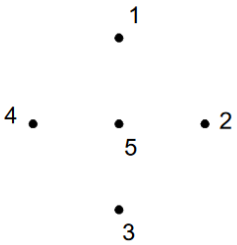
Gambar 2.9 Bersisian

9. Simpul Terpencil (*Isolated Vertex*)
Simpul terpencil merupakan simpul yang tidak mempunyai sisi yang menghubungkan dengan simpul lain. Pada gambar 2.10, simpul 5 merupakan simpul terpencil atau *isolated vertex*.



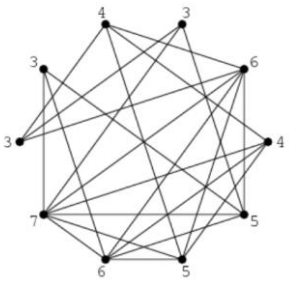
Gambar 2.10 Simpul terpencil

10. Graf Kosong (*Null Graph* atau *Empty Graph*)
Graf yang himpunan sisinya adalah himpunan kosong.



Gambar 2.11 Graf kosong

11. Derajat (*Degree*)
Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul. Gambar 2.12 menunjukan banyaknya derajat setiap simpul.



Gambar 2.12 Derajat setiap simpul

Teori graf adalah topik dasar pada matematika diskrit dan *computer science*. Teori ini cenderung abstrak sehingga dapat merepresentasikan banyak objek – objek diskrit dan hubungan antar objek tersebut. Abstraksi tersebut juga memberikan banyak cara dalam mereperesentasikan suatu graf, misalnya graf dapat diwakilkan sebagai matriks atau gambar dengan titik atau

bulatan sebagai simpul dan garis sebagai sisi.

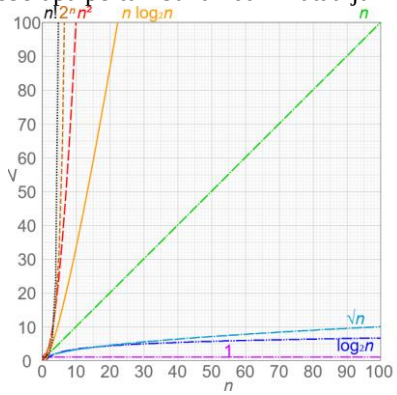
2.2 Kompleksitas Algoritma

Kompleksitas algoritma juga merupakan salah satu topik umum yang sering dibahas pada matematika diskrit dan *computer science*. Pada proses pendesainan suatu algoritma, banyak sekali cara untuk melakukan suatu hal, meskipun berbeda dalam cara pengerjaannya hasil akhirnya tetap sama. Cara pengerjaan yang berbeda tersebutlah yang menyebabkan topik terhadap kompleksitas algoritma muncul.

Cara-cara tersebut dapat dibedakan dengan seberapa kompleks kebutuhan waktu dan ruang dalam pengerjaannya. Pada analisa kompleksitas, ruang umumnya merunjuk kepada memori yang digunakan, meskipun pengertian ruang disini dapat diambil secara harfiah jika ingin menganalisa hal yang sangat besar. Sekarang permasalahan tentang kompleksitas ruang tidak terlalu dipikirkan karena penggunaan ruang memori umumnya tidak terlalu banyak, sehingga analisa kompleksitas waktu cenderung lebih ditekankan daripada ruang.

Umumnya kompleksitas waktu dan ruang suatu algoritma bergantung kepada ukuran masukan yang ada. Tujuan utama pada kompleksitas waktu adalah menghitung banyaknya jumlah operasi atau komputasi dalam suatu algoritma. Pada perhitungan dapat disederhanakan dengan hanya menghitung operasi yang penting dalam algoritma dan mengabaikan operasi yang tidak terlalu berdampak pada tujuan asli algoritma. Contoh-contoh operasi yang umumnya dihitung yaitu operasi airtmetika, pengisian nilai (*assignment*), perbandingan, dan pemanggilan fungsi atau prosedur.

Pada kompleksitas waktu sering digunakan *Big-O Notation* yang mewakili batas atas waktu asimptotik suatu algoritma. Notasi tersebut mempermudah dalam membandingkan antara algoritma satu dan algoritma lain. Gambar 2.13 menunjukan beberapa pertumbuhan dari n atau jumlah input.



Gambar 2.13 Kompleksitas waktu asimtotik

2.3 Algoritma A*

Algoritma A* merupakan algoritma *pathfinding* yang sering digunakan untuk *pathfinding* dan *graph traversal*. Algoritma A* memiliki beberapa optimisasi *heuristic* yang digunakan untuk mencapai hasil yang lebih baik. Algoritma A* menggunakan *Best First Search* untuk menemukan jalur dengan jumlah *weight* terkecil dari simpul awal (*Starting Node*) ke simpul akhir (*Endpoint*). Algoritma ini menggunakan fungsi *heuristik* jarak ditambah dengan *weight* setiap node untuk menentukan urutan pencarian. Algoritma ini menggunakan dua *list* yang biasanya dinamai *open* dan *closed*. *List open* menyimpan simpul-simpul yang pernah dicek tetapi belum terpilih sebagai simpul terbaik dan *closed* menyimpan simpul yang telah terpilih menjadi simpul terbaik. Beberapa terminologi yang digunakan pada algoritma A* adalah :

1. f(n) = Estimasi *weight* terendah
2. g(n) = *Weight* dari simpul awal ke simpul n
3. h(n) = Perkiraan *weight* dari simpul n ke simpul akhir atau fungsi *heuristik.*

III. Penerapan

Permasalahan *travelling salesman* sendiri memiliki kompleksitas waktu yang sangat besar jika digunakan algoritma *brute force* untuk mencari solusi eksak atau jalur optimal global. Dalam notasi *Big-O* algoritma *brute force* dapat mencapai *O(n!)*. Karena kompleksitas waktu untuk algoritma *brute force* sangatlah tidak optimal untuk kasus n yang besar, digunakanlah algoritma yang tidak eksak namun optimal seperti algoritma A* untuk mencapai waktu yang lebih cepat.

Algoritma A* merupakan algoritma yang tergolong sering digunakan dikarenakan kompleksitas waktu yang cenderung lebih kecil daripada algoritma *pathfinding* yang lain. Ketika algoritma A* melewati graf, algoritma ini mengikuti jalur pertama yang terendah *weight*-nya dan menyusun antrian jalan alternatif terendah lain. Karena algoritma ini bergantung pada *heuristik*, kompleksitas waktu sangat bergantung pada *heuristik* yang digunakan. Algoritma A* secara umum memiliki kompleksitas waktu $O(b^d)$ dengan b adalah faktor percabangan dari graf dan d adalah jarak terpendek dari jalur penghubung simpul awal dan akhir. Untuk *heuristik* yang baik dan graf yang digunakan adalah pohon, kompleksitas waktu dari algoritma A* adalah *O(*log h**(x))* dengan h* adalah fungsi *heuristik* yang optimal.

IV. Kesimpulan

Algoritma A* bukanlah algoritma eksak untuk mencari jalur terpendek dalam suatu graf. Namun pada *software enginnering* dan *enginnering* secara general, selalu ada *trade-off* ketika menyelesaikan suatu masalah. Pada kasus ini ditukar antara solusi eksak dari algoritma *brute force* dengan kompleksitas waktu yang ditawarkan oleh algoritma A*. Solusi yang diberikan oleh algoritma A* tergolong optimal untuk komplektitas waktu yang jauh lebih cepat dibandingkan *brute force* untuk kasus n yang besar.

Referensi

[1] Ida Bagus G. W. A. D., “Penerapan Algoritma A* (Star) Menggunakan Graph untuk Menghitung Jarak Terpendek”.

[2] Rinaldi Munir, http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/matdis20-21.htm

[3] *Cmglee*, Graphs of number of operations, N vs input size, n for common complexities, assuming a coefficient of 1.

[4] Leo Willyanto Santoso, Alexander Setiawan, Andre K. Prajogo, “Performance Analysis of Dijkstra, A* and Ant Algorithm for Finding Optimal Path Case Study: Surabaya City Map “

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Desember 2020

Tanur Rizaldi Rahardjo
13519214