JISKa, Vol. 5, No. 3, NOVEMBER, 2020, Pp. 194 – 205
ISSN: 2527 – 5836 (print) | 2528 – 0074 (online)

Komparasi Kinerja Algoritma BFS, Dijkstra, Greedy BFS, dan A* dalam Melakukan *Pathfinding*

Nadila Sugianti [(1)*], Ainatul Mardhiyah [(2)], Nurma Romihim Fadlilah [(3)]
[1,2,3] Teknik Informatika, Fakultas Sains dan Teknologi, UIN Maulana Malik Ibrahim, Malang
e-mail : {17660036,18650017}@student.uin-malang.ac.id, ainadimana@gmail.com.
* Penulis korespondensi.
Artikel ini diajukan 20 Mei 2020, direvisi 4 Agustus 2020, diterima 5 Agustus 2020, dan dipublikasikan 9 November 2020.

Abstract

*Pathfinding is a computational process in finding the best route between two points or nodes to find the shortest path. This method has many algorithms that can be applied in various fields. In carrying out the pathfinding, speed and distance are considered as important. Through the test diagram, this paper illustrates the execution steps related to the pathfinding algorithm which includes BFS, Dijkstra, Greedy BFS, and A * for comparison. From several studies, the authors identified that execution time and mileage can be used optimally in the comparison process. Input variables as well as media used are 2-dimensional grids and heuristic function calculations. The analogy is carried out on a unity platform with the C# programming language, producing A * as a more flexible pathfinding algorithm to be implemented in various domains.*

Keywords: Pathfinding, Breadth-First Search, Dijkstra, Greedy Best-First Search, A*

Abstrak

Pathfinding merupakan proses komputasi dalam menemukan rute terbaik di antara dua titik atau *node* untuk menemukan *shortest path*. Metode ini memiliki banyak algoritma yang dapat diterapkan di berbagai bidang. Dalam melakukan *pathfinding* kecepatan dan jarak dipertimbangkan sebagai hal yang penting. Melalui diagram pengujian, paper ini menggambarkan langkah-langkah eksekusi terkait algoritma *pathfinding* yang meliputi BFS, Dijkstra, Greedy BFS, dan A* untuk dibandingkan. Dari beberapa penelitian, penulis mengidentifikasi bahwa waktu eksekusi dan jarak tempuh dapat digunakan secara optimal dalam proses komparasi. Variabel input sekaligus media yang digunakan ialah *grid* 2 dimensi serta kalkulasi fungsi *heuristic*. Analogi dilakukan pada platform *unity* dengan bahasa pemrograman C#, menghasilkan A* sebagai algoritma *pathfinding* yang lebih fleksibel untuk diimplementasikan dalam berbagai domain.

Kata Kunci: *Pathfinding, Breadth-First Search, Dijkstra, Greedy Best-First Search, A**

1. PENDAHULUAN

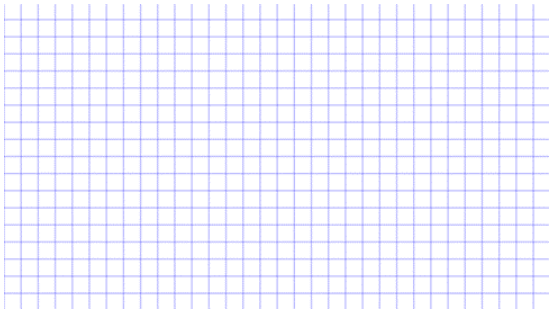
Teori *pathfinding* menggambarkan proses pencarian rute dari titik awal ke titik akhir pada lokasi tertentu. Dalam kebanyakan kasus, tujuan dari *pathfinding* adalah untuk menemukan *shortest path* distingtif yang optimal. *Shortest path* sendiri adalah komputerisasi oleh algoritma pencarian graf, yaitu mengidentifikasi rute optimal dengan biaya minimum dari titik awal ke titik tujuan. Optimal di sini berarti meminimalkan waktu dan memori yang digunakan dalam pencarian rute. Secara umum, solusi *shortest path* dibuat dalam graf terhubung, di mana berbagai simpul atau *node* terhubung melalui garis direksional atau nondireksional (Zhao & Zhao, 2017).

Aplikasi *shortest path* sering diterapkan dalam kehidupan nyata, misalnya dalam *map software* atau sistem navigasi, hal ini melibatkan masalah pencarian rute dari satu titik lokasi ke titik lainnya, dan jalur untuk mengeluarkan biaya minimum akan menjadi rute terpendek. Kasus manajemen transportasi dan desain teknik juga berhubungan erat dengan masalah *shortest path* dalam beberapa hal seperti untuk mengatur berbagai rute proses, penempatan jaringan listrik atau pipa, akuisisi rute dalam peta listrik, dan lain-lain (Zhao & Zhao, 2017). *Pathfinding* memiliki banyak algoritma yang dapat diterapkan pada bidang spesifiknya masing-masing Meskipun demikian, masih ada juga jenis algoritma yang dapat diterapkan secara umum dengan hasil memuaskan. Tetapi tidak selalu jelas apa kelebihan algoritma tertentu dibandingkan dengan alternatifnya.

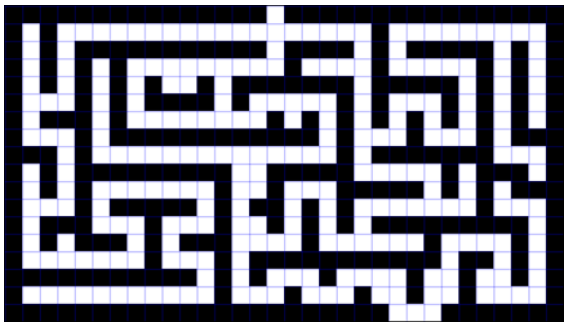
Beberapa algoritma dari metode *pathfinding* yaitu A*, *best first search* (BFS), algoritma Dijkstra, dan *Greedy Best First Search* akan diimplementasi untuk menganalisis efisiensinya dalam *environment* yang didasarkan pada *grid* dua dimensi. Faktor-faktor seperti ukuran dan jenis *grid*, jarak yang ditempuh, dan waktu eksekusi akan dipertimbangkan melalui serangkaian percobaan. Pada percobaan *grid* dua dimensi ini, setiap algoritma akan mencari jalur terpendek di antara dua *node* yang ditempatkan secara acak lalu waktu eksekusi, dan jarak tempuh akan diukur untuk mencari kelebihan dan kekurangan dari algoritma-algoritma tersebut.

2. METODE PENELITIAN

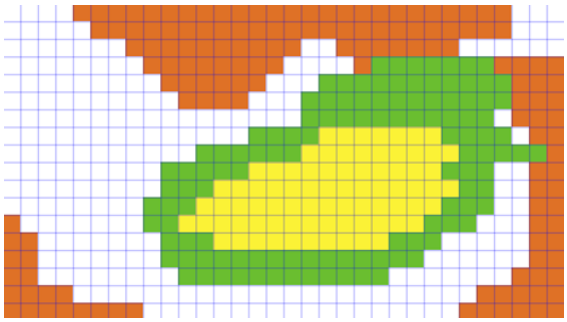
Perbandingan setiap algoritma dilakukan di atas *grid* dua dimensi yang dibuat menggunakan Aseprite. Pengujian ini menggunakan empat macam *grid* yang dikelompokkan sebagai lebih sederhana dan lebih kompleks. *Grid* yang lebih sederhana terdiri dari *empty grid* yang hanya memiliki satu macam *node* yakni *open node* serta *open walls* di mana pada *grid* terdapat dua macam *node* yaitu *open node* dan *blocked node*. Sedangkan *grid* yang lebih kompleks terdiri dari *maze* dan *terrain*. Untuk menilai efisiensi algoritma, percobaan juga dilakukan dengan berbagai ukuran. Ukuran *grid* yang digunakan terbagi menjadi skala kecil dan skala besar. Pada skala kecil ukuran *grid* yaitu 32x18 dan 64x36 *node.* Sedangkan *grid* yang digunakan pada skala besar berukuran 128x72 dan 256x144 *node*. Gambar 1 sampai 3 adalah beberapa sampel dari *grid* dua dimensi yang digunakan.



Gambar 1. Empty Grid 32x18 Node



Gambar 2. *Maze* 32x18 *Node*



Gambar 3. *Terrain* 32x18 *Node*

Dalam penerapannya beberapa algoritma juga menggunakan fungsi *heuristic*. Fungsi *heuristic* yang digunakan dalam percobaan ini adalah *manhattan distance* karena jaraknya yang berdasarkan pada *grid*.

$$H = |x_1 - x_2| + |y_1 - y_2| \quad (1)$$

Setiap percobaan pada masing-masing jenis dan ukuran *grid* diulang sebanyak 10 kali lalu dirata-rata untuk memastikan keakuratan hasilnya. Pada seluruh jenis dan ukuran *grid, node* awal serta *node* tujuan akan diletakkan ditempat yang sama untuk masing-masing algoritma agar mendapatkan hasil penelitian yang adil. Algoritma yang diimplementasikan dapat melakukan *pathfinding* secara horizontal, vertikal, maupun diagonal. Setiap transisi dua *node* yang bersebelahan secara horizontal dan vertikal memiliki biaya 1 dan untuk yang berdekatan secara diagonal memiliki biaya 1.4. Sedangkan pada *terrain, node* terdiri dari empat macam yaitu, *open node, light terrain, medium terrain,* dan *heavy terrain.* Secara berurutan, *node terrain* memiliki biaya tambahan sebesar 2, 3, dan 4.

Program yang digunakan untuk membandingkan algoritma *pathfinding* ini dirancang menggunkaan Unity dengan bahasa pemrograman C#, dan semua percobaan dilakukan pada komputer dengan CPU yang berjalan pada frekuensi1,60 Ghz.

2.1. *Breadth-First Search* (BFS)

Breadth-first search (BFS) adalah algoritma pencarian graf yang paling sederhana dan merupakan dasar dari beberapa algoritma yang lebih maju. Algoritma Prim dari *minimal spanning tree* dan Dijkstra dengan *single source* menggunakan prinsip yang mirip dengan BFS. BFS sangat berguna dalam menemukan jalur terpendek pada graf yang tidak memiliki *weight* atau biaya (Cormen et al., 2009).

Dengan graf *G = (V, E)* di mana *V* adalah *vertex* atau *node* dan *E* adalah *edges* atau jalur dan *node* awal yaitu *s*, BFS akan secara sistematis melintasi jalur *G* untuk menemukan setiap *node* yang dapat dijangkau dari *node s.* Algoritma ini menghitung jarak (jumlah jalur terkecil) dari *node s* ke setiap *node* yang dapat dicapai dan menghasilkan "*breadth-first tree*" dengan akar *s* yang semua jalurnya dapat dijangkau. Setiap jalur *v* yang dapat dicapai dari *node s* pada *breadth-first tree* akan menciptakan rute terpendek dari *s* ke *v*, yaitu rute yang memiliki jumlah jalur terkecil. Algoritma ini dapat digunakan dalam graf berarah maupun graf tidak berarah.

Untuk mempermudah proses traversal, algoritma BFS menggunakan struktur data *queue* dalam menyimpan dan menandai *node* yang telah dikunjungi. Hal ini akan terus berlangsung hingga semua *node* yang berdekatan telah ditandai. Metode *queue* yang digunakan ialah *first in first out* (FIFO).

Untuk melacak proses traversal dari BFS setiap *node* akan diberi warna putih, biru, dan abu-abu. Semua *node* akan dimulai dengan warna putih. *Node* yang masuk ke dalam *queue* dan akan dikunjungi berubah menjadi biru sedangkan *node* abu-abu dianggap telah dikunjungi. BFS mengurutkan *node-node* tersebut untuk memastikan pencarian brelangsung secara *breadth-first*.

2.2. Dijkstra

Algoritma Dijkstra merupakan salah satu algoritma yang sering dipakai dalam pencarian rute terpendek karena mudah digunakan. Dalam mencari solusi, algoritma Dijkstra menggunakan prinsip *greedy*, yaitu untuk menemukan solusi optimal pada setiap langkahnya agar bisa mendapatkan solusi optimal untuk langkah selanjutnya yang mengarah pada pilihan terbaik. Hal ini membuat kompleksitas waktu eksekusi algoritma Dijkstra menjadi cukup besar, yaitu $(v * \log(v + e))$ di mana *v* adalah *node* dan *e* adalah sisi yang menghubungkan *node* (Risald et al., 2017).

Algoritma Dijkstra selalu menghitung seberapa jauh proses traversal setiap mencapai suatu *node.* Algoritma ini juga membangun jalur minimal dari *node* awal (*s*) ke *node* lainnya (*v*),

mengeksplor setiap *node* yang berdekatan hingga menemukan rute terpendek menuju *node* tujuan *(t)*. Proses eksplorasi ini dikenal sebagai *edge relaxation* (Ortega-Arranz et al., 2014).

Dijkstra berkerja mirip dengan BFS, yaitu dengan menggunakan prinsip *queue*. Tetapi yang digunakan pada Dijkstra ialah *queue* prioritas sehingga satu-satunya *node* yang memiliki prioritas tertinggi akan dicari. Dalam menentukan *node* prioritas, algoritma ini membandingkan setiap nilai dari setiap *node* yang disimpan untuk dibandingkan dengan nilai yang ditemukan dari rute berikutnya dan begitu seterunya hingga *node* tujuan ditemukan (Wang et al., 2015). Sama seperti BFS, pada algoritma ini juga diberikan warna *node* berbeda sesuai dengan keterangannya masing-masing untuk melacak proses eksplorasi.

2.3. *Greedy Best First Search*

Greedy BFS merupakan jenis algoritma BFS yang paling sederhana (Mahmud et al., 2012). Sama seperti algoritma sebelumnya, *greedy* BFS juga menggunakan struktur data *queue* tetapi yang dijadikan prioritas bukanlah jarak yang ditempuh. Sebagai gantinya, penulis perlu membuat *heuristic* untuk menentukan apakah suatu *node* lebih baik dibanding yang lain untuk mencapai *node* tujuan. *Heuristic* sendiri ialah metode pemecahan masalah yang menggunakan jalan pintas untuk menghasilkan solusi yang cukup baik. Fungsi *heuristic* sebenarnya dapat sangat bervariasi tergantung bagaimana sebuah graf diatur (Lawrence & Bulitko, 2012). Namun dalam kasus ini, graf dibuat untuk melakukan *pathfinding* pada *grid* dua dimensi sehingga fungsi *heuristic* akan digunakan untuk mengukur jarak di antara suatu *node* (bukan *start node*) dengan *node* tujuan sehingga *greedy* BFS akan mempertimbangkan *node* yang lebih dekat dengan tujuan menjadi kandidat yang lebih baik untuk dieksplor.

Algoritma *greedy* BFS menggunakan fungsi evaluasi yang meniadakan perkiraan biaya *G. G* di sini adalah jarak *node* awal ke suatu *node*. Sehingga fungsi evaluasi yang digunakan adalah seperti pada Pers. (2).

$$F = H \qquad (2)$$

2.4. A*

Algoritma A* adalah salah satu metode *pathfinding* paling popular yang banyak digunakan dan telah terjamin optimalisasinya (Hart et al., 1968). Namun, algoritma ini dapat dikatakan boros sumber daya (Yiu et al., 2018). A* sendiri merupakan perpaduan dari algoritma BFS dan Dijkstra (Gonçalves et al., 2019). Algoritma ini menggunakan pencarian BFS untuk menemukan jalur dengan biaya terendah dari suatu *node* ke *node* awal. Sedangkan fungsi *heuristic* yang mirip dengan Dijkstra digunakan untuk mencari jarak paling kecil dari suatu *node* ke *node* tujuan. Oleh karena itu, A* menggunakan fungsi evaluasi yang teridiri dari dua bagian, yaitu *heuristic* $H(n)$ dan perkiraan biaya $G(n)$, di mana

$$F(n) = G(n) + H(n) \qquad (3)$$

$$fCost = gCost + hCost \qquad (4)$$

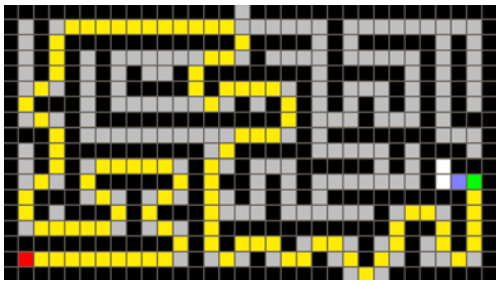
gCost di sini berarti jarak dari *node* awal sedangkan *hCost* ialah fungsi *heuristic* yang berarti jarak dari *node* tujuan. Dalam percobaan ini fungsi *heuristic* yang digunakan ialah *manhattan distance*. Perlu diingat bahwa fungsi *heuristic* hanya memberi nilai estimasi karena fungsi ini tidak mempertimbangkan *blocked node.* Jarak sebenarnya bisa saja lebih besar dari *heuristic* yang telah dihitung. Sehingga dapat disimpulkan bahwa *fCost* merupakan jarak pasti dari *node* awal ditambah dengan perkiraan jarak menuju *node* tujuan. Hasilnya akan digunakan untuk memilih *node* berikutnya yang masuk ke dalam *queue*.

3. HASIL DAN PEMBAHASAN

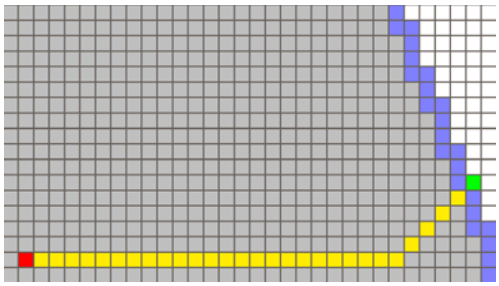
Dalam percobaan membandingkan algoritma *pathfinding* ini terdapat dua variabel *input* dan dua variabel *output.* Variabel *input* berupa algoritma *pathfinding* seperti BFS, Dijkstra, *greedy* BFS,

dan A* dan *grid* dua dimensi yang memiliki ukuran serta jenis bervariasi. Sedangkan variabel *output* yaitu, waktu eksekusi dan jarak yang ditempuh

Sebelum masuk ke pembahasan hasil, berikut adalah contoh gambar akhir dari proses pencarian pada program yang telah dikembangkan.



Gambar 4. *Pathfinding* pada *maze*



Gambar 5. Pathfinding pada Empty Grid

Node dengan warna merah berarti *node* awal sedangkan *node* akhir ditandai dengan warna hijau. Setiap *node* yang akan dikunjungi memiliki warna biru dan warna abu-abu berlaku untuk *node* yang telah dikunjungi. Setelah proses traversal selesai *node* yang dilintasi suatu algoritma untuk mencapai *node* tujuan akan ditandai dengan warna kuning.

Hasil dari percobaan setiap algoritma secara berurutan dimulai dari waktu eksekusi. Waktu eksekusi di sini berarti waktu yang dibutuhkan setiap algoritma untuk melakukan eksplorasi pada *grid* dua dimensi sekaligus menemukan rute dari *node* awal menuju *node goal*. Waktu eksekusi diukur dengan satuan *ms.*

Tabel 1 menunjukkan hasil waktu eksekusi dari algoritma *breadth-first search* (BFS). Dapat dilihat bahwa algoritma BFS memerlukan waktu lebih lama untuk melakukan eksplorasi pada *grid* terbuka. Sedangkan *grid* yang memiliki banyak *blocked node* seperti *maze* dapat ditelusuri dengan lebih cepat. Waktu eksekusi juga meningkat secara signifikan untuk area yang lebih luas seperti yang terlihat pada *grid* dengan ukuran 256x144.

Tabel 1. Hasil Waktu Eksekusi BFS

Jenis *grid*	Ukuran *grid*			
	32x18	64x36	128x72	256x144
Empty grid	45	1098	9024	196109
Open Walls	26	465	5407	116192
Terrain	44	605	10091	183733
Maze	15	69	2680	27838

BFS memerlukan waktu selama 196 detik dan 183 detik dalam menelusuri *empty grid* dan *terrain grid* berukuran 256x144. Waktu tersebut termasuk cukup besar mengingat ukuran *grid* yang relatif kecil. Hasil dari waktu eksekusi ini menunjukkan bahwa algoritma BFS tidak

direkomendasikan untuk diterapkan pada masalah pencarian *real-time* dalam *grid* terbuka berskala besar.

Pada hasil algoritma Dijkstra yang dapat dilihat di Tabel 2, waktu eksekusi meningkat secara linier sampai dengan *grid* berukuran 128x72. Namun, angka tersebut meningkat pesat pada *grid* berukuran 256x144. Dijkstra memerlukan waktu selama lebih dari 100 detik untuk melakukan eksplorasi pada *empty grid* dan *terrain*. Angka tersebut 25 kali lebih besar daripada ukuran *grid* di bawahnya.

Tabel 2. Hasil Waktu Eksekusi Dijkstra

Jenis *grid*	Ukuran *grid*			
	32x18	64x36	128x72	256x144
Empty grid	28	343	4133	104082
Open Walls	18	234	2640	67632
Terrain	32	245	4300	106790
Maze	12	43	1435	14683

Jika dibandingkan dengan Dijkstra, kinerja algoritma *greedy* BFS jauh lebih baik dalam setiap jenis dan ukuran *grid*. Dapat dilihat pada *empty grid* dan *terrain* yang tidak memaliki *blocked node* sama sekali, waktu eksekusi untuk setiap ukuran *grid* tidak terpaut jauh. Sedangkan untuk *open walls* yang memiliki sedikit *blocked node, greedy* BFS memerlukan waktu lima kali lebih lama untuk *grid* berskala besar. *Greedy* BFS terlihat lebih susah dalam melakukan *pathfinding* pada area yang lebih luas dan memiliki banyak *obstacle* seperti *maze* dengan ukuran 256x144. Nilai yang dihasilkan dalam percobaan pada *grid* tersebut meningkat jauh dibandingkan dengan *grid* lainnya. Namun, waktu eksekusi dari *greedy* BFS masih lebih baik daripada dua algoritma sebelumnya.

Tabel 3. Hasil Waktu Eksekusi *Greedy* BFS

Jenis *grid*	Ukuran *grid*			
	32x18	64x36	128x72	256x144
Empty grid	10	13	17	34
Open Walls	8	11	50	395
Terrain	11	12	18	35
Maze	10	14	688	8718

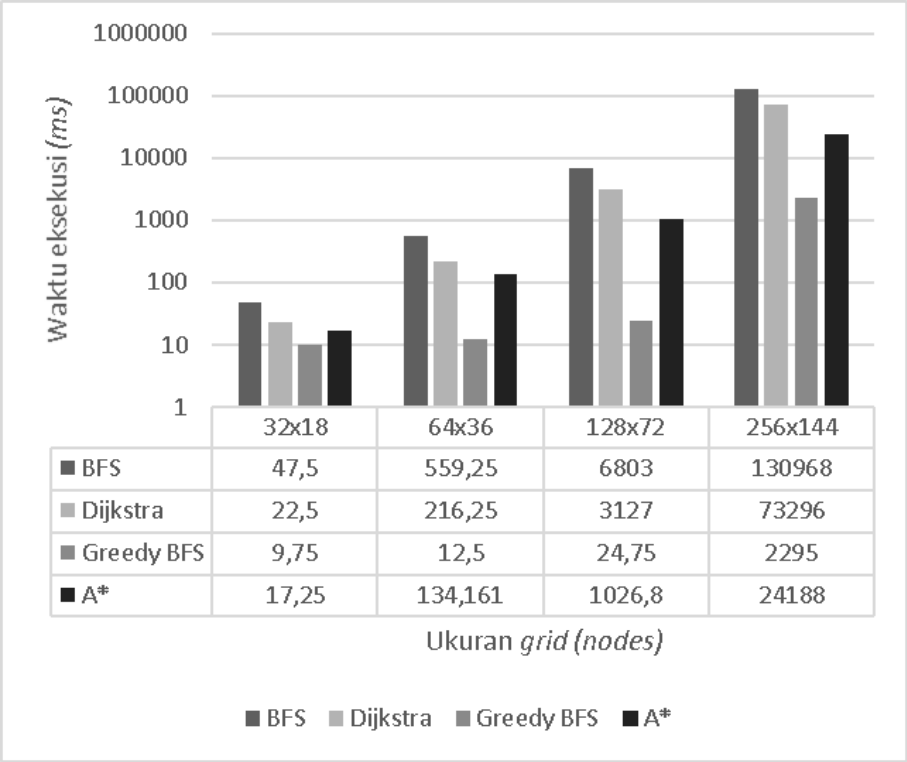
Berikutnya adalah hasil dari algoritma A* yang dapat dilihat pada Tabel 3. Sama seperti *greedy* BFS, algoritma A* memiliki hasil waktu eksekusi yang lebih baik dibandingkan BFS dan Dijkstra. Namun, pada jenis *map* yang lebih rumit seperti *maze* dan *terrain,* A* memerlukan waktu yang jauh lebih banyak untuk menyelasaikan proses *pathfinding*.

Tabel 4. Hasil Waktu Eksekusi A*

Jenis *grid*	Ukuran *grid*			
	32x18	64x36	128x72	256x144
Empty grid	11	13	17	158
Open Walls	15	44	59	6038
Terrain	30	127	2711	77295
Maze	13	34	1319	13261

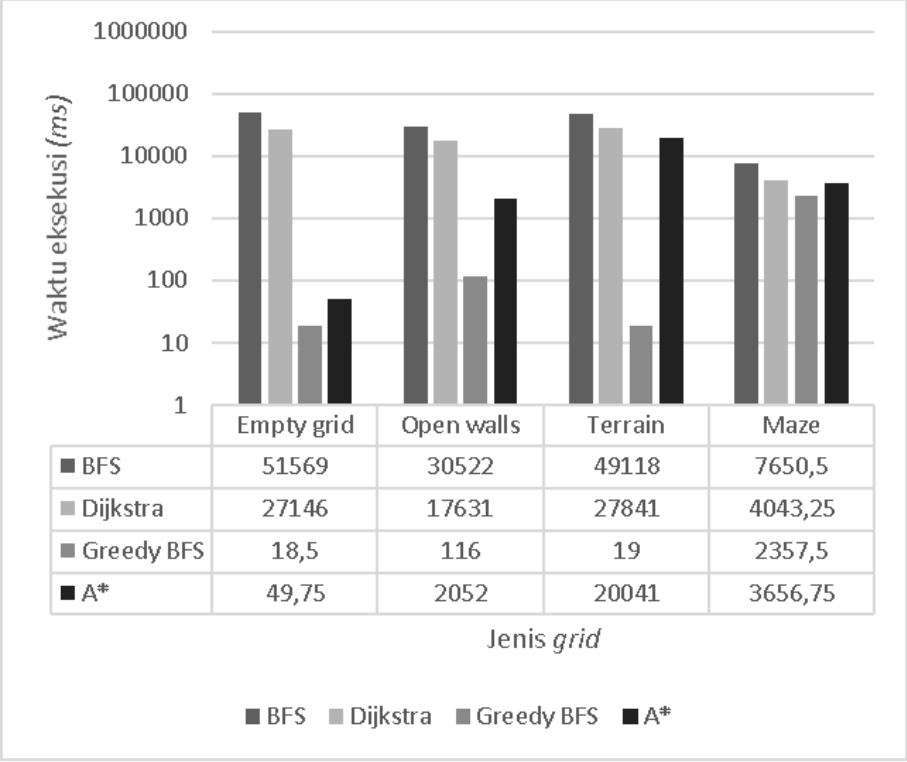
Selanjutnya rata-rata hasil eksekusi dicari berdasarkan ukuran *grid* dan jenis *grid.* Rumus yang digunakan untuk mencari rata-rata dapat dilihat pada Pers. (5) di mana $\bar{x}$ adalah nilai rata-rata, x adalah jumlah data, dan n merupakan berapa banyak data yang ada pada ukuran atau jenis *grid* tertentu. Seluruh waktu eksekusi yang nilai rata-ratanya telah diketahui dijabarkan pada Gambar 6 dan 7.

$$\bar{x} = \frac{x_1+x_2+x_3+\cdots+x_n}{n} \quad (5)$$



	32x18	64x36	128x72	256x144
BFS	47,5	559,25	6803	130968
Dijkstra	22,5	216,25	3127	73296
Greedy BFS	9,75	12,5	24,75	2295
A*	17,25	134,161	1026,8	24188

Gambar 6. Rata-rata Hasil Waktu Eksekusi (Ukuran *grid*)



	Empty grid	Open walls	Terrain	Maze
BFS	51569	30522	49118	7650,5
Dijkstra	27146	17631	27841	4043,25
Greedy BFS	18,5	116	19	2357,5
A*	49,75	2052	20041	3656,75

Gambar 7. Rata-rata Hasil Waktu Eksekusi (Jenis *Grid*)

Hasil percobaan untuk variabel *output* waktu eksekusi menunjukkan bahwa *greedy* BFS merupakan algoritma yang paling cepat dalam melakukan proses *pathfinding* disemua ukuran dan jenis *grid. Grid* berskala lebih kecil tidak mempengaruhi kecepatan *greedy* BFS melihat peningkatan waktunya yang tidak seberapa. Namun, peningkatan waktu eksekusi terlihat jelas pada jenis *grid* berukuran besar atau yang memiliki banyak hambatan seperti *maze.* Tetapi nilai tersebut masih lebih baik dibandingkan algoritma lainnya. Algoritma A* sendiri mampu menyelesaikan proses *pathfinding* dalam waktu yang tidak jauh berbeda dibandingkan *greedy* BFS pada *grid* yang yang lebih sederhana dan berskala kecil. Namun, untuk *grid* yang lebih rumit dan berskla besar A* membutuhkan waktu lebih lama. Waktu eksekusi paling lambat ditunjukkan oleh algoritma BFS yang mana nilainya masih jauh di bawah Dijkstra yang merupakan algoritma paling lambat kedua.

Masuk pada pembahasan variabel *output* berikutnya yaitu, jarak yang ditempuh. Selain waktu eksekusi, rute terpendek juga merupakan hal yang dibutuhkan dalam proses *pathfinding.* Semakin kecil jaraknya maka semakin baik. Dalam percobaan ini, jarak yang ditempuh diukur berdasarkan biaya yang dimiliki oleh setiap *node* yang telah disinggung pada poin 2. Tabel 5 menunjukkan jarak yang ditempuh algoritma BFS dari *node* awal ke *node* tujuan untuk setiap jenis dan ukuran *grid* yang berbeda.

Tabel 5. Hasil Jarak Tempuh BFS

Jenis *grid*	Ukuran *grid*			
	32x18	64x36	128x72	256x144
Empty grid	37,8	75,4	155,4	310,8
Open Walls	55,4	86,2	176,8	336,8
Terrain	58,6	159,4	290,6	579,4
Maze	110,6	129,6	743,6	1816,4

Data yang dihasilkan pada algoritma Dijkstra, yang dapat dilihat pada Tabel 6, tidak berbeda jauh dari BFS. Hal ini dikarenakan kedua algoritma tersebut menggunakan prinsip traversal yang serupa. Namun, perbedaan masih dapat dilihat pada jenis *terrain* di mana *grid* tersebut memiliki 4 macam *node* dengan biaya berbeda, membuatnya menjadi lebih rumit untuk dieksplor. Algoritma Dijkstra menggunakan *queue* prioritas sehingga mampu memperhitungkan biaya *node* yang berbeda untuk mengambil rute dengan jarak tempuh yang lebih kecil.

Tabel 6. Hasil Jarak Tempuh Dijkstra

Jenis *grid*	Ukuran *grid*			
	32x18	64x36	128x72	256x144
Empty grid	37,8	75,4	155,4	312,2
Open Walls	55,4	86,2	176,8	336,8
Terrain	58,6	151,4	281	579,4
Maze	110,6	129,6	743,6	1816,4

Berikutnya ialah jarak yang ditempuh pada *greedy* BFS. Dalam penelitian ini, *greedy* BFS memiliki jarak tempuh paling jauh karena algoritma tersebut minim dalam melakukan eksplorasi. Hal ini memang membawa pada waktu eksekusi yang lebih cepat, tetapi rute yang diambil menjadi tidak terarah dan membuat banyak belokan yang tidak perlu. Hasil dari *greedy* BFS dapat dilihat pada Tabel 7.

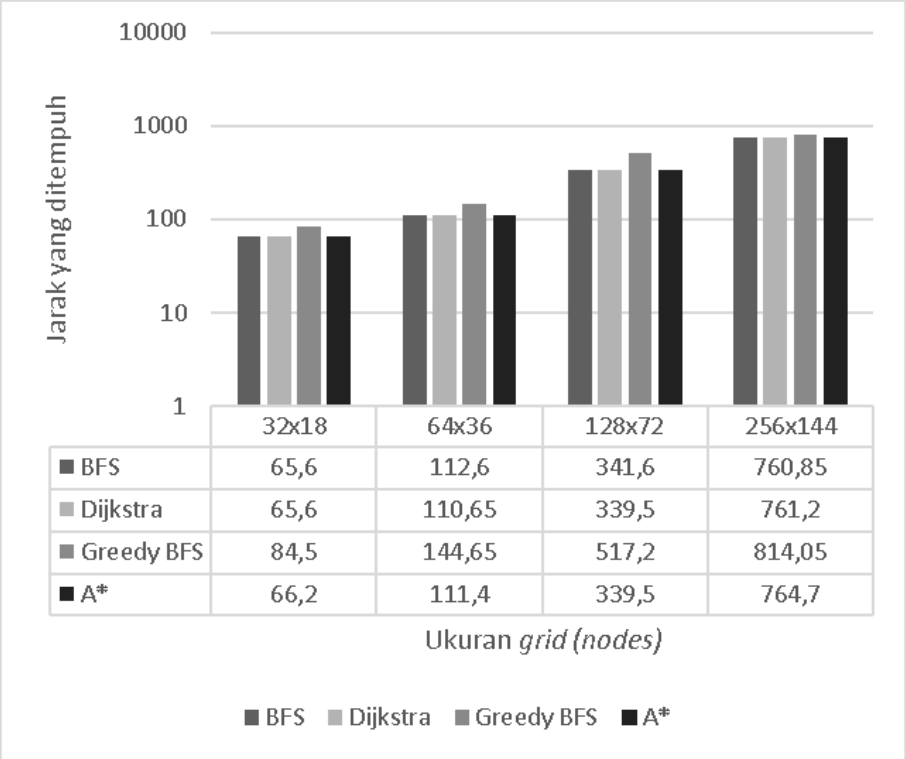
Tabel 7. Hasil Jarak Tempuh *Greedy* BFS

Jenis *grid*	Ukuran *grid*			
	32x18	64x36	128x72	256x144
Empty grid	37,8	75,4	155,4	312,2
Open Walls	64,8	99,8	233,8	450,8
Terrain	113,8	271,4	319,4	676,2
Maze	121,8	132	843	2034

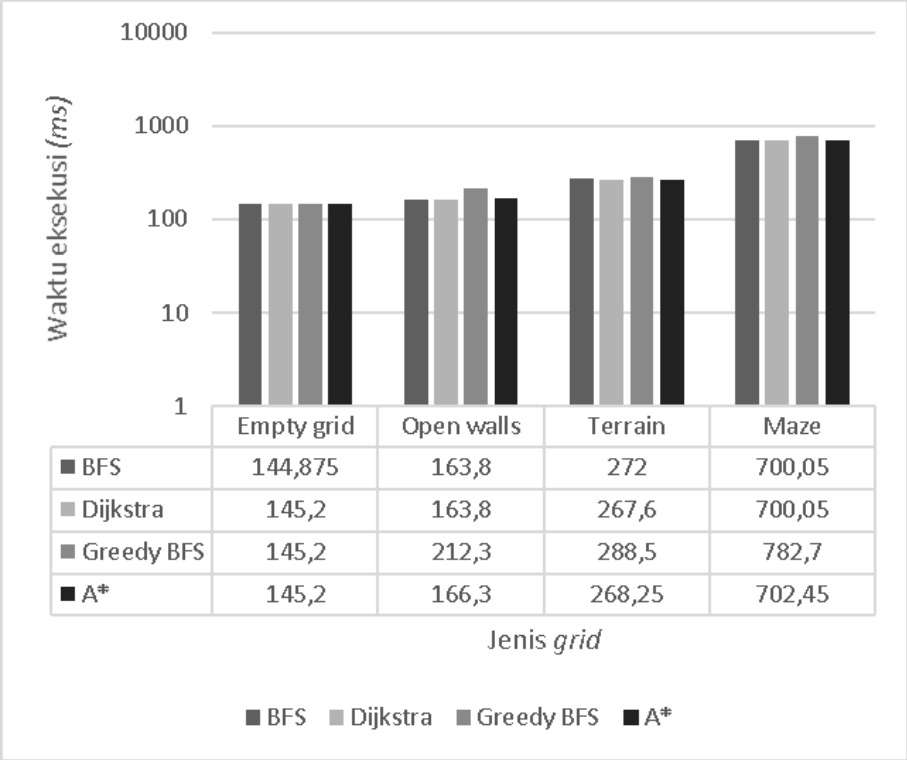
Algoritma A* menghasilkan jarak tempuh yang lebih baik dibandingkan algoritma sebelumnya namun masih di bawah Dijkstra. Hal tersebut dikarenakan algoritma ini menggabungkan dua fungsi *heuristic* sehingga membuat waktu eksekusi dan jarak tempuh sebagai prioritas yang setara. Tabel 8 menunjukkan hasil jarak tempuh dari algoritma A* pada setiap jenis dan ukuran *grid*. Sama seperti variabel waktu eksekusi, agar hasil yang dibandingkan dapat terlihat dengan jelas, rata-rata jarak tempuh dicari dan dijabarkan dalam Gambar 8 dan 9.

Tabel 8. Hasil Jarak Tempuh A*

Jenis *grid*	Ukuran *grid*			
	32x18	**64x36**	**128x72**	**256x144**
Empty grid	37,8	75,4	155,4	312,2
Open Walls	57,8	88,4	177,6	341,4
Terrain	58,6	152,2	281	581,4
Maze	110,6	129,6	745,4	1823,8



Gambar 8. Rata-rata Hasil Jarak Tempuh (Ukuran *Grid*)



Gambar 9. Rata-rata Hasil Jarak Tempuh (Jenis *Grid*)

Algoritma BFS, Dijkstra, dan A* tidak memiliki perbedaan yang signifikan dalam perbandingan berdasarkan jarak tempuh. Ketiga algoritma itu mampu memiliki sedikit keunggulan pada jenis dan ukuran *grid* tertentu. Seperti BFS yang lebih unggul dalam *empty grid,* sedangkan Dijkstra menunjukkan keunggulan pada hampir semua jenis dan ukuran *grid.* A* sendiri masih sedikit di bawah BFS pada hampir semua jenis *grid*. Namun, untuk *grid* yang lebih rumit seperti *terrain* A* memiliki hasil yang lebih baik walaupun masih di bawah Dijkstra. Sebaliknya *greedy* BFS menghasilkan jarak tempuh yang jauh lebih buruk dibandingkan ketiga algoritma lainnya.

4. KESIMPULAN

Bedasarkan perbandingan waktu eksekusi untuk empat algoritma *pathfinding*, BFS memiliki waktu paling lambat. Hal ini dikarenakan BFS menggunakan operasi algoritma yang sangat sederhana dan tidak menggunakan heuristik. Algoritma Dijkstra lebih cepat daripada BFS tetapi masih termasuk lambat dibanding dua algoritma lainnya. Kinerja *greedy* BFS dan A* tidak berbeda jauh pada *grid* dengan skala kecil dan denah yang lebih sederhana. Namun, pada *grid* berskala besar A* lebih lambat daripada *greedy* BFS tetapi masih termasuk cepat dibandingkan BFS dan Dijkstra.

Sedangkan untuk jarak tempuh, *greedy* BFS memilih rute yang lebih panjang dalam mencapai tujuannya. Tidak seperti ketiga algoritma lainnya yang mampu menghasilkan jarak konsisten untuk setiap jenis dan ukuran *grid.*

Uraian tadi mengarah pada kesimpulan bahwa setiap algorima dapat diterapkan dalam domain tertentu tergantung kebutuhan si pengguna. Dalam pengujian komparasi ini, setiap algoritma memiliki kelebihan dan kekurangan masing-masing yang dijabarkan dalam Tabel 9.

Tabel 9. Tabel Komparasi dari Empat Algoritma

Algoritma	Waktu eksekusi				Jarak tempuh			
	Skala		Area		Skala		Area	
	Besar	Kecil	Sederhana	Kompleks	Besar	Kecil	Sederhana	Kompleks
BFS	×	√	×	√	√	√	√	√
Dijkstra	×	√	×	√	√	√	√	√
Greedy BFS	√	√	√	√	×	×	×	×
A*	√	√	√	√	√	√	√	√

Namun, jika mencari fleksibelitas untuk setiap domain maka algoritma A* dapat menjadi pilihan sebagaimana yang telah dipaparkan dalam Tabel 9.

DAFTAR PUSTAKA

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

Gonçalves, S. M. M., Da Rosa, L. S., & Marques, F. S. (2019). An improved heuristic function for A*-based path search in detailed routing. *Proceedings - IEEE International Symposium on Circuits and Systems*, *2019-May*. https://doi.org/10.1109/ISCAS.2019.8702460

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. https://doi.org/10.1109/TSSC.1968.300136

Lawrence, R., & Bulitko, V. (2012). Database-Driven Real-Time Heuristic Search in Video-Game Pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, *5*(3), 227–241. https://doi.org/10.1109/TCIAIG.2012.2230632

Mahmud, M. S., Sarker, U., Islam, M. M., & Sarwar, H. (2012). A Greedy Approach in Path Selection for DFS Based Maze-map Discovery Algorithm for an autonomous robot. *Proceeding of the 15th International Conference on Computer and Information Technology, ICCIT 2012*. https://doi.org/10.1109/ICCITechn.2012.6509798

Ortega-Arranz, H., Llanos, D. R., & Gonzalez-Escribano, A. (2014). The Shortest-Path Problem: Analysis and Comparison of Methods. *Synthesis Lectures on Theoretical Computer Science*, *1*(1), 1–87. https://doi.org/10.2200/s00618ed1v01y201412tcs001

Risald, Mirino, A. E., & Suyoto. (2017). Best routes selection using Dijkstra and Floyd-Warshall algorithm. *Proceedings of the 11th International Conference on Information and Communication Technology and System, ICTS 2017*, 155–158. https://doi.org/10.1109/ICTS.2017.8265662

Wang, C., Liu, J., Deng, L., Yunyun, X., Lin, S., Xu, H., Zheng, R., & Cao, X. (2015). Research on the optimization of the DC converter station recovery path based on BFS. *10th International Conference on Advances in Power System Control, Operation & Management (APSCOM 2015)*, *2015*(8). https://doi.org/10.1049/ic.2015.0282

Yiu, Y. F., Du, J., & Mahapatra, R. (2018). Evolutionary Heuristic A∗ Search: Heuristic Function Optimization via Genetic Algorithm. *Proceedings - 2018 1st IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2018*, 25–32. https://doi.org/10.1109/AIKE.2018.00012

Zhao, L., & Zhao, J. (2017). Comparison study of three shortest path algorithm. *Proceedings - 2017 International Conference on Computer Technology, Electronics and Communication, ICCTEC 2017*, 748–751. https://doi.org/10.1109/ICCTEC.2017.00165