TELEKONTRAN, VOL. 9, NO. 1, APRIL 2021
DOI : 10.34010/telekontran.v9i1.5677
p-ISSN : 2303 – 2901
e-ISSN : 2654 – 7384

Analisa Penggunaan Nilai Bobot Heuristik yang Berbeda pada Algoritma *Weighted* A*

Analysis of the Use of Different Heuristic Weights in the Weighted A Algorithm*

Ipan Diana*, Budi Herdiana
Program Studi Teknik Elektro, Fakultas Teknik dan Ilmu Komputer
Universitas Komputer Indonesia Jl. Dipati ukur No 112, Bandung
*Email : ipandiana@mahasiswa.unikom.ac.id

Abstrak - *Path planning* merupakan urutan keadaan untuk memindahkan objek dari keadaan awal ke keadaan akhir, serta menghindari daerah yang tidak dapat dilalui. Objek disini dapat berupa robot, mobil otonom dan yang lainnya. Algoritma A* merupakan algoritma pencarian jalur yang menggunakan estimasi jarak dengan menggunakan pencarian jalur terdekat untuk mencapai tujuan. *Weighted* A* adalah algoritma yang digunakan untuk memecahkan masalah pencarian jalur dengan mengubah nilai bobot pada fungsi heuristiknya. Tujuan dari penelitian ini yaitu menganalisa perbandingan algoritma *Weighted* A* dengan algoritma A*, serta menganalisa pengaruh nilai bobot heuristik pada algoritma *Weighted* A*. Pengujian yang dilakukan yaitu menggunakan lingkungan *maze, narrow, trap, clutter*. Hasil yang didapat pada perbandingan algoritma *Weighted* A* dan A*, diperoleh algoritma *Weighted* A* menghasilkan waktu pencarian rata-rata yang lebih baik yaitu sebesar 3,49 detik, sedangkan algoritma A* menghasilkan waktu rata-rata sebesar 4,68 detik. Tetapi algoritma A* dapat menghasilkan jalur rata-rata yang lebih optimal yaitu 53,90 dibandingkan algoritma *Weighted* A* yang menghasilkan jalur rata-rata sebesar 53,91. Dengan strategi yang lebih menekankan pemilihan *node* yang lebih dekat dengan *node goal*, maka *Weighted* A* dapat menghasilkan jalur dengan waktu komputasi yang lebih cepat. Sedangkan algoritma A* karena memilih *node* dengan nilai heuristik terkecil, maka dapat menghasilkan jalur yang lebih optimal. *Weighted* A* cocok di implementasikan pada sistem yang membutuhkan waktu pencarian jalur yang lebih singkat tapi tidak harus optimal. Algoritma A* cocok di implementasikan pada sistem yang membutuhkan jalur optimal walaupun waktu pencariannya tidak terlalu cepat.

Kata kunci *:* Algoritma A*, Heuristik, Perencanaan Jalur, Rute Terpendek, *Weighted* A*

Abstract *- Path planning is a sequence of states to move objects from the initial state to the final state and avoid impassable areas. Objects here can be robots, autonomous cars, and others. The A* algorithm is a path search algorithm that uses distance estimation by using the closest path search to reach the destination. Weighted A* is an algorithm used to solve the pathfinding problem by changing the weight value in the heuristic function. The purpose of this study is to analyze the comparison of the Weighted A* algorithm with the A* algorithm and analyze the effect of the heuristic weight value on the Weighted A* algorithm. The tests carried out are using maze, narrow, trap, clutter environments. The results obtained in the comparison of the Weighted A* and A* algorithms, the Weighted A* algorithm produces a better average search time of 3.49 seconds, while the A* algorithm produces an average time of 4.68 seconds. But the A* algorithm can produce a more optimal average path of 53.90 than the Weighted A* algorithm which produces an average path of 53.91. With a strategy that emphasizes choosing nodes that are closer to the goal node, Weighted A* can produce a path with a faster computation time. While the A* algorithm because it chooses the node with the smallest heuristic value, it can produce a more optimal path. Weighted A* is suitable to be implemented on systems that require shorter path-finding times but do not have to be optimal. The A* algorithm is suitable to be implemented in systems that require optimal paths even though the search time is not too fast.*.

Keywords *: A* Algorithm, Heuristic, Path Planning, Shortest Route, Weighted A**

I. PENDAHULUAN

A. Latar Belakang

Perencanaan jalur menggambarkan urutan keadaan dalam skenario untuk memindahkan objek dari keadaan awal ke keadaan akhir, serta menghindari daerah yang tidak dapat dilalui (rintangan, zona bahaya) dari ruang pencarian. Setiap keadaan adalah elemen dari ruang pencarian, yang merupakan himpunan kemungkinan yang dapat diterapkan suatu objek, pada momen lintasannya [1]. Salah satu pendekatan berbasis sampling pertama yaitu *probabilistic roadmap* (PRM), yang dikembangkan oleh Kavraki [2]. Jenis lain dari algoritma jenis sampling adalah Ariadne's Clew [3]. Contoh lainnya adalah algoritma *Rapidly-exploring Random Tree* (RRT) dan RRT*. Algoritma ini memiliki sifat *completeness*, *optimality* dan konvergen menuju nilai optimal [4]. kemudian ada algoritma *Dijkstra*, A* (AStar), *Breadth-first Search* (BFS), *Depth-first Search* (DFS).

Dijkstra dirumuskan oleh ilmuwan komputer Edsger W. *Dijikstra's* pada tahun 1956 [5]. Algoritma *Dijkstra* memecahkan masalah jalur terpendek dengan biaya jalur tepi non-negatif [6]. Disisi lain, algoritma A* adalah salah satu dari sekian banyak algoritma perencanaan jalur yang mengambil input, mengevaluasi beberapa jalur yang memungkinkan, dan mengembalikan solusinya. A* adalah algoritma pencarian terbaik pertama yang memodifikasi fungsi heuristik [7]. Algoritma ini meminimalkan total biaya yang ditempuh, dan dalam kondisi yang tepat akan memberikan solusi terbaik dalam waktu yang optimal.

Weighted A* (WA*) adalah algoritma yang banyak digunakan untuk memecahkan masalah perencanaan dan pencarian dengan cepat [8]. Pada WA*, parameter *h* dikalikan dengan sebuah nilai *weighted* (bobot) dalam memprioritaskan *node* terbuka. *Weighted* dituliskan dengan huruf W, ia merupakan variabel yang terikat untuk solusi yang dihasilkan oleh WA*.

B. Tinjauan *State of Art*

A* digunakan pada banyak bidang aplikasi, Akshay dkk telah melakukan penelitian mengenai efisiensi waktu pada algoritma A* untuk aplikasi robot. Dalam penelitiannya, sebuah algoritma harus dirancang untuk memastikan jalur bebas hambatan bagi robot dalam bidang industri. Modifikasi terhadap algoritma A* disajikan dalam penelitian ini. Hasil yang didapat yakni dengan algoritma A* menunjukkan pengurangan waktu pemrosesan maksimum sebesar 95% [9].

Shrawan dan B.L.Pal membahas perbandingan algoritma *Dijkstra's* dan A* untuk kebutuhan pencarian jalur terpendek pada lalulintas dalam kota. Pencarian jalan terpendek sangat penting dalam beberapa kasus khusus seperti pemadam kebakaran, ambulans yang membawa pasien, penangkapan pencuri oleh pihak kepolisian, dll. Penelitian ini mengatakan bahwa algoritma A* berkinerja lebih baik daripada algoritma *Dijkstra's* di semua kasus (dengan *obstacle* dan tanpa *obstacle*) [10].

Adapun Jordan dan Wheeler membahas pendekatan baru pencarian heuristik dengan suboptimalitas terbatas yaitu *op-timistic search*. Pencarian ini menggabungkan *aggressively greedy search* dengan fase pembersihan. Dari hasil demonstrasi memperlihatkan bahwa algoritma ini secara konsisten melampaui *Weighted* A* dalam empat domain *benchmark* yang berbeda termasuk perencanaan temporal dan pencarian jalur grid [11].

Eric dan Rong Zhou telah melakukan penelitian mengenai cara mengubah algoritma pencarian heuristik A* menjadi algoritma *Anytime* A*. Pendekatan yang diadopsi yaitu menggunakan pencarian heuristik berbobot untuk menemukan solusi perkiraan yang cepat, dan kemudian melanjutkan pencarian berbobot untuk menemukan solusi yang lebih baik [12].

C. Tujuan

Tujuan dari penelitian ini adalah untuk menganalisa perbandingan antara algoritma *Weighted* A* dengan algoritma A*, serta menganalisa pengaruh nilai bobot heuristik pada algoritma *Weighted* A*. Parameter yang akan dibandingkan meliputi waktu komputasi, *path cost*, dan penggunaan memori. Pengujian dilakukan pada lingkungan *maze, narrow, trap* dan *clutter.*

D. Sistematika Pembahasan

Makalah ini diorganisasikan sebagai berikut. Bagian 2 akan menjelaskan mengenai perancangan program pada aplikasi *python*. Bagian 3 akan menyajikan hasil pengujian dan analisa. Adapun kesimpulkan akan disajikan pada Bagian 4.

II. METODOLOGI

A. Algoritma A*

A* adalah algoritma pencarian terbaik pertama yang dianggap sebagai versi lanjutan dari algoritma *Dijkstra's* [32]. Diperkenalkan pada

tahun 1968 oleh Peter dan Bertram [33]. Algoritma ini mencoba untuk mengurangi jumlah total keadaan yang akan dijelajahi dengan memasukkan perkiraan biaya heuristik untuk mencapai tujuan dari keadaan tertentu. A* memiliki dua karakteristik yang membentuk fungsi baru yaitu sebagai berikut:

$$f(n) = g(n) + h(n) \quad (1)$$

Untuk setiap *node* n, total biaya $f(n)$ diberikan sebagai jumlah dari biaya $g(n)$ dan $h(n)$. g adalah biaya yang ditempuh dari titik awal *node* menuju *node* n. h adalah faktor heuristik yang memperkirakan biaya perpindahan dari *node* n menuju *node* tujuan [34].

A* diyakini didasarkan pada algoritma yang disebutkan diatas karena A* seperti algoritma *Dijkstra's* yang menemukan jalur terpendek tanpa gagal dan seperti *Greedy Best-First Search* yang menggunakan fungsi heuristik untuk memperkirakan jarak ke tujuan. *Pseudocode* A* diperlihatkan pada **Gambar 1**.

```
function AStar
initialize start_node and calculate xy_index from start_node
initialize goal_node and calculate xy_index from goal_node
initialize unvisited_list and visited_list
calculate grid_index start_node and put into unvisited_list
while unvisited_list is not empty
    c_id = unvisited_list.cost + heuristic_value(goal_node,unvisited_list)
    current = the node in unvisited_list having the lowest c_id value
    if current = goal_node, stop the search
    remove current from unvisited_list
    add current to visited_list
    generate eight successor and calculate c_id
    for each successor
        node = find paths from eight possible directions
        n_id = calculate grid_index from node
        if node not safe, do nothing
        if n_id in visited_list, do nothing
        if n_id not in unvisited_list:
            unvisited_list=node
        if cost from unvisited list greater than cost from node
            unvisited_list=node
calculate final path
```

Gambar 1. *Pseudocode* A*.

Algoritma *Dijkstra's* bekerja sangat baik untuk menemukan jalur terpendek dari satu titik akhir ke titik akhir lainnya, tetapi juga menghabiskan waktu dan sumber daya untuk menjelajahi arah yang tidak terlalu meyakinkan. Sebaliknya, *Greedy Best-First Search* mengeksplorasi arah yang memiliki harapan terbaik, tetapi gagal untuk secara konsisten menemukan jalur terpendek ke tujuan. Menggabungkan dua algoritma ini, A* menggunakan jarak dari awal dan perkiraan jarak ke tujuan untuk menghilangkan keterbatasan algoritma konvensional ini.

B. *Weighted* A*

Untuk masalah pencarian yang sulit, A* mungkin membutuhkan terlalu lama waktu untuk menemukan solusi yang optimal, dan solusi perkiraan yang relatif lebih cepat dapat lebih berguna. Banyak peneliti telah mengeksplorasi efek pembobotan nilai $g(n)$ dan $h(n)$ dalam evaluasi *node* secara berbeda, untuk memungkinkan A* menemukan solusi optimal dengan upaya komputasi yang lebih sedikit. Dalam pendekatan ini, fungsi evaluasi *node* didefinisikan sebagai:

$$f'(n) = g(n) + w.h(n) \quad (2)$$

Dimana parameter *weight* w ≥ 1 ditetapkan oleh pengguna. Heuristik berbobot mempercepat solusi pencarian karena membuat *node* lebih dekat ke tujuan. Pencarian heuristik berbobot paling efektif untuk masalah pencarian dengan solusi yang mendekati optimal.

C. Blok Pemrograman

Pertama akan dijelaskan mengenai blok inisialisasi. Inisialisasi awal berfungsi untuk memberikan suatu nilai terhadap objek atau variabel. Variabel merupakan tempat untuk menyimpan data. Pemberian nilai variabel dalam *python* dapat menggunakan simbol = (sama dengan). Pada saat akan membuat variabel baru, kita diharuskan untuk membuat *self* sebagai parameter pertama. Keyword *self* digunakan untuk merepresentasikan setiap objek yang dibuat. Jika tidak ada *self*, maka *class* tidak akan bisa menampung informasi yang terdapat pada objek tersebut.

```
1  import math
2  import matplotlib.pyplot as plt
3  show_animation = True
4  class AStar:
5      def __init__(self, ax, ay, grid_size, robot_radius):
6          self.grid_size = grid_size
7          self.robot_radius = robot_radius
8          self.min_x, self.min_y = 0, 0
9          self.max_x, self.max_y = 0, 0
10         self.map_obstacle = None
11         self.grid_x_num, self.grid_y_num = 0, 0
12         self.direction = self.get_direction_model()
13         self.obstacle_map(ax, ay)
14     class Node:
15         def __init__(self, x, y, cost, parent):
16             self.x = x
17             self.y = y
18             self.cost = cost
19             self.parent = parent
```

Gambar 2. Inisialisasi awal.

Berdasarkan **Gambar 2**, pertama-tama dilakukan penambahan *library* yaitu *math* dan *matplotlib*. *Library math* merupakan salah satu fungsi bawaan pada *python* yang digunakan untuk melakukan operasi matematika. *Matplotlib* adalah *library plotting* dua maupun tiga dimensi yang menghasilkan visualisasi data statis dan interaktif. *Matplotlib* digunakan untuk memvisualisasikan jalur yang dihasilkan oleh algoritma A*.

Selanjutnya, dilakukan penambahan fungsi *class* dengan nama AStar. *Class* merupakan kerangka untuk membentuk suatu objek. Inisialisasi selanjutnya yaitu untuk titik *start* dan titik *goal*, dimana titik awal berada pada koordinat 0,0 sedangkan titik akhir berada pada koordinat 50,50. Selain itu ada inisialisasi *grid_size* dengan nilai 2, dan *robot_radius* dengan nilai 1. Inisialisasi ini dapat dilihat pada **Gambar 3**.

```
154    sx = 0.0
155    sy = 0.0
156    gx = 50.0
157    gy = 50.0
158    grid_size = 2.0
159    robot_radius = 1.0
```

Gambar 3. Inisialisasi titik *start, goal, grid_size, robot_radius*.

Kemudian dilakukan pembuatan batas pencarian dan juga *obstacle*. pertama-tama dibuat variabel ax dan ay dengan tipe *array*. Baris 162-173 merupakan nilai batasan yang digunakan pada area pencarian, adapun batas tersebut yaitu mulai dari -10 hingga 60 untuk sumbu x maupun y. Pada baris 174-179 merupakan contoh batasan yang akan digunakan untuk pembuatan *obstacle*. Program ini dapat dilihat pada **Gambar 4**.

```
161    ax, ay = [], []
162    for i in range(-10, 60):
163        ax.append(i)
164        ay.append(-10.0)
165    for i in range(-10, 60):
166        ax.append(60.0)
167        ay.append(i)
168    for i in range(-10, 61):
169        ax.append(i)
170        ay.append(60.0)
171    for i in range(-10, 61):
172        ax.append(-10.0)
173        ay.append(i)
174    for i in range(-10, 40):
175        ax.append(20.0)
176        ay.append(i)
177    for i in range(0, 40):
178        ax.append(40.0)
179        ay.append(60.0 -i)
```

Gambar 4. Inisialisasi titik *obstacle*.

Selanjutnya yakni pembuatan program inti. Ini merupakan bagian utama dari algoritma A*, dimana perhitungan untuk mencari jalur terpendek mulai dilakukan. Pada program utama ini terdapat beberapa sub-program yang akan digunakan. Beberapa sub-program tersebut meliputi *xy_index, grid_index, heuristic_value, grid_position, direction, verify_node* dan *final_path.*

Berdasarkan **Gambar 5**, pertama kita membuat sebuah fungsi dengan nama *planning*. Fungsi ini memiliki masukkan berupa parameter *self*, sx, sy, gx, gy seperti tertera pada baris 40. Kemudian kita insialisasi *start_node* dan *goal_node*. Selanjutnya kita membuat dua buah variabel untuk menyimpan antrian data dari tiap-tiap titik yang diproses. *Unvisited_list* digunakan untuk menyimpan titik yang akan dikunjungi, sedangkan *visited_list* digunakan untuk menyimpan *node* yang telah dikunjungi. Untuk eksekusi pertama, *start_node* dimasukkan ke *unvisited_list.*

Pada baris 45 dilakukan proses perulangan program. Pertama, apabila nilai pada *unvisited_list* sama dengan nol maka program akan sepenuhnya dihentikan, karena tidak ada data awal yang dapat diproses. Kemudian pada baris 48-49 dilakukan perhitungan fungsi *cost* pada *unvisited_list* menggunakan rumus: $f = g + h$. Pada baris 49, *self.heuristic_value* memiliki nilai masukan *goal_node* dan *unvisited_list*. Proses *self.heuristic_value* menghitung jarak dari *goal node* ke titik *start_node* dalam peta grid. Pada baris yang sama, *unvisited_list [o].cost* menghitung jarak dari titik *start node* ke suatu *node* yang sedang dikunjungi. Setelah fungsi *cost* selesai dihitung, nilai dari f yang paling kecil akan dijadikan *node* selanjutnya. Pada baris 50, *current* digunakan untuk mengetahui titik *node* yang saat ini sedang dikunjungi.

Pada baris 51, apabila *current node* sama dengan *goal* maka pencarian akan dihentikan. Sebaliknya, jika *current node* tidak sama dengan *goal* maka pencarian akan dilanjutkan. Proses ini akan membandingkan *node* x pada titik *goal* dengan *node* x pada *current node*. Sementara itu, *node* y pada titik *goal* akan dibandingkan dengan *node* y pada *current node*.

Pada baris 55 dilakukan penghapusan *node* pada variabel *unvisited_list* dan pada baris 56, *node* yang berhasil dihapus tersebut selanjutnya ditambahkan pada *visited_list* . Pada bagian 57-60, dilakukan perhitungan untuk memilih kemungkinan posisi berikutnya yang akan diambil. Percobaan gerakan yang dilakukan meliputi arah x (bergerak ke kanan atau kiri), arah y (bergerak ke atas atau bawah) dan arah diagonal (bergerak ke

tiap sudut). Jika ada posisi *node* yang merupakan *obstacle* maka perhitungan pada arah tersebut akan diabaikan. Selanjutnya menambahkan *children* (anak) *node* yang bebas sebagai *selected parent*. apabila *children node* berada pada *visited_list* maka *node* tersebut diabaikan dan pencarian dilakukan ke arah yang lain. Setelahnya dilakukan perhitungan nilai g,h, dan f untuk setiap *children node* yang bebas.

Pada baris 61 dibuat sebuah variabel dengan nama n_id, dengan masukan dari *node* setelah dilakukan kalkulasi *grid_index* terlebih dahulu. Pada baris 62 jika *node* dalam keadaan tidak aman, maka jangan lakukan aksi apapun dan lanjutkan proses. Pada baris 64, jika *children node* sudah pernah berada pada *visited_list*, maka jangan lakukan proses apapun. Pada baris 66, apabila *children node* tidak berada pada *unvisited_list*, maka tambahkan *children node* kedalam *unvisited_list*.

Pada baris 71-76 merupakan fungsi untuk menampilkan posisi x dan y pada peta grid, dimana input untuk setiap nilai x dilakukan kalkulasi *grid_position* terlebih dahulu dengan masukan *current.x* dan *self.min_x*. Sementara itu, untuk setiap nilai y dilakukan kalkulasi *grid_position* terlebih dahulu dengan masukan *current.y* dan *self.min_y*.

```
40  def planning(self, sx, sy, gx, gy):
41      start_node = self.Node(self.xy_index(sx, self.min_x), self.xy_index(sy, self.min_y), 0.0, -1)
42      goal_node = self.Node(self.xy_index(gx, self.min_x), self.xy_index(gy, self.min_y), 0.0, -1)
43      unvisited_list, visited_list = dict(), dict()
44      unvisited_list[self.grid_index(start_node)] = start_node
45      while True:
46          if len(unvisited_list) == 0:
47              break
48          c_id = min(
49              unvisited_list, key=lambda o: unvisited_list[o].cost + self.heuristic_value(goal_node,unvisited_list[o]))
50          current = unvisited_list[c_id]
51          if current.x == goal_node.x and current.y == goal_node.y:
52              goal_node.parent = current.parent
53              goal_node.cost = current.cost
54              break
55          del unvisited_list[c_id]
56          visited_list[c_id] = current
57          for i, _ in enumerate(self.direction):
58              node = self.Node (current.x + self.direction[i][0],
59                                current.y + self.direction[i][1],
60                                current.cost + self.direction[i][2], c_id)
61              n_id = self.grid_index(node)
62              if not self.verify_node(node):
63                  continue
64              if n_id in visited_list:
65                  continue
66              if n_id not in unvisited_list:
67                  unvisited_list[n_id] = node
68              else:
69                  if unvisited_list[n_id].cost > node.cost:
70                      unvisited_list[n_id] = node
71          if show_animation:
72              plt.plot(self.grid_position(current.x, self.min_x), self.grid_position(current.y, self.min_y), "xc")
73              plt.gcf().canvas.mpl_connect('key_release_event', lambda event: [exit(0)
74              if event.key == 'escape' else None])
75              if len(visited_list.keys()) % 10 == 0:
76                  plt.pause(0.001)
77      rx, ry = self.final_path(goal_node, visited_list)
78      return rx, ry
```

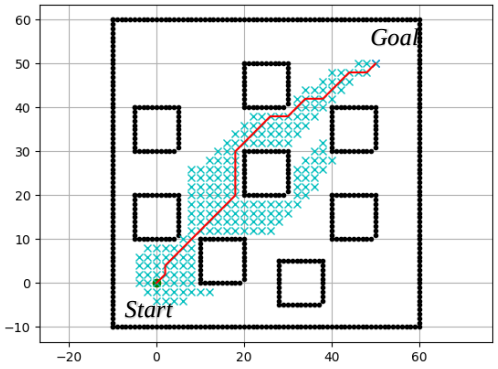
Gambar 5. Program utama.

III. HASIL DAN PEMBAHASAN

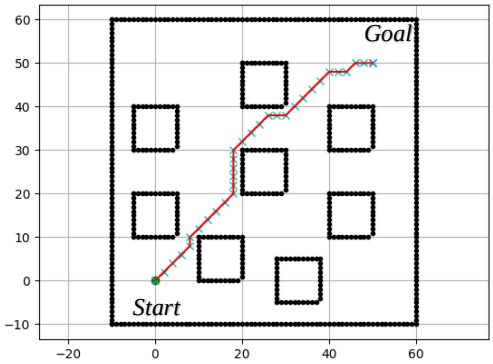
A. Perbandingan A* dan *Weighted* A*

1. Lingkungan *clutter*

Pengujian pertama dilakukan pada lingkungan *obstacle clutter*. Pada pengambilan data ini dilakukan pengujian sebanyak sepuluh kali percobaan. Data yang diambil yaitu waktu, panjang jalur yang dihasilkan, dan memori yang dipakai oleh algoritma. Pada **Gambar 6** dan **Gambar 7** diperlihatkan lingkungan pengujian *clutter*.



Gambar 6. A* pada Lingkungan pengujian *clutter*



Gambar 7. *Weighted* A* pada Lingkungan pengujian *clutter*

Pengujian *clutter* dilakukan pada *obstacle* dengan banyak rintangan yang berantakan. Titik *start* berada pada koordinat (0,0) sedangkan titik *goal* pada koordinat (50,50). **Gambar 6** merupakan proses perkembangan algoritma A* dari *node* pertama sampai *node* tersebut menjumpai titik *goal*. Sedangkan **Gambar 7** adalah proses perkembangan algoritma *Weighted* A*. Hasil pengujian lebih lanjut dapat dilihat pada **Tabel I** dan **Tabel II**.

Tabel I. A* pada lingkungan *clutter*

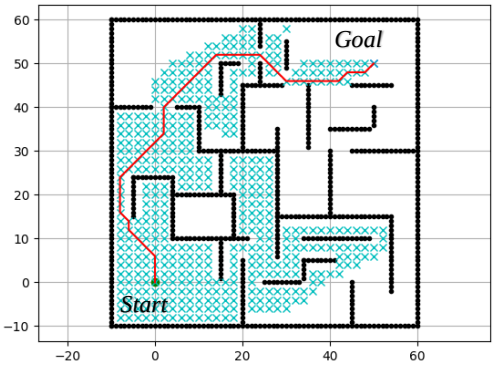
	A*		
	Path cost	*Time* (s)	*Memory* (MB)
Best	38,87	1,40	53,82
Worst	38,87	1,49	54,09
Mean	38,87	1,45	53,96

Tabel II. *Weighted A** pada lingkungan *clutter*

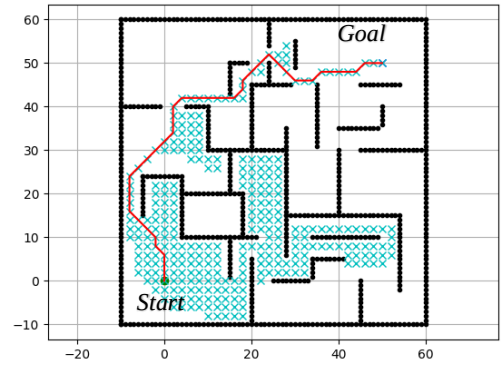
	Weighted A*		
	Path cost	*Time* (s)	*Memory* (MB)
Best	38,87	0,33	53,73
Worst	38,87	0,44	54,29
Mean	38,87	0,38	53,93

2. Lingkungan *maze*

Pada percobaan kedua ini dilakukan pengujian algoritma pada lingkungan *maze* (labirin). Titik *start* berada di koordinat (0,0) dan titik *goal* berada di koordinat (50,50). Untuk lebih lengkapnya dapat dilihat pada **Gambar 8** dan **Gambar 9**.



Gambar 8. A* pada Lingkungan pengujian *maze*



Gambar 9. *Weighted* A* pada Lingkungan pengujian *maze*

Hasil yang diperoleh dari algoritma A* pada lingkungan *maze* seperti yang diperlihatkan pada **Gambar 8** yaitu, pertumbuhan jalur sedikit lebih banyak apabila dibandingkan dengan *Weighted* A* pada **Gambar 9,** namun untuk *Weighted* A* ada sedikit pengambilan jalur yang berbeda dimulai dari koordinat (5,40) dan berakhir di koordinat (25,55). Disini A* lebih memilih jalur yang mengarah ke atas, lalu berbelok. Sedangkan pada *Weighted* A* langsung berbelok arah ketika ditemukan adanya celah kosong yang mengarah ke tujuan. Hasil data pada percobaan ini ada pada **Tabel III** dan **Tabel IV**.

Tabel III. A* pada lingkungan *maze*

	A*		
	Path cost	*Time* (s)	*Memory* (MB)
Best	52,28	4,16	53,76
Worst	52,28	5,02	54,10
Mean	52,28	4,73	53,94

Tabel IV. *Weighted A** pada lingkungan *maze*

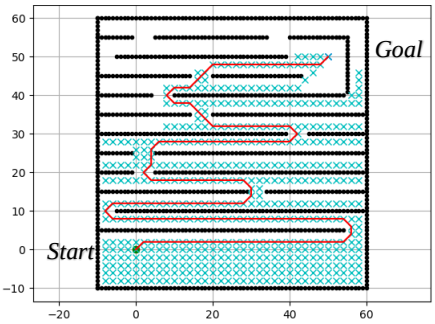
	Weighted A*		
	Path cost	*Time* (s)	*Memory* (MB)
Best	52,87	2,44	53,70
Worst	52,87	2,80	54,15
Mean	52,87	2,61	53,95

Meskipun A* terlihat lebih banyak melakukan percabangan *node* saat bekerja, namun *path cost* yang didapat memperlihatkan kondisi yang optimal. Berdasarkan **Tabel III**, diperlihatkan bahwa nilai *path cost* Algoritma A* yang didapat sedikit lebih baik apabila dibandingkan dengan *Weighted* A*. Namun apabila kita melihat waktu pemrosesan, *Weighted* A* jauh meninggalkan A* hampir dua kali lipat untuk perolehan waktu terbaiknya. Disini dapat dilihat bahwa A* tidak begitu unggul dalam masalah waktu namun jalur yang didapat akan seoptimal mungkin. *Weighted* A* sebaliknya, pada permasalahan waktu algoritma ini bisa sangat cepat dalam pencarian jalur, namun solusi yang diberikan mungkin tidak cukup optimal.

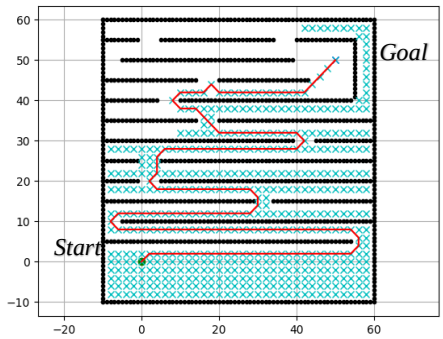
3. Lingkungan *narrow*

Narrow merupakan lingkungan pengujian dengan bidang sisi yang sempit. Pengujian ini dilakukan dengan menempatkan *obstacle* memanjang dan berjajar. Ruang untuk *node* lewat hanya diperbolehkan satu atau dua *node* saja. Titik *goal* ditempatkan disudut *obstacle* dengan beberapa penambahan *obstacle* yang persegi. Hasil visualisasi dapat dilihat pada **Gambar 10** dan **Gambar 11**.

Berdasarkan **Gambar 10** dan **Gambar 11** hanya ada satu jalan yang bisa dilewati saat pertama kali *node* berkembang, yaitu jalan yang berada di sebelah kanan. kemudian memutar ke sebelah kiri dan seterusnya. Pada saat mendekati titik *goal* atau tepatnya berada di koordinat (20,35) jalan yang dapat dilalui ada dua, diperbolehkan ke kanan maupun ke kiri. Berdasarkan **Gambar 10** *node* yang bergerak ke sebelah kanan cenderung lebih sedikit apabila dibandingkan dengan *Weighted* A* pada **Gambar 11**. selain itu, pada *Weighted* A* terdapat jalur yang sedikit tidak optimal, tepatnya pada koordinat (18,45). *Node* yang dikembangkan memilih bergerak ke arah diagonal atas daripada bergerak ke arah kanan. Ketika jalur ini dipilih sebagai jalur akhir, akibatnya jalur yang akan dilalui akan bergerak terlebih dahulu keatas kemudian kembali lagi kebawah. **Tabel V** dan **Tabel VI** memperlihatkan data dari percobaan ini.



Gambar 10. A* pada Lingkungan pengujian *narrow*



Gambar 11. *Weighted* A* pada Lingkungan pengujian *narrow*

Berdasarkan **Gambar 10** dan **Gambar 11** hanya ada satu jalan yang bisa dilewati saat pertama kali *node* berkembang, yaitu jalan yang berada di sebelah kanan. kemudian memutar ke sebelah kiri dan seterusnya. Pada saat mendekati titik *goal* atau tepatnya berada di koordinat (20,35) jalan yang dapat dilalui ada dua, diperbolehkan ke kanan maupun ke kiri. Berdasarkan **Gambar 10** *node* yang bergerak ke sebelah kanan cenderung lebih sedikit apabila dibandingkan dengan *Weighted* A* pada **Gambar 11**. selain itu, pada *Weighted* A* terdapat jalur yang sedikit tidak optimal, tepatnya pada koordinat (18,45). *Node* yang dikembangkan memilih bergerak ke arah diagonal atas daripada bergerak ke arah kanan. Ketika jalur ini dipilih sebagai jalur akhir, akibatnya jalur yang akan dilalui akan bergerak terlebih dahulu keatas kemudian kembali lagi kebawah. **Tabel V** dan **Tabel VI** memperlihatkan data dari percobaan ini.

Tabel V. A* pada lingkungan *narrow*

	A*		
	Path cost	*Time* (s)	*Memory* (MB)
Best	163,69	4,45	53,50
Worst	163,69	4,59	54,10
Mean	163,69	4,50	53,74

Tabel VI. *Weighted A** pada lingkungan *narrow*

	Weighted A*		
	Path cost	*Time* (s)	*Memory* (MB)
Best	164,52	4,20	53,60
Worst	164,52	4,41	54,04
Mean	164,52	4,33	53,85

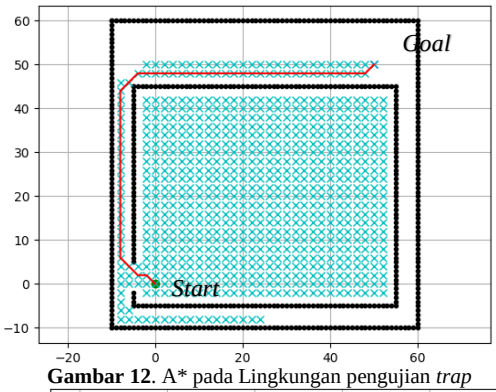
Waktu terbaik yang diperoleh dalam percobaan ini tidak terpaut jauh. Berdasarkan **Tabel V** dan **Tabel VI**, *Weighted* A* memperoleh waktu terbaik di 4,20 detik, sementara waktu eksekusi A* pada lingkungan *narrow* sebesar 4,45 detik. *Path cost* yang diperoleh pada lingkungan *narrow* ini menghasilkan nilai optimal lebih tinggi untuk algoritma A* dibanding *Weighted* A*.

4. Lingkungan *trap*

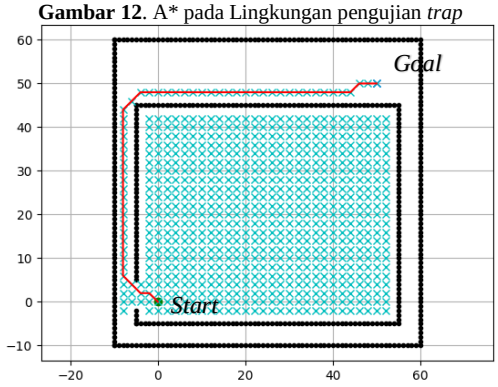
Lingkungan *trap* merupakan area pencarian dimana salah satu dari titik *start* atau titik *goal* ada disebuah area yang hampir tertutup. Area ini biasanya berbentuk persegi dimana hanya ada satu jalan untuk masuk dan keluar. Jalan yang akan dilaluinya pun cukup sempit namun satu atau dua *node* masih bisa keluar masuk area ini. Hasil percobaan ini dapat dilihat pada **Gambar 12** dan **Gambar 13**.

Seperti yang terlihat **Gambar 10** dan **Gambar 11** *node* yang berkembang sepenuhnya memenuhi area pengujian baik untuk A* pada **Gambar 10** maupun *Weighted* A* pada **Gambar 11.** Pada A* *node* masih sedikit berkembang kearah bawah sedangkan *Weighted* A* cenderung datar pada saat keluar dari lingkungan *trap*. Hasil dari data yang didapat baik untuk A* maupun *Weighted* A* bisa dikatakan mirip. Hasil pengujian dapat dilihat pada **Tabel VII** dan **Tabel VIII**.

Berdasarkan **Tabel VII** dan **Tabel VIII** nilai *path cost* yang didapat untuk kedua algoritma sama yaitu 54,48. Baik A* maupun *Weighted* A* untuk masalah waktu yang dihabiskan dalam lingkungan trap merupakan yang paling tinggi dari semua percobaan yang telah dilakukan. Ini karena kedua algoritma membuat *node* baru berdasarkan nilai $f(n)$ terkecil. Dimana nilai *node* terkecil ini mengarah ke dalam area trap, yang pada ujungnya terhalang oleh *obstacle*



Gambar 12. A* pada Lingkungan pengujian *trap*



Gambar 11. *Weighted* A* pada Lingkungan pengujian *trap*

Tabel VII. A* pada lingkungan *trap*

	A*		
	Path cost	*Time* (s)	*Memory* (MB)
Best	54,48	7,97	53,85
Worst	54,48	8,22	54,09
Mean	54,48	8,07	53,96

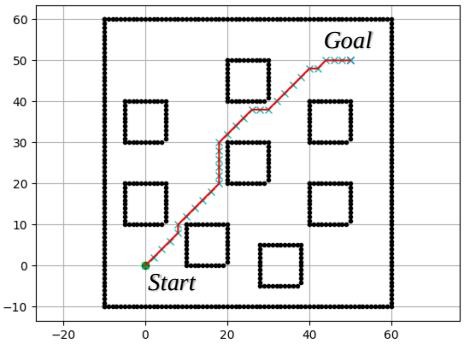
Tabel VIII. *Weighted* A* pada lingkungan *trap*

	Weighted A*		
	Path cost	*Time* (s)	*Memory* (MB)
Best	54,48	7,31	53,66
Worst	54,48	7,97	54,09
Mean	54,48	7,47	53,87

B. Pengujian perubahan bobot pada *Weighted* A*

1. Lingkungan *clutter*

Selanjutnya dilakukan pengujian terhadap perubahan nilai bobot pada fungsi heuristik. Nilai yang diberikan dari 2 sampai 5. Hasil pengujian dapat terlihat pada **Gambar 12**.



Gambar 12. *Weighted* A* pada lingkungan *clutter*

Pada **Gambar 12** diperlihatkan hasil dari penggunaan nilai bobot=3. Dengan memberikan nilai 3, jalur sedikit berbelok sebelum titik *goal* ditemukan. Hasil data yang didapat bisa dilihat pada **Tabel IX**, **Tabel X**, dan **Tabel XI**.

Tabel IX. Pengujian *path cost* pada lingkungan *clutter*

	Path cost		
Bobot	*Best*	*Worst*	*Mean*
W=2	38,87	38,87	38,87
W=3	38,87	38,87	38,87
W=4	38,87	38,87	38,87
W=5	38,87	38,87	38,87

Tabel X. Pengujian waktu pada lingkungan *clutter*

	Time (s)		
Bobot	*Best*	*Worst*	*Mean*
W=2	0,33	0,44	0,38
W=3	0,34	0,38	0,37
W=4	0,34	0,38	0,36
W=5	0,36	0,39	0,37

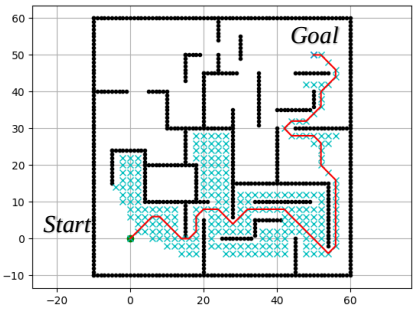
Tabel XI. Pengujian *memory* pada lingkungan *clutter*

	Memory (MB)		
Bobot	*Best*	*Worst*	*Mean*
W=2	53,73	54,29	53,93
W=3	53,70	54,21	53,92
W=4	53,68	54,10	53,91
W=5	53,76	54,01	53,90

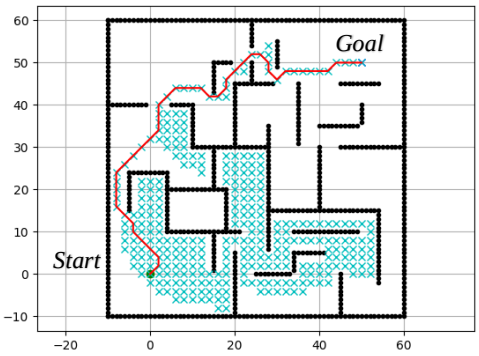
Berdasarkan **Tabel IX** *path cost* yang didapatkan sama untuk semua nilai W yang berbeda. Pada kasus lingkungan *clutter* waktu yang diperoleh dari pemberian nilai W yang berbeda berdampak pada waktu proses tercepat yang semakin menurun saat W semakin besar. Namun waktu rata-rata yang diperoleh dari 10 kali percobaan, didapat adanya penurunan waktu proses saat nilai W semakin besar. Penggunaan memori untuk algoritma ini ada dibawah 54MB untuk data terbaik dan dibawah 55MB untuk data terburuknya.

2. Lingkungan *maze*

Pada lingkungan *maze* disajikan juga bagaimana *Weighted* A* bekerja. Dari hasil pengujian didapatkan jalur yang berbeda arah, untuk nilai W yang berbeda. *Path cost* yang didapat memiliki nilai yang berbeda pula. Semakin tinggi nilai W, *path cost* yang didapat semakin besar. **Gambar 11** menunjukkan hasil *Weighted* A* pada lingkungan *maze*.



Gambar 13. *Weighted* A* pada lingkungan *maze*



Gambar 14. *Weighted* A* pada lingkungan *maze*

Terlihat pada **Gambar 13** dan **Gambar 14** jalur yang dibuat berbeda arah, yang satu berjalan ke arah atas untuk membuat jalur, kemudian satu lagi bergerak ke arah bawah. **Gambar 13** mempunyai bobot W dengan nilai 3, sementara pada **Gambar 14** nilai yang diberikan adalah 5. Hasil pengujian secara lengkap bisa dilihat pada **Tabel XII**, **Tabel XIII**, dan **Tabel XIV**.

Tabel XII. Pengujian *path cost* pada lingkungan *maze*

	Path cost		
Bobot	*Best*	*Worst*	*Mean*
W=2	52,87	52,87	52,87
W=3	55,11	55,11	55,11
W=4	74,42	74,42	74,42
W=5	74,42	74,42	74,42

Tabel XIII. Pengujian waktu pada lingkungan *maze*

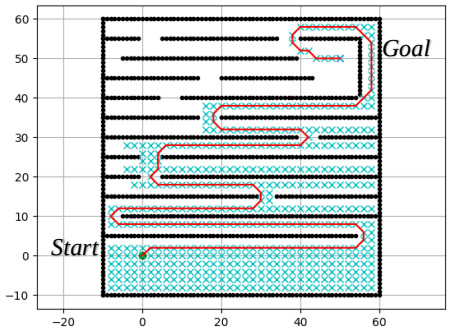
	Time (s)		
Bobot	*Best*	*Worst*	*Mean*
W=2	2,44	2,80	2,61
W=3	2,41	2,57	2,47
W=4	2,25	2,52	2,32
W=5	1,83	1,98	1,90

Tabel XIV. Pengujian *memory* pada lingkungan *maze*

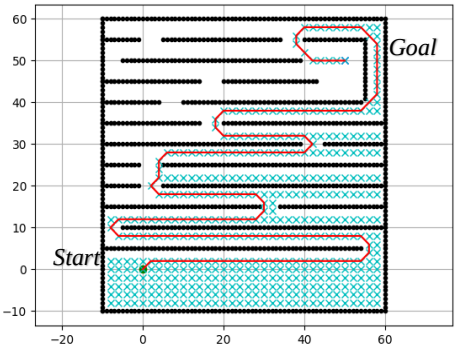
	Memory (MB)		
Bobot	*Best*	*Worst*	*Mean*
W=2	53,70	54,15	53,93
W=3	53,76	54,14	53,95
W=4	53,77	54,08	53,91
W=5	53,83	54,06	53,92

3. Lingkungan *narrow*

Pada lingkungan *narrow*, waktu yang diperlukan *Weighted* A* untuk menyelesaikan eksekusi program sedikit agak lama. 4,20 detik untuk waktu tercepat dengan rata-rata sebesar 4.33 detik. Hasil pengujian dapat dilihat pada **Gambar 15** dan **Gambar 16**



Gambar 15. *Weighted* A* pada lingkungan *narrow*



Gambar 16. *Weighted* A* pada lingkungan *narrow*

Nilai W yang diberikan untuk **Gambar 15** yaitu W=3. Jalur yang didapatkan yaitu bergerak kebagian atas. **Gambar 16** juga menghasilkan arah jalur yang sama yaitu berbelok ke kanan pada koordinat (18,35) lalu kearah sumbu Y sampai menemukan titik akhir. Untuk hasil pengujian bisa dilihat pada **Tabel XV**, **Tabel XVI**, dan **Tabel XVII**.

Tabel XV. Pengujian *path cost* pada lingkungan *narrow*

	Path cost		
Bobot	*Best*	*Worst*	*Mean*
W=2	164,52	164,52	164,52
W=3	181,69	181,69	181,69
W=4	181,69	181,69	181,69
W=5	181,69	181,69	181,69

Tabel XVI. Pengujian waktu pada lingkungan *narrow*

	Time (s)		
Bobot	*Best*	*Worst*	*Mean*
W=2	4,20	4,41	4,33
W=3	4,00	4,16	4,07
W=4	3,69	3,85	3,78
W=5	3,65	3,84	3,76

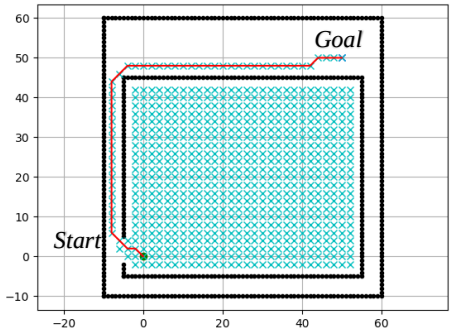
Tabel XVII. Pengujian *memory* pada lingkungan *narrow*

	Memory (MB)		
Bobot	*Best*	*Worst*	*Mean*
W=2	53,60	54,04	53,85
W=3	53,76	54,09	53,88
W=4	53,76	54,13	53,96
W=5	53,68	54,12	53,93

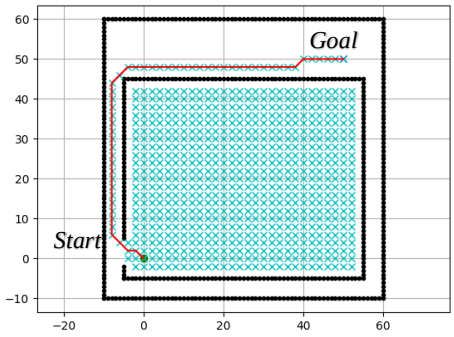
Pada percobaan di lingkungan ini, nilai *path cost* yang didapat cukup besar yaitu ada di angka 164,52-181,69. Ini karena jalur yang dilewati berkelok kelok, serta hanya ada satu jalur saja pada saat awal pencarian dimulai. Akibat dari jalur yang berbelok, mempengaruhi juga kepada lamanya waktu penyelesaian. Contohnya dari nilai W=4 dengan W=5, perbedaan waktu yang didapat hanya 0,04 detik saja. Tidak menunjukkan adanya perbedaan waktu yang terpaut jauh dari perubahan nilai W pada proses ini.

4. Lingkungan *trap*

Pengujian *Weighted* A* pada lingkungan *trap* disajikan pada **Gambar 17** dan **Gambar 18**. pada percobaan kali ini, *node* yang terbentuk memenuhi semua area lingkungan *trap* untuk semua nilai W yang diujikan. Pencarian jalur yang berawal dari titik *start* bergerak ke sudut kanan atas ruang pencarian. setelah itu *node* yang terbentuk bergerak kebawah mendekati titik dan keluar area lingkungan *trap*. Setelah itu *node* bergerak menuju sumbu Y dan menemukan titik akhir pencarian.



Gambar 17. *Weighted* A* pada Lingkungan pengujian *trap*



Gambar 18. *Weighted* A* pada Lingkungan pengujian *trap*

Berdasarkan **Gambar 17** dan **Gambar 18** hasil yang terbentuk memiliki kesamaan jalur. Perubahan nilai W=3 pada **Gambar 17** menghasilkan jalur yang tidak terlalu berbeda dengan nilai W=5 pada **Gambar 18**. Waktu yang dihasilkan juga tidak terlalu baik untuk pengujian pada lingkungan *trap* ini. Hasil pengujian dapat dilihat pada **Tabel XVIII**, **Tabel XIX**, dan **Tabel XX**.

Tabel XVIII. Pengujian *path cost* pada lingkungan *trap*

	Path cost		
Bobot	*Best*	*Worst*	*Mean*
W=2	54,48	54,48	54,48
W=3	54,48	54,48	54,48
W=4	54,48	54,48	54,48
W=5	54,48	54,48	54,48

Tabel XIX. Pengujian waktu pada lingkungan *trap*

	Time (s)		
Bobot	*Best*	*Worst*	*Mean*
W=2	7,31	7,97	7,47
W=3	7,10	7,37	7,24
W=4	7,15	7,34	7,22
W=5	7,09	7,37	7,23

Tabel XX. Pengujian *memory* pada lingkungan *trap*

	Memory (MB)		
Bobot	*Best*	*Worst*	*Mean*
W=2	53,66	54,09	53,87
W=3	53,62	53,98	53,86
W=4	53,69	53,98	53,86
W=5	53,76	54,09	53,95

Berdasarkan pengujian yang telah dilakukan, diperoleh hasil dimana perubahan nilai bobot heuristik untuk pengujian pada lingkungan *trap* kurang menunjukkan adanya peningkatan yang signifikan. Ini dilihat dari waktu yang dibutuhkan saat memproses program dari awal sampai akhir. Waktu yang didapat dari percobaan pada lingkungan *trap* merupakan waktu yang paling lama dari beberapa percobaan yang telah dilakukan. 7,97 detik untuk waktu yang paling lama, 7,09 untuk waktu terbaik, dan rata-rata dengan nilai terendah ada di angka 7,22 detik.

Christopher dalam penelitiannya menyebutkan bahwa meskipun dibanyak domain ada tren umum dimana bobot yang lebih besar di *Weighted* A* mengarah ke pencarian lebih cepat, ada juga domain yang bobotnya lebih besar mengarah ke pencarian yang lebih lambat [35]. Seperti yang telah di paparkan sebelumnya bahwa waktu yang dihasilkan algoritma *Weighted* A* pada lingkungan pengujian *trap* hampir tidak menunjukkan perbedaan yang terpaut jauh dengan algoritma A*. Kontribusi yang dapat diberikan penulis dari hasil penelitian ini yaitu memberikan pengetahuan tentang hasil penelitian yang telah dilakukan baik kepada *engineer*, maupn pembaca.

IV.KESIMPULAN

Berdasarkan hasil pengujian didapatkan algoritma *Weighted* A* menghasilkan waktu pencarian yang lebih baik daripada algoritma A*. Tetapi algoritma A* menghasilkan jalur yang lebih optimal dibandingkan algoritma *Weighted* A*. Dengan strategi yang lebih menekankan pemilihan *node* yang lebih dekat dengan *node goal*, maka *Weighted* A* dapat menghasilkan jalur dengan waktu komputasi yang lebih cepat. Sedangkan algoritma A* karena memilih *node* dengan nilai heuristik terkecil, maka dapat menghasilkan jalur yang lebih optimal. Penulis sangat mengharapkan pengembangan lebih lanjut dari penelitian ini yaitu dapat diimplementasikan pada *hardware.*

DAFTAR PUSTAKA

[1] L. Chen, Y. Shan, W. Tian, B. Li, dan D. Cao, "A Fast and Efficient Double-Tree RRT*-Like Sampling-Based Planner Applying on Mobile Robotic Systems," *IEEE/ASME Trans. Mechatronics*, vol. 23, no. 6, hal. 2568–2578, 2018.

[2] L. Janson, B. Ichter, dan M. Pavone, "Deterministic Sampling-Based Motion Planning: Optimality, Complexity, and Performance," *Springer Proc. Adv. Robot.*, vol. 3, hal. 507–525, 2018.

[3] M. P. Mann, L. Damti, dan D. Zarrouk, "Minimally actuated hyper-redundant robots: Motion planning methods based on fractals and self-organizing systems," *Int. J. Adv. Robot. Syst.*, vol. 16, no. 2, hal. 1–16, 2019.

[4] M. Aria, "Algoritma Perencanaan Jalur Kendaraan Otonom di Lingkungan Perkotaan dari Sudut Pandang Filosofi Kuhn dan Filosofi Popper," *Telekontran J. Ilm. Telekomun. Kendali dan Elektron. Terap.*, vol. 7, no. 2, hal. 145–156, 2020.

[5] N. A. M. Sabri, A. S. H. Basari, B. Husin, dan K. A. F. A. Samah, "The utilisation of dijkstra's algorithm to assist evacuation route in higher and close building," *J. Comput. Sci.*, vol. 11, no. 2, hal. 330–336, 2015.

[6] B. Popa dan D. Popescu, "Analysis of algorithms for shortest path problem in parallel," *Proc. 2016 17th Int. Carpathian Control Conf. ICCC 2016*, no. April 2018, hal. 613–617, 2016.

[7] A. Candra, M. A. Budiman, dan K. Hartanto, "Dijkstra's and A-Star in Finding the Shortest Path: A Tutorial," *2020 Int. Conf. Data Sci. Artif. Intell. Bus. Anal. DATABIA 2020 - Proc.*, hal. 28–32, 2020.

[8] J. H. Zhou, J. Q. Zhou, Y. S. Zheng, dan B. Kong, "Research on path planning algorithm of intelligent mowing robot used in large airport lawn," *Proc. - 2016 Int. Conf. Inf. Syst. Artif. Intell. ISAI 2016*, hal. 375–379, 2017.
[9] A. K. Guruji, H. Agarwal, dan D. K. Parsediya, "Time-efficient A* Algorithm for Robot Path Planning," *Procedia Technol.*, vol. 23, hal. 144–149, 2016.
[10] S. K. Sharma dan B. L. Pal, "Shortest Path Searching for Road Network using A * Algorithm," *Int. J. Comput. Sci. Mob. Compfile*, vol. 4, no. 7, hal. 513–522, 2015.
[11] J. T. Thayer and W. Ruml, "Faster than weighted A*: An optimistic approach to bounded suboptimal search," *ICAPS 2008 - Proc. 18th Int. Conf. Autom. Plan. Sched.*, hal. 355–362, 2008.
[12] E. A. Hansen dan R. Zhou, "Anytime heuristic search," *J. Artif. Intell. Res.*, vol. 28, hal. 267–297, 2009.
[13] K. R. Srinath, "Python – The Fastest Growing Programming Language," *Int. Res. J. Eng. Technol.*, hal. 354–357, 2017.
[14] A. Bogdanchikov, M. Zhaparov, dan R. Suliyev, "Python to learn programming," *J. Phys. Conf. Ser.*, vol. 423, no. 1, hal. 1–5, 2013.
[15] P. N. Siva dan R. Yamaganti, "A Review on Python for Data Science, Machine Learning and IOT," *Int. J. Sci. Eng. Res.*, vol. 10, no. 12, hal. 851–858, 2019.
[16] N. Thaker dan A. Shukla, "Python as Multi Paradigm Programming Language," *Int. J. Comput. Appl.*, vol. 177, no. 31, hal. 38–42, 2020.
[17] L. Chen, B. Xu, T. Zhou, dan X. Zhou, "A constraint based bug checking approach for python," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 2, hal. 306–311, 2009.
[18] T. Nayl, M. Q. Mohammed, dan S. Q. Muhamed, "Obstacles Avoidance for an Articulated Robot Using Modified Smooth Path Planning," *2017 Int. Conf. Comput. Appl. ICCA 2017*, hal. 185–189, 2017.
[19] Z. Zhu, J. Xiao, J. Q. Li, F. Wang, dan Q. Zhang, "Global path planning of wheeled robots using multi-objective memetic algorithms," *Integr. Comput. Aided. Eng.*, vol. 22, no. 4, hal. 387–404, 2015.
[20] E. Hernandez, M. Carreras, dan P. Ridao, "A comparison of homotopic path planning algorithms for robotic applications," *Rob. Auton. Syst.*, vol. 64, hal. 44–58, 2015.
[21] J. Sa, K. Lim, dan J. Kim, "The research of Multi-Layer-Based on Path Planning for generating optimal path," *2016 IEEE Transp. Electrif. Conf. Expo, Asia-Pacific, ITEC Asia-Pacific 2016*, hal. 896–899, 2016.
[22] I. Noreen, A. Khan, dan Z. Habib, "Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 11, hal. 97–107, 2016
[23] M. Kobilarov, "Cross-entropy randomized motion planning," *Robot. Sci. Syst.*, vol. 7, no. 3, hal. 153–160, 2012.
[24] I. Noreen, A. Khan, dan Z. Habib, "A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 16, no. 10, hal. 20–27, 2016.
[25] B. Cohen, S. Chitta, dan M. Likhachev, "Single- and dual-arm motion planning with heuristic search," *Int. J. Rob. Res.*, vol. 33, no. 2, hal. 305–320, 2014.
[26] C. Hernández dan J. A. Baier, "Escaping heuristic depressions in real-time heuristic search," *10th Int. Conf. Auton. Agents Multiagent Syst. 2011, AAMAS 2011*, vol. 2, hal. 1197–1198, 2011.
[27] B. Bonet dan H. Geffner, "Faster heuristic search algorithms for planning with uncertainty and full feedback," *IJCAI Int. Jt. Conf. Artif. Intell.*, hal. 1233–1238, 2003.
[28] G. Röger dan M. Helmert, "The more, the merrier: Combining heuristic estimators for satisficing planning," *ICAPS 2010 - Proc. 20th Int. Conf. Autom. Plan. Sched.*, no. ICAPS, hal. 246–249, 2010.
[29] J. Virseda, D. Borrajo, dan V. Alc´azar, "Learning heuristic functions for cost-based planning," *Prepr. ICAPS'13 PAL Work. Plan. Learn.*, hal. 6–13, 2013.
[30] Y. F. Yiu, J. Du, dan R. Mahapatra, "Evolutionary Heuristic A∗ Search: Heuristic Function Optimization via Genetic Algorithm," *Proc. - 2018 1st IEEE Int. Conf. Artif. Intell. Knowl. Eng. AIKE 2018*, hal. 25–32, 2018.
[31] T. J. Misa, "Interview: An interview with Edsger W. Dijkstra," *Commun. ACM*, vol. 53, no. 8, hal. 41–47, 2010.
[32] S. Baldi, N. Maric, R. Dornberger, dan T. Hanne, "Pathfinding optimization when solving the paparazzi problem comparing A∗ and Dijkstra's algorithm," *Proc. - 6th Int. Symp. Comput. Bus. Intell. ISCBI 2018*, hal. 16–22, 2019.
[33] F. Duchon *et al.*, "Path planning with modified A star algorithm for a mobile robot," *Procedia Eng.*, vol. 96, hal. 59–69, 2014.
[34] X. Cui dan H. Shi, "A * -based Pathfinding in Modern Computer Games," *Int. J. Comput. Sci. Netw. Secur.*, vol. 11, no. 1, hal. 125–130, 2011.
[35] C. Wilt dan W. Ruml, "When does weighted A* fail?," *Proc. 5th Annu. Symp. Comb. Search, SoCS 2012*, pp. 137–144, 2012.