

Penerapan Algoritma Heuristic Search pada Smart Thermostat Agent

Muchammad Ivan Maulana^{*1}, Anang Kukuh Adisusilo², Nonot Wisnu Karyanto³

^{1,2,3} Program Studi Informatika, Unniversitas Wijaya Kusuma Surabaya
Email: ¹mim063@mhs.uwks.ac.id, ²anang65@uwks.ac.id, ³nonotwk@uwks.ac.id
^{*}Penulis Korespondensi

Abstrak

Smart thermostat merupakan sistem pengendali suhu cerdas yang mampu mengatur temperatur ruangan secara otomatis berdasarkan kondisi lingkungan dan preferensi pengguna. Penggunaan sistem pendingin dan pemanas ruangan yang tidak efisien menyebabkan tingginya konsumsi energi pada bangunan modern sehingga diperlukan solusi berbasis kecerdasan buatan untuk meningkatkan efisiensi pengendalian HVAC (Heating, Ventilation, and Air Conditioning). Penelitian ini bertujuan untuk menerapkan dan membandingkan beberapa algoritma pencarian pada smart thermostat agent, yaitu Breadth-First Search (BFS), Depth-First Search (DFS), Generate and Test, Hill Climbing, Greedy Best-First Search, dan A* Algorithm. Metode penelitian dilakukan dengan memodelkan smart thermostat sebagai intelligent agent menggunakan task environment PEAS serta membangun state space graph dari kondisi suhu ruangan menuju suhu target 24°C. Hasil pengujian menunjukkan bahwa algoritma A* menghasilkan solusi paling optimal karena mampu menemukan jalur terpendek dengan mempertimbangkan fungsi biaya dan heuristik secara bersamaan. BFS juga menghasilkan solusi optimal namun membutuhkan penggunaan memori lebih besar. DFS memiliki proses pencarian lebih cepat tetapi tidak menjamin optimalitas solusi. Sementara itu, Hill Climbing dan Greedy Best-First Search lebih efisien dalam waktu pencarian namun rentan terhadap local optimum. Berdasarkan hasil penelitian, algoritma A* direkomendasikan sebagai metode terbaik untuk pengambilan keputusan pada smart thermostat agent.[1]–[5].

Kata kunci: A*, BFS, DFS, heuristic search, smart thermostat

Abstract

Smart thermostat is an intelligent temperature control system capable of automatically regulating room temperature based on environmental conditions and user preferences. Inefficient use of heating and cooling systems has led to high energy consumption in modern buildings, creating the need for artificial intelligence-based solutions to improve the efficiency of HVAC (Heating, Ventilation, and Air Conditioning) control systems. This study aims to implement and compare several search algorithms in a smart thermostat agent, namely Breadth-First Search (BFS), Depth-First Search (DFS), Generate and Test, Hill Climbing, Greedy Best-First Search, and the A* Algorithm. The research method was conducted by modeling the smart thermostat as an intelligent agent using the PEAS task environment and constructing a state space graph from room temperature conditions toward the target temperature of 24°C. The experimental results show that the A* algorithm produces the most optimal solution because it can determine the shortest path by simultaneously considering cost and heuristic functions. BFS also produces optimal solutions but requires larger memory usage. DFS provides a faster search process but does not guarantee solution optimality. Meanwhile, Hill Climbing and Greedy Best-First Search are more efficient in search time but are vulnerable to local optimum problems. Based on the research findings, the A* algorithm is recommended as the best method for decision-making in smart thermostat agents.[1]–[5].

Keywords: A*, BFS, DFS, heuristic search, smart thermostat.

I. PENDAHULUAN

1.1. Latar Belakang

Peningkatan penggunaan sistem HVAC pada bangunan modern menyebabkan konsumsi energi listrik mengalami peningkatan signifikan setiap tahun. Sistem pendingin dan pemanas ruangan yang bekerja secara konvensional sering kali tidak mampu menyesuaikan kondisi lingkungan secara adaptif sehingga menyebabkan pemborosan energi. Teknologi smart thermostat hadir sebagai solusi pengendalian suhu otomatis yang mampu meningkatkan efisiensi energi melalui pemanfaatan sensor dan kecerdasan buatan.[2], [5]

Perkembangan artificial intelligence memungkinkan smart thermostat bekerja sebagai intelligent agent yang dapat mengambil keputusan berdasarkan kondisi lingkungan secara real-time. Sistem ini dapat

memonitor suhu ruangan, keberadaan pengguna, serta status perangkat pendingin atau pemanas untuk menentukan tindakan yang paling optimal. Implementasi algoritma pencarian heuristik pada smart thermostat mampu membantu proses pengambilan keputusan agar sistem mencapai suhu nyaman dengan biaya dan waktu minimum.[1], [3]

Algoritma heuristic search seperti Hill Climbing, Greedy Best-First Search, dan A* banyak digunakan dalam penyelesaian masalah pencarian jalur optimal karena mampu mengurangi kompleksitas pencarian dibandingkan metode brute force. Selain itu, algoritma BFS dan DFS juga sering digunakan sebagai dasar evaluasi performa algoritma pencarian lainnya. Kombinasi algoritma tersebut menjadi penting untuk dianalisis pada sistem smart thermostat guna menentukan metode paling efektif dalam pengendalian suhu otomatis.[1], [4]

1.2. Identifikasi Masalah

Berdasarkan latar belakang penelitian, terdapat beberapa permasalahan yang dapat diidentifikasi. Pertama, bagaimana memodelkan smart thermostat sebagai task environment pada intelligent agent AI sehingga sistem mampu melakukan pengambilan keputusan secara otomatis.[4]

Kedua, bagaimana menentukan algoritma pencarian yang paling optimal dalam mencapai suhu nyaman pada smart thermostat agent dengan mempertimbangkan efisiensi langkah, optimalitas solusi, dan penggunaan memori.[1]

Ketiga, bagaimana membandingkan performa algoritma BFS, DFS, Generate and Test, Hill Climbing, Greedy Best-First Search, dan A* dalam menyelesaikan permasalahan pengaturan suhu ruangan pada smart thermostat.[2], [5]

1.3. Tujuan Penelitian

Penelitian ini bertujuan untuk merancang model smart thermostat agent menggunakan task environment yang terdiri atas percept, action, goal, dan environment sehingga sistem dapat bekerja secara cerdas dalam mengatur suhu ruangan.[4]

Selain itu, penelitian ini bertujuan untuk mengimplementasikan dan membandingkan algoritma BFS, DFS, Generate and Test, Hill Climbing, Greedy Best-First Search, dan A* pada sistem smart thermostat guna mengetahui karakteristik masing-masing algoritma.[1]

Tujuan akhir penelitian adalah menentukan algoritma yang paling optimal dalam mencapai suhu nyaman ruangan pada rentang 22–24°C dengan efisiensi pencarian terbaik.[2], [3]

1.4. Penelitian Terkait

Penelitian mengenai heuristic search algorithm telah banyak dilakukan dalam berbagai bidang artificial intelligence. Penelitian oleh Bulletin of the National Research Centre membandingkan performa algoritma BFS, DFS, dan A* pada penyelesaian permasalahan pencarian jalur dan menunjukkan bahwa algoritma A* memiliki tingkat optimalitas lebih tinggi dibandingkan algoritma lainnya.[1]

Penelitian lain yang dilakukan pada sistem HVAC berbasis smart thermostat menunjukkan bahwa penerapan heuristic algorithm mampu meningkatkan efisiensi konsumsi energi melalui pengambilan keputusan otomatis berbasis kondisi lingkungan.[2]

Studi mengenai AI-based occupant-centric HVAC control system menjelaskan bahwa penggunaan AI pada pengaturan suhu ruangan mampu meningkatkan kenyamanan pengguna sekaligus mengurangi konsumsi energi bangunan secara signifikan.[5]

Perbedaan penelitian ini dibandingkan penelitian sebelumnya terletak pada penerapan beberapa algoritma pencarian sekaligus dalam satu smart thermostat agent sehingga memungkinkan analisis komprehensif mengenai performa setiap algoritma dalam menyelesaikan masalah pengaturan suhu ruangan.[1]–[5]

II. METODE PENELITIAN

2.1. Desain Task Environment Smart Thermostat

Pada penelitian ini, smart thermostat dimodelkan sebagai intelligent agent yang bekerja berdasarkan konsep task environment dalam kecerdasan buatan. Model task environment dirancang menggunakan pendekatan PEAS (Performance Measure, Environment, Actuator, Sensor) sehingga agent mampu melakukan pengambilan keputusan secara otomatis berdasarkan kondisi lingkungan.[4]

Parameter percept pada sistem terdiri atas suhu ruangan, suhu target, status AC atau heater, keberadaan orang di dalam ruangan, serta waktu pengoperasian sistem. Data percept diperoleh melalui sensor virtual yang berfungsi membaca kondisi lingkungan secara real-time.[2], [5]

Action pada smart thermostat meliputi tindakan menghidupkan AC, menghidupkan heater, mematikan seluruh perangkat, dan mempertahankan kondisi suhu saat ini. Seluruh tindakan dilakukan berdasarkan hasil evaluasi suhu terhadap suhu target yang telah ditentukan pengguna.[4]

Goal utama sistem adalah menjaga suhu ruangan tetap nyaman pada rentang 22°C–24°C sekaligus menghemat penggunaan energi listrik. Environment pada penelitian ini berupa ruangan tertutup yang dilengkapi AC, heater, sensor suhu, sensor gerak, dan koneksi internet.[2]

Definisi task environment smart thermostat direpresentasikan dalam bentuk percept sequence sebagai berikut:

{[30, AC_mati], [28, AC_nyala], [25, AC_nyala], [24, AC_mati],}
{[18, Heater_mati], [18, Heater_nyala], [21, Heater_nyala], [24, Heater_mati],}

Percept sequence tersebut menunjukkan perubahan kondisi suhu ruangan dan status perangkat selama proses pencarian menuju suhu target.[4]

2.2. Agent Function THERMOSMART

Agent function THERMOSMART™ dirancang untuk menentukan aksi berdasarkan percept sequence yang diterima sistem. Fungsi agent bekerja dengan mengevaluasi suhu ruangan, suhu target, dan keberadaan pengguna sebelum menentukan tindakan pengendalian HVAC.

Aturan agent function THERMOSMART™ dijelaskan sebagai berikut:

- 1) Jika tidak ada orang di dalam ruangan, maka sistem akan menjalankan aksi *MatikanSemua* untuk menghemat energi.
- 2) Jika suhu ruangan lebih besar dari suhu target + 1°C, maka sistem menjalankan aksi *NaikkanAC*.
- 3) Jika suhu ruangan lebih kecil dari suhu target – 1°C, maka sistem menjalankan aksi *HidupkanHeater*.
- 4) Jika suhu berada dalam rentang target $\pm 1^\circ\text{C}$, maka sistem menjalankan aksi *Pertahankan*. [2]

Representasi fungsi agent dituliskan sebagai berikut:

$$f(\{..., [*, AC_mati], ada_orang = False\}) \rightarrow MatikanSemua$$

$$f(\{..., [suhu > target + 1], ada_orang = True\}) \rightarrow NaikkanAC$$

$$f(\{..., [suhu < target - 1], ada_orang = True\}) \rightarrow HidupkanHeater$$

$$f(\{..., [|suhu - target| \leq 1], ada_orang = True\}) \rightarrow Pertahankan$$

Agent function tersebut memungkinkan smart thermostat melakukan pengambilan keputusan secara otomatis dan adaptif terhadap perubahan lingkungan.[5]

2.3. Agent Program THERMOSMART

Program agent THERMOSMART™ mengimplementasikan fungsi agent dalam bentuk logika pemrograman sederhana berbasis kondisi. Program menerima input berupa suhu ruangan, suhu target, dan status keberadaan pengguna.[4]

Algoritma program agent dituliskan sebagai berikut:

```
function ThermoSmart(suhu_ruangan, suhu_target, ada_orang)
    if not ada_orang then
        return MatikanSemua

    else if suhu_ruangan > suhu_target + 1 then
        return NaikkanAC

    else if suhu_ruangan < suhu_target - 1 then
        return HidupkanHeater

    else
        return Pertahankan
```

Implementasi program tersebut memungkinkan sistem thermostat mengontrol perangkat HVAC secara otomatis berdasarkan kondisi suhu ruangan dan keberadaan pengguna.[2]

2.4. Formulasi Problem Smart Thermostat Agent

Formulasi problem digunakan untuk mendefinisikan kondisi awal, aksi, goal test, dan biaya pencarian pada smart thermostat agent.[1]

2.4.1. Initial State

Initial state pada penelitian ini adalah:

Suhu ruangan = 30°C

Suhu target = 24°C

AC = mati

Heater = mati

ada_orang = True

State awal direpresentasikan sebagai berikut:

```
state_awal = {
    suhu_ruangan : 30,
    suhu_target : 24,
    ac_nyala : False,
    heater_nyala : False,
    ada_orang : True
}
```

Kondisi awal tersebut menggambarkan ruangan panas yang memerlukan pendinginan agar mencapai suhu nyaman.[5]

2.4.2. Actions / Successor Function S(x)

Successor function digunakan untuk menghasilkan state baru berdasarkan aksi tertentu.

Tabel 1. Successor function

Nama Aksi	Kondisi Pemicu	State Baru
CekKehadiran	Selalu dilakukan pertama	ada_orang = True/False
CekSuhu	Jika ada_orang = True	suhu dibandingkan target
NaikkanAC	$suhu > target + 1$	suhu turun bertahap
HidupkanHeater	$suhu < target - 1$	suhu naik bertahap
Pertahankan	$ suhu - target \leq 1$	standby
MatikanSemua	ada_orang = False	semua perangkat mati

Contoh successor state:

$$S(suhu = 30, ada_orang = True) \\ \rightarrow \{ < CekSuhu >, < NaikkanAC, suhu = 28 > \}$$

Successor function memungkinkan sistem berpindah dari satu state menuju state berikutnya selama proses pencarian solusi.[1]

2.4.3. Goal Test

Goal test digunakan untuk menentukan apakah suhu ruangan telah mencapai kondisi nyaman.

Persamaan goal test dituliskan sebagai berikut:

$$|suhu_ruangan - suhu_target| \leq 1$$

Kondisi dianggap berhasil apabila suhu berada pada rentang 23°C–25°C.[2]

2.4.4. Path Cost

Path cost digunakan untuk menghitung biaya setiap langkah perubahan suhu.

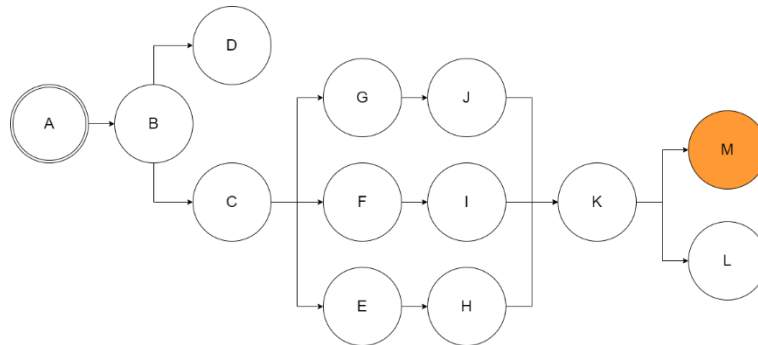
Setiap perubahan suhu sebesar 1°C memiliki cost sebesar 1. Contohnya:

$$30^{\circ}C \rightarrow 29^{\circ}C \rightarrow 28^{\circ}C \rightarrow 27^{\circ}C \rightarrow 26^{\circ}C \rightarrow 25^{\circ}C \rightarrow 24^{\circ}C$$

Total langkah = 6 sehingga path cost = 6.

2.5. State Space Graph dan Tree Terminology

State space graph digunakan untuk menggambarkan seluruh kemungkinan perpindahan state.



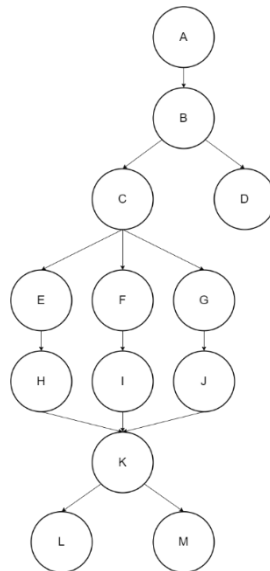
Gambar 1. State space graph

Isi Node:

- A = Start: Sistem Aktif
- B = Cek Kehadiran (PIR Sensor)
- C = Cek Suhu vs Target
- D = Matikan Semua (tdk ada orang)
- E = Naikkan AC (suhu panas)
- F = Hidupkan Heater (suhu dingin)
- G = Pertahankan (suhu normal)
- H = Suhu Turun / I = Suhu Naik
- J = Suhu Stabil / K = Goal Test
- L = Loop → C / M = GOAL 24°C

Node A berfungsi sebagai root node sedangkan node M merupakan goal node yang menunjukkan suhu nyaman telah tercapai.

Tree terminology pada penelitian ini terdiri atas root node, intermediate node, leaf node, loop node, dan goal node. Struktur tree digunakan untuk mempermudah proses pencarian jalur solusi oleh algoritma pencarian.



Gambar 2. Tree Terminology

2.6. Implementasi Algoritma Pencarian

Penelitian ini mengimplementasikan enam algoritma pencarian, yaitu BFS, DFS, Generate and Test, Hill Climbing, Greedy Best-First Search, dan A* Algorithm.

Breadth-First Search (BFS) menggunakan struktur queue FIFO dan melakukan ekspansi node berdasarkan level pencarian. BFS menghasilkan solusi optimal namun membutuhkan penggunaan memori besar.[4]

Depth-First Search (DFS) menggunakan struktur stack LIFO dan melakukan eksplorasi node terdalam terlebih dahulu. DFS lebih cepat tetapi tidak selalu menghasilkan solusi optimal.[1]

Generate and Test bekerja dengan membangkitkan berbagai kemungkinan jalur kemudian menguji apakah jalur tersebut mencapai goal state.

Hill Climbing menggunakan pendekatan steepest ascent dengan memilih node tetangga yang memiliki nilai heuristik terkecil.

Greedy Best-First Search melakukan ekspansi node berdasarkan nilai heuristik minimum sehingga proses pencarian berlangsung lebih cepat.[4]

A* Algorithm menggunakan fungsi evaluasi:

$$f(n) = g(n) + h(n)$$

dengan:

$g(n)$ = biaya lintasan dari start node

$h(n)$ = estimasi jarak menuju goal

Algoritma A* dipilih karena mampu menghasilkan solusi optimal dan lengkap selama heuristik yang digunakan bersifat admissible.[1]

III. HASIL DAN PEMBAHASAN

3.1. Hasil Generate and Test

Generate and Test bekerja dengan menghasilkan kemungkinan jalur solusi kemudian menguji apakah jalur tersebut mencapai goal node M.

Pada penelitian ini, beberapa jalur yang dihasilkan ditunjukkan pada tabel berikut.

Tabel 2. Generate and Test

Rute yang Di-Generate	Total Cost	Hasil
A → B → D	2	Gagal
A → B → C → F → I → K → L	6	Gagal
A → B → C → G → J → K → L	6	Gagal
A → B → C → E → H → K → L	6	Gagal
A → B → C → E → H → K → M	6	SUKSES

Berdasarkan hasil pengujian, jalur valid yang berhasil mencapai goal adalah:

$$A \rightarrow B \rightarrow C \rightarrow E \rightarrow H \rightarrow K \rightarrow M$$

dengan total path cost sebesar 6.

Generate and Test memiliki implementasi sederhana karena hanya membutuhkan proses pembangkitan dan pengujian solusi.

Namun, metode ini kurang efisien pada ruang pencarian besar karena jumlah kemungkinan rute meningkat secara eksponensial sehingga waktu pencarian menjadi lebih lama.

3.2. Hasil Hill Climbing (Steepest Ascent)

Hill Climbing menggunakan pendekatan heuristik dengan memilih node tetangga yang memiliki nilai heuristik terkecil pada setiap langkah pencarian.

Nilai heuristik digunakan sebagai estimasi kedekatan node terhadap goal state. Proses pencarian Hill Climbing ditunjukkan pada tabel berikut.

Tabel 3. Hill Climbing

Langkah	Posisi Saat Ini	Tetangga	Nilai h(n)	Keputusan
1	A	B	5	Pilih B
2	B	C, D	4, 10	Pilih C
3	C	E, F, G	3, 7, 7	Pilih E
4	E	H	2	Pilih H
5	H	K	1	Pilih K
6	K	L, M	8, 0	Pilih M

Rute yang dihasilkan adalah:

$$A \rightarrow B \rightarrow C \rightarrow E \rightarrow H \rightarrow K \rightarrow M$$

dengan total cost sebesar 6.

Hill Climbing terbukti lebih efisien dibandingkan BFS karena algoritma hanya mengevaluasi node yang memiliki peluang terbaik menuju goal state.

Akan tetapi, algoritma ini memiliki kelemahan berupa risiko *local optimum*, yaitu kondisi ketika algoritma berhenti pada solusi lokal dan gagal menemukan solusi global terbaik.

3.3. Hasil Greedy Best-First Search

Greedy Best-First Search melakukan pencarian berdasarkan nilai heuristik terkecil tanpa mempertimbangkan biaya lintasan yang telah ditempuh. Urutan proses pencarian ditunjukkan pada tabel berikut.

Tabel 4. Greedy Best-First Search

Fringe	Node Diekspansi
[A(6)]	A
[B(5)]	B
[C(4), D(10)]	C
[E(3), D(10), F(7), G(7)]	E
[H(2), D(10), F(7), G(7)]	H
[K(1), D(10), F(7), G(7)]	K
[M(0), L(8), D(10)]	M

Greedy Best-First Search menghasilkan jalur:

$$A \rightarrow B \rightarrow C \rightarrow E \rightarrow H \rightarrow K \rightarrow M$$

dengan total cost sebesar 6.

Algoritma ini memiliki waktu pencarian relatif cepat karena hanya mempertimbangkan nilai heuristik terkecil pada setiap langkah.

Namun, Greedy Best-First Search tidak menjamin solusi optimal karena algoritma mengabaikan biaya lintasan sebelumnya sehingga dapat memilih jalur yang terlihat dekat tetapi sebenarnya lebih mahal.

3.4. Hasil A* Algorithm

A* Algorithm mengombinasikan biaya lintasan $g(n)$ dan heuristik $h(n)$ melalui fungsi evaluasi:

$$f(n) = g(n) + h(n)$$

Pendekatan ini memungkinkan algoritma melakukan pencarian secara optimal dan efisien.

Proses pencarian A* ditunjukkan pada tabel berikut.

Tabel 5. A* Algorithm

Node	$g(n)$	$h(n)$	$f(n)=g+h$	Keputusan
B	1	5	6	Dipilih
C	2	4	6	Dipilih
D	2	10	12	Diabaikan
E	3	3	6	Dipilih
F	3	7	10	Diabaikan
G	3	7	10	Diabaikan
H	4	2	6	Dipilih
K	5	1	6	Dipilih
M	6	0	6	GOAL

Rute optimal yang dihasilkan adalah:

$$A \rightarrow B \rightarrow C \rightarrow E \rightarrow H \rightarrow K \rightarrow M$$

dengan total path cost sebesar 6.

Hasil tersebut menunjukkan bahwa algoritma A* mampu mempertahankan nilai $f(n) = 6$ secara konsisten hingga mencapai goal state. Hal ini membuktikan bahwa heuristik yang digunakan bersifat admissible dan konsisten.

Dibandingkan algoritma lainnya, A* memberikan keseimbangan terbaik antara optimalitas solusi dan efisiensi pencarian sehingga sangat cocok diterapkan pada smart thermostat agent.

3.5. Perbandingan Keseluruhan Algoritma
Perbandingan keseluruhan algoritma ditunjukkan pada tabel berikut.

Tabel 6. Perbandingan Keseluruhan Algoritma

Algoritma	Rute	Langkah	Optimal	Kelengkapan	Memori
Generate & Test	$A \rightarrow B \rightarrow C \rightarrow E \rightarrow H \rightarrow K \rightarrow M$	6	Ya	Tidak efisien	Sedang
Hill Climbing	$A \rightarrow B \rightarrow C \rightarrow E \rightarrow H \rightarrow K \rightarrow M$	6	Tidak selalu	Tidak lengkap	Kecil
Greedy BFS	$A \rightarrow B \rightarrow C \rightarrow E \rightarrow H \rightarrow K \rightarrow M$	6	Tidak selalu	Tidak lengkap	Sedang
A*	$A \rightarrow B \rightarrow C \rightarrow E \rightarrow H \rightarrow K \rightarrow M$	6	Ya	Lengkap	Efisien

Berdasarkan hasil pengujian, BFS dan A* mampu menghasilkan solusi optimal. Namun, BFS memiliki kebutuhan memori lebih besar dibandingkan A*.

DFS menunjukkan proses pencarian tercepat tetapi tidak menjamin optimalitas solusi. Hill Climbing dan Greedy Best-First Search memiliki efisiensi pencarian tinggi namun rentan terhadap kesalahan heuristik dan local optimum.

Secara keseluruhan, A* Algorithm menjadi metode paling efektif pada smart thermostat agent karena mampu menghasilkan jalur optimal dengan efisiensi pencarian yang baik serta penggunaan heuristik yang konsisten.

IV. KESIMPULAN

Penelitian ini berhasil merancang dan mengimplementasikan smart thermostat sebagai intelligent agent berbasis artificial intelligence dengan menggunakan konsep task environment yang terdiri atas percept, action, goal, dan environment. Sistem mampu melakukan pengambilan keputusan otomatis untuk menjaga suhu ruangan tetap berada pada kondisi nyaman dalam rentang 22°C–24°C sekaligus mendukung efisiensi penggunaan energi.

Berdasarkan hasil pengujian, algoritma heuristic search mampu diterapkan dengan baik pada sistem smart thermostat untuk menentukan jalur pengendalian suhu menuju goal state. Algoritma Generate and Test berhasil menemukan solusi valid, namun efisiensinya menurun ketika ruang pencarian semakin besar karena harus membangkitkan banyak kemungkinan jalur.

Hill Climbing menunjukkan performa pencarian yang cepat dan efisien karena hanya memilih node dengan nilai heuristik terbaik pada setiap langkah. Akan tetapi, algoritma ini memiliki kelemahan berupa kemungkinan terjebak pada local optimum sehingga tidak selalu menjamin solusi global terbaik.[4]

Greedy Best-First Search mampu mempercepat proses pencarian dengan memprioritaskan node yang memiliki nilai heuristik terkecil. Namun, algoritma ini hanya mempertimbangkan estimasi jarak menuju goal tanpa memperhatikan biaya lintasan yang telah ditempuh sehingga optimalitas solusi tidak selalu terjamin.

Hasil terbaik diperoleh oleh algoritma A* karena mampu mengombinasikan biaya lintasan $g(n)$ dan nilai heuristik $h(n)$ melalui fungsi evaluasi $f(n) = g(n) + h(n)$. Algoritma A* berhasil menghasilkan jalur optimal secara konsisten dengan total cost minimum serta efisiensi pencarian yang lebih baik dibandingkan algoritma lainnya.

Penelitian selanjutnya dapat dikembangkan dengan implementasi langsung pada perangkat keras seperti sensor suhu, mikrokontroler, dan sistem Internet of Things (IoT) agar smart thermostat dapat diterapkan pada lingkungan nyata dengan state space yang lebih kompleks dan multi-zona.

REFERENSI

- [1] G. B. Balogun, D. Ibisagba, A. Bajeh, T. O. Debo, A. Muyideen, and O. J. Peter, "Comparative analysis of AI-based search algorithms in solving 8 puzzle problems," *Bull. Natl. Res. Cent.*, vol. 48, no. 1, Nov. 2024, doi: 10.1186/s42269-024-01274-3.
- [2] R. Adhikari, M. Pipattanasomporn, and S. Rahman, "Heuristic Algorithms for Aggregated HVAC Control via Smart Thermostats for Regulation Service," *IEEE Trans. Smart Grid*, vol. 11, no. 3, pp. 2023–2032, 2020, doi: 10.1109/TSG.2019.2946170.
- [3] B. Elkari *et al.*, "Exploring Maze Navigation: A Comparative Study of DFS, BFS, and A* Search Algorithms," *Statistics, Optimization and Information Computing*, vol. 12, no. 3, pp. 761–781, 2024, doi: 10.19139/SOIC-2310-5070-1939.

-
- [4] A. Kherrou, M. Robol, M. Roveri, and P. Giorgini, “A multi-algorithm pathfinding method: Exploiting performance variations for enhanced efficiency,” *Ann. Math. Artif. Intell.*, vol. 93, no. 4, pp. 569–588, Aug. 2025, doi: 10.1007/s10472-024-09957-3.
 - [5] A. Yayla *et al.*, “Artificial Intelligence (AI)-Based Occupant-Centric Heating Ventilation and Air Conditioning (HVAC) Control System for Multi-Zone Commercial Buildings,” *Sustainability (Switzerland)*, vol. 14, no. 23, Dec. 2022, doi: 10.3390/su142316107.