Optimisasi Algoritma Backtracking dalam Penyelesaian Permainan *Sudoku* Menggunakan Pewarnaan Graf

Nicholas Andhika Lucas, 13523014[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523014@std.stei.itb.ac.id, [2]realandhikalucas@gmail.com

***Abstract*— Sudoku adalah salah satu permainan teka-teki angka yang sangat populer dan menantang. Dalam permainan tersebut, pemain dituntut untuk mengisi kotak kosong pada grid 9x9 sesuai aturan tertentu. Salah satu pendekatan untuk menyelesaikan permainan sudoku yang dibuat adalah adalah algoritma *backtracking*, yang meskipun efektif, dapat menjadi tidak efisien pada kasus tertentu. Penelitian ini memanfaatkan teori pewarnaan graf untuk memodelkan Sudoku sebagai graf, dengan simpul merepresentasikan kotak Sudoku dan warna merepresentasikan angka 1-9. Optimisasi dilakukan menggunakan heuristik Most Constrained Variable (MCV) dan Least Constraining Value (LCV), yang memungkinkan algoritma memilih variabel dengan jumlah kemungkinan solusi paling sedikit dan solusi yang paling tidak membatasi variabel lainnya. Implementasi algoritma ini dilakukan menggunakan Python, dan hasil eksperimen menunjukkan algoritma dapat menyelesaikan Sudoku dengan baik.**

***Keywords*—sudoku, backtracking, pewarnaan graf**

I. INTRODUCTION

Sudoku merupakan salah satu jenis permainan yang sangat popular. Permainan ini lahir pada abad ke-20 menjadi populer di Jepang, hingga dipublikasikan pertama kalinya dengan nama *sudoku* yang berarti "*single number*". Hal ini berhubungan dengan prinsip dan aturan utama permainan *sudoku*. Prinsipnya adalah untuk menyelesaikan sekumpulan angka yang kosong pada suatu kotak 9x9, yang sudah diisi sebagai petunjuk awal. Aturan utama dalam permainan *sudoku* adalah mengisi setiap baris dan kolom dengan angka 1-9, dengan syarat tidak boleh ada angka yang sama dalam 1 baris, kolom, dan 1 *subgrid* (kotak 3x3). Dengan demikian, pemain diuji untuk menggunakan logika dan kemampuan deduksi untuk mengisi kotak-kotak yang kosong dengan angka yang tepat.

Perkembangan permainan ini memunculkan beberapa cara untuk mencari solusi dari permainan *sudoku*, ketimbang mencari jawaban berdasarkan deduksi logika dan uji coba berulang kali. Namun, hingga sekarang, belum ditemukan algoritma penyelesaian dari permainan sudoku yang dianggap konstan, atau dapat ditebak. Dalam kata lain, permainan ini dianggap *NP-Complete*. Meskipun demikian, beberapa algoritma pemrograman muncul untuk menyelesaikan masalah ini, seperti *bit masking* dan *cross-hatching* – tidak akan dibahas pada penelitian ini –, dan *backtracking*.

Salah satu teknik yang dianggap paling efisien adalah *backtracking*, walaupun kompleksitas algoritmanya tidak menentu. Prinsip utama dari algoritma ini adalah terus menerus melakukan iterasi pada setiap kotak dan berusaha mengisi angka yang tepat, mempertimbangkan kecocokannya dengan baris, kolom, dan *subgrid* sekitar. Teknik ini dapat dimanfaatkan dalam salah satu teori graf yang dipelajari dalam matematika diskrit, yaitu pewarnaan graf. Mengingat algoritma ini masih terbatas secara efektivitas, pengembangan dan optimisasi dapat dilakukan agar algoritma *backtracking semakin baik*. Pada penelitian ini, akan dilakukan optimisasi dari algoritma *backtracking* dalam pewarnaan graf sebagai metode penyelesaian dari permainan *sudoku*.

II. BASIC THEORY

A. Sudoku

Sudoku adalah salah satu permainan teka-teki angka yang sangat terkenal. Pada umumnya, antarmuka Sudoku terdiri dari persegi sejumlah 9 x 9 yang disusun teratur pada subsegmen 3 x 3. Beberapa di antara persegi yang tersedia sudah terisi angka 1-9. Tujuan dari teka-teki ini adalah mengisi semua persegi kosong yang tersisa sehingga setiap persegi terisi oleh angka 1-9 tepat satu kali pada setiap baris, kolom, dan kesembilan subgrid 3 x 3. Akan diberikan sejumlah petunjuk kotak-kotak yang diisi, dan pemain didorong untuk mampu mengisi kotak-kotak kosong sesuai dengan aturan yang ada. Berdasarkan konfigurasi dan aturan tersebut, secara matematis, setiap kategori (baris, kolom, atau subsegmen) berjumlah 45. Dengan demikian, teka-teki Sudoku diketahui mempunyai hingga 6.670.903.752.021.072.936.960 variasi pengisian angka.

Sudoku merupakan teka-teki yang mengandalkan kemampuan logika dan kombinatorika karena tingkat

kesulitan permainannya ditentukan berdasarkan jumlah dan posisi awal mula kotak yang sudah terisi. Dalam mengisi setiap kotak, disarankan menggunakan pensil dibanding dengan bolpoin karena pemain harus melalui banyak trial-and-error dalam rangka memperoleh susunan angka yang sempurna. Dalam studi kesehatan multidisiplin, Sudoku diprediksi mampu memelihara kemampuan otak seseorang setara dengan 10 tahun lebih muda dari usia pemain yang sesungguhnya.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

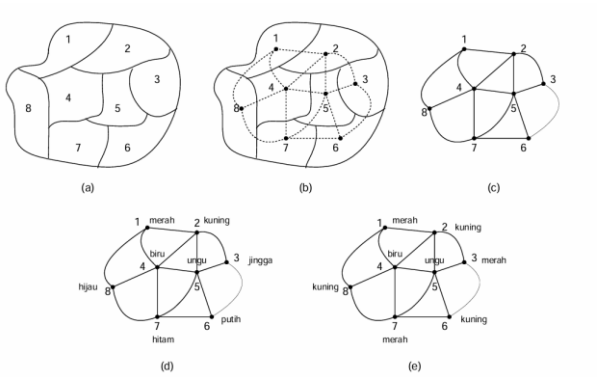
Gambar 1a. Contoh permainan sudoku dan 1b. Solusi permainan sudoku. Sumber: Wikimedia.org

B. Pewarnaan Graf

Pewarnaan graf merujuk pada pewarnaan simpul dalam suatu graf sedemikian rupa sehingga dua simpul bertetangga memiliki warna yang berbeda. Pewarnaan graf dikenal juga dengan pewarnaan simpul. Suatu graf dapat diwarnai dengan sejumlah warna selama banyak warnanya lebih kecil dari atau sama dengan jumlah simpul; dilambangkan dengan $m \leq n(v)$, di mana m adalah jumlah warna dan n(v) adalah jumlah simpul. Pewarnaan graf yang dilakukan dengan paling banyak m-warna disebut juga pewarnaan-m.

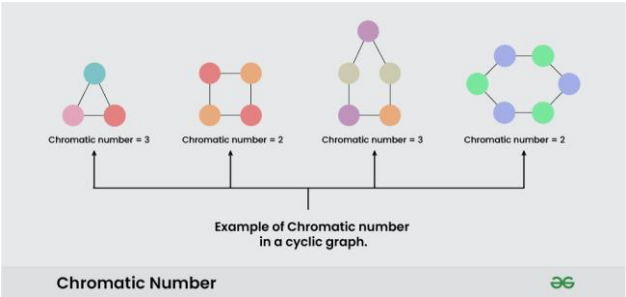
Pewarnaan graf memiliki banyak manfaatnya dan dapat diaplikasikan dalam banyak situasi. Salah satunya adalah untuk mewarnai suatu peta yang terdiri dari sejumlah wilayah yang bertetanggaan. Agar tidak ada dua wilayah yang memiliki warna yang sama – demi kenyamanan pengalaman pembaca –, maka pewarnaan graf dapat diaplikasikan.

Dalam konteks pengaplikasian pewarnaan graf untuk peta, Rinaldi (2024) menjelaskan langkah-langkahnya sebagai berikut. Pertama, menyatakan suatu wilayah sebagai simpul dan batas antar kedua wilayah yang bertetangga sebagai sisi. Kedua, mewarnai simpul pada graf yang terbuat, memperhatikan algoritma pewarnaan graf. Warna simpul yang bertetangga tidak boleh sama, agar warna wilayah pada peta berbeda-beda dan terpisah.



Gambar 2. Langkah pewarnaan peta menggunakan pewarnaan graf. Sumber: Rinaldi (2024).

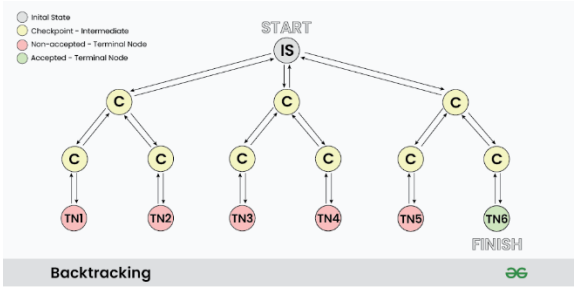
Dalam pewarnaan graf, dikenal istilah bilangan kromatik dengan simbol: $\chi(G)$. Istilah ini merujuk pada jumlah minimum warna yang dibutuhkan untuk mewarnai suatu graf. Misalnya, suatu graf yang paling sedikit memilki 3 warna unik untuk pewarnaan graf dinyatakan dengan nilai bilangan kromatik $\chi(G) = 3$. Untuk suatu graf lengkap K_n, semua simpul akan saling terhubung dengan simpul lainnya. Oleh karena itu, semua simpul akan saling bertetanggaan dan bilangan kromatiknya adalah $\chi(G) = n$.



Gambar 3. Ilustrasi bilangan kromatik pada graf. Sumber: geeksforgeeks.com

C. Algoritma Backtracking

Backtracking adalah suatu teknik pencarian yang melakukan rekursi untuk mencari berbagai macam cara untuk mendapatkan suatu solusi. Sederhananya, *backtracking* akan menguji coba suatu cara/lintasan, kemudian kembali ke titik sebelumnya dan menguji coba cara lainnya. Pencarian ini akan berhenti apabila suatu solusi ditemukan atau semua kemungkinan cara sudah diuji coba. Algoritma ini sering dimanfaatkan dalam kasus dibutuhkan pencarian berbagai pilihan atau kemungkinan, seperti permainan sudoku. Namun, risiko dari algoritma ini adalah kompleksitas algoritma pada kasus terburuk, yaitu eksponensial $O(|\text{Domain}|^n)$.



Gambar 4. Pencarian *backtracking*. Sumber: geeksforgeeks.com

Langkah-langkah algoritma *backtracking* yang memanfaatkan rekursi dijelaskan oleh Melanie (2024) sebagai berikut:

1. Pemilihan Opsi
   Salah satu opsi dari seluruh opsi lainnya dipilih untuk memulai pencarian. Pilihan ini dapat dilakukan secara sekuensial maupun secara spesifik, tergantung dengan permasalahan.

2. Validasi
   Algoritma akan melakukan pengecekan pada opsi yang dipilih, apakah sesuai dengan batasan dan apa yang diinginkan. Apabila opsi tersebut sesuai, maka akan dilakukan pencarian lebih lanjut untuk akar-akar (opsi) di bawahnya, kembali ke langkah 1.

3. *Backtracking*
   Apabila ternyata suatu opsi yang dipilih melanggar batasan dari algoritma atau sudah dilakukan pencarian pada semua opsi dalam cabang, maka opsi tersebut gagal dan akan dilakukan *backtracking*. Pencarian akan kembali ke akar atau titik sebelumnya pada pohon pencarian.

4. Solusi
   Jika opsi yang divalidasi terpenuhi dan tidak memiliki cabang-cabang lagi – atau dalam pohon, opsi tersebut adalah daun –, maka pencarian selesai.

D. Most Constrained Variable (MCV) dan Least Constrained Value (LCV)

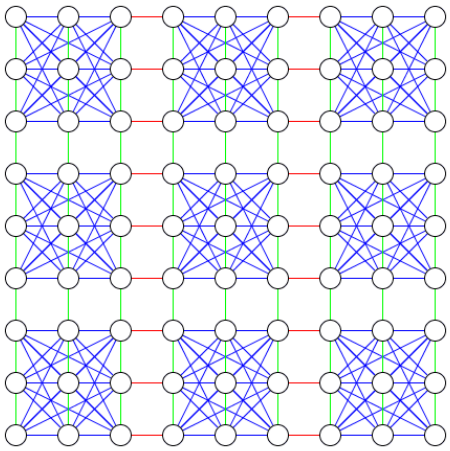
Dalam pencarian menggunakan algoritma *backtracking*, pencarian dapat dioptimisasi agar lebih efisien. Salah satunya adalah melakukan pengurutan secara dinamis, ketimbang melakukannya secara statis. Bentuk pengurutan secara dinamis ini contohnya adalah menggunakan MCV dan LCV.

Most Constrained Variable (MCV) adalah strategi pengurutan yang memprioritaskan pemilihan variabel dengan jumlah kemungkinan nilai paling terbatas. Dengan implementasi ini, identifikasi variabel yang paling terbatas dapat dilakukan lebih awal. Prinsip yang sama digunakan untuk *Least Constrained Value (LCV)*, yaitu variabel dengan nilai paling banyak dari variabel bertetanggaan lainnya. Menggunakan prosedur ini, pencarian akan memilih nilai-nilai yang mungkin berhasil dan membatasi variabel tetangga lainnya seminimal mungkin.

III. IMPLEMENTASI DAN EKSPERIMEN

A. Menggambarkan Sudoku sebagai Pewarnaan Graf

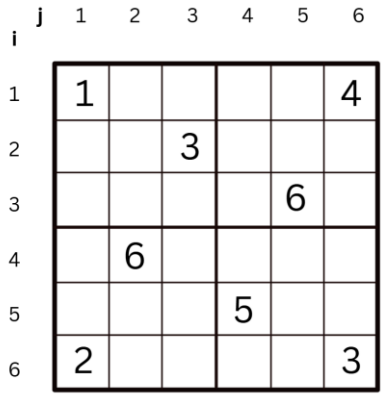
Permainan sudoku dengan kotak 9x9 dapat dilihat sebagai suatu graf dengan 81 simpul pada setiap kotaknya. Setiap simpul akan dihubungkan dengan semua simpul lainnya pada baris, kolom, dan subgrid 3x3 masing-masing. Maka, akan tercipta suatu graf dengan bilangan kromatik $\chi(G) = 9$, sehingga dapat diselesaikan dengan pewarnaan graf.



Gambar 5. Terhubungnya setiap simpul pada setiap subgrid dalam graf sudoku. Sumber: Gupta (2020).

B. Implementasi MCV dan LCV pada Algoritma Backtracking

Seperti yang dijelaskan sebelumnya, pencarian berbasis *backtracking* dapat dikembangkan agar lebih efektif, khususnya dalam pemilihan opsi pencarian. Oleh karena itu, MCV dan LCV akan diimplementasikan dengan tujuan memilih variabel yang paling memungkinkan untuk pencarian selanjutnya, sehingga mengurangi iterasi yang harus dilakukan. Dalam konteks permainan sudoku, gabungan kedua Teknik ini digunakan untuk pertama-tama, mencari kotak kosong yang memiliki kemungkinan solusi (opsi pengisian angka 1-9) yang paling sedikit, dan kedua memilih solusi yang sedemikian rupa tidak membatasi opsi solusi dari kotak lainnya. Sebagai contoh, akan dijelaskan pada ilustrasi di bawah.



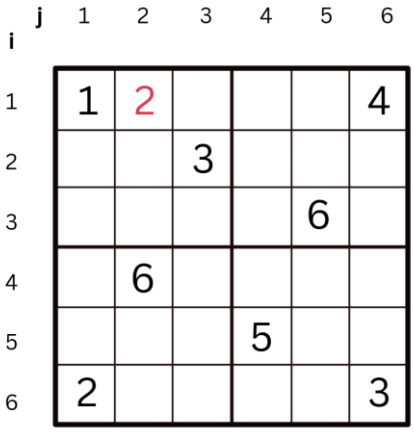
Gambar 5. Ilustrasi 1 implementasi MCV dan LCV pada sudoku. Sumber: peneliti.

Misalkan suatu kotak sudoku berukuran 6x6 yang dinyatakan sebagai M_{ij}. Terdapat kotak kosong di titik-ij dengan opsi solusi masing-masing adalah:

1. (1, 2) = {2, 3, 5},
2. (2, 3) = {1, 2, 4, 5, 6}, dan
3. (3,4) = {1, 2, 3, 4, 5}.

MCV akan digunakan untuk memilih kotak yang paling terbatas, yaitu kotak dengan jumlah solusi angka yang paling sedikit. Maka, akan dipilih kotak (1,2) dengan opsi solusi {2, 3, 5}.

Kemudian, LCV akan digunakan untuk memilih opsi solusi yang sesuai. Dengan opsi solusi {2, 3, 5} ditinjau dampaknya pada kotak yang bertetanggaan. Contohnya, apabila terpilih angka 2 sebagai solusi dari (1,2), maka opsi solusi pada kotak bertetangganya, kotak (2,2) adalah{4, 5, 6}, atau sebanyak 3 opsi. Sedangkan, jika terpilih angka 3 dan 5, opsi untuk (2,2) akan menjadi 4 kemungkinan. Meninjau dari jumlah kemungkinan solusi untuk kotak (2,2), maka solusi dari (1,2) yang dipilih harus yang menyebabkan kemungkinan terkecil. Sehingga, angka 2 menjadi solusi untuk kotak (1,2). Dengan demikian, penyelesaian permainan sudoku akan lebih cepat.



Gambar 6. Ilustrasi 2 implementasi MCV dan LCV pada sudoku. Sumber: peneliti.

C. Implementasi dalam Python

Implementasi algoritma ini akan menggunakan bahasa pemrograman Python. *Source code* yang digunakan diadaptasi dari eksperimen yang telah dilakukan oleh Gupta (2020). Pembuatan *source code* memanfaatkan kelas-kelas berupa Node dan Graph untuk menggambarkan simpul dan pemodelan graf sederhana.

Pemodelan Kelas

```python
class Node :

    def __init__(self, idx, data = 0) : # Constructor
        """
        id : Integer (1, 2, 3, ...)
        """
        self.id = idx
        self.data = data
        self.connectedTo = dict()
```

Gambar 6. Kelas Node. Sumber: Gupta (2020).

```python
def addNeighbour(self, neighbour , weight = 0) :
    """
    neighbour : Node Object
    weight : Default Value = 0

    adds the neightbour_id : wt pair into the dictionary
    """
    if neighbour.id not in self.connectedTo.keys() :
        self.connectedTo[neighbour.id] = weight

# setter
def setData(self, data) :
    self.data = data

#getter
def getConnections(self) :
    return self.connectedTo.keys()

def getID(self) :
    return self.id

def getData(self) :
    return self.data

def getWeight(self, neighbour) :
    return self.connectedTo[neighbour.id]

def __str__(self) :
    return str(self.data) + " Connected to : "+ \
     str([x.data for x in self.connectedTo])
```

Gambar 6. Implementasi Kelas Node. Sumber: Gupta (2020).

```python
class Graph :

    totalV = 0 # total vertices in the graph

    def __init__(self) :
        """
        allNodes = Dictionary (key:value)
                   idx : Node Object
        """
        self.allNodes = dict()
```

Gambar 6. Kelas Graf. Sumber: Gupta (2020).

```python
def addNode(self, idx) :
    """ adds the node """
    if idx in self.allNodes :
        return None

    Graph.totalV += 1
    node = Node(idx=idx)
    self.allNodes[idx] = node
    return node

def addNodeData(self, idx, data) :
    """ set node data acc to idx """
    if idx in self.allNodes :
        node = self.allNodes[idx]
        node.setData(data)
    else :
        print("No ID to add the data.")

def addEdge(self, src, dst, wt = 0) :
    """
    Adds edge between 2 nodes
    Undirected graph

    src = node_id = edge starts from
    dst = node_id = edge ends at

    To make it a directed graph comment the second line
    """
    self.allNodes[src].addNeighbour(self.allNodes[dst], wt)
    self.allNodes[dst].addNeighbour(self.allNodes[src], wt)

def isNeighbour(self, u, v) :
    """
    check neighbour exists or not
    """
    if u >=1 and u <= 81 and v >=1 and v<= 81 and u !=v :
        if v in self.allNodes[u].getConnections() :
            return True
    return False



def printEdges(self) :
    """ print all edges """
    for idx in self.allNodes :
        node =  self.allNodes[idx]
        for con in node.getConnections() :
            print(node.getID(), " --> ",
            self.allNodes[con].getID())
```

Gambar 6. Implementasi Kelas Graf. Sumber: Gupta (2020).

Selain ini, program juga memanfaatkan library sudoku_connections yang akan membantu implementasi keterhubungan dari suatu simpul dengan simpul lainnya.

Pemodelan Algoritma

Pemodelan algoritma *backtracking* akan menggunakan referensi langkah-langkah dari tinjauan pustaka beserta implementasi dari MCV dan LCV pada algoritma tersebut.

```python
def graphColoringInitializeColor(self):
    """
    fill the already given colors
    """
    color = [0] * (self.sudokuGraph.graph.totalV+1)
    given = [] # list of all the ids whos value is already given. Thus cannot be changed
    for row in range(len(self.board)) :
        for col in range(len(self.board[row])) :
            if self.board[row][col] != 0 :
                #first get the idx of the position
                idx = self.mappedGrid[row][col]
                #update the color
                color[idx] = self.board[row][col] # this is the main imp part
                given.append(idx)
    return color, given

def solveGraphColoringMCV(self, m=9):
    color, given = self.graphColoringInitializeColor()


    if not self.__graphColorUtility_with_mcv_lcv(m=m, color=color, given=given):
        print(":(")
        return False


    count = 1
    for row in range(9):
        for col in range(9):
            self.board[row][col] = color[count]
            count += 1
    return color

def __graphColorUtility_with_mcv_lcv(self, m, color, given):
    # find the most constrained variable (MCV)
    cell = self.find_mcv_cell(color, given)
    if not cell:
        return True

    v = cell

    options = self.get_valid_colors(v, color, given)
    if not options:
        return False

    options = self.least_constraining_value(v, options, color, given)

    for c in options:
        if self.__isSafe2Color(v, color, c, given) == True :
            color[v] = c
            if self.__graphColorUtility_with_mcv_lcv(m, color, given):
                return True
        if v not in given:
            color[v] = 0

    return False
```

```python
def find_mcv_cell(self, color, given):
    min_options = float('inf')
    mcv_node = None

    for v in range(1, self.sudokuGraph.graph.totalV + 1):
        if v not in given and color[v] == 0:  # Node is uncolored
            options = self.get_valid_colors(v, color, given)
            if len(options) < min_options:
                min_options = len(options)
                mcv_node = v

    return mcv_node


def get_valid_colors(self, v, color, given):
    invalid_colors = set()

    # Check neighbors
    for neighbor in range(1, self.sudokuGraph.graph.totalV + 1):
        if self.sudokuGraph.graph.isNeighbour(v, neighbor):
            invalid_colors.add(color[neighbor])

    # Return valid colors
    return [c for c in range(1, 10) if c not in invalid_colors]


def least_constraining_value(self, v, options, color, given):
    impact = {}

    for c in options:
        # Simulate coloring
        color[v] = c
        impact[c] = 0

        # Check impact on neighbors
        for neighbor in range(1, self.sudokuGraph.graph.totalV + 1):
            if self.sudokuGraph.graph.isNeighbour(v, neighbor) and color[neighbor] == 0:
                valid_colors = self.get_valid_colors(neighbor, color, given)
                impact[c] += len(valid_colors)

        color[v] = 0  # Reset color

    # Return colors sorted by their impact (least impact first)
    return sorted(options, key=lambda x: impact[x])
```

Gambar 6. Implementasi Fungsi Pewarnaan Graf. Sumber: penulis.

Proses implementasi pewarnaan graf berjalan dengan pertama mengisi warna-warna awal pada simpul graf sesuai dengan angka yang ada pada kotak Sudoku yang diberikan. Dalam konteks ini, akan digunakan angka 1-9 sebagai representasi dari warna unik.

Kemudian, algoritma MCV dan LCV dimplementasikan pada fungsi masing-masing, untuk menentukan opsi warna yang paling sedikit bagi MCV dan menentukan warna paling tidak membatasi pada simpul yang terpilih. Sebelum dilakukan pewarnaan, dilakukan validasi untuk memastikan pewarnaan dapat berjalan atau tidak.

Ketika salah satu pengecekan opsi gagal, maka akan dilakukan *backtracking* untuk mencari opsi lain yang dapat memenuhi solusi. Setelah semua simpul diwarnai, maka permainan sudoku berhasil diselesaikan.

D. Hasil Eksperimen

Sampel yang digunakan untuk menguji coba program yang telah dibuat adalah sebagai berikut. Bilangan 0 digunakan untuk menandakan kotak kosong, dengan setiap kotak dinyatakan dalam i-j. Jumlah petunjuk yang diberikan pada uji coba ini adalah sebanyak 22. Jumlah petunjuk ini ditinjau sebagai kesulitan tingkat sedang dalam permainan sudoku. Tujuan eksperimen ini adalah mengimplementasikan algoritma yang dioptimisasi dan mendapatkan solusi sudoku yang sesuai.

```
BEFORE SOLVING ...


   1 2 3    4 5 6    7 8 9
 - - - - - - - - - - - - -
| 0 0 0 | 4 0 0 | 0 0 0 | 1
| 4 0 9 | 0 0 6 | 8 7 0 | 2
| 0 0 0 | 9 0 0 | 1 0 0 | 3
 - - - - - - - - - - - - -
| 5 0 4 | 0 2 0 | 0 0 9 | 4
| 0 7 0 | 8 0 4 | 0 6 0 | 5
| 6 0 0 | 0 3 0 | 5 0 2 | 6
 - - - - - - - - - - - - -
| 0 0 1 | 0 0 7 | 0 0 0 | 7
| 0 4 3 | 2 0 0 | 6 0 5 | 8
| 0 0 0 | 0 0 5 | 0 0 0 | 9
 - - - - - - - - - - - - -
```

Gambar 6. Uji coba yang digunakan. Sumber: penulis.

```
Solving ...


AFTER SOLVING ...


   1 2 3    4 5 6    7 8 9
 - - - - - - - - - - - - -
| 1 8 5 | 4 7 3 | 9 2 6 | 1
| 4 2 9 | 5 1 6 | 8 7 3 | 2
| 3 6 7 | 9 8 2 | 1 5 4 | 3
 - - - - - - - - - - - - -
| 5 3 4 | 6 2 1 | 7 8 9 | 4
| 9 7 2 | 8 5 4 | 3 6 1 | 5
| 6 1 8 | 7 3 9 | 5 4 2 | 6
 - - - - - - - - - - - - -
| 2 5 1 | 3 6 7 | 4 9 8 | 7
| 7 4 3 | 2 9 8 | 6 1 5 | 8
| 8 9 6 | 1 4 5 | 2 3 7 | 9
 - - - - - - - - - - - - -
Time taken: 0.02856 seconds
```

Gambar 6. Hasil uji coba yang digunakan. Sumber: penulis.

Implementasi program berhasil karena mampu memberikan solusi sudoku yang tepat. Optimisasi dari algoritma pewarnaan graf menghasilkan penyelesaian waktu yang relatif cepat, yaitu sekitar 0,03 detik.

D. CONCLUSION

Penelitian ini didasarkan pada kebutuhan optimisasi algoritma backtracking yang umum digunakan dalam pewarnaan graf. Dalam konteks penyelesaian permainan sudoku, dikembangkan dengan optimisasi *Most Constrained Variable (MCV)* dan *Least Constraining Value (LCV)*, hasil uji coba menunjukkan bahwa implementasi ini berhasil dilaksanakan. Bagi peneliti selanjutnya, penelitian dapat dikembangkan melalui perbandingan dengan jenis algoritma lainnya maupun jumlah uji coba yang jauh lebih bervariasi.

VII. ACKNOWLEDGMENT

The author thanks all parties that have directly contributed to the creation of this paper. The author especially thanks Mr. Rinaldi Munir and Rila Mandala as the teacher of the 2024/2025 Discrete Mathemathics course.

REFERENCES

[1] Sudoku Solver — Graph Coloring. Solving a Sudoku Puzzle using Graph… | by Ishaan Gupta | Code Science | Medium
[2] Solving Sudoku puzzles with Graph Theory - Online Technical Discussion Groups—Wolfram Community
[3] CS 221 - Variables-based Models Cheatsheet
[4] Introduction to Backtracking - GeeksforGeeks
[5] Introduction to Graph Coloring - GeeksforGeeks
[6] Rinaldi Munir. "Graf Bagian 3," *Institut Teknologi Bandung*, 2024 (Accessed Jan. 8, 2025).

PERNYATAAN

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

Bandung, 8 Januari 2024

Nicholas Andhika Lucas (13523014)