Analisis Batas P dan NP-Complete pada Puzzle Sudoku Melalui Implementasi Solver Logika-Heuristik dan Backtracking

Ivant Samuel Silaban - 13523129
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: smart058440@gmail.com , 13523129@std.stei.itb.ac.id

***Abstract*—Puzzle Sudoku, meskipun populer dengan aturan yang sederhana, merepresentasikan sebuah masalah yang secara komputasi tergolong "sulit" dan telah terbukti NP-Complete. Fenomena ini menciptakan dualisme menarik antara kemudahan pemahaman dan kesulitan penyelesaiaan. Makalah ini akan mendemonstrasikan batas antara kelas masalah P dan NP Complete dengan menggunakan Sudoku sebagai studi kasus. Untuk mencapai tujuan inim diimplementasikan dua jenis solver dengan pendekatan yang berbeda secara fundamental: sebuah solver berbasis logika-heuristik yang mengadopsi teknik penyelesaian manusia (seperti Singles, Naked Pairs, dan X-Wing), dan sebuah solver berbasis backtracking murni yang menggunakan pencarian brute-force. Hasil pengujian menunjukkan bahwa solver logika-heuristik mampu menyelesaikan puzzle tingkat mudah dan sulit dalam waktu polinomial yang sangat efisien, namun gagal pada puzzle "diabolical" yang tidak memiliki pola logis yang dapat dikenali. Kegagalan ini memaksa penggunaan solver backtracking, uang meskipun berhasil menemukan solusi, menunjukkan peningkatan waktu dan langkah komputasi yang bersifat eksponensial. Melalui makalah ini, dapat divisualisasikan batas kemampuan dari algoritma polinomial dan mengkonfirmasi secara praktis implikasi dari sifat NP-Complete pada sebuah masalah.**

Keywords—component; Sudoku; Strategi Algoritma; Backtracking; Logika Heuristik; P vs NP, NP-Complete

I. Introduction

Sudoku adalah salah satu permainan puzzle logika yang paling populer di dunia, digemari karena perpaduan unik antara kesederhanaan aturan dan kedalaman strategi. Aturan dasarnya adalah dengan mengisi grid 9x9 dengan angka 1 hingga 9 tanpa pengulangan pada setiap baris, kolom, dan sub-grid 3x3. Namun, dibalik *gameplay* yang sederhana ini, tersembunyi sebuah tantangan komputasi yang signifikan, dimana tingkat kesulitan pada sebuah puzzle dapat bervariasi secara drastis. Dualisme antara aturan yang mudah dan solusi yang bisa jadi sangat sulit ini menjadikan sudoku bukan hanya sekadar permainan biasa, tetapi sebuah laboratorium ideal untuk menjelajahi konsep fundamental dalam ilmu komputer, yaitu teori kompleksitas komputasi. Fenomena ini memunculkan pertanyaan inti: apa yang secara fundamental membedakan sudoku yang "mudah" dari yang "sulit", dan bagaimana kita bisa memodelkan perbedaan ini secara algoritmik?

Makalah ini akan bertujuan untuk memberikan demonstrasi empiris mengenai batasan antara masalah P dan NP-Complete menggunakan puzzle Sudoku sebagai mediumnya. Makalah ini akan menjawab bagaimana serangkaian teknik logika heuristik dapat memetakan bagian masalah sudoku yang berada di kelas P? Dan pada titik mana teknik-teknik in mencapai batas kemampuannya, sehingga menunjukkan kebutuhan akan pendekatan eksponensial yang mencerminkan sifat NP-Complete pada umumnya? Penelitian ini akan menjawab dengan berfokus pada tiga tujuan utama, yaitu merancang tiga tingkatan solver dasar (logika sederhana, logika advanced, backtracking), melakukan analisis kinerja komparatif terhadap ketiga solver dengan berbagai kesulitan, dan menyajikan analisis hasil sebagai bukti empiris dari konsep P dan NP-Complete.

Implementasi solver logika yang digunakan akan dibatasi menjadi logika-logika yang "minimal" dikuasai oleh manusia untuk menyelesaikan sebuah sudoku kompleks, mulai dari Naked/Hidden Singles untuk solver sederhana, penambahan Naked Pairs, Pointing Pairs, dan X-Wing untuk advanced. Analisis yang dilakukan berfokus pada kinerja algoritmik yang terukur, seperti waktu eksekusi dan jumlah langkah komputasi.

II. Theoretical Basis

A. *Puzzle Sudoku*

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

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

Gambar 2.1 Kiri: Puzzle, Kanan: Solusi[7]

Sudoku adalah puzzle penempatan angka berbasis logika yang dimainkan pada sebuah grid berukuran 9x9, yang terbagi lagi menjadi sembilan sub-grid atau “kotak” berukuran 3x3. Tujuan dari permainan ini adalah untuk mengisi semua sel kosong dengan angka dari 1 hingga 9 dengan mematuhi tiga aturan atau batasan utama secara simultan: pertama, setiap baris horizontal haru berisi setiap angka tepat satu kali, lalu setiap kolom vertikal harus berisi setiap angka tepat satu kali; dan ketiga, setiap sub-grid exe harus berisi setiap angka tepat satu kali. Sebuah puzzle Sudoku yang valid diasumsikan memiliki satu solusi unik.

B. Teori Kompleksitas Komputasi

Sudoku memiliki kompleksitas yang mendalam yang dapat dianalisis menggunakan teori kompleksitas komputasi untuk mengklasifikasikan “kesulitan” suatu masalah. Terdapat beberapa kelas kompleksitas yang relevan untuk penelitian ini. Kelas P (Polynomial Time) adalah himpunan masalah keputusan yang dapat diselesaikan oleh sebuah algoritma deterministik dalam waktu polinomial. Ini adalah masalah-masalah yang dianggap “mudah” atau dapat diselesaikan secara efisien oleh komputer.

Selanjutnya ada kelas NP (Nondeterministic Polynomial TIme), yaitu himpunan masalah yang di mana usulan solusi dapat diverifikasi kebenarannya dalam waktu polinomial. Jika diberi sebuah papan sudoku yang sudah terisi penuh, kita dapat dengan cepat memeriksa apakah solusinya mematuhi semua aturan. Dan terakhir ada kelas NP-Complete, yaitu sub-himpunan dari NP yang berisi masalah-masalah paling sulit di dalamnya. Sebuah masalah tergolong NP Complete jika ia berada di kelas NP dan semua masalah lain di NP dapat direduksi kepadanya dalam waktu polinomial. MAsalah umum penyelesaian Sudoku telah terbukti secara formal sebagai masalah NP_Complete, yang mengimplikasikan bahwa belum ada algoritma efisien (waktu polinomial) yang diketahui dapat menyelesaikan semua kemungkinan instance Sudoku.

C. Algoritma dan Heuristik Penyelesaian Sudoku

Untuk memecahkan puzzle Sudoku, terdapat dua pendekatan algoritmik utama yang merepresentasikan dualisme antara efisiensi dan kelengkapan, yang akan diimplementasikan dalam makalah ini.

Pendekatan pertama adalah Logika-Heuristik, yang meniru proses penalaran deduktif manusia. Pendekatan ini tidak menebak, melainkan menerapkan serangkaian aturan atau teknik untuk secara pasti menempatkan angka atau mengeliminasi kandidat dari sel-sel kosong. Teknik-teknik ini bersifat deterministik dan berjalan dalam waktu polinomial. Teknik dasar seperti Naked Singles dan Hidden Singles bertujuan untuk menempatkan angka secara langsung. Seiring meningkatnya kompleksitas puzzle, diperlukan teknik eliminasi lain seperti Naked Pairs dan Pointing Pairs untuk mengurangi ruang pencarian. Dan untuk kasus lebih sulit lain, diperlukan pengenalan pola yang lebih kompleks dan untuk makalah ini, akan digunakan pendekatan X-Wing yang menganalisis hubungan kandidat di beberapa baris dan kolom secara bersamaan untuk melakukan eliminasi.

Pendekatan kedua adalah algoritma Backtracking, yang merupakan representasi dari metode brute-force. Algoritma ini akan bekerja secara rekursif. Ia akan mencari sel kosong pertama lalu mencoba menempatkan angka valid pertama (dari 1 hingga 9), lalu memanggil dirinya sendiri untuk melanjutkan penyelesaian. Jika nanti ada kebuntuan atau konflik, ia akan mundur (backtrack) ke langkah sebelumnya, membatalkan pilihan angka tersebut, dan mencoba angka valid berikutnya. Meskipun algoritma ini menjamin penemuan solusi jika memang ada, pada kasus terburuk ia harus menjelajahi pohon kemungkinan yang sangat besar, sehingga memiliki kompleksitas waktu eksponensial. Pendekatan ini menjadi satu-satunya pilihan ketika semua teknik logika-heuristik telah gagal menyederhanakan puzzle lebih lanjut.

III. METHODS

A. Implementasi Logika Heuristic

Solver ini dirancang untuk menyelesaikan Sudoku dengan menerapkan serangkaian aturan deduktif secara berulang, mirip dengan cara berpikir manusia. Pendekatan ini diimplementasikan dalam sebuah loop utama yang akan terus berjalan selama masih ada kemajuan (eliminasi kandidat atau penempatan angka yang dapat dibuat.

Pseudocode Alur Kerja Utama Solver Logika:

```
FUNCTION Solve_Logic(board, active_techniques):
   candidates = Initialize_Candidates_From_Board(board)
   total_steps = 0

   LOOP WHILE TRUE:
      progress_made_this_iteration = FALSE

      IF 'singles' in active_techniques:
         changed, steps = Apply_Singles(board, candidates)
         if changed: progress_made_this_iteration = TRUE;
total_steps += steps
      ENDIF

      IF 'naked_pairs' in active_techniques:
         changed, steps = Apply_Naked_Pairs(board,
candidates)
         if changed: progress_made_this_iteration = TRUE;
total_steps += steps
      ENDIF

      IF 'pointing_pairs' in active_techniques:
         changed, steps = Apply_Pointing_Pairs(board,
candidates)
         if changed: progress_made_this_iteration = TRUE;
total_steps += steps
      ENDIF

      IF 'xwing' in active_techniques:
         changed, steps = Apply_X_Wing(board, candidates)
         if changed: progress_made_this_iteration = TRUE;
total_steps += steps
```

```
    ENDIF

    IF progress_made_this_iteration IS FALSE:
      BREAK LOOP
    ENDIF
  ENDWHILE

  RETURN final board and total_steps
ENDFUNCTION
```

Berikut adalah pseudocode untuk setiap teknik logika yang diimplementasikan:

1. Teknik Naked & Hidden Singles

```
FUNCTION Apply_Singles(board, candidates):
  FOR each empty cell (r, c):
    IF size of candidates[r, c] is 1:
      Place the only candidate in board[r, c]
      Update candidates of affected peers
      RETURN TRUE // Ada perubahan
    ENDIF
  ENDFOR

  FOR each unit (row, col, box):
    FOR each digit D from 1 to 9:
      Find all positions in the unit where D is a candidate
      IF there is only one such position:
        Place D in that position on the board
        Update candidates of affected peers
        RETURN TRUE
      ENDIF
    ENDFOR
  ENDFOR

  RETURN FALSE
ENDFUNCTION
```

2. Teknik Naked Pairs

```
FUNCTION Apply_Naked_Pairs(board, candidates):
  FOR each unit (row, col, box):
    Find all cells in the unit with exactly 2 candidates.
    Group these cells by their candidate set.

    FOR each group of cells with identical candidates:
      IF the group contains 2 cells:
        digit1, digit2 = the candidates of the pair

        FOR each other_cell in the unit:
          Remove digit1 and digit2 from candidates of other_cell
          IF removal causes change: RETURN TRUE
        ENDFOR
      ENDIF
    ENDFOR
  ENDFOR
  RETURN FALSE
ENDFUNCTION
```

3. Teknik Pointing Pairs

```
FUNCTION Apply_Pointing_Pairs(board, candidates):
  FOR each box from 1 to 9:
    FOR each digit D from 1 to 9:
      Find all positions in the box where D is a candidate.

      IF all such positions are in the same row R:
        Remove D as a candidate from row R, outside of the current box.
        IF removal causes change: RETURN TRUE
      ENDIF

      IF all such positions are in the same col C:
        Remove D as a candidate from col C, outside of the current box.
        IF removal causes change: RETURN TRUE
      ENDIF
    ENDFOR
  ENDFOR
  RETURN FALSE
ENDFUNCTION
```

4. Teknik X-Wing

```
FUNCTION Apply_X_Wing(board, candidates):
  FOR each digit D from 1 to 9:
    Find all rows where D is a candidate in exactly two columns.
    FOR each pair of these rows (R1, R2):
      IF their candidate columns are identical (C1, C2):
        FOR each other_row (not R1 or R2):
          Remove D from candidates at (other_row, C1) and (other_row, C2).
          IF removal causes change: RETURN TRUE
        ENDFOR
      ENDIF
    ENDFOR
```

```
    ENDFOR
    RETURN FALSE
ENDFUNCTION
```

B. *Implementasi Solver Backtracking*

Solver ini adalah implementasi dari algoritma pencarian brute force rekursif yang akan mencoba semua kemungkinan solusi secara sistematis. Pendekatan ini akan menjamin penemuan solusi jika ada.

Pseudocode alur kerja Solver Backtracking:

```
FUNCTION Solve_Backtracking(board, time_limit,
step_limit):
   IF current_time > time_limit OR current_steps >
step_limit:
      RETURN "Limit Reached"
   ENDIF

   empty_cell = Find_Empty_Cell(board)
   IF empty_cell IS NULL:
      RETURN TRUE
   ENDIF

   row, col = coordinates of empty_cell
   FOR digit FROM 1 TO 9:
      IF Is_Valid_Placement(board, row, col, digit) IS
TRUE:
         board[row][col] = digit

         result = Solve_Backtracking(board, time_limit,
step_limit)
         IF result IS NOT FALSE:
            RETURN result
         ENDIF

         board[row][col] = 0 // Backtrack
      ENDIF
   ENDFOR

   RETURN FALSE
ENDFUNCTION
```

C. *Skenario dan Metrik Pengujian*

Setiap puzzle dari dataset diuji secara independen oleh Solver Logika–Heuristik (Advanced) dan Solver Backtracking Murni. Solver Backtracking diberikan batas waktu eksekusi 5 detik dan batas langkah 20 juta pemanggilan rekursif untuk menangani kasus yang sangat kompleks.

Dataset yang digunakan untuk penelitian ini terdiri dari empat kategori puzzle, yang masing-masing memiliki karakteristik untuk untuk menguji hipotesis penelitian:

1. Puzzle ‘Mudah’: Puzzle ini dirancang untuk dapat diselesaikan hanya dengan menggunakan teknik-teknik logika dasar, seperti Naked Singles dan Hidden Singles. Tujuannya adalah untuk menjadi benchmark kinerja dasar dan demonstrasikan kasus di mana pendekatan heuristik dapat menemukan solusi dengan sangat efisien, yang merepresentasikan masalah di kelas P.
2. Puzzle ‘Sulit’: Puzzle ini dipilih karena tidak dapat diselesaikan hanya dengan teknik dasar dan secara spesifik membutuhkan penerapan teknik eliminasi kandidat yang lebih canggih, seperti Naked Pairs dan X-Wing. Tujuannya adalah untuk menguji kekuatan dan efisiensi dari Solver Logika dan menunjukkan batas kepraktisan dari pendekatan Backtracking murni pada puzzle yang sudah mulai kompleks.
3. Puzzle ‘Diabolical’: Puzzle ini tergolong sangat sulit dan dirancang untuk resisten terhadap semua teknik logika yang diimplementasikan di makalah ini. Tujuannya adalah untuk menciptakan sebuah “dinding” di mana pendekatan heuristik akan gagal total, sehingga memaksa penggunaan algoritma backtracking untuk menemukan solusi. Kasus ini digunakan untuk mendemonstrasikan secara empiris biaya komputasi eksponensial saat menyelesaikan masalah yang merepresentasikan sifat NP-Complete.
4. Puzzle ‘Kosong’: Kasus ini bukan sebagai representasi kesulitan, melainkan sebagai kasus kontrol untuk menguji perilaku algoritma pada kondisi tanpa batasan. Tujuannya adalah untuk menunjukkan bahwa worst-case dari algoritma backtracking bukanlah papan yang paling kosong, melainkan papan yang memiliki batasan-batasan awal yang rumit dan memaksa algoritma untuk melakukan ‘Salah langkah’ dan mundur.

Kinerja setiap solve akan diukur berdasarkan:

1. Status Penyelesaian: apakah Terpecahkan, Macet, atau Batas Tercapai (Timeout/Steps).
2. Waktu Eksekusi: Diukur dalam milidetik (ms)
3. Jumlah Langkah Komputasi: Diukur sebagai jumlah estimasi eliminasi kandidat untuk solver logika, dan jumlah total pemanggilan fungsi rekursif untuk solver backtracking.

IV. Result

A. *Hasil Pengujian*

Berikut adalah hasil pengujian dari empat jenis puzzle:

Puzzle 1 (Mudah):

```
Papan Awal:
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

--- Hasil Solver Logika-Heuristik (Advanced) ---
Status: Terpecahkan
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
Waktu: 0.6026 ms | Langkah: 153 (estimasi eliminasi kandidat)

--- Hasil Solver Backtracking (Murni) ---
Status: Terpecahkan
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
Waktu: 23.4820 ms | Langkah: 4,209 pemanggilan rekursif
```

Gambar 4.1 Hasil Puzzle Mudah

Puzzle 2 (Sulit):

```
Papan Awal:
. . .  | 8 . 1  | . . .
. . .  | . . .  | . 4 3
5 . .  | . . .  | . . .
- - - - - - - - - - - -
. . .  | . 7 .  | 8 . .
. . .  | . . .  | 1 . .
. 2 .  | . 3 .  | . . .
- - - - - - - - - - - -
6 . .  | . . .  | . 7 5
. . 3  | 4 . .  | . . .
. . .  | 2 . .  | 6 . .

--- Hasil Solver Logika-Heuristik (Advanced) ---
Status: Terpecahkan
2 3 7  | 8 4 1  | 5 6 9
1 8 6  | 7 9 5  | 2 4 3
5 9 4  | 3 2 6  | 7 1 8
- - - - - - - - - - - -
3 1 5  | 6 7 4  | 8 9 2
4 6 9  | 5 8 2  | 1 3 7
7 2 8  | 1 3 9  | 4 5 6
- - - - - - - - - - - -
6 4 2  | 9 1 8  | 3 7 5
8 5 3  | 4 6 7  | 9 2 1
9 7 1  | 2 5 3  | 6 8 4
Waktu: 5.8374 ms | Langkah: 227 (estimasi eliminasi kandidat)

--- Hasil Solver Backtracking (Murni) ---
Status: Batas Tercapai (Timeout/Steps)
2 3 4  | 8 5 1  | 7 6 9
1 9 8  | 7 6 2  | 5 4 3
5 6 7  | 9 4 3  | 2 8 1
- - - - - - - - - - - -
3 5 9  | 1 7 4  | 8 2 6
4 7 6  | 5 . .  | 1 . .
. 2 .  | . 3 .  | . . .
- - - - - - - - - - - -
6 . .  | . . .  | . 7 5
. . 3  | 4 . .  | . . .
. . .  | 2 . .  | 6 . .
Waktu: 5000.0169 ms | Langkah: 836,740 pemanggilan rekursif
```

Gambar 4.2 Hasil Puzzle Sulit

Puzzle 3 (Diabolical):

```
Papan Awal:
8 . .  | . . .  | . . .
. . 3  | 6 . .  | . . .
. 7 .  | . 9 .  | 2 . .
- - - - - - - - - - - -
. 5 .  | . . 7  | . . .
. . .  | . 4 5  | 7 . .
. . .  | 1 . .  | . 3 .
- - - - - - - - - - - -
. . 1  | . . .  | . 6 8
. . 8  | 5 . .  | . 1 .
. 9 .  | . . .  | 4 . .

--- Hasil Solver Logika-Heuristik (Advanced) ---
Status: Macet
8 . .  | . . .  | . . .
. . 3  | 6 . .  | . . .
. 7 .  | . 9 .  | 2 . .
- - - - - - - - - - - -
. 5 .  | . . 7  | . . .
. . .  | . 4 5  | 7 . .
. . .  | 1 . .  | . 3 .
- - - - - - - - - - - -
. . 1  | . . .  | . 6 8
. . 8  | 5 . .  | . 1 .
. 9 .  | . . .  | 4 . .
Waktu: 0.5257 ms | Langkah: 0 (estimasi eliminasi kandidat)

--- Hasil Solver Backtracking (Murni) ---
Status: Terpecahkan
8 1 2  | 7 5 3  | 6 4 9
9 4 3  | 6 8 2  | 1 7 5
6 7 5  | 4 9 1  | 2 8 3
- - - - - - - - - - - -
1 5 4  | 2 3 7  | 8 9 6
3 6 9  | 8 4 5  | 7 2 1
2 8 7  | 1 6 9  | 5 3 4
- - - - - - - - - - - -
5 2 1  | 9 7 4  | 3 6 8
4 3 8  | 5 2 6  | 9 1 7
7 9 6  | 3 1 8  | 4 5 2
Waktu: 279.9553 ms | Langkah: 49,559 pemanggilan rekursif
```

Gambar 4.3 Hasil Puzzle Diabolical

Puzzle 4 (Kosong):

```
Papan Awal:
. . .  | . . .  | . . .
. . .  | . . .  | . . .
. . .  | . . .  | . . .
- - - - - - - - - - - -
. . .  | . . .  | . . .
. . .  | . . .  | . . .
. . .  | . . .  | . . .
- - - - - - - - - - - -
. . .  | . . .  | . . .
. . .  | . . .  | . . .
. . .  | . . .  | . . .

--- Hasil Solver Logika-Heuristik (Advanced) ---
Status: Macet
. . .  | . . .  | . . .
. . .  | . . .  | . . .
. . .  | . . .  | . . .
- - - - - - - - - - - -
. . .  | . . .  | . . .
. . .  | . . .  | . . .
. . .  | . . .  | . . .
- - - - - - - - - - - -
. . .  | . . .  | . . .
. . .  | . . .  | . . .
. . .  | . . .  | . . .
Waktu: 0.6131 ms | Langkah: 0 (estimasi eliminasi kandidat)

--- Hasil Solver Backtracking (Murni) ---
Status: Terpecahkan
1 2 3  | 4 5 6  | 7 8 9
4 5 6  | 7 8 9  | 1 2 3
7 8 9  | 1 2 3  | 4 5 6
- - - - - - - - - - - -
2 1 4  | 3 6 5  | 8 9 7
3 6 5  | 8 9 7  | 2 1 4
8 9 7  | 2 1 4  | 3 6 5
- - - - - - - - - - - -
5 3 1  | 6 4 2  | 9 7 8
6 4 2  | 9 7 8  | 5 3 1
9 7 8  | 5 3 1  | 6 4 2
Waktu: 1.9377 ms | Langkah: 392 pemanggilan rekursif
```

Gambar 4.4 Hasil Puzzle Kosong

B. Penyajian Data Hasil Pengujian

Berikut adalah hasil perbandingkan kinerja solver Logika-Heuristik dan Backtracking Murni.

Tabel 4.1: Hasil Perbandingkan Kinerja Solver Logika-Heuristik dan Backtracking Murni.

Tingkatan Puzzle	Metode Solver	Status Akhir	Waktu Eksekusi (ms)	Jumlah Langkah
Mudah	Logika-Heuristic	Terpecahkan	0.6026	153
	Backtracking	Terpecahkan	23.4820	4,209
Sulit	Logika-Heuristic	Terpecahkan	5.8374	227
	Backtracking	Batas Tercapai	5000.0169	836,740
Diabolical	Logika-Heuristic	Macet	0.5257	0
	Backtracking	Terpecahkan	279.9553	49,559
Kosong	Logika-Heuristic	Macet	0.6131	0
	Backtracking	Terpecahkan	1.9377	392

C. Analisis Kinerja Algoritma

Dari data pada tabel 4.1, dapat dianalisis kinerja masing-masing pendekatan pada tiap kesulitan.

1. Kinerja pada Puzzle Mudah

   Pada puzzle tingkat “Mudah”, kedua solver berhasil menemukan solusi yang benar. Namun, terdapat perbedaan kinerja yang sangat signifikan. Solver Logika-Heuristik (Advanced) mampu menyelesaikan puzzle dalam waktu yang sangat singkat, yaitu 0.6026 ms dengan hanya 153 langkah estimasi. Untuk Solver Backtracking membutuhkan waktu 23.4820 ms dan 4.209 pemanggilan rekursif. Ini menunjukkan bahwa kasus di mana solusi dapat ditemukan melalui deduksi logika sederhana, pendekatan heuristik secara dramatis lebih efisien dibandingkan backtracking/bruteforce.

2. Kinerja pada Puzzle Sulit

   Pada puzzle tingkat “Sulit”, kekuatan dari heuristik yang lebih canggih menjadi terlihat. Solver Logika-Heuristik kembali berhasil menyelesaikan puzzle sepenuhnya dalam waktu yang sangat efisien, yaitu 5.8374. Ini membuktikan bahwa serangkain teknik logika dasar mampu menyelesaikan puzzle dengan kompleksitas yang tinggi. Di sisi lain, Solver Backtracking murni menunjukkan ketidak praktisaannya. Dengan batas waktu 5 detik, solver ini gagal menemukan solusi dan berhenti setelah melakukan 836.740 langkah rekursif. Perbandingkan ini secara jelas menggarisbawahi bagaimana heuristik yang baik dapat mengubah masalah yang tadinya tidak dapat diselesaikan oleh-brute force dalam waktu wajar menjadi masalah yang dapat ditangani dengan mudah.

3. Kinerja pada Puzzle Diabolical

   Pada puzzle “Diabolical”, batas kemampuan dari solver logika mulai kelihatan. Logika-Heuristik langsung melaporkan status “Macet” dengan 0 langkah, menandakan tidak ada satupun dari kelima teknik logika yang dapat diterapkan pada papan awal. Kegagalan ini kemudian akan bergantung pada solver Backtracking. Solver Backtracking berhasil menemukan solusi, namun dengan biaya komputasi yang terukur dan signifikan: 279.9553 ms an 49.559 pemanggilan rekursif. Hasil ini menjadi demonstrasi kunci dari sebuah masalah yang resisten terhadap penyelesaian heuristik dan memerlukan pencarian eksponensial untuk dipecahkan.

4. Kinerja pada Puzzle Kosong

   Pada puzzle “Kosong”, kita diberikan wawasan yang menarik. Sama seperti puzzle Diabolical, server logika langsung “Macet” karena tidak adanya petunjuk awal. Namun, yang menarik adalah kinerja Solver Backtracking yang justru sangat cepat, menyelesaikan papan dalam waktu 1.9377 ms dengan hanya 392 langkah rekursif. Hasil yang tampaknya kontradiktif ini menjelaskan bahwa “kasus tersulit” untuk algoritma backtracking bukanlah papan yang paling kosong, melainkan papan yang memiliki batasan-batasan awal yang rumit dan memaksa algoritma untuk sering salah belok dan mundur (backtrack). Papan kosong tidak memiliki batasan, sehingga algoritma dapat mengisi angka secara berurutan tanpa perlu melakukan backtrack.

D. Pembahasan: Implikasi terhadap Teori P, NP, NPC

Hasil eksperimen ini memberikan demonstrasi empiris yang gamblang mengenai batasan antara kelas masalah P dan NP-Complete.

1. Demonstrasi Kelas P: Puzzle “Mudah” dan “Sulit” yang berhasil diselesaikan oleh olver Logika-Heuristik dalam waktu yang sangat singkat merupakan representasi dari himpunan bagian masalah Sudoku yang berada di dalam kelas P.

Keberadaan algoritma deterministik (solver logika kita) yang dapat menyelesaikannya secara efisien adalah bukti empiris dari klaim ini.

2. Demonstrasi Batas Kemampuan Polinomial: Kegagalan total dari Solver Logika-Heuristik pada puzzle “Diabolical” menunjukkan adanya sebuah “dinding” yang membuat teknik-teknik cerdas berbasis pola tidak lagi cukup. Ini menandakan transisi dari masalah yang bisa ditangani secara efisien ke masalah yang memerlukan pendekatan yang lebih fundamental dan mahal secara komputasi.
3. Demonstrasi Sifat NP-Complete: Kebutuhan akan algoritma backtracking untuk menyelesaikan puzzle “Diabolical” dan hasil “Batas Tercapai” pada puzzle “Sulit” untuk backtracking murni adalah manifestasi dari kompleksitas waktu eksponensial. Ini adalah ciri khas dari penyelesaian masalah NP-Complete dengan metode brute-force. Hasil “Batas Tercapai” bukanlah sebuah kegagalan eksperimen, melainkan sebuah data kuantitatif yang berhasil mengukur bahwa ruang pencarian untuk puzzle tersebut terlalu besar untuk dijelajahi secara praktis.

Secara keseluruhan,, eksperimen ini berhasil memvisualisasikan teori P, NP, NPC, meskipun masalah umum Sudoku termasuk NPC, banyak instansinya di dunia nyata yang memiliki struktur yang membuatnya dapat diselesaikan secara efisien (berada di P), namun selalu ada instance yang benar-benar “sulit” yang menunjukkan inti dari kompleksitas masalah tersebut.

Acknowledgment

Pertama, penulis ingin berterima kasih kepada Tuhan yang Maha Esa, karena atas berkatnya, penulis dapat menyelesaikan makalah ini. Kemudian, penulis ingin berterima kasih kepada para dosen IF2211 Strategi Algoritma, terkhusus untuk Monterico Adrian, S.T., M.T. sebagai pengampu mata kuliah ini di kelas K03 Jatinangor yang sudah mengajari penulis tentang banyak strategi algoritma yang ada di dunia informatika ini. Dan Penulis juga berterima kasih kepada pihak-pihak yang secara tidak langsung membantu penulis menyelesaikan makalah ini.

References

[1] R. Munir, "Teori P, NP, dan NP-Complete (Bagian 1)," Materi Kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, 2019. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Teori-P-NP-dan-NPC-(Bagian%201).pdf

[2] R. Munir, "Teori P, NP, dan NP-Complete (Bagian 2)," Materi Kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, 2019. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Teori-P-NP-dan-NPC-(Bagian%202).pdf

[3] R. Munir, "Algoritma Runut-balik (Bagian 1)," Materi Kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, 2024. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian1.pdf

[4] R. Munir, "Algoritma Runut-balik (Bagian 2)," Materi Kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung, 2024. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/16-Algoritma-backtracking-(2025)-Bagian2.pdf

[5] GeeksforGeeks, “Algorithm to Solve Sudoku | Sudoku Solver,” GeeksforGeeks. [Online]. Tersedia: https://www.geeksforgeeks.org/dsa/sudoku-backtracking-7/. [Diakses: 21-Juni-2025].

[6] A. Stuart, "Sudoku Solving Techniques," SudokuWiki.org. [Online]. Tersedia: https://www.sudokuwiki.org/Strategy_Families. [Diakses: 22-Juni-2025].

[7] D. Carmel, “Solving Sudoku by Heuristic Search,” Medium, Nov. 02, 2023. [Online]. Tersedia: https://medium.com/@davidcarmel/solving-sudoku-by-heuristic-search-b0c2b2c5346e. [Diakses: 22-Juni-2025].

[8] J. M. F. M. e. Costa, E. A. C. e. Costa, and M. S. D. e. Costa, “The Chaos Within Sudoku,” Scientific Reports, vol. 2, art. no. 725, Oct. 2012. [Online]. Tersedia: https://www.nature.com/articles/srep00725.

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, 24 Juni 2025

Ivant Samuel Silaban
13523129