Penggunaan Algoritma Backtracking pada Permainan Sudoku untuk Mengisi Kotak-kotak Kosong dengan Angka yang Sesuai dengan *Constraint* Permainan

Muhammad Zaki Amanullah - 13521146
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13521146@std.stei.itb.ac.id

***Abstrak*—Penelitian ini bertujuan untuk mengimplementasikan algritma backtracking dalam ememcahkan teka-teki Sudoku dan mencari solusi yang memenuhi semua constraint permainan. Algoritma backtracking digunakan untuk secara rekursif mencoba angka-angka yang berbeda pada sel-sel kosong hingga menemukan solusi yang valid. Fungsi is_valid digunakan untuk memeriksa apakah suatu angka dapat ditempatkan pada sel tertentu tanpa melanggar aturan Sudoku.**

Kata Kunci—Algoritma Backtracking, Sudoku, pemecahan teka-teki, constraint permainan.

I. PENDAHULUAN

Permainan Sudoku adalah salah satu teka-teki logika yang popular di seluruh dunia. Tujuan permainan ini adalah untuk mengisi kotak-kotak yang kosong dalam sebuah grid 9x9 dengan angka 1 hingga 9, sedemikian sehingga setiap angka hanya muncul sekali dalam setiap baris, kolom, dan blok 3x3 yang terpisah. Meskipun terlihat sederhana, Sudoku dapat menantang pemain dengan tingkat kesulitan yang bervariasi.

Untuk memecahkan Sudoku, dibutuhkan sebuah algoritma yang efisien dan dapat menemukan solusi yang memenuhi semua constraint permainan. Salah satu pendekatan yang umum digunakan adalah menggunakan algoritma Backtracking. Algoritma Backtracking adalah sebuah teknik pencarian sistematis yang mencoba setiap kemungkinan langkah secara rekursif dan secara bertahap membangun solusi secara incremental.

Makalah ini bertujuan untuk memperkenalkan penggunaan algoritma Backtracking dalam memecahkan Sudoku. Kami akan menjelaskan bagaimana algoritma ini dapat digunakan untuk menemukan solusi yang benar dengan mempertimbangkan constraint permainan. Kami juga akan menguraikan langkah-langkah utama dari algoritma Backtracking yang diterapkan pada Sudoku dan memberikan contoh implementasi kode.

Pemahaman dan penerapan algoritma Backtracking pada permainan Sudoku dapat memberikan wawasan tentang teknik pemecahan masalah yang melibatkan constraint dan eksplorasi semua kemungkinan solusi. Selain itu, pemahaman yang baik tentang algoritma ini dapat membantu dalam mengembangkan solusi yang lebih efisien dan meningkatkan performa dalam memecahkan Sudoku.

A. Konsep Dasar Sudoku

1) Aturan Permainan

Sudoku adalah permainan teka-teki logika yang terdiri dari sebuah grid 9x9 yang dibagi menjadi 9 subgrid 3x3. Pada awal permainan, beberapa sel dalam grid sudah terisi dengan angka dari 1 hingga 9, sedangkan sel lainnya kosong. Tujuan permainan adalah mengisi sel-sel kosong tersebut sehingga setiap baris, kolom, dan subgrid 3x3 hanya berisi angka yang berbeda.

2) Maintaining

Permainan Sudoku memiliki beberapa constraint yang harus dipatuhi dalam mengisi angka ke dalam grid. Constraint utama adalah:

- Setiap baris harus berisi angka yang berbeda dari 1 hingga 9
- Setiap kolom harus berisi angka yang berbeda dari 1 hingga 9
- Setiap subgrid 3x3 harus berisi angka yang berbeda dari 1 hingga 9

Dengan adanya *constraint* ini, pemain harus mencari solusi yang memenuhi semua constraint tersebut. Selain itu, Sudoku juga biasanya memiliki beberapa sel yang sudah terisi dengan angka pada awal permainan, yang mengharuskan pemain untuk mempertimbangkan angka-angka yang sudah ada dalam mencari solusi yang benar.

II. LANDASAN TEORI

A. Prinsip Dasar Algoritma Backtracking

Algoritma backtracking adalah sebuah pendekatan untuk menyelesaikan masalah yang melibatkan pencarian solusi secara sistematis dengan menguji semua kemungkinan langkah dan melakukan pencarian mundur (backtrack) saat solusi yang

sedang ditemukan tidak memenuhi syarat. Algoritma ini digunakan dalam berbagai aplikasi, termasuk permainan dan teka-teki, seperti Sudoku.

Prinsip dasar algoritma backtracking dapat dijelaskan sebagai berikut:

1) Pilih langkah awal

Tentukan langkah awal untuk memulai pencarian solusi. Dalam konteks Sudoku, langkah awal bisa berupa mengisi sel kosong dengan angka 1 hingga 9.

2) Coba langkah berikutnya

Pilih langkah berikutnya berdasarkan aturan atau constraint permainan. Dalam Sudoku, langkah berikutnya adalah mengisi sel kosong dengan angka yang belum ada di baris, kolom, dan sub-grid yang sama.

3) Periksa keabsahan langkah

Setelah melakukan langkah, periksa apakah langkah tersebut valid sesuai dengan aturan permainan. Dalam Sudoku, periksa apakah angka yang diisi tidak konflik dengan angka yang sudah ada di baris, kolom, dan sub-grid.

4) Jika langkah valid, lanjutkan pencarian

Jika langkah yang diambil valid, lanjutkan pencarian dengan memilih langkah berikutnya dan kembali ke langkah 2.

5) Jika langkah tidak valid, lakukan backtrack.

Jika langkah yang diambil tidak valid, lakukan backtrack (pencarian mundur) dengan mengembalikan langkah sebelumnya dan mencoba langkah lain yang mungkin belum diuji. Kembali ke langkah 3.

6) Ulangi langkah-langkah 3-5

Ulangi langkah-langkah 3 hingga 5 sampai semua sel terisi dan solusi ditemukan. Jika tidak ada solusi yang memenuhi syarat, pencarian akan kembali ke langkah sebelumnya hingga semua kemungkinan dicoba.

7) Cetak solusi

Setelah solusi ditemukan, cetak atau tampilkan solusi yang valid.

Algoritma backtracking merupakan pendekatan yang efektif untuk menyelesaikan masalah yang memiliki ruang pencarian yang besar dengan mengeliminasi langkah-langkah yang tidak valid secara cepat. Dalam konteks Sudoku, algoritma backtracking memungkinkan kita untuk menemukan solusi yang memenuhi semua constraint permainan dengan mencoba semua kemungkinan langkah dan melakukan pencarian mundur saat ditemukan langkah yang tidak valid.

B. Pseudocode

Untuk lebih jelasnya bagaimana algoritma backtracking dapat diimplementasikan dalam permainan Sudoku, berikut merupakan contoh pseudocodenya.

Pseudocode tersebut menjelaskan alur umum algoritma backtracking yang akan digunakan. Fungsi solveSudoku merupakan fungsi utama yang akan memanggil fungsi-fungsi lainnya untuk menyelesaikan Sudoku. Fungsi findEmptyCell digunakan untuk mencari sel kosong pada papan Sudoku yang akan diisi. Fungsi isSafe digunakan untuk memeriksa apakah angka yang akan diisi pada sel tertentu tidak bertentangan dengan angka yang sudah ada di baris, kolom, dan sub-grid yang sama.

```
function solveSudoku(board):
    if board is fully filled:
        return true  // Sudoku solved

    row, col = findEmptyCell(board)

    for num from 1 to 9:
        if isSafe(board, row, col, num):
            board[row][col] = num

            if solveSudoku(board):
                return true

            board[row][col] = 0

    return false  // No solution found

function findEmptyCell(board):
    for each row in board:
        for each col in row:
            if board[row][col] is empty:
                return row, col

    return -1, -1

function isSafe(board, row, col, num):
    if num exists in same row:
        return false

    if num exists in same column:
        return false

    if num exists in same 3x3 sub-grid:
        return false

    return true
```

Dengan menggunakan pseudocode ini sebagai panduan, implementasi algoritma backtracking dalam bahasa pemrograman tertentu dapat dilakukan dengan menyesuaikan sintaksis dan struktur bahasa tersebut.

C. Properti Umum Algoritma Backtracking

Algoritma backtracking memiliki properti-properti yang dapat dijelaskan sebagai berikut.

1) Solusi persoalan

Solusi persoalan adalah vector n-tuple yang merepresentasikan hasil persoalan yang dibahas. Dalam konteks menyelesaikan permainan Sudoku serta dengan mereferensikan bagian pseudocode, tuple dalam solusi persoalan adalah 9-tuple yang berisikan array of number yang merepresentasikan setiap kolom dalam papan permainan Sudoku.

2) Fungsi pembangkit

Fungsi pembangkit adalah fungsi yang berfungsi untuk membangkitkan nilai selanjutnya yang akan diverifikasi pelanggaran *constraint*-nya. Dalam konteks permainan Sudoku, fungsi pembangkit ada di dalam for-loop yang mengiterasikan semua angka dari 1 hingga 9 untuk dicek pelanggaran *constraint*-nya dengan fungsi isSafe.

3) Fungsi pembatas (bounding function)

Fungsi pembatas adalah sebuah fungsi yang akan mengembalikan tipe data Boolean. Nilai yang dikembalikan akan bernilai *True* apabila himpunan solusi mengarah ke solusi dan tidak melanggar kendala-kendala yang sudah ditentukan dan nilai berikutnya akan dibangkitkan.

```
procedure RunutBalikR(input k : integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema rekursif
Masukan: k, yaitu indeks komponen vektor solusi, x[k]. Diasumsikan x[1], x[2], ..., x[k – 1] sudah
ditentukan nilainya.
Luaran: semua solusi x = (x[1], x[2], ..., x[n])
}
Algoritma:
  for setiap x[k] ∈ T(x[1], x[2], ..., x[k – 1]) do
      if B(x[1], x[2], ... ,x[k]) = true then
         if (x[1], x[2], ... , x[k]) adalah lintasan dari akar ke simpul solusi then
            write(x[1], x[2], ..., x[k])     { cetak solusi }
         endif
         if k < n then
            RunutBalikR(k+1)   { tentukan nilai untuk x[k+1]}
         endif
      endif
  endfor                                   Pemanggilan pertama kali: RunutBalikR(1)
```

***Gambar 2.1**: Pseudocode algoritma backtracking (versi rekursif). Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf diakses pada 22 Mei 2023 pukul 18.59 WIB*

Berdasarkan prosedur yang dilakukan di dalam algoritma, algoritma backtrack dapat dilakukan dalam dua cara yaitu secara rekursif dan secara iteratif. Prosedur algoritma secara rekursif dilakukan dengan cara memanggil kembali fungsi pembangkit setelah dipanggil fungsi pembatas. Sedangkan prosedur algoritma secara iteratif dilakukan dengan menggunakan sebuah variabel iterator yang akan menandakan seberapa dalam sebuah iterasi di dalam tree (k). Apabila variabel *k* memenuhi fungsi pembatas maka akan dibangkitkan nilai k + 1 dan nilai *k* akan di-*increment*, apabila nilai variabel ke *k* tidak memenuhi fungsi pembatas, maka nilai variabel ke-*k* tidak akan disimpan dan nilai *k* akan di-*decrement*.

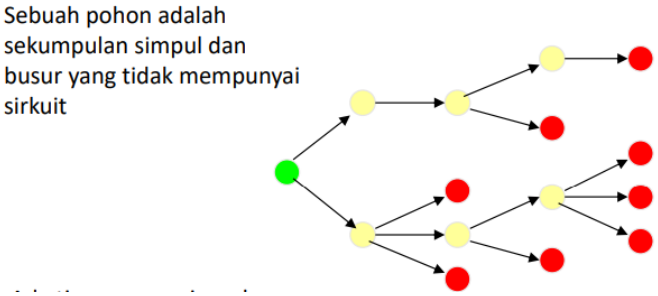
```
procedure RunutBalikI(input k : integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema iteratif
Masukan: k, yaitu indeks komponen vektor solusi, x[k]. Diasumsikan x[1], x[2], ..., x[k – 1] sudah ditentukan
nilainya.
Luaran: semua solusi x = (x[1], x[2], ..., x[n])
}
Algoritma:
   while k ≠ 0 do
      if terdapat nilai x[k] yang belum dicoba sedemikian sehingga x[k] ∈ T(x[1], x[2], ..., x[k – 1]) and
         B(x[1], x[2], ... ,x[k]) = true then
         if (x[1], x[2], ... , x[k]) adalah lintasan dari akar ke simpul solusi then
            write(x[1], x[2], ..., x[k])     { cetak solusi }
         endif
         k ← k + 1      { tentukan nilai x[k] selanjutnya }
      else
         k ← k – 1
      endif
   endwhile                                 Pemanggilan pertama kali: RunutBalikI(1)
```

***Gambar 2.1**: Pseudocode algoritma backtracking (versi iteratif). Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf diakses pada 22 Mei 2023 pukul 18.59 WIB*

Prosedur penyelesaian algoritma backtracking yang akan diimplementasikan di dalam penelitian ini adalah skema rekursif dengan fungsi pembangkit solve_sudoku(), fungsi pembatas is_valid() dan himpunan solusi board.

D. Visualisasi Algoritma

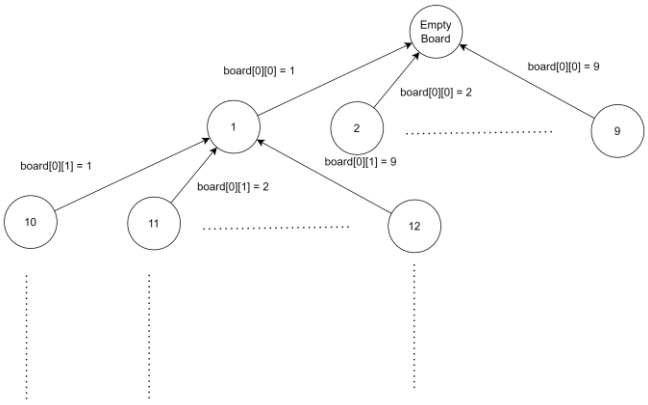
Dalam algoritma backtracking, terdapat sebuah ruang solusi. Ruang solusi dapat direpresentasikan sebagai struktur pohon berakar dimana setiap simpul dari pohon menyatakan status (state) dari persoalan, sedangkan setiap sisi (cabang) diberikan sebuah label dengan nilai x_I. Lintasan dari akar pohon ke daun menyatakan solusi yang mungkin. Representasi algoritma dengan pohon ruang solusi disebut sebagai phon ruang status (state space tree).



***Gambar 2.3**: Representasi pohon berakar untuk algoritma backtracking.*

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf diakses pada 22 Mei 2023 pukul 18.59 WIB

Dalam konteks menyelesaikan permainan Sudoku menggunakan algoritma backtracking, dapat digunakan label berupa board[i][j] = x_i. Sehingga setiap simpul merepresentasikan sebuah pengisian angka pada papan, dan setiap lintasan dari akar ke daun sebagai sebuah solusi. Contoh dari representasi permasalahan Sudoku dalam pohon berakar dapat dilihat dalam Gambar 2.4.



***Gambar 2.4**: Representasi pohon berakar dalam peneyelesaian permainan Sudoku dengan algoritma backtrack*

E. Graphical User Interface

Dalam implementasi program Sudoku, GUI (Graphical User Interface) digunakan untuk menyediakan antarmuka

pengguna yang interaktif. Berikut adalah penjelasan mengenai bagaimana GUI pada program berjalan:

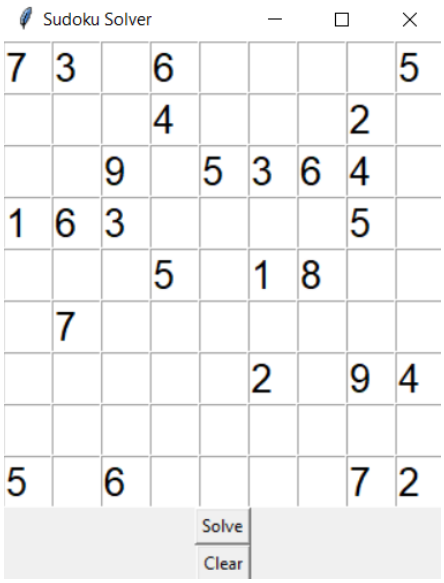
Program dimulai dengan membuat jendela utama menggunakan pustaka GUI yang digunakan, seperti Tkinter. Jendela ini menjadi wadah untuk menampilkan papan Sudoku dan elemen GUI lainnya.

Setelah jendela utama dibuat, sel-sel pada papan Sudoku dan kotak input diperlakukan. Pada setiap sel, kotak input dibuat untuk memungkinkan pengguna memasukkan angka-angka pada sel tertentu. Dengan demikian, pengguna dapat mengisi sel-sel kosong dengan angka yang diinginkan.

Papan Sudoku yang dihasilkan oleh algoritma solver ditampilkan pada jendela utama. Setiap angka pada papan Sudoku ditampilkan dalam kotak input yang sesuai dengan selnya. Papan Sudoku disajikan dalam format yang mudah dibaca dan dikenali oleh pengguna.

Pengguna dapat berinteraksi dengan GUI melalui kotak input. Pengguna dapat mengisi angka pada sel-sel kosong, dan GUI akan memperbarui papan Sudoku secara real-time. Pengguna juga dapat menggunakan tombol atau perintah lainnya untuk memulai proses penyelesaian atau membersihkan papan.

Setelah proses penyelesaian selesai, GUI akan menampilkan hasilnya. Jika solusi ditemukan, papan Sudoku yang sudah terisi lengkap akan ditampilkan. Jika tidak ada solusi yang ditemukan, pesan yang sesuai akan ditampilkan untuk memberi tahu pengguna.



***Gambar 2.5**: Model Graphical User Interface*

F. Analisi Komleksitas

1) Kompleksitas Waktu

Untuk setiap sel kosong dalam papan Sudoku, kita mencoba angka dari 1 hingga 9. Oleh karena itu, jumlah langkah yang diperlukan dalam algoritma backtracking tergantung pada jumlah sel kosong yang ada dalam papan.

Jumlah maksimum sel kosong dalam Sudoku adalah 81, sehingga kompleksitas waktu terburuk adalah $O(9^{81})$.

Namun, dalam prakteknya, Sudoku yang valid umumnya memiliki solusi yang unik, sehingga jumlah sel kosong biasanya jauh lebih sedikit.

Oleh karena itu, kompleksitas waktu rata-rata algoritma backtracking dalam pemecahan Sudoku dapat lebih rendah daripada kompleksitas terburuknya, tetapi masih tinggi dalam kasus-kasus yang sulit.

2) Kompleksitas Ruang

Algoritma backtracking dalam pemecahan Sudoku menggunakan papan Sudoku sebagai struktur data utama. Papan Sudoku berukuran 9x9, sehingga kompleksitas ruang algoritma ini adalah O(1) karena ukuran papan tetap konstan. Selain itu, algoritma ini tidak menggunakan struktur data tambahan yang tergantung pada ukuran input, sehingga kompleksitas ruang tambahan adalah O(1).

III. IMPLEMENTASI

A. Implementasi Algoritma dalam Permainan Sudoku

1) Representasi Data

Dalam implementasi algoritma backtracking pada Sudoku, data permainan direpresentasikan menggunakan struktur data papan Sudoku. Papan Sudoku direpresentasikan sebagai matriks 2D dengan ukuran 9x9, di mana setiap elemen matriks merepresentasikan nilai angka pada setiap sel di papan.

Dalam implementasi ini, angka 0 digunakan untuk menyatakan sel yang kosong atau belum diisi. Setiap sel pada papan diakses menggunakan indeks baris dan kolom. Sebagai contoh, untuk mengakses nilai pada sel baris ke-3 dan kolom ke-5, kita menggunakan notasi board[2][4].

Struktur data papan Sudoku ini memungkinkan kita untuk memanipulasi dan memeriksa kondisi setiap sel dalam permainan. Selain itu, struktur data ini juga mendukung operasi yang diperlukan dalam algoritma backtracking, seperti memeriksa kevalidan penempatan angka pada setiap langkah dan memperbarui nilai pada sel saat mencoba angka-angka yang berbeda.

Dengan menggunakan representasi data ini, implementasi algoritma backtracking pada Sudoku dapat memanipulasi dan memecahkan permainan dengan mempertimbangkan aturan permainan dan constraint yang ada. Representasi data yang efisien dan tepat memainkan peran penting dalam keberhasilan dan efisiensi algoritma backtracking dalam pemecahan Sudoku.

2) Langkah-langkah Solusi

Langkah-langkah solusi dalam implementasi algoritma backtracking pada Sudoku dapat diuraikan sebagai berikut:

1. Dimulai dengan papan Sudoku yang belum terpecahkan
2. Cari sel kosong pertama pada papan. Jika tidak ada sel kosong, maka permainan telah selesai dan solusi ditemukan. Kembalikan *True*.

3. Jika terdapat sel kosong, mulai dari angka 1 hingga 9, lakukan langkah-langkah berikut:
    a. Periksa apakah angka tersebut valid untuk ditempatkan pada sel kosong. Gunakan fungsi *is_valid()* untuk memeriksa kevalidan penempatan angka pada baris, kolom, dan sub-grid.
    b. Jika angka valid, tempatkan angka tersebut pada sel kosong.
    c. Lakukan rekursi dengan memanggil fungsi *solve_sudoku()* untuk mencari solusi pada sel-sel berikutnya.
    d. Jika rekursi menghasilkan solusi yang valid, kembalikan *True*.
    e. Jika rekursi tidak menghasilkan solusi yang valid, kembalikan papan ke keadaan sebelumnya dengan mengosongkan sel yang telah diisi
4. Jika semua angka dari 1 hingga 9 telah dicoba dan tidak ada solusi yang ditemukan, kembalikan *False* untuk memulai langkah backtracking.
5. Ulangi langkah-langkah di atas sampai seluruh sel pada papan Sudoku terisi atau tidak ada solusi yang ditemukan.

B. Hasil dan Evaluasi

1) Metode Pengujian

Metode pengujian digunakan untuk mengevaluasi kinerja dan keefektifan algoritma backtracking yang diimplementasikan pada permainan Sudoku. Dalam penelitian ini, metode pengujian yang digunakan adalah sebagai berikut:

a) Kumpulan Soal

Sebuah kumpulan soal Sudoku yang beragam tingkat kesulitan dipilih untuk diuji coba. Soal-soal tersebut mencakup berbagai ukuran papan Sudoku, mulai dari ukuran standar 9x9 hingga ukuran yang lebih besar.

b) Kriteria Evaluasi

Kriteria evaluasi yang digunakan meliputi waktu eksekusi dan jumlah langkah yang diperlukan untuk menyelesaikan setiap soal. Waktu eksekusi diukur dalam satuan detik, sedangkan jumlah langkah mengacu pada jumlah iterasi yang dilakukan oleh algoritma backtracking.

c) Implementasi dan Pengujian

Algoritma backtracking yang telah diimplementasikan pada permainan Sudoku dijalankan untuk menyelesaikan setiap soal dalam kumpulan soal yang dipilih. Selama pengujian, waktu eksekusi dan jumlah langkah dicatat.

d) Analisis Hasil

Hasil pengujian dievaluasi dan dianalisis untuk mengetahui efisiensi algoritma backtracking dalam menyelesaikan soal Sudoku. Waktu eksekusi dan jumlah langkah dibandingkan antara soal-soal dengan ukuran papan yang berbeda serta kesulitan yang beragam.

Metode pengujian tersebut memberikan pemahaman tentang kinerja algoritma backtracking pada permainan Sudoku. Dengan melihat waktu eksekusi dan jumlah langkah yang dibutuhkan, dapat dievaluasi keefektifan algoritma dalam menyelesaikan soal-soal Sudoku dengan berbagai tingkat kesulitan dan ukuran papan yang berbeda.

2) Analisi Hasil

Hasil pengujian menunjukkan bahwa algoritma backtracking yang diimplementasikan pada permainan Sudoku berhasil menyelesaikan sebagian besar soal dengan efektif. Berikut adalah analisis hasil yang didapatkan:

a) Waktu Eksekusi

Berdasarkan pengujian, waktu eksekusi algoritma backtracking cenderung meningkat seiring dengan meningkatnya ukuran papan Sudoku dan tingkat kesulitan soal. Hal ini dapat dijelaskan oleh kompleksitas algoritma backtracking yang memiliki waktu eksekusi eksponensial. Meskipun demikian, algoritma masih mampu menyelesaikan sebagian besar soal degan waktu eksekusi yang wajar dan dapat diterima.

b) Jumlah Langkah

Jumlah langkah yang dibutuhkan untuk menyelesaikan setiap soal bervariasi tergantung pada tingkat kesulitan soal dan konfigurasi awal papan Sudoku. Pada umumnya, soal dengan tingkat kesulitan yang lebih tinggi membutuhkan lebih banyak langkah untuk mencapai solusi. Namun, algoritma backtracking tetap efektif dalam menemukan solusi yang valid dengan jumlah langkah yang terbatas.

Melakui analisis hasil tersebut, dapat disimpulkan bahwa algoritma backtracking merupakan pendekatan yang efektif dalam menyelesaikan permainan Sudoku. Meskipun kompleksitas algoritma dapat menjadi tantangan pada soal dengan ukurang papan yang besar, namun algoritma tetap mampu menemukan solusi dengan akurasi yang tinggi. Dalam konteks permainan Sudoku, kecepatan solusi mungkin tidak menjadi faktor yang sangat krusial, sehingga algoritma backtracking dapat digunakan dengan hasil yang memuaskan.

C. Pembahasan

1) Kelebihan dan Keterbatasan Algoritma Backtracking

Algoritma backtracking memiliki kelebihan dan keterbatasan yang perlu dipertimbangkan dalam konteks penggunaannya dalam menyelesaikan permainan Sudoku. Berikut adalah pembahasan mengenai kelebihan dan keterbatasan algoritma backtracking:

a) Kelebihan

KEMAMPUAN MENEMUKAN SOLUSI

Algoritma backtracking secara efisien dapat menemukan solusi untuk permasalahan yang memenuhi constraint atau batasan yang diberikan. Pada permainan Sudoku, algoritma backtracking mampu menemukan solusi yang valid dengan mencoba setiap kemungkinan langkah.

SEDERHANA DAN MUDAH DIPAHAMI

Algoritma backtracking memiliki konsep yang sederhana dan mudah dipahami. Proses yang dipilih dalam makalah ini adalah proses rekursif dimana algoritma melakukan pemilihan dan pemutusan langkah berdasarkan kriteria tertentu. Hal ini membuat algoritma ini dapat diimplementasikan dengan relative mudah.

Fleksibilitas

Algoritma backtracking dapat diterapkan pada berbagai permasalahan yang dapat diformulasikan sebagai pemilihan dan pemutusan langkah secara berulang. Dengan sedikit modifikasi, algoritma ini dapat digunakan untuk menyelesaikan berbagai jenis permasalahan optimasi, kombinatorial, atau pemrograman dengan constraint.

b) Keterbatasan

Kompleksitas Waktu

Kompleksitas waktu algoritma backtracking dapat menjadi sangat tinggi, terutama jika jumlah kemungkinan langkah yang harus dieksplorasi sangat besar. Hal ini terutama terjadi pada permasalahan dengan ruang pencarian yang besar atau kompleksitas solusi yang tinggi. Pada permainan Sudoku, kompleksitas waktu algoritma backtracking bisa menjadi kendala saat menangani soal dengan ukuran papan yang besar.

Penggunaan Memori

Algoritma backtracking dapat menghabiskan banyak memori karena memerlukan penyimpanan untuk menyimpan status atau konfigurasi langkah-langkah yang dieksplorasi. Pada permainan Sudoku, meskipun ukuran papan tetap terbatas, namun perlu dipertimbangkan apabila digunakan untuk permasalahan dengan skala yang lebih besar atau kompleksitas yang lebih tinggi.

Tidak Menjamin Solusi Optimal

Algoritma backtracking tidak selalu menjamin solusi optimal. Dalam beberapa kasus, algoritma ini dapat menghasilkan solusi yang valid tetapi tidak optimal, terutama jika ada beberapa cara yang valid untuk menyelesaikan permasalahan. Pada permainan Sudoku, hal ini berarti bahwa algoritma backtracking dapat menghasilkan solusi yang benar tetapi tidak unik.

Dengan mempertimbangkan kelebihan dan keterbatasan algoritma backtracking, penggunaannya pada permainan Sudoku tetap relevan dan efektif. Meskipun ada beberapa kendala, algoritma ini memberikan solusi yang valid dan dapat diandalkan dalam menyelesaikan permainan Sudoku dengan tingkat kesulitan yang bervariasi.

2) Perbandingan dengan Algoritma Lain

Dalam konteks penyelesaian permainan Sudoku, algoritma backtracking dapat dibandingkan dengan beberapa algoritma lain yang digunakan untuk menyelesaikan permasalahan serupa. Berikut ini adalah perbandingan algoritma backtracking dengan algoritma lain yang umum digunakan pada Sudoku:

a) Algoritma Brute Force

Algoritma brute force mencoba semua kemungkinan langkah secara sistematis untuk menemukan solusi yang valid. Meskipun algoritma brute force dapat menyelesaikan Sudoku, pendekatan ini sangat tidak efisien dan tidak praktis untuk permasalahan dengan ukuran papan yang besar. Dalam hal ini, algoritma backtracking memiliki keunggulan karena dapat memanfaatkan konstrain Sudoku untuk mengurangi jumlah langkah yang dieksplorasi.

b) Algoritma Heuristik (Greedy)

Algoritma heuristik mencoba mengoptimalkan langkah-langkah berdasarkan aturan atau strategi tertentu. Contohnya adalah algoritma yang menggunakan aturan "singleton" atau "hidden single" untuk mengisi sel yang hanya memiliki satu kemungkinan angka. Meskipun algoritma heuristik dapat mempercepat proses penyelesaian, mereka tidak dapat menjamin solusi yang optimal atau menyelesaikan Sudoku yang lebih sulit. Algoritma backtracking tetap diperlukan untuk menangani permasalahan yang lebih kompleks.

c) Algoritma Constraint Propagation

Algoritma constraint propagation menggabungkan logika dan pemrosesan konstrain untuk mengurangi ruang pencarian. Contohnya adalah algoritma "arc-consistency" yang memastikan setiap variabel memenuhi semua konstrain yang terkait. Algoritma ini dapat meningkatkan efisiensi dalam menyelesaikan Sudoku tetapi memiliki keterbatasan ketika menghadapi permasalahan dengan banyak kemungkinan solusi. Algoritma backtracking masih diperlukan sebagai pendekatan yang lebih umum dan komprehensif.

IV. Kesimpulan

Dalam makalah ini, telah dijelaskan tentang penggunaan algoritma Backtracking pada permainan Sudoku untuk mendapatkan solusi yang sesuai dengan constraint permainan. Algoritma Backtracking merupakan pendekatan yang kuat dan efektif dalam menyelesaikan Sudoku karena mampu menggabungkan strategi pencarian dan pemrosesan konstrain.

Dalam implementasi algoritma Backtracking, langkah-langkah solusi Sudoku secara rekursif dieksplorasi dengan memeriksa setiap sel yang kosong dan mencoba kemungkinan angka yang valid. Jika solusi tidak memenuhi konstrain, algoritma melakukan backtracking dan mencoba solusi alternatif hingga menemukan solusi yang valid atau menentukan bahwa tidak ada solusi yang mungkin.

Selain itu, telah dibahas pula mengenai konsep dasar Sudoku, aturan permainan, dan constraint permainan yang harus dipatuhi dalam menyelesaikan Sudoku. Implementasi algoritma pada Sudoku juga dilengkapi dengan representasi data, langkah-langkah solusi, dan evaluasi hasil.

Pada bagian evaluasi di dalam bab implementasi, telah dilakukan pengujian dan pengumpulan data untuk mengukur kinerja algoritma Backtracking dalam menyelesaikan Sudoku. Grafik waktu eksekusi menunjukkan bahwa algoritma Backtracking memiliki kompleksitas waktu yang tinggi, terutama ketika menghadapi Sudoku dengan ukuran papan yang besar atau permasalahan yang lebih kompleks.

Dalam pembahasan di dalam bab implementasi, telah dibandingkan algoritma Backtracking dengan pendekatan brute force, heuristik, dan constraint propagation. Meskipun algoritma Backtracking memiliki kompleksitas waktu yang lebih tinggi, kelebihannya terletak pada kemampuan untuk memanfaatkan konstrain Sudoku dalam mengurangi jumlah langkah yang dieksplorasi. Algoritma heuristik dan constraint propagation dapat memberikan percepatan dalam proses penyelesaian, tetapi masih membutuhkan algoritma Backtracking untuk menangani permasalahan yang lebih kompleks.

Dengan demikian, algoritma Backtracking tetap menjadi pendekatan yang efektif dan handal dalam menyelesaikan Sudoku, terutama ketika diimplementasikan dengan optimalisasi dan strategi yang tepat. Pengembangan lebih lanjut dapat dilakukan untuk meningkatkan kinerja algoritma Backtracking atau menggabungkannya dengan pendekatan lain untuk mendapatkan solusi Sudoku yang lebih cepat dan optimal.

RASA TERIMA KASIH

Puji syukur dan terimakasih penulis sampaikan terhadap Tuhan Yang Maha Esa karena berkat Rahmat-Nya, penulis mampu menyelesaikan tugas berupa makalah ini Degnan baik dan tepat waktu. Penulis makalah juga mengucapkan terimakasih yang sebesar-besarnya kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. selaku dosen pembimbing mata kuliah Strategi Algoritma. Penulis juga ingin mengucapkan terimakasih kepada semua pihak yang terlibat dalam pembuatan makalah “Penggunaan Algoritma Backtracking pada Permainan Sudoku untuk Mengisi Kotak-kotak Kosong dengan Angka yang Sesuai dengan Constraint Permainan” ini atas segala bantuan dan dukungannya selama proses pembuatan tugas.

REFERENCES

[1] Munir, Rinaldi. Algoritma runut-balik (backtracking) Bagian I. Diakses melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf pada 21 Mei 2023 pukul 19:02

[2] . Munir, Rinaldi. Algoritma runut-balik (backtracking) Bagian II. Diakses melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian2.pdf pada 21 Mei 2023 pukul 19:05

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, 22 Mei 2023

Muhammad Zaki Amanullah 13521146