Implementasi dan Analisis Algoritma Runut-balik (*Backtracking*) dalam Penyelesaian Permainan Kakurasu

Fardhan Indrayesa - 12821046
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: findrayesa@gmail.com , 12821046@mahasiswa.itb.ac.id

***Abstrak*—Kakurasu merupakan permainan logika berbasis teka-teki yang menantang kemampuan berpikir sistematis. Penelitian ini mengimplementasikan algoritma runut-balik (*backtracking*) untuk menyelesaikan kakurasu, dengan memanfaatkan prinsip pencarian *depth-first* dan fungsi pembatas untuk memangkas ruang solusi. Representasi permainan digunakan dalam bentuk vektor biner, dengan setiap elemen menunjukkan status sel dalam papan. Hasil menunjukkan bahwa algoritma efektif untuk papan berukuran kecil (hingga 5×5), namun mengalami peningkatan waktu komputasi dan simpul yang dikunjungi secara eksponensial, dengan kompleksitas waktu $O(n \cdot 2^N)$. Efisiensi algoritma juga dipengaruhi oleh urutan pembangkitan kandidat dan nilai angka kunci; urutan {0, 1} terbukti lebih efisien dibandingkan {1, 0}, terutama pada angka kunci yang kecil.**

Kata kunci—kakurasu; permainan; algoritma; runut-balik;

I. PENDAHULUAN

Permainan logika merupakan permainan yang dirancang untuk melatih otak sehingga mengharuskan pemain berpikir dengan logika. Permainan ini merupakan permainan teka-teki yang dapat merangsang jalan berpikir dan kendali pada otak manusia [3]. Saat ini, permainan logika masih banyak yang memainkannya karena selain populer, jenis permainan ini memiliki banyak manfaat untuk perkembangan otak. Contoh permainan yang menggunakan logika yang sudah populer dari dulu hingga saat ini adalah sudoku, *minesweeper*, kakurasu, nonogram, dll. Permainan-permainan ini dapat diselesaikan dengan logika dasar yang mampu meningkatkan fungsi kognitif seseorang dalam memecahkan suatu masalah [9]. Salah satu cara yang dapat digunakan untuk menyelesaikan permainan berbasis logika adalah menggunakan prosedur atau algoritma yang tepat agar persoalan dapat diselesaikan dengan efektif dan efisien. Contoh algoritma yang dapat digunakan untuk menyelesaikan permainan-permainan ini adalah algoritma yang berbasis pencarian dan enumerasi terhadap ruang status, seperti *brute force*, *Breadth-First Search* (BFS), *Depth-First Search* (DFS), atau *backtracking*.

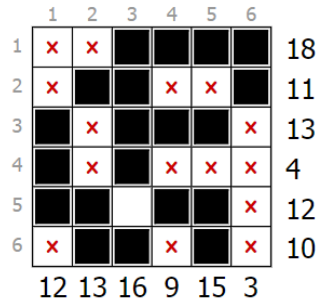
Salah satu permainan logika dari Jepang, yaitu kakurasu, adalah permainan yang menggunakan kemampuan matematika dasar untuk menyelesaikannya [2]. Permainan ini memiliki bentuk persegi yang memiliki angka di sisi kiri-kanan dan atas-bawahnya sebagai petunjuk. Permainan dianggap selesai apabila seluruh seluruh grid memiliki nilai yang bersesuaian dengan jumlah di sebelah kanan atau bawahnya. Penyelesaian kakurasu ini dapat dioptimalisasi menggunakan pendekatan algoritma pencarian yang efektif dan efisien. Beberapa contoh algoritma yang dapat digunakan untuk menyelesaikan permainan ini adalah algoritma *brute force* [10], *exhautive search*, dan DFS [2].

Pada algoritma *brute force*, semua kemungkinan konfigurasi akan dievaluasi secara menyeluruh di akhir, sehingga memerlukan waktu komputasi yang sangat besar untuk ukuran papan kakurasu yang besar. Pada algoritma *exhautive search*, mirip seperti brute force, semua kombinasi papan yang dieksplorasi dan dievaluasi adalah satu per satu di setiap langkah. Selain itu, algoritma DFS memeriksa seluruh kombinasi papan secara mendalam sampai ditemukan solusi, sehingga algoritma DFS juga masih belum efisien dalam menyelesaikan kakurasu. Oleh karena itu, pada penelitian ini akan dilakukan implementasi algoritma *backtracking* yang memiliki basis pencarian DFS dan merupakan pengembangan dari algoritma *exhautive search* untuk menyelesaikan permainan kakurasu.

II. KAJIAN LITERATUR

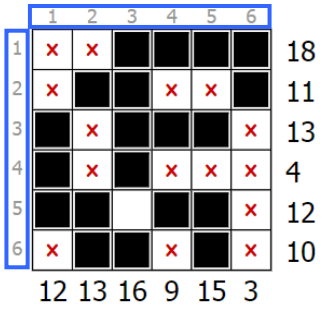
A. Kakurasu

Kakurasu adalah permainan asal Jepang yang menggunakan kemampuan matematika dasar untuk menyelesaikannya [2]. Penyelesaian permainan ini adalah dengan membuat beberapa kotak hitam sedemikian rupa sehingga kotak hitam pada setiap baris memiliki jumlah yang sama dengan angka yang terdapat di sebelah kanan dan di bawah tabel kakurasu (angka kunci). Selain itu, jika kotak hitam berada di posisi pertama pada baris/kolom, maka nilainya adalah 1. Jika berada pada baris pertama dan kolom kedua (atau baris kedua dan kolom pertama), maka nilainya adalah 2 [4].



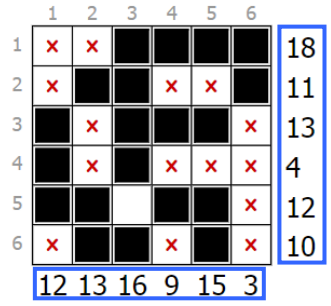
Gambar 1. Contoh papan permainan kakurasu ukuran 6 × 6 [4].

Permainan kakurasu ini memiliki berbagai macam ukuran, mulai dari ukuran 4 × 4, 5 × 5, hingga 9 × 9 atau lebih. Pada Gambar 1, diperlihatkan contoh ukuran kakurasu untuk ukuran 6 × 6. Berdasarkan gambar tersebut, sel yang berwarna hitam menunjukkan sel-sel yang dipilih sebagai bagian dari penjumlahan dalam suatu baris atau kolom, yang hasilnya harus sesuai dengan angka yang tertera di sisi kanan baris atau sisi bawah kolom tersebut. Sedangkan, sel yang ditandai dengan tanda silang merah, menunjukkan bahwa sel tersebut tidak akan dipilih dari penjumlahan dalam suatu baris atau kolom.



Gambar 2. Contoh papan permainan kakurasu ukuran 6 × 6. Kotak berwarna biru menunjukkan angka yang bersesuaian dengan kotak yang dipilih sebagai elemen penjumlahan [4].

Selain itu, berdasarkan Gambar 2, kotak berwarna biru menunjukkan angka-angka yang bersesuaian dengan sel yang sudah ditandai sebagai elemen penjumlahan dalam satu baris atau kolom. Angka ini merupakan salah satu petunjuk untuk menyelesaikan permainan kakurasu.



Gambar 3. Contoh papan permainan kakurasu ukuran 6 × 6. Kotak berwarna biru menunjukkan jumlah yang harus dipenuhi pada setiap baris atau kolom berdasarkan sel yang sudah ditandai dengan kotak hitam [4].

Petunjuk lain yang ada pada permainan ini tertera pada Gambar 3, yaitu angka-angka yang berada pada kotak biru menunjukkan jumlah total berdasarkan angka yang bersesuaian pada sel yang ditandai dengan kotak berwarna hitam.

Penyelesaian permainan ini terlihat seperti pada Gambar 1, yang menunjukkan angka yang bersesuaian dengan sel berwarna hitam memiliki jumlah yang sama dengan angka yang berada di sebelah kanan ataupun bawah papan kakurasu. Misal, pada baris pertama sel yang ditandai dengan warna hitam adalah sel kolom 3, 4, 5, dan 6. Jumlah dari keempat angka ini sudah sesuai dengan angka pada sebelah kanan papan, yaitu 18. Begitu pula untuk kolom pertama, sel yang ditandai dengan warna hitam adalah baris 3, 4, dan 5, sehingga jumlahnya sama dengan angka yang berada di bawah papan kakurasu pada kolom pertama, yaitu 12.

Namun, pada kasus lain, angka-angka yang digunakan agar sesuai dengan jumlah petunjuknya tidak selalu terdiri dari angka-angka yang sama pada kasus ini. Misal, pada kolom pertama, untuk mencapai jumlah angka 12, tidak harus menjumlahkan angka 3, 4, dan 5, tetapi bisa saja solusinya pada kasus lain adalah penjumlahan dari angka 2, 4, dan 6.

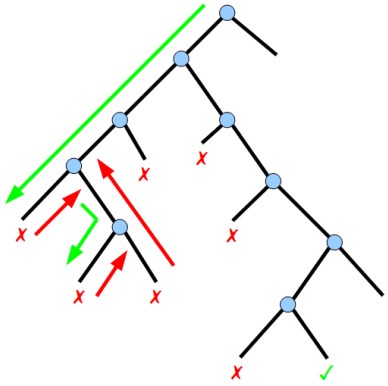
B. Algoritma Backtracking

Algoritma runut-balik (*backtracking*) merupakan algoritma yang termasuk ke dalam strategi algoritma berbasis pencarian pada ruang status [6]. *Backtracking* dapat dipandang sebagai sebuah fase dalam algoritma traversal DFS atau sebagai sebuah metode pemecahan masalah yang efisien, terstruktur, dan sistematis, baik untuk persoalan optimasi maupun non-optimasi. Terdapat beberapa komponen dalam algoritma ini [5, 7], yaitu sebagai berikut

- Kandidat: Sebuah kandidat merupakan pilihan atau elemen yang mungkin dapat ditambahkan pada solusi saat ini.
- Solusi: Solusi adalah sebuah konfigurasi persoalan yang valid dan lengkap, serta memenuhi semua kendala-kendala (*constraints*) masalah dan dapat dinyatakan menjadi vektor solusi.
- Solusi parsial: Sebuah solusi parsial merupakan konfigurasi yang belum lengkap dan sedang dibangun atau diproses saat penelusuran dengan *backtracking* dilakukan.
- *Decision space*: *Decision space* adalah himpunan semua kandidat atau pilihan yang mungkin pada setiap *decision point*.
- *Decision point*: *Decision point* adalah langkah saat kandidat dipilih dan ditambahkan ke solusi parsial.
- *Feasible solution*: *Feasible solution* adalah solusi parsial atau lengkap yang layak, yaitu solusi yang memenuhi semua kendala (*constraints*).
- *Dead end*: *Dead end* (jalan buntu) terjadi ketika solusi parsial tidak dapat diperluas karena telah melanggar kendala (*constraints*).

- *Backtrack*: Backtracking bertujuan untuk mengurung keputusan sebelumnya dan kembali ke *decision point* sebelumnya.
- *Search space*: *Search space* mencakup semua kemungkinan kombinasi kandidat dan beberapa pilihan konfigurasi.
- Solusi optimal: Dalam permasalahan optimasi, solusi optimal merupakan solusi terbaik yang mungkin.
- Fungsi pembangkit: Fungsi pembangkit merupakan fungsi yang digunakan untuk membangkitkan *decision space* pada setiap *decision point*.
- Fungsi pembatas (*bounding function*): Fungsi ini merupakan fungsi yang digunakan untuk memutuskan apakah suatu kandidat konfigurasi solusi memenuhi kendala (constraints) atau tidak. Jika memenuhi, artinya kandidat tersebut mengarah ke solusi. Sedangkan jika tidak memenuhi, artinya kandidat tersebut tidak mengarah ke solusi, sehingga pencarian dihentikan untuk konfigurasi tersebut.

Semua kemungkinan solusi dari suatu persoalan disebut sebagai ruang solusi (*solution space*). Ruang solusi ini diatur ke dalam struktur pohon berakar, dengan simpul menyatakan status (*state*) persoalan dan sisi (cabang) ditandai dengan nilai-nilai komponen vektor solusi. Lintasan yang dibentuk dari akar ke daun berdasarkan pohon berakar ini menyatakan solusi yang mungkin, sehingga lintasan dari akar ke daun dapat membentuk ruang solusi. Selain itu, pengaturan pohon ruang solusi ini disebut sebagai pohon ruang status (*state space tree*) yang dapat diilustrasikan seperti Gambar 4.



Gambar 4. Contoh pohon ruang status pada pencarian solusi menggunakan algoritma *backtracking*. *Node* (lingkaran) paling atas merupakan akar (*root*), yang menunjukkan kondisi awal permasalahan. *Node* biru di bawahnya merupakan kandidat solusi yang dibangkitkan dari node di atasnya. Panah berwarna hijau merupakan proses pencarian secara DFS dan berwarna merah merupakan proses mundur (*backtrack*) ke node (*parent*) di atasnya. Tanda silang merupakan kandidat yang melanggar kendala dan tanda centang merupakan kandidat solusi [8].

Algoritma ini melakukan pencarian dengan mencoba berbagai pilihan status dan mundur (*backtrack*) jika status tidak mengarah ke solusi. Metode pencarian ini bekerja dengan cara menelusuri semua kemungkinan solusi untuk suatu masalah secara rekursif. Pada awal pencarian, ambil status inisial sebagai solusi awal, lalu telusuri seluruh kemungkinan perluasan dari solusi tersebut. Apabila status hasil perluasan itu mengarah ke suatu solusi, algoritma ini akan mengembalikan solusi tersebut. Namun, apabila status hasil perluasan tersebut tidak mengarah ke solusi, maka algoritma ini akan mundur (*backtrack*) ke solusi sebelumnya dan mencoba kemungkinan perluasan status yang lain [7].

Algoritma *backtracking* banyak digunakan pada berbagai permasalahan, seperti menyelesaikan permainan *puzzle* (seperti sudoku, nonogram, kakurasu), mencari lintasan terpendek dalam labirin, permasalahan penjadwalan, permasalahan alokasi sumber daya, serta masalah kombinatorial yang lain [1].

III. METODOLOGI

A. Formulasi Masalah

Formulasi masalah pada penelitian ini terdiri dari representasi solusi permainan kakurasu menggunakan algoritma *backtracking*, fungsi pembangkit, dan fungsi pembatas. Representasi solusi yang digunakan adalah berupa vektor *n-tuple*, seperti $(x_1, x_2, x_3, \ldots, x_n)$, $x_i \in S_i = \{0, 1\}$, dengan $S_1 = S_2 = \ldots = S_n$ dan n merupakan banyak sel pada papan kakurasu, serta nilai 1 menandakan bahwa sel pada elemen solusi ke-i dipilih (sel ke-i ditandai kotak hitam). Elemen pertama pada vektor solusi tersebut bersesuaian dengan sel baris dan kolom pertama pada papan kakurasu, begitupun untuk sel di baris dan kolom yang lain.

Selain itu, fungsi pembangkit yang digunakan pada pencarian solusi kakurasu menggunakan *backtracking* adalah berupa pembangkitan status konfigurasi setiap sel yang mungkin, ditandai dengan berubahnya satu sel menjadi 0 atau 1 pada papan kakurasu. Sedangkan fungsi pembatas yang digunakan adalah aturan dari permainan kakurasu itu sendiri, yaitu jumlah setiap sel dalam satu baris/kolom tidak boleh lebih dari jumlah petunjuk angka pada kanan/bawah papan kakurasu. Jika jumlahnya lebih besar dari angka petunjuknya, maka tandai status papan tersebut menjadi simpul mati.

B. Struktur Data

Status konfigurasi setiap papan disimpan pada struktur data bertipe *n-tuple*, dengan n adalah banyaknya sel pada papan kakurasu. Data konfigurasi setiap status disimpan dalam struktur data bernama NodeBoard yang memiliki struktur sebagai berikut

```
STRUCTURE NodeBoard:
  board: array of integer
  path: 2D array of integer
  isBound: boolean (default is true)
  children: array of NodeBoard
END STRUCTURE
```

Struktur data ini memiliki atribut board yang menyimpan konfigurasi papan dalam bentuk *n-tuple*, atribut path menyimpan kandidat konfigurasi yang sudah dilalui dari simpul akar hingga simpul saat ini, atribut isBound menyimpan status apakah simpul ini dimatikan (*false*) atau tidak (*true*), dan atribut children yang menyimpan semua kandidat simpul saat ini yang mungkin.

C. Algoritma Backtracking dalam Permainan Kakurasu

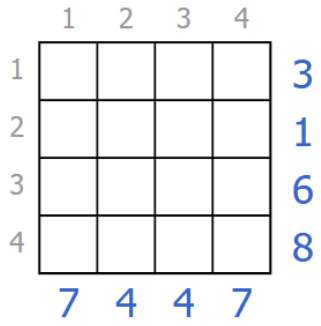
Pada penelitian ini, permainan kakurasu akan diselesaikan menggunakan algoritma backtracking dengan mengevaluasi langkah pada setiap sel papan kakurasu yang sudah dibangkitkan. Langkah-langkah penyelesaian kakurasu dengan algoritma backtracking secara umum adalah sebagai berikut.

- Mulai dengan papan kakurasu yang kosong.
- Bangkitkan kandidat solusi yang mungkin untuk sel saat ini, yaitu sel saat ini yang bernilai 0 atau 1.
- Periksa kandidat tersebut, apakah setelah dijumlahkan indeks elemen-elemennya menghasilkan jumlah yang sama dengan angka kunci (angka sebelah kanan/bawah papan).
- Apabila sama, hentikan pencarian, solusi ditemukan. Jika tidak sama, lanjutkan pencarian lalu evaluasi dengan fungsi pembatas, apakah jumlah indeks elemen-elemennya tidak lebih besar dari angka kunci.
- Apabila memenuhi, lanjutkan pencarian (ulangi dari langkah urutan kedua). Jika tidak, matikan simpul tersebut, lalu mundur (*backtrack*) ke kandidat yang lain.
- Ulangi langkah dari urutan kedua hingga ditemukan solusi.

Secara lebih detail, berikut merupakan pseudocode yang digunakan dalam menyelesaikan permainan kakurasu menggunakan algoritma *backtracking*.

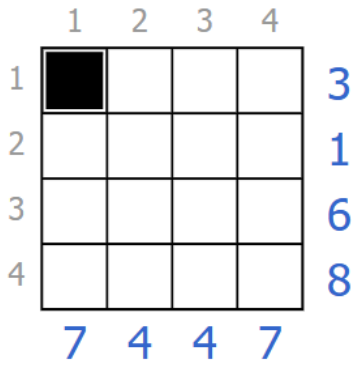
```
FUNCTION Solve(node, idx):
  IF idx == width * height THEN
    IF IsSolution(node.board) THEN
      RETURN true
    END IF
    RETURN false
  END IF

  FOR val IN [1, 0]:
    newBoard = COPY node.board
    newBoard[idx] = val

    child = CreateNodeBoard(newBoard, node.path)
    B(child)
    AddChild(node, child)

    IF child.isBound AND Solve(child, idx + 1)
THEN
      RETURN true
    END IF
  END FOR

  RETURN false
END FUNCTION
```

Sebagai contoh, misalkan terdapat papan kakurasu berukuran 4 × 4, sesuai yang tertera pada Gambar 5. Akan diselesaikan menggunakan algoritma *backtracking*. Konfigurasi papan kakurasu pada langkah ini merupakan akar dari pohon status yang ingin dibangkitkan kandidatnya.



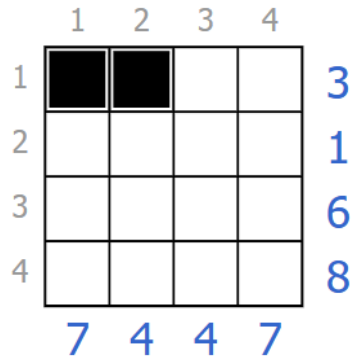
Gambar 5. Papan kakurasu berukuran 5 × 5 yang masih kosong [4].

Berdasarkan Gambar 6, tandai sel pertama, yaitu kolom dan baris pertama, dengan kotak hitam. Setelah itu evaluasi, apakah jumlah indeks yang sudah ditandai lebih besar dari angka kuncinya. Karena angka indeks yang ditandai bernilai 1 dan angka kunci pada baris dan kolom pertama adalah 3 dan 7, maka status ini mengarah ke solusi. Lanjutkan pencarian untuk sel berikutnya (baris pertama, kolom kedua).



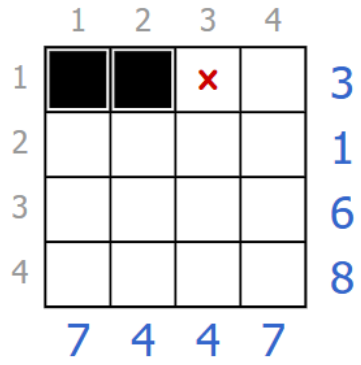
Gambar 6. Papan kakurasu berukuran 5 × 5 dengan sel pertama yang sudah ditandai [4].

Selanjutnya, untuk sel baris pertama dan kolom kedua, tandai dengan kotak hitam, seperti yang terlihat pada Gambar 7. Evaluasi jumlah baris dan kolomnya. Karena jumlah baris dan kolom masih memenuhi kondisi fungsi pembatas, maka status ini mengarah ke solusi. Lanjutkan pencarian untuk sel selanjutnya.



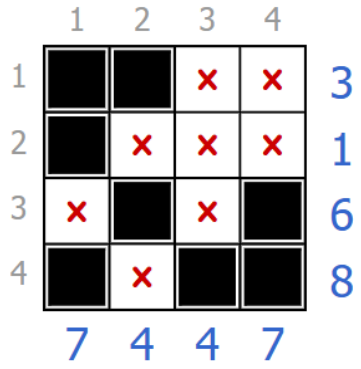
Gambar 7. Papan kakurasu berukuran 5 × 5 dengan sel kedua yang sudah ditandai [4].

Untuk sel ketiga, apabila ditandai dengan kotak hitam, jumlah pada baris pertama (6) akan tidak sesuai dengan jumlah angka kunci baris pertama (3), sehingga status tersebut tidak lagi mengarah ke solusi. Oleh karena itu, mundur (*backtrack*) ke status pada Gambar 7, tandai sel ketiga ini dengan tanda silang.



Gambar 8. Papan kakurasu berukuran 5 × 5 dengan sel ketiga yang sudah ditandai [4].

Begitupun untuk sel kelima, dan seterusnya hingga sebuah solusi ditemukan, sesuai dengan Gambar 9. Perhatikan bahwa kombinasi untuk beberapa angka yang lebih besar, seperti 6, memiliki kombinasi penjumlahan dari angka 2 dan 4 pada kasus ini. Namun, pada kasus lain, angka sel yang ditandai bisa saja berbeda, yaitu penjumlahan dari angka 1, 2, dan 3.



Gambar 9. Solusi papan kakurasu berukuran 5 × 5 [4].

IV. Hasil dan Pembahasan

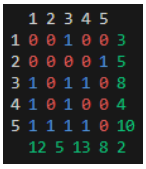
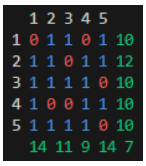
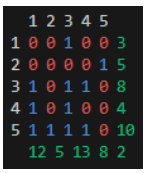
Setelah dilakukan implementasi algoritma *backtracking* dalam menyelesaikan permainan kakurasu, diperoleh hasil untuk berbagai ukuran dan tingkat kesulitan permainan kakurasu yang tertera pada Tabel I. Konfigurasi papan yang digunakan adalah papan berukuran 4 × 4 dan 5 × 5 saja, karena permainan kakurasu memiliki kemungkinan status sebanyak 2^n, dengan n adalah jumlah sel pada papan kakurasu, sehingga pada ukuran yang lebih besar, proses pencarian solusi dengan algoritma *backtracking* membutuhkan waktu komputasi yang sangat besar.

TABEL I. Hasil implementasi algoritma *backtracking* dalam penyelesaian kakurasu.

Konfigurasi Papan	**Hasil**	
Ukuran: 4 × 4 Kesulitan: Mudah		Jumlah simpul: 2616 Waktu: 56 ms
Ukuran: 4 × 4 Kesulitan: Sulit		Jumlah simpul: 4560 Waktu: 105 ms
Ukuran: 5 × 5 Kesulitan: Mudah		Jumlah simpul: 4594683 Waktu: 220766 ms
Ukuran: 5 × 5 Kesulitan: Sulit		Jumlah simpul: 39397 Waktu: 1293 ms

Tabel I memperlihatkan hasil yang diperoleh dalam implementasi algoritma backtracking dalam penyelesaian permainan kakurasu dengan berbagai ukuran papan dan tingkat kesulitan. Konfigurasi permainan dan tingkat kesulitan diperoleh dari laman puzzle-kakurasu.com [4]. Berdasarkan hasil pada Tabel I, terlihat bahwa secara umum, ukuran papan kakurasu sangat mempengaruhi waktu dan jumlah simpul pencarian. Pada ukuran 4 × 4 membutuhkan waktu kurang dari 1 detik dalam mencari solusi kakurasu, sedangkan pada ukuran 5 × 5 proses pencariannya dapat mencapai lebih dari 1 detik hingga lebih dari 1 menit. Hal ini mengindikasikan bahwa semakin besar ukuran papan kakurasu, maka waktu pencarian yang dibutuhkan semakin lama karena lebih banyak sel yang harus dievaluasi satu-per-satu dalam papan tersebut. Selain itu, tingkat kesulitan permainan kakurasu tidak terlalu berpengaruh terhadap waktu pencarian dan jumlah simpul yang dikunjungi. Hal ini dapat mengindikasikan bahwa selain tingkat kesulitan, terdapat faktor lain yang mempengaruhi waktu pencarian, seperti nilai angka kunci yang besar atau kecil, serta urutan pembangkitan status antara sel yang dihitamkan terlebih dahulu atau tidak.

TABEL II. HASIL IMPLEMENTASI ALGORITMA *BACKTRACKING* DALAM PENYELESAIAN KAKURASU DENGAN TINGKAT KESULITAN YANG SAMA, URUTAN PEMBANGKITAN KANDIDAT YANG BERBEDA, DAN ANGKA KUNCI YANG BERBEDA PADA URUTAN PEMBANGKITAN KANDIDAT YANG SAMA.

Konfigurasi Papan	Hasil	
Ukuran: 5 × 5 Urutan: {1, 0}		Jumlah simpul: 112354 Waktu: 3398 ms
Ukuran: 5 × 5 Urutan: {1, 0}		Jumlah simpul: 9444287 Waktu: 351215 ms
Ukuran: 5 × 5 Urutan: {0, 1}		Jumlah simpul: 44081 Waktu: 1379 ms
Ukuran: 5 × 5 Urutan: {0, 1}		Jumlah simpul: 9075646 Waktu: 335764 ms

Berdasarkan Tabel II, terlihat bahwa waktu yang dibutuhkan untuk menyelesaikan kakurasu pada angka kunci yang lebih besar membutuhkan waktu yang lebih lama pada kedua urutan pembangkitan kandidat ({1, 0} dan {0, 1}). Begitupun sebaliknya untuk angka kunci yang lebih kecil membutuhkan waktu yang lebih pendek pada kedua urutan pembangkitan kandidat ({0, 1} dan {1, 0}). Hal ini disebabkan oleh sifat algoritma *backtracking* yang melakukan pencarian secara rekursif atau menggunakan pendekatan DFS. Pada angka kunci yang kecil, jumlah sel yang perlu ditandai dengan nilai 1 lebih sedikit, sehingga ruang kombinasi yang dibangkitkan juga lebih sempit. Bahkan, jika fungsi pembatas diterapkan, simpul yang tidak memenuhi batas langsung dapat dieliminasi, sehingga simpul-simpul di tingkat lebih dalam tidak perlu diperiksa.

Sebaliknya, pada angka kunci yang besar, lebih banyak sel yang perlu bernilai 1 agar memenuhi syarat, sehingga kombinasi yang dibangkitkan menjadi lebih banyak. Dalam hal ini, meskipun fungsi pembatas digunakan, simpul-simpul dengan jumlah sementara yang masih di bawah target angka kunci tetap dianggap valid dan dilanjutkan eksplorasinya. Akibatnya, algoritma tetap menyusuri jalur tersebut hingga ke tingkat lebih dalam meskipun pada akhirnya tidak mengarah ke solusi, sehingga menyebabkan waktu penyelesaian menjadi lebih lama.

Selain itu, urutan pembangkitan kandidat juga mempengaruhi waktu yang dibutuhkan algoritma untuk menemukan solusi. Berdasarkan Tabel II, terlihat bahwa urutan pembangkitan kandidat {1, 0} lebih lama dibandingkan urutan {0, 1}. Pada urutan {0, 1}, jumlah simpul yang dikunjungi algoritma untuk menemukan solusi pada angka kunci yang lebih kecil memiliki jumlah 60,77% lebih sedikit daripada urutan {1, 0}. Sedangkan, pada angka kunci yang lebih besar, algoritma dengan urutan {0, 1} memiliki jumlah simpul 3,9% lebih sedikit daripada urutan {1, 0}. Hal ini menandakan bahwa perbedaan pada angka kunci yang lebih kecil memiliki pengaruh yang lebih signifikan pada urutan pembangkitan kandidat daripada angka kunci yang lebih besar.

Hal ini terjadi karena pada urutan {1, 0}, algoritma akan lebih dahulu mencoba menandai sebuah sel dengan nilai 1. Jika akibatnya jumlah sementara pada suatu baris atau kolom langsung melebihi angka kunci, maka simpul tersebut akan langsung dinyatakan tidak valid oleh fungsi pembatas. Sebagai akibatnya, algoritma harus melakukan *backtracking* dan mencoba kandidat lainnya. Karena banyak kombinasi yang diawali dengan nilai 1 berujung pada pelanggaran fungsi pembatas, maka lebih banyak simpul yang harus dibangkitkan dan dievaluasi sebelum solusi ditemukan.

Pada urutan pembangkitan {0, 1}, algoritma akan cenderung menelusuri simpul hingga ke tingkat terdalam, yaitu sebanyak n simpul (dengan n adalah jumlah sel dalam papan Kakurasu), dalam kondisi seluruh nilai sel bernilai 0. Hal ini terjadi karena konfigurasi tersebut masih dianggap valid oleh fungsi pembatas, mengingat jumlah sementara pada setiap baris dan kolom belum melebihi angka kunci. Dengan demikian, simpul-simpul awal tetap dieksplorasi hingga terbentuk konfigurasi penuh, dan evaluasi ketidaksesuaian baru terjadi di simpul terdalam jika jumlah total ternyata tidak memenuhi angka kunci. Proses ini menghasilkan jalur eksplorasi yang lebih dalam namun dengan jumlah simpul yang dibangkitkan lebih sedikit secara keseluruhan dibandingkan urutan {1, 0}, karena lebih jarang terjadi pelanggaran batas di tahap awal pencarian.

Berdasarkan penelitian ini, diperoleh bahwa jumlah simpul yang pada pohon ruang status adalah 2^N dengan N adalah banyak sel papan kakurasu, sehingga kompleksitas waktu algoritma dalam kasus terburuknya adalah $O(n \cdot 2^N)$ [5], dengan n adalah banyaknya jumlah baris/kolom.

V. KESIMPULAN

Penyelesaian permainan kakurasu dapat menggunakan algoritma *backtracking* yang memiliki basis pencarian DFS dan merupakan pengembangan dari algoritma *exhautive search*. Berdasarkan hasil yang sudah diperoleh, didapatkan bahwa algoritma *backtracking* dapat menghasilkan solusi pada jumlah sel yang terbatas (5 × 5) karena semakin besar ukuran papan, maka waktu dan jumlah simpul yang dikunjungi semakin besar secara eksponensial ($O(n \cdot 2^N)$). Selain itu, terdapat faktor penting lain yang dapat mempengaruhi proses pencarian solusi kakurasu dengan algoritma *backtracking*, yaitu urutan pembangkitan kandidat dan besar-kecilnya angka kunci pada sebelah kanan/bawah papan. Semakin besar angka kunci, maka semakin banyak sel yang harus dipilih, sehingga

jumlah kombinasi yang perlu dieksplorasi meningkat, yang berdampak pada bertambahnya waktu pencarian. Sementara itu, urutan pembangkitan kandidat {0, 1} memiliki waktu lebih singkat daripada urutan {1, 0} karena lebih sedikit simpul tidak valid yang dibangkitkan di awal pencarian.

UCAPAN TERIMA KASIH

Alhamdulillah, segala puji dan syukur penulis panjatkan kepada Allah Subhanahu wa ta'ala atas rahmat dan ridha-Nya, sehingga penulis dapat menyelesaikan tugas makalah yang berjudul "Implementasi dan Analisis Algoritma Runut-Balik (*Backtracking*) dalam Penyelesaian Permainan Kakurasu" dengan baik. Terselesaikannya tugas makalah ini tidak lepas dari dukungan dan bantuan dari banyak pihak. Oleh karena itu, penulis ingin menyampaikan terima kasih, terutama kepada dosen-dosen pengampu mata kuliah IF2211 – Strategi Algoritma, yaitu Bapak Dr. Ir. Rinaldi Munir, M.T., Ibu Nur Ulfa Maulidevi S.T., M.T., dan Bapak Monterico Adrian, S.T., M.T.., yang telah memberikan ilmu bermanfaat sehingga penulis dapat menyelesaikan tugas makalah ini. Penulis berharap, makalah ini tidak hanya menjadi bentuk implementasi dari ilmu yang telah dipelajari, tetapi juga dapat memberikan manfaat bagi pembaca dan menjadi referensi bagi mahasiswa lainnya yang mempelajari topik serupa.

REFERENSI

[1] H. Kumar, "Backtracking Algorithm." Accessed: Jun. 24, 2025. [Online]. Available: https://www.geeksforgeeks.org/dsa/backtracking-algorithms/

[2] Izharulhaq, "Penerapan Algoritme Depth First Search dalam Menyelesaikan Kakurasu Puzzle," 2020.

[3] J. Rahmadian and E. Istighfarin, "Perancangan Permainan Teka-Teki Logika Labirin sarang Lebah," Jakarta Selatan, Jan. 2016. [Online]. Available: http://stti.i-tech.ac.id/

[4] "Kakurasu." Accessed: Jun. 24, 2025. [Online]. Available: https://id.puzzle-kakurasu.com/

[5] Kartik, "Introduction To Backtracking." Accessed: Jun. 24, 2025. [Online]. Available: https://www.geeksforgeeks.org/introduction-to-backtracking-2/

[6] M. , H. Rifqo and Y. Apridiansyah, "Implementasi Algoritma Backtracking dalam Sistem Informasi Perpusatakaan Untuk Pencarian Judul Buku (Studi Kasus Unit Pelayanan Terpadu Perpustakaan Universitas Muhammadiyah Bengkulu)," *Jurnal Pseudocode*, vol. 4, pp. 90–96, Feb. 2017.

[7] R. Munir, "Algoritma Runut-Balik (Backtracking) (Bagian 1)." Accessed: Jun. 24, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian1.pdf

[8] S. Pemberton, "The Computer as Extended Phenotype (Computers, Genes, and You)." Accessed: Jun. 24, 2025. [Online]. Available: https://www.w3.org/2011/Talks/01-14-steven-phenotype/

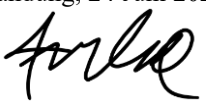
[9] T. S. Weng, "Enhancing Problem-Solving Ability through a Puzzle-Type Logical Thinking Game," *Scientific Programming*, vol. 2022, Mar. 2022, doi: 10.1155/2022/7481798.

[10] Y. A. Putra, "Penyelesaian Puzzle Kakurasu dengan Algoritma Brute Force," 2014.

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



Fardhan Indrayesa
12821046