Implementasi Algoritma *Backtracking* dengan Optimasi Pemangkasan Ruang Status pada Permainan Nonogram

Geraldo Artemius - 13524005
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
Email : [1]*13524005@std.stei.itb.ac.id*, [2]*geraldoartemius2005@gmail.com*

***Abstract*—Permainan Nonogram merupakan teka-teki logika visual termasuk NP-Complete, di mana solusi dicari dengan mewarnai sel pada *grid* berdasarkan petunjuk angka. Pencarian solusi secara naif sangat tidak efisien karena kompleksitas waktu yang eksponensial seiring bertambahnya ukuran papan. Makalah ini menyajikan implementasi algoritma Runut-Balik (*Backtracking*) untuk menyelesaikan permainan Nonogram secara otomatis. Untuk mengatasi banyaknya kombinasi pada ruang pencarian, diterapkan optimasi melalui pemangkasan ruang status (*state space pruning*). Dalam implementasi ini, ruang status diekspansi secara bertahap, dan sebuah fungsi pembatas (*bounding function*) sebagai pengevaluasi validitas konfigurasi baris dan kolom setiap kali sel diwarnai. Jika penempatan warna melanggar *bounding function* seperti melebihi batas kapasitas blok atau ketidaksesuaian urutan maka simpul tersebut langsung dimatikan dan algoritma melakukan *backtracking*. Algoritma ini diupayakan mencegah program untuk menelusuri cabang yang tidak valid. Penggunaan *bounding function* secara signifikan mengurangi jumlah simpul di dalam pohon pencarian. Hasilnya menunjukkan bahwa algoritma Backtracking dengan optimasi pemangkasan merupakan pendekatan komputasional yang sangat efektif untuk menyelesaikan permainan Nonogram.**

Kata Kunci—Nonogram, *Backtracking*, Pemangkasan Ruang Status, *Bounding Function*, *Depth-First Search*

I. PENDAHULUAN

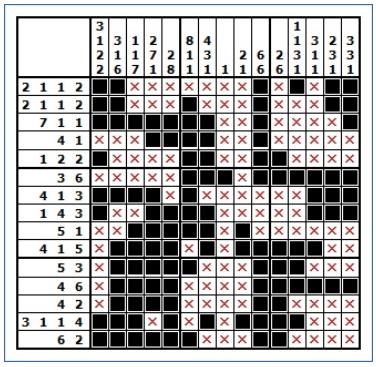

Fig. 1. Contoh Papan Permainan Nonogram. sumber

Permainan Nonogram, yang juga dikenal sebagai *Griddlers* atau *picture cross puzzles*, merupakan permainan yang melatih kemampuan kognitif seperti memori, perhatian, dan penalaran. Tujuan utama dari permainan ini adalah mengisi papan permainan dengan warna untuk menunjukkan sebuah gambar tersembunyi. Pengisian grid ini tidak dilakukan dengan menebak secara acak, melainkan menggunakan logika deduktif berdasarkan deret angka petunjuk yang terdapat pada sisi setiap baris dan kolom.

Meskipun aturan dasarnya cukup sederhana, penyelesaian Nonogram menghadirkan tantangan komputasi yang sangat tinggi. Bagi manusia, penyelesaian *puzzle* ini mengandalkan pengenalan pola visual. Namun, dari sudut pandang ilmu komputer, mencari solusi pada *grid* Nonogram adalah sebuah masalah optimasi kombinatorial yang kompleks dan diklasifikasikan sebagai masalah NP-Complete. Seiring bertambahnya dimensi papan permainan, jumlah kemungkinan kombinasi warna pada kotak akan meningkat secara eksponensial, sehingga pencarian solusi secara manual atau *brute force* sangat tidak efisien.

Salah satu metode yang paling sesuai untuk masalah pencarian kombinatorial adalah algoritma Runut-Balik (*Backtracking*). Algoritma ini bekerja dengan membangun kandidat solusi secara bertahap dan segera melakukan proses *backtracking* apabila sebuah kandidat/simpul terbukti melanggar aturan. Agar pencarian solusi dapat berjalan optimal dan tidak terjebak dalam ledakan kombinasi, algoritma ini memerlukan optimasi berupa pemangkasan ruang status (*state space pruning*). Dengan menerapkan fungsi pembatas (*bounding function*), algoritma dapat mendeteksi ketidaksesuaian tata letak blok warna lebih awal dan membuang cabang kemungkinan yang salah tanpa harus menelusurinya hingga akhir.

Makalah ini bertujuan untuk mengimplementasikan algoritma *Backtracking* dengan mengaplikasikan teknik pemangkasan ruang status untuk menyelesaikan permainan Nonogram. Tujuan dari makalah ini adalah untuk menganalisis dan mendemonstrasikan bagaimana pembatasan ekspansi simpul pada pohon pencarian dapat mengefisienkan waktu penyelesaian masalah teka-teki gambar ini. Objektif utama dari penulisan ini terletak pada perancangan logika komputasional *bounding function* yang secara selalu memvalidasi aturan baris dan kolom, sehingga menghasilkan program pencari solusi yang sistematis dan optimal.

II. LANDASAN TEORI

A. Permainan Nonogram

Pengisian *Grid*

Dalam proses pencarian solusi, pemain dapat mengeksekusi dua jenis aksi pengisian terhadap kotak-kotak di dalam papan:

- **Mode Isi (■):** Aksi untuk mengisi kotak dengan warna, yang menandakan bahwa kotak tersebut adalah bagian valid dari gambar.
- **Mode Tandai (×):** Aksi untuk memberikan tanda silang pada kotak sebagai penanda bahwa kotak tersebut dipastikan bukan bagian dari gambar.

Aturan Angka dan Jarak (*Constraints*)

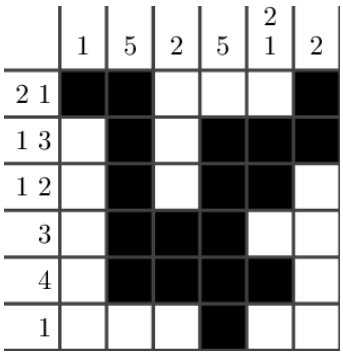

Fig. 2. Implementasi Jarak Pada Nonogram. sumber

Setiap baris (horizontal) dan kolom (vertikal) pada papan, baik yang berukuran 5×5, 10×10, maupun ukuran lainnya dalam format $N \times N$, dilengkapi dengan angka petunjuk di sebelah kiri atau atas. Angka-angka ini merepresentasikan aturan yang harus dipenuhi:

- **Deret Berurutan:** Angka petunjuk menunjukkan berapa banyak kotak berurutan yang harus diisi. Sebagai contoh, petunjuk ”3” berarti pemain harus mengisi 3 kotak secara berurutan.
- **Aturan Jarak:** Ketika terdapat beberapa angka petunjuk untuk satu baris atau kolom (misalnya ”1 2”), blok kotak yang diisi harus dipisahkan oleh setidaknya satu kotak kosong. Dalam contoh ”1 2”, pemain harus mengisi 1 kotak, memberikan jarak minimal satu kotak kosong, lalu mengisi 2 kotak berurutan.

Strategi Deduksi

Penyelesaian Nonogram tidak mengandalkan tebakan acak, melainkan menggunakan deduksi berdasarkan panjang kisi-kisi dan petunjuk angka. Sebagai contoh pada papan dengan kapasitas lebar 5 kotak:

- **Kapasitas Penuh:** Jika petunjuknya adalah ”5”, maka seluruh 5 kotak dipastikan terisi. Jika petunjuknya adalah ”2 2”, berdasarkan aturan jarak wajib, susunannya dipastikan menjadi 2 kotak isi, 1 kotak kosong, dan 2 kotak isi.
- **Irisan Pasti (*Overlapping*):** Jika petunjuknya adalah ”4”, terdapat dua kemungkinan posisi (dimulai dari kiri atau kanan). Namun, tiga kotak di bagian tengah akan selalu terisi pada kedua kemungkinan tersebut. Maka kotak-kotak ini dapat langsung diisi sebagai langkah awal deduksi.

Penandaan kotak yang dipastikan kosong menggunakan × akan membantu mempersempit ruang kemungkinan, sehingga proses deduksi atau penyimpulan langkah selanjutnya menjadi lebih efisien.

Kondisi Penyelesaian (*Goal State*)

Fig. 3. Contoh Mencapai Goal State. sumber

Kondisi penyelesaian yang valid (*goal state*) tercapai apabila seluruh kotak telah dievaluasi dan diisi sehingga memenuhi semua syarat petunjuk angka di seluruh baris dan kolom. Pada saat kondisi ini terpenuhi secara serentak, konfigurasi kotak-kotak tersebut akan menampilkan sebuah karya seni piksel utuh.

B. Teori Kompleksitas dan NP-Complete

Kompleksitas Waktu Algoritma

Berdasarkan kebutuhan waktu eksekusinya, algoritma untuk menyelesaikan sebuah persoalan dibagi menjadi dua kelompok besar: algoritma waktu polinom (*polynomial-time algorithms*) dan algoritma waktu non-polinom (*nonpolynomial-time algorithms*). Algoritma waktu-polinom adalah algoritma yang kompleksitas waktunya dibatasi oleh fungsi polinom. Sebuah persoalan dikatakan *tractable* jika ia dapat diselesaikan dalam waktu yang wajar. Sebaliknya, persoalan dikatakan *intractable* jika tidak dapat diselesaikan dalam waktu yang wajar seiring dengan bertambahnya ukuran masukan. Algoritma waktu non-polinom dibatasi oleh fungsi non-polinom dan sering kali dianggap sebagai persoalan yang sulit.

Kelas P dan NP

Dalam pembahasan teori kompleksitas, batasan utamanya sering kali difokuskan pada persoalan keputusan (*decision problem*), yaitu persoalan yang solusinya hanya berupa jawaban ”ya” atau ”tidak”.

- **Kelas P (*Polynomial*):** Merupakan semua persoalan keputusan yang dapat dipecahkan oleh algoritma deterministik dalam waktu polinom. Semua persoalan keputusan yang dapat diselesaikan dalam waktu polinom adalah anggota himpunan P.
- **Kelas NP (*Non-deterministic Polynomial*):** Merupakan persoalan keputusan yang dapat diselesaikan oleh algoritma non-deterministik dalam waktu polinomial (*non-deterministic polynomial-time algorithm*). Ini berarti, jika diberikan sebuah kandidat solusi, kebenaran solusi tersebut dapat diverifikasi dalam waktu polinom. Setiap persoalan keputusan yang kompleksitas waktunya polinomial

solusinya juga dapat diverifikasi dalam waktu polinom, sehingga kelas P adalah himpunan bagian dari NP ($P \subseteq NP$).

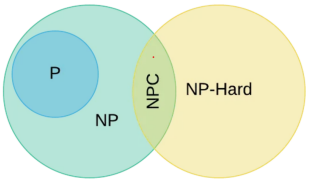

Fig. 4. Hubungan P, NP, NP-Complete, NP-Hard

Kelas NP-Complete (NPC)

NP-Complete (NPC) adalah himpunan persoalan NP yang paling sulit namun sangat menarik. Sebuah persoalan X didefinisikan sebagai NP-Complete jika memenuhi dua syarat berikut:

1) Persoalan X termasuk ke dalam kelas NP.
2) Setiap persoalan di dalam kelas NP dapat direduksi (ditransformasikan) menjadi X dalam waktu polinom.

Apabila sebuah algoritma waktu-polinom berhasil ditemukan untuk menyelesaikan satu persoalan NPC, maka seluruh persoalan di dalam kelas NP dapat dipecahkan dalam waktu polinom.

Nonogram sebagai Masalah NP-Complete

Permainan Nonogram diklasifikasikan ke dalam kategori NP-Complete. Hal ini dapat dibuktikan melalui dua parameter berikut:

- **Pembuktian Kelas NP:** Jika sebuah tebakan solusi Nonogram diberikan (sebuah matriks yang telah diwarnai secara penuh), kebenaran solusi tersebut dapat diverifikasi dalam waktu polinom. Program hanya perlu melakukan iterasi satu kali pada setiap baris dan kolom untuk mencocokkannya dengan petunjuk angka.
- **Kompleksitas Pencarian:** Pada matriks berukuran $N \times M$, setiap kotak memiliki dua kemungkinan status (diisi atau disilang). Mencari solusi yang tepat dengan mencoba semua kombinasi secara *brute force* membutuhkan waktu eksponensial.

C. Algoritma Runut-Balik (Backtracking)

Konsep Dasar Algoritma Runut-Balik

Algoritma runut-balik merupakan perbaikan dari algoritma pencarian solusi secara *exhaustive search*. Pada *exhaustive search*, semua kemungkinan solusi dieksplorasi dan dievaluasi satu per satu. Sebaliknya, pada algoritma *backtracking*, hanya pilihan yang mengarah ke solusi yang dieksplorasi. Pilihan yang tidak mengarah ke solusi tidak akan dipertimbangkan lagi melalui proses pemangkasan (*pruning*).

Properti Umum Algoritma

Dalam merumuskan penyelesaian dengan *backtracking*, terdapat tiga properti umum yang harus didefinisikan:

1) **Vektor Solusi:** Solusi dari sebuah persoalan dinyatakan sebagai vektor dengan n-tuple:

$$X = (x_1, x_2, ..., x_n)$$

2) **Fungsi Pembangkit:** Dinyatakan sebagai fungsi

$$T(x_1, x_2, ..., x_{k-1})$$

bertugas untuk membangkitkan nilai bagi x_k, yaitu komponen vektor solusi pada langkah ke-k.

3) **Fungsi Pembatas (*Bounding Function*):** Dinyatakan sebagai predikat

$$B(x_1, x_2, ..., x_k)$$

yang mengembalikan nilai *true* atau *false*. Fungsi ini bernilai *true* apabila nilai $(x_1, x_2, ..., x_k)$ mengarah ke solusi dan tidak melanggar *constraints* persoalan. Jika mengembalikan nilai *true*, proses pembangkitan diteruskan ke langkah $k + 1$, namun jika *false*, maka kandidat kombinasi tersebut dibuang/dimatikan.

Pohon Ruang Status (*State Space Tree*)

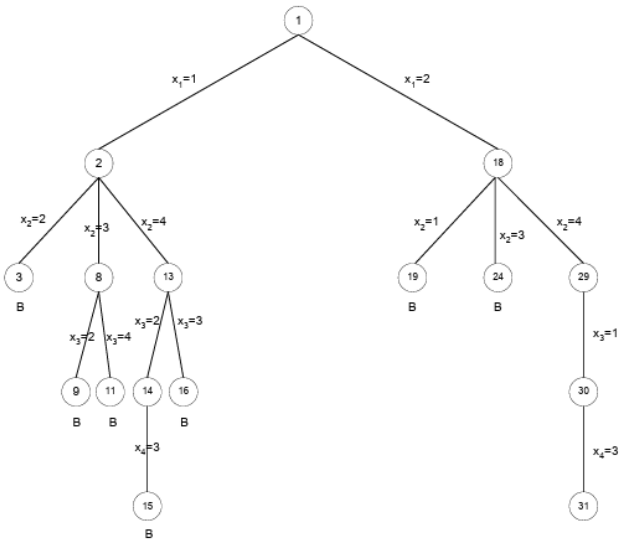

Fig. 5. Contoh *State Space Tree* Persoalan N-Queens 4×4

Semua kemungkinan solusi dari persoalan (ruang solusi) diorganisasikan ke dalam sebuah struktur pohon berakar. Algoritma *backtracking* pada dasarnya dapat dipandang sebagai proses pencarian di dalam pohon tersebut, bergerak dari akar menuju daun.

Pohon ruang status memiliki tiga macam simpul, yaitu simpul akar, simpul dalam, dan simpul daun. Setiap simpul di dalam pohon ini merepresentasikan status (*state*) persoalan pada tahap tertentu, sementara sisi atau cabangnya dilabeli dengan nilai-nilai penugasan variabel solusi. Sebuah solusi yang mungkin direpresentasikan oleh sebuah lintasan dari simpul akar hingga ke simpul daun.

Prinsip Pencarian dan Pemangkasan Ruang Status

Pencarian solusi secara sistematis pada pohon ruang status dilakukan melalui tahapan berikut:

1) **Pembangkitan Simpul:** Pencarian dimulai dengan membangkitkan simpul-simpul status sehingga membentuk lintasan dari akar ke daun mengikuti mekanisme

pencarian *Depth-First Search* (DFS). Simpul yang telah dibangkitkan dinamakan simpul hidup (*live node*), dan simpul hidup yang saat ini sedang diperluas dinamakan simpul-E (*Expand-node*).

2) **Pemangkasan (*Pruning*):** Tiap kali simpul-E diperluas, lintasan solusi yang dibangun akan bertambah panjang. Jika evaluasi menunjukkan bahwa lintasan tersebut tidak mengarah ke solusi, simpul-E akan "dimatikan" menggunakan *bounding function*. Statusnya berubah menjadi simpul mati (*dead node*), dan algoritma secara implisit memangkas seluruh cabang simpul anaknya.
3) **Runut-Balik (*Backtrack*):** Apabila pembentukan lintasan berujung pada simpul mati, algoritma akan melakukan proses runut-balik ke simpul hidup pada aras (*level*) tepat di atasnya.
4) **Eksplorasi Lanjutan:** Dari simpul hidup tersebut, algoritma meneruskan eksplorasi dengan membangkitkan simpul anak cabang lainnya untuk dijadikan simpul-E yang baru. Proses runut-balik dan ekspansi ini berulang hingga algoritma mencapai simpul solusi atau seluruh ruang pencarian habis dievaluasi.

III. Perancangan dan Implementasi Algoritma

A. Pemodelan dan Representasi Data Nonogram

Representasi Papan Permainan (Ruang Pencarian)

Papan permainan Nonogram dimodelkan sebagai matriks (array dua dimensi) berukuran $N \times M$, di mana N adalah jumlah baris dan M adalah jumlah kolom. Misalkan matriks ini direpresentasikan dengan variabel `board[N][M]`. Pada awal permainan (status awal atau *initial state*), seluruh elemen matriks diinisialisasi dengan nilai yang menandakan bahwa kotak tersebut belum dievaluasi.

Nilai Variabel (Sel)

Setiap sel atau kotak pada koordinat (i, j) di dalam matriks bertindak sebagai variabel keputusan. Dalam pendekatan runut-balik ini, domain nilai untuk setiap sel `board[i][j]` dibatasi pada tiga *state*:

- `0` (*UNASSIGNED*): Menandakan bahwa sel masih kosong dan belum diberikan tebakan warna.
- `1` (*FILLED*): Menandakan bahwa sel ditebak berisi blok warna.
- `2` (*CROSSED*): Menandakan bahwa sel ditebak pasti kosong (X).

Representasi (*Clues*)

Petunjuk angka yang berada di luar kisi-kisi permainan bertindak sebagai himpunan (*constraints*). Petunjuk ini direpresentasikan menggunakan struktur data *array of lists* (atau *array of vectors* dalam implementasinya).

- `rowClue[i]` menyimpan deretan angka petunjuk untuk baris ke-i.
- `colClue[j]` menyimpan deretan angka petunjuk untuk kolom ke-j.

Sebagai contoh, jika baris pertama memiliki petunjuk angka "1 2", maka `RowClues[0]` akan berisi himpunan nilai {`1, 2`}.

Definisi Transisi Status

Sebuah status (*state*) didefinisikan sebagai satu konfigurasi spesifik dari matriks `board` pada waktu tertentu. Transisi dari satu status ke status lainnya di dalam pohon ruang status terjadi ketika algoritma memilih satu sel `board[i][j]` yang masih bernilai `0`, dan mencoba mengubah nilainya menjadi `1` (cabang kiri pohon) atau `2` (cabang kanan pohon). Transisi ini akan terus dilakukan hingga matriks terisi penuh dan memenuhi seluruh elemen pada `rowClue` dan `colClue`, yang menandakan tercapainya status solusi (*goal state*).

Deklarasi Tipe Data dan Ruang Status

```
type CellState : enum {UNASSIGNED=0, FILLED=1, CROSSED=2}

class Nonogram
  Deklarasi Atribut (private):
    N, M : integer // N baris dan M kolom
    board : matriks N × M of CellState
    rowClue : array of list of integer
    colClue : array of list of integer
```

B. Fungsi Pembangkit

Pada penyelesaian Nonogram, fungsi ini diimplementasikan melalui sebuah prosedur rekursif yang mengevaluasi setiap sel `board` secara sekuensial dari kiri atas hingga kanan bawah.

Mekanisme Pemilihan Variabel

Pencarian solusi dilakukan sel demi sel. Misalkan algoritma saat ini berada pada sel dengan koordinat baris i dan kolom j. Algoritma pertama-tama akan memeriksa apakah indeks baris i telah mencapai nilai N (jumlah total baris). Jika $i = N$, maka seluruh sel pada matriks telah berhasil diisi, yang menandakan bahwa status solusi (*goal state*) telah tercapai (kondisi basis/ *base case*). Jika belum, program akan menghitung koordinat sel berikutnya, yaitu bergeser ke kanan $(j + 1)$, atau turun ke baris baru $(i+1)$ apabila indeks j telah mencapai ujung kolom.

Pembangkitan Cabang dan *Backtracking*

Untuk setiap sel (i, j) yang diekspansi, fungsi pembangkit akan menghasilkan dua kemungkinan status baru secara berurutan:

1) **Eksplorasi Cabang Kiri (Blok Warna):** Fungsi akan mencoba *assign* status `FILLED` pada `board[i][j]`. Setelah ini, (*bounding function*) dipanggil untuk memvalidasi apakah penempatan blok warna tersebut tidak melanggar petunjuk angka (*clues*) baris dan kolom. Jika valid, fungsi memanggil dirinya sendiri secara rekursif untuk mengekspansi sel selanjutnya.
2) **Eksplorasi Cabang Kanan (Sel Kosong):** Apabila cabang pertama mengembalikan nilai salah, baik karena ditolak langsung oleh fungsi pembatas maupun karena menemui jalan buntu di kedalaman rekursi, algoritma akan melakukan *backtracking*. Nilai `board[i][j]` diubah menjadi `CROSSED`, dan proses validasi serta pemanggilan rekursif diulang untuk status baru ini.

Jika kedua penugasan (`FILLED` dan `CROSSED`) sama-sama menghasilkan status tidak valid atau jalan buntu, maka sel (i, j) dikembalikan ke status awalnya (`UNASSIGNED`), dan

fungsi akan mengembalikan nilai salah (*false*) kepada pemanggil di aras atasnya untuk memicu *backtracking* lebih lanjut.

C. Perancangan Bounding Function

Fungsi ini akan mengevaluasi validitas konfigurasi matriks `board` sesaat setelah sebuah sel (i, j) diberikan penugasan status baru, serta mendeteksi jalan buntu (*dead end*) secepat mungkin.

Fungsi pembatas, diimplementasikan dengan `boundingFunction(i, j)`, fungsi akan membandingkan status sel-sel pada baris ke-i dan kolom ke-j terhadap himpunan *constraints* `rowClue[i]` dan `colClue[j]`. Sebuah simpul dinyatakan mati dan ruang statusnya dipangkas apabila memenuhi salah satu dari pelanggaran berikut:

1) **Pelanggaran Penjumlahan Petunjuk:**
   Fungsi menghitung total akumulasi sel bernilai `FILLED` pada baris i dan kolom j saat ini. Jika jumlah sel `FILLED` melebihi total penjumlahan angka pada `rowClue[i]` atau `colClue[j]`, maka status tersebut dipastikan tidak valid.
2) **Pelanggaran Ukuran Blok:**
   Algoritma menelusuri deretan sel `FILLED` yang berurutan. Jika terdapat blok warna yang panjangnya melebihi angka petunjuk (*clue*) yang sedang dievaluasi pada baris atau kolom tersebut, maka simpul langsung dimatikan.
3) **Pelanggaran Aturan Jarak:**
   Dua angka petunjuk yang berbeda mengisyaratkan adanya dua blok warna yang terpisah. Jika fungsi mendeteksi dua blok `FILLED` menyatu tanpa adanya minimal satu sel `CROSSED` atau `UNASSIGNED` di antaranya, maka konfigurasi tersebut melanggar aturan jarak dan algoritma akan memicu runut-balik.

Apabila ketiga uji kendala di atas berhasil dilewati, maka fungsi pembatas mengembalikan nilai benar (*true*), yang mengisyaratkan bahwa lintasan dari akar ke simpul saat ini masih berpotensi mengarah ke simpul solusi. Sebaliknya, jika salah satu evaluasi gagal, fungsi mengembalikan nilai salah (*false*), memberitahu fungsi pembangkit bahwa status saat ini adalah jalan buntu dan harus segera dianulir.

Pseudocode Fungsi Pembatas

```
function BoundingFunction(row, col : integer) → boolean
  // 1. Evaluasi Kapasitas Maksimal (Overfill)
  maxAllowedRow ← jumlah angka petunjuk rowClue[row]
  curFilledRow ← total FILLED pada board[row][0 ... col]
  if curFilledRow > maxAllowedRow then
    return false
  end if

  maxAllowedCol ← jumlah angka petunjuk colClue[col]
  curFilledCol ← total FILLED pada board[0 ... row][col]
  if curFilledCol > maxAllowedCol then
    return false
  end if

  // 2. Evaluasi Ukuran Blok dan Jarak pada Baris
  idxRow ← 0, curBlockRow ← 0
  for i ← 0 to col do
    if board[row][i] == FILLED then
      curBlockRow ← curBlockRow + 1
    else if board[row][i] == CROSSED and
curBlockRow > 0 then
      if idxRow ≥ ukuran(rowClue[row]) or
      curBlockRow ≠ rowClue[row][idxRow] then
        return false
      end if
      idxRow ← idxRow + 1, curBlockRow ← 0
    end if
  end for

  // Evaluasi ujung blok terakhir
  if curBlockRow > 0 then
    if idxRow ≥ ukuran(rowClue[row]) or
    curBlockRow > rowClue[row][idxRow] then
      return false
    end if
  end if

  // 3. Evaluasi Kolom (Logika serupa dengan baris)
  if terdapat blok kolom tidak valid or melebihi batas then
    return false
  end if

  return true
end function
```

D. Preprocessing dan Deduksi Logis

Meskipun algoritma *backtracking* dengan fungsi pembatas mampu memangkas ruang pencarian secara signifikan, proses eksplorasi sel demi sel dari awal masih membutuhkan waktu komputasi yang besar, terutama pada matriks berukuran luas. Untuk menanggulangi hal tersebut, diimplementasikan sebuah tahap prapemrosesan (*pre-processing*) yang dieksekusi satu kali sebelum algoritma pencarian utama berjalan. Tahap ini bertujuan untuk menggabungkan heuristik deduksi logis manusia dengan kecepatan komputasi mesin.

1) Irisan Kepastian

Salah satu teknik deduksi yang paling efektif dalam Nonogram adalah strategi irisan kepastian. Teknik ini mengidentifikasi sel-sel yang dipastikan bernilai `FILLED` sesuai dengan ukuran *board*.

Apabila sebuah baris atau kolom dengan kapasitas panjang L memiliki sebuah petunjuk angka tunggal C, dan nilai C tersebut memenuhi syarat $C \frac{L}{2}$, maka akan selalu terdapat irisan blok warna yang tumpang tindih. Secara matematis, sel-sel yang dipastikan terisi berada pada rentang indeks j (dengan basis indeks 0) sebagai berikut:

$$(L - C) \leq j \leq (C - 1)$$

Sebagai contoh, pada matriks berukuran 10×10 ($L = 10$) dengan petunjuk baris $C = 9$, sel yang pasti terisi adalah dari indeks $j = (10 - 9)$ hingga $j = (9 - 1)$, yaitu indeks ke-1 sampai ke-8. Pada matriks 15×15 ($L = 15$) dengan petunjuk

$C = 14$, sel yang pasti terisi berada pada rentang indeks 1 hingga 13.

2) Strategi Baris Penuh

Selain irisan kepastian, algoritma prapemrosesan juga mendeteksi kondisi di mana sebuah baris atau kolom memiliki konfigurasi yang pasti tanpa perlu ditebak. Kondisi ini terjadi apabila total panjang seluruh blok warna yang diisyaratkan oleh petunjuk, ditambah dengan jumlah spasi minimum yang memisahkan blok-blok tersebut, sama persis dengan kapasitas panjang baris atau kolom (L).

Secara matematis, misalkan sebuah baris memiliki himpunan petunjuk $C = \{c_1, c_2, \ldots, c_k\}$ dengan jumlah elemen k. Kondisi baris penuh terpenuhi jika:

$$\left(\sum_{i=1}^{k} c_i\right) + (k-1) = L$$

Sebagai contoh, pada matriks berukuran 10×10 ($L = 10$), jika terdapat petunjuk baris bernilai $\{5, 4\}$, maka total ruang yang dibutuhkan adalah $5 + 4 + (2 - 1) = 10$. Karena kebutuhan ruang sama dengan kapasitas matriks, konfigurasi ini bersifat pasti. Fungsi prapemrosesan akan langsung mengisi sel-sel tersebut secara berurutan dengan status `FILLED` dan menyisipkan `CROSSED` sebagai pemisah.

```
procedure PreProcess()
  // 1. Evaluasi Setiap Baris
  for i ← 0 to N − 1 do
    L ← M // Kapasitas baris
    // Strategi Irisan Kepastian (Overlapping)
    if ukuran(rowClue[i]) == 1 then
      C ← rowClue[i][0]
      if C > L/2 then
        for j ← (L − C) to (C − 1) do
          board[i][j] ← FILLED
        end for
      end if
    end if
    // Strategi Baris Penuh (Perfect Match)
    sumClue ← jumlah total elemen pada rowClue[i]
    minSpace ← ukuran(rowClue[i]) - 1
    if (sumClue + minSpace) == L then
      j ← 0
      for each c in rowClue[i] do
        // Isi blok warna
        for k ← 1 to c do
          board[i][j] ← FILLED
          j ← j + 1
        end for
        // Sisipkan spasi pembatas
        if j < L then
          board[i][j] ← CROSSED
          j ← j + 1
        end if
      end for
    end if
  end for
  // 2. Evaluasi Setiap Kolom
  // (Menerapkan strategi Irisan Kepastian dan Baris Penuh
  // secara identik pada setiap indeks kolom  j ← 0 . . . M − 1)
end procedure
```

E. Implementasi Backtracking

Algoritma berada di dalam sebuah fungsi rekursif solve yang bertugas menelusuri pohon ruang status sel demi sel secara sekuensial.

Fungsi ini menyatukan seluruh komponen yang telah dirancang sebelumnya. Alur kerjanya mencakup:

1) **Pengecekan Kondisi Basis:** Jika indeks baris telah mencapai akhir matriks (N), pencarian dihentikan dan program mengembalikan nilai benar (*true*) yang menandakan solusi telah ditemukan.
2) **Integrasi Prapemrosesan:** Fungsi mengecek apakah sel saat ini sudah diisi oleh tahap prapemrosesan. Jika sel tidak bernilai `UNASSIGNED`, fungsi tidak akan mencoba menebak, melainkan langsung melakukan rekursi ke sel berikutnya.
3) **Pembangkitan dan Pemangkasan (*Pruning*):** Fungsi membangkitkan cabang pertama dengan menugaskan status `FILLED`. Jika lolos dari *bounding function*, pencarian dilanjutkan. Jika gagal, fungsi membangkitkan cabang kedua (`CROSSED`).
4) **Runut-Balik (*Backtrack*):** Jika kedua cabang menghasilkan jalan buntu (*false*), fungsi menganulir tebakan dengan mengembalikan status sel menjadi `UNASSIGNED`, lalu memicu runut-balik ke kedalaman rekursi sebelumnya.

Pseudocode Solve

```
function Solve(row, col : integer) → boolean
  // 1. Kondisi Basis: Berhasil mencapai batas akhir baris
  if row == N then return true

  // Menghitung koordinat sel berikutnya
  nextRow, nextCol ← hitung_koordinat_sel_berikutnya(row, col)

  // 2. Integrasi Prapemrosesan: Lompat jika sudah terisi
  if board[row][col] ≠ UNASSIGNED then
    return Solve(nextRow, nextCol)
  end if

  // 3. Pembangkitan Cabang Pertama (Blok Warna)
  board[row][col] ← FILLED
  if BoundingFunction(row, col) == true then
    if Solve(nextRow, nextCol) == true then
      return true
    end if
  end if
  // 4. Pembangkitan Cabang Kedua (Sel Kosong/Silang)
  board[row][col] ← CROSSED
  if BoundingFunction(row, col) == true then
    if Solve(nextRow, nextCol) == true then
      return true
    end if
  end if
  // 5. Jalan Buntu (Runut-balik / Backtrack)
  board[row][col] ← UNASSIGNED
  return false
end function
```

IV. HASIL DAN PEMBAHASAN

A. *Test Case 1*

10×10

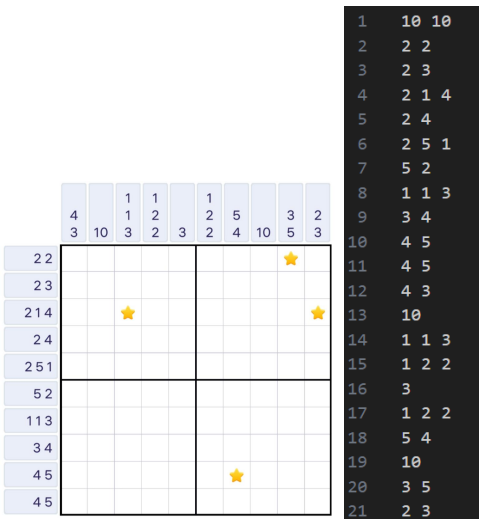

Fig. 6. Test Case 1 dari Game Nonogram.

Hasil Program

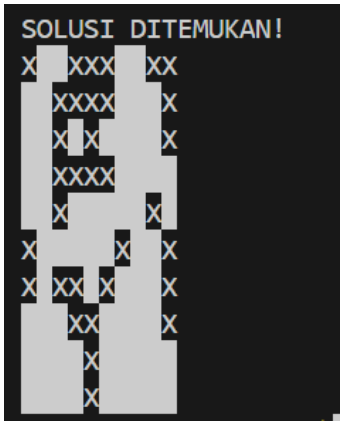

Fig. 7. Hasil Input *Test Case* 1 pada Program (Mencapai Goal State)

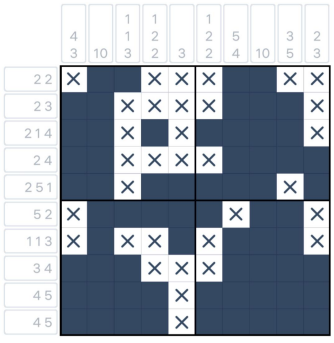

Fig. 8. Solusi *Test Case* 1 di Game Nonogram

Program membaca masukan berekstensi `.txt` yang berisi ukuran papan dan (*clue*) permainan. Dua nilai pertama pada berkas masukan menyatakan ukuran papan, yaitu 10×10. Selanjutnya, program membaca petunjuk untuk setiap baris, dimulai dari baris pertama dengan *clue* "2 2", baris kedua "2 3", hingga baris kesepuluh "4 5". Setelah seluruh petunjuk baris dibaca, program melanjutkan pembacaan petunjuk untuk setiap kolom, dimulai dari kolom pertama dengan *clue* "4 3" hingga kolom kesepuluh "2 3".

Fig. 7 menunjukkan hasil eksekusi program menggunakan *test case* 1. Solusi yang dihasilkan oleh program sesuai dengan solusi referensi permainan nonogram yang ditunjukkan pada Fig. 8 yang merupakan solusi asli pada permainan Nonogram. Kesesuaian tersebut menunjukkan bahwa algoritma *backtracking* berhasil menemukan konfigurasi papan yang memenuhi seluruh *clue* baris dan kolom.

B. *Test Case 2*

15×15

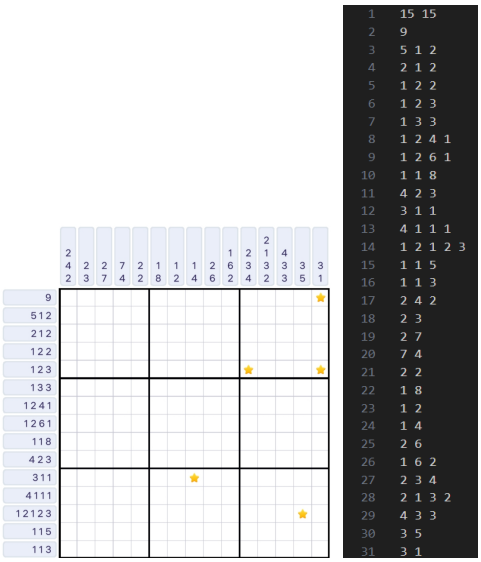

Fig. 9. *Test Case* 2 dari Game Nonogram

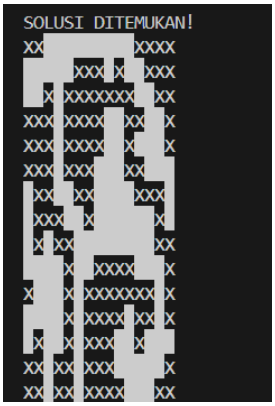

Fig. 10. Hasil Input *Test Case* 2 pada Program (Mencapai Goal State)

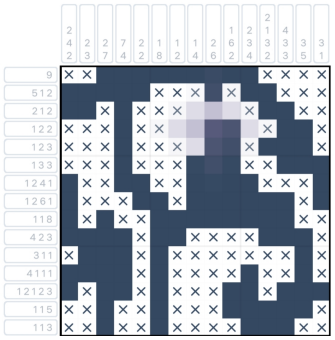

Fig. 11. Solusi *Test Case* 2 di Game Nonogram

Pada *test case* 2, program diuji menggunakan papan berukuran 15×15. Dibandingkan dengan *test case* 1, ukuran papan yang lebih besar menghasilkan ruang pencarian yang lebih luas. Meskipun demikian, melalui mekanisme pencarian rekursif dan proses *backtracking*, algoritma berhasil menemukan konfigurasi papan yang memenuhi seluruh *clue* yang diberikan. Hasil eksekusi program ditunjukkan pada Fig. 10.

Berdasarkan hasil yang diperoleh, solusi yang dihasilkan identik dengan solusi asli permainan Nonogram yang ditunjukkan pada Fig. 11. Kesesuaian tersebut menunjukkan bahwa algoritma *backtracking* tetap mampu menemukan solusi yang benar meskipun ukuran papan dan kompleksitas permasalahan meningkat. Dengan demikian, hasil pengujian pada *test case* 2 semakin memperkuat bahwa program dapat menyelesaikan permainan Nonogram secara akurat.

V. KESIMPULAN

Algoritma *backtracking* dapat secara efektif digunakan untuk mencari solusi pada teka-teki logika Nonogram, sebuah persoalan yang secara komputasional diklasifikasikan ke dalam kelas *NP-Complete*. Dengan menerapkan fungsi pembatas (*bounding function*) pada setiap tahap ekspansi pohon ruang status, pemangkasan (*pruning*) terhadap cabang-cabang yang tidak valid dapat dilakukan secepat mungkin sebelum memicu ekspansi jalan tidak valid yang lebih dalam.

Meskipun persoalan *NP-Complete* pada dasarnya memiliki ruang pencarian yang tumbuh secara eksponensial, hasil implementasi mengindikasikan bahwa algoritma *backtracking* berbasis *pruning* memberikan pendekatan yang sederhana dan efektif untuk penyelesaian Nonogram. Tujuan dari makalah ini adalah untuk memperbaiki proses penyelesaian teka-teki Nonogram, dari yang semula hanya mengandalkan komputasi tebakan buta (*brute-force*) yang tidak konsisten, menjadi sebuah eksekusi yang konsisten dan cerdas dalam mencapai solusi akhir yang optimal.

VI. APENDIKS

GitHub : https://github.com/GeraldoA07/Nonogram-Backtracking-Solver

Youtube : https://youtu.be/juz-4aWNsVw

VII. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih yang sebesar-besarnya kepada Dr. Ir. Rinaldi Munir, M.T., atas dedikasi beliau dalam mengajar dan membimbing mata kuliah IF2211 Strategi Algoritma. Penulis juga sangat mengapresiasi dukungan serta dorongan semangat dari keluarga dan teman-teman selama proses penyelesaian makalah ini. Akhir kata, penulis memanjatkan puji syukur kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya yang senantiasa memberikan kekuatan di sepanjang perjalanan penulisan ini.

REFERENCES

[1] R. Munir, "Strategi Algoritma 2025-2026," Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/stima25-26.htm. [Accessed: Jun. 13, 2026].

[2] Collepic, "Permainan Nonogram," Collepic.net. [Online]. Available: https://collepic.net/. [Accessed: Jun. 13, 2026].

[3] J. M. Buades Rubio, A. Jaume-i-Capó, D. López González, and G. Moyà Alcover, "Solving nonograms using neural networks," *Entertainment Computing* (Elsevier). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187595212400020X. [Accessed: Jun. 13, 2026].

[4] K. Sandvick, "NP Completeness in Nonograms." [Online]. Available: https://kelbybsandvick.github.io/pdf/NPProblemPaper.pdf. [Accessed: Jun. 13, 2026].

[5] R. A. Oosterman, "Complexity and solvability of Nonogram puzzles". [Online]. Available: https://fse.studenttheses.ub.rug.nl/15287/1/Master_Educatie_2017_RAOosterman.pdf. [Accessed: Jun. 13, 2026].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah benar-benar hasil karya saya sendiri, bukan merupakan saduran atau terjemahan dari karya orang lain, dan bukan merupakan tindakan plagiasi.

Bandung, 13 Juni 2026

Geraldo Artemius
13524005