Penyelesaian Parsial Permainan Nonogram dengan Kompleksitas Polinomial

Farras Mohammad Hibban Faddila 13518017
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
farras.faddila@students.itb.ac.id

***Abstrak*—Nonogram merupakan sebuah permainan logika yang berasal dari Jepang. Menyelesaikan sebuah *puzzle* nonogram merupakan permasalahan NP-*complete*. Walaupun begitu, sudah ada banyak algoritma yang dibuat untuk menyelesaikan nonogram. Salah satu metode yang paling sederhana adalah dengan menggunakan *bruteforce* dan DFS, namun metode ini memakan waktu yang cukup lama. Salah satu algoritma yang cukup efektif dalam menyelesaikan permasalahan nonogram adalah dengan memanfaatkan pendekatan *line solving* dan *propagation*. Untuk beberapa kasus tertentu, algoritma ini memiliki kompleksitas waktu polinomial. Pada makalah ini penulis akan melakukan implementasi dan analisis terhadap kompleksitas waktu algoritma-algoritma tersebut.**

***Kata Kunci*—Permainan, nonogram, algoritma, kompleksitas**

I. PENDAHULUAN

Terdapat banyak permainan yang mengasah logika seperti sudoku, kakuro, catur, go. Permainan-permainan tersebut memiliki strategi menang serta alur permainan yang berbeda-beda. Banyak orang yang telah membuat algoritma yang membantu memudahkan penyelesaian permainan-permainan tersebut, baik dalam bentuk algoritma prosedural maupun algoritma yang berbasis pembelajaran mesin dan kecerdasan buatan. Karena permainan-permainan tersebut cukup kompleks, algoritma-algoritma yang dihasilkan memiliki kompleksitas waktu yang cukup besar.

Nonogram merupakan sebuah permainan *puzzle* yang masih cukup jarang diketahui, tidak seperti sudoku yang terkenal. Persoalan nonogram termasuk ke dalam NP-complete, sehingga tidak ada algoritma dengan kompleksitas polinomial yang mampu menyelesaikan persoalan ini secara umum. Banyak riset yang ditujukan untuk mencari pendekatan yang mangkus untuk menyelesaikan permasalahan ini. Metode *bruteforce* pada persoalan ini akan menghasilkan kompleksitas $O((M+N)2^{MN})$, yakni mencoba seluruh kemungkinan untuk tiap petak. Salah satu metode lain yang dapat digunakan untuk menyelesaikan permainan ini adalah dengan menggunakan pendekatan teori graf, yakni dengan membuat sebuah pohon yang menyimpan semua kemungkinan pewarnaan baris dan melakukan *Depth First Search* traversal pada seluruh kemungkinan. Metode ini memiliki kompleksitas eksponensial, namun jauh lebih baik daripada metode *bruteforce* sebelumnya.

Pada artikel ini akan dibahas salah satu algoritma penyelesaian permainan nonogram yang memiliki kompleksitas polinomial untuk beberapa kasus tertentu.

II. LANDASAN TEORI

A. Kompleksitas Algoritma

Setiap algoritma memiliki sebuah properti yang bernama kompleksitas. Kompleksitas dari sebuah algoritma menentukan seberapa mangkus atau tidak algoritma tersebut. Terdapat dua parameter pada algoritma yang diukur, yakni waktu eksekusi dan ruang memori yang dihabiskan. Kedua hal ini menentukan kemangkusan sebuah algoritma karena untuk setiap permasalahan, tentu metode penyelesaian yang diinginkan adalah yang tercepat. Selain itu, agar dapat diakomodasi oleh mesin komputasi yang menjalankan algoritma tersebut, memori yang digunakan oleh algoritma juga harus efisien. Karena kedua hal ini merupakan hal yang penting pada sebuah algoritma, maka terdapat dua jenis kompleksitas, yakni kompleksitas waktu dan kompleksitas memori.

Kompleksitas dari sebuah algoritma merupakan sebuah fungsi yang memiliki parameter yang bergantung pada nilai masukan dari algoritma tersebut. Misalkan algoritma tersebut memiliki input yang dinyatakan dalam sebuah peubah n. Maka, kompleksitas dari algoritma tersebut merupakan sebuah fungsi dalam n. Kompleksitas waktu dinyatakan dalam notasi $T(n)$, sedangkan kompleksitas ruang dinyatakan dalam notasi $S(n)$. Setelah ini, kompleksitas yang akan dibahas hanyalah kompleksitas waktu, sehingga pernyataan kompleksitas akan diartikan sebagai kompleksitas waktu. Kompleksitas waktu dari sebuah algoritma ditentukan dari banyaknya operasi dasar yang terjadi di dalam algoritma tersebut. Operasi dasar ini dapat berupa operasi aritmatika, logika, *assignment*, maupun operasi lainnya. Beberapa operasi tersebut mungkin tidak akan dieksekusi karena, misalnya, berada pada sebuah percabangan yang tidak dimasuki. Hal ini dipengaruhi oleh nilai yang ada pada memori maupun input. Oleh karena itu, terdapat kemungkinan terbaik serta terburuk saat sedang mengeksekusi sebuah algoritma. Kompleksitas waktu dibedakan ke dalam tiga jenis bergantung pada kasus ekstremnya tersebut seperti di bawah ini.

1) $T_{min}(n)$, yaitu kompleksitas waktu untuk kasus terbaik. Kasus terbaik ini terjadi apabila input dari algoritma menyebabkan algoritma selesai dengan cepat, dan dengan waktu yang paling minimal.

2) $T_{max}(n)$, yaitu kompleksitas waktu untuk kasus terburuk. Kasus terburuk ini terjadi apabila input dari algoritma menyebabkan algoritma selesai dalam waktu yang paling lama.
3) $T_{avg}(n)$, yaitu kompleksitas waktu rata-rata untuk semua kasus. Untuk menghitung nilai dari $T_{avg}(n)$ diperlukan nilai dari $T(n)$ untuk setiap input n yang mungkin.

B. Notasi Asimtotik

Dalam perhitungan kompleksitas algoritma, biasanya hal yang paling menentukan adalah pertumbuhan (*growth*) dari fungsi kompleksitas tersebut. Dua fungsi yang memiliki pertumbuhan yang sama, namun hanya berbeda faktor pengali saja, dapat dianggap memiliki kompleksitas yang sama. Oleh karena itu muncul notasi kompleksitas asimtotik.

Misalkan terdapat sebuah algoritma A yang memiliki kompleksitas sebesar $T(n)$. Notasi asimtotik untuk menentukan kompleksitas asimtotik dari A dibedakan ke dalam tiga jenis sebagai berikut.

1) *Notasi Big-O*

$T(n) = O(f(n))$ jika dan hanya jika pertumbuhan dari fungsi $f(n)$ tidak lebih lambat dari pertumbuhan $T(n)$. Secara matematis, pernyataan tersebut dapat ditulis sebagai berikut.

$$\exists n_0 \in \mathbb{N}: \exists c: n \geq n_0 \rightarrow T(n) \leq cf(n)$$

2) *Notasi Big-Omega*

$T(n) = \Omega(f(n))$ jika dan hanya jika pertumbuhan dari fungsi $f(n)$ tidak lebih cepat dari pertumbuhan $T(n)$. Secara matematis, pernyataan tersebut dapat ditulis sebagai berikut.

$$\exists n_0 \in \mathbb{N}: \exists c: n \geq n_0 \rightarrow T(n) \geq cf(n)$$

3) *Notasi Big-Theta*

$T(n) = \Theta(f(n))$ jika dan hanya jika pertumbuhan dari $f(n)$ sama dengan $T(n)$. Pernyataan tersebut ekuivalen dengan $T(n) = O(f(n))$ dan $T(n) = \Omega(f(n))$. Dengan kata lain, syarat tersebut ekuivalen dengan syarat berikut ini.

$$\exists n_0 \in \mathbb{N}: \exists c_1, c_2: n \geq n_0 \rightarrow c_1 f(n) \leq T(n) \leq c_2 f(n)$$

III. NONOGRAM

Permainan nonogram merupakan permainan *puzzle* yang berasal dari Jepang. Permainan ini diciptakan oleh Non Ishida pada tahun 1987. Pada tahun 1990, James Dalgety mempopulerkan permainan ini dengan sebutan nonogram, yang diambil dari nama Non Ishida, dan permainan ini mulai muncul pada media *The Sunday Telegraph* tiap minggu.

Pada permainan nonogram terdapat sebuah grid persegi panjang berukuran $w \times h$. Pada setiap baris dan kolom, terdapat sebuah larik yang berisi bilangan-bilangan bulat positif. Larik tersebut dapat kosong. Larik tersebut mendeskripsikan aturan yang harus dipenuhi oleh petak-petak pada baris atau kolom tersebut. Awalnya grid tersebut kosong. Tujuan dari permainan nonogram ini adalah mewarnai hitam beberapa petak pada grid tersebut sehingga petak-petak yang diwarnai hitam tersebut memenuhi deskripsi yang terdapat pada larik-larik tersebut. Perhatikan bahwa karena pada tiap baris dan tiap kolom terdapat tepat sebuah larik, maka terdapat $w + h$ larik yang memuat deskripsi bagaimana petak-petak tersebut harus diwarnai.

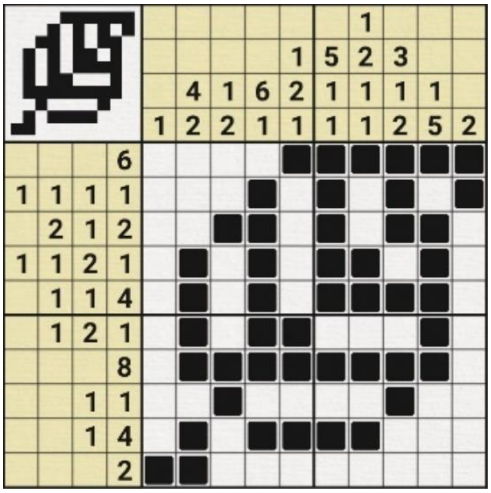
Definisikan sebuah garis sebagai sebuah kolom atau baris. Sehingga, penggunaan istilah baris dapat mengacu kepada kedua kata tersebut. Definisikan awal dari sebuah garis sebagai petak terkiri suatu baris atau petak teratas suatu kolom, dan akhir dari sebuah garis sebagai petak terkanan suatu baris atau petak terbawah suatu kolom. Definisikan sebuah blok sebagai himpunan petak yang saling bersisian dan terletak pada baris atau kolom yang sama, dan dengan ukuran yang paling maksimal.

Sebut S sebagai grid pada permainan nonogram yang beberapa petaknya telah diwarnai (bisa tidak ada sama sekali). Maka, S disebut sebagai kondisi akhir, atau grid target, apabila S memenuhi syarat berikut ini.

Jika pada suatu garis, larik yang ada pada garis tersebut adalah $\{d_1, d_2, \ldots, d_k\}, k \geq 0$, maka beberapa petak pada garis tersebut diwarnai hitam sedemikian sehingga petak-petak hitamnya membentuk blok-blok dengan panjang $d_1, d_2, \ldots, d_k$ secara berturutan.

Tujuan dari permainan nonogram adalah mewarnai petak-petak pada grid sehingga terbentuk sebuah grid target. Berikut ini adalah salah satu contoh grid nonogram dengan larik-larik seperti berikut yang merupakan sebuah grid target.

1. Baris 1: {6}
2. Baris 2: {1,1,1,1}
3. Baris 3: {2,1,2}
4. Baris 4: {1,1,2,1}
5. Baris 5: {1,1,4}
6. Baris 6: {1,2,1}
7. Baris 7: {8}
8. Baris 8: {1,1}
9. Baris 9: {1,4}
10. Baris 10: {2}
11. Kolom 1: {1}
12. Kolom 2: {4,2}
13. Kolom 3: {1,2}
14. Kolom 4: {6,1}
15. Kolom 5: {1,2,1}
16. Kolom 6: {5,1,1}
17. Kolom 7: {1,2,1,1}
18. Kolom 8: {3,1,2}
19. Kolom 9: {1,5}
20. Kolom 10: {2}



Gambar 1. Nonogram 10 × 10, diambil dari [8]

Perhatikan bahwa pada tiap baris dan kolom, petak-petak hitamnya tersusun atas beberapa blok, dan panjang dari masing-

masing blok secara terurut sesuai dengan larik yang ada pada baris atau kolom yang bersesuaian. Oleh karena itu, pewarnaan grid ini merupakan salah satu penyelesaian yang mungkin, sehingga grid tersebut merupakan grid target.

Biasanya, sebuah *puzzle* nonogram hanya memiliki solusi unik. Namun, sebarang *assignment* larik pada tiap baris dan kolom tidak akan menjamin keunikan solusi ataupun ketersediaan solusi. Sebagai contoh, kedua nonogram dengan masing-masing memiliki grid berukuran 2×2 di bawah ini memiliki larik yang sama untuk setiap baris, yakni {1}, tetapi memiliki pewarnaan yang berbeda,

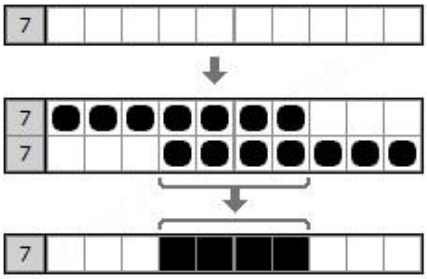
Gambar 2. Dua nonogram berbeda 2×2 yang memiliki larik yang sama pada tiap baris dan kolomnya

Dalam beberapa variasi permainan nonogram, selain dapat mewarnai sebuah petak dengan warna hitam, hal lain yang dapat dilakukan adalah menandai sebuah petak dengan sebuah tanda lain, seperti tanda silang. Hal ini bertujuan untuk memberikan tanda bahwa petak tersebut tidak mungkin diwarnai. Petak yang diberi tanda silang disebut sebagai petak yang tercoret. Melakukan sebuah pencoreta ekuivalen dengan melakukan sebuah pewarnaan, karena kedua hal tersebut sama-sama memberikan informasi mengenai kondisi petak-petak pada grid target.

Tanpa menggunakan algoritma komputasi yang terstruktur, terdapat beberapa strategi, yang bergantung pada kondisi grid, untuk menyelesaikan persoalan nonogram. Strategi-strategi tersebut dibedakan ke dalam jenis langkah yang dilakukan, yakni pewarnaan atau pencoretan.

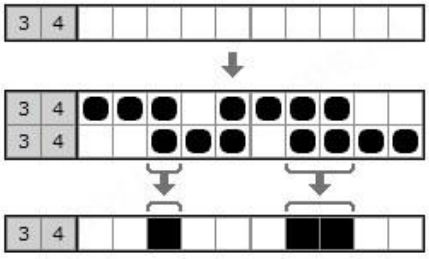
1) *Pewarnaan*

Apabila ukuran larik pada suatu garis hanya satu dan bilangan pada larik tersebut bernilai lebih dari setengah panjang garis tersebut, maka terdapat beberapa petak di bagian tengah garis yang pasti dijamin harus bewarna hitam seperti contoh berikut.



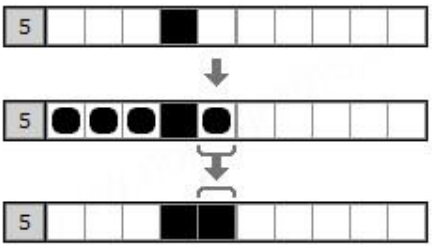
Gambar 3. Skenario nonogram, diambil dari [4]

Jika ukuran larik lebih dari satu, cara ini masih dapat diaplikasikan jika pada kedua kasus ekstrim (penempatan blok di rata kiri dan rata kanan) terdapat petak yang saling tindih seperti contoh berikut.



Gambar 4. Skenario nonogram, diambil dari [4]

Jika pada suatu garis sudah terdapat petak yang bewarna hitam dan jarak petak tersebut ke awal dari garis tersebut kurang dari elemen pertama pada larik tersebut, yaitu l, maka terdapat sejumlah petak pada garis tersebut setelah petak bewarna hitam tersebut yang dapat diwarnai. Apabila petak yang terletak tepat setalah petak bewarna tersebut tidak diwarnai, maka tidak akan ada tempat yang cukup untuk menemukan blok dengan panjang l pada garis tersebut.

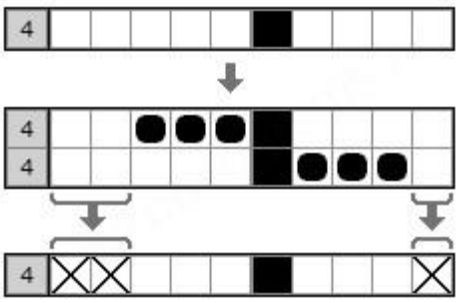


Gambar 5. Skenario nonogram, diambil dari [4]

Hal di atas juga berlaku untuk bagian akhir dari suatu garis.

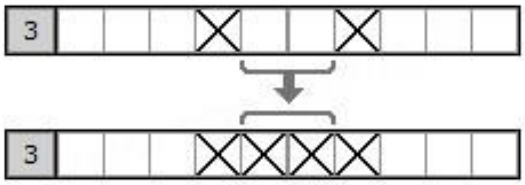
2) *Pencoretan*

Jika ada petak-petak pada suatu garis yang tidak mungkin diwarnai, maka menandai petak tersebut merupakan sebuah strategi yang bagus, karena hal itu memberi informasi mengenai grid target.



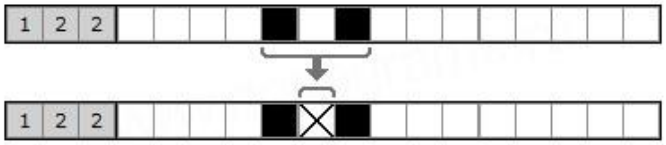
Gambar 6. Skenario nonogram, diambil dari [4]

Apabila terdapat dua petak tercoret pada sebuah garis dan seluruh anggota larik pada garis tersebut bernilai lebih besar daripada banyaknya petak di antara kedua petak tercoret tersebut, maka tidak mungkin ada blok petak bewarna yang terletak di antara kedua petak tercoret tersebut. Sehingga, seluruh petak di antara mereka berdua dapat dicoret juga.



Gambar 7. Skenario nonogram, diambil dari [4]

Pencoretan juga dapat dilakukan pada contoh berikut, ketika petak di antara dua petak bewarna hitam pada suatu garis tidak mungkin diwarnai karena akan membentuk blok dengan panjang lebih besar dari elemen larik pada garis tersebut.



Gambar 8. Skenario nonogram, diambil dari [4]

Nonogram yang disebutkan di atas hanyalah terdiri atas satu warna. Terdapat variasi dari permainan nonogram yang terdiri atas dua atau lebih warna. Pada nonogram jenis ini, setiap elemen larik pada baris atau kolom tidak hanya menyimpan informasi panjang blok, tetapi juga informasi mengenai warna yang harus dimiliki oleh kotak-kotak pada blok tersebut. Makalah ini hanya akan membahas nonogram yang terdiri atas satu warna saja.

IV. Algoritma Penyelesaian Nonogram

A. Notasi dan Definisi

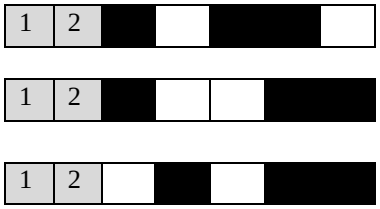
Seperti yang telah disebutkan sebelumnya, pada permainan nonogram terdapat sebuah grid berukuran $w \times h$. Notasikan r_i dan c_j sebagai larik pada baris ke-i dan kolom ke-j secara berturut-turut, dan t_{ij} sebagai bilangan pada petak yang berada pada baris ke-i dan kolom ke-j dengan ketentuan $t_{ij} = 0$ apabila petak pada baris ke-i dan kolom ke-j tidak diwarnai dan bernilai 1 apabila petak tersebut bewarna hitam. Notasikan k sebagai ukuran maksimal dari seluruh larik-larik yang ada pada petak tersebut.

Larik a dan kondisi pewarnaan garis b dikatakan cocok apabila dengan kondisi pewarnaan garis b sekarang, yakni beberapa sudah diwarnai hitam dan beberapa sudah dicoret, masih dapat dilakukan gerakan (warna atau coret) pada petak yang masih kosong sehingga pola pewarnaan yang dihasilkan cocok dengan larik a

Pewarnaan garis a dan pewarnaan garis b dikatakan cocok apabila terdapat serangkaian gerakan pada petak-petak yang masih kosong pada kedua garis sehingga keduanya membentuk pola yang sama.

B. Metode Penyelesaian

Hal pertama yang dilakukan adalah menerima larik-larik untuk setiap garis. Setelah itu, untuk setiap larik akan dibuat sebuah larik yang isinya merupakan seluruh kemungkinan pewarnaan dari garis yang bersesuaian dengan larik tersebut yang cocok dengan larik tersebut. Sebagai contoh, apabila pada larik pada garis tersebut adalah $\{1,2\}$ dan panjang garis tersebut adalah 5, maka larik yang terbentuk adalah $\{\{1,0,1,1,0\}, \{1,0,0,1,1\}, \{0,1,0,1,1\}\}$, karena terdapat tiga jenis pewarnaan yang cocok dengan larik$\{1,2\}$ pada garis dengan panjang 5.



Gambar 9. Kemungkinan pengisian garis sesuai dengan elemen pada larik

Pada setiap kemungkinan ini, berikutnya dicari petak yang pasti akan diwarnai. Langkah ini merupakan implementasi dari salah satu trik yang telah dijelaskan pada bab sebelumnya. Pada contoh di atas, petak yang pasti diwarnai ini adalah petak yang terletak di urutan keempat dari kiri, karena di tiap kemungkinan pewarnaan baris tersebut, petak tersebut selalu diwarnai.

Langkah selanjutnya yang dilakukan adalah propagasi (*propagation*). Pertama dibuat terlebih dahulu sebuah struktur data himpunan *(set)* G_1 dan G_2 yang masing-masing akan memuat indeks-indeks baris dan indeks-indeks kolom. Mereka diinisialisasi sebagai himpunan kosong. Selanjutnya, G_2 diisi dengan seluruh indeks kolom.

Berikutnya, dibuat sebuah fungsi boolean $Fix(i,j)$ dan fungsi bilangan bulat $Warna(i,j)$ untuk setiap pemrosesan garis. Pemrosesan garis ini disebut dengan proses *line solving*. Jika garis yang sedang diproses adalah garis l dan larik yang mendeskripsikan garis tersebut adalah $\{m_1, m_2, \dots, m_k\}$, maka $Fix(i,j)$ akan bernilai $True$ jika kondisi pewarnaan garis l mulai dari awal hingga petak ke-i dan larik $\{m_1, m_2, \dots, m_j\}$ cocok. Nilai $Fix(Length(l), k)$ dicari dengan menggunakan *Dynamic Programming* untuk menentukan apakah nonogram tersebut masih dapat diselesaikan atau tidak.

Fungsi $Warna(i,j)$ sendiri berfungsi untuk mencari pewarnaan maksimum yang mungkin pada saat pemrosesan garis l di atas, di mana $Warna(i,j)$ berarti banyak petak kosong maksimum yang dapat diwarnai pada upagaris l, yakni i petak pertama, dengan larik $\{m_1, m_2, \dots, m_j\}$. Nilai dari fungsi warna ini juga dicarai menggunakan teknik *Dynamic Programming*. Total kompleksitas dari kedua algoritma di atas adalah $O(k \times Length(l)^2)$. Karena $Length(l)$ dapat bernilai w atau h, maka kompleksitas dari algoritma *line solving* adalah $O(k \times \max(h,w)^2)$. Hubungan rekursif untuk pengisian tabel *Dynamic Programming* kedua fungsi dapat ditemukan lebih lanjut pada [1].

Selanjutnya, selama G_2 tidak kosong, dilakukan hal berikut.

1) Untuk setiap elemen i pada G_2, dilakukan *line solving* pada kolom indeks ke-i, sehingga terdapat petak baru yang diwarnai. Indeks baris posisi petak baru yang diwarnai ini dimasukkan pada himpunan G_1.
2) Setelah iterasi pada G_2 selesai, G_2 dikosongkan.
3) Selanjutnya, dilakukan iterasi pada G_1, dengan langkah yang serupa dengan langkah pada 1).
4) Setelah iterasi pada G_1 selesai, G_1 dikosongkan.

procedure Propagate(G_1, G_2)

1. ***while*** $(G_2 \neq \emptyset)$ ***do***
2. ***for*** *each* $c \in G_2$
3. ***if*** $\left(not\ Fix\left(c, Larik(c)\right)\right)$ ***then*** $status \leftarrow False, \textbf{\textit{return}}$ \\ Grid tidak dapat diselesaikan
4. $c \leftarrow Warna(c, Larik(c))$
5. $\Delta \leftarrow$ himpunan semua petak yang baru diwarnai
6. ***for*** *each* $i \in \Delta$
7. Masukkan indeks baris i pada G_1
8. Keluarkan c dari G_2
9. ***for*** *each* $r \in G_1$
   if $(not\ Fix(r, Larik(r)))$ *then*
10. $return$ \\ Grid tidak dapat diselesaikan
11. $r \leftarrow Warna(r, Larik(r))$
12. $\Delta \leftarrow$ himpunan semua petak yang baru diwarnai
13. ***for*** *each* $i \in \Delta$
14. Masukkan indeks kolom i pada G_2
15. Keluarkan r dari G_1
16. ***end*** *while*
17. ***if*** $(semua\ petak\ diwarnai)$ ***then*** $status \leftarrow success, \textbf{\textit{return}}$ \\ Grid selesai diwarnai
18. ***else*** $status \leftarrow incomplete$ \\ Perlu dilakukan *backtracking* lebih lanjut

end procedure

Tabel 1. *Pseudocode* dari prosedur propagasi

Karena total hanya terdapat sebanyak $w \times h$ petak secara keseluruhan, maka pada langkah di atas, banyak penambahan elemen yang terjadi pada G_1 dan G_2 jika dijumlah tidak akan melebihi $O(wh)$. Karena kompleksitas dari algoritma *line solving* adalah $O(k \times \max(w,h)^2)$, maka total dari kompleksitas algoritma nonogram dengan *line solver* dan *propagation* ini adalah $O(k \times \max(w,h)^3)$.

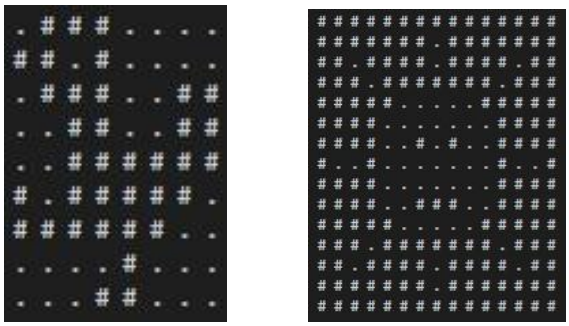
Salah satu masalah yang akan muncul saat proses propagasi adalah, bisa jadi ditemui sebuah situasi di mana tidak ada lagi garis yang bisa diwarnai petaknya secara pasti yang manapun, atau G_2 sudah kosong tetapi belum mencapai grid target. Jika hal ini terjadi, maka dilakukan iterasi pada semua pewarnaan petak yang mungkin, yang tidak akan menyalahi deskripsi pada larik baris tersebut. Lalu, dilakukan DFS dan *backtracking*, dengan cabangnya adalah seluruh petak yang mungkin dapat diwarnai pada suatu baris. Algoritma yang dihasilkan akan menjadi eksponensial. Pada [1] dijelaskan sebuah optimisasi untuk pemilihan cabang saat melakukan traversal DFS ini, tetapi hal tersebut diluar bahasan makalah ini.

V. KASUS UJI DAN PERFORMA

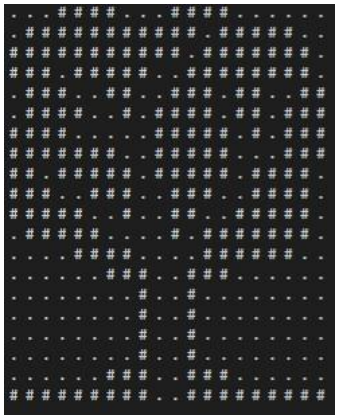
Berikut ini adalah performa dari algoritma di atas, dengan hasil grid target masing-masing kasus uji mengikuti berikutnya. Program diimplementasi dalam bahasa pemrograman Python 3, dan dijalankan pada komputer dengan prosesor AMD A6 2.50 GHz dan RAM 4GB.

No.	Lebar Grid (w)	Tinggi Grid (h)	Ukuran larik maksimal (k)	Waktu (ms)
1.	8	9	2	54.549
2.	15	15	5	42.129
3.	20	20	5	141.836
4.	30	30	9	31699.393
5.	35	40	10	136614.556

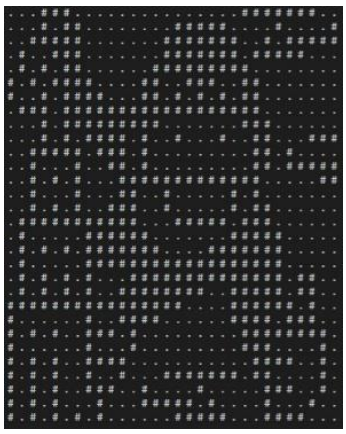
Tabel 2. Performa kasus uji



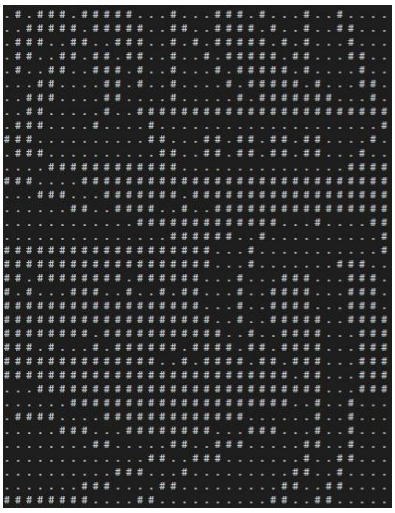
Gambar 10. Kasus Uji 1 dan 2



Gambar 11. Kasus Uji 3



Gambar 12. Kasus Uji 4



Gambar 12. Kasus Uji 5

V. KESIMPULAN

Permainan nonogram merupakan persoalan NP-*complete* sehingga tidak ada algoritma dengan kompleksitas waktu polinomial yang mampu memecahkan permainan ini secara general. Berdasarkan tabel kasus uji, setelah ukuran grid menjadi sebesar sekitar di atas 30, kenaikan dari waktu eksekusi program juga meningkat cepat. Akan tetapi, untuk kasus-kasus nonogram dengan grid kecil dan menengah (maksimal hingga 30) yang masih dapat diselesaikan secara normal, algoritma pemecahan nonogram yang berbasis *line solving* dan *propagation* ini masih dapat menemukan solusinya dalam waktu yang relatif cepat, karena untuk menyelesaikan nonogram secara manual, waktu minimal yang dibutuhkan bisa berkisar sepuluh menit.

VII. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa, beserta kedua orang tua yang telah mendukung penulis, serta seluruh Bapak dan Ibu dosen pengampu mata kuliah IF2120 Matematika Diskrit atas bantuan selama proses pengerjaan makalah ini dalam rangka memenuhi tugas terakhir mata kuliah Matematika Diskrit. Penulis juga mengucapkan terima kasih kepada teman-teman dan pihak-pihak lain yang telah mendukung saya selama proses pengerjaan makalah ini.

REFERENSI

[1] I. Wu, D. Sun, L. Chen, K. Chen, C. Kuo, H. Kang, H. Lin, "An Efficient Approach to Solving Nonograms", IEEE Transactions on Computational Intelligence and AI in Games, 2013, Vol. 5/3.
[2] https://rosettacode.org/wiki/Nonogram_solver, diakses pada 5 Desember 2019.
[3] R. Munir, *Matematika Diskrit*. Bandung: Departemen Teknik Informatika Institut Teknologi Bandung, 2003.
[4] https://www.nonograms.org/methods, diakses pada 5 Desember 2019.
[5] https://www.puzzlemuseum.com/griddler/gridhist.htm, diakses pada 5 Desmber 2019.
[6] C. Yu, H. Lee, L. Chen, "An Efficient Algorithm for Solving Nonograms", Published online: 13 November 2009, Springer Science-Business Media.
[7] K. Batenburga, W. Kostersb, "Solving Nonograms by Combining Relaxations", Pattern Recognition 42, 2008.
[8] "Nonogram Katana" v.11.23 (ucdevs, 2019).
[9] https://puzzlygame.com/pages/nonogram_history/, diakses pada 6 Desember 2019.

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, 7 Desember 2019

Farras Mohammad Hibban Faddila - 13518017