Implementasi Algoritma Backtracking dengan Constraint Propagation untuk Menyelesaikan Permainan Futoshiki

Ignacio Kevin Alberiann - 15223090
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ignacioalberiann@gmail.com , 15223090@mahasiswa.itb.ac.id

Abstrak*—Futoshiki adalah permainan teka-teka logika berbasis grid N x N yang memadukan aturan Sudoku dengan kendala perbandingan ketaksamaan (< atau >). Sebagai sebuah Constraint Satisfaction Problem (CSP), metode pencarian solusinya perlu dilakukan secara efisien melalui implementasi algoritma runut-balik (backtracking) dengan peningkatan constraint propagation. Program dibuat menggunakan Bahasa Python dan memanfaatkan heuristik Minimum Remaining Values (MRV) dan Least Constraining Values (LCV) untuk optimasi proses. Perbandingan kinerja antara backtracking dengan dan tanpa constraint propagation dilakukan pada berbagai ukuran grid papan, dengan mengukur waktu eksekusi dan jumlah simpul yang dikunjungi. Hasil makalah ini menunjukkan bahwa constraint propagation membantu secara drastis untuk mencari solusi teka-teki secara efisien, terutama untuk papan berukuran besar.

***Kata Kunci*—Futoshiki; constraint propagation; backtracking; runut-balik; puzzle; teka-teki; mrv; lcv**

I. PENDAHULUAN

Permainan Futoshiki atau *Unequal* adalah permainan teka-teki logika berbasis grid yang menggabungkan elemen permainan Sudoku dengan kendala atau *constraint* perbandingan ketaksamaan (lebih besar dari atau kurang dari). Pada permainan ini, pemain dihadapkan pada sebuah papan persegi N x N yang harus diisi dengan angka 1 hingga N di setiap selnya dengan peraturan yang sama seperti Sudoku, yaitu setiap baris dan setiap kolom harus berisi setiap angka tepat satu kali. Keunikan Futoshiki terletak pada simbol perbandingan ketaksamaan (< atau >) yang ditempatkan di antara beberapa sel untuk menunjukkan relasi nilai antar sel yang berdekatan. Pemain dalam permainan ini harus memecahkan teka-teki dengan memastikan kedua peraturan tersebut sudah dijawab dengan benar, yaitu semua kendala keuniikan baris/kolom dan kendala perbandingan terpenuhi secara bersamaan.

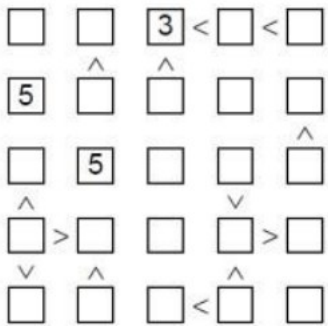

Fig. 1. Ilustrasi Kondisi Awal Papan Futoshiki (*Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_082.pdf*)

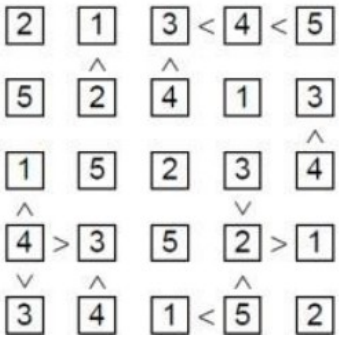

Fig. 2. Ilustrasi Kondisi Solusi Papan Futoshiki (*Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_082.pdf*)

Oleh karena kompleksitasnya, digunakan algoritma yang dibuat dan dijalankan untuk mempercepat proses pencarian solusi ketimbang pencarian solusi manual dalam waktu normal berpikir manusia. Permainan Futoshiki termasuk dalam kategori *Constraint Satisfaction Problems* (CSPs), di mana tujuan utama dalam menyelesaikan masalah ini adalah menemukan penugasan nilai ke variabel yang ada sehingga semua kendala yang ada pada permainan terpenuhi.

Meski algorimta *brute force* atau *exhaustive search* dapat digunakan untuk mencari semua solusi yang ada pada permainan, algoritma runut-balik (*backtracking*) digunakan karena sudah dikenal jauh lebih efisien dan cepat dalam mencari solusi. Hal ini memungkinkan pencarian yang jauh lebih cepat karena program komputer tidak perlu mencari seluruh kombinasi pola yang ada pada kondisi papan tertentu. Algoritma runut-balik (*backtracking*) adalah algoritma *Depth-First-Search* (DFS) yang akan melakukan perhentian node (*pruning*) apabila simpul tidak mengarah ke solusi yang diinginkan.

Constraint Propagation adalah Teknik yang digunakan secara bersamaan dengan algoritma runut-balik (*backtracking*) untuk lebih meningkatkan efisiensi pencarian solusi pada CSPs. Teknik ini bekerja dengan secara proaktif mengurangi *domain* (nilai-nilai yang mungkin atau ruang solusi) dari variabel-variabel yang belum diberi nilai segera setelah suatu variabel diberi nilai atau ketika suatu kendala terpenuhi atau suatu kendala dilanggar. Dengan melakukan ini, propagasi kendala dapat mendeteksi masalah konflik lebih awal dalam proses pencarian dan memangkas cabang-cabang simpul pohon pencarian yang tidak menuju solusi *valid*, serta mengurangi jumlah total simpul yang harus dieksplorasi, mempercepat pencarian solusi.

II. DASAR TEORI

A. Futoshiki

Futoshiki (不等式), juga dikenal sebagai "*More or Less" atau "Unequal"*, adalah teka-teki (*puzzle*) logika dari negara Jepang yang pertama kali dipublikasikan oleh Nikoli. Tujuan dalam permainan ini adalah untuk menemukan angka-angka yang tersembunyi di dalam sel-sel papan dengan setiap sel diisi dengan angka antara 1 dan ukuran papan. Pada setiap baris dan kolom, setiap angka muncul tepat satu kali, membentuk suatu urutan angka pada papan yang disebut sebagai kotak Latin [4]. Pada awal permainan, beberapa angka dapat terungkap dari awal pada sel-sel papan dengan simbol ketaksamaan di antara sel-selnya [5].

Aturan dasar rincinya adalah sebagai berikut:

1. Pengisian Grid

   Setiap sel dalam *grid* N x N harus diisi dengan angka unik dari 1 hingga N

2. Keunikan Baris dan Kolom

   Setiap baris dan kolom harus berisi setiap angka dari 1 hingga N tepat satu kali, mirip dengan aturan Sudoku

3. Kendala Ketidaksamaan

   Simbol ketaksamaan (< atau >) yang ditempatkan di antara sel-sel yang berdekatan harus terpenuhi. Misal, jika ada simbol > di antara sel A dan sel B (A > B), maka nilai yang diisi pada sel A harus lebih besar dari nilai di sel B, dan begitu pula sebaliknya.

Permainan Futoshiki dibuat untuk memiliki teka-teki dengan satu atau lebih solusi. Teka-tekinya hanya didasarkan oleh logika dan deduksi. Permainan ini menantang kemampuan deduksi logis pemain dan seringkali membutuhkan kombinasi penalaran langsung dan strategi coba-coba untuk menemukan solusinya.

B. Backtracking

Algoritma runut-balik (*backtracking*) adalah teknik langkah-langkah pencarian umum yang digunakan untuk menemukan solusi dari masalah komputasi dengan cara secara bertahap membangun solusi dan, ketika menemui jalan buntu, secara rekursif mundur (*backtrack*) untuk mencoba jalur lain. Algoritma ini dapat digambarkan sebagai algoritma pencarian mendalam (*Depth-First*-Search - DFS) yang dimodifikasi menjadi lebih efisien dengan memangkas jalur cabang pohon pencarian yang tidak mengarah ke suatu solusi yang *valid*.

Algoritma ini bekerja dengan melakukan pencarian melalui ruang keadaan (*state-space*) yang direpresentasikan sebagai pohon. Setiap node dalam pohon adalah sebuah keadaan parsial masalah, dan setiap cabang adalah sebuah Keputusan atau penugasan nilai. *Backtracking* menjelajahi pohon pencarian solusi secara mendalam *(depth-first*), dan apabila tidak mengarah ke suatu solusi, *backtrack* ke node induk untuk mencoba alternatif solusi lain. [2]

Properti umum algoritma *backtracking* adalah sebagai berikut:

1. Solusi Persoalan

   Solusi dinyatakan sebagai vector dengan n-tuple: X = (x_1 , x_2 , …, x_n), xi ∈ Si . Umumnya $S_1 = S_2 = \dots = S_n$. Contoh: Pada persoalan 1/0 knapsack Si = {0, 1}, xi = 0 atau 1

2. Fungsi Pembangkit

   Fungsi pembangkit nilai xk dinyatakan dalam predikat T(k) dimana T(k) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi

3. Fungsi Pembatas (*Bounding Function*)

   Dinyatakan dalam predikat : B (x_1 , x_2,..,x_k). B bernilai true jika (x_1 , x_2,..,x_k) mengarah ke solusi. Jika true, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika false, maka (x_1 , x_2,..,x_k) dibuang

C. *Constraint Propagation* (Propagasi Kendala)

Constraint Propagation atau propagasi kendala dalah teknik yang digunakan dalam *Constraint Satisfaction Problems* (CSPs) untuk secara efisien mengurangi ruang pencarian dengan menghilangkan nilai-nilai yang tidak konsisten dari domain variabel. Tujuannya adalah untuk memperketat kendala serapat mungkin, sehingga mengurangi jumlah pilihan yang harus dipertimbangkan oleh algoritma pencarian [1], [6], [7].

Salah satu bentuk paling umum dari *constraint propagation* yang sering digunakan dengan algoritma *backtracking* adalah *Forward Checking* [7]. Ketika sebuah variabel diberi nilai, *forward checking* memeriksa variabel-variabel lain yang belum diberi nilai, tetapi memiliki kendala dengan variabel yang baru diisi dengan suatu nilai. Nilai-nilai dari *domain* variabel lain yang tidak konsisten dengan penugasan yang baru saja

dilakukan akan dihapus dari daftar pilihan. Jika *domain* variabel manapun menjadi kosong setelah proses penempatan nilai ke dalam variabel, penugasan dianggap mengarah ke suatu konflik yang tidak memberikan solusi, dan algoritma dapat langsung melakukan *backtrack* tanpa perlu menjelajahi cabang pohon pencarian tersebut lebih lanjut lagi. [7]

Manfaat dari implementasi *constraint propagation* adalah kemampuannya untuk mendeteksi kegagalan lebih awal, atau yang disebut sebagai *early failure detection* [7]. Hal ini secara signifikan mengurangi ukuran besar pohon pencarian yang harus dijelajahi program sehingga proses menemukan solusi menjadi lebih cepat. Teknik ini memastikan bahwa *domain*/himpunan solusi setiap variabel hanya berisi nilai-nilai yang masih mungkin untuk diisi sehingga kendala yang ada menjadi terpenuhi. [1]

III. IMPLEMENTASI

Untuk mengimplementasikan dan menyelesaikan permasalahan pencarian solusi pada permainan ini, penulis menggunakan program sederhana dalam Visual Studio Code dengan bahasa pemrograman Python karena kemampuan pencarian dan analisis datanya yang baik, didukung oleh sintaks yang bersih dan dukungan dalam membangun struktur data yang fleksibel.

Permasalahan Futoshiki adalah sebuah *Constraint Satisfaction Problems* (CSPs) yang umumnya melibatkan dua teknik utama, yaitu melakukan pencarian / *search* secara *backtracking (*metode pencarian paling umum untuk CSPs) yang secara rekursif menugaskan nilai ke variabel satu per satu, serta *constraint propagation* (propagasi kendala) sebagai sebuah teknik untuk mengurangi ruang pencarian dengan mendeteksi konflik kendala lebih awal dan memangkas cabang pencarian pohon yang tidak *valid*.

Cara kerja program ini secara sederhana adalah menerima input papan dalam sebuah file .txt, memprosesnya menggunakan algoritma *backtracking* dengan dua aturan heuristik, serta memberikan output dari hasil *backtracking*, baik saat menggunakan *constraint propagation* maupun tidak.

A. Teknis Program

Input utama dalam program ini adalah sebuah papan representasi permainan yang berupa sebuah matriks atau *array* dua dimensi dengan ukuran N kali N di mana setiap sel dapat berisi nilai awal (jika ada) atau nol untuk sel kosong. Selanjutnya, input dilengkapi dengan daftar *constraints* atau kendala yang mencakup semua kendala ketaksaamaan ("lebih besar dari" atau "kurang dari") yang direpresentasikan dalam tupel (koordinat baris dan kolom sel pertama, koordinat baris dan kolom sel kedua, dan tipe kendala perbandingan ketaksamaan).

Proses pencarian solusi inti dimulai dengan melakukan representasi domain variabel di mana setiap sel kosong pada papan dianggap sebagai sebuah variabel. Setiap variabel akan memiliki *domain* atau himpunan solusi dari nilai-nilai yang mungkin (angka 1 hingga N). *Domain* akan diinisialisasi untuk semua sel kosong dengan {1, 2, … , N}.

Langkah selanjutnya adalah menggunakan fungsi validasi atau *constraint checking* di mana fungsinya adalah untuk memeriksa apakah penempatan nilai pada sel tertentu memenuhi semua kendala yang relevan (kendala baris, kolom, dan kendala ketaksamaan).

B. Backtracking tanpa menggunakan Constraint Propagation

Program kemudian dilanjutkan dengan algoritma *backtracking*. Program mencoba menggunakan algoritma *backtracking* murni tanpa *constraint propagation*. Algoritma *backtracking* menggunakan dua jenis aturan heuristik yang paling umum digunakan dalam permasalahan CSPs, yaitu aturan heuristik *Minimum Remaining Values* (MRV) dan *Least Constraining Value* (LCV).

Pada aturan RMV, program memilih variabel (sel) yang belum diberi nilai dengan jumlah nilai *domain* yang paling sedikit sehingga dapat memangkas pohon pencarian lebih awal. Program pada permainan Futoshiki akan mengisi sel dengan jumlah solusi yang paling sedikit untuk diisi. Pada aturan LCV, program memilih nilai *domain* yang paling sedikit membatasi *domain* variabel lain untuk variabel (sel) yang dipilih. Program pada permainan Futoshiki akan mengisi sel dengan memilih nilai yang tidak dibatasi sel lain pada baris dan kolom yang sama.

Algoritma kemudian menggunakan langkah rekursif untuk menempatkan nilai yang dipilih ke variabel. Fungsi rekursif ini mengembalikan *boolean* True jika solusi ditemukan dan False jika tidak ditemukan sebuah solusi. Apabila solusi gagal ditemukan, algoritma akan melakukan *backtrack* dengan menghapus nilai yang ditempatkan dari variabel (sel) dan mengembalikan *domain* variabel lain ke keadaan sebelum nilai ditempatkan. Program kemudian melanjutkan algoritma dengan mencoba nilai berikutnya dari *domain* variabel yang sama. Program akan mengembalikan nilai True jika menemukan seluruh solusi dan nilai False jika semua nilai dari *domain* variabel telah dicoba dan tidak ada yang menghasilkan solusi. Setelah semua sel terisi dan kendala terpenuhi, solusi ditemukan.

C. Backtracking menggunakan Constraint Propagation

Pada teknik kedua, program algoritma *backtracking* menggunakan *constraint propagation* untuk memberi efisiensi. Untuk membandingkan kedua teknik, program dilengkapi dengan perhitungan durasi pencarian solusi dalam milisekon (ms) dan jumlah simpul/node yang dikunjungi (jumlah berapa kali nilai dimasukkan/diubah ke dalam semua variabel/sel yang ada pada papan).

Perbedaan program terletak pada proses rekusi. Setelah program menempatkan suatu nilai pada sebuah variabel, program akan langsung memperbaharui *domain* variabel lain supaya nilai yang dipakai pada variabel tersebut tidak lagi dijadikan sebagai opsi nilai yang dapat digunakan untuk mengisi variabel. Misal, menempatkan nilai "X" pada sel (b, k), maka program akan menghapus nilai "X" dari *domain* semua sel di baris 'b' dan kolom 'k'. Progam kemudian memeriksa kendala ketaksamaan yang ada pada baris dan kolom tersebut. Jika nilai yang baru ditempatkan melanggar kendala, maka program menghapus nilai-nilai yang

kontradiktif dan menyebabkan konflik. Program akan memangil fungsi rekursif untuk mengisi variabel selanjutnya jika berhasil dan *backtrack* jika gagal.

D. Solusi Akhir

Untuk memastikan kebenaran program, selain memeriksa pemenuhan kendala secara manual, hasil program dapat dibandingkan dengan kunci jawaban salah satu soal pada situs web – memastikan program berjalan dengan benar. Meski jawaban yang dimiliki berbeda antara situs web dengan program komputer, kendala yang ada masih terpenuhi. Pada teka-teki tersebut seperti pada gambar di bawah menunjukkan bahwa teka-teki ini dapat memiliki lebih dari satu solusi, dan dapat menggunakan beberapa langkah dalam algoritma yang berbeda seperti penggunaan aturan heuristik. Namun, pembahasan dan penelitian dalam lingkup tersebut tidak dibahas dalam makalah ini.

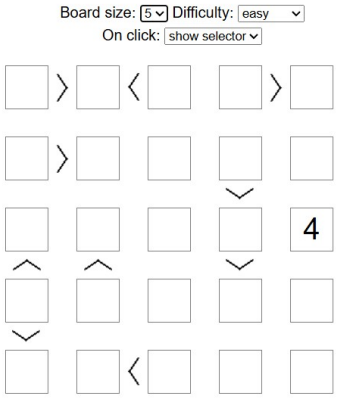

Fig. 3. Kondisi Awal Papan Futoshiki untuk Persoalan 9 x 9 dari situs web (*Sumber: futoshiki.com*)

Fig. 4. Kondisi Akhir Solusi Papan Futoshiki untuk Persoalan 9 x 9 dari situs web (*Sumber: futoshiki.com)*

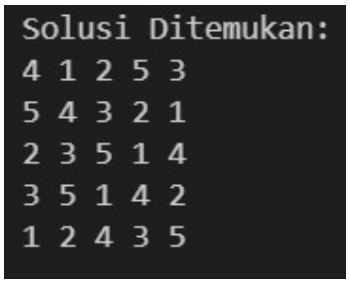

Fig. 5. Kondisi Akhir Solusi Papan Futoshiki untuk Persoalan 5 x 5 dari program komputer (*Sumber: dokumentasi pribadi*)

Program pada akhirnya memberikan dua *output* akhir sebagai solusi dari permainan Futoshiki, yaitu hasil proses pencarian solusi menggunakan algoritma runut-balik tanpa penggunaan constraint propagation, dan dengan menggunakan *constraint propagation*. Efisiensi penggunaan *constraint propagation* dalam menjawab permasalahan dibuktikan dengan memperlihatkan dan membandingkan waktu eksekusi pencarian solusi dan jumlah simpul yang dikunjungi oleh setiap proses. Perbandingan ini dapat terlihat secara jelas melalui tabel berikut untuk papan Futoshiki berukuran 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, dan 9x9.

TABLE I. DATA JUMLAH SIMPUL YANG DIKUNJUNGI DAN WAKTU EKSEKUSI DARI IMPLEMENTASI PROGRAM

Soal Jumlah Grid Papan (N x N)	***Proses* Backtracking**			
	Dengan Constraint Propagation		***Tanpa Constraint Propagation***	
	***Waktu Eksekusi* (detik)**	***Jumlah Simpul***	***Waktu Eksekusi* (detik)**	***Jumlah Simpul***
3 x 3	0.000169	9	0.000103	9
4 x 4	0.001332	17	0.000315	22
5 x 5	0.002456	49	0.001201	42
5 x 5 (dari web)	0.049971	991	0.106734	2873
6 x 6	0.002463	37	0.004344	77
7 x 7	0.007203	57	0.036851	542
8 x 8	0.006822	66	0.019400	156
9 x 9	0.129608	1531	0.400668	4047
9 x 9 (dari web)	0.175773	587	3.032490	11565

Fig. 6. Tabel Hasil Implementasi Algoritma pada Program Komputer

Dari tabel tersebut, tampak bahwa perbedaan kedua proses tidak terlalu mencolok ketika ukuran papan Futoshiki masih tergolong kecil. Bahkan, setelah beberapa percobaan, proses tanpa menggunakan *constraint propagation* memberikan *output* jumlah node yang dikunjugi dan waktu eksekusi pencarian solusi yang lebih baik ketimbang proses yang menggunakan *constraint propagation*. Namun, dari pencarian solusi untuk ukuran papan Futoshiki yang besar (dari 7 hingga ke 9), proses yang menggunakan *constraint propagation* jauh lebih efektif dalam menyelesaikan solusi. *Gap* antara kedua proses juga semakin terlihat seiring papan Futoshiki bertambah

semakin besar. Oleh karena itu, hasil program ini menunjukkan bahwa penggunaan *constraint propagation* pada algoritma *backtracking* sangat membantu program untuk mengurangi *resource* yang digunakan untuk mencari solusi, terutama untuk permasalahan yang lebih kompleks seperti papan Futoshiki yang tergolong besar. Namun, setelah beberapa percobaan lagi untuk menyelesaikan persoalan papan dengan *grid* 9 x 9 yang lebih sulit dan kompleks, program masih mengambil terlalu lama waktu untuk mencari solusi yang tepat untuk menjawab persoalan tersebut sehingga menghasilkan lebih dari 3 juta simpul yang dikunjungi.

Efek utama dari tidak adanya pemakaian *constraint propagation* pada program adalah penurunan drastik dalam efisiensi pencarian solusi. Dari jumlah simpul yang lebih besar, kita mengetahui bahwa ruang pencarian solusinya menjadi jauh leih besar. Tanpa *constraint propagation* yang secara aktif mengurangi kemungkinan nilai-nilai yang dipakai oleh sel lain, *domain* setiap sel kosong akan tetap berisi nilai 1 hingga N sampai semua nilai benar-benar dicoba di sel tersebut. Algoritma akhirnya mencoba banyak kombinasi nilai yang akhirnya gagal di akhir – meski bisa dideteksi lebih awal.

Proses ini juga memungkinkan deteksi konflik yang terlambat di mana penempatan nilai pada *domain* variabel memungkinkan *domain* variabel lain menjadi kosong sehingga melanggar kendala baris-kolom dan menjadikan solusi tidak sah/tidak benar. Dengan demikian, konflik semacam itu baru akan terdeteksi ketika giliran sel yang *domainnya* kosong tersebut dicoba untuk ditempatkan sebuah nilai. Algoritma menjadi menghabiskan banyak waktu dalam mendalami cabang pohon pencarian yang sudah tidak *valid* sejah awal, memperbanyak pula jumlah operasi *backtracking*. Hal ini juga berdampak pada waktu eksekusi pencarian solusi yang lebih lama, dan berpotensi menghampat penyelesaian permainan Futoshiki dengan kondisi papan yang jauh lebih kompleks.

E. Kompleksitas Algoritma Backtracking

Secara umum, pencarian solusi untuk permainan Futoshiki adalah sebuah masalah *NP-Complete*, yakni permasalahan yang hingga saat ini belum ditemukan algoritma yang dapat memecahkan masalah dalam waktu polinomial. Dalam kasus terburuk (*worst-case*), kompleksitasnya berupa eksponensial, bukan polinomial. Di sini proses dengan *constraint propagation* dianggap tidak berpengaruh terlalu besar terhadap kompleksitas waktu dan ruang karena dalam kasus terburuk, kedua proses masih melakukan pencarian yang sama terhadap setiap kombinasi yang ada.

Secara teoritis dalam menghitung kompleksitas waktu, dalam kasus terburuk, algortima *backtracking* bisa mencoba hampir setiap kombinasi nilai untuk setiap sel pada papan N^2. Untuk setiap sel, ada N pilihan nilai. Tanpa pemangkasan efektif melalui propagansi kendala, program dapat memiliki N pilihan untuk mengisi sel pertama, N pilihan untuk sel kedua, dan seterusnya hingga sel ke- N^2. Ketika dijumlahkan, akan menghasilkan N^{N^2} kemungkinan penugasan.

Lalu, untuk setiap penugasan, kita perlu memvalidasi seluruhnya yang melibatkan iterasi melalui baris, kolom (O(N)), dan semua kendala. Jika jumlah kendala adalah C, maka validasi memiliki kompleksitas waktu O(N+C). Kompleksitas akhirnya mendekati $O(D^{N^2} \times (N+C))$, yang disederhanakan menjadi eksponensial ganda atau eksponensial dengan basis N dan eksponen N^2. Hal ini menunjukkan bahwa tanpa propagansi kendala, program dapat masuk sangat dalam dalam cabang pohon pencarian yang tidak *valid*. Bahkan dengan propagasi, program masih perlu melakukan banyak *backtracking*. Propagasi kendala hanya mengurangi faktor basis dari eksponen, tetapi tidak menghilangkannya.

Dalam menghitung ruang kompleksitasnya, secara teoritis, algoritma membutuhkan ruang papan sebesar $O(N^2)$ untuk menyimpan nilai-nilai sel/variabel dengan *domain* pada setiap sel berisi 1 hingga N nilai. Oleh karena terdapat N^2 sel, maka kompleksitas ruangnya menjadi $O(N^2 \times N) = O(N^3)$ apabila dilihat secara naif. Namun, dalam implementasi ini, program hanya menyimpan himpunan solusi untuk sel yang belum terisi dengan himpunan solusi tersebut berisi angka 1 hingga N. Umumnya, dilihat sebagai $O(N^2)$ karena nilai *domain* itu sendiri kecil. Secara keseluruhan, kompleksitas ruangnya adalah $O(N^2)$.

IV. KESIMPULAN, SARAN, DAN REFLEKSI

A. Kesimpulan

Algoritma runut-balik (*backtracking*) dapat digunakan untuk menyelesaikan permainan Futoshiki, baik dengan maupun tanpa *constraint propagation*, telah berhasil dilakukan. Dari pengujian yang dilakukan pada berbagai ukuran papan, penulis mendapatkan kesimpulan sebagai berikut:

1. Algoritma *backtracking* mampu menemukan solusi untuk permainan futoshiki, mengonfirmasi sebagai metode pencarian paling umum dalam menyelesaikan *constraint satisfaction problems* (CSPs)
2. Penerapan *constraint propagation* terbukti meningkatkan efisiensi algoritma *backtracking* pada permainan futoshiki, terutama untuk papan berukuran sedang hingga besar. Teknik ini efisien dalam mempercepat proses dan waktu eksekusi dan mengurangi jumlah simpul yang dikunjungi.
3. Penggunaan aturan heuristik MRV dan LCV secara konsisten membantu pencarian pada kedua proses *backtracking* murni dan *constraint propagation*. Kedua aturan bersinergi dengan penggunaan *constraint propagation* yang mengatur soal variabel dan nilai pada *domain* atau himpunan solusi.

B. Saran

Dari implementasi yang telah dilakukan pada program untuk menguji algoritma dengan *constraint propagation*, penulis mendapatkan beberapa hal yang bisa ditingkatkan pada implementasi, eksplorasi, atau penelitian lebih lanjut ke depannya, yaitu:

1. Program komputer dan penelitian perlu mengeksplorasi pengaruh jumlah dan pilihan heuristik yang bisa digunakan pada proses *backtracking* sehingga dapat mengurangi jumlah *backtrack* lebih lanjut dan mempercepat proses pencarian solusi,

terutama untuk persoalan papan Futoshiki yang lebih kompleks

2. Penggunaan *constraint propagation* yang bisa lebih dieksplorasi seperti penggunaan *Arc Consistency* untuk meningkatkan kinerja program komputer untuk pencarian solusi
3. Eksplorasi data dan perbandingan pengujian yang lebih komprehensif dapat dilanjutkan untuk setiap jenis persoalan grid papan futoshiki N x N.

C. *Refleksi*

Makalah ini telah memberikan penulis pengetahuan dan pemahaman mandala tentang bagaimana *constraint propagation* digunakan untuk mempermudah pencarian solusi sebuah permasalahan, dan menjadi salah satu bahan ajaran dalam konteks kecerdasan buatan. Penulis mendapatkan pengalaman, meski bukan mahasiswa jurusan Teknik Informatika, untuk memiliki fondasi dan logika yang kuat, serta keterampilan yang cukup untuk menentukan strategi algoritma dalam menghadapi masalah dan menggunakan ilmu informatika. Makalah ini mengajarkan bahwa ketika mencari sebuah solusi dari suatu persoalan, efisiensi perlu dipikirkan untuk memberikan hasil yang terbaik.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih sebesar-besarnya kepada Tuhan Yang Maha Esa atas Rahmat dan berkatnya sehingga penulis dapat menyelesaikan makalah ini dengan lancar dan tepat waktu. Penulis juga sangat berterima kasih kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma 2025 kelas 01, karena berkat ilmu dan pengalaman yang beliau ajarkan pada setiap sesi kelas sehingga penulis dapat memahami dan mengimplementasikan materi mata kuliah ini. Penullis juga sangat berterimakasih kepada orangtua penulis dan rekan-rekan penulis yang telah memberikan semangat, bantuan, dan dukungan kepada penulis. Penulis berharap persoalan yang dibahas dalam makalah ini dapat dibahas lebih lanjut sehingga ditemukan solusi, analisis, atau eksplorasi lain yang lebih baik.

REFERENSI

[1] G BM. (n.d.). Optimizer: Constraint Propagation. Diakses dari https://www.ibm.com/docs/en/icos/22.1.2?topic=optimizer-constraint-propagation. (Diakses pada 23 Juni 2025).

[2] Munir, R. (2025). Algoritma Runut-balik (backtracking) Bahan Kuliah IF2211 Strategi Algoritma (Bagian 1). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian1.pdf. (Diakses pada 22 Juni 2025).

[3] Munir, R. (2025). Algoritma Runut-balik (backtracking) Bahan Kuliah IF2211 Strategi Algoritma (Bagian 2). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/15-Algoritma-backtracking-(2025)-Bagian2.pdf. (Diakses pada 22 Juni 2025).

[4] Savigny, J. (2015). Penerapan Algoritma Backtracking dalam Permainan Futoshiki Puzzle. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_082.pdf (Diakses pada 23 Juni 2025).

[5] The Futoshiki. (n.d.). Futoshiki.com. Diakses dari https://www.futoshiki.com/ (Diakses pada 23 Juni 2025).

[6] University of California, Berkeley. (n.d.). *CS188: Introduction to Artificial Intelligence, Constraint Satisfaction Problems*. Diakses dari https://www.cs.cmu.edu/~15281/coursenotes/constraints/index.html (Diakses pada 23 Juni 2025).

[7] University of Cornell. (2011). *CS4700: Artificial Intelligence, Lecture 5: Constraint Satisfaction Problems*. Diakses dari https://www.cs.cornell.edu/courses/cs4700/2011fa/lectures/05_CSP.pdf (Diakses pada 23 Juni 2025).

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

Ignacio Kevin Alberiann, 15223090