SKRIPSI

PERBANDINGAN ALGORITMA BACKTRACKING DENGAN ALGORITMA HYBRID GENETIC UNTUK MENYELESAIKAN PERMAINAN CALCUDOKU

MICHAEL ADRIAN

NPM: 2013730039

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN

«tahun»

UNDERGRADUATE THESIS

COMPARISON OF THE BACKTRACKING ALGORITHM AND THE HYBRID GENETIC ALGORITHM TO SOLVE THE CALCUDOKU PUZZLE

MICHAEL ADRIAN

NPM: 2013730039

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY

«tahun»

LEMBAR PENGESAHAN

PERBANDINGAN ALGORITMA BACKTRACKING DENGAN ALGORITMA HYBRID GENETIC UNTUK MENYELESAIKAN PERMAINAN CALCUDOKU

MICHAEL ADRIAN

NPM: 2013730039

Bandung, «tanggal» «bulan» «tahun»

Menyetujui,

Pembimbing Utama	Pembimbing Pendamping
Dr.rer.nat. Cecilia Esti Nugraheni	«pembimbing pendamping/2»

Ketua Tim Penguji	Anggota Tim Penguji
«penguji 1»	«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PERBANDINGAN ALGORITMA BACKTRACKING DENGAN ALGORITMA HYBRID GENETIC UNTUK MENYELESAIKAN PERMAINAN CALCUDOKU

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal «tanggal» «bulan» «tahun»

Meterai
Rp. 6000

Michael Adrian
NPM: 2013730039

ABSTRAK

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius. Claritas est etiam processus dynamicus, qui sequitur mutationem consuetudium lectorum. Mirum est notare quam littera gothica, quam nunc putamus parum claram, anteposuerit litterarum formas humanitatis per seacula quarta decima et quinta decima. Eodem modo typi, qui nunc nobis videntur parum clari, fiant sollemnes in futurum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Kata-kata kunci: lorem, ipsum

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius. Claritas est etiam processus dynamicus, qui sequitur mutationem consuetudium lectorum. Mirum est notare quam littera gothica, quam nunc putamus parum claram, anteposuerit litterarum formas humanitatis per seacula quarta decima et quinta decima. Eodem modo typi, qui nunc nobis videntur parum clari, fiant sollemnes in futurum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Keywords: lorem, ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

KATA PENGANTAR

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius. Claritas est etiam processus dynamicus, qui sequitur mutationem consuetudium lectorum. Mirum est notare quam littera gothica, quam nunc putamus parum claram, anteposuerit litterarum formas humanitatis per seacula quarta decima et quinta decima. Eodem modo typi, qui nunc nobis videntur parum clari, fiant sollemnes in futurum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bandung, «bulan» «tahun»

Penulis

DAFTAR ISI

Kata Pengantar **xv**

Daftar Isi **xvii**

Daftar Gambar **xviii**

Daftar Tabel **xx**

1 Pendahuluan **1**
- 1.1 Latar Belakang . . . 1
- 1.2 Rumusan Masalah . . . 2
- 1.3 Tujuan . . . 3
- 1.4 Batasan Masalah . . . 3
- 1.5 Metodologi Penelitian . . . 3
- 1.6 Sistematika Pembahasan . . . 4

2 Landasan Teori **5**
- 2.1 Calcudoku [1] [2] . . . 5
- 2.2 Algoritma *Backtracking* [1] . . . 7
- 2.3 Algoritma *Hybrid Genetic* [2] . . . 14
  - 2.3.1 Algoritma *Rule Based* . . . 14
  - 2.3.2 Algoritma Genetik . . . 15
  - 2.3.3 Algoritma *Hybrid Genetic* . . . 16

3 Analisis **21**
- 3.1 Algoritma *Backtracking* . . . 21
- 3.2 Algoritma *Hybrid Genetic* . . . 35

Daftar Referensi **37**

DAFTAR GAMBAR

1.1 Contoh permainan teka-teki Calcudoku dengan penjelasan tentang elemen-elemen dari teka-teki ini [2] . . . 2

2.1 Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 4 x 4 yang belum diselesaikan. [1] . . . 6
2.2 Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 2.1 [1] . . . 7
2.3 Ilustrasi *State space tree* yang digunakan dalam algoritma *backtracking* [1] . . . 9
2.4 Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 3 x 3 [1] . . . 10
2.5 Ilustrasi *state* 3, 4, dan 5 pada sebuah *grid* teka-teki Calcudoku [1] . . . 11
2.6 Ilustrasi *state* 19 pada sebuah *grid* teka-teki Calcudoku [1] . . . 12
2.7 *State* 25, simpul tujuan, sebagai hasil yang dicapai [1] . . . 13
2.8 *state space tree* yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 2.4 [1] . . . 13
2.9 Contoh bagaimana cara mendeteksi aturan *naked pair* [2] . . . 14
2.10 Contoh aturan *evil twin* [2] . . . 15
2.11 Contoh aturan *hidden single* [2] . . . 15
2.12 Contoh aturan *killer combination* untuk *cage* dengan ukuran 2 sel [2] . . . 15
2.13 Contoh aturan *X-wing* [2] . . . 16
2.14 Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 6 x 6 [2] . . . 17
2.15 Contoh proses kawin silang antara dua kromosom [2] . . . 18
2.16 Contoh proses mutasi [2] . . . 18
2.17 Alur penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* [2] . . . 19

3.1 Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 4 x 4 yang belum diselesaikan, seperti yang digambarkan pada Gambar 2.1. [1] . . . 21
3.2 *State* 4 . . . 22
3.3 *State* 11 . . . 22
3.4 *State* 12 . . . 22
3.5 *State* 17 . . . 23
3.6 *State* 18 . . . 23
3.7 *State* 19 . . . 23
3.8 *State* 23 . . . 24
3.9 *State* 24 . . . 24
3.10 *State* 31 . . . 25
3.11 *State* 32 . . . 25
3.12 *State* 34 . . . 25
3.13 *State* 37 . . . 26
3.14 *State* 47 . . . 26
3.15 *State* 48 . . . 27
3.16 *State* 52 . . . 27
3.17 *State* 53 . . . 27
3.18 *State* 68 . . . 28

3.19 *State* 69 29
3.20 *State* 71 29
3.21 *State* 72 29
3.22 *State* 74 30
3.23 *State* 75 30
3.24 *State* 76 30
3.25 *State* 77 31
3.26 *State* 78 31
3.27 *State* 81 31
3.28 *State* 83 32
3.29 *State* 85 32
3.30 *State* 88 32
3.31 *State* 92 33
3.32 *State* 93 33
3.33 *State space tree* yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.1 34
3.34 Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 6 x 6 yang belum diselesaikan, seperti yang digambarkan pada Gambar 1.1. [2] 35
3.35 Permainan teka-teki Calcudoku setelah diselesaikan dengan algoritma *rule based* . . 36

DAFTAR TABEL

BAB 1

PENDAHULUAN

Bab ini membahas tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan dari skripsi ini.

1.1 Latar Belakang

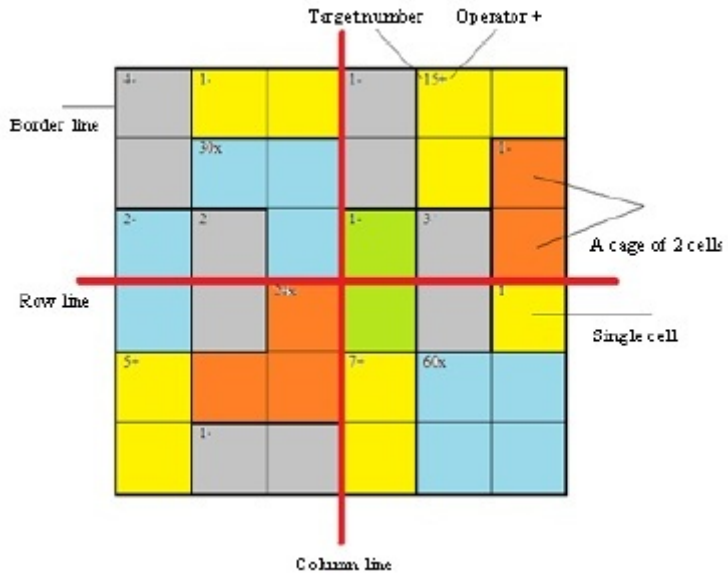
Calcudoku, atau dikenal juga sebagai KenKen, atau Mathdoku, adalah sebuah permainan teka-teki (*puzzle*) angka yang untuk menyelesaikannya memerlukan perpaduan dari logika dan kemampuan aritmatika yang sederhana. Permainan ini adalah sebuah permainan teka-teki logika yang sederhana, namun, untuk menemukan solusinya cukup rumit, terutama untuk masalah yang lebih susah.

Teka-teki ini mirip dengan Sudoku. Persamaannya, tujuan dari teka-teki ini adalah mengisi setiap sel (*cell*) dalam (*grid*) dengan angka 1 sampai n tanpa pengulangan angka dalam setiap kolomnya dan barisnya untuk *grid* berukuran $n \times n$, dengan n adalah ukuran *grid*. Tidak ada angka yang boleh muncul lebih dari sekali dalam setiap baris atau kolom dalam *grid*. Perbedaannya, jika pada Sudoku *grid* berukuran $n \times n$ dibagi menjadi n (*cage*) dengan setiap *cage* terdiri atas n sel, pada Calcudoku *grid* dibagi menjadi sejumlah *cage* yang jumlah selnya bervariasi. Setiap *cage* dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel. Angka-angka dalam satu *cage* yang sama harus menghasilkan angka tujuan yang telah ditentukan jika dihitung menggunakan operasi matematika yang telah ditentukan (penjumlahan, pengurangan, perkalian, atau pembagian). Angka-angka dalam satu *cage* juga boleh berulang, selama pengulangan tidak terjadi dalam satu kolom atau baris yang sama. Jika *cage* hanya berisi satu sel, maka satu-satunya kemungkinan jawaban untuk sel tersebut adalah angka tujuan dari *cage* tersebut. Angka tujuan dan operasi matematika dituliskan di sudut kiri atas *cage*. Pada awalnya, setiap sel dalam setiap *cage* dalam teka-teki ini kosong. *Border line* adalah garis pembatas terluar, *row line* adalah garis pembatas antar baris, dan *column line* adalah garis pembatas antar kolom. Gambar 1.1 menggambarkan contoh sebuah permainan teka-teki Calcudoku. [1] citejohanna:12:hybrid

Calcudoku dapat diselesaikan menggunakan beberapa algoritma. Skripsi ini membahas tentang penyelesaian Calcudoku menggunakan algoritma *backtracking* dan algoritma *hybrid genetic*, dan perbandingan performansi *performance* antara kedua algoritma tersebut.

Algoritma *backtracking* adalah sebuah algoritma umum yang mencari solusi dengan mencoba salah satu dari beberapa pilihan, jika pilihan yang dipilih ternyata salah, komputasi dimulai lagi pada titik pilihan dan mencoba pilihan lainnya. Untuk bisa melacak kembali langkah-langkah yang telah dipilih, maka algoritma harus secara eksplisit menyimpan jejak dari setiap langkah yang sudah pernah dipilih, atau menggunakan rekursi (*recursion*). Rekursi dipilih karena jauh lebih mudah daripada harus menyimpan jejak setiap langkah yang pernah dipilih, hal ini menyebabkan algoritma ini hampir selalu berbasis DFS (*Depth First Search*). [1]

Algoritma *rule based* adalah sebuah algoritma berbasis aturan logika untuk menyelesaikan permainan teka-teki Sudoku dan variasinya, termasuk Calcudoku. Beberapa aturan logika yang digunakan dalam algoritma ini adalah *single square rule*, *naked subset rule*, *hidden single rule*, *evil twin rule*, *killer combination*, dan *X-wing*.



Gambar 1.1: Contoh permainan teka-teki Calcudoku dengan penjelasan elemen-elemen dari teka-teki ini [2]

Pencarian heuristik adalah sebuah teknik pencarian kecerdasan buatan (*artifical intelligence*) yang menggunakan heuristik dalam langkah-langkahnya. Heuristik adalah semacam aturan tidak tertulis yang mungkin menghasilkan solusi. Heuristik kadang-kadang efektif, tetapi tidak dijamin akan berhasil. dalam setiap kasus. Heuristik memerankan peran penting dalam strategi pencarian karena sifat eksponensial dari kebanyakan masalah. Heuristik membatu mengurangi jumlah alternatif solusi dari angka yang bersifat eksponensial menjadi angka yang bersifat polinomial. Contoh teknik pencarian heuristik adalah *Generate and Test*, *Hill Climbing*, dan *Best First Search*.

Algoritma genetik adalah salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam. Algoritma ini adalah perpaduan dari bidang biologi dan ilmu komputer. Algoritma ini adalah salah satu dari teknik pencarian heuristik.

Algoritma ini memanipulasi informasi, biasanya disebut sebagai kromosom. Kromosom ini meng-*encode* kemungkinan jawaban untuk sebuah masalah yang diberikan. Kromosom dievaluasi dan diberi *fitness value* berdasarkan seberapa baikkah kromosom dalam menyelesaikan maslah yang diberikan berdasarkan kriteria yang ditentukan oleh pembuat program. Nilai kelayakan ini digunakan sebagai probabilitas kebertahanan hidup kromosom dalam satu siklus reproduksi. Kromosom baru (kromosom anak, *child chromosome*) diproduksi dengan menggabungkan dua (atau lebih) kromosom orang tua (*parent chromosome*). Proses ini dirancang untuk menghasilkan kromosom-kromosom keturunan yang lebih layak, kromosom-kromosom ini menyandikan jawaban yang lebih baik, sampai solusi yang baik dan yang bisa diterima ditemukan.

Algoritma *hybrid genetic* adalah gabungan antara algoritma genetik dan algoritma-algoritma lainnya. Dalam kasus ini, algoritma genetik digabungkan dengan algoritma *rule based*. Algoritma *rule based* akan dijalankan sampai pada titik dimana algoritma tidak bisa menyelesaikan permainan teka-teki Calcudoku. Jika algoritma sudah tidak bisa menyelesaikan permainan, maka algoritma genetik akan mulai dijalankan. [2]

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan di atas, dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana cara mengimplementasikan perangkat lunak (*software*) permainan teka-teki Cal-

cudoku?

2. Bagaimana cara mengimplementasikan algoritma *backtracking* untuk menyelesaikan Calcudoku?
3. Bagaimana cara mengimplementasikan algoritma *hybrid genetic* untuk menyelesaikan Calcudoku?
4. Bagaimana perbandingan performansi algoritma *backtracking* dengan algoritma *hybrid genetic* dalam menyelesaikan Calcudoku?

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dirumuskan, maka tujuan dari pembuatan skripsi ini adalah:

1. Membuat perangkat lunak permainan teka-teki Calcudoku.
2. Membuat *solver* untuk Calcudoku menggunakan algoritma *backtracking*.
3. Membuat *solver* untuk Calcudoku menggunakan algoritma *hybrid genetic*.
4. Membandingkan performansi algoritma *backtracking* dengan algoritma *hybrid genetic* dalam menyelesaikan Calcudoku.

1.4 Batasan Masalah

Ruang lingkup dari skripsi ini dibatasi oleh batasan-batasan masalah sebagai berikut:

1. Ukuran *grid* untuk permainan teka-teki Calcudoku adalah antara 3×3 sampai dengan 9×9.

1.5 Metodologi Penelitian

Langkah-langkah yang akan dilakukan dalam pembuatan skripsi ini adalah:

1. Studi literatur
   (a) Melakukan studi literatur tentang permainan teka-teki Calcudoku.
   (b) Melakukan studi literatur tentang algoritma *backtracking*.
   (c) Melakukan studi literatur tentang algoritma *rule based* dan algoritma genetik.
2. Analisis, perancangan, dan pengembangan perangkat lunak
   (a) Melakukan analisis dan menentukan fitur-fitur yang diperlukan dalam perangkat lunak permainan teka-teki Calcudoku.
   (b) Membuat perangkat lunak Calcudoku dengan fitur-fitur yang telah ditentukan.
   (c) Mengimplementasikan algoritma *backtracking* untuk Calcudoku.
   (d) Mengimplementasikan algoritma *hybrid genetic* untuk Calcudoku.
   (e) Membandingkan performansi algoritma *backtracking* dengan algoritma *hybrid genetic* dalam menyelesaikan Calcudoku.
3. Melakukan pengujian terhadap perangkat lunak Calcudoku yang telah dibuat.
4. Membuat kesimpulan berdasarkan hasil pengujian perangkat lunak yang telah dibuat.

1.6 Sistematika Pembahasan

Sistematika pembahasan skripsi ini adalah sebagai berikut:

1. Bab 1 membahas tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan dari skripsi ini.
2. Bab 2 membahas tentang landasan teori yang digunakan dalam skripsi ini, yaitu tentang permainan teka-teki Calcudoku, algoritma *backtracking* dan algoritma *hybrid genetic*.
3. Bab 3 membahas tentang analisis perangkat lunak Calcudoku dan analisis algoritma *backtracking* dan algoritma *hybrid genetic*.
4. Bab 4 membahas tentang perancangan dan pembuatan perangkat lunak Calcudoku dan algoritma *backtracking* dan algoritma *hybrid genetic* untuk menyelesaikan permainan, perancangan antarmuka (*interface*), input dan output, diagram kelas (*class diagram*), dan diagram aktivitas (*activity diagram*).
5. Bab 5 membahas tentang implementasi dari perangkat lunak Calcudoku dan algoritma *backtracking* dan algoritma *hybrid genetic* yang telah dirancang, implementasi antarmuka, input dan output yang telah dirancang, dan pengujian perangkat lunak Calcudoku dalam hal perbandingan performansi algoritma *backtracking* dan algoritma *hybrid genetic* dalam menyelesaikan permainan.
6. Bab 6 membahas tentang kesimpulan dari pembuatan perangkat lunak Calcudoku dan hasil pengujiannya, dan saran untuk penelitian pengembangan perangkat lunak selanjutnya.

BAB 2

LANDASAN TEORI

Bab ini membahas tentang landasan teori yang akan digunakan dalam skripsi ini yang diambil dari dua sumber, yaitu "KenKen Puzzle Solver using Backtracking Algorithm" karya Asanilta Fahda [1] dan "Solving and Modeling Ken-ken Puzzle by Using Hybrid Genetics Algorithm" karya Olivia Johanna, Samuel Lukas, dan Kie Van Ivanky Saputra [2].

2.1 Calcudoku [1] [2]

Sebagai salah satu jenis permainan teka-teki aritmatika dan *grid*, Calcudoku, atau dikenal juga sebagai KenKen, KenDoku, atau Mathdoku, diciptakan pada tahun 2004 oleh seorang guru matematika dari Jepang yang bernama Tetsuya Miyamoto untuk memenuhi tujuannya untuk melatih kemampuan matematika dan logika siswa-siswinya dengan cara yang menyenangkan. Nama KenKen diambil dari kata bahasa Jepang yang berarti kepandaian. Permainan yang mengasah otak ini dengan cepat menyebar ke seluruh Jepang dan Amerika Serikat, menggantikan permainan teka-teki silang di banyak koran. Permainan ini kemudian menjadi sensasi di seluruh dunia setelah munculnya versi *online* dan *mobile* dari permainan teka-teki ini, khususnya menarik untuk pecinta permainan teka-teki angka seperti Sudoku.

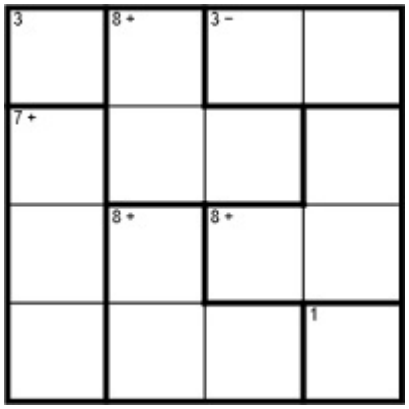
Seperti dalam Sudoku, dalam teka-teki ini, pemain diberikan sebuah *grid* dengan ukuran $n \times n$, dengan n biasanya antara 3 sampai dengan 9. *Grid* ini harus diisi dengan angka 1 sampai dengan n sehingga dalam setiap baris setiap angka hanya muncul sekali, dalam setiap kolom setiap angka hanya muncul sekali. Perbedaannya dengan Sudoku adalah, Calcudoku dibagi ke dalam *cage* (sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel dengan angka tujuan dan operator yang telah ditentukan), dan angka-angka dalam setiap *cage* harus mencapai angka tujuan jika dihitung menggunakan operator yang telah ditentukan. Angka tujuan dan operasi yang telah ditentukan ditulis di sudut kiri atas *cage*. Ada lima kemungkinan operator:

1. $+$, sebuah operator n-ary yang menandakan penjumlahan.
2. -, sebuah operator biner yang menandakan pengurangan.
3. $\times$, sebuah operator n-ary yang menandakan perkalian.
4. $\div$ sebuah operator biner yang menandakan pembagian.
5. $=$, (simbol ini biasanya dihilangkan), sebuah operator uner yang menandakan persamaan.

Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuruan *cage* harus berukuran dua sel. Pada beberapa versi dari teka-teki ini, hanya angka tujuan yang diberikan, dan pemain harus menebak operator dari setiap *cage* untuk menyelesaikan teka-tekinya [1] [2].

Untuk menyelesaikan sebuah teka-teki Calcudoku, pemain harus pertama-tama memahami dua permasalahan utama dari teka-teki ini, yaitu:

1. Angka-angka mana yang harus dimasukkan ke dalam sebuah *cage*



Gambar 2.1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan. [1]

2. Dalam urutan apa angka-angka tersebut harus dimasukkan ke dalam sebuah *cage*

Seperti kebanyakan permainan teka-teki angka, cara yang paling mudah untuk menyelesaikan teka-teki ini adalah dengan mengeliminasi angka-angka yang sudah digunakan dan mencoba satu per satu angka yang mungkin (*trial and error*).

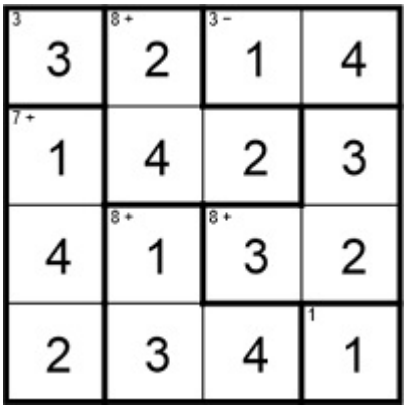
Dalam pengisian teka-teki ini ada dua tahapan, yaitu:

1. Mencari *cage* yang hanya berukuran 1 sel, karena *cage* ini tidak menghasilkan pertanyaan angka apa dan urutan apa. Tahap ini adalah tahap yang paling jelas. Contoh, pada Gambar 2.1, *cage* pada sudut kiri atas dan *cage* pada sudut kanan bawah hanya berukuran 1 sel, dan dapat langsung diisi dengan angka tujuannya.

2. Mencari mencari *cage* yang hanya mempunyai satu kemungkinan kombinasi angka, sehingga masalah angka-angka apa yang harus diisi dalam *cage* tersebut terjawab. Contoh, *cage* pada sudut kanan atas mempunyai aturan "3-", artinya angka tujuannya adalah 3 dengan menggunakan operasi pengurangan. Satu-satunya pasangan angka dari himpunan {1,2,3,4} yang akan menghasilkan angka 3 saat satu angka dikurangkan dari angka yang lainnya adalah {1,4}. Namun masalahnya adalah urutan angka-angka yang harus dimasukkan. Dalam kasus ini, untungnya, sel pada sudut kanan bawah sudah diisi dengan angka 1, maka angka 1 tidak bisa digunakan lagi pada kolom yang paling kanan. Jadi, dengan menggunakan cara eliminasi, sel pada sudut kanan atas harus diisi dengan angka 4 dan sel di sebelah kirinya, yaitu sel pada baris yang paling atas dan kolom ketiga dari kiri, harus diisi dengan angka 1. Hal ini memberikan solusi untuk sel pada baris yang paling atas dan kolom kedua dari kiri, yaitu angka 2, karena angka 2 adalah angka yang belum pernah dipakai dalam baris tersebut. Proses ini berlanjut sampai semua sel dalam *grid* terisi dan menghasilkan solusi pada Gambar 2.2 [1].

Seiring dengan meningkatnya tingkat kesulitan, langkah berikutnya tidak akan langsung muncul dengan jelas. Kadang-kadang, pemain mencapai titik dimana langkah berikutnya tidak pasti. Pemain harus menebak langkah-langkah berikutnya dan melihat apakah langkah ini akan menghasilkan solusinya. Jika tidak, pemain harus mundur kembali ke titik ketidakpastian tersebut.

Sebuah teka-teki Calcudoku dengan ukuran $n \times n$, dengan n melambangkan jumlah sel dalam satu baris atau kolom, mempunyai n^2 sel. Sel yang terletak dalam baris b dan kolom k diberi label $C_{b,k} = bn + k$ dan nilai dari sel tersebut adalah $V(C_{b,k}) \in \{1, 2, ..., n\}$. Sebuah *cage*, yang diberi label A_i adalah sebuah himpunan dari sel, yaitu $A_i = \{C_{b,k}\}$. Setiap *cage* terhubung dengan satu operator aritmatika $O_i \in \{+, -, \times, \div\}$ dan satu angka tujuan $H_i \in N$. Menurut Johanna, Lukas, dan Saputra, tiga aturan dalam mendefinisikan masalah dalam Calcudoku adalah sebagai berikut [2]:



Gambar 2.2: Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 2.1. [1]

1. $|A_i| = 1 \rightarrow O_i = \phi$, artinya setiap *cage* yang jumlah selnya 1 dengan operasi matematika yang terkait dengan *cage* tersebut bersifat homeomorfik (setara).
2. $O_i \in -, \div \rightarrow |A_i| = 2$, artinya jika operasi yang digunakan dalam sebuah *cage* adalah pengurangan atau pembagian, maka jumlah sel dalam *cage* tersebut harus 2.
3. $\forall C_{b,k} \rightarrow C_{b,k} \in \exists! A_i$, artinya setiap sel hanya boleh menjadi anggota dari satu dan hanya satu *cage*.

Menurut Johanna, Lukas, dan Saputra, tujuan dari teka-teki ini adalah untuk mencari nilai $V(C_{b,k})$ dan memenuhi persyaratan berikut [2]:

1. $|A_i| = 1 \wedge C_{b,k} \in A_i \rightarrow V(C_{b,k}) = H_i$, artinya jika sel adalah bagian dari sebuah *cage* yang jumlah selnya 1, maka nilai dari sel tersebut adalah angka tujuan dari *cage* tersebut.
2. $O_i \in \{-\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow |V(C_{a,b}) - V(C_{p,q})| = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah pengurangan, maka nilai absolut dari hasil pengurangan nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
3. $O_i \in \{\div\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow V(C_{a,b})/V(C_{p,q}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah pembagian, maka nilai dari hasil pembagian nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
4. $O_i \in \{+\} \rightarrow \sum_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah penjumlahan, maka nilai dari hasil penjumlahan dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
5. $O_i \in \{\times\} \rightarrow \prod_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah perkalian, maka nilai dari hasil perkalian dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.

2.2 Algoritma *Backtracking* [1]

Algoritma *backtracking* adalah sebuah algoritma umum yang mencari solusi dengan mencoba salah satu dari beberapa pilihan, jika pilihan yang dipilih ternyata salah, komputasi dimulai lagi pada titik pilihan dan mencoba pilihan lainnya. Untuk bisa melacak kembali langkah-langkah yang telah dipilih, maka algoritma harus secara eksplisit menyimpan jejak dari setiap langkah yang sudah pernah dipilih, atau menggunakan rekursi (*recursion*). Rekursi dipilih karena jauh lebih mudah daripada harus menyimpan jejak setiap langkah yang pernah dipilih. Hal ini menyebabkan algoritma ini biasanya berbasis DFS (*Depth First Search*).

Algoritma *backtracking* pertama kali diperkenalkan pada tahun 1950 oleh D.H. Lehmer sebagai perbaikan algoritma *brute force*. Algoritma ini lalu dikembangkan lebih lanjut oleh R.J. Walker, S.W. Golomb, dan L.D. Baumert. Algoritma ini terbukti efektif untuk menyelesaikan banyak permainan logika (misalnya *tic tac toe*, *maze*, catur, dan lain-lain) karena algoritma itu terutama berguna untuk menyelesaikan masalah-masalah *constraint satisfaction*, di mana sekumpulan objek harus memenuhi sejumlah batasan.

Menurut Fahda, implementasi algoritma *backtracking* memiliki beberapa sifat umum, yaitu [1]:

1. *Solution space*
   Solusi untuk masalah ini dinyatakan sebagai sebuah vektor X dengan *n-tuple*:

   $$X = (x_1, x_2, ..., x_n), x_i \in S_i$$

   di mana adalah mungkin bahwa:

   $$S_1 = S_2 = ... = S_n$$

   n adalah jumlah sel dalam satu baris atau kolom. X adalah sebuah *tuple* yang berukuran n^2, yang mereprentasikan isi dari setiap sel dalam *grid*, dimulai pada sel pada sudut kiri atas, lalu bergerak ke sel di sebelah kanannya dalam baris yang sama, jika sudah mencapai sel yang paling kanan maka bergerak ke sel yang paling kiri pada baris dibawahnya, hingga berakhir di sel pada sudut kanan bawah. S_i adalah sebuah himpunan yang berisi angka-angka dari 1 sampai n.

2. Fungsi pembangkit X_k
   Fungsi pembangkit X_k dinyatakan sebagai:

   $$T(k)$$

   di mana $T(k)$ membangkitkan nilai X_k, dari 1 sampai n, yang merupakan komponen dari vektor solusi.

3. Fungsi pembatas
   Fungsi pembatas dinyatakan sebagai:

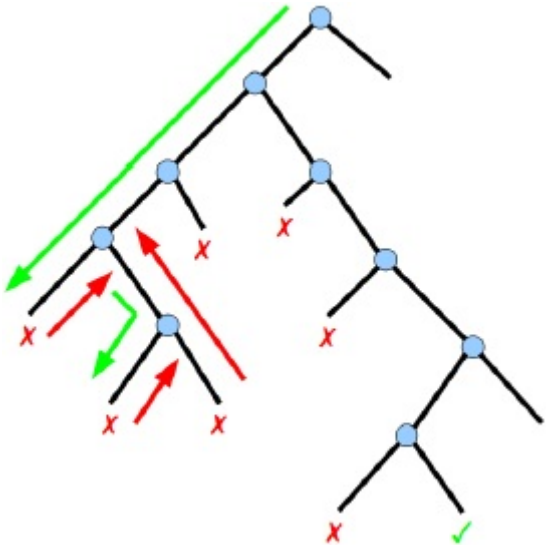
   $$B(x_1, x_2, ..., x_k)$$

   di mana B bernilai *true* jika $(x_1, x_2, ..., x_k)$ mengarah ke solusi. Jika B bernilai *true*, maka nilai $x_k + 1$ akan terus dibangkitkan, dan jika B bernilai *false*, maka $(x_1, x_2, ..., x_k)$ akan dibuang.

Ruang solusi untuk algoritma *backtracking* disusun dalam sebuah struktur berbentuk pohon (*tree*), di mana setiap simpul (*node*) merepresentasikan keadaan masalah dan sisi (*edge*) diberi label x_i. Jalur dari akar (*root*) ke daun (*leaf*) merepresentasikan sebuah jawaban yang mungkin, dan semua jalur yang dikumpulkan bersama-sama membentuk ruang solusi. Struktur pohon ini disebut sebagai *state space tree*. Gambar 2.3 menggambarkan contoh sebuah *state space tree*.

Langkah-langkah dalam menggunakan *state space tree* untuk mencari solusi adalah [1]:

1. Solusi dicari dengan membangun jalur dari akar ke daun menggunakan algoritma DFS.
2. Simpul yang terbentuk disebut sebagai simpul hidup (*live nodes*).
3. Simpul yang sedang diperluas disebut sebagai *expand nodes* atau *E-nodes*.
4. Setiap kali sebuah *E-node* sedang diperluas, jalur yang dikembangkannya menjadi lebih panjang.



Gambar 2.3: Ilustrasi *State space tree* yang digunakan dalam algoritma *backtracking* [1]

5. Jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka *E-node* dimatikan dan menjadi simpul mati (*dead node*).
6. Fungsi yang digunakan untuk mematikan *E-node* adalah implementasi dari fungsi pembatas.
7. Simpul mati tidak akan diperluas.
8. Jika jalur yang sedang dibangun berakhir dengan simpul mati, proses akan mundur ke simpul sebelumnya.
9. Simpul sebelumnya terus membangkitkan simpul anak (*child node*) lainnya, yang kemudian menjadi *E-node* baru.
10. Pencarian selesai jika simpul tujuan tercapai.

Setiap simpul di dalam *state space tree* terkait dengan panggilan rekursif. Jika jumlah simpul di dalam pohon $2n$ atau $n!$, maka pada kasus terburuk untuk algoritma *backtracking* ini memiliki kompleksitas waktu $O(p(n)2n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ sebagai polinomial dengan n-derajat menyatakan waktu komputasi untuk setiap simpul.

Algoritma *backtracking* mempunyai beberapa sifat-sifat umum, yaitu ruang solusi (*solution space*), fungsi pembangkit (*generating function*), dan fungsi pembatas (*bounding function*).

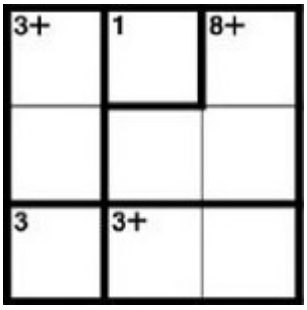
Ruang solusi untuk sebuah permainan teka-teki Calcudoku dengan *grid* yang berukuran $n \times n$ adalah $X = (x_1, x_2, ..., x_m), x_i \in \{1, 2, ..., n\}$, dengan $m = n^2$. Fungsi pembangkit membangkitkan sebuah integer secara berurutan dari 1 sampai n sebagai x_k. Fungsi pembatas menggabungkan tiga fungsi pemeriksa pembatas (*constraint checking*), yaitu fungsi pemeriksa kolom (*column checking*), fungsi pemeriksa baris (*row checking*), dan fungsi pemeriksa *grid* (*grid checking*).

Fungsi pemeriksa kolom menghasilkan nilai *true* jika x_k belum ada di dalam kolom dan menghasilkan nilai *false* jika x_k sudah ada di dalam kolom.

Fungsi pemeriksa baris menghasilkan nilai *true* jika x_k belum ada di dalam baris dan menghasilkan nilai *false* jika x_k sudah ada di dalam baris.

Fungsi pemeriksa *grid* memeriksa operator pada *grid* dan memeriksa berdasarkan operator yang telah ditentukan. Ada 5 operator yang digunakan dalam fungsi ini, yaitu:

1. Operator penjumlahan (+), fungsi menghasilkan nilai *true* jika hasil penjumlahan semua nilai yang ada pada *grid* ditambah dengan x_k kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika jumlah semua nilai yang ada pada *grid* ditambah x_k lebih dari nilai tujuan.



Gambar 2.4: Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 3 x 3 [1]

2. Operator pengurangan (-), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari x_k dikurangi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dikurangi dengan x_k menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika ada satu sel kosong dan hasil dari x_k dikurangi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dikurangi dengan x_k tidak menghasilkan nilai tujuan.

3. Operator perkalian ($\times$), fungsi menghasilkan nilai *true* jika hasil perkalian dari semua nilai yang ada pada *grid* dikali dengan x_k kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika hasil perkalian dari semua nilai yang ada pada *grid* dikali dengan x_k lebih dari nilai tujuan.

4. Operator pembagian ($\div$), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari x_k dibagi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dibagi dengan x_k menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika ada satu sel yang kosong dan hasil dari x_k dibagi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dibagi dengan x_k tidak menghasilkan nilai tujuan.

5. Operator $=$, fungsi akan menghasilkan nilai *true* jika x_k sama dengan nilai tujuan, dan menghasilkan nilai *false* jika x_k tidak sama dengan nilai tujuan.

State space tree bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan. Untuk mengilustrasikan berkembangnya *state space tree*, teka-teki Calcudoku yang digambarkan pada Gambar 2.4 akan digunakan. Berikut ini adalah tahap-tahap berkembangnya *state space tree* untuk teka-teki tersebut.

1. *State space tree* dimulai dengan *state* 1 yang merepresentasikan sebuah *grid* yang kosong.

2. Fungsi pembangkit pertama-tama akan membangkitkan angka 1 sebagai x_1, yang akan diisikan pada sel pertama yang kosong, yaitu sel yang terletak di sudut kiri atas *grid*, atau sel pada kolom ke-1 dan baris ke-1 (*state* 2). Fungsi pembatas akan memeriksa jika langkah ini adalah langkah yang berlaku, dan ternyata langkah ini berlaku.

3. Untuk sel yang kosong berikutnya, yaitu x_2, atau sel pada kolom ke-2 dan baris ke-1, fungsi pembangkit akan membangkitkan angka 1 (*state* 3), tetapi langkah ini gagal dalam pemeriksaan baris dalam fungsi pembatas karena angka 1 sudah pernah digunakan pada baris tersebut, ini membentuk sebuah simpul mati.

4. Fungsi pembangkit akan mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 4), tetapi langkah ini gagal dalam pemeriksaan *grid* dalam fungsi pembatas karena angka 2 tidak sama dengan angka tujuan, yaitu angka 1.

Gambar 2.5: Ilustrasi *state* 3, 4, dan 5 pada sebuah *grid* teka-teki Calcudoku [1]

5. Fungsi pembangkit akan mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 5), tetapi langkah ini juga gagal dalam pemeriksaan *grid* dalam fungsi pembatas karena angka 3 tidak sama dengan angka tujuan, yaitu angka 1. Gambar 2.5 menggambarkan *state* 3, *state* 4, dan *state* 5 dalam penyelesaian teka-teki Calcudoku ini.

6. Karena tidak ada solusi yang mungkin, maka algoritma *backtracking* akan mundur ke *state* 1. Fungsi pembangkit akan membangkitkan kemungkinan angka berikutnya sebagai x_1, yaitu 2, dan ternyata angka 2 berlaku sebagai x_1 (*state* 6), sehingga algoritma bisa maju ke x_2, yaitu sel pada kolom ke-2 dan baris ke-1.

7. Fungsi pembangkit akan membangkitkan angka 1 (*state* 7), dan ini memenuhi syarat yang ditentukan dalam fungsi pembatas, karena angka 1 sama dengan angka tujuan, yaitu angka 1, sehingga algoritma bisa maju ke x_3, yaitu sel pada kolom ke-3 dan baris ke-1.

8. Angka 1 (*state* 8) gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan pada baris tersebut.

9. Angka 2 (*state* 9) juga gagal dalam pemeriksaan baris karena angka 2 sudah pernah digunakan pada baris tersebut.

10. Hal ini menyebabkan hanya tersisa angka 3 sebagai angka yang bisa dimasukkan ke dalam x_3 (*state* 10). Karena *state* 10 ternyata berlaku, maka algoritma telah selesai mengisi baris ke-1, dan akan mulai mengisi baris ke-2.

11. Algoritma lalu membuat *state* baru dengan mengisikan angka 1 pada x_4, yaitu sel pada kolom ke-1 dan baris ke-2 (*state* 11). Ini memenuhi pemeriksaan pembatas, karena $2 + 1 = 3$, sehingga algoritma akan maju ke sel berikutnya, yaitu x_5, atau sel pada kolom ke-2 dan baris ke-2.

12. Angka 1 (*state* 12) jelas tidak bisa digunakan karena gagal dalam pemeriksaan kolom dan pemeriksaan baris; angka 1 sudah pernah digunakan pada kolom dan baris tersebut.

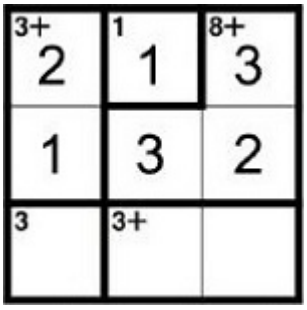
13. Angka 2 (*state* 13) adalah langkah yang berlaku, sehingga algoritma bisa maju ke sel berikutnya, yaitu x_6, atau sel pada kolom ke-3 dan baris ke-2.

14. Algoritma mengisikan x_6 dengan angka 1 (*state* 14), tetapi gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan pada baris tersebut.

15. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 15), tetapi juga gagal dalam pemeriksaan baris karena angka 2 sudah pernah digunakan pada baris tersebut.

16. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 16), tetapi juga gagal, kali ini angka 3 gagal dalam pemeriksaan kolom karena angka 3 sudah pernah digunakan pada kolom tersebut.

17. Karena semua kemungkinan angka gagal dalam pemeriksaan baris dan kolom, maka algoritma akan mundur ke *state* 11 dan mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state*



Gambar 2.6: Ilustrasi *state* 19 pada sebuah *grid* teka-teki Calcudoku [1]

17), dan ternyata angka 3 berlaku sebagai x_5, sehingga algoritma bisa maju ke sel berikutnya, yaitu x_6.

18. Algoritma lalu mencoba angka 1 (*state* 18) sebagai x_6, tetapi gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan dalam baris tersebut.

19. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 19), dan ternyata angka 2 berlaku. Algoritma telah selesai mengisi baris ke-2. Gambar 2.6 menggambarkan *state* 19 dalam penyelesaian teka-teki Calcudoku ini.

20. Algoritma mulai mengisikan sel-sel yang terletak pada baris ke-3. Algoritma mengisi dari kolom yang paling kiri ke kolom yang paling kanan. Algoritma mengisikan x_7, yaitu sel pada kolom ke-1 dan baris ke-3 dengan angka 1 (*state* 20), tetapi gagal dalam pemeriksaan kolom, karena angka 1 sudah pernah digunakan dalam kolom tersebut.

21. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 21), tetapi juga gagal dalam pemeriksaan kolom, karena angka 2 sudah pernah digunakan dalam kolom tersebut.

22. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 22), dan ternyata berhasil, sehingga algoritma bisa maju ke sel berikutnya, yaitu x_8, atau sel pada kolom ke-2 dan baris ke-3.

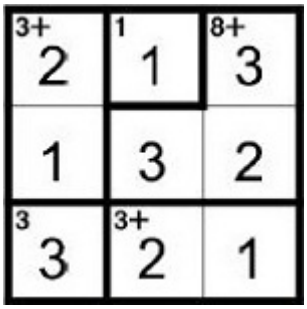
23. Algoritma lalu mencoba mengisikan angka 1 pada x_8 (*state* 23), tetapi gagal dalam pemeriksaan kolom, karena angka 1 sudah pernah digunakan dalam kolom tersebut.

24. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 24), dan ternyata berhasil, sehingga algoritma bisa maju ke sel berikutnya, yaitu x_9, atau sel pada kolom ke-3 dan baris ke-3.

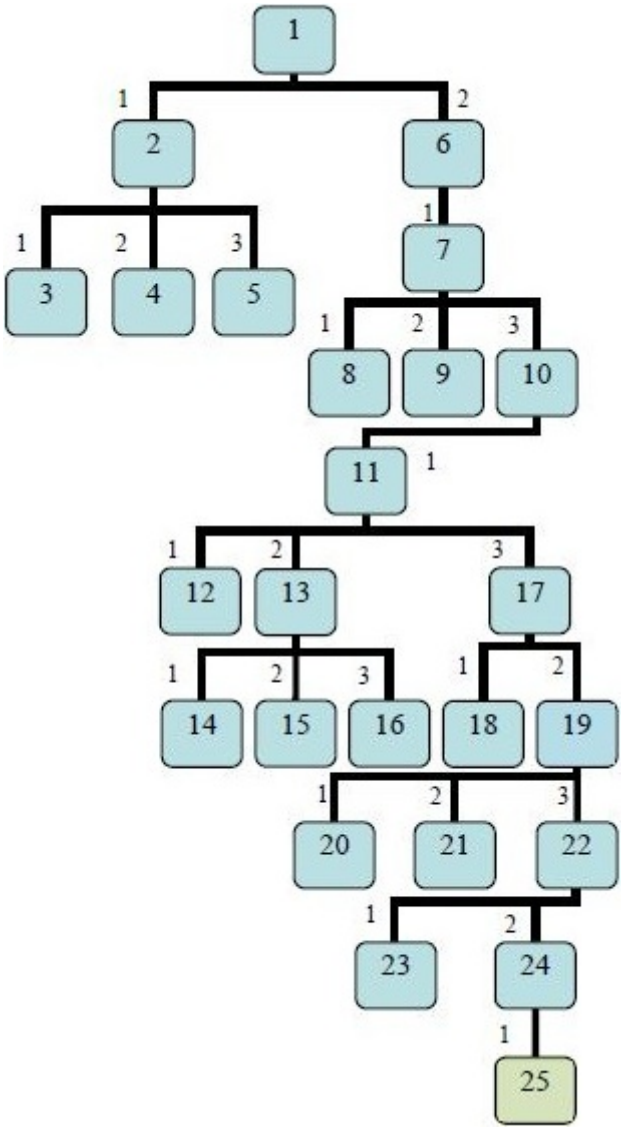
25. x_9 adalah sel terakhir, terletak pada sudut kanan bawah *grid*. Algoritma lalu mencoba mengisikan x_9 dengan angka 1 (*state* 25), dan ternyata berhasil. Algoritma telah selesai mengisikan seluruh sel dalam *grid* dengan benar. Gambar 2.7 menggambarkan *state* 25 dalam penyelesaian teka-teki Calcudoku ini. Algoritma ini mencapai solusinya pada *state* 25, seperti pada *state space tree* yang digambarkan dalam Gambar 2.8. *State space tree* ini telah mencapai simpul tujuannya, yaitu simpul 25, dengan jalur 2-1-3-1-3-2-3-2-1.

Tinggi pohon yang dikembangkan untuk menyelesaikan sebuah teka-teki dengan ukuran $n \times n$ seharusnya memiliki tinggi $n^2 + 1$ saat mencapai simpul tujuannya, dengan jalur dari simpul akar ke simpul tujuan merepresentasikan semua angka yang digunakan untuk mengisi *grid* dari sel pada sudut kiri atas ke sel pada sudut kanan bawah.

Singkatnya, langkah-langkah dasar dari implementasi algoritma *backtracking* dapat dijelaskan sebagai berikut [1]:



Gambar 2.7: *State* 25, simpul tujuan, sebagai hasil yang dicapai [1]



Gambar 2.8: *state space tree* yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 2.4 [1]

Gambar 2.9: Contoh bagaimana cara mendeteksi aturan *naked pair* [2]

1. Carilah sel pertama atau sel yang kosong di dalam *grid*.
2. Isilah sel dengan sebuah angka dimulai dari 1 sampai n sampai sebuah angka yang berlaku (*valid*) ditemukan atau sampai angka sudah melebihi n.
3. Jika angka untuk sel berlaku, ulangi langkah 1 dan 2.
4. Jika angka untuk sel sudah melebihi n dan tidak ada angka dari 1 sampai n yang berlaku untuk sel tersebut, mundur ke sel sebelumnya dan cobalah kemungkinan angka berikutnya yang berlaku untul sel tersebut.
5. Jika tidak ada lagi sel yang kosong, solusi sudah ditemukan.

2.3 Algoritma *Hybrid Genetic* [2]

Dalam kasus ini, algoritma *hybrid genetic* adalah gabungan dari algoritma *rule based* dan algoritma genetik. Algoritma *rule based* akan dijalankan sampai pada titik dimana algoritma tidak bisa menyelesaikan permainan teka-teki Calcudoku. Jika algoritma sudah tidak bisa menyelesaikan permainan, maka algoritma genetik akan mulai dijalankan.

2.3.1 Algoritma *Rule Based*

Algoritma *rule based* adalah sebuah algoritma berbasis aturan logika untuk menyelesaikan teka-teki Sudoku dan variasinya, termasuk Calcudoku. Menurut Johanna, Lukas, dan Saputra, beberapa aturan logika yang digunakan dalam algoritma ini adalah *single square rule*, *naked subset rule*, *hidden single rule*, *evil twin rule*, *killer combination*, dan *X-wing* [2].

Aturan *single square* digunakan jika sebuah *cage* hanya berisi satu sel. Hal ini berarti nilai dari sel tersebut sama dengan angka tujuan yang telah ditentukan.

Aturan *naked subset* digunakan jika ada n sel dalam kolom atau baris yang sama yang mempunyai n kemungkinan nilai yang sama persis untuk mengisikannya, dengan $n \geq 2$. Hal ini berarti sel-sel lainnya dalam baris dan kolom tersebut tidak mungkin diisi dengan nilai yang sama dengan nilai milik n sel tersebut. Gambar 2.9 menunjukkan bagaimana cara kerja aturan ini. Sel-sel pada kolom ke-4 dan ke-6 mempunyai tepat dua kemungkinan nilai (1 atau 7). Ini disebut sebagai *naked pair*. Karena angka 1 dan 7 harus diisi pada sel-sel pada kolom ke-4 dan ke-6, maka angka 1 dan 7 bisa dieliminasi dari sel-sel pada kolom ke-7 dan ke-8.

Aturan *evil twin* digunakan jika sebuah *cage* berisikan dua sel, dan salah satu dari kedua sel sudah terisi, maka sel yang satunya lagi diisi dengan angka yang jika kedua angka dihitung dengan operasi matematika yang ditentukan maka akan menghasilkan angka tujuan yang ditentukan. Aturan ini adalah aturan yang paling mudah. Kenyataannya, aturan ini bisa digeneralisasikan untuk *cage* yang berukuran lebih dari 2 sel. Sel yang belum terisi yang terakhir dalam sebuah area diisi oleh sebuah nilai yang diperlukan untuk mencapai nilai tujuan menggunakan operasi matematika yang telah ditentukan. Contohnya, pada Gambar 2.10, begitu sel di sudut kiri bawah diisi oleh angka 4, maka sel diatasnya harus diisi oleh angka 9.

Aturan *hidden single* digunakan jika sebuah angka hanya bisa diisikan dalam satu sel dalam sebuah baris atau kolom. Aturan ini secara konsep cukup mudah, tetapi kadang-kadang sulit untuk diamati. Pada Gambar 2.11, nilai-nilai yang mungkin untuk sel yang paling kiri adalah 3, 5, dan 7, tetapi dalam baris ini, angka 7 harus muncul dalam salah satu selnya, dan hanya sel yang paling

Gambar 2.10: Contoh aturan *evil twin* [2]

Gambar 2.11: Contoh aturan *hidden single* [2]

kiri tersebut yang memiliki kemungkinan nilai 7. Ini disebut sebagai *hidden single*. Sel tersebut harus diisi dengan angka 7.

Aturan *killer combination* adalah aturan yang paling krusial. Aturan ini digunakan jika sebuah *cage* berisikan sel-sel yang berada dalam baris atau kolom yang sama dan operasi yang ditentukan adalah penjumlahan. Kemungkinan angka yang unik untuk aturan *killer combination* berhubungan dengan ukuran *cage*. Contoh, jika sebuah *cage* memiliki dua sel dan angka tujuannya adalah 3, maka kemungkinan angka yang bisa diisikan ke dalam kedua sel tersebut adalah 1 atau 2. Hal ini berarti semua angka lainnya tidak mungkin diisikan ke dalam kedua sel tersebut. Contoh lain, jika sebuah *cage* memiliki tiga sel dan angka tujuannya adalah 24, maka kemungkinan angka yang bisa diisikan ke dalam ketiga sel tersebut adalah 7, 8, atau 9. Gambar 2.12 menampilkan contoh penerapan aturan *killer combination* untuk *cage* dengan ukuran 2 sel. Tabel ini juga bisa diperluas untuk ukuran *cage* lainnya.

Aturan *X-wing* digunakan jika hanya ada dua kemungkinan angka yang bisa diisikan ke dalam dua sel yang berada di dalam dua baris yang berbeda, dan dua kemungkinan angka tersebut juga berada di dalam kolom yang sama maka sel-sel lainnya dalam kolom tersebut tidak mungkin diisi oleh dua kemungkinan angka tersebut, atau jika hanya ada dua kemungkinan angka yang bisa diisikan ke dalam dua sel yang berada di dalam dua kolom yang berbeda, dan dua kemungkinan angka tersebut juga berada di dalam baris yang sama maka sel-sel lainnya dalam baris tersebut tidak mungkin diisi oleh dua kemungkinan angka tersebut. Gambar 2.13 menampilkan contoh penggunaan aturan *X-wing*. Misalnya, jika sel A diisi oleh angka 7, maka angka 7 akan dieliminasi dari sel B dan sel C. Karena sel A dengan sel C dan sel D 'terkunci', maka sel D harus diisi oleh angka 7. Jadi, angka 7 harus di isi pada sel A dan sel D atau pada sel B dan sel C. Angka 7 bisa dieliminasi dari sel-sel yang berwarna hijau.

2.3.2 Algoritma Genetik

Pencarian heuristik adalah sebuah teknik pencarian kecerdasan buatan (*artifical intelligence*) yang menggunakan heuristik dalam langkah-langkahnya. Heuristik adalah semacam aturan tidak tertulis yang mungkin menghasilkan solusi. Heuristik kadang-kadang efektif, tetapi tidak dijamin akan berhasil. dalam setiap kasus. Heuristik memerankan peran penting dalam strategi pencarian karena

Cage size	Cage value	Combination
2	3	1/2
2	4	1/3
2	17	8/9
2	16	7/9

Gambar 2.12: Contoh aturan *killer combination* untuk *cage* dengan ukuran 2 sel [2]

Gambar 2.13: Contoh aturan *X-wing* [2]

sifat eksponensial dari kebanyakan masalah. Heuristik membantu mengurangi jumlah alternatif solusi dari angka yang bersifat eksponensial menjadi angka yang bersifat polinomial. Contoh teknik pencarian heuristik adalah *Generate and Test*, *Hill Climbing*, dan *Best First Search*.

Algoritma genetik adalah salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam. Algoritma ini adalah perpaduan dari bidang biologi dan ilmu komputer. Algoritma ini adalah salah satu dari teknik pencarian heuristik.

Algoritma genetik adalah salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam. Algoritma ini adalah perpaduan dari bidang biologi dan ilmu komputer. Algoritma ini memanipulasi informasi, biasanya disebut sebagai kromosom. Kromosom ini meng-*encode* kemungkinan jawaban untuk sebuah masalah yang diberikan. Kromosom dievaluasi dan diberi *fitness value* berdasarkan seberapa baikkah kromosom dalam menyelesaikan masalah yang diberikan berdasarkan kriteria yang ditentukan oleh pembuat program. Nilai kelayakan ini digunakan sebagai probabilitas kebertahanan hidup kromosom dalam satu siklus reproduksi. Kromosom baru (kromosom anak, *child chromosome*) diproduksi dengan menggabungkan dua (atau lebih) kromosom orang tua (*parent chromosome*). Proses ini dirancang untuk menghasilkan kromosom-kromosom keturunan yang lebih layak, kromosom-kromosom ini menyandikan jawaban yang lebih baik, sampai solusi yang baik dan yang bisa diterima ditemukan.

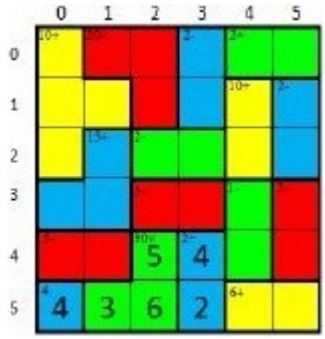
Cara kerja algoritma genetik adalah sebagai berikut [2]:

1. Menentukan populasi kromosom kemungkinan jawaban awal.
2. Membangkitkan populasi kemungkinan jawaban awal secara acak.
3. Mengevaluasi fungsi objektif.
4. Melakukan operasi terhadap kromosom menggunakan operator genetik (reproduksi, kawin silang, dan mutasi).
5. Ulangi langkah 3 dan 4 sampai mencapai kriteria untuk menghentikan algoritma.

Langkah-langkah utama dalam penggunaan algoritma genetik adalah membangkitkan populasi kemungkinan jawaban, mencari fungsi objektif dan fungsi kelayakan, dan penggunaan operator genetik.

2.3.3 Algoritma *Hybrid Genetic*

Pencarian *rule based* dimulai dengan mengasumsikan semua nilai sel yang tidak diketahui dengan semua kemungkinan nilai untuk mengisi sel tersebut tanpa melanggar batasan, dengan $P(C_{b,k}) = 1, 2, ..., n$. Setelah nilai dari satu sel sudah ditentukan, kemungkinan nilai untuk beberapa sel tertentu diperbaharui. Misalnya, penggunaan aturan *naked single* yang dinyatakan dalam persamaan



Gambar 2.14: Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 6 x 6 [2]

1 di bawah ini, akan mengakibatkan semua kemungkinan nilai untuk semua sel lain dalam baris yang sama dan dalam kolom yang sama harus diperbaharui, seperti dinyatakan dalam persamaan 2 dan 3 di bawah ini. Aturan *naked pair*, salah satu dari aturan jenis *naked subset*, dinyatakan dalam persamaan 4 untuk baris dan persamaan 5 untuk kolom. [2]

1. $|P(C_{b,k})| = 1 \wedge x \in P(C_{b,k}) \rightarrow V(C_{b,k}) = x$, artinya jika sebuah *cage* berukuran 1 sel, dan x adalah nilai tujuan dari *cage* tersebut, maka nilai dari sel tersebut adalah x.

2. $(V(C_{b,k}) = x) \wedge (\forall a \in \{1, 2, ..., n\}) \rightarrow P(C_{a,k}) = P(C_{a,k}) - \{x\}$, artinya jika nilai suatu sel pada baris b dan kolom k adalah x, maka x dihapus dari kemungkinan angka-angka yang bisa digunakan untuk mengisi sel-sel lain pada baris b.

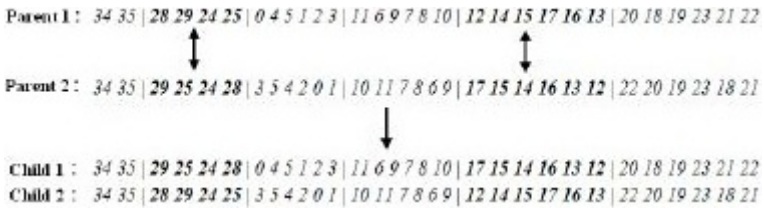
3. $(V(C_{b,k}) = x) \wedge (\forall q \in \{1, 2, ..., n\}) \rightarrow P(C_{b,q}) = P(C_{b,q}) - \{x\}$ artinya jika nilai suatu sel pada baris b dan kolom k adalah x, maka x dihapus dari kemungkinan angka-angka yang bisa digunakan untuk mengisi sel-sel lain pada kolom k.

4. $|P(C_{b,k1})| = |P(C_{b,k2})| = 2 \wedge P(C_{b,k1}) = P(C_{b,k2}) \rightarrow P(C_{b,q}) = P(C_{b,q}) - P(C_{b,k1})$, artinya jika ada dua sel dalam satu baris yang hanya bisa diisi oleh dua kemungkinan angka, maka kedua angka tersebut dihapus dari kemungkinan angka-angka yang bisa digunakan untuk mengisi sel-sel lain pada baris tersebut.

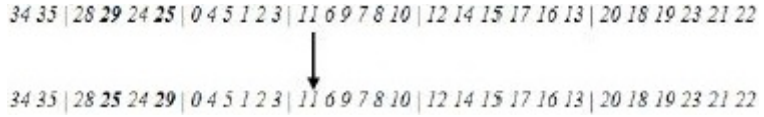
5. $|P(C_{b1,k})| = |P(C_{b2,k})| = 2 \wedge P(C_{b1,k}) = P(C_{b2,k}) \rightarrow P(C_{p,k}) = P(C_{p,k}) - P(C_{b1,k})$, artinya jika ada dua sel dalam satu kolom yang hanya bisa diisi oleh dua kemungkinan angka, maka kedua angka tersebut dihapus dari kemungkinan angka-angka yang bisa digunakan untuk mengisi sel-sel lain pada kolom tersebut.

Algoritma genetik digunakan saat teka-teki masih tidak bisa diselesaikan setelah mengerjakan semua aturan logika secara berulang-ulang. Algoritma ini dimulai dengan meng-*encode* kromosom. Satu kromosom terdiri dari k segmen, dengan $m \leq n$. Satu segmen berisikan sekumpulan gen yang belum diselesaikan yang berada di dalam segmen tersebut. Sebuah segmen merepresentasikan sebuah baris atau kolom. Dalam sebuah kromosom, segmen diurutkan dari baris yang paling atas ke baris yang paling bawah atau dari kolom yang paling kiri ke kolom yang paling kanan. Contoh, salah satu kromosom dari permainan teka-teki Calcudoku pada Gambar 2.14 adalah 34 35 | 28 29 24 25 | 0 4 5 1 2 3 | 11 6 9 7 8 10 | 12 14 15 17 16 13 | 20 18 19 23 21 22. Setiap segmen dalam contoh kromosom ini merepresentasikan sebuah baris yang belum terselesaikan.

Menurut Johanna, Lukas, dan Saputra, fungsi objektif, yang direpresentasikan dengan x_j, akan dihitung setelah pembangkitan nilai dari gen pada kromosom sudah dilakukan. Nilai untuk gen ke-j pada sebuah kromosom direpresentasikan dengan w_j. x_j akan bernilai 0 jika belum diselesaikan ($w_j = 0$), dan bernilai 1 jika sudah diselesaikan ($w_j \neq 0$). Untuk kromosom dengan jumlah gen k, fungsi kelayakan, yaitu hasil penjumlahan dari hasil fungsi objektif untuk setiap gen dibagi dengan



Gambar 2.15: Contoh proses kawin silang antara dua kromosom [2]



Gambar 2.16: Contoh proses mutasi [2]

jumlah gen, dinyatakan dalam persamaan di bawah ini [2]:

$$x_j = \{\, 0\ , w_j = 01, w_j \neq 0$$

$$fitness = \frac{\sum_{j=0}^{k} x_j}{k}$$

Jadi, solusi dari teka-teki ini adalah mencari kromosom yang nilai kelayakannya 1.

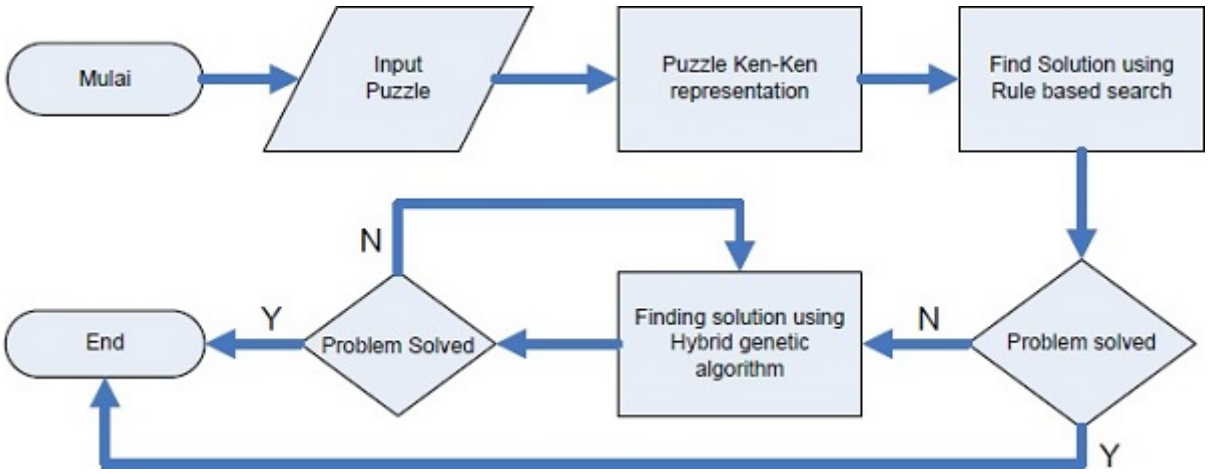
Dalam proses reproduksi kawin silang, dua kromosom, yaitu kromosom orang tua, disilangkan untuk membuat dua kromosom yang baru, yaitu kromosom anak, dengan metodologi kawin silang *N-segments*. Gambar 2.15 menggambarkan contoh proses kawin silang antara dua kromosom.

Pertukaran mutasi digunakan untuk mendapatkan kemungkinan kromosom yang lain. Mutasi dilakukan di antara gen yang berada dalam segmen yang sama. Gambar 2.16 adalah contoh proses mutasi antara dua gen dalam segmen yang sama.

Cara kerja algoritma *hybrid genetic* menurut Johanna, Lukas, dan Saputra adalah sebagai berikut [2]:

1. Masukkan teka-teki yang akan diselesaikan sebagai input.
2. Program akan merepresentasikan input yang dimasukkan dalam format teka-teki.
3. Program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* terlebih dahulu.
4. Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based*, maka algoritma selesai.
5. Jika program gagal dengan menggunakan algoritma *rule based*, maka program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik.
6. Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik, maka algoritma selesai.
7. Jika program gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik, artinya algoritma gagal dalam menyelesaikan teka-teki terseebut.

Alur (*flow chart*) penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* dapat dilihat di Gambar 2.17.



Gambar 2.17: Alur penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* [2]

BAB 3

ANALISIS

Bab ini membahas tentang cara kerja algoritma *backtracking* dan algoritma *hybrid genetic* untuk menyelesaikan permainan teka-teki Calcudoku.

3.1 Algoritma *Backtracking*

Untuk mengilustrasikan cara kerja algoritma *backtracking*, akan digunakan permainan teka-teki Calcudoku yang digambarkan pada Gambar 3.1 sebagai contoh.

1. Algoritma *backtracking* dimulai dengan teka-teki yang belum diselesaikan, seperti yang digambarkan pada Gambar 3.1 (*state* 1).

2. Algoritma mengisikan sel pada baris ke-1 dan kolom ke-1 dengan angka 1 (*state* 2), tetapi angka 1 tidak sesuai dengan angka tujuan dari *cage* tersebut.

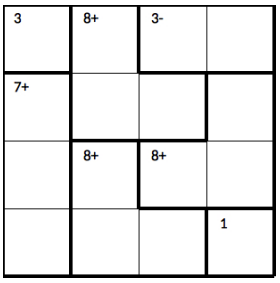
3. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 3), tetapi angka 2 juga tidak sesuai dengan angka tujuan dari *cage* tersebut.

4. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 4), seperti dapat dilihat pada Gambar 3.2, dan ternyata angka 3 sesuai dengan angka tujuan dari *cage* tersebut, sehingga algoritma dapat maju ke sel berikutnya.

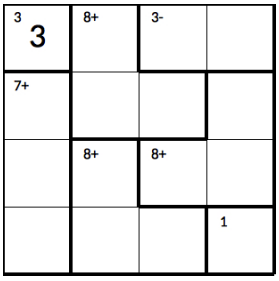
5. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-2 dengan angka 1 (*state* 5). Algoritma lalu maju ke sel berikutnya.

6. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state* 6), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

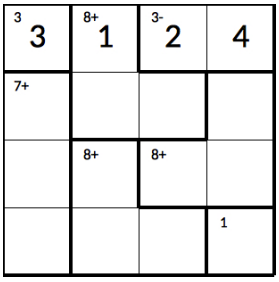
7. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 7). Algoritma lalu maju ke sel berikutnya.



Gambar 3.1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan, seperti yang digambarkan pada Gambar 2.1. [1]



Gambar 3.2: *State* 4



Gambar 3.3: *State* 11

8. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state* 8), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

9. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 9), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

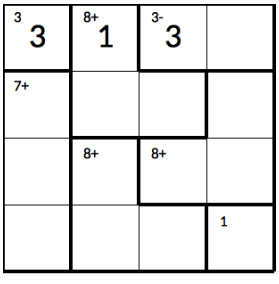
10. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 10), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.

11. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 11), seperti dapat dilihat pada Gambar 3.3, tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.

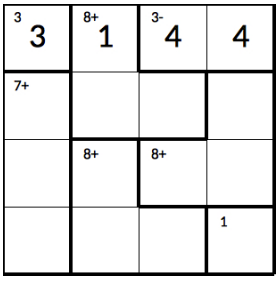
12. Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 7). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 12), seperti dapat dilihat pada Gambar 3.4, tetapi angka 3 sudah pernah digunakan dalam baris tersebut. Algoritma lalu maju ke sel berikutnya.

13. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 13). Algoritma lalu maju ke sel berikutnya.

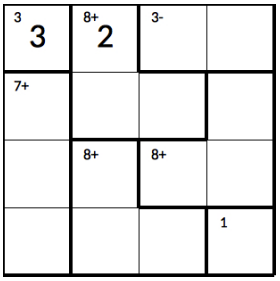
14. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state* 14), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.



Gambar 3.4: *State* 12



Gambar 3.5: *State* 17



Gambar 3.6: *State* 18

15. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 15), tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.

16. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 16), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.

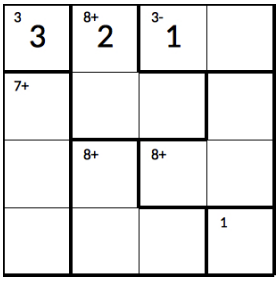
17. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 17), seperti dapat dilihat pada Gambar 3.5, tetapi angka 4 sudah pernah digunakan dalam baris tersebut.

18. Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-3 dan ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 5). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 18), seperti dapat dilihat pada Gambar 3.6. Algoritma lalu maju ke sel berikutnya.

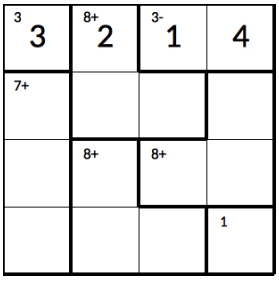
19. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state* 19), seperti dapat dilihat pada Gambar 3.7. Algoritma lalu maju ke sel berikutnya.

20. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state* 20), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

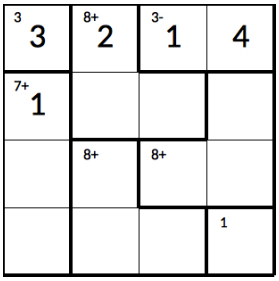
21. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 21), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.



Gambar 3.7: *State* 19



Gambar 3.8: *State* 23



Gambar 3.9: *State* 24

22. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 22), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.

23. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 23), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.8. Algoritma telah selesai mengisikan baris ke-1, sehingga bisa maju ke baris berikutnya.

24. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-1 dengan angka 1 (*state* 24), seperti dapat dilihat pada Gambar 3.9. Algoritma lalu maju ke sel berikutnya.

25. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-2 dengan angka 1 (*state* 25), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

26. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 26), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.

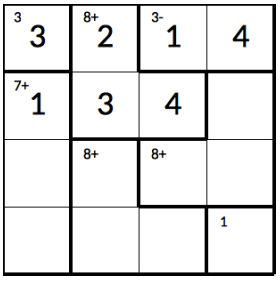
27. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 27). Algoritma lalu maju ke sel berikutnya.

28. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-3 dengan angka 1 (*state* 28), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.

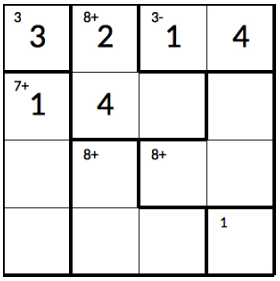
29. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 29), tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.

30. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 30), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.

31. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 31), seperti dapat dilihat pada Gambar 3.10, tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.



Gambar 3.10: *State* 31



Gambar 3.11: *State* 32

32. Karena semua kemungkinan angka untuk baris ke-2 dan kolom ke-3 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 27). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 32), seperti dapat dilihat pada Gambar 3.11. Algoritma lalu maju ke sel berikutnya.

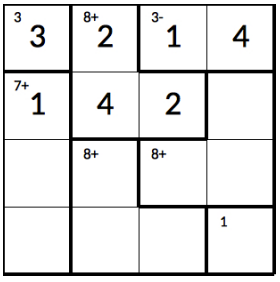
33. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-3 dengan angka 1 (*state* 33), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

34. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 34), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.12. Algoritma lalu maju ke sel berikutnya.

35. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-4 dengan angka 1 (*state* 35), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

36. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 36), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

37. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 37), seperti dapat dilihat pada Gambar 3.13. Algoritma telah selesai mengisikan baris ke-2, sehingga bisa maju ke baris berikutnya.



Gambar 3.12: *State* 34

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
	8+	8+	
			1

Gambar 3.13: *State* 37

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 1	8+ 3	4
			1

Gambar 3.14: *State* 47

38. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-1 dengan angka 1 (*state* 38), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.

39. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 39). Algoritma lalu maju ke sel berikutnya.

40. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-2 dengan angka 1 (*state* 40). Algoritma lalu maju ke sel berikutnya.

41. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state* 41), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.

42. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 42), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.

43. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 43). Algoritma lalu maju ke sel berikutnya.

44. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-4 dengan angka 1 (*state* 44), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

45. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 45), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

46. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 46), tetapi angka 3 sudah pernah digunakan dalam baris dan kolom tersebut.

47. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 47), seperti dapat dilihat pada Gambar 3.14, tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.

48. Karena semua kemungkinan angka untuk baris ke-3 dan kolom ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 43). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 48), seperti dapat dilihat pada Gambar 3.15. Algoritma lalu maju ke sel berikutnya.

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 1	8+ 4	
			1

Gambar 3.15: *State* 48

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 1	8+ 4	4
			1

Gambar 3.16: *State* 52

49. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-4 dengan angka 1 (*state* 49), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

50. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 50), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

51. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 51), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.

52. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 52), seperti dapat dilihat pada Gambar 3.16, tetapi angka 3 sudah pernah digunakan dalam baris dan kolom tersebut.

53. Karena semua kemungkinan angka untuk baris ke-3 dan kolom ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 48). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 53) seperti dapat dilihat pada Gambar 3.17, tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.

54. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 54). Algoritma lalu maju ke sel berikutnya.

55. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state* 55), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.

3	2	1	4
7+ 1	4	2	3
2	8+ 2	8+	
			1

Gambar 3.17: *State* 53

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 3	8+ 4	1
4	1	4	1

Gambar 3.18: *State* 68

56. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 56), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.

57. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 57), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.

58. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 58). Algoritma lalu maju ke sel berikutnya.

59. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-4 dengan angka 1 (*state* 59). Algoritma telah selesai mengisikan baris ke-2, sehingga bisa maju ke baris berikutnya.

60. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-1 dengan angka 1 (*state* 60), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.

61. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 61), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.

62. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 62), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.

63. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 63). Algoritma lalu maju ke sel berikutnya.

64. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-2 dengan angka 1 (*state* 64). Algoritma lalu maju ke sel berikutnya.

65. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-3 dengan angka 1 (*state* 65), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.

66. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 66), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.

67. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 67), tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.

68. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 68), seperti dapat dilihat di Gambar 3.18. tetapi angka 4 sudah pernah digunakan dalam baris dan kolom tersebut.

69. Karena semua kemungkinan angka untuk baris ke-4 dan kolom ke-3 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 64). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 69), seperti dapat dilihat pada Gambar 3.19, tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 3	8+ 4	1
4	2		1

Gambar 3.19: *State* 69

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 3	8+ 4	1
4	4		1

Gambar 3.20: *State* 71

70. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 70), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.

71. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 71), seperti dapat dilihat di Gambar 3.20, tetapi angka 4 sudah pernah digunakan dalam kolom tersebut.

72. Karena semua kemungkinan angka untuk baris ke-4 dan kolom ke-1 dan ke-2 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 59). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 72), seperti dapat dilihat pada Gambar 3.21, tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

73. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 73), tetapi angka 3 sudah pernah digunakan dalam baris dan kolom tersebut.

74. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 74), seperti dapat dilihat di Gambar 3.22, tetapi angka 4 sudah pernah digunakan dalam baris dan kolom tersebut.

75. Karena semua kemungkinan angka untuk baris ke-3 dan kolom ke-3 dan ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 54). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 75), seperti dapat dilihat pada Gambar 3.23, tetapi angka 4 sudah pernah digunakan dalam kolom tersebut.

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 3	8+ 4	2
			1

Gambar 3.21: *State* 72

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 3	8+ 4	4
			1

Gambar 3.22: *State* 74

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
2	8+ 4	8+	
			1

Gambar 3.23: *State* 75

76. Karena semua kemungkinan angka untuk baris ke-3 dan kolom ke-2 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 54). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 76), seperti dapat dilihat pada Gambar 3.24, tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.

77. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 77), seperti dapat dilihat di Gambar 3.25. Algoritma lalu maju ke sel berikutnya.

78. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-2 dengan angka 1 (*state* 78), seperti dapat dilihat di Gambar 3.26. Algoritma lalu maju ke sel berikutnya.

79. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state* 79), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.

80. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 80), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.

81. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 81), seperti dapat dilihat pada Gambar 3.27. Algoritma lalu maju ke sel berikutnya.

82. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state* 82), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
3	8+	8+	
			1

Gambar 3.24: *State* 76

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
4	8+	8+	
			1

Gambar 3.25: *State* 77

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
4	8+ 1	8+	
			1

Gambar 3.26: *State* 78

3 3	8+ 2	3- 1	4
7+ 1	4	2	3
4	8+ 1	8+ 3	
			1

Gambar 3.27: *State* 81

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
4	8+ 1	8+ 3	2
			1

Gambar 3.28: *State* 83

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
4	8+ 1	8+ 3	2
2			1

Gambar 3.29: *State* 85

83. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 83), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.28. Algoritma telah selesai mengisikan baris ke-3, sehingga bisa maju ke baris berikutnya.

84. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-1 dengan angka 1 (*state* 84), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.

85. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 85), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.29. Algoritma lalu maju ke sel berikutnya.

86. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-2 dengan angka 1 (*state* 86), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.

87. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 87), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.

88. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 88), seperti dapat dilihat pada Gambar 3.30. Algoritma lalu maju ke sel berikutnya.

89. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-3 dengan angka 1 (*state* 89), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
4	8+ 1	8+ 3	2
2	3		1

Gambar 3.30: *State* 88

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
4	8+ 1	8+ 3	2
2	3	4	1

Gambar 3.31: *State* 92

3+ 3	8+ 2	3- 1	4
7+ 1	4	2	3
4	8+ 1	8+ 3	2
2	3	4	1 1

Gambar 3.32: *State* 93

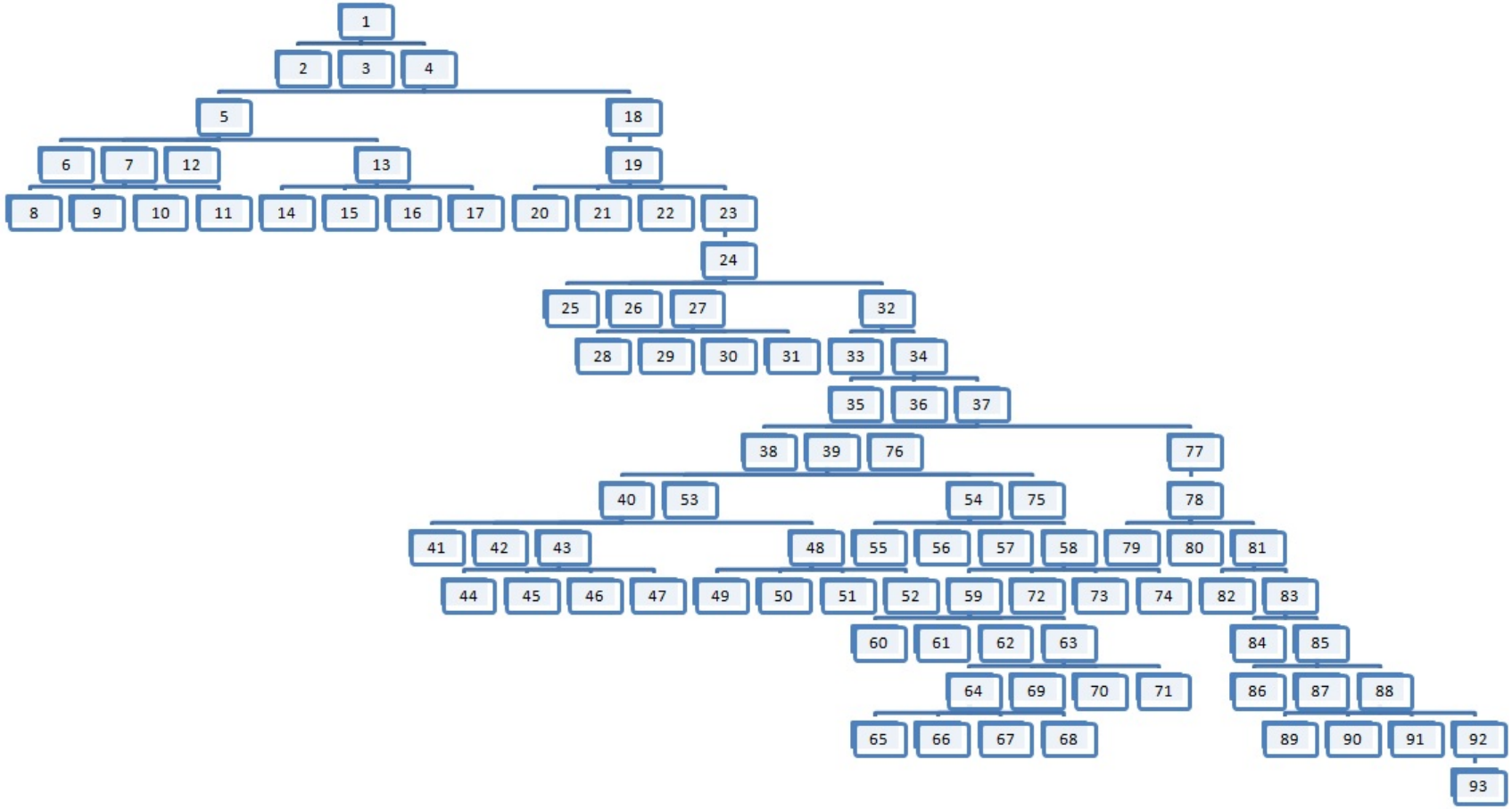
90. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 90), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.

91. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 91), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.

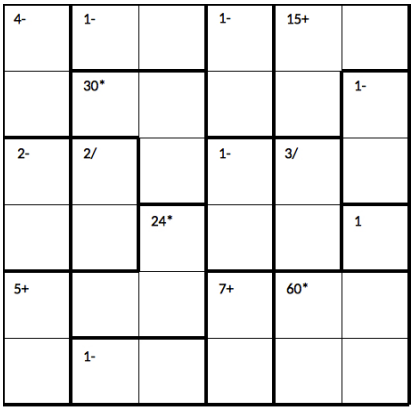
92. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state* 92), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.31. Algoritma lalu maju ke sel berikutnya.

93. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-4 dengan angka 1 (*state* 93), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.32. Algoritma *backtracking* telah selesai mengisi semua sel dalam permainan teka-teki Calcudoku ini dengan benar.

Algoritma ini mencapai solusinya pada state 93, seperti pada *state space tree* yang digambarkan dalam Gambar 3.33. *State space tree* ini telah mencapai simpul tujuannya, yaitu simpul 93, dengan jalur 3-2-1-4-1-4-2-3-4-1-3-2-2-3-4-1.



Gambar 3.33: *State space tree* yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.1

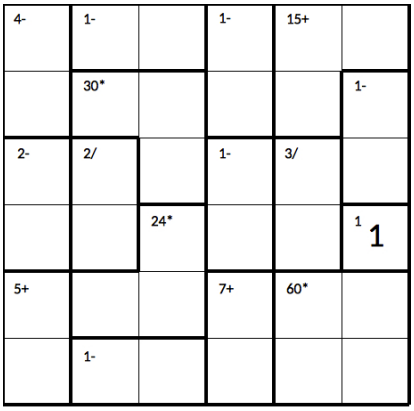


Gambar 3.34: Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 6 x 6 yang belum diselesaikan, seperti yang digambarkan pada Gambar 1.1. [2]

3.2 Algoritma *Hybrid Genetic*

Untuk mengilustrasikan cara kerja algoritma *hybrid genetic*, akan digunakan permainan teka-teki Calcudoku yang digambarkan pada Gambar 3.34 sebagai contoh.

Algoritma *hybrid genetic* dimulai dengan mencoba menyelesaikan permainan teka-teki Calcudoku dengan algoritma *rule based*. Sel pada baris ke-4 dan kolom ke-6 adalah bagian dari sebuah *cage* yang berukuran hanya 1 sel, dan oleh karena itu, angka tujuan dari sel tersebut adalah angka tujuan dari *cage* tersebut. Dalam kasus ini, angka tujuan tersebut adalah 1. Maka, sel tersebut dapat langsung diisi dengan angka 1, seperti dapat dilihat pada Gambar 3.35. Sayangnya, algoritma *rule based* tidak bisa mengisi sel-sel lainnya berdasarkan aturan-aturan yang telah didefinisikan setelah beberapa kali percobaan, sehingga algoritma *hybrid genetic* akan mencoba menyelesaikan teka-teki Calcudoku dengan algoritma genetik.



Gambar 3.35: Permainan teka-teki Calcudoku setelah diselesaikan dengan algoritma *rule based*

DAFTAR REFERENSI

[1] Fahda, A. (2015) Kenken puzzle solver using backtracking algorithm. Makalah IF2211 Strategi Algoritma - Semester II Tahun 2014/2015, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_016.pdf.

[2] Johanna, O., Lukas, S., dan Saputra, K. V. I. (2012) Solving and modeling ken-ken puzzle by using hybrid genetics algorithm. *1st International Conference on Engineering and Technology Development (ICETD 2012)*, Bandar Lampung, Lampung, Indonesia, 20-21 Juni, pp. 98–102. Faculty of Engineering and Faculty of Computer Science, Bandar Lampung University.