Perbandingan Algoritma *Backtracking* dan Algoritma *Hybrid Genetic* untuk Menyelesaikan Permainan Calcudoku

Who? Michael Adrian
2013730039
michaeladrian39@gmail.com

From? Jurusan Teknik Informatika
Fakultas Teknologi Informasi dan Sains
Universitas Katolik Parahyangan

When? 6 Desember 2016

Perbandingan Algoritma *Backtracking* dan Algoritma *Hybrid Genetic* untuk Menyelesaikan Permainan Calcudoku

Who? Michael Adrian
2013730039
michaeladrian39@gmail.com

From? Jurusan Teknik Informatika
Fakultas Teknologi Informasi dan Sains
Universitas Katolik Parahyangan

When? 6 Desember 2016

Calcudoku

- Salah satu jenis permainan teka-teki aritmatika dan logika
- Dikenal juga sebagai KenKen, KenDoku, atau Mathdoku
- Diciptakan pada tahun 2004 oleh Tetsuya Miyamoto, seorang guru matematika dari Jepang
- Diciptakan untuk melatih kemampuan matematika dan logika dengan cara yang menyenangkan

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.

Aturan Permainan

- Pemain diberikan sebuah *grid* dengan ukuran $n \times n$
- n biasanya antara 3 sampai dengan 9
- *Grid* ini harus diisi dengan angka 1 sampai dengan n
- Dalam setiap baris setiap angka hanya muncul sekali
- Dalam setiap kolom setiap angka hanya muncul sekali
- *Grid* dibagi ke dalam *cage*
- *Cage* adalah sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel dengan angka tujuan dan operator yang telah ditentukan
- 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*

Aturan Permainan
- Pemain diberikan sebuah *grid* dengan ukuran $n \times n$
- n biasanya antara 3 sampai dengan 9
- *Grid* ini harus diisi dengan angka 1 sampai dengan n
- Dalam setiap baris setiap angka hanya muncul sekali
- Dalam setiap kolom setiap angka hanya muncul sekali
- *Grid* dibagi ke dalam *cage*
- *Cage* adalah sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel dengan angka tujuan dan operator yang telah ditentukan
- 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*

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*.

Operator-Operator Matematika

- Ada 5 kemungkinan operator:
  - $+$ (penjumlahan)
  - - (pengurangan)
  - $\times$ (perkalian)
  - $\div$ (pembagian)
  - $=$ (sama dengan)
- Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuruan *cage* harus berukuran dua sel

Operator-Operator Matematika
- Ada 5 kemungkinan operator:
  - + (penjumlahan)
  - - (pengurangan)
  - × (perkalian)
  - ÷ (pembagian)
  - = (sama dengan)
- Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuruan cage harus berukuran dua sel

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.

Contoh Permainan

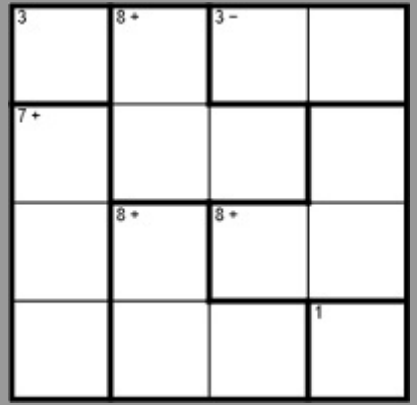


Figure 1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan.

Contoh Permainan

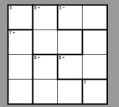


Figure 1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan.

Contoh Solusi

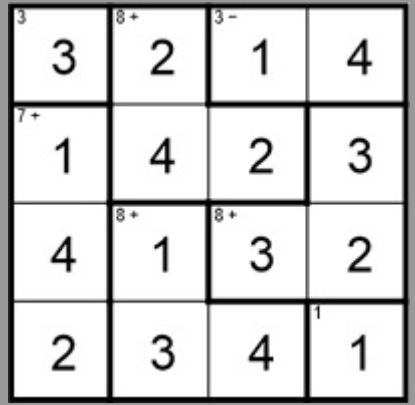


Figure 2: Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 1.

Contoh Solusi
3 2 1 4
1 4 2 3
4 1 3 2
2 3 4 1
Figure 2: Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 1.

Permasalahan Utama dalam Menyelesaikan Calcudoku

- Untuk menyelesaikan sebuah teka-teki Calcudoku, pemain harus pertama-tama memahami dua permasalahan utama dari teka-teki ini, yaitu:
  - Angka-angka mana yang harus dimasukkan ke dalam sebuah *cage*
  - Dalam urutan apa angka-angka tersebut harus dimasukkan ke dalam sebuah *cage*
- 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*).

Permasalahan Utama dalam Menyelesaikan Calcudoku

- Untuk menyelesaikan sebuah teka-teki Calcudoku, pemain harus pertama-tama memahami dua permasalahan utama dari teka-teki ini, yaitu:
  - Angka-angka mana yang harus dimasukkan ke dalam sebuah cage
  - Dalam urutan apa angka-angka tersebut harus dimasukkan ke dalam sebuah cage
- 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*).

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*
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*).

Tahapan Pengisian Calcudoku

- Dalam pengisian teka-teki ini ada dua tahapan, yaitu:
  - Mencari *cage* yang hanya berukuran 1 sel
  - Mencari mencari *cage* yang hanya mempunyai satu kemungkinan kombinasi angka

Tahapan Pengisian Calcudoku
Dalam pengisian teka-teki ini ada dua tahapan, yaitu:
Mencari cage yang hanya berukuran 1 sel
Mencari mencari cage yang hanya mempunyai satu kemungkinan kombinasi angka

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 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

Tahapan Pengisian Calcudoku

- 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.

Tahapan Pengisian Calcudoku

- 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.

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.

Mendefinisikan Permasalahan Calcudoku

- 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 $C_{b,k} = bn + k$
- 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 $H_i \in N$

Mendefinisikan Permasalahan Calcudoku

- 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 $C_{b,k} = bn + k$
- 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 $H_i \in N$

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 $H_i \in N$.

Mendefinisikan Permasalahan Calcudoku

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

Menurut Johanna, Lukas, dan Saputra, tiga aturan dalam mendefinisikan masalah dalam Calcudoku adalah sebagai berikut:

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*

Mendefinisikan Permasalahan Calcudoku

- Tujuan dari teka-teki ini adalah untuk mencari nilai $V(C_{b,k})$ dan memenuhi persyaratan berikut
- $|A_i| = 1 \wedge C_{b,k} \in A_i \rightarrow V(C_{b,k}) = H_i$, artinya jika sel adalah bagian dari sebuah *cage* yang jumlah selnya 1, maka nilai dari sel tersebut adalah angka tujuan dari *cage* tersebut
- $O_i \in \{-\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow |V(C_{a,b}) - V(C_{p,q})| = H_i$, artinya nilai absolut dari hasil pengurangan nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut
- $O_i \in \{\div\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow V(C_{a,b})/V(C_{p,q}) = H_i$, artinya nilai dari hasil pembagian nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut
- $O_i \in \{+\} \rightarrow \sum_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya nilai dari hasil penjumlahan dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut
- $O_i \in \{\times\} \rightarrow \prod_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya nilai dari hasil perkalian dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut

Mendefinisikan Permasalahan Calcudoku
- Tujuan dari teka-teki ini adalah untuk mencari nilai $V(C_{b,k})$ dan memenuhi persyaratan berikut
- $|A_i| = 1 \wedge C_{b,k} \in A_i \rightarrow V(C_{b,k}) = H_i$, artinya jika sel adalah bagian dari sebuah cage yang jumlah selnya 1, maka nilai dari sel tersebut adalah angka tujuan dari cage tersebut
- $O_i \in \{-\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow |V(C_{a,b}) - V(C_{p,q})| = H_i$, artinya nilai absolut dari hasil pengurangan nilai kedua sel di dalam cage tersebut adalah angka tujuan dari cage tersebut
- $O_i \in \{\div\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow V(C_{a,b})/V(C_{p,q}) = H_i$, artinya nilai dari hasil pembagian nilai kedua sel di dalam cage tersebut adalah angka tujuan dari cage tersebut
- $O_i \in \{+\} \rightarrow \sum_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya nilai dari hasil penjumlahan dari nilai semua sel di dalam cage tersebut adalah angka tujuan dari cage tersebut
- $O_i \in \{\times\} \rightarrow \prod_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya nilai dari hasil perkalian dari nilai semua sel di dalam cage tersebut adalah angka tujuan dari cage tersebut

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

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

Algoritma *Backtracking*

- 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*
- 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* 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 terbukti efektif untuk menyelesaikan banyak permainan logika karena algoritma itu terutama berguna untuk menyelesaikan masalah-masalah *constraint satisfaction*, di mana sekumpulan objek harus memenuhi sejumlah batasan

Algoritma *Backtracking*

- Pertama kali diperkenalkan pada tahun 1950 oleh D.H. Lehmer sebagai perbaikan algoritma *brute force*
- Algoritma ini terbukti efektif untuk menyelesaikan banyak permainan logika karena algoritma itu terutama berguna untuk menyelesaikan masalah-masalah *constraint satisfaction*, di mana sekumpulan objek harus memenuhi sejumlah batasan

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.

Sifat-Sifat Umum Algoritma *Backtracking*

- Implementasi algoritma *backtracking* memiliki beberapa sifat umum, yaitu:
  - Ruang solusi (*solution space*)
  - Fungsi pembangkit (*generating function*)
  - Fungsi pembatas (*generating function*)

Sifat-Sifat Umum Algoritma *Backtracking*
- Implementasi algoritma backtracking memiliki beberapa sifat umum, yaitu:
  - Ruang solusi (solution space)
  - Fungsi pembangkit (generating function)
  - Fungsi pembatas (generating function)

Ruang Solusi

- 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

Ruang Solusi
- 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

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.

Fungsi Pembangkit

- 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

Fungsi Pembangkit

- 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

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.

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

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

Fungsi pembangkit X_k 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

- Disusun dalam sebuah struktur berbentuk pohon (*tree*)
- Setiap simpul (*node*) merepresentasikan keadaan masalah
- Setiap sisi (*edge*) diberi label x_i
- Jalur dari akar (*root*) ke daun (*leaf*) merepresentasikan sebuah jawaban yang mungkin
- Semua jalur yang dikumpulkan bersama-sama membentuk ruang solusi
- Struktur pohon ini disebut sebagai *state space tree*

Ruang Solusi
- Disusun dalam sebuah struktur berbentuk pohon (*tree*)
- Setiap simpul (*node*) merepresentasikan keadaan masalah
- Setiap sisi (*edge*) diberi label x_i
- Jalur dari akar (*root*) ke daun (*leaf*) merepresentasikan sebuah jawaban yang mungkin
- Semua jalur yang dikumpulkan bersama-sama membentuk ruang solusi
- Struktur pohon ini disebut sebagai *state space tree*

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 3 menggambarkan contoh sebuah *state space tree*.

Ruang Solusi

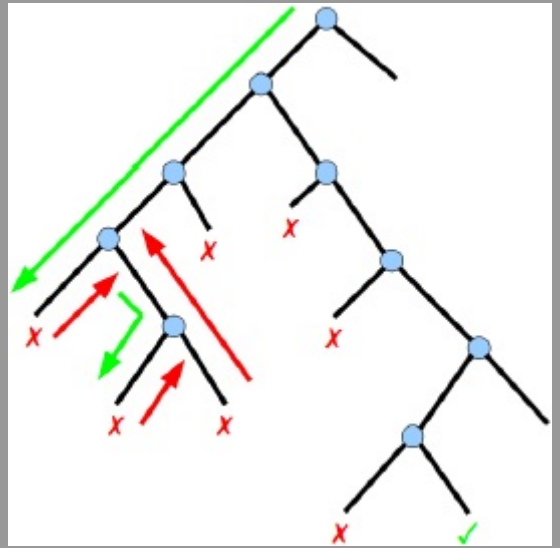


Figure 3: Ilustrasi *State space tree* yang digunakan dalam algoritma *backtracking*

Ruang Solusi

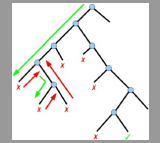


Figure 3: Ilustrasi State space tree yang digunakan dalam algoritma backtracking

Langkah-Langkah Penggunaan *State Space Tree*

- Langkah-langkah dalam menggunakan *state space tree* untuk mencari solusi adalah:
  - Solusi dicari dengan membangun jalur dari akar ke daun menggunakan algoritma DFS
  - Simpul yang terbentuk disebut sebagai simpul hidup (*live nodes*)
  - Simpul yang sedang diperluas disebut sebagai *expand nodes* atau *E-nodes*
  - Setiap kali sebuah *E-node* sedang diperluas, jalur yang dikembangkannya menjadi lebih panjang
  - Jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka *E-node* dimatikan dan menjadi simpul mati (*dead node*)

Langkah-Langkah Penggunaan *State Space Tree*
- Langkah-langkah dalam menggunakan *state space tree* untuk mencari solusi adalah:
  - Solusi dicari dengan membangun jalur dari akar ke daun menggunakan algoritma DFS
  - Simpul yang terbentuk disebut sebagai simpul hidup (*live nodes*)
  - Simpul yang sedang diperluas disebut sebagai *expand nodes* atau *E-nodes*
  - Setiap kali sebuah *E-node* sedang diperluas, jalur yang dikembangkannya menjadi lebih panjang
  - Jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka *E-node* dimatikan dan menjadi simpul mati (*dead node*)

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

- Solusi dicari dengan membangun jalur dari akar ke daun menggunakan algoritma DFS.
- Simpul yang terbentuk disebut sebagai simpul hidup (*live nodes*).
- Simpul yang sedang diperluas disebut sebagai *expand nodes* atau *E-nodes*.
- Setiap kali sebuah *E-node* sedang diperluas, jalur yang dikembangkannya menjadi lebih panjang.
- Jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka *E-node* dimatikan dan menjadi simpul mati (*dead node*).

Langkah-Langkah Penggunaan *State Space Tree*

- Lanjutan dari slide sebelumnya:
  - Fungsi yang digunakan untuk mematikan *E-node* adalah implementasi dari fungsi pembatas
  - Simpul mati tidak akan diperluas
  - Jika jalur yang sedang dibangun berakhir dengan simpul mati, proses akan mundur ke simpul sebelumnya
  - Simpul sebelumnya terus membangkitkan simpul anak (*child node*) lainnya, yang kemudian menjadi *E-node* baru
  - Pencarian selesai jika simpul tujuan tercapai

Langkah-Langkah Penggunaan *State Space Tree*
- Lanjutan dari slide sebelumnya:
- Fungsi yang digunakan untuk mematikan *E-node* adalah implementasi dari fungsi pembatas
- Simpul mati tidak akan diperluas
- Jika jalur yang sedang dibangun berakhir dengan simpul mati, proses akan mundur ke simpul sebelumnya
- Simpul sebelumnya terus membangkitkan simpul anak (*child node*) lainnya, yang kemudian menjadi *E-node* baru
- Pencarian selesai jika simpul tujuan tercapai

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

- Fungsi yang digunakan untuk mematikan *E-node* adalah implementasi dari fungsi pembatas.
- Simpul mati tidak akan diperluas.
- Jika jalur yang sedang dibangun berakhir dengan simpul mati, proses akan mundur ke simpul sebelumnya.
- Simpul sebelumnya terus membangkitkan simpul anak (*child node*) lainnya, yang kemudian menjadi *E-node* baru.
- Pencarian selesai jika simpul tujuan tercapai.

Kompleksitas Waktu

- 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

Kompleksitas Waktu

- 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

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.

Ruang Solusi

- 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*)
  - Fungsi pemeriksa *grid* (*grid checking*)

Ruang Solusi

- 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*)
  - Fungsi pemeriksa *grid* (*grid checking*)

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 Pembatas

- Fungsi pemeriksa kolom menghasilkan nilai *true* jika x_k belum ada di dalam kolom dan menghasilkan nilai *false* jika sebaliknya
- Fungsi pemeriksa baris menghasilkan nilai *true* jika x_k belum ada di dalam baris dan menghasilkan nilai *false* jika sebaliknya
- Fungsi pemeriksa *grid* memeriksa operator pada *grid* dan memeriksa berdasarkan operator yang telah ditentukan

Fungsi Pembatas
- Fungsi pemeriksa kolom menghasilkan nilai *true* jika x_k belum ada di dalam kolom dan menghasilkan nilai *false* jika sebaliknya
- Fungsi pemeriksa baris menghasilkan nilai *true* jika x_k belum ada di dalam baris dan menghasilkan nilai *false* jika sebaliknya
- Fungsi pemeriksa *grid* memeriksa operator pada *grid* dan memeriksa berdasarkan operator yang telah ditentukan

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.

Operator-Operator untuk Fungsi Pemeriksa *Grid*

- Ada 5 operator yang digunakan dalam fungsi ini, yaitu:
  - 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 sebaliknya
  - 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 sebaliknya

Ada 5 operator yang digunakan dalam fungsi ini, yaitu:

- 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.
- 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.

Operator-Operator untuk Fungsi Pemeriksa *Grid*

- Lanjutan dari slide sebelumnya:
  - 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 sebaliknya
  - 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 sebaliknya
  - Operator $=$, fungsi akan menghasilkan nilai *true* jika x_k sama dengan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya

Operator-Operator untuk Fungsi Pemeriksa *Grid*
- Lanjutan dari slide sebelumnya:
  - 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 sebaliknya
  - 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 sebaliknya
  - Operator $=$, fungsi akan menghasilkan nilai *true* jika x_k sama dengan nilai tujuan, dan menghasilkan nilai *false* jika sebaliknya

Ada 5 operator yang digunakan dalam fungsi ini, yaitu:

- 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.
- 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.
- 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

- *State space tree* bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan
- 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

State Space Tree

- *State space tree* bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan
- 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

State space tree bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan. 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.

Cara Kerja Algoritma *Backtracking* Secara Singkat

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

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

Cara Kerja Algoritma *Backtracking* Secara Singkat
Singkatnya, langkah-langkah dasar dari implementasi algoritma *backtracking* dapat dijelaskan sebagai berikut:
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

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

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.

Algoritma *Hybrid Genetic*

- 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

Dalam kasus ini, algoritma ini 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.

Algoritma *Rule Based*

- Sebuah algoritma berbasis aturan logika untuk menyelesaikan 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*
  - *X-wing*

Algoritma *Rule Based*
- Sebuah algoritma berbasis aturan logika untuk menyelesaikan 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*
  - *X-wing*

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
- Nilai dari sel tersebut sama dengan angka tujuan yang telah ditentukan

Aturan *Single Square*
- Digunakan jika sebuah *cage* hanya berisi satu sel
- Nilai dari sel tersebut sama dengan angka tujuan yang telah ditentukan

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 $n \geq 2$.
- Sel-sel lainnya dalam baris dan kolom tersebut tidak mungkin diisi dengan nilai yang sama dengan nilai milik n sel tersebut.

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 $n \geq 2$.
- Sel-sel lainnya dalam baris dan kolom tersebut tidak mungkin diisi dengan nilai yang sama dengan nilai milik n sel tersebut.

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.

Aturan *Naked Subset*

Figure 4: Contoh bagaimana cara mendeteksi aturan *naked pair*

- Gambar 4 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

Gambar 4 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
- 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.

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
- 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.

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.

Aturan *Evil Twin*

Figure 5: Contoh aturan *evil twin*

- Contohnya, pada Gambar 5, begitu sel di sudut kiri bawah diisi oleh angka 4, maka sel diatasnya harus diisi oleh angka 9.

Contohnya, pada Gambar 5, begitu sel di sudut kiri bawah diisi oleh angka 4, maka sel diatasnya harus diisi oleh angka 9.

Aturan *Hidden Single*

Figure 6: Contoh aturan *hidden single*

- Digunakan jika sebuah angka hanya bisa diisikan dalam satu sel dalam sebuah baris atau kolom.
- Pada Gambar 6, 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 kiri tersebut yang memiliki kemungkinan nilai 7
- Ini disebut sebagai *hidden single*
- Sel tersebut harus diisi dengan angka 7.

Aturan *Hidden Single*
Figure 6: Contoh aturan *hidden single*
- Digunakan jika sebuah angka hanya bisa diisikan dalam satu sel dalam sebuah baris atau kolom.
- Pada Gambar 6, 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 kiri tersebut yang memiliki kemungkinan nilai 7
- Ini disebut sebagai *hidden single*
- Sel tersebut harus diisi dengan angka 7.

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

Aturan *Killer Combination*

- 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
- 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

Aturan *Killer Combination*
- 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
- 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

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.

Aturan *Killer Combination*

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

Figure 7: Contoh aturan *killer combination* untuk *cage* dengan ukuran 2 sel

Aturan *Killer Combination*

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

Figure 7: Contoh aturan *killer combination* untuk *cage* dengan ukuran 2 sel

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
- Juga digunakan 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

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
- Juga digunakan 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

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.

Aturan *X-Wing*

Figure 8: Contoh aturan *X-wing* [2]

Aturan *X-Wing*

- Gambar 8 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

Aturan *X-Wing*
- Gambar 8 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

Gambar 8 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.

Heuristik

- Semacam aturan tidak tertulis yang mungkin menghasilkan solusi
- Kadang-kadang efektif, tetapi tidak dijamin akan berhasil dalam setiap kasus
- Memerankan peran penting dalam strategi pencarian karena sifat eksponensial dari kebanyakan masalah
- Membantu mengurangi jumlah alternatif solusi dari angka yang bersifat eksponensial menjadi angka yang bersifat polinomial

Heuristik

- Semacam aturan tidak tertulis yang mungkin menghasilkan solusi
- Kadang-kadang efektif, tetapi tidak dijamin akan berhasil dalam setiap kasus
- Memerankan peran penting dalam strategi pencarian karena sifat eksponensial dari kebanyakan masalah
- Membantu mengurangi jumlah alternatif solusi dari angka yang bersifat eksponensial menjadi angka yang bersifat polinomial

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 membantu mengurangi jumlah alternatif solusi dari angka yang bersifat eksponensial menjadi angka yang bersifat polinomial.

Pencarian Heuristik

- Sebuah teknik pencarian kecerdasan buatan (*artifical intelligence*) yang menggunakan heuristik dalam langkah-langkahnya
- Contoh teknik pencarian heuristik adalah:
  - *Generate and Test*
  - *Hill Climbing*
  - *Best First Search*

Pencarian Heuristik
- Sebuah teknik pencarian kecerdasan buatan (*artifical intelligence*) yang menggunakan heuristik dalam langkah-langkahnya
- Contoh teknik pencarian heuristik adalah:
  - *Generate and Test*
  - *Hill Climbing*
  - *Best First Search*

Pencarian heuristik adalah sebuah teknik pencarian kecerdasan buatan (*artifical intelligence*) yang menggunakan heuristik dalam langkah-langkahnya. Contoh teknik pencarian heuristik adalah *Generate and Test*, *Hill Climbing*, dan *Best First Search*.

Algoritma Genetik

- Salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam
- Perpaduan dari bidang biologi dan ilmu komputer.
- Algoritma ini memanipulasi informasi, biasanya disebut sebagai kromosom.

Algoritma Genetik

- Salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam
- Perpaduan dari bidang biologi dan ilmu komputer.
- Algoritma ini memanipulasi informasi, biasanya disebut sebagai kromosom.

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

- Meng-*encode* kemungkinan jawaban untuk sebuah masalah yang diberikan.
- Dievaluasi dan diberi *fitness value* berdasarkan seberapa baikkah kromosom dalam menyelesaikan masalah yang diberikan berdasarkan kriteria yang ditentukan
- Nilai kelayakan ini digunakan sebagai probabilitas kebertahanan hidup kromosom dalam satu siklus reproduksi
- *Child chromosome* diproduksi dengan menggabungkan dua (atau lebih) *parent chromosome*
- Proses ini dirancang untuk menghasilkan kromosom-kromosom keturunan yang lebih layak
- Kromosom-kromosom ini meng-*encode* jawaban yang lebih baik, sampai solusi yang baik dan yang bisa diterima ditemukan.

Kromosom
- Meng-*encode* kemungkinan jawaban untuk sebuah masalah yang diberikan.
- Dievaluasi dan diberi *fitness value* berdasarkan seberapa baikkah kromosom dalam menyelesaikan masalah yang diberikan berdasarkan kriteria yang ditentukan
- Nilai kelayakan ini digunakan sebagai probabilitas kebertahanan hidup kromosom dalam satu siklus reproduksi
- *Child chromosome* diproduksi dengan menggabungkan dua (atau lebih) *parent chromosome*
- Proses ini dirancang untuk menghasilkan kromosom-kromosom keturunan yang lebih layak
- Kromosom-kromosom ini meng-*encode* jawaban yang lebih baik, sampai solusi yang baik dan yang bisa diterima ditemukan.

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 meng-*encode* jawaban yang lebih baik, sampai solusi yang baik dan yang bisa diterima ditemukan.

Cara Kerja Algoritma Genetik

- Cara kerja algoritma genetik adalah sebagai berikut:
  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 algoritm.

Cara Kerja Algoritma Genetik
- Cara kerja algoritma genetik adalah sebagai berikut:
  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 algoritm.

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

- Langkah-langkah utama dalam penggunaan algoritma genetik adalah:
  - Membangkitkan populasi kemungkinan jawaban
  - Mencari fungsi objektif dan fungsi kelayakan
  - Penggunaan operator genetik

Langkah-Langkah Utama dalam Penggunaan Algoritma Genetik
- Langkah-langkah utama dalam penggunaan algoritma genetik adalah:
  - Membangkitkan populasi kemungkinan jawaban
  - Mencari fungsi objektif dan fungsi kelayakan
  - Penggunaan operator genetik

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

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 1 di slide berikutnya, 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 slide berikutnya
- Aturan *naked pair*, salah satu dari aturan jenis *naked subset*, dinyatakan dalam persamaan 4 untuk baris dan persamaan 5 untuk kolom

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 1 di slide berikutnya, 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 slide berikutnya
- Aturan *naked pair*, salah satu dari aturan jenis *naked subset*, dinyatakan dalam persamaan 4 untuk baris dan persamaan 5 untuk kolom

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 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]

Algoritma *Hybrid Genetic*

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

Algoritma *Hybrid Genetic*

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

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.

Algoritma *Hybrid Genetic*

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 *Hybrid Genetic*
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

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 *Hybrid Genetic*

- 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

Algoritma *Hybrid Genetic*
- 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

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.

Pemodelan 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
- Dalam sebuah kromosom, segmen diurutkan dari baris yang paling atas ke baris yang paling bawah
- Setiap segmen merepresentasikan sebuah baris yang belum terselesaikan

Pemodelan 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
- Dalam sebuah kromosom, segmen diurutkan dari baris yang paling atas ke baris yang paling bawah
- Setiap segmen merepresentasikan sebuah baris yang belum terselesaikan

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. Setiap segmen merepresentasikan sebuah baris yang belum terselesaikan.

Pemodelan Kromosom

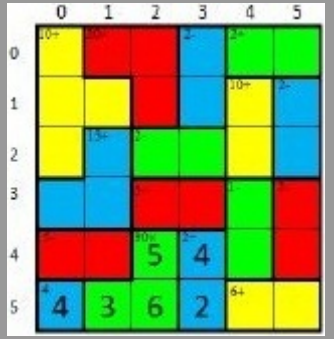


Figure 9: Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 6 x 6

Contoh salah satu kromosom dari permainan teka-teki Calcudoku pada Gambar 9 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

Contoh, salah satu kromosom dari permainan teka-teki Calcudoku pada Gambar 9 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.

Fungsi Objektif

- 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 ($w_j \neq 0$)

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 ($w_j \neq 0$).

Fungsi Kelayakan

- Untuk kromosom dengan jumlah gen k, fungsi kelayakan, yaitu hasil penjumlahan dari hasil fungsi objektif untuk setiap gen dibagi dengan jumlah gen, dinyatakan dalam persamaan di bawah ini:

$$x_j = \begin{cases} 0, w_j = 0 \\ 1, w_j \neq 0 \end{cases}$$

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

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

Fungsi Kelayakan

- Untuk kromosom dengan jumlah gen k, fungsi kelayakan, yaitu hasil penjumlahan dari hasil fungsi objektif untuk setiap gen dibagi dengan jumlah gen, dinyatakan dalam persamaan di bawah ini:

$$x_j = \begin{cases} 0, w_j = 0 \\ 1, w_j \neq 0 \end{cases}$$

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

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

Fungsi kelayakan, yaitu hasil penjumlahan dari hasil fungsi objektif untuk setiap gen dibagi dengan jumlah gen, dinyatakan dalam persamaan di bawah ini [2]:

$$x_j = \begin{cases} 0, w_j = 0 \\ 1, w_j \neq 0 \end{cases}$$

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

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

Kawin Silang

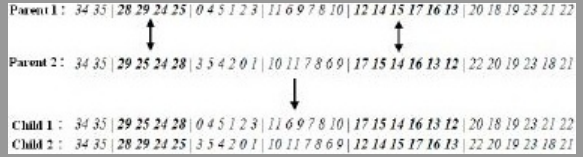


Figure 10: Contoh proses kawin silang antara dua kromosom [2]

- Dua kromosom, yaitu kromosom orang tua, disilangkan untuk membuat dua kromosom yang baru, yaitu kromosom anak, dengan metodologi kawin silang *N-segments*
- Gambar 10 menggambarkan contoh proses kawin silang antara dua kromosom.

Kawin Silang
Figure 10: Contoh proses kawin silang antara dua kromosom [2]
- Dua kromosom, yaitu kromosom orang tua, disilangkan untuk membuat dua kromosom yang baru, yaitu kromosom anak, dengan metodologi kawin silang *N-segments*
- Gambar 10 menggambarkan contoh proses kawin silang antara dua kromosom.

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 10 menggambarkan contoh proses kawin silang antara dua kromosom.

Mutasi

Figure 11: Contoh proses mutasi [2]

- Digunakan untuk mendapatkan kemungkinan kromosom yang lain
- Dilakukan di antara gen yang berada dalam segmen yang sama
- Gambar 11 adalah contoh proses mutasi antara dua gen dalam segmen yang sama

Pertukaran mutasi digunakan untuk mendapatkan kemungkinan kromosom yang lain. Mutasi dilakukan di antara gen yang berada dalam segmen yang sama. Gambar 11 adalah contoh proses mutasi antara dua gen dalam segmen yang sama.

Cara Kerja Algoritma *Hybrid Genetic*

- Cara kerja algoritma *hybrid genetic* menurut Johanna, Lukas, dan Saputra adalah sebagai berikut:
  - Masukkan teka-teki yang akan diselesaikan sebagai input.
  - Program akan merepresentasikan input yang dimasukkan dalam format teka-teki.
  - Program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* terlebih dahulu.
  - Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based*, maka algoritma selesai.
  - Jika program gagal dengan menggunakan algoritma *rule based*, maka program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik.
  - Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik, maka algoritma selesai.
  - Jika program gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik, artinya algoritma gagal dalam menyelesaikan teka-teki terseebut.

Cara Kerja Algoritma *Hybrid Genetic*
- Cara kerja algoritma *hybrid genetic* menurut Johanna, Lukas, dan Saputra adalah sebagai berikut:
  - Masukkan teka-teki yang akan diselesaikan sebagai input.
  - Program akan merepresentasikan input yang dimasukkan dalam format teka-teki.
  - Program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* terlebih dahulu.
  - Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based*, maka algoritma selesai.
  - Jika program gagal dengan menggunakan algoritma *rule based*, maka program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik.
  - Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik, maka algoritma selesai.
  - Jika program gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik, artinya algoritma gagal dalam menyelesaikan teka-teki terseebut.

Cara kerja algoritma *hybrid genetic* adalah sebagai berikut [2]:

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

Alur Cara Kerja Algoritma *Hybrid Genetic*

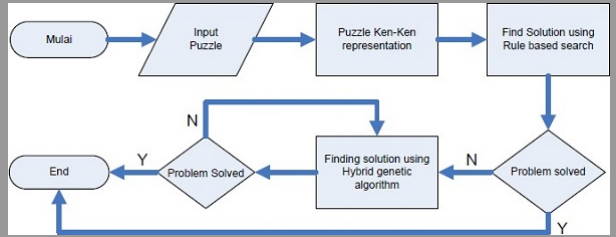


Figure 12: Alur penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic*

- Alur (*flow chart*) penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* dapat dilihat di Gambar 12.

Alur Cara Kerja Algoritma *Hybrid Genetic*
Alur (*flow chart*) penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* dapat dilihat di Gambar 12.

Analisis Algoritma *Backtracking*

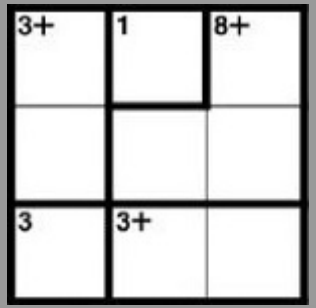


Figure 13: Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 3 x 3

- Untuk mengilustrasikan cara kerja algoritma *backtracking*, teka-teki Calcudoku yang digambarkan pada Gambar 13 akan digunakan.

Analisis Algoritma *Backtracking*
3+ 1 8+
3 3+
Figure 13: Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 3 x 3
Untuk mengilustrasikan cara kerja algoritma *backtracking*, teka-teki Calcudoku yang digambarkan pada Gambar 13 akan digunakan.

Untuk mengilustrasikan cara kerja algoritma *backtracking*, teka-teki Calcudoku yang digambarkan pada Gambar 13 akan digunakan.

Analisis Algoritma *Backtracking*

- Algoritma *backtracking* dimulai dengan teka-teki yang belum diselesaikan (*state* 1).
- Algoritma mengisikan sel pada baris ke-1 dan kolom ke-1 dengan angka 1 (*state* 2). Algoritma lalu maju ke sel berikutnya.
- Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-2 dengan angka 1 (*state* 3), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 4), tetapi angka 2 tidak sesuai dengan angka tujuan dari *cage* tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 5), tetapi angka 3 tidak sesuai dengan angka tujuan dari *cage* tersebut.
- *State* 3, *state* 4, dan *state 5* digambarkan pada Gambar 14.

Analisis Algoritma *Backtracking*
- Algoritma backtracking dimulai dengan teka-teki yang belum diselesaikan (state 1).
- Algoritma mengisikan sel pada baris ke-1 dan kolom ke-1 dengan angka 1 (state 2). Algoritma lalu maju ke sel berikutnya.
- Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-2 dengan angka 1 (state 3), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (state 4), tetapi angka 2 tidak sesuai dengan angka tujuan dari cage tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (state 5), tetapi angka 3 tidak sesuai dengan angka tujuan dari cage tersebut.
- State 3, state 4, dan state 5 digambarkan pada Gambar 14.

- 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.
- 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.
- 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.
- 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 14 menggambarkan *state* 3, *state* 4, dan *state* 5 dalam penyelesaian teka-teki Calcudoku ini.

Analisis Algoritma *Backtracking*

Figure 14: Ilustrasi *state* 3, 4, dan 5 pada sebuah *grid* teka-teki Calcudoku

Analisis Algoritma *Backtracking*

Analisis Algoritma *Backtracking*

- Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-2 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 1). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 6). Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-1 dan kolom ke-2 dengan angka 1 (*state* 7), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut. Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state* 8), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 9), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

Analisis Algoritma *Backtracking*
- Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-2 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 1). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 6). Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-1 dan kolom ke-2 dengan angka 1 (*state* 7), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut. Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state* 8), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 9), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

- 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.
- 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.
- Angka 1 (*state* 8) gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan pada baris tersebut.
- Angka 2 (*state* 9) juga gagal dalam pemeriksaan baris karena angka 2 sudah pernah digunakan pada baris tersebut.

Analisis Algoritma *Backtracking*

- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 10). Algoritma telah selesai mengisikan baris ke-1, sehingga bisa maju ke baris berikutnya.
- Algoritma mengisikan sel pada baris ke-2 dan kolom ke-1 dengan angka 1 (*state* 11), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut. Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-2 dan kolom ke-2 dengan angka 1 (*state* 12), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 13). Algoritma lalu maju ke sel berikutnya.

Analisis Algoritma *Backtracking*
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (state 10). Algoritma telah selesai mengisikan baris ke-1, sehingga bisa maju ke baris berikutnya.
- Algoritma mengisikan sel pada baris ke-2 dan kolom ke-1 dengan angka 1 (state 11), dan ternyata hasilnya sesuai dengan angka tujuan dari cage tersebut. Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-2 dan kolom ke-2 dengan angka 1 (state 12), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (state 13). Algoritma lalu maju ke sel berikutnya.

- 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.
- 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.
- 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.
- 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.

Analisis Algoritma *Backtracking*

- Algoritma mengisikan sel pada baris ke-2 dan kolom ke-3 dengan angka 1 (*state* 14), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 15), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 16), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.
- Karena semua kemungkinan angka untuk baris ke-2 dan kolom ke-3 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state* 13). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 17). Algoritma lalu maju ke sel berikutnya.

- Algoritma mengisikan x_6 dengan angka 1 (*state* 14), tetapi gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan pada baris tersebut.
- 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.
- 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.
- 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* 17), dan ternyata angka 3 berlaku sebagai x_5, sehingga algoritma bisa maju ke sel berikutnya, yaitu x_6.

Analisis Algoritma *Backtracking*

- Algoritma mengisikan sel pada baris ke-2 dan kolom ke-3 dengan angka 1 (*state* 18), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 19), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti digambarkan pada Gambar 15. Algoritma telah selesai mengisikan baris ke-1, sehingga bisa maju ke baris berikutnya.

- Algoritma lalu mencoba angka 1 (*state* 18) sebagai x_6, tetapi gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan dalam baris tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 19), dan ternyata angka 2 berlaku. Algoritma telah selesai mengisi baris ke-2. Gambar 15 menggambarkan *state* 19 dalam penyelesaian teka-teki Calcudoku ini.

Analisis Algoritma *Backtracking*

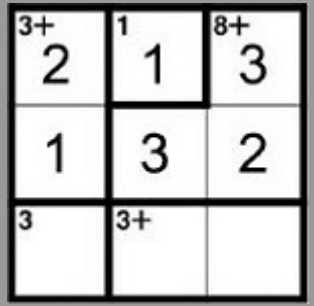


Figure 15: Ilustrasi *state* 19 pada sebuah *grid* teka-teki Calcudoku

Analisis Algoritma *Backtracking*
3+ 2 | 1 1 | 8+ 3
1 | 1 3 | 2
3 | 3+ |
Figure 15: Ilustrasi *state* 19 pada sebuah *grid* teka-teki Calcudoku

Analisis Algoritma *Backtracking*

- Algoritma mengisikan sel pada baris ke-3 dan kolom ke-1 dengan angka 1 (*state* 20), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 21), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 22), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut. Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-3 dan kolom ke-2 dengan angka 1 (*state* 23), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.

Analisis Algoritma *Backtracking*
- Algoritma mengisikan sel pada baris ke-3 dan kolom ke-1 dengan angka 1 (*state* 20), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 21), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 22), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut. Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-3 dan kolom ke-2 dengan angka 1 (*state* 23), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.

- 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.
- 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.
- 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.
- 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.

Analisis Algoritma *Backtracking*

- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 24). Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state* 25), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti digambarkan pada Gambar 16. Algoritma *backtracking* telah selesai mengisi semua sel dalam permainan teka-teki Calcudoku ini dengan benar. *State space tree* ini telah mencapai simpul tujuannya, yaitu simpul 25, dengan jalur 2-1-3-1-3-2-3-2-1, seperti digambarkan pada Gambar 17.

Analisis Algoritma *Backtracking*
- Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 24). Algoritma lalu maju ke sel berikutnya.
- Algoritma mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state* 25), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti digambarkan pada Gambar 16. Algoritma *backtracking* telah selesai mengisi semua sel dalam permainan teka-teki Calcudoku ini dengan benar. *State space tree* ini telah mencapai simpul tujuannya, yaitu simpul 25, dengan jalur 2-1-3-1-3-2-3-2-1, seperti digambarkan pada Gambar 17.

- 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.
- 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 16 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 17. *State space tree* ini telah mencapai simpul tujuannya, yaitu simpul 25, dengan jalur 2-1-3-1-3-2-3-2-1.

Analisis Algoritma *Backtracking*

Figure 16: *State* 25, simpul tujuan, sebagai hasil yang dicapai

Analisis Algoritma *Backtracking*

Figure 16: *State* 25, simpul tujuan, sebagai hasil yang dicapai

Analisis Algoritma *Backtracking*

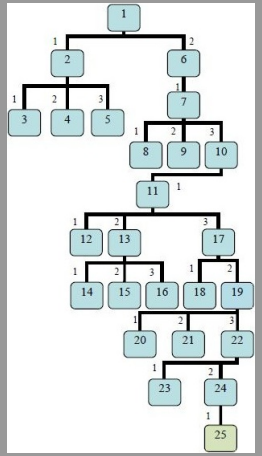
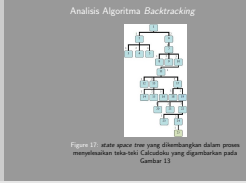


Figure 17: *state space tree* yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 13



Daftar Pustaka

- Asanilta Fahda, *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 2015.
- Olivia Johanna, Samuel Lukas, Kie Van Ivanky Saputra, *Solving and Modeling Ken-ken Puzzle by Using Hybrid Genetics Algorithm*, 1st International Conference on Engineering and Technology Development (ICETD 2012), Faculty of Engineering and Faculty of Computer Science, Bandar Lampung University, 2012.

Daftar Pustaka

Asanilta Fahda, *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 2015.

Olivia Johanna, Samuel Lukas, Kie Van Ivanky Saputra, *Solving and Modeling Ken-ken Puzzle by Using Hybrid Genetics Algorithm*, 1st International Conference on Engineering and Technology Development (ICETD 2012), Faculty of Engineering and Faculty of Computer Science, Bandar Lampung University, 2012.