Jurnal EurekaMatika 13(2) (2025) 149-160

Jurnal EurekaMatika

Journal homepage: https://ejournal.upi.edu/index.php/JEM

Implementasi Gabungan Algoritma *Backtracking* dan *Simulated Annealing* untuk Menyelesaikan *Puzzle* Sudoku

Silmi Nur Jannah, Khusnul Novianingsih, dan Ririn Sispiyati*

Program Studi Matematika, Fakultas Pendidikan Matematika dan Ilmu Pengetahuan Alam, Universitas Pendidikan Indonesia

*Correspondence: E-mail: k_novianingsih@upi.edu

A B S T R A K

Puzzle Sudoku adalah salah satu jenis permainan populer yang dapat dipandang sebagai permasalahan kombinatorial. Penelitian ini merancang penyelesaian permainan Sudoku dengan cara mengimplementasikan gabungan Algoritma *Backtracking* dan *Simulated Annealing*. Algoritma *Backtracking* digunakan untuk membangkitkan solusi awal permainan. Solusi ini selanjutnya diperbaiki oleh Algoritma Simulated Annealing melalui pembangkitan solusi tetangga dengan menukar isi sel selain angka petunjuk dan menurunkan temperatur secara bertahap hingga tidak ada pelanggaran yang tersisa. Hasil pengujian menunjukkan bahwa gabungan kedua algoritma ini mampu menyelesaikan *Puzzle* Sudoku berukuran hingga 20×20. Khusus untuk Sudoku berukuran 20×20, Algoritma yang diusulkan memerlukan waktu komputasi yang lebih singkat dalam menyelesaikan Sudoku dibandingkan dengan implementasi Algoritma *Backtracking* atau *Simulated Annealing* secara terpisah. Hasil ini menunjukkan bahwa gabungan algoritma *Backtracking* dan *Simulated Annealing* dapat digunakan sebagai alternatif yang efektif dalam menyelesaikan permainan Sudoku.

INFORMASI ARTIKEL

Sejarah Artikel:

Diterima 27 Juni 2025
Direvisi 20 September 2025
Disetujui 10 Oktober 2025
Tersedia online 2 November 2025
Dipublikasikan 2 November 2025

Kata Kunci:
Backtracking,
Optimisasi,
Simulated annealing,
Sudoku.

A B S T R A C T

A Sudoku puzzle is a popular type of game that can be viewed as a combinatorial problem. This study proposes a solution to Sudoku by hybridizing Backtracking and Simulated Annealing algorithms. The Backtracking algorithm is used to generate an initial solution for the puzzle. This solution is then improved using the Simulated Annealing algorithm, which creates neighboring solutions by swapping cell contents, excluding the clue numbers. The temperature in the algorithm is gradually decreased until no violations remain. Test results indicate that this hybrid of algorithms can effectively solve Sudoku puzzles up to 20x20 in size. Specifically, for the 20x20 puzzles, the proposed approach requires a shorter computation time compared to using the Backtracking or Simulated Annealing algorithms individually. These findings suggest that the hybrid of Backtracking and Simulated Annealing algorithms is an effective alternative for solving Sudoku puzzles.

Keywords:
Backtracking,
Optimization,
Simulated annealing,
Sudoku.

1. PENDAHULUAN

Permainan *puzzle* merupakan salah satu permainan yang memerlukan logika dan nalar dalam penyelesaiannya. Salah satunya adalah *puzzle* angka yang bernama Sudoku. Sudoku revisi singkatan Bahasa Jepang dari "*Suuji wa dokushin ni kagiru*", artinya "angka-angkanya harus tetap tunggal" (Yusuf & Hendra, 2013). Sudoku merupakan *puzzle* logika di mana pemain harus mengisi sebuah kotak dengan angka-angka 1 sampai *n* dengan kemunculan tepat satu kali pada setiap baris, kolom, dan sub-matriksnya. Teka-teki dimulai dengan sejumlah bilangan bulat yang diberikan pada beberapa sel sebagai petunjuk untuk menyelesaikan permainan sebagaimana dipaparkan oleh Harrysson & Laestander pada tahun 2014 dalam penelitiannya berjudul *Solving Sudoku Efficiently with Dancing Links* sebagai tugas akhir S1 di KTH Computer Science and Communication. Tingkat kesulitan dari *Puzzle* Sudoku didasarkan pada banyaknya angka petunjuk yang muncul ketika permainan dimulai. Semakin sedikit angka petunjuk yang ditunjukkan, maka semakin sulit *Puzzle* Sudoku ini untuk diselesaikan. Selain itu, tingkat kesulitan *puzzle* juga bergantung pada ukuran *puzzle* itu sendiri. Semakin besar *puzzle* yang dimainkan, maka semakin sulit *puzzle* diselesaikan.

Berbagai macam algoritma heuristik telah digunakan untuk menyelesaikan *puzzle* Sudoku dikarenakan Sudoku termasuk sebuah masalah *NP-complete* (Jana *et al*., 2015), yaitu masalah kombinatorial yang memerlukan waktu komputasi yang lama untuk diselesaikan dengan metode eksak. Gambar 1.1 merupakan contoh dari *Puzzle* Sudoku beserta solusinya yang diterbitkan oleh *The Times* yang ditunjukkan pada Santos-García & Palomino (2007):

		3	7	2		1		
		6	9		5	8		
4	9			1			5	2
	5						6	1
8		4				2		9
7	6						4	
2	3			5			1	7
		1	2		3	9		
		5		8	6	4		

5	8	**3**	**7**	**2**	4	**1**	9	6
1	2	**6**	**9**	3	**5**	**8**	7	4
4	**9**	7	6	**1**	8	3	**5**	**2**
3	**5**	9	8	4	2	7	**6**	**1**
8	1	**4**	5	6	7	**2**	3	**9**
7	**6**	2	3	9	1	5	**4**	8
2	**3**	8	4	**5**	9	6	**1**	**7**
6	4	**1**	**2**	7	**3**	**9**	8	5
9	7	**5**	1	**8**	**6**	**4**	2	3

Gambar 1. 1 *Puzzle* Sudoku dan Solusinya.

Dalam penelitiannya, Sari (2011) mengemukakan bahwa metode yang paling sederhana dan langsung untuk menyelesaikan *Puzzle* Sudoku adalah dengan menerapkan Algoritma *Brute Force*. Algoritma ini bekerja dengan cara mengenumerasi atau mencoba semua kemungkinan kombinasi angka yang dapat mengisi setiap sel kosong dalam *grid* Sudoku, yaitu angka 1 hingga 9. Proses ini dilakukan secara sistematis tanpa mempertimbangkan efisiensi, sehingga seluruh ruang solusi dijelajahi satu per satu hingga ditemukan konfigurasi yang memenuhi aturan permainan Sudoku. Meskipun pendekatan ini menjamin bahwa solusi akan ditemukan jika memang ada, kelemahannya terletak pada tingginya kompleksitas komputasi yang dihasilkan, terutama ketika jumlah sel kosong cukup banyak. Hal ini menyebabkan waktu eksekusi menjadi lama dan penggunaan sumber daya komputasi menjadi tidak efisien.

Untuk mengatasi keterbatasan tersebut, Sari menyarankan penggunaan Algoritma Backtracking sebagai alternatif yang lebih cerdas dan efisien. Algoritma Backtracking merupakan pengembangan dari Brute Force yang dilengkapi dengan mekanisme pengambilan keputusan yang lebih selektif. Dalam prosesnya, algoritma ini akan mencoba mengisi sel dengan angka yang memungkinkan, namun jika pada suatu titik ditemukan bahwa pilihan tersebut tidak mengarah pada solusi yang valid, maka algoritma akan mundur (*backtrack*) ke langkah sebelumnya dan mencoba alternatif lain. Dengan cara ini, pencarian

solusi menjadi lebih terarah karena hanya mempertimbangkan jalur yang berpotensi menghasilkan solusi akhir. Pendekatan ini tidak hanya mengurangi jumlah langkah pencarian yang diperlukan, tetapi juga meningkatkan efektivitas dalam menyelesaikan masalah Sudoku secara keseluruhan.

Kamal *et al*. (2015) mengusulkan deteksi digital dan deskripsi *Puzzle* Sudoku menggunakan teknik berbasis penglihatan dan pemecahan *puzzle* selanjutnya dengan tiga Algoritma *Backtracking*, *Simulated Annealing*, dan *Genetic Algorithms*. Hasil dari analisis perbandingan ketiga algoritma ini menunjukkan bahwa *Backtracking* memiliki kinerja terbaik di antara ketiga algoritma. Penelitian lain oleh Rahayu *et al*. (2017) fokus pada Algoritma *Brute Force*, *Backtracking*, dan *Simulated Annealing* untuk dianalisa waktu komputasinya ketika menyelesaikan *puzzle* Sudoku. Hasil yang diperoleh menunjukkan bahwa Algoritma *Backtracking* memiliki waktu komputasi lebih cepat dibandingkan dengan Algoritma *Brute Force*. Selanjutnya, ketika melakukan perbandingan waktu komputasi Algoritma *Backtracking* dan *Simulated Annealing*, diperoleh bahwa *Simulated Annealing* memiliki rata-rata waktu komputasi lebih kecil di level sulit. Hal ini menunjukkan bahwa *Simulated Annealing* lebih cepat dalam menyelesaikan *Puzzle* Sudoku pada tingkat kesulitan tinggi, meskipun keefektivitasan langkahnya bergantung pada solusi yang diperoleh sebelumnya.

Berdasarkan kelebihan dan kekurangan dalam penelitian-penelitian sebelumnya, penulis merancang metode penyelesaian permainan Sudoku melalui implementasi gabungan Algoritma *Backtracking* dan *Simulated Annealing*. Sejauh ini, belum ada penelitian terkait penyelesaian Sudoku dengan menggunakan gabungan kedua algoritma tersebut. Ide utama dari penggabungan ini adalah memanfaatkan Algoritma Backtracking untuk membangkitkan solusi awal yang digunakan dalam Simulated Annealing agar proses komputasi menjadi lebih cepat dan efektif. Hasil dari penelitian ini diharapkan dapat menjadi alternatif metode dalam penyelesaian *Puzzle* Sudoku maupun masalah lain yang relevan untuk diselesaikan melalui kombinasi kedua algoritma tersebut.

2. METODE

Pada bagian ini dibahas mengenai model *Puzzle* Sudoku secara umum dan teknik penyelesaiannya melalui algoritma gabungan *Backtracking* dan *Simulated Annealing.*

2.1. Model Optimisasi

Penyelesaian masalah *Puzzle* Sudoku akan dimodelkan sebagai model optimisasi berkendala tanpa fungsi tujuan karena hanya diperlukan solusi fisibel untuk menyelesaikan permainan. Didefinisikan himpunan berikut:

$K = \{1, \dots, n\}$

R: Sub matriks berukuran $r \times s$ dengan $r, s < n$ dan $r \times s = n$.

Didefinisikan variabel keputusan model sebagai berikut:

$$x_{i,j,k} = \begin{cases} 1, \text{jika bilangan } k \text{ menempati sel } (i,j) \\ 0, \text{yang lainnya.} \end{cases}$$

Variabel di atas digunakan untuk mengidentifikasi peletakkan angka pada baris dan kolom tertentu. Adapun kendala model optimasi masalah penyelesaian *Puzzle* Sudoku merupakan aturan permainan Sudoku. Kendala-kendala tersebut adalah sebagai berikut:

1. Setiap sel harus diisi sebuah angka.
   Kendala ini diformulasikan sebagai:

$$\sum_{k \in K} x_{i,j,k} = 1, i, j \in K.$$

2. Setiap baris harus berisi setiap angka.
   Kendala ini dirumuskan sebagai:
$$\sum_{j \in K} x_{i,j,k} = 1, i, k \in K$$

3. Setiap kolom harus berisi setiap angka

   Kendala ini dirumuskan sebagai:
$$\sum_{i \in K} x_{i,j,k} = 1, j, k \in K.$$

4. Setiap sub matriks $r \times s$ harus berisi semua angka

   Kendala ini dirumuskan sebagai:
$$\sum_{i=(i_0-1)r+1}^{ri_0} \sum_{j=(j_0-1)s+1}^{sj_0} x_{i,j,k} = 1, i_0 = 1, \dots, \frac{n}{r}, j_0 = 1, \dots, \frac{n}{s}.$$

Adapun batasan variabel model adalah sebagai berikut:

$$x_{i,j,k} \in \{0,1\}, \qquad i, j, k \in K.$$

2.2. Teknik Penyelesaian

Pada penelitian ini, masalah *Puzzle* Sudoku diselesaikan dengan menggunakan Algoritma *Backtracking* dalam mencari solusi awal dan *Simulated Annealing* dalam mencari solusi optimal. Barikut adalah tahapan penyelesaian masalah:

1. Inisiasi Parameter Input

   Menurut Panggabean (2004) parameter-parameter yang terlibat untuk mencapai konvergensi Algoritma *Simulated Annealing* adalah sebagai berikut:
   a. Nilai awal untuk parameter kontrol temperatur (T_0)
   b. Fungsi penurunan nilai temperatur (α)
   c. Banyak iterasi untuk setiap nilai temperatur (L)
   d. Nilai akhir untuk temperatur (T_f) atau kriteria penghentian untuk berhenti melakukan eksekusi.
2. Pembangkitan Solusi Awal

   Pada penelitian ini, solusi awal dari *Puzzle* Sudoku dibangkitkan dengan Algoritma *Backtracking* hingga sebagian sel terisi angka. Berikut tahapan untuk membangkitkan solusi awal menurut Indriyono *et al*. (2023):
   a. Cari sebuah sel yang kosong yang dimulai dari sel pada sebuah baris ke kanan.
   b. Cari semua kandidat angka yang dapat dituliskan pada sel tersebut, lalu urutkan angka tersebut.
   c. Sesuai dengan urutannya, tempatkan sebuah kandidat angka pada sel tersebut.
   d. Jika angka tersebut valid berdasarkan ketentuan *Puzzle* Sudoku, maka tempatkan angka tersebut pada sel terpilih. Lanjutkan ke sel kosong selanjutnya dan kembali ke Langkah b.
   e. Jika angka tersebut tidak valid, pasangkan sel tersebut dengan kandidat angka selanjutnya.
   f. Jika tidak ada angka yang valid, mundur ke elemen matriks sebelumnya untuk mengganti angka yang ditulis dengan kemungkinan angka lain.
   g. Ulangi Langkah b sampai f sampai sebagian sel terisi angka.

3. Pencarian Solusi *Neighborhood*

   Pada tahapan ini dicari solusi baru (s') yang merupakan solusi tetangga yang diperoleh dari solusi awal. Kondisi baru (s') merupakan solusi yang mirip atau secara signifikan berbeda dengan solusi saat ini. Pada penelitian ini, pembangkitan solusi *neighborhood* dilakukan dengan cara menukarkan isi sel pada sub-matriks yang sama. Berdasarkan solusi awal yang sudah diperoleh melalui Algoritma *Backtracking*, sel-sel yang masih kosong diisi dengan angka random sedemikian sehingga dalam setiap sub-matriks hanya memuat tepat satu angka (Lewis, 2007). Secara umum, penukaran sel oleh operator *neighborhood* untuk mendapatkan kandidat solusi dilakukan dengan cara berikut:

   a. Pilih *i* dan *j* secara acak, di mana $1 \leq i, j \leq n, dan\ x_{i,j}$ tidak tetap.
   b. Pilih *k* dan *l* secara acak, di mana $x_{k,l}$ ada di kotak yang sama dengan $x_{i,j}$, $x_{k,l}$ tidak tetap, dan $x_{k,l} \neq x_{i,j}$.
   c. Tukar $x_{i,j}$ dengan $x_{k,l}$.

   Menurut Rahayu *et al*. (2017), *Simulated Annealing* hanya akan menganggap benar tidaknya pengisian angka jika kandidat solusi memenuhi salah satu syarat berikut:

   a. s' (tetangga) memiliki fungsi biaya (*cost*) yang lebih baik dibandingkan dengan *s* (kandidat solusi).
   b. Probabilitasnya diterima dengan persamaan $\exp\left(\frac{-\delta}{T}\right)$ dimana δ merupakan nilai yang berubah pada fungsi biaya dan *T* (temperatur) adalah parameter kontrol.

4. Evaluasi Kandidat Solusi

   Pendekatan ini, digunakan juga proses evaluasi delta. Jadi, setelah sebuah perpindahan tunggal di *neighbourhood*, dibandingkan dengan mengevaluasi ulang semua kandidat solusi, hanya bagian-bagian yang telah diubah saja yang dihitung ulang.

5. Penurunan Suhu

   Selama berjalannya proses, pengurangan suhu dilakukan dengan *geometric cooling schedule* yang sederhana di mana suhu saat ini (t_i) dimodifikasi menjadi suhu baru (t_{i+1}) dengan formulasi:

$$t_{i+1} = \alpha \cdot t_i$$

   di mana α merupakan parameter kontrol yang dikenal sebagai *cooling rate*, dan $0 < \alpha < 1$. Semakin tinggi nilai untuk α, contohnya $0{,}999$, akan mengakibatkan penurunan suhu yang sangat perlahan.

6. Jumlah Iterasi pada Setiap Suhu

   Penentuan jumlah iterasi pada setiap suhu bisa ditentukan nilai iterasi yang tetap untuk setiap suhu. Metode lainnya yang direkomendasikan oleh Lundy & Mees (1986) adalah dengan dilakukan satu iterasi saja pada setiap temperatur dengan penurunan suhu secara perlahan.

7. Kriteria Pemberhentian

   Dalam penelitian ini, Algoritma *Simulated Annealing* akan dihentikan ketika berada dalam kondisi berikut:

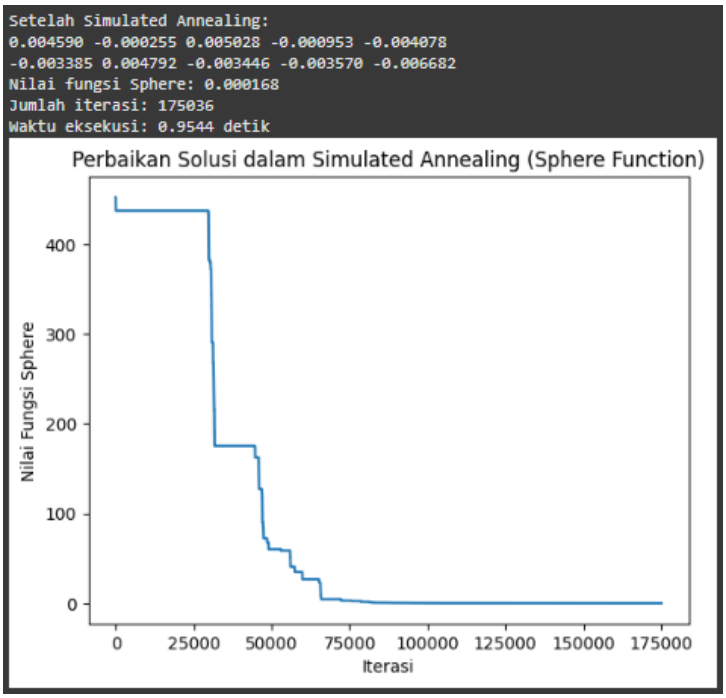
   a. Suhu telah mencapai suhu akhir yang sudah ditentukan.
   b. Iterasi yang dilakukan telah mencapai batas maksimal iterasi.
   c. Biaya pelanggaran (*cost*) telah mencapai 0.

   Kriteria ketiga adalah ketika biaya pelanggaran telah mencapai nilai minimum yang ditunjukkan dengan tidak adanya pelanggaran yang terjadi pada setiap baris dan kolom *Puzzle* Sudoku.

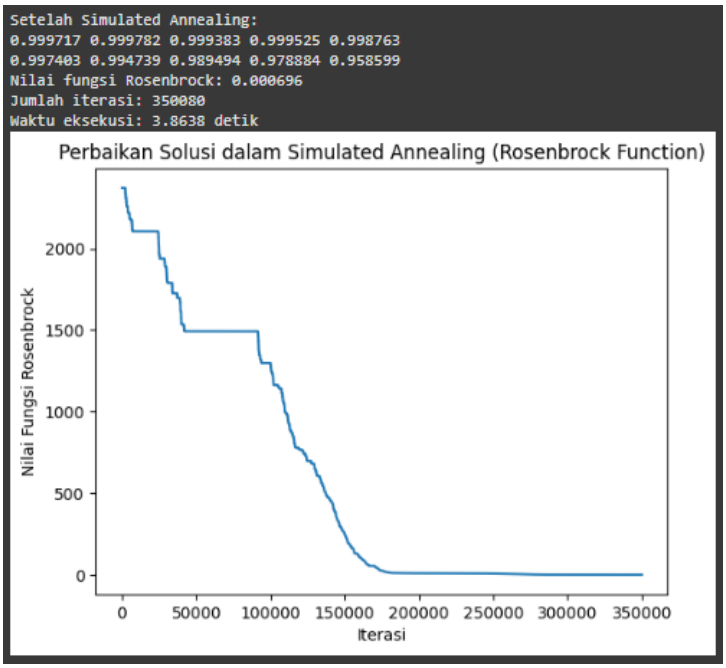
3. HASIL DAN PEMBAHASAN

Bagian ini membahas penerapan gabungan kedua algoritma pada penyelesaian *Puzzle* Sudoku. Implementasi metode dilakukan secara komputasi menggunakan Bahasa Pemrograman *Python*. Tahapan pertama implementasi adalah melakukan validasi menggunakan fungsi *Sphere* dan fungsi *Rosenbrock* untuk menguji kemampuan metode yang diusulkan dalam mencari nilai optimal global. Nilai optial global dari kedua fungsi dapat dilhat pada Digalakis & Margaritis (2000). Hasil pengujian untuk kedua fungsi tersebut dapat dilihat pada Gambar 3.1 dan Gambar 3.2 Hasil pengujian menunjukkan bahwa metode yang diusulkan mampu mendapatkan solusi yang sangat mendekati nilai optimum global. Dengan demikian metode yang diusulkan telah valid dan siap untuk diimplementasikan.

Tahapan selanjutnya adalah mengimplementasikan algoritma yang diusulkan pada penyelesaian *Puzzle* Sudoku berukuran 9×9. Misalkan *Puzzle* tersebut memiliki 38 sel yang sudah terisi. Nilai Parameter algoritma ditetapkan sebagai berikut: temperatur awal sebesar 4000, fungsi penurunan temperatur sebesar 99% atau α=0,99, temperatur akhir sebesar 0,001 dan penurunan suhu dilakukan setiap satu kali iterasi. Gambar 4 adalah input *Puzzle* Sudoku pada program.



Gambar 3. 1 Hasil algoritma gabungan pada Fungsi *Sphere*



Gambar 3. 2 Hasil algoritma gabungan pada Fungsi *Rosenbrock*

Penyelesaian *puzzle* dimulai dengan mengisi sel kosong menggunakan Algoritma *Backtracking* hingga baris ke-3 sehingga akan diperoleh *puzzle* pada Gambar 3.3. Selanjutnya, sel-sel kosong diisi dengan angka acak yang muncul satu kali untuk setiap sub-matriks sebelum dicari solusi optimal dengan *Simulated Annealing*. Langkah pengisian ini memungkinkan adanya pelanggaran yang terjadi di baris dan atau kolom tertentu, sehingga program akan menghitung dan menampilkan posisi terjadinya pelanggaran.

Sel-sel kosong pada Gambar 3.4 kemudian diisi dengan angka acak yang muncul satu kali untuk setiap sub-matriks sebelum dicari solusi optimal dengan *Simulated Annealing*. Langkah pengisian ini memungkinkan adanya pelanggaran yang terjadi di baris dan/atau kolom tertentu, sehingga program akan menghitung dan menampilkan posisi terjadinya pelanggaran seperti pada Gambar 3.5. Berdasarkan Gambar 3.5 dapat terlihat baris dan kolom mana saja yang memuat pelanggaran dengan total keseluruhan 33 pelanggaran, 19 pelanggaran pada kolom dan 14 pelanggaran pada baris. Pada kolom 1 terdapat 3 pelanggaran karena tidak adanya angka {1, 5, 9}, kolom 2 terdapat 2 pelanggaran karena tidak adanya angka {2, 8}, kolom 3 dan 4 masing-masing terdapat 1 pelanggaran karena tidak adanya angka {6}, kolom 5 terdapat 2 pelanggaran karena tidak adanya angka {4,7}, kolom 6 terdapat 2 pelanggaran karena tidak adanya angka {1, 2}, kolom 7 terdapat 3 pelanggaran karena tidak adanya angka {1, 4, 6}, kolom 8 terdapat 3 pelanggaran karena tidak adanya angka {3, 5, 9}, dan pada kolom 9 terdapat 2 pelanggaran karena tidak adanya angka {2, 6}. Selain itu, terdapat juga 2 pelanggaran pada baris 4 dengan angka {1, 9} yang tidak ada, 3 pelanggaran pada baris 5 dengan angka {4, 7, 8} yang tidak ada, 1 pelanggaran pada baris 6 dengan angka {2} yang tidak ada, 3 pelanggaran pada baris 7 dengan angka {5, 6, 9} yang tidak ada, 3 pelanggaran pada baris 8 dengan angka {3, 4, 7} yang tidak ada, dan 2 pelanggaran pada baris 9 dengan angka {2, 7} yang tidak ada.

```
# Input Sudoku (9x9)
board = np.array([
    [2, 0, 5, 8, 1, 0, 3, 0, 9],
    [8, 9, 0, 5, 0, 0, 2, 0, 7],
    [0, 0, 1, 0, 0, 0, 0, 6, 0],
    [7, 0, 2, 0, 5, 0, 0, 8, 4],
    [0, 0, 0, 2, 0, 3, 0, 0, 1],
    [0, 1, 4, 7, 9, 0, 0, 0, 3],
    [3, 0, 0, 4, 8, 0, 0, 7, 0],
    [0, 5, 0, 9, 0, 6, 0, 0, 0],
    [6, 4, 0, 1, 3, 0, 0, 0, 8]
])
```

Gambar 3. 3 Input elemen *Puzzle* Sudoku berukuran 9×9 pada program

```
Tiga baris pertama berhasil diisi dengan Backtracking:
[[2 6 5 8 1 7 3 4 9]
 [8 9 3 5 6 4 2 1 7]
 [4 7 1 3 2 9 8 6 5]
 [7 0 2 0 5 0 0 8 4]
 [0 0 0 2 0 3 0 0 1]
 [0 1 4 7 9 0 0 0 3]
 [3 0 0 4 8 0 0 7 0]
 [0 5 0 9 0 6 0 0 0]
 [6 4 0 1 3 0 0 0 8]]
```

Gambar 3. 4 Pengisian *Puzzle* Sudoku hingga baris ke-3 dengan Algoritma *Backtracking*

```
Setelah mengisi baris 4-9 dengan angka random unik:
[[2 6 5 8 1 7 3 4 9]
 [8 9 3 5 6 4 2 1 7]
 [4 7 1 3 2 9 8 6 5]
 [7 3 2 4 5 6 7 8 4]
 [6 9 5 2 1 3 9 2 1]
 [8 1 4 7 9 8 5 6 3]
 [3 1 7 4 8 7 2 7 3]
 [2 5 8 9 2 6 5 6 1]
 [6 4 9 1 3 5 9 4 8]]
Kolom 1: 3 pelanggaran
Kolom 2: 2 pelanggaran
Kolom 3: 1 pelanggaran
Kolom 4: 1 pelanggaran
Kolom 5: 2 pelanggaran
Kolom 6: 2 pelanggaran
Kolom 7: 3 pelanggaran
Kolom 8: 3 pelanggaran
Kolom 9: 2 pelanggaran
Baris 4: 2 pelanggaran
Baris 5: 3 pelanggaran
Baris 6: 1 pelanggaran
Baris 7: 3 pelanggaran
Baris 8: 3 pelanggaran
Baris 9: 2 pelanggaran
```

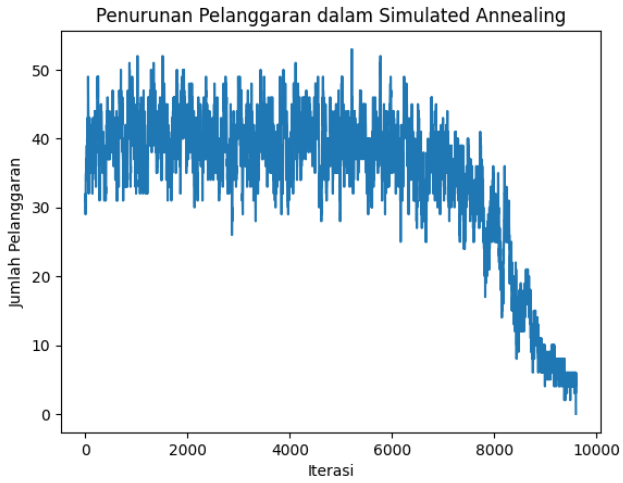
Gambar 3. 5 *Puzzle* Sudoku yang telah diisi angka acak dan dihitung jumlah pelanggarannya

Setelah diperoleh solusi awal seperti pada Gambar 3.5, maka dilakukan pencarian solusi optimal dengan menggunakan Algoritma *Simulated Annealing*. Algoritma akan berhenti ketika kriteria pemberhentian terpenuhi, yaitu ketika tidak ada pelanggaran yang terjadi. Hasil keluaran dari program ditunjukkan pada Gambar 3.6. Hasil ini diperoleh pada iterasi ke 9597 selama 1,4510 detik dengan suhu terakhir 0,2704 derajat. Jumlah pelanggaran per iterasi dapat dilihat pada Gambar 3.7. Berdasarkan Gambar 3.7 dapat dilihat bahwa algoritma

berhasil menurunkan pelanggaran hingga ditemukannya solusi optimal. Hasil ini menunjukkan bahwa metode penyelesaian yang diusulkan mampu mencari solusi optimal *Puzzle* Sudoku seperti yang diharapkan.

```
Setelah Simulated Annealing:
[[2 6 5 8 1 7 3 4 9]
 [8 9 3 5 6 4 2 1 7]
 [4 7 1 3 2 9 8 6 5]
 [7 3 2 6 5 1 9 8 4]
 [9 8 6 2 4 3 7 5 1]
 [5 1 4 7 9 8 6 2 3]
 [3 2 9 4 8 5 1 7 6]
 [1 5 8 9 7 6 4 3 2]
 [6 4 7 1 3 2 5 9 8]]
Suhu terakhir: 0.270427277574003
Jumlah iterasi: 9597
Total pelanggaran akhir: 0
Alasan berhenti: Pelanggaran 0 tercapai
Waktu eksekusi: 1.4510 detik
```

Gambar 3.6. Solusi *Puzzle* Sudoku hasil keluaran program



Gambar 3.7. Jumlah pelangaran per iterasi

Gabungan Algoritma *Backtracking* dan *Simulated Annealing* juga di ujikan pada *Puzzle* Sudoku berukuran 4×4 hingga 20×20. Untuk mengetahui kinerja algoritma, maka dilakukan membandingkan waktu komputasi yang dibutuhkan untuk menyelesaikan permainan oleh algoritma gabungan dengan masing-masing algoritma. Percobaan dilakukan sebanyak 5 kali, kemudian dihitung rata-rata jumlah pelanggaran (*cost*) dan waktu komputasi solusi optimalnya yang dirangkum pada Tabel 3.1.

Tabel 3. 1 Hasil Perbandingan Algoritma *Backtracking*, *Simulated Annealing*, dan *Backtracking-Simulated Annealing.*

Ukuran	***Backtracking***		***Simulated Annealing***		***Backtracking-Simulated Annealing***	
	Cost	**Waktu**	***Cost***	**Waktu**	***Cost***	**Waktu**
4×4	0	0,00070	0	0,018866	0	0,00834
6×6	0	0,00136	0	0,078868	0	0,07884
8×8	0	0,00485	0	0,293406	0	0,14611

9×9	0	0,00503	0	0,35962	0	0,14936
10×10	0	0,00568	0	0,42816	0	0,311932
12×12	0	0,023812	0	0,625728	0	0,64143
14×14	0	0,024428	0	2,935626	0	3,053316
15×15	0	0,081506	0	3,66031	0	3,566556
16×16	0	0,0844414	0	4,475866	0	4,620332
20×20	0	112,832696	0	52,459996	0	12,954974

Berdasarkan hasil pada Tabel 3.1 dapat dilihat bahwa waktu komputasi yang dibutuhkan untuk mencari solusi optimal *puzzle* akan meningkat seiring bertambahnya ukuran *puzzle*. Untuk *puzzle* berukuran 4×4 hingga 16×16, Algoritma *Backtracking* memiliki waktu komputasi yang lebih cepat dibanding algoritma lainnya, akan tetapi menjadi yang lebih lama ketika menyelesaikan *puzzle* berukuran 20×20. Tetapi, algoritma gabungan antara *Backtracking* dan *Simulated Annealing* mampu menyelesaikan *puzzle* lebih cepat dibandingkan dengan Algoritma *Simulated Annealing* saja. Selain itu, algoritma gabungan *Backtracking* dan *Simulated Annealing* mampu menyelesaikan *puzzle* berukuran 20×20 dalam waktu yang lebih cepat dibandingkan masing-masing algoritma.

Tahapan terakhir adalah melakukan pengujian terhadap pengaruh perubahan nilai parameter *Simulated Annealing* yang digunakan terhadap solusi optimal dan waktu komputasi yang dihasilkan oleh algoritma gabungan. Parameter yang diuji terdiri dari suhu awal (T_0) dan fungsi penurunan temperatur (α). Untuk setiap nilai parameter dilakukan percobaan sebanyak lima kali, kemudian dihitung rata-rata *cost*, waktu komputasi, dan banyaknya iterasi yang dibutuhkan untuk menyelesaikan permainan. Hasil pengujian dapat dilihat pada pada Tabel 3.2 dan Tabel 3.3.

Tabel 3. 2 Hasil pengujian parameter dengan $\alpha = 0{,}99$**.**

Temperatur Awal	Cost	Waktu Komputasi (Detik)	Banyak Iterasi
2.000	0	0,15348	1.162
4.000	0	0,17774	1.358,2
6.000	0	0,21084	1368,4
8.000	0	0,25684	1349,2
10.000	0	0,27856	1547,4

Tabel 3. 3 Hasil pengujian parameter dengan $\alpha = 0{,}995$.

Temperatur Awal	*Cost*	Waktu Komputasi (Detik)	Banyak Iterasi
2.000	0	0,24414	2.141
4.000	0	0,29832	2.257,2
6.000	0	0,31818	2.381,8
8.000	0	0,32764	2.562,2
10.000	0	0,33078	2686,8

Berdasarkan hasil pengujian pada Tabel 3.2 dan Tabel 3.3 dapat dilihat bahwa nilai penurunan temperatur α dan temperatur awal (T_0) berpengaruh pada solusi optimal yang diperoleh dan waktu komputasi serta banyaknya iterasi yang dibutuhkan untuk menyelesaikan permainan. Hasil pengujian menunjukkan bahwa $\alpha = 0{,}995$ memerlukan lebih banyak iterasi dengan waktu komputasi yang lebih lama jika dibandingkan dengan $\alpha = 0{,}99$. Dengan demikian nilai parameter yang direkomendasikan untuk digunakan pada algoritma yang diusulkan adalah $T_0 = 2000$ dan $\alpha = 0{,}99$.

4. KESIMPULAN

Implementasi dari gabungan Algoritma *Backtracking* dan *Simulated Annealing* untuk menyelesaikan *Puzzle* Sudoku dilakukan melalui pembangkitan solusi awal secara acak menggunakan Algoritma *Backtracking*. Selanjutnya, *Simulated Annealing* diimplementasikan secara iteratif untuk menentukan solusi tetangga yang dapat memperbaiki solusi sebelumnya sampai diperoleh solusi yang tidak memuat pelanggaran aturan Sudoku. Hasil implementasi menunjukkan bahwa gabungan Algoritma *Backtracking* dan *Simulated Annealing* berhasil menyelesaikan *Puzzle* Sudoku ukuran 4×4 sampai 20×20. Khusus untuk *puzzle* berukuran 20×20, algoritma gabungan dapat menyelesaikan permainan *Puzzle* Sudoku lebih cepat dibandingkan Algoritma *Backtracking* atau *Simulated Annealing* secara terpisah. Hasil ini menunjukkan kemampuan yang cukup baik dari metode yang diusulkan dalam menyelesaikan *Puzzle* Sudoku.

5. DAFTAR PUSTAKA

Digalakis, J. G., & Margaritis, K. G. (2002). An experimental study of benchmarking functions for genetic algorithms. *International Journal of Computer Mathematics*, *79*(4), 403-416.

Indriyono, B., Pamungkas, N., Pratama, Z., Mintorini, E., Dimentieva, I., & Mellati, P. (2023). Comparative analysis of the performance testing results of the backtracking and genetics algorithm in solving Sudoku games. *International Journal of Artificial Intelligence & Robotics (IJAIR)*, *5*(1), 29-35.

Jana, S., Maji, A. K., & Pal, R. K. (2015). A novel Sudoku solving technique using column based permutation. In *2015 International Symposium on Advanced Computing and Communication (ISACC).* 71-77. IEEE.

Lewis, R. (2007). Metaheuristics can solve sudoku puzzles. *Journal of heuristics*, *13*(4), 387-401.

Kamal, S., Chawla, S. S., & Goel, N. (2015). Detection of Sudoku puzzle using image processing and solving by Backtracking, Simulated Annealing and Genetic Algorithms: A comparative analysis. In *2015 Third International Conference on Image Information Processing (ICIIP)* 179-184.

Lundy, M., & Mees, A. (1986). Convergence of an annealing algorithm. *Mathematical programming*, *34*(1), 111-124.

Panggabean, H. P. (2004). Algoritma *Simulated Annealing* untuk pembentukan sel mesin dengan dua tipe fungsi objektif dan dua cara pembatasan sel. *Jurnal Teknik Industri: Jurnal Keilmuan dan Aplikasi Teknik Industri*, *6*(1), 10-24.

Rahayu, D. S., Suryapratama, A., Amongsaufa, A. Z., & Koloay, B. I. K. (2017). Evaluasi Algoritma Runut Balik dan *Simulated Annealing* pada permainan Sudoku. *Jurnal Teknik Informatika dan Sistem Informasi*, *3*(1), 1–8.

Santos-García, G., & Palomino, M. (2007). Solving Sudoku puzzles with rewriting

rules. *Electronic Notes in Theoretical Computer Science*, *176*(4), 79-93.

Sari, R. D. (2008). Analisis penyelesaian *Puzzle* Sudoku dengan menerapkan Algoritma *Backtracking* memanfaatkan Bahasa Pemrograman Visual Basic 6.0. *Jurnal Ilmiah Teknologi Informasi Asia, 2*(2), 1-18.

Yusuf, A., & Hendra, H. (2013). Penyelesaian *Puzzle* Sudoku menggunakan Algoritma *Brute Force* dan *Backtracking*. *Techno Nusa Mandiri, 10*(1), 207-215.