International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3798-3805
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0499

A Hybrid Evolutionary Approach With Adaptive Local Search For Solving Constraint-Based Sudoku Puzzles

Muvva Himasri[1], N Bhuvaneswar Reddy[2], Nannuri Manogna[3], Dr. Chaparala Aparna[4]
[1, 2, 3]UG Scholar, Dept. of CSE, R.V.R. & J.C. College of Engineering, Guntur, Andhra Pradesh, India.
[4]Professor, Dept. of CSE, R.V.R. & J.C. College of Engineering, Guntur, Andhra Pradesh, India.
Email ID: *mhimasri2005@gmail.com[1], Sasivardha2007@email.com[2], manognanannuri15@gmail.com[3], aparna@rvrjc.ac.in[4]*

Abstract
Sudoku is a constraint satisfaction problem that can be solved using an algorithm that fills a 9×9 grid in which each column, row, and sub-grid has the numbers from 1 to 9 once. As a combinatorial problem, Sudoku can also be modelled as an NP-Hard Optimization Problem. There are several genetic algorithms (GAs) that have been used with considerable success; however, most GAs converge slowly and often converge prematurely to local optimums. This paper describes a new Adaptive Hybrid Local Search Genetic Algorithm (AH-LSGA). An AH-LSGA is a combination of a Genetic Algorithm, Constraint Propagation, and Adaptive Local Search. The adaptive aspect of this algorithm allows a local search to be performed only when stagnation occurs, thereby saving computation time and increasing the speed of convergence. A comparison between the AH-LSGA and the traditional Genetic Algorithm and Local Search Genetic Algorithm will be made based on a number of performance metrics, such as fitness value, execution time, number of generations, and effective generations. The results of the experiment indicate that the proposed hybrid algorithm consolidates the speed of convergence, and reduces execution time, while maintaining a high quality solution.

Keywords: *Sudoku, Genetic Algorithm, Local Search, Hybrid Algorithm, Constraint Satisfaction Problem, Optimization*

1. Introduction

Sudoku is a well-known and common combinatorial optimization problem that has received a great deal of attention by researchers in the fields of artificial intelligence and evolutionary computation [5]. A standard Sudoku consists of a grid in the shape of a 9 x 9 matrix, which contains 9 subgrids, each measuring 3 x 3. The objective of Sudoku is to fill in the empty cells using the numbers 1 through 9 in such a way that no number occurs more than once in any row, column, or subgrid. Sudoku can be viewed as a constraint satisfaction problem (CSP). The objective of CSP is to assign the variables with values that satisfy every constraint of the problem. The solution of Sudoku has a large search space and is restricted in its search space; thus, the overall solution to a Sudoku problem is considered NP-Hard [1]. As such, Sudoku will be utilized to provide evaluations of both optimization models and hybrid metaheuristic models [2].

1.1.Sudoku as a Constraint Optimization Problem

Sudoku can also be solved using a mathematical formulation that is based on constraints of the game that define what a valid solution looks like; in practice this is called a Constraint Optimisation Problem (COP). As a COP the constraint optimisation will ensure that no digit may appear twice in any row, column, or 3×3 sub-grid of a Sudoku board; the objective of the COP is therefore to minimise the number of violations of all of these constraints whilst satisfying the requirements of the game [13]. The search space of Sudoku as a COP is extremely large, making exhaustive search techniques such as backtracking or constraint propagation prohibitively expensive for many complex puzzles. Traditional exact techniques such as backtracking and constraint propagation will always yield the correct solution; however, they may require a significant amount of computational time

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3798-3805
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0499

before achieving a solution for complex puzzles [32].

1.2. Genetic Algorithm For Sudoku

Genetic algorithms (GAs) are fundamentally population (or, more generally, candidate)-based methods of optimization and, thus, are capable of efficiently exploring large search spaces. Each individual (candidate solution) corresponds to a candidate grid; hence, each solution is evaluated based on fitness determined by the number of violations against the Sudoku puzzle constraints [4]. Across multiple generations, selection, crossover, and mutation operations are used to evolve the population toward a better solution. Unfortunately, the standard GA can exhibit premature convergence by being trapped in a local optima characterized by slow convergence.

1.3. Hybrid And Local Search-Based Approaches

To enhance the performance of Genetic Algorithms (GA), the Local Search Genetic Algorithm (LSGA) incorporates a local improvement pathway to mitigate conflicts. While LSGA enables quicker convergence rates, performing local searches in each generation adds additional costs to the process. This paper describes an Adaptive Hybrid Genetic Algorithm (AHGA) that integrates Constraint Propagation (CP), Genetic Algorithm (GA), and Adaptive Local Search (ALS) to solve these problems. Adaptive mechanisms for applying local search only occur when convergence has ceased, reducing the number of computations performed while still achieving rapid convergence [24].

2. Literature Review

2.1. Exact and Constraint-Based Methods

Various techniques have been employed in order to assess the completion of a given Sudoku puzzle. Backtracking can be implemented via depth first search; however, backtracking constrains the unnecessary examination of combinations through a complete search of all possible valid combinations until the solution is found. In addition to backtracking, there are several other techniques that can prune some or all parts of the search space by applying forward checking, which removes invalid values based on constraints, and arc consistency. Dancing links is one of the more efficient implementations of Algorithm X to solve problems associated with Sudoku [11]. Although all of the aforementioned techniques can provide an optimal solution in an efficient manner, some Sudoku puzzles will be difficult to solve using these approaches because of their computational complexity attributed to the exponential increase in search space [12]. This provides justification for the use of heuristic and/or optimization approaches.

2.2. Genetic Algorithm for Sudoku Solving

Genetic Algorithms (or GAs) solve problems by simulating the process of evolution using a population of solution candidates. Candidates for solving Sudoku would be considered “individuals” and evaluated based on a fitness score derived from the number of constraint violations. The GAs will continuously be applied to the existing population until they produce improved solutions through the use of three genetic operators (selection, crossover, and mutation) [25]. GAs work well in exploring extensive search spaces and generating approximate solutions. However, GAs can also converge rapidly or be stuck at local optima, making solving challenging Sudoku puzzles even more difficult.

2.3. Nb-Coin Algorithm

NB-COIN (Node-Based Coincidence Algorithm) employs a probabilistic optimization technique to solve Sudoku puzzles, differing from typical Genetic Algorithms by generating candidate solutions based on probabilities instead of utilizing genetic operators. This non-genetic approach decreases the chances of being stuck in a local optimum and performs better than traditional GAs on easy Sudoku puzzles. However, NB-COIN may take a much greater number of iterations to find a solution to difficult Sudoku puzzles. Thus, hybrid techniques that merge both global and adaptive local searches will typically perform more effectively.

2.4. Local Search And LSGA Approaches

To enhance the effectiveness of Genetic Algorithm (GA), it has been developed with a Local Search component that can be implemented in each generation of GA [14]. The Local Search Genetic Algorithm (LSGA) executes a Local Search method on individuals of each generation, reducing violations of constraints by applying Local Search on them. Most frequently, a swap-based Local Search procedure is performed at the column/subgrid level.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3798-3805
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0499

As a result, the hybridization of LSGA enhances the speed of convergence and quality of the solution compared to conventional GA. However, performing Local Search at each generation introduces additional computational cost, thus decreasing overall efficiency [22].

2.5. Hybrid Metaheuristic Optimization Approaches

Hybrid Metaheuristic Algorithms combine many algorithms to diversify the optimization approach by balancing exploration and exploitation. These methods generally include elements of a global search algorithm such as Genetic Algorithms as well as components of a local search or simulated annealing, or a combination [7]. The hybrid Metaheuristic Algorithms have been effectively applied to many NP-hard combinatorial optimization problems displaying significantly higher convergence times and solution quality [23]. This paper presents an Adaptive Hybrid Local Search Genetic Algorithm that applies Constraint Propagation, Genetic Algorithm and Adaptive Local Search to improve the complexity of solutions for solving Sudoku puzzles Figure 1.

3. Methodology

3.1. Problem Formulation (Sudoku as CSP)

In this study the Sudoku is a combinatorial constraint satisfaction problem in which a 9×9 grid must be filled so that each row, column and little 3x3 box has the numbers 1 to 9 without repeating any number. The Sudoku grid has some numbers already filled in and some empty spaces that we need to fill in. Let us look at the Sudoku grid like a matrix: S=[sij] i, j = 1, 2… 9, where each little box sij can have a number from 1 to 9. To solve the Sudoku problem we need to follow these rules:

Row rule: each number can only appear once in each row. Column rule: each number can only appear once in each column. Subgrid rule: each number can only appear once in each 3x3 box. The goal of the Sudoku problem is to make sure we do not break any of these rules. If we do break a rule it is like making a mistake. We want to make as few mistakes as possible so we try to find the solution that breaks the fewest rules.

3.2.Sudoku Representation and Initialization

Each solution is shown as a 9x9 grid. The cells that are already filled stay the same while the empty ones get filled in when we start. We create the set of solutions by filling in the empty cells row by row. We randomly add the numbers 1 to 9 in each row making sure each number only appears once. This way we make sure each row is correct. We only have to worry about making sure the columns and smaller 3x3 squares are correct.

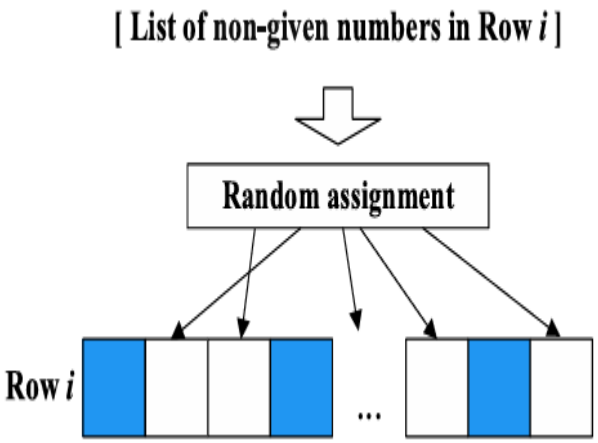

Figure 1 Initialization Stage: Non-Given Numbers In Row I Are Randomly Assigned To Empty Spaces

3.3. Fitness Function

The fitness function evaluates the quality of each candidate Sudoku solution by counting the number of conflicts in columns and subgrids.

$$F = C_c + C_s$$

Where: C_c = Number of column conflicts;
C_s = Number of subgrid conflicts. The objective of the algorithm is to minimize the fitness value. A fitness value of zero indicates a valid Sudoku solution.

3.4. Crossover and Mutation

In this study Genetic operators such as crossover and mutation are used to come up with new candidate solutions and keep things diverse in the population. The crossover operator takes two parent Sudoku grids and makes a new one by swapping rows between them. Each row has the numbers 1 to 9 without repeating any. The mutation operator picks two numbers in a row that are not fixed and swaps them making sure that the given cells stay the same. The crossover and mutation work together to help the

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3798-3805
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0499

algorithm look at all the possibilities and come up with candidate solutions over time Figure 2.

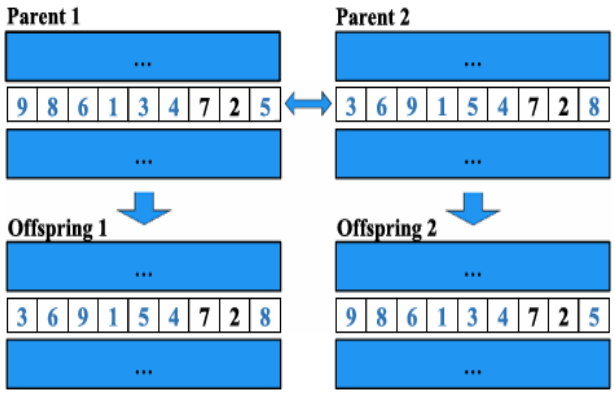

Figure 2 Crossover between two individuals

Table 1 Crossover

Algorithm 1: Crossover
Input: Parent1, Parent2, crossover rate Pc *Output: Offspring* *1: Generate a random number r* *2. If r < Pc:* *3. Select random rows* *4. Swap selected rows between Parent1 and Parent2* *5. Create Offspring1 and Offspring2* *6. Else* *7. Offspring = Parent1 (no crossover)* *8. Return Offspring*

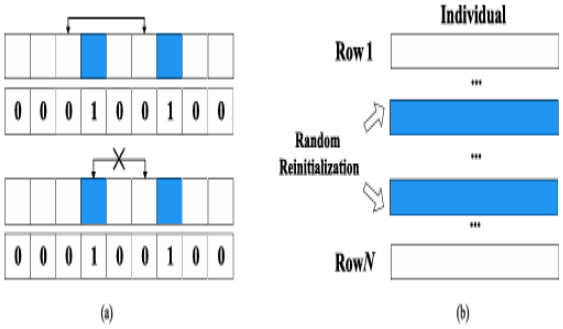

Figure 3 Two designed mutation operations. (a) Swap mutation. (b) Reinitialize mutation

Table 2 Mutation

Algorithm 2: Mutation
Input: Individual, mutation rate Pm *Output: Mutated Individual* *1. For each row in the individual:* *2. Generate random number r* *3. If r < Pm:* *4. Select two non-fixed positions in the row* *5. Swap the values* *6. Return mutated individual*

3.5. Local Search

Local search is used to improve candidate solutions by reducing conflicts. The local search operator swaps two non-fixed values within a column or subgrid and evaluates the fitness. If the swap reduces the number of conflicts, the swap is accepted; otherwise, it is reverted. This greedy swap-based local search helps reduce conflicts quickly and improves convergence speed Table 2. This method makes the current solution better by fixing the mistakes. It guides the algorithm step by step to get a Sudoku solution. The method reduces constraint violations little by little. This helps to get closer, to a Sudoku solution Table 1.

Table 3 Local Search

Algorithm 3: Local Search
Input: Individual *Output: Improved Individual* *1. Identify columns and subgrids with conflicts* *2. Select two non-fixed cells causing conflict* *3. Swap the values* *4. Calculate new fitness* *5. If fitness improves:* *6. Accept swap* *7. Else:* *8. Revert swap* *9. Return improved individual*

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3798-3805
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0499

3.6. Stagnation Detection

Stagnation occurs when the fitness value does not improve for a number of generations. It is defined as:

$$S = \begin{cases} 1, & \text{if } F_{best}(g) = F_{best}(g-k) \\ 0, & \text{otherwise} \end{cases}$$

Where:

F_best (g)= Best fitness at generation g

k= Stagnation threshold

S=1means stagnation detected

When stagnation occurs, local search is applied Table 3.

This adaptive mechanism helps the algorithm escape local optima and improves convergence efficiency.

3.7. Hybrid Control Mechanism

The hybrid algorithm switches between GA and Local Search based on stagnation:

$$H = \begin{cases} \text{Apply Local Search}, & \text{if stagnation detected} \\ \text{Apply GA}, & \text{otherwise} \end{cases}$$

This adaptive switching improves convergence efficiency Figure 3.

3.8. Proposed Adaptive Hybrid Approach

To make the algorithm work better we can use a method that combines a few different techniques. This method is called Adaptive Hybrid. It uses Constraint Propagation, Genetic Algorithm and Local Search. The Constraint Propagation technique is used at the beginning to fill in some of the cells with values that we are sure about. Then the Genetic Algorithm looks at the problem to find a good solution. The Local Search is only used when the algorithm gets stuck. The algorithm gets stuck when it cannot find a solution even after trying many times. We call this “getting stagnation”. When stagnation happens the Local Search is used to try and find a solution. This way of doing things helps to avoid wasting time on Local Search operations and makes the algorithm work faster. The Adaptive Hybrid method is really good at finding a balance between how fast and how accurate it's when solving a problem Table 4.

Table 4 Adaptive Hybrid GA with Local Search

Algorithm 4: Adaptive Hybrid GA with Local Search
1: Generate initial population
2: Apply constraint propagation
3: Evaluate fitness
4: bestFitness = current best fitness
5: stagnationCount = 0
6: while termination condition not met do
7: Perform selection
8: Perform crossover
9: Perform mutation
10: Evaluate fitness
11: If bestFitness improved then
12: stagnationCount = 0
13: Else
14: stagnationCount = stagnationCount + 1
15: If stagnationCount > Threshold then
16: Apply local search
17: stagnationCount = 0
18: end while
19: Return best solution

4. Results And Discussion

4.1. Experimental Results

To evaluate, the Adaptive Hybrid algorithm is being compared to other algorithms to see how well it works. These other algorithms are the Genetic Algorithm, the node-based coincidence algorithm and the Local Search Genetic Algorithm. All of these algorithms are being tested on Sudoku puzzles, which have different levels of difficulty, such as easy, medium and hard. Each algorithm is run multiple times on each puzzle and the average results are written down. The things that are being measured include how many generations it takes on average to solve the puzzle, how long it takes to run the algorithm, the fitness value and how many generations it takes to actually improve the solution.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3798-3805
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0499

The Adaptive Hybrid algorithm does the best, taking fewer generations to solve the puzzle, especially for medium and hard puzzles. To see just how robust these algorithms are, some experiments were done on hard Sudoku puzzles like SD1, SD2 and SD3. The Adaptive Hybrid algorithm takes fewer generations to solve them and achieves higher success rates Table 5.

Table 5 Comparison of Average Generations Required by Different Algorithms to Solve Sudoku Puzzles.

Puzzle ID	GA Avg Gen	NB-COIN Avg Gen	LSGA AvgGen	Hybrid AvgGen
Easy 1	62	6	4	4
Easy 2	137	8	6	4
Medium 1	910	28	20	15
Medium 2	3193	57	42	30
Hard 1	9482	210	160	120
Hard 2	26825	480	300	220

Table 6 Experimental results for Super Difficult Puzzles.

Puzzle ID	LSGA Success (%)	LSGA Avg Gen	Hybrid Success (%)	Hybrid Avg Gen
SD1	100	420	100	290
SD2	100	610	100	430
SD3	92	2100	100	1500

Table 6 shows the performance comparison on super difficult Sudoku puzzles. The Hybrid algorithm requires fewer generations compared to LSGA and achieves a higher success rate in difficult puzzles. This demonstrates that the adaptive local search mechanism helps the algorithm escape local optima Table 6.

Table 7 Performance by Difficulty Level

Level	Mean Generati-ons	Max Genera-tions	Min Generatio-ns
Easy	5.2	9	2
Medium	21.4	38	8
Hard	82.6	160	25
Evil	140.3	420	40

Table 7 shows how well the Hybrid algorithm works with puzzles of different difficulty levels. The Hybrid algorithm is really good at solving medium and hard puzzles and it even does well with very hard puzzles. The LSGA method works better than the NB-COIN method, and the Hybrid algorithm works best because it uses a local search that adapts to the puzzle.

4.2. Comparative Analysis

We compare LSGA and the proposed Adaptive Hybrid algorithm in Table I. LSGA does a lot better than the Genetic Algorithm because it uses local search, which helps it converge faster and reduces constraint violations. However LSGA applies local search every time it generates something new, sometimes doing unnecessary local searches. The proposed Hybrid algorithm only uses local search when it gets stuck, which means when it does not get better for a certain number of generations. This helps reduce unnecessary local searches and makes the whole process more efficient. As a result the Hybrid algorithm needs fewer generations to get the job done and takes less time to execute compared to LSGA. The Hybrid algorithm works better especially for medium and hard Sudoku puzzles. These puzzles have large search space and it is easy to get stuck in a bad solution. The Hybrid algorithm finds the solution more efficiently than LSGA by only doing local searches when necessary. Overall the Hybrid algorithm converges better, costs less to compute and searches more efficiently than LSGA.

4.3. Performance Discussion

The improved performance of the Hybrid algorithm can be attributed to the combination of global search and adaptive local search mechanisms. The crossover

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3798-3805
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0499

and mutation operators provide global exploration by generating diverse candidate solutions and preventing premature convergence. The local search operator focuses on exploitation by reducing conflicts in columns and sub-blocks. Unlike LSGA, where local search is applied in every generation, the proposed Hybrid algorithm applies local search only when the algorithm stagnates. This reduces unnecessary computations and allows the algorithm to focus on promising regions of the search space.

4.4. Discussion On Algorithm Efficiency

From the experimental results, it is observed that the Hybrid algorithm reduces the number of effective generations compared to LSGA. Effective generations represent the number of generations in which the fitness value improves, and this metric reflects the actual progress of the algorithm rather than the total number of generations executed.

Additionally, the execution time of the Hybrid algorithm is lower than LSGA in most cases, demonstrating that the adaptive local search mechanism improves computational efficiency without compromising solution quality. The reduction in execution time becomes more significant as puzzle difficulty increases, showing that the Hybrid approach scales better for complex problems.

Conclusion

This study presented an Adaptive Hybrid Algorithm for solving Sudoku puzzles by integrating crossover, mutation, constraint propagation, and adaptive local search into a unified optimization framework. The proposed approach improves convergence efficiency by applying local search only when stagnation occurs, rather than applying it in every generation as in the Local Search Genetic Algorithm (LSGA). The Hybrid algorithm demonstrates improved performance. It works better than LSGA when we look at the fitness value and the time it takes to finish. The Hybrid algorithm is also good at solving puzzles that are easy or hard. It does this by not wasting time on things it does not need to do. This means it can solve puzzles faster and usually gets the right answer. The performance improvement is achieved by finding a balance between trying new things and using what we already know. The crossover and mutation parts of the process help us look at the bigger picture while the adaptive local search makes our solutions better by cutting down on mistakes. Overall, this work demonstrates that adaptive hybridization of evolutionary operators with local search can significantly improve the performance of evolutionary algorithms for solving constraint satisfaction problems such as Sudoku. The proposed method is efficient, robust, and scalable. Future work may focus on parameter tuning, larger Sudoku puzzles (16×16, 25×25), and other problems such as scheduling and resource allocation.

References

[1]. T. Yato and T. Seta, “Complexity and Completeness of Finding Another Solution and Its Application to Puzzles,” IEICE Transactions on Fundamentals, vol. E86-A, no. 5, pp. 1052–1060, 2003.
[2]. A. Lewis, “Metaheuristics can solve Sudoku puzzles,” Journal of Heuristics, vol. 13, no. 4, pp. 387–401, 2007.
[3]. M. Mitchell, An Introduction to Genetic Algorithms. MIT Press, 1998.
[4]. D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989.
[5]. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Prentice Hall, 2010.
[6]. R. E. Smith, “Genetic Algorithms and Constraint Satisfaction Problems,” in Proc. International Conference on Genetic Algorithms, 1995.
[7]. E. Talbi, Metaheuristics: From Design to Implementation. Wiley, 2009.
[8]. P. Moscato, “On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts,” Caltech Concurrent Computation Program, 1989.
[9]. J. Kennedy and R. Eberhart, “Particle Swarm Optimization,” Proc. IEEE International Conference on Neural Networks, 1995.
[10]. S. Kirkpatrick, C. D. Gelatt, and M. Vecchi, “Optimization by Simulated Annealing,” Science, vol. 220, pp. 671–680, 1983.
[11]. A. Lewis and J. Cochrane, “Solving Sudoku using Constraint Propagation and Genetic Algorithms,” IEEE CIG, 2007.
[12]. N. Monette and G. V. Cormack, “Constraint-

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 05 May 2026
Page No: 3798-3805
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0499

based Sudoku Solving using Evolutionary Algorithms," Proc. AAAI Conference, 2008.
[13]. K. I. M. McGuire, B. Tugemann, and G. Civario, "There is no 16-Clue Sudoku," Experimental Mathematics, 2014.
[14]. R. Lewis, "A Guide to Solving Sudoku Puzzles using Evolutionary Algorithms," University of Essex Technical Report, 2007.
[15]. T. Mantere and J. Koljonen, "Solving and rating Sudoku puzzles with genetic algorithms," Proc. Finnish AI Conference, 2006, pp. 86–92.
[16]. H. Lloyd and M. Amos, "Solving Sudoku with ant colony optimization," IEEE Transactions on Games, vol. 12, no. 3, pp. 302–311, 2020.
[17]. D. E. Knuth, "Dancing links," in Millennial Perspectives in Computer Science, 2000, pp. 187–214.
[18]. F. Gerges, G. Zouein, and D. Azar, "Genetic algorithms with local optima handling to solve Sudoku puzzles," Proc. ICCAI, 2018, pp. 19–22.
[19]. A. Z. Sevkli and K. A. Hamza, "General variable neighborhood search for solving Sudoku puzzles," Soft Computing, vol. 23, no. 15, pp. 6585–6601, 2019.
[20]. M. A. Al-Betar et al., "β-hill climbing algorithm for Sudoku game," Proc. PICICT, 2017.
[21]. J. Horn, "Solving a large Sudoku by co-evolving numerals," Proc. GECCO, 2017.
[22]. C. Wang et al., "A Novel Evolutionary Algorithm With Column and Sub-Block Local Search for Sudoku Puzzles," IEEE Transactions on Games, vol. 16, no. 1, pp. 162–172, 2024.
[23]. X. Deng and Y. Li, "A Novel Hybrid Genetic Algorithm for Solving Sudoku Puzzles," Optimization Letters, vol. 7, no. 2, pp. 241–257, 2013.
[24]. N. Musliu and F. Winter, "A Hybrid Approach for the Sudoku Problem Using Constraint Programming and Local Search," IEEE Intelligent Systems, vol. 32, no. 2, pp. 52–63, 2017.
[25]. Y. Sato and H. Inoue, "Solving Sudoku With Genetic Operations That Preserve Building Blocks," IEEE CIG, 2010.
[26]. Y. Sato, N. Hasegawa, and M. Sato, "GPU Acceleration for Sudoku Solution With Genetic Operations," IEEE CEC, 2011.
[27]. N. Pillay, "Solving Sudoku Puzzles Using Human Intuitive Heuristics," South African Computer Journal, 2012.
[28]. A. Groza, "Modelling Sudoku Puzzles Using Constraint Programming," Springer, 2021.
[29]. Z. Zhan, J. Li, and J. Zhang, "Evolutionary Computation for Combinatorial Optimization: A Survey," IEEE Transactions on Evolutionary Computation, 2022.
[30]. J. Kennedy, Swarm Intelligence. Morgan Kaufmann, 2001.
[31]. P. Norvig, "Solving Every Sudoku Puzzle," 2006. [Online]. Available: http://norvig.com/sudoku.html
[32]. H. Simonis, "Sudoku as a Constraint Problem," in Proc. CP Workshop on Modelling and Reformulating Constraint Satisfaction Problems, 2005.
[33]. N. Pathak and R. Kumar, "Improved wisdom of crowds heuristic for solving sudoku puzzles," in Proc. Soft Comput. Signal Process., 2019, pp. 369–377.
[34]. J. P. Delahaye, "The science behind SUDOKU," Sci.Amer., vol.294, no.6, pp. 80–87, Jun. 2006.
[35]. M. Horoufiany and R. Ghandehari, "Optimization of the Sudoku based reconfiguration technique for PV arrays power enhancement undermutual shading conditions," Sol. Energy, vol. 159, pp. 1037–1046, Jan. 2018.