electronics

MDPI

Article

A Novel Approach for Solving the N-Queen Problem Using a Non-Sequential Conflict Resolution Algorithm

Omid Moghimi [1] and Amin Amini [2,*]

[1] REDARC Electronics Pty., Ltd., Lonsdale, SA 5160, Australia; omoghimi@redarc.com.au
[2] School of Engineering and Computing, University of Central Lancashire, Preston PR1 2HE, UK
* Correspondence: aiamini@uclan.ac.uk

Abstract: The N-Queens problem is a fundamental challenge in combinatorial optimization, commonly used as a benchmark for assessing the efficiency of algorithms. Traditional algorithms, such as Backtracking with Forward Checking (BFC), constraint satisfaction problem (CSP) techniques, Lookahead algorithms, and heuristic-based methods, often face challenges with exponential time complexity, making them less practical for large-scale instances. This paper introduces a novel algorithm, non-sequential conflict resolution (NSCR), which improves performance over traditional algorithms through dynamic conflict resolution. The NSCR algorithm iteratively resolves conflicts among queens by adjusting their positions, aiming to optimize both time complexity and memory usage. While NSCR also operates within exponential time bounds, it demonstrates improved scalability and efficiency compared to traditional methods. A significant strength of the NSCR algorithm lies in its space complexity, which is O(n), and a time complexity that, while typically lower than traditional methods, can reach O(n^3) in the worst-case scenario. This linear space complexity is highly advantageous, particularly when dealing with large problem sizes, as it ensures efficient use of memory resources. Comparative analysis with the aforementioned algorithms shows that NSCR offers superior resource management, using up to 60% less memory and reducing runtime by approximately 50%, making it an efficient option for large-scale instances of the N-Queens problem. The algorithm's performance, evaluated on problem sizes ranging from 8 to 1000 queens, highlights its ability to manage computational resources effectively, despite the inherent challenges of exponential time complexity.

Keywords: N-Queens problem; non-sequential conflict resolution; combinatorial optimization; backtracking algorithms; space complexity optimization; conflict resolution algorithm

check for updates

Citation: Moghimi, O.; Amini, A. A Novel Approach for Solving the N-Queen Problem Using a Non-Sequential Conflict Resolution Algorithm. *Electronics* **2024**, *13*, 4065. https://doi.org/10.3390/electronics13204065

Academic Editor: Flavio Canavero

Received: 5 September 2024
Revised: 29 September 2024
Accepted: 14 October 2024
Published: 16 October 2024

1. Introduction

The N-Queens problem is a fundamental challenge in combinatorial optimization, often serving as a benchmark for evaluating the efficiency of various algorithmic approaches. Traditionally, this problem has been tackled using backtracking methods, which, while straightforward in implementation, suffer from significant drawbacks as the problem size increases. Specifically, backtracking algorithms experience exponential growth in both time and memory complexities as the number of queens increases, rendering these methods impractical for large-scale instances [1–4]. This exponential growth necessitates the exploration of more efficient alternative approaches that are capable of handling larger instances of the N-Queens problem effectively.

Over the years, extensive research has been conducted to develop more efficient methods that overcome the limitations of traditional backtracking. One notable advancement is Rakhya's method, which significantly reduces the computational steps required to solve large instances, such as the 30-Queens problem, achieving a solution in just 32 steps [2]. Similarly, Sosič and Gu introduced a local search algorithm based on conflict minimization, which is capable of solving extremely large instances, such as the 3,000,000-Queens problem, in linear time without the need for backtracking [3]. This approach represents a major leap in scalability, offering solutions where traditional methods fail. Additionally, Kralev et al. proposed an optimized backtracking algorithm that reduces the time complexity from exponential to quadratic by focusing on a specific subset of configurations, further demonstrating progress in addressing the N-Queens problem more efficiently [4]. The N-Queens problem has also been extensively studied within the framework of constraint satisfaction problems (CSPs), which provide a structured approach to comparing the performance of different algorithms across various problem sizes [5]. However, traditional backtracking algorithms, when applied to the N-Queens problem, continue to suffer from exponential time complexity, making them unsuitable for large values of 'n' [4,6].

Beyond CSP techniques, backtracking algorithms have been widely applied to constraint-based problems like the N-Queens problem. However, they often suffer from inefficiencies due to redundant computations [7]. Traditional backtracking can lead to exponential growth in computing time as the problem size increases, prompting the development of various techniques to mitigate these inefficiencies [3]. For example, constraint propagation and learning have been employed to reduce the search tree size and prevent the reoccurrence of similar conflicts [7]. Gaschnig proposed an algorithm that eliminates most redundant tests, achieving significant speedups for larger problems [8]. Stone and Stone demonstrated that reordering the search to prioritize queens with fewer possible moves can dramatically improve performance, solving problems up to n = 97 much faster than traditional lexicographic backtracking [6]. Recent advancements in hybrid algorithms, which combine elements from multiple approaches, have shown promise in further improving the efficiency and scalability of solutions to the N-Queens problem. For example, hybrid methods that integrate Genetic Algorithms (GAs) with local search techniques or incorporate advanced mutation operators have been proposed to reduce the number of queen clashes more effectively and accelerate convergence [9,10]. A recent study by El Abidine [11] introduced an incremental approach to the N-Queens problem, demonstrating a polynomial time complexity of $O(n^2)$. This method constructs solutions for larger $n \times n$ chessboards by incrementally building from solutions of smaller $(n - 1 \times n - 1)$ boards, leveraging symbolic computation on the complement of the N-Queen graph and complex graph theory concepts, such as maximum cliques and Kneser coding. While this approach offers a theoretically efficient solution, its reliance on advanced graph-theoretic constructs makes it more intricate and less adaptable for dynamic or real-time problem instances. Despite these developments, the quest for algorithms that can efficiently handle the N-Queens problem at larger scales without incurring prohibitive computational costs remains ongoing.

The algorithm introduced in this paper seeks to overcome the limitations inherent in traditional backtracking techniques, such as BFC, CSP, and Lookahead. These techniques were chosen for comparison because they represent commonly used methods for solving the N-Queens problem, providing a relevant benchmark for evaluating the performance of the NSCR algorithm. While backtracking methods often suffer from exponential time complexity and become impractical for larger problem sizes, they are effective for smaller instances and offer a foundation for exploring more advanced optimization strategies. By comparing against these traditional techniques, we demonstrate how NSCR offers improved scalability and resource management, especially for larger instances of the N-Queens problem, with lower time complexity and memory consumption compared to these conventional methods in average cases.

The remainder of this paper is organized as follows: Section 2 describes the materials and methods used, including a detailed explanation of the NSCR algorithm and the experimental setup. Section 3 presents the results of the comparative analysis between NSCR and other algorithms, highlighting key findings. In Section 4, we discuss the implications of these results, the algorithm's performance, and its potential applications. Finally, Section 5 concludes the paper by summarizing the main contributions and suggesting future research directions.

2. Materials and Methods

The N-Queens algorithms' comparisons were written in the C# programming language carried out on an 8-core AMD Ryzen 7 5700X CPU with 32 GB of main memory running Windows 11. Three algorithms have been analyzed and compared against the developed algorithm, including BFC, Lookahead, and CSP.

In contrast to traditional backtracking, which typically abandons a particular path immediately upon detecting a conflict, the NSCR algorithm attempts to resolve conflicts by moving the conflicting queen to a different position on the board before deciding to backtrack. The NSCR algorithm employs a hybrid approach that combines iterative repair with backtracking, a concept used in other heuristic solvers for combinatorial problems. However, NSCR diverges significantly from traditional iterative repair methods by introducing a dynamic conflict resolution mechanism that adapts queen placements in real time. Unlike conventional techniques, which often rely on fixed heuristics or predefined adjustment rules, NSCR evaluates the conflict intensity and adjusts the queen's position based on the overall threat landscape at each iteration.

This dynamic adjustment enables NSCR to explore alternative paths before resorting to backtracking, thereby reducing unnecessary iterations and improving efficiency. For example, when a conflict is detected, the algorithm evaluates multiple potential positions for the conflicting queen and selects the one with the least threat level, rather than blindly following a preset adjustment. This selective adjustment process significantly enhances the algorithm's adaptability and reduces the likelihood of revisiting previously explored configurations.

Moreover, while iterative repair methods often struggle with becoming trapped in local optima or reaching suboptimal solutions, NSCR mitigates this issue by integrating strategic backtracking steps. These steps allow the algorithm to escape local minima and continue the search process, leading to more optimal placements. Empirically, this approach has demonstrated improved performance compared to existing iterative repair techniques, as evidenced by NSCR's ability to handle larger problem sizes (up to 1000 queens) with lower memory usage and shorter runtime.

While NSCR is not immune to suboptimal solutions, especially in highly constrained configurations, its dynamic conflict resolution and selective backtracking strategy substantially reduce the likelihood of such occurrences, ensuring more consistent and efficient performance across various problem sizes.

This makes it more of a hybrid between backtracking and a local search or an iterative repair method. The algorithm has a more complex state management system, using lists to track which queens have been placed and which queens are in conflict. This contrasts with more traditional approaches that typically rely on simpler mechanisms like recursion with fewer explicit state variables. Instead of immediately backtracking when a conflict is found, the NSCR algorithm tries to resolve the conflict by iteratively moving the queen in question. This is not typically carried out in standard backtracking algorithms where the algorithm would instead backtrack and attempt the next possible configuration.

Figure 1 shows the algorithm procedure to solve the N-Queens problem. It begins with the initialization phase, where it sets up the necessary structures to keep track of the queens' positions on the chessboard, any conflicts that arise between them, and the status of each queen's placement. The chessboard is conceptualized as a grid, where each queen is initially assigned a starting position. The first queen is placed on the board at a predetermined position, which is typically at the beginning of the first column.

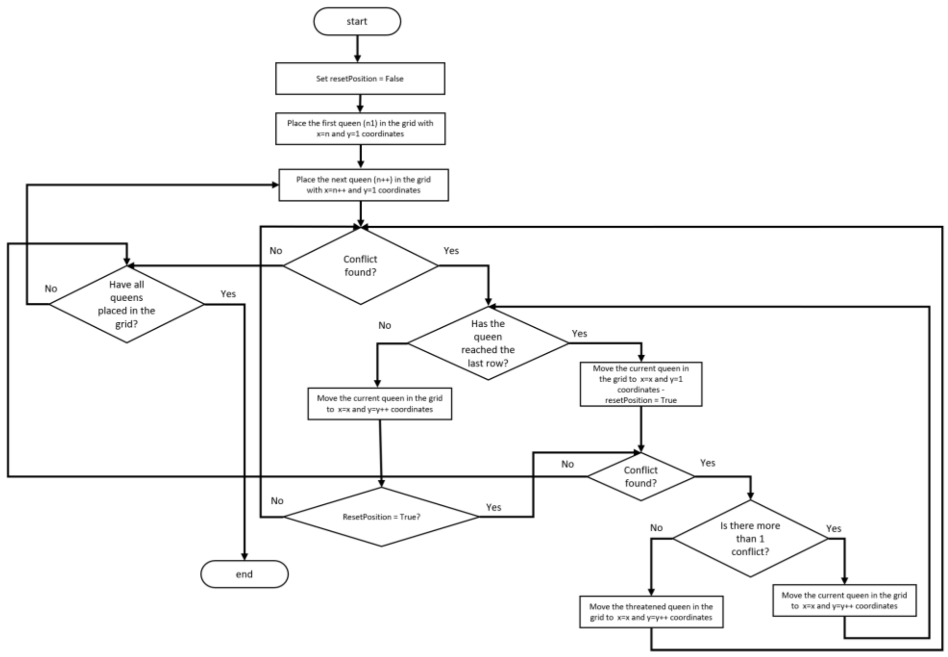
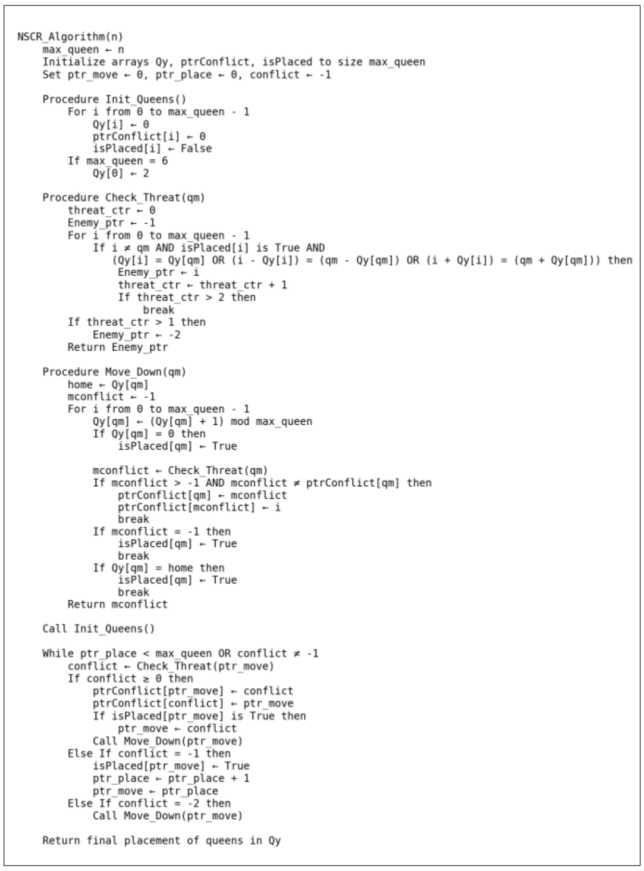

start
Set resetPosition = False
Place the first queen (n1) in the grid with x=n and y=1 coordinates
Place the next queen (n++) in the grid with x=n++ and y=1 coordinates
Conflict found?
No
Yes
Have all queens placed in the grid?
No
Yes
Has the queen reached the last row?
No
Yes
Move the current queen in the grid to x=x and y=y++ coordinates
Move the current queen in the grid to x=x and y=1 coordinates - resetPosition = True
Conflict found?
No
Yes
ResetPosition = True?
No
Yes
Is there more than 1 conflict?
No
Yes
Move the threatened queen in the grid to x=x and y=y++ coordinates
Move the current queen in the grid to x=x and y=y++ coordinates
end

Figure 1. Flowchart for the NSCR Algorithm.

The pseudocode for the NSCR algorithm is presented (Figure 2), illustrating its hybrid approach that combines iterative repair with backtracking. This pseudocode provides a clear and detailed representation of the algorithm's main steps, including conflict detection, queen placement, and iterative adjustments, offering insight into its structured problem-solving process for the N-Queens problem.


```
NSCR_Algorithm(n)
    max_queen ← n
    Initialize arrays Qy, ptrConflict, isPlaced to size max_queen
    Set ptr_move ← 0, ptr_place ← 0, conflict ← -1

    Procedure Init_Queens()
        For i from 0 to max_queen - 1
            Qy[i] ← 0
            ptrConflict[i] ← 0
            isPlaced[i] ← False
        If max_queen = 6
            Qy[0] ← 2

    Procedure Check_Threat(qm)
        threat_ctr ← 0
        Enemy_ptr ← -1
        For i from 0 to max_queen - 1
            If i ≠ qm AND isPlaced[i] is True AND
               (Qy[i] = Qy[qm] OR (i - Qy[i]) = (qm - Qy[qm]) OR (i + Qy[i]) = (qm + Qy[qm])) then
                Enemy_ptr ← i
                threat_ctr ← threat_ctr + 1
                If threat_ctr > 2 then
                    break
        If threat_ctr > 1 then
            Enemy_ptr ← -2
        Return Enemy_ptr

    Procedure Move_Down(qm)
        home ← Qy[qm]
        mconflict ← -1
        For i from 0 to max_queen - 1
            Qy[qm] ← (Qy[qm] + 1) mod max_queen
            If Qy[qm] = 0 then
                isPlaced[qm] ← True

            mconflict ← Check_Threat(qm)
            If mconflict > -1 AND mconflict ≠ ptrConflict[qm] then
                ptrConflict[qm] ← mconflict
                ptrConflict[mconflict] ← i
                break
            If mconflict = -1 then
                isPlaced[qm] ← True
                break
            If Qy[qm] = home then
                isPlaced[qm] ← True
                break
        Return mconflict

    Call Init_Queens()

    While ptr_place < max_queen OR conflict ≠ -1
        conflict ← Check_Threat(ptr_move)
        If conflict ≥ 0 then
            ptrConflict[ptr_move] ← conflict
            ptrConflict[conflict] ← ptr_move
            If isPlaced[ptr_move] is True then
                ptr_move ← conflict
            Call Move_Down(ptr_move)
        Else If conflict = -1 then
            isPlaced[ptr_move] ← True
            ptr_place ← ptr_place + 1
            ptr_move ← ptr_place
        Else If conflict = -2 then
            Call Move_Down(ptr_move)

    Return final placement of queens in Qy
```

Figure 2. Pseudocode for NSCR Algorithm.

Following this initial placement, the algorithm proceeds to place the subsequent queens one by one. For each queen, it starts by attempting to place it in the first row of the next available column. Once a queen is placed, the algorithm immediately checks for conflicts. A conflict is detected if the newly placed queen is in the same row, column, or diagonal as any of the previously placed queens. If no conflict is found, the algorithm moves on to the next queen. If a conflict is detected, the algorithm takes corrective action by attempting to move the current queen to a different position within the same column. The movement is generally downward to the next row in the same column. This adjustment aims to find a position where the queen is not under threat from any other queens. If the queen reaches the bottom of the column without finding a conflict-free position, the algorithm resets the queen to the top of the column and flags this column for re-evaluation, marking the need for further adjustments. The algorithm also handles situations where multiple conflicts are detected for a queen. In such cases, it prioritizes resolving these conflicts by adjusting the positions of the queens involved. The current queen may be moved further down the column or repositioned entirely, depending on the situation. This process of conflict detection and resolution is iterative and continues until all queens are placed on the board in positions where they do not threaten each other. Throughout the process, the algorithm maintains a loop that repeatedly checks the board's state after each queen is placed. If a queen's position causes conflicts, it will attempt to resolve these by moving the queen or by backtracking and adjusting the positions of the previously placed queens. This loop continues until all queens have been placed on the board without any conflicts. At this point, the algorithm concludes, having successfully found a solution to the N-Queens problem, where no two queens are in a position to attack one another. The step-by-step nature of this algorithm ensures that each queen's placement is carefully considered and adjusted as necessary, making it a methodical approach to solving the N-Queens problem. The use of conflict detection, position adjustments, and possible resets allows the algorithm to methodically navigate through potential solutions, ultimately arriving at a configuration where all queens coexist without threatening each other.

Figure 3 illustrates a step-by-step execution of the NSCR algorithm for solving the 4-Queens problem.

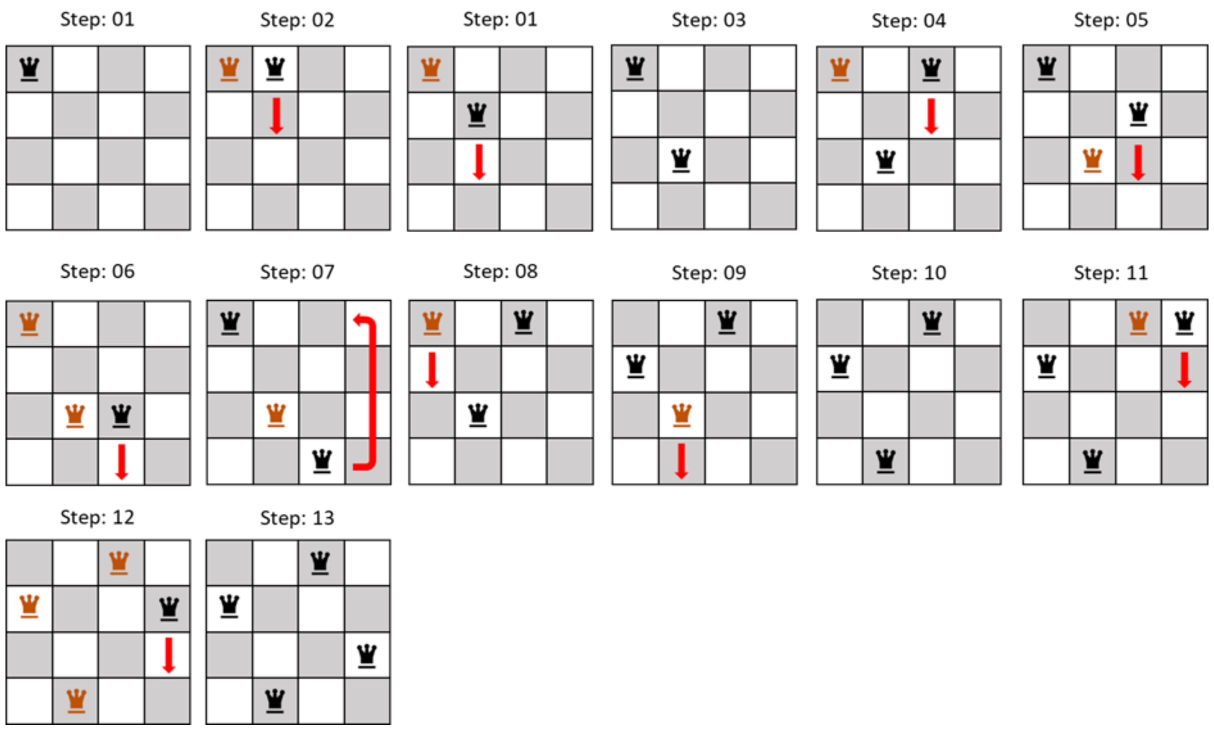


Figure 3. Step-by-step execution of the NSCR algorithm for the 4-Queens problem.

3. Results

Table 1 presents a comparative analysis of four algorithms, including NSCR, BFC, Lookahead, and CSP. Each experimental test was conducted three times to ensure the reliability and consistency of the results. The reported values represent the average of these three runs. It was observed that the outcomes were consistent across all instances, with negligible variation. This averaging process has been included to provide a more accurate representation of the algorithm's performance in terms of their runtime and memory usage when applied to the N-Queens problem with varying numbers of queens (n = 8, n = 12, n = 20, and n = 25). The newly developed NSCR algorithm demonstrates a consistent and efficient performance, with its runtime gradually increasing from 1.22 ms for n = 8 to 2.38 ms for n = 25. This suggests that NSCR scales well as the problem size increases, likely exhibiting a low polynomial time complexity for the given number of queens. In terms of memory usage, NSCR also proves to be resource-efficient, with a modest increase from 0.312 KB to 0.457 KB across the same range of problem sizes.

Table 1. Comparative analysis of algorithm performance for the N-Queens problem.

Size (n)	NSCR Runtime (ms)	BFC Runtime (ms)	Lookahead Runtime (ms)	CSP Runtime (ms)	NSCR Memory (KB)	BFC Memory (KB)	Lookahead Memory (KB)	CSP Memory (KB)
8	1.22	3.12	2.44	0.34	0.312	2.570	0.398	0.062
12	1.62	3.4	2.46	0.36	0.352	2.586	0.422	0.078
20	1.69	2.30	32.67	6.60	0.422	2.625	2.445	0.109
25	2.38	3.38	169.95	10.94	0.457	2.648	2.469	0.133

In comparison, the BFC algorithm exhibits a slightly higher runtime than NSCR, starting at 3.12 ms for n = 8, and, while it briefly improves at n = 20, it increases again at n = 25 to 3.38 ms. This fluctuation in performance suggests that while BFC is effective for certain configurations, it may not consistently handle larger instances as efficiently as NSCR. Its memory usage remains relatively stable but higher than NSCR, indicating that it consumes more resources while performing comparably or less efficiently.

The Lookahead algorithm, while initially competitive with runtimes close to those of NSCR at smaller problem sizes, shows a significant deterioration in performance as n increases. The runtime jumps dramatically from 2.46 ms at n = 12 to 32.67 ms at n = 20 and further to 169.95 ms at n = 25, revealing a steep exponential growth in complexity that makes it unsuitable for larger problem sizes. Its memory usage also increases notably as the problem size grows, further highlighting its inefficiency for more complex instances of the N-Queens problem.

On the other hand, the CSP algorithm starts with the lowest runtimes among all algorithms, demonstrating excellent efficiency for smaller problems with only 0.34 ms for n = 8. However, as n increases, its runtime grows more significantly, reaching 10.94 ms at n = 25, suggesting that while CSP is very efficient for small-scale problems, its performance diminishes as the complexity increases. Nevertheless, CSP consistently uses the least amount of memory, maintaining a minimal footprint from 0.062 KB to 0.133 KB, which makes it particularly resource efficient.

Figure 4 depicts the runtime of the Iterative Conflict Resolution (NSCR) algorithm as a function of the number of queens 'n' ranging from 8 to 1000. The data reveal a non-linear increase in computational time, with the curve exhibiting a gradual ascent that becomes markedly steeper as 'n' approaches the upper end of the spectrum. This trend suggests that the algorithm's time complexity scales at a rate slightly faster than linear, potentially indicating a polynomial relationship. The observed increase in runtime is consistent with the expected growth in computational overhead required to manage the escalating complexity of resolving conflicts as the problem size expands.

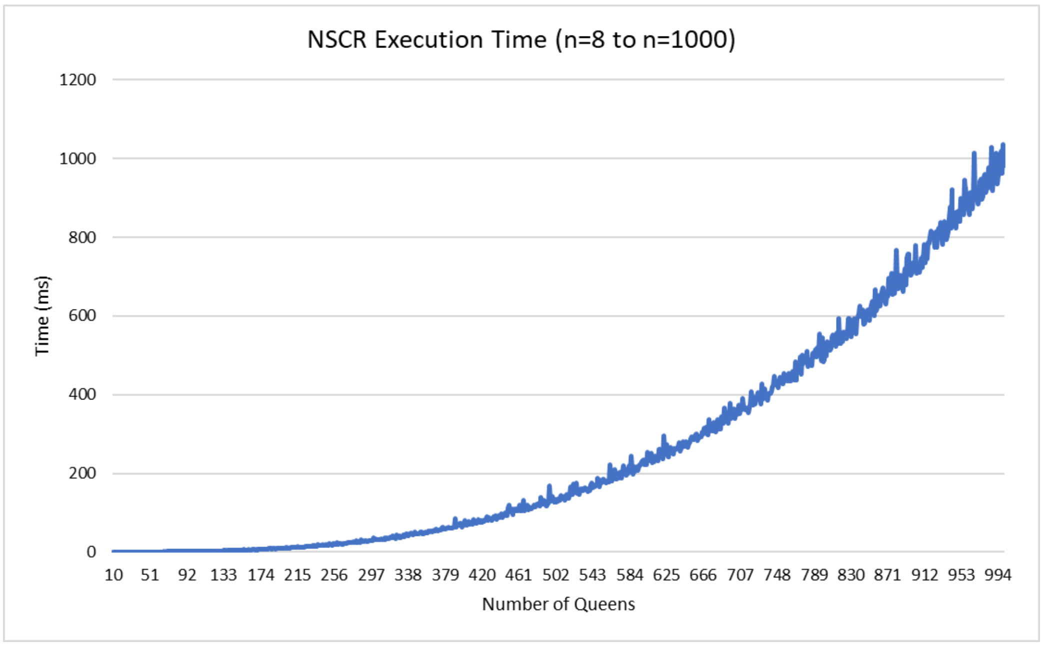


Figure 4. Runtime of the NSCR algorithm as a function of the number of queens (n).

Figure 5 presents the memory utilization of the NSCR algorithm relative to the number of queens. The linear trajectory of the graph indicates a direct proportionality between the number of queens and the memory consumed by the algorithm. This linear scaling suggests that the NSCR algorithm maintains a constant memory footprint per additional queen, demonstrating its efficiency in terms of memory usage. Such linear memory growth underscores the algorithm's scalability, making it particularly advantageous for handling large instances of the N-Queens problem where memory constraints are a critical consideration.

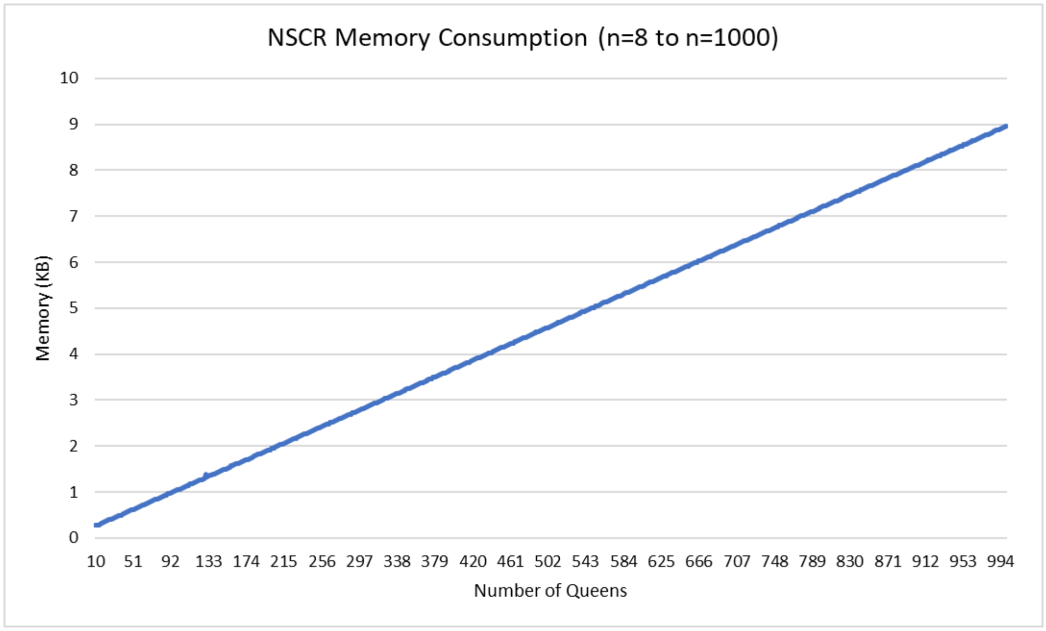


Figure 5. Memory utilization of the NSCR algorithm across varying numbers of queens (n).

Figure 6 illustrates the number of iterations (or steps) executed by the NSCR algorithm as the number of queens increases. The exponential curve depicted in the graph indicates a rapid escalation in the number of iterations required as the problem size grows. This exponential trend highlights the increasing computational complexity faced by the algorithm when attempting to resolve conflicts in larger configurations. The significant rise in iteration count for higher values of n underscores the intensive nature of the algorithm's conflict resolution process, reflecting the substantial increase in the number of potential conflicts that must be managed as the board size expands.

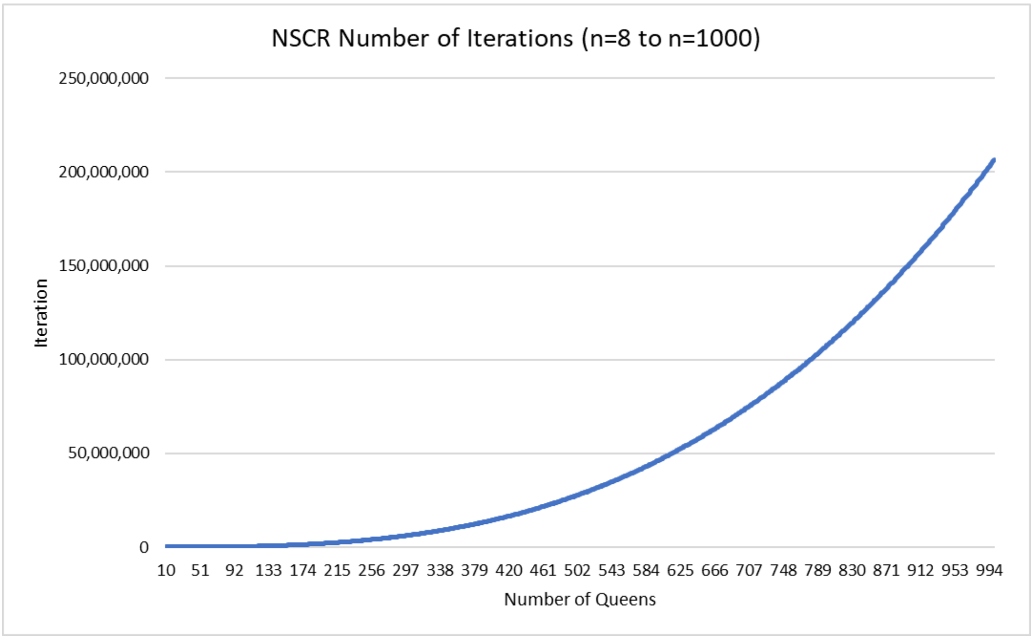


Figure 6. Number of iterations of the NSCR algorithm across varying numbers of queens (n).

4. Discussion

The performance analysis of the NSCR algorithm, demonstrated through a series of experiments, reveals its considerable advantages over traditional methods for solving the N-Queens problem. Unlike conventional backtracking algorithms, which suffer from exponential time complexity, the NSCR algorithm manages to achieve a more manageable growth in runtime. This improvement is likely due to its hybrid approach, which combines elements of iterative repair and conflict resolution within a backtracking framework. The algorithm's ability to dynamically adjust queen positions, rather than immediately backtracking upon detecting a conflict, significantly enhances its efficiency, particularly for larger problem sizes.

When compared to other algorithms, the NSCR algorithm showcases a balanced performance. While the CSP algorithm performs strongly for smaller problem sizes, it struggles to maintain its efficiency as the problem size increases. Conversely, the Lookahead algorithm, despite its initial promise, fails to scale effectively, with its runtime increasing exponentially as the number of queens grows. Although the BFC algorithm offers more stability, it does not consistently outperform NSCR, especially in larger instances. A significant advantage of NSCR is its linear memory usage, which ensures that the algorithm remains practical even as the problem size scales up to 1000 queens. This contrasts sharply with the Lookahead algorithm, which not only suffers from an escalating runtime but also shows increased memory consumption, making it less viable for large-scale problems. The findings suggest that the NSCR algorithm is not only a viable alternative to traditional methods, but is also a superior choice for solving large-scale N-Queens problems, offering a balanced trade-off between computational efficiency and resource utilization.

The performance of the NSCR algorithm was evaluated across problem sizes ranging from 8 to 25 queens. As ‘n’ increased beyond 25, the other algorithms, such as BFC, Lookahead, and CSP, experienced significantly longer runtimes, causing the computer to halt. Therefore, the comparative analysis was conducted up to n = 25, where the NSCR algorithm consistently demonstrated superior efficiency and scalability. For example, the NSCR algorithm achieved an average runtime of 1.22 ms for n = 8, compared to 3.12 ms for BFC and 2.44 ms for Lookahead, and maintained a manageable performance as the problem size increased.

On the other hand, it was noticed that the NSCR algorithm encounters an issue when the number of queens is six, leading to an infinite loop. This problem arises due to the nature of the algorithm, where for n = 6, the first and last queens perpetually threaten each other, preventing a solution. To address this, instead of placing the first queen in the first row, first column, it is recommended to place it in the fourth row, first column. The NSCR algorithm can be viewed as a specialized variant of backtracking that integrates iterative repair and conflict resolution within the backtracking framework. Unlike the traditional “abandon and retry” approach, it dynamically attempts to resolve conflicts before deciding to backtrack. While it shares the high-level objective of incrementally constructing a solution and backtracking when needed, its implementation is more complex, incorporating additional steps to optimize the search process, making it distinct from standard backtracking methods.

The NSCR algorithm operates deterministically by always placing the first queen in the first row and first column, leading to consistent behavior across different runs for the same value of ‘n’. The time complexity of the algorithm is determined by its key operations: initializing lists with O(n) complexity, checking for threats with O(n) complexity, and resolving conflicts through the queen placement function, which operates in O(n^2) time. As these steps are consistently executed in the same sequence for a given n, the overall time complexity can reach O(n^3) in the worst case, particularly when extensive conflict resolution is necessary. This worst-case scenario typically occurs in configurations with dense conflicts, where multiple queens require iterative adjustments across multiple columns. In such cases, the algorithm must perform repeated conflict resolution steps, resulting in increased computational effort. However, the NSCR algorithm employs a hybrid approach that combines iterative repair and conflict resolution strategies, which helps to mitigate the impact of worst-case complexity. By dynamically adjusting queen positions rather than immediately resorting to backtracking, the algorithm effectively reduces the number of iterations needed to resolve conflicts. Consequently, while O(n^3) complexity represents the theoretical worst case, the practical performance of NSCR often exhibits significantly lower time complexity, with the runtime only increasing from 1.22 ms to 2.38 ms between n = 8 and n = 25, demonstrating a more manageable growth. The space complexity remains O(n) due to the storage requirements for the lists used in the algorithm. This analysis highlights that while the algorithm’s behavior is consistent, it can still exhibit high computational costs in complex scenarios, such as when n = 6, where specific difficulties arise.

5. Conclusions

The NSCR algorithm introduced in this study represents a significant advancement in solving the N-Queens problem, particularly when dealing with large-scale instances. Through extensive comparative analysis, the NSCR algorithm has consistently demonstrated its superiority over traditional methods, including BFC, Lookahead, and CSP algorithms. This performance is particularly notable in terms of both runtime and memory efficiency, where the NSCR algorithm excels by maintaining low polynomial time complexity and linear memory growth, even as the problem size increases.

The hybrid nature of NSCR, which effectively integrates dynamic conflict resolution within a backtracking framework, is central to its success. This approach allows the algorithm to adaptively manage conflicts among queens, avoiding the pitfalls of conventional methods that often suffer from exponential growth in complexity. By dynamically adjusting queen positions rather than resorting to immediate backtracking, NSCR optimizes the search process, making it both more efficient and more scalable. This capability is especially critical in large-scale scenarios where traditional algorithms may falter or become impractical due to resource constraints.

Moreover, the consistent performance of NSCR across various problem sizes highlights its reliability and versatility. Unlike some traditional approaches that may perform well only under specific conditions or for smaller problem sizes, NSCR offers a balanced solution that can handle a wide range of instances effectively. Its linear memory usage further underscores its practicality, ensuring that the algorithm remains feasible even for extremely large configurations.

In summary, the NSCR algorithm not only addresses the limitations of existing methods but also sets a new standard for solving the N-Queens problem. Its ability to scale efficiently and maintain performance across different scenarios makes it a valuable tool for researchers and practitioners in combinatorial optimization. The innovative strategies embedded within NSCR provide a strong foundation for future research, potentially inspiring the development of even more advanced algorithms that can tackle complex combinatorial challenges with greater efficiency and effectiveness.

Author Contributions: Conceptualization, O.M.; Algorithm Development, O.M.; Validation, O.M. and A.A.; Formal Analysis, A.A.; Methodology, A.A.; Investigation, A.A.; Resources, A.A.; Data Curation, A.A.; Writing—Original Draft Preparation, O.M.; Writing—Review and Editing, A.A.; Visualization, A.A.; Supervision, O.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data supporting the findings of this study are available upon reasonable request from the corresponding author.

Acknowledgments: The authors acknowledge the use of ChatGPT by OpenAI solely for the purpose of paraphrasing certain sections of this manuscript. The authors conducted all other aspects of the research, analysis, and writing independently.

Conflicts of Interest: Author Omid Moghimi was employed by the company REDARC Electronics Pty., Ltd., The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Izadkhah, H. Backtracking Algorithms. In *Problems on Algorithms*; Springer: Cham, Switzerland, 2022. [CrossRef]
2. Rakhya, S.; Singh, S. Rakhya's method: A case-based approach to solve n-queens problem. In Proceedings of the 2014 11th International Conference on Electronics, Computer and Computation (ICECCO), Abuja, Nigeria, 29 September–1 October 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 1–8. [CrossRef]
3. Sosic, R.; Gu, J. Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Trans. Knowl. Data Eng.* **1994**, *6*, 661–668. [CrossRef]
4. Kralev, V.; Kraleva, R.; Chakalov, D. Development of an Application for Interactive Research and Analysis of the N-Queens Problem. *Int. J. Adv. Sci. Eng. Inf. Technol.* **2021**, *11*, 1811–1818. [CrossRef]
5. Ayub, M.A.; Kalpoma, K.A.; Proma, H.T.; Kabir, S.M.; Chowdhury, R.I.H. Exhaustive study of essential constraint satisfaction problem techniques based on N-Queens problem. In Proceedings of the 2017 20th International Conference of Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 22–24 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–6. [CrossRef]
6. Stone, H.S.; Stone, J.M. Efficient search techniques—An empirical study of the N-Queens Problem. *IBM J. Res. Dev.* **1987**, *31*, 464–474. [CrossRef]
7. Lynce, I.; Manquinho, V.; Marques-Silva, J. Backtracking. In *Wiley Encyclopedia of Computer Science and Engineering*; Wiley: Hoboken, NJ, USA, 2009; pp. 283–289. [CrossRef]
8. Gaschnig, J. A general backtrack algorithm that eliminates most redundant tests. In Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI), Cambridge, MA, USA, 22–25 August 1977; p. 457.
9. Agarwal, K.; Sinha, A.; Hima Bindu, M. A Novel Hybrid Approach to N-Queen Problem. In *Computational Science and Its Applications—ICCSA 2012*; Springer: Berlin, Germany, 2012; pp. 519–527. [CrossRef]

10. Jain, V.; Prasad, J.S. Solving N-queen Problem Using Genetic Algorithm by Advance Mutation Operator. *Int. J. Electr. Comput. Eng. (IJECE)* **2018**, *8*, 4519–4523. [CrossRef]
11. El Abidine, A.Z. An Incremental Algorithm to Solve the N-Queens Problem in Polynomial Time. *Alex. Eng. J.* **2023**, *67*, 157–167. [CrossRef]