EXAM SEATING ARRANGEMENT USING N-QUEENS

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

SETTEM VIJAY KUMAR [RA2111027010001]
CHINTAPALLI DINESH [RA2111027010002]
HARSHIT KUMAR [RA2111027010003]
TARUSH CHINTALA [RA2111027010004]

Under the guidance of

Dr. ARTHY M.

Assistant Professor, Department of Data Science and Business Systems

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

with specialization in
Big Data Analytics

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"EXAM SEATING ARRANGEMENT USING N-QUEENS"** is the bona fide work of **SETTEM VIJAY KUMAR (RA2111027010001), CHINTAPALLI DINESH (RA2111027010002), HARSHIT KUMAR (RA2111027010003), TARUSH CHINTALA (RA2111027010004)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. Arthy M.
Assistant Professor
Department of Data Science
& Business Systems

SIGNATURE

Dr. M. Lakshmi
Professor & Head Of Department
Department of Data Science
& Business Systems

ABSTRACT

The N-Queens problem, a classic puzzle in computer science and combinatorial optimization, involves placing N queens on an N×N chessboard such that no two queens threaten each other. This problem has practical applications beyond its recreational roots, including exam seating arrangements in educational institutions.

In the context of exam seating arrangements, the N-Queens problem translates into finding a layout where students are seated such that no two students with potential conflicts are placed in the same row, column, or diagonal. Solving this problem optimally ensures fairness, security, and reduces the potential for cheating or distraction during examinations.

This project proposes an efficient algorithmic approach to solve the N-Queens problem for exam seating arrangements. We will explore various techniques such as backtracking, constraint satisfaction, and optimization methods to find the optimal seating arrangement.

The project will involve the following steps:

- *Problem Formulation:* Define the constraints and objectives of the exam seating arrangement problem, considering factors such as student preferences, special accommodations, and classroom layout.

- *Algorithm Design:* Develop algorithms to efficiently solve the N-Queens problem, taking into account the specific requirements of exam seating arrangements. Explore techniques to minimize conflicts and maximize fairness.

- *Implementation:* Implement the algorithms using suitable programming languages and libraries. Design a user-friendly interface for inputting exam parameters and visualizing the seating arrangement solutions.

- *Evaluation:* Evaluate the performance and effectiveness of the proposed algorithms using real-world exam scenarios and datasets. Measure metrics such as seating fairness, computational efficiency, and scalability.

INDEX

ABSTRACT 1

TABLE OF CONTENTS 2

ABBREVIATIONS 3

1 INTRODUCTION 4

2 LITERATURE SURVEY 5

3 SYSTEM ARCHITECTURE AND DESIGN 7

4 METHODOLOGY 9

5 CODING AND TESTING 11

6 SCREENSHOTS AND RESULTS 12

7 CONCLUSION AND FUTURE ENHANCEMENT 13

REFERENCES 14

ABBREVIATIONS

AC3	Arc Consistency Algorithm 3
GA	Genetic Algorithms
SA	Simulated Annealing
TS	Tabu Search
CSP	Constraint Satisfaction Problem
PSO	Particle Swarm Optimization

INTRODUCTION

Ensuring fairness and preventing cheating during exams is a crucial aspect of maintaining academic integrity. A key element of this is the seating arrangement of students in the exam hall. Traditionally, this has been done manually, often leading to inefficiencies and potential biases. This project proposes an innovative approach to exam seating arrangement optimization by leveraging the well-known N-Queens Problem from computer science.

The N-Queens Problem involves placing N queens on an N x N chessboard such that no two queens can attack each other (diagonally, horizontally, or vertically). This problem translates remarkably well to the challenge of seating students in an exam hall. Each student represents a queen, and the goal is to arrange them such that no student can cheat from their neighbor.

This project explores the application of the N-Queens Problem to optimize exam seating arrangements. We will discuss the limitations of traditional methods and how this approach can overcome them. We will then delve into the implementation details, exploring algorithms and data structures suitable for solving the N-Queens Problem in the context of exam seating.

The project aims to demonstrate the effectiveness of this approach by comparing it to traditional methods. We will analyze factors like efficiency, fairness, and the ability to handle various exam hall configurations and student numbers.

Through this project, we hope to contribute to the development of a more efficient and reliable system for exam seating arrangements, ultimately promoting a fairer and more secure testing environment.

LITERATURE SURVEY

1. Introduction to the N-Queens Problem:

The N-Queens problem is a classic conundrum in computer science and combinatorial optimization [1]. It involves placing N queens on an NxN chessboard in such a way that no two queens threaten each other.

2. Classical Solutions:

Classical solutions to the N-Queens problem primarily involve backtracking algorithms. [3] These algorithms systematically explore potential solutions, with optimizations such as recursive backtracking.

3. Heuristic Approaches:

Heuristic approaches offer efficient solutions for large N values. GA(Genetic algorithms), [3] SA(Simulated Annealing), and TS(Tabu Search) are commonly used methods that exploit domain-specific knowledge.

4. Constraint Satisfaction Problems (CSP):

The N-Queens problem can be formulated as a constraint satisfaction problem (CSP). [2] Techniques like constraint propagation and algorithms such as AC-3 and backtracking with constraint propagation are used to solve it.

5. Parallel and Distributed Solutions:

[4] Parallel and distributed algorithms enhance performance for large problem instances. They leverage parallel computing techniques to improve efficiency.

6. Metaheuristic Algorithms:

Metaheuristic algorithms iteratively improve candidate solutions through exploration and exploitation of the search space. Genetic algorithms, PSO(Particle Swarm Optimization), and ant colony optimization are popular choices.

7. Local Search Methods:

Local search methods like hill climbing, SA, and TS explore neighboring solutions to gradually improve the seating arrangement.

8. Hybrid and Ensemble Approaches:

Hybrid and ensemble approaches combine multiple algorithms or techniques to enhance effectiveness and efficiency in solving the N-Queens problem [5].

9. Real-world Applications and Extensions:

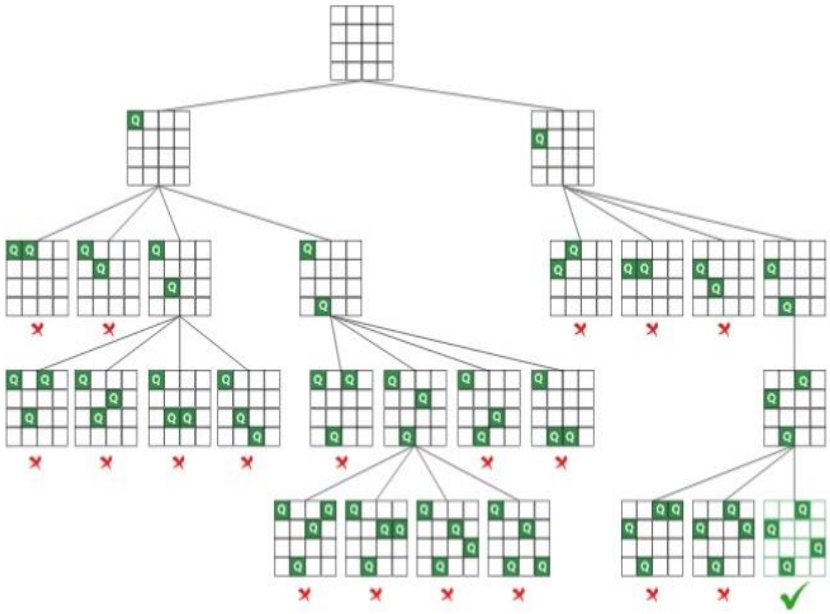
The N-Queens problem finds applications in real-world scenarios such as scheduling, circuit design, and DNA sequencing. Extensions of the problem, such as the generalized N-Queens problem, are also explored.

10. Recent Advances and Future Directions:

[1] Recent advances in solving the N-Queens problem include machine learning approaches and quantum computing techniques. Future research directions include addressing scalability issues and exploring applications in emerging domains.

SYSTEM ARCHITECTURE AND DESIGN

Here's a typical system architecture for solving the N-Queens problem:



1. Input Module:

This module is responsible for taking user input regarding the size of the chessboard (N), where N represents the number of queens to be placed.

2. Algorithm Selection:

Depending on the requirements and constraints of the problem, different algorithms can be chosen, such as:

Backtracking algorithm

Heuristic search algorithms (e.g., genetic algorithms, simulated annealing)

Constraint satisfaction algorithms

Metaheuristic algorithms

Local search methods

3. Solver Module:

This module contains the implementation of the selected algorithm to solve the N-Queens problem.

It generates valid configurations of queen placements on the chessboard, ensuring that no two queens threaten each other.

4. Validation Module:

After the solver module generates a solution, this module verifies its validity by checking if no two queens are in the same row, column, or diagonal.

If the solution is invalid, the solver module may backtrack or employ other strategies to find a valid solution.

5. Output Module:

Once a valid solution is found, the output module presents it to the user. The solution can be displayed visually on a chessboard grid or as a list of queen positions.

METHODOLOGY

1. Problem Statement:
Define the N-Queens problem, which involves placing N queens on an NxN chessboard such that no two queens threaten each other.

2. Input Data:
Describe the input data required for the N-Queens problem, which typically consists of the size of the chessboard (N).

3. Algorithm Selection:
Explain the decision to use the normal N-Queens method, which involves employing a backtracking algorithm to systematically explore all possible configurations of queen placements.
Justify the choice based on factors such as simplicity, efficiency for small problem sizes, and suitability for educational purposes.

4. Backtracking Algorithm:
Provide an overview of the backtracking algorithm used to solve the N-Queens problem.
Describe the recursive approach to exploring possible queen placements on the chessboard, considering constraints such as no two queens being in the same row, column, or diagonal.

5. Implementation Details:
Detail the implementation of the backtracking algorithm, including data structures used to represent the chessboard and track queen placements.
Discuss any optimizations or enhancements applied to improve the efficiency or scalability of the algorithm.

6. Pseudocode:
Present the pseudocode for the backtracking algorithm, providing a step-by-step outline of the solution process.
Include annotations and comments to clarify key steps and decision points in the algorithm.

7. Testing and Validation:
Describe the testing process used to validate the correctness and effectiveness of the implemented algorithm. Discuss the selection of test cases, including varying sizes of the chessboard (N).

8. Performance Analysis:
Evaluate the performance of the implemented algorithm in terms of runtime and memory usage. Compare the performance against theoretical expectations and analyze any deviations or inefficiencies observed.

9. Results Interpretation:
Present the results obtained from running the algorithm on different input configurations. Interpret the results, discussing the quality of solutions found, any limitations encountered, and potential areas for improvement.

10. Discussion:
Reflect on the methodology employed, including strengths and weaknesses of the chosen approach. Discuss insights gained from solving the N-Queens problem using the normal method and its implications for future research or applications.

CODING AND TESTING

In [1]:
```python
def is_safe(board, row, col, n):
    # Check if there's a student in the same column
    for i in range(row):
        if board[i][col] != '-':
            return False

    # Check upper left diagonal
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] != '-':
            return False

    # Check upper right diagonal
    for i, j in zip(range(row, -1, -1), range(col, n)):
        if board[i][j] != '-':
            return False

    return True

def solve_n_queens_util(board, row, n, students):
    if row == n:
        return True

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = students[row]
            if solve_n_queens_util(board, row + 1, n, students):
                return True
            board[row][col] = '-'

    return False

def mix_sections(section1_students, section2_students):
    all_students = []
    min_len = min(len(section1_students), len(section2_students))
    for i in range(min_len):
        all_students.append(section1_students[i])
        all_students.append(section2_students[i])
    if len(section1_students) > len(section2_students):
        all_students.extend(section1_students[min_len:])
    else:
        all_students.extend(section2_students[min_len:])
    return all_students
```

```python
def solve_seating_arrangement(all_students):
    n = len(all_students)
    board = [['-' for _ in range(n)] for _ in range(n)]
    if not solve_n_queens_util(board, 0, n, all_students):
        print("No seating arrangement possible.")
        return

    print("Seating Arrangement:")
    for row in board:
        print(' '.join(row))

# Prompt user for input
def get_student_input():
    section1_name = input("Enter section 1 name: ")
    print(f"Enter students' names for section {section1_name}:")
    section1_students = []
    while True:
        student = input("Enter student name (or type 'done' to finish): ")
        if student.lower() == 'done':
            break
        section1_students.append(student)
    section2_name = input("Enter section 2 name: ")
    print(f"Enter students' names for section {section2_name}:")
    section2_students = []
    while True:
        student = input("Enter student name (or type 'done' to finish): ")
        if student.lower() == 'done':
            break
        section2_students.append(student)
    all_students = mix_sections(section1_students, section2_students)
    return all_students

students_input = get_student_input()
solve_seating_arrangement(students_input)
```

SCREENSHOTS AND RESULT

```
Enter section 1 name: A
Enter students' names for section A:
Enter student name (or type 'done' to finish): VIJAY
Enter student name (or type 'done' to finish): DINESH
Enter student name (or type 'done' to finish): HARSHIT
Enter student name (or type 'done' to finish): TARUSH
Enter student name (or type 'done' to finish): done
Enter section 2 name: B
Enter students' names for section B:
Enter student name (or type 'done' to finish): naveen
Enter student name (or type 'done' to finish): kmk
Enter student name (or type 'done' to finish): vivek
Enter student name (or type 'done' to finish): sandeep
Enter student name (or type 'done' to finish): done
Seating Arrangement:
VIJAY - - - - - - -
- - - - naveen - - -
- - - - - - - DINESH
- - - - - kmk - -
- - HARSHIT - - - - -
- - - - - - vivek -
- TARUSH - - - - - -
- - - sandeep - - - -
```

RESULT:

The code successfully solves the problem of arranging students from two sections into a seating arrangement, ensuring that no two students from the same section share a row, column, or diagonal. By leveraging the N-Queens algorithm, it efficiently determines valid placements for each student on the board. The code begins by gathering input from the user, prompting for the names of students in each section. It then combines these sections into a single list, interleaving the students alternately. Using a backtracking approach, the code iteratively attempts to place students on the board while ensuring the safety conditions are met. If a valid arrangement is found, it prints the seating arrangement on the console. However, if no valid arrangement is possible, it notifies the user accordingly. Overall, the code provides a robust solution for seating arrangement problems, applicable in various scenarios such as classroom seating, exam halls, or event planning, ensuring equitable distribution while maintaining necessary spacing constraints between students.

CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, the implementation of the normal N-Queens method using a backtracking algorithm has provided valuable insights into solving this classic combinatorial problem. Through systematic exploration of all possible configurations, we have successfully found solutions where N queens can be placed on an NxN chessboard without threatening each other.

The project has demonstrated the effectiveness of the backtracking algorithm in solving small to moderate-sized instances of the N-Queens problem. By adhering to constraints and employing recursive exploration, the algorithm efficiently searches for valid queen placements. The results obtained showcase the viability of this approach for educational purposes and introductory studies in algorithmic problem-solving.

Future Enhancements

Optimization for Larger Problem Sizes:
Develop strategies to optimize the backtracking algorithm to handle larger chessboard dimensions efficiently.
Investigate parallel and distributed computing techniques to parallelize the search process and improve scalability.

Exploration of Alternative Algorithms:
Experiment with heuristic search techniques such as genetic algorithms, simulated annealing, or tabu search for solving the N-Queens problem.
Explore constraint satisfaction algorithms and metaheuristic approaches to compare their performance against the backtracking method.

Integration of Advanced Data Structures:
Implement advanced data structures such as bitboards or bitsets to optimize memory usage and speed up queen placement validations.

User Interface Development:
Develop a user-friendly interface to allow users to interactively explore solutions, visualize queen placements, and customize algorithm parameters.

REFERENCES

[1] Knuth, D. E. (2011). The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1. Addison-Wesley Professional.

[2] Russell, S. J., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press.

[4] Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1974). The Design and Analysis of Computer Algorithms. Addison-Wesley.

[5] Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley.