JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Design and Evaluation of Algorithms on Backtracking

[1]Mayuri Sahay, [2]Purnima Gupta
[1]Student of Bachelor of Computer Application, [2]Assistant Professor of Bachelor of Computer Application
[1,2]Institute of Management Studies, Ghaziabad (University Courses Campus)
Uttar Pradesh, India

Abstract
The term backtracking search is used for depth-first search that elects values for a single variable at once and backtracks when the same has no appropriate values left to designate. This study sheds light on the strengths, weaknesses, and nuances of backtracking as an essential problem-solving methodology. It is easy to implement and understand as it follows a neutral trial-and-error approach. It is complete it can find a solution if one exists or prove that none exists by exploring the entire solution space. Backtracking can be described as a general algorithm technique which searches every possible solution for solving a computational equation.

I. Introduction
Backtracking is a problem-solving algorithmic technique that is used to find every single one (or few) solutions to a problem by exploring all possible candidates and eliminating those that do not satisfy the problem's constraints or requirements [5].
It is a recursive approach that involves a systematic trial and error method for solving a problem by incrementally building candidates to the solution, and when one of the candidate solutions fails to meet the constraints, the algorithm backtracks and tries a new candidate solution. Backtracking follows the Brute Force methodology [3], [4].It follows the Depth-first search approach. More than one solution can exist for a particular problem solved using backtracking [9]. The Backtracking process begins by initiating a search from a starting point and proceeds by iteratively constructing a potential solution. At each step, the algorithm evaluates the feasibility of the current candidate by checking if it satisfies the problem's constraints. If the candidate solution is valid, the algorithm continues its exploration further, incrementing towards the solution. However, if the constraints are not met, the algorithm backtracks to the previous step and explores a different candidate solution. This cycle repeats until a valid solution is found or all possible candidates have been examined.

Index terms: Constraint Satisfaction, Data Structure, Optimization, Problem Formulation

II. Related work
A method for structuring and utilising information effectively to make it understandable [3], [6], [5]. The approach involves sequentially positioning queens in distinct columns, commencing with the leftmost columns. As each queen is placed, we assess for conflicts with queens already in place [17], [15]. To position N-queens on an n * n Chessboard in such a way that they don't threaten each other by sharing the diagonal [21], [10], [16]. The chess board is progressively populated with queens, one by one, and reversed if a valid solution isn't discovered. At each stage, the algorithm verifies whether the newly positioned queen clashes with the queens placed earlier, and if it does, it performs a backtrack [25], [23]. We utilise a 2-D array with N line and N columnar, and since recursion is employed, the recursive stack space grows linearly. Consequently, the total space complexity amounts to O(N^2) [23]. The Queens problem is tackled using the backtracking algorithm, which systematically attempts to position the queens of the chessboard columns to columns. It verifies the validity of each placement and reverts if it proves to be invalid [10], [11], [17]. The algorithm of backtracking is executed by creating a tree structure of decisions referred to as the "search tree "or "state space tree" [27], [16], [15]. A broad algorithm approach that involves exploring all potential combinations to address a computational problem [7], [3], [6]. Solves problems incrementally through a step-by-step process, gradually advancing complexity by employing recursive calls [9]. In this context, we seek a workable solution. In optimization problems, we aim to find the perfect or most favourable solution [1], [2], [3], [4]. Position N- queens on an N * N chessboard in a way that ensures none of the queens can threaten each other, whether horizontally, vertically, or diagonally [12], [16], [28].

Literature survey

Data Integration Scheme	Proposed By	Year	Strength	Weakness
Artificial Intelligence A Modern Approach" Third Edition[3]	Stuart j. Russell, Peter Norvig	1995	• Increases accuracy and precision • Reduces error • Automating repetitive tasks	• Costly implemented • Anticipated reduction in human employment • Absence of emotion and creativity
"Backtracking Introduction "[9]	Abdul Bari	1962	• Solves constraint satisfaction problems • Helps to avoid infinite recursion	• Complex execution • Sophisticated • Antiquated
A multipurpose backtracking algorithm. *Journal of Symbolic Computation [1]*	Priestley, H. A., & Ward, M. P.	1994	• Very intuitive to code • Easy to implement • Contains less LOC	• Very time inefficient • Large space complexity • Function information is stored in stacks
"A Multipurpose Backtracking Algorithm"[7]	Martin Ward, Hilary Priestley	1994	• Straightforward to understand • Avoids the need for complex data structures or algorithm	• High computational cost • Uses a lot of memory and the CPU
Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering [12]*	Sosic, R., & Gu, J.	1994	• Conflict minimization • Parallelization	• This leads to suboptimal solutions • Rely on heuristics and probabilistic methods
An Exhaustive study of essential constraint satisfaction problem techniques based on the N-Queens problem. In *2017 20th International*	Ayub, M. A., Kalpoma, K. A., Proma, H. T., Kabir, S. M., & Chowdhury, R. I. H. ().	2017, December	• Theoretical foundation • Provides benchmark problems for evaluating and comparing CSP algorithms	• Problems need to be specified • Complex • Outdated

Conference of Computer and Information Technology (ICCIT) (pp. 1-6). IEEE[11]				
N-Queens solving algorithms by sets and backtracking. In *SoutheastCon 2016* (pp. 1-8). IEEE.[10]	Güldal, S., Baugh, V., & Allehaibi, S.	2016, March	• Efficiency • Scalable • Optimizable	• Exponentially time complex • Deterministic in nature • Complex debugging
A Multipurpose Backtracking Algorithm (Doctoral dissertation, Ph. D. thesis, Mathematical Institute 24/29, St. Giles Oxford OX1 3LB).[27]	Priestley, H. A., & Ward, M. P	2003	• Versatile • Highly efficient • Improves efficiency and effectiveness	• Complex implementation • Search space may be extensive
N-Queens solving algorithms by sets and backtracking. In *SoutheastCon 2016* (pp. 1-8). IEEE[25]	Güldal, S., Baugh, V., & Allehaibi, S. ().	2016, March	• Efficiency • Scalable • Optimizable	• Exponentially time complex • Deterministic nature • Complex debugging
N-queens pattern generation: An insight into space complexity of a backtracking algorithm.[23]	Bozinovski, A., & Bozinovski, S.	2004, June	• Visualise and explore different solutions to a problem • Algorithmic testing • Variability	• Exponential growth • Lack of optimization information • Limited generalisation
Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering,*	Sosic, R., & Gu, J. ().	1994	• Efficient for large problem spaces • Heuristic nature • Easy implementation	• Local optima • Solution quality variability • Difficulty in handling constraints

6(5), 661-668.[21]				
An incremental approach to the n-queen problem with polynomial time. Journal of King Saud University-Computer and Information Sciences, 35(3),1-7 [17]	Abidine, B. Z.	2023	• Step-by-step construction • Potential and optimization • Simplifies • debugging	• Don't guarantee a polynomial time solution • Often relies on the choice of heuristics • Limited applicability

III. Brute Force methodology

In backtracking, the brute force methodology requires calculating all possible candidates for the solution and then testing each candidate for validity. This methodology leads to an exponentially high number of candidates to be checked, making it impractical for large problems. However, in certain cases, brute force methodology may be the ideal approach, especially when there are no clear constraints or requirements.

One universal example of the Brute Force methodology is the problem of detecting the shortest path in a maze. The Brute Force methodology in backtracking would involve exploring all possible paths from the starting point to the destination point and then selecting the path with the shortest distance [23] [25]. However, this approach would require exploring all possible paths, which can be computationally expensive and impractical for larger mazes.

IV. Application of Backtracking

N-queens puzzle

The N-Queens puzzle which involves constraint satisfaction can be dealt by designating N chess queens on N×N chessboard [25]. Thus, in this solution no two queens pose issues for one other. A backtracking algorithm is ideal for solving such a problem.

Algorithm:-

N-Queens puzzle is solved by using Backtracking algorithm by taking following steps:

1. Leftmost column is where we start the process.
2. If the value = true when all queens are on the board.
3. Try every row in the active column. Following steps has to be taken for each row.
    a. If the queens can be parked in that row, identify that cell and place the queen.
    b. Make the same recursive call for the next column.
    c. If the recursive call returns true, then return true.
    d. If the recursive call returns false, unmark this cell. Move to the next row.
4. Value is equal to false if nothing worked after vetting all the rows. Backtracking is thus triggered

In the N-Queens puzzle, start by placing a queen in left column of the chessboard. Then try to park the next queen in the subsequent column, but not in the same row or diagonal as the previously placed queens. If we find a safe position for the queen, we mark the cell and recursively park the next queen.

If there comes a scenario of no safe positions left in the active column, Backtrack to the column minus one position (previous column) and try to park the queen into the previous column. This logic is followed till all the queens are parked on the board.

Pseudo Code for N-Queens Problem

```
Algorithm Place(k, i)
// returns true if Q is placed at the k-th row and i-th column otherwise returns false.
// x[] = global array
{

  for j=1 to (k-1) do
  {
    if(x[j]=i) //in the same column
    or (Abs(x[j]-i)+(Abs(j-k))) then //in the same diagonal
```

```
        return false
    }
 return true
 }

 Algorithm NQueens(k, n)
// Using backtracking, this procedure prints all combinations of solutions
{
    for i=1 and iterates up to the value of "n" do
    {
         if (Place(k, i)) then
         {
              x[k]=i
              if(k==n) then
                   print(x[1:n])
          else
              Nqueens(k+1, n)
          }
     }
}
```

"Brute force" solution is one of the many ways to resolve the N-Queen puzzle. Since this model runs through each and every possible combination, the N-queen puzzle will have O(n^n) complexity, which means the solution will run every combination on an NxN board, N sets, for N queens.

The N-Queens puzzle is an everyday indication of a constraint satisfaction puzzle solved by using the backtracking algorithm. The backtracking algorithm provides an efficient way to search through all possible solutions while pruning the search space by eliminating invalid candidates. By using the backtracking algorithm, we can efficiently calculate all possible combinations for the N-Queens puzzle, and also other similar problems that require the placement of objects on a board or grid, subject to certain constraints, can be solved using the backtracking algorithm.

V. Future works

In the future, we will try to enhance the work of backtracking Parallelization, Machine Learning Integration, Heuristic Selection, GPU Acceleration, and Constraint Propagation. These ideas can help modernise backtracking techniques and make them more effective and applicable in various domains. Depending on the specific application or problem, we may choose to focus on one or more of these approaches.

VI. Conclusion

Backtracking is a widely employed algorithmic approach utilised in data structures to recursively explore potential solutions while backtracking or reverting when they prove to be unsuccessful. This method involves trying out different alternatives and retracting those that do not result in a solution. Backtracking constructs a solution incrementally, step by step, discarding those attempts that fail to meet the problem's constraints at any given moment. It is a versatile algorithmic technique used in various problem-solving scenarios.

References

1. Priestley, H. A., & Ward, M. P. (1994). A multipurpose backtracking algorithm. *Journal of Symbolic Computation*, *18*(1), 1-40.
2. Van Beek, P. (2006). Backtracking search algorithms. In *Foundations of artificial intelligence* (Vol. 2, pp. 85-134). Elsevier.
3. j. Russell, Peter Norvig "Artificial Intelligence A Modern Approach" Third Edition
4. Backtracking Algorithm "GeeksforGeeks" 2023 https://www.geeksforgeeks.org/backtracking-algorithms/
5. Introduction to Backtracking " GeekForGeeks" 2023 https://www.geeksforgeeks.org/introduction-to-backtracking-data-structure-and-algorithm-tutorials/
6. Backtracking Algorithm "Huihoo.com" https://book.huihoo.com/data-structures-and-algorithms-with-object-oriented-design-patterns-in-c++/html/page446.html
7. Priestley, H. A., & Ward, M. P. (1994). A multipurpose backtracking algorithm. *Journal of Symbolic Computation*, *18*(1), 1-40.
8. "Constraints Satisfaction Problem backtracking algorithm" https://www.cs.miami.edu/home/visser/csc545-files/csp.pdf
9. Abdul Bari's "Backtracking Introduction " 2021 DAA UNIT-4 (Backtracking)
10. Güldal, S., Baugh, V., & Allehaibi, S. (2016, March). N-Queens solving algorithms by sets and backtracking. In *SoutheastCon 2016* (pp. 1-8). IEEE.
11. Ayub, M. A., Kalpoma, K. A., Proma, H. T., Kabir, S. M., & Chowdhury, R. I. H. (2017, December). An exhaustive study of essential constraint satisfaction problem techniques based on the N-Queens problem. In *2017 20th International Conference of Computer and Information Technology (ICCIT)* (pp. 1-6). IEEE.

12. Sosic, R., & Gu, J. (1994). Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering*, *6*(5), 661-668.
13. Jana, S., Mallik, M., Khan, A., Maji, A. K., & Pal, R. K. (2023). Design and Analysis of a Modified 3D Sudoku Solver. IEEE Access, 11, 27352-27368.
14. El Abidine, B. Z. (2023). An incremental approach to the n-queen problem with polynomial time. Journal of King Saud University-Computer and Information Sciences, 35(3), 1-7.
15. Arteaga, A., Orozco-Rosas, U., Montiel, O., & Castillo, O. (2022). Evaluation and Comparison of Brute-Force Search and Constrained Optimization Algorithms to Solve the N-Queens Problem. In New Perspectives on Hybrid Intelligent System Design based on Fuzzy Logic, Neural Networks and Metaheuristics (pp. 121-140). Cham: Springer International Publishing.
16. Wang, Z., Huang, D., Tan, J., Liu, T., Zhao, K., & Li, L. (2015). A parallel algorithm for solving the n-queens problem based on an inspired computational model. BioSystems, 131, 22-29.
17. El Abidine, B. Z. (2023). An incremental approach to the n-queen problem with polynomial time. Journal of King Saud University-Computer and Information Sciences, 35(3), 1-7.
18. Uehara, R., & Uehara, R. (2019). Backtracking. First Course in Algorithms through Puzzles, 111-127.
19. Sherine, A., Jasmine, M., Peter, G., & Alexander, S. A. (2023). *Algorithm and Design Complexity*. CRC Press.
20. Edelkamp, S. (2023). Programming Primer. In *Algorithmic Intelligence: Towards an Algorithmic Foundation for Artificial Intelligence* (pp. 3-33). Cham: Springer International Publishing.
21. Sosic, R., & Gu, J. (1994). Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering*, *6*(5), 661-668.
22. Sosic, R., & Gu, J. (1994). Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering*, *6*(5), 661-668.
23. Bozinovski, A., & Bozinovski, S. (2004, June). N-queens pattern generation: An insight into space complexity of a backtracking algorithm. In *Proceedings of the 2004 International Symposium on Information and Communication Technologies* (pp. 281-286).
24. Sosic, R., & Gu, J. (1994). Efficient local search with conflict minimization: A case study of the n-queens problem. *IEEE Transactions on Knowledge and Data Engineering*, *6*(5), 661-668.
25. Güldal, S., Baugh, V., & Allehaibi, S. (2016, March). N-Queens solving algorithms by sets and backtracking.In *SoutheastCon 2016* (pp. 1-8). IEEE.
26. Lijo, V. P., & Jose, J. T. (2015). Solving N-Queen Problem by Prediction. *Int. J. Comput. Sci. Inf. Technol*, *6*, 3844-3848.
27. Priestley, H. A., & Ward, M. P. (2003). *A Multipurpose Backtracking Algorithm* (Doctoral dissertation, Ph. D. thesis, Mathematical Institute 24/29, St. Giles Oxford OX1 3LB).
28. Lutati, B., Gontmakher, I., Lando, M., Netzer, A., Meisels, A., & Grubshtein, A. (2014). Agentzero: A framework for simulating and evaluating multi-agent algorithms. *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, 309-327.