University of Nottingham
UK | CHINA | MALAYSIA

Artificial Intelligence Methods (COMP2051 or AE2AIM)

Coursework Report: Variable Neighbourhood Search for Magic Square Problem

Fan Zhang (20320347, scyfz2)

Submit date: 2023.5.4

School of Computer Science University of Nottingham Ningbo China

Contents

1 Introduction

The objective of the Magic Square Problem is to fill a square grid with integers such that the sum of each row, column, and diagonal is identical. Despite its simple structure, large-order Magic Square Problems are considered a complex optimization challenge. At present, no polynomial time algorithm exists for this problem, so this problem can be transformed into the objective function optimization problem of magic squares.

$$\min \ obj = \sum_{i=0}^{n-1} \left| M - \sum_{j=0}^{n-1} x_{i,j} \right| + \sum_{j=0}^{n-1} \left| M - \sum_{i=0}^{n-1} x_{i,j} \right| + \left| M - \sum_{i=0}^{n-1} x_{i,i} \right| + \left| M - \sum_{i=0}^{n-1} x_{i,(n-i-1)} \right|$$

Figure 1: Objective function

Various methods have been developed to address the Magic Square Problem, including exact algorithms, heuristic algorithms, and metaheuristic algorithms. Among these, Variable Neighborhood Search (VNS) has emerged as a popular and effective approach for discovering high-quality solutions to combinatorial optimization problems. VNS is a metaheuristic algorithm that combines local search procedures with systematic exploration of the solution space using a set of neighborhood structures.

In this report, I start with an analysis of the algorithm and code. Then there are ten test presentations and analysis. The last part is my reflection and summary of this assignment.

2 Code Structure and Optimization

2.1 The structure of VNS

The Variable Neighborhood Search (VNS) algorithm is an effective metaheuristic for solving combinatorial optimization problems. In VNS, the search process is structured around multiple neighborhood structures, which allows the algorithm to explore the solution space more effectively.

The VNS algorithm comprises three neighborhoods: neighborhood1, neighborhood2, and neighborhood3. In neighborhood 1, two points are exchanged within the magic square matrix. Neighborhood 2 involves swapping two pairs of points with the same column coordinates,

while Neighborhood 3 consists of swapping two pairs of points with the same row coordinates. These neighborhood structures enable the VNS algorithm to search through a diverse range of solution configurations and avoid being trapped in local optima.

```
function VNS(matrix, max_time, kmax, result_matrix, best_fitness, magic_constant):
    initial best_matrix, best_fitness, current_matrix, current_fitness

    while True:
        k = 1
        current_fitness = calculate_fitness(current_matrix, magic_constant)
        while k <= kmax:
            initial new_matrix, delta

            if k == 1:
                neighborhood1(current_matrix, new_matrix, current_fitness, delta, magic_constant)
            elif k == 2:
                neighborhood2(current_matrix, new_matrix, current_fitness, delta, magic_constant)
            elif k == 3:
                neighborhood3(current_matrix, new_matrix, current_fitness, delta, magic_constant)
            else:
                break

            check_time_constraint()

            if delta <= 0:
                k += 1
                continue
            else:
                new_fitness = current_fitness - delta
                current_matrix = copy(new_matrix)
                current_fitness = new_fitness
                k = 1

                if new_fitness == 0:
                    result_matrix = copy(current_matrix)
                    return

            update_optimal_solution()
            shake(current_matrix, shake_time)
```

Figure 2: Pseudocode

The VNS algorithm proceeds by iteratively performing local searches within each neighborhood, followed by a shaking phase to diversify the search. The shaking phase consists of randomly exchanging points in the current solution to escape local optima. The search continues until a termination criterion is met, which can be based on time constraints or reaching a desired fitness value.

In summary, the VNS algorithm offers a robust and flexible approach to solving the Magic

Square Problem by systematically exploring various neighborhood structures and incorporating a shaking phase to avoid local optima. This combination of features makes it a powerful tool for tackling complex combinatorial optimization problems.

2.2 Fitness function

For the fitness calculation, I used 3 functions for different situations. The calculate_diagonal_ fitness function is used to calculate the diagonal fitness of the matrix, which assists the following two functions in the calculation. calculate_fitness function is used to calculate the total fitness of the matrix, including the sum of the fitness of rows, columns, and diagonals. calculate_delta function is used to calculate the total fitness difference of two matrices after the exchange of two points, because after the exchange of two points, only the fitness difference of rows and columns involving these two points will be changed, and this change adds to the diagonal fitness is the fitness difference after the exchange of two points, which can reduce the amount of calculation and optimize the performance of code.

2.3 Neighborhood optimization for Intensification mechanism

In this study, I enhance the search intensification process by implementing three distinct neighborhoods for optimization.

Neighborhood 1: In this neighborhood, the magic square randomly swaps two positions, providing a fundamental level of optimization through entirely random matrix adjustments.

Neighborhood 2: In this neighborhood, the magic square exchanges two pairs of points with the same column coordinates. This approach ensures that row fitness remains unaffected, while column fitness and diagonal fitness undergo adjustments.

Neighborhood 3: In this neighborhood, the magic square swaps two pairs of points with the same row coordinates, similar to Neighborhood 2, ensuring that column fitness remains unaffected.

If the objective function is reduced after adjusting the matrix using these three neighborhoods, the best optimal solution is updated. By iteratively using these three neighborhoods, the matrix's objective value is continuously decreased.

Using the VNS code provided above, I implement these three neighborhoods and demon-

strate their effectiveness in optimizing the magic square problem. The results reveal the utility of this intensification strategy in enhancing search performance and reducing the objective function value.

2.4 Diversification mechanism

In order to avoid falling into the local optimal solution during the search process, I added a diversification mechanism to explore more solution space during the search process. Because I chose to apply the Best Descent Strategy in the local search. In each neighborhood function, they traverse all possible exchange points and update the optimal solution when a better one is found (when the delta is greater than 0 and greater than the current optimal delta). Under this strategy, the best improvement scheme will be selected in each iteration, so it is very easy to fall into the local optimal solution. I randomly shuffle the current matrix by using the shake function every time it drops to the optimum. Moreover, when solving the magic square problem of higher order (order greater than 15), I added an extra 25 pairs of random two-point exchanges to increase the diversity of the solution, avoiding disrupting the insufficient. Before this mechanism is added, it is easy to get stuck in the optimal solution when solving the 20-order problem. After 100 seconds of running, it may occur that the objective function still does not drop to 0. But after adding this extra shuffling mechanism, 80% of the 20-order magic square can be optimized between 60 and 80 seconds.

3 Statistical results

3.1 Results of ten tests

In order to avoid the chance of optimizing the resulting test results, I will loop the test ten times to get ten test results reflecting the optimization performance of the algorithm.

3.2 Test based analysis

As can be seen from the above table, magic square problems below order 20 can be solved with high efficiency. For order 20 magic square problems, the fastest solution time is 61 seconds, the worst objective value optimization case is 2, 80% of the cases can be completed in 100

Case	1	2	3	4	5	6	7	8	9	10	Average
Order 5	0	0	0	0	0	0	0	0	0	0	0
Order 6	0	0	0	0	0	0	0	0	0	0	0
Order 10	0	0	0	0	0	0	0	0	0	0	0
Order 20	0	0	2	0	0	0	0	2	0	0	0.2
Order 30	6	12	17	3	46	24	8	52	2	28	19.8

Figure 3: Test results of object value

Case	1	2	3	4	5	6	7	8	9	10	Average (s)
Order 5	1	0	5	1	6	0	1	0	5	0	1.9
Order 6	17	6	5	15	24	7	5	2	13	4	9.8
Order 10	6	7	30	5	25	18	23	17	13	22	16.6
Order 20	66	70	99	81	78	86	72	99	61	88	80
Order 30	99	99	99	99	99	99	99	99	99	99	99

Figure 4: Test results of run time

seconds for order 20 problems. For the 30 order magic square problem, which has not been solved in 100 seconds, the worst objective value optimization case is 52, and the best objective function optimization case is 2.

4 Reflection

In the process of completing this program, I began to learn the knowledge about VNS in detail, and then set up several special cases according to the nature of the magic square problem, and wrote the first version of the code in C language. After updating the requirements, I tried to divide the magic square problem into two parts by reading the literature about VNS solving the magic square problem. The first part is to reduce the row and column fitness to 0 by VNS, and the second part is to reduce the diagonal fitness to 0 by a specific interchange. By the time I finished the second edition, it was pretty good at solving the magic square problem of

order 10. After exchanging ideas with my classmates, I compared the operation efficiency of C and Python, and I changed my mind and wrote the third version of the program, which is the final version.

5 Conclusion

By analyzing magic square problem, we learn the properties of neighborhood and local search of VNS algorithm, and determine the optimization direction of neighborhood and local search. The score of the test provided is stable above 23 points, which reflects the stability of the algorithm and the rationality of the neighborhood design. In addition, due to time constraints, I consider and optimize the time complexity and part of the calculation function. For future improvements, I think it is possible to increase the amount of data tested, so as to determine the number of shakes per shake and the special exchange mode in the face of higher-order problems, so as to make the algorithm more widely applicable.