GENETIC ALGORITHM WITH MODIFIED OPERATORS FOR AN INTEGRATED TRAVELING SALESMAN AND COVERAGE PATH PLANNING PROBLEM

Wen-Chieh Tung, Jing-Sin Liu
Institute of Information Science, Academia Sinica
No 128, Academia Road, Section 2
Nankang, Taipei 11529, Taiwan

ABSTRACT

Coverage path planning (CPP) is a fundamental task to many robotics applications such as cleaning, mine sweeping, lawn mowing, UAV mapping and surveillance. One approach for a known environment with obstacles is to decompose the environment into cells such that each cell can be covered individually. We can then decide the visiting order of cells to connect those intra-cell paths together. Finding the shortest inter-cell path that visits every cell and returns to the origin cell is similar to the traveling salesman problem (TSP), the additional variation to consider is that there are multiple intra-cell path choices for each cell, those choices will result in different entry and exit points of cells, and therefore affect the inter-cell path. This integrated traveling salesman and coverage path planning problem is called TSP-CPP. Recent approaches for TSP-CPP include adapting dynamic programming algorithm for TSP, or brute force on combinations of entry and exit points of every cell and solve each entry and exit points combination with TSP-solver. Both of them suffer from exponential complexity and are prohibitive for complex environments with large number of cells. In this paper, we propose a genetic algorithm approach with modified operators for TSP-CPP. Our approach can find the same solution as the optimal solution of DP in all experiments. When cell number is large, our approach is more than one thousand times faster than DP approach. The proposed GA approach is also a potential solution for environments with cell number beyond the limit of DP.

KEYWORDS

Coverage path planning, traveling salesman problem, TSP-CPP, genetic algorithm, dynamic programming, modified operators.

1. INTRODUCTION

Coverage path planning (CPP) is a fundamental task to many robotics applications such as cleaning, mine sweeping, lawn mowing, UAV mapping and surveillance. A lot of CPP problems have been studied for different application scenarios (Galceran, E. and Carreras, M., 2013, Khan, A. et al, 2017). A commonly encountered problem is to completely cover the environment with known static obstacles and return to the starting position. One efficient approach is to decompose the environment into cells such that each cell can be covered individually. We can then decide the visiting order of cells to connect those intra-cell paths together. Finding the shortest inter-cell path that visits every cell and returns to the origin cell is similar to the traveling salesman problem (TSP), the additional variation to consider is that there are multiple intra-cell path choices for each cell, those choices will result in different entry and exit points of cells, and therefore affect the inter-cell path. This integrated traveling salesman and coverage path planning problem is called TSP-CPP (Xie, J. et al, 2018). As TSP is already a NP-hard problem, TSP-CPP introduces great challenges with exponentially larger search space than TSP for choosing entry and exit points of each cell. Recent approaches for TSP-CPP include adapting dynamic programming for TSP, or brute force on combinations of entry and exit points of every cell and solve each entry and exit points combination with TSP-solver (Khsheibun, E. et al, 2018). Both of them suffer from exponential complexity and are prohibitive for complex environments with large number of cells.

Genetic algorithm has been developed as one effective approach to TSP (Potvin, J.Y., 1996, Scholz, J., 2019). A practical advantage of GA-based approaches is that they can find a decent solution within limited time and memory even when the data is large. Although genetic algorithm has already been considered for a similar problem in Jimenez, P.A. et al 2007, the paper only applied simple single-point crossover and swap mutation, and no performance evaluation was shown. In this paper, we propose a genetic algorithm approach for TSP-CPP. Our contributions in this paper are as follows : 1. Even though the optimality of the GA solutions are no longer guaranteed due to its heuristic nature, our GA approach with modified operators can find a solution identical to the optimal solution of DP in all experiments. When cell number is large, our approach is more than one thousand times faster than DP approach. It is also a potential solution for environments with cell number beyond the limit of DP. 2. We provide a complete flow and implementation code to transform our original CPP problem to the TSP-CPP problem, and the DP and proposed GA solutions for TSP-CPP are also available at https://github.com/WJTung/GA-TSPCPP. The paper will be structured as : In section 2, we describe how to decompose the environment into cells. In section 3, we display different choices for entry and exit points of each cell and how they affect the inter-cell path. In section 4, DP approach for TSP-CPP is briefly introduced, and our proposed GA approach is illustrated. In section 5, performances of GA and DP in six simulation environments are compared to show the advantages of GA. In section 6, we conclude our paper and list some possible directions for future improvements.

2. SPACE DECOMPOSITION AND COVERAGE PATH OF SINGLE CELL

In this paper, we assume that the workspace is a closed and bounded region W in R^2. W is cluttered with a finite number of static disjoint polygonal obstacles $\{P_i\}$. For simplicity, the vehicle V is modeled as a point with a constant sensing range, and it can translate in any direction so both a piecewise linear path and a turn of any specified angle are executable. The vehicle's task is to cover the environment W without collisions with the obstacles $\{P_i\}$. We apply the boustrophedon cellular decomposition to break the vehicle's free space down into cells, and assume for simplicity that the vehicle will cover each cell with back-and-forth boustrophedic motions (Choset, H. and Pignon, P., 1998). To implement the boustrophedon cellular decomposition, we use a vertical slice to sweep through the environment, and open and close cells according to the connectivity change of the slice. The distance between the boustrophedon path and the boundary of cell must be smaller than the sensing radius, and the distance between two adjacent vertical lines in the boustrophedon path must be smaller than twice of the sensing radius. Figure 1 demonstrates the result of the boustrophedon cellular decomposition for one example environment and how each cell can be covered with back-and-forth boustrophedon path.

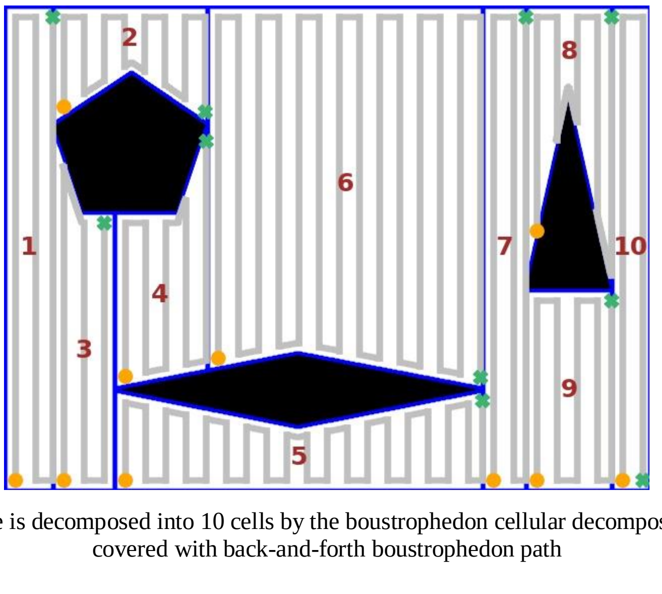


Figure 1. The free space is decomposed into 10 cells by the boustrophedon cellular decomposition. Each cell can be covered with back-and-forth boustrophedon path

3. DETERMINE THE ENTRY AND EXIT POINTS FOR EACH CELL AND THE VISITING ORDER OF CELLS

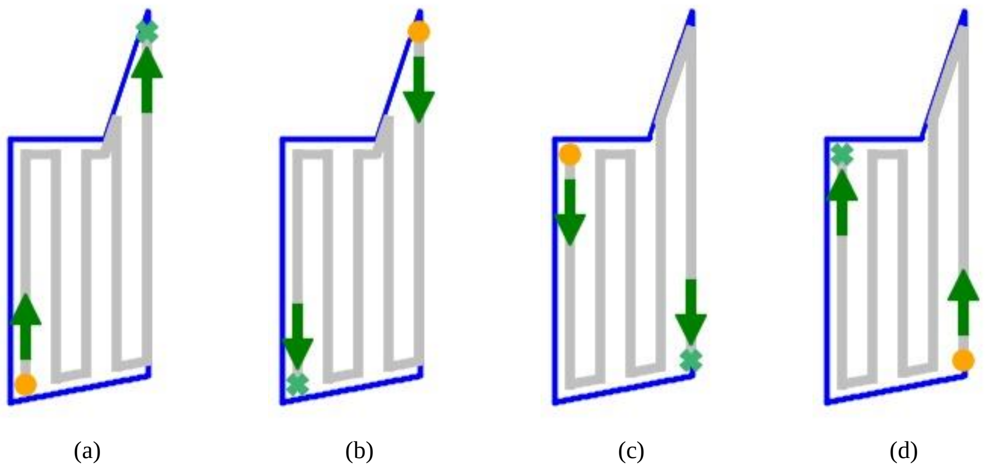


Figure 2. Four possible boustrophedon path (entry and exit points) choices for cell 4 in Figure 1. The entry point is marked by the orange circle, and the exit point is marked by the sea green cross. The direction of the boustrophedon path is indicated by the green arrow

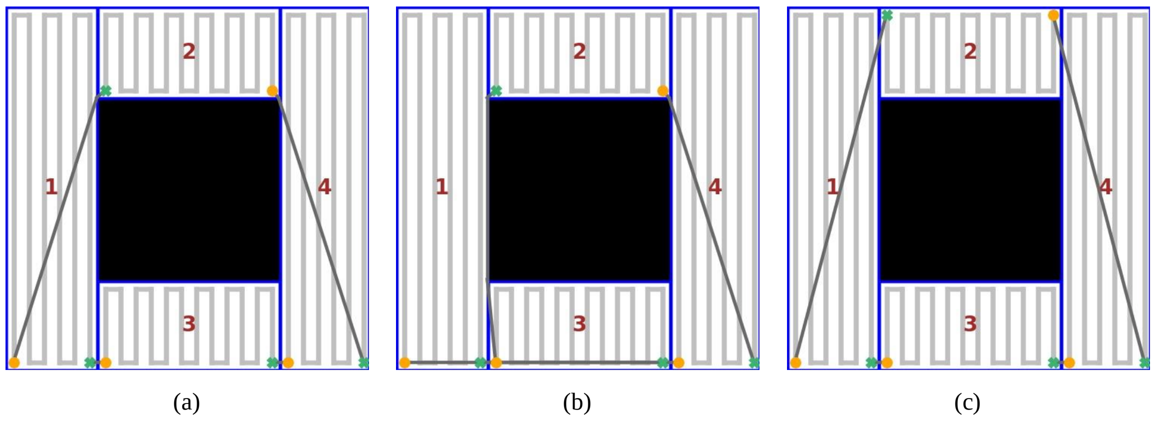


Figure 3. We can see the influence of the visiting order of cells and entry and exit points of each cell on inter-cell path length through (a), (b), (c). (a) is the optimal visiting order (1, 3, 4, 2) and entry and exit points solution for this environment with shortest inter-cell path length 802.78. For (b), the entry and exit points of each cell is the same as (a), but the visiting order is 1, 4, 2, 3, which results in a total inter-cell path length of 1347.40 (67.8% larger than (a)). For (c), the visiting order of cells is the same as (a), but the intra-cell path of 2 starts from top right instead of bottom right, which results in a total inter-path length of 991.36 (23.5% larger than (a))

Our next step is to determine the visiting order to connect intra-cell paths of every individually covered cell together. We assume that we can only connect the end point of one intra-cell path to the start point of another intra-cell path. Therefore, the start point and end point of the boustrophedon path in each cell can be viewed as the entry point and exit point of the cell (Khsheibun, E. et al, 2018). If the entry and exit points of each cell are fixed, this is actually the travelling salesman problem (TSP), which is a classical NP-hard problem to find the shortest possible route that visits each city and returns to the origin city. To find the shortest path between two points for the distance between cells, when the forbidden regions occupied by the non-overlapping obstacles are polygonal (bounded by straight line segments), we can construct the visibility graph of the vertices of obstacles and the start and goal points as additional vertices. We then mark the edges of the visibility graph with their Euclidean lengths, and use Dijkstra's algorithm to find the shortest paths from the start point (Alt, H. and Welzl, E., 1988). (In our simulation, we consider the vehicle as a point. If the vehicle’s radius is considered, the configuration space has to be constructed). We will use distance(i, j) to represent the shortest path length between the exit point of cell i and the entry point of cell j (inter-cell path from i to j) obtained by the visibility graph. Since distance(i, j) might be different from distance(j, i), this is an asymmetric TSP.

However, for the CPP problem in each cell, the boustrophedon path can start from either top left, top right, bottom left, bottom right, which gives us four choices for each cell (see Figure 2 for illustration). In addition to the change in intra-cell path length (difference between intra-cell path length of (a) = (b) and (c) = (d)), the determination of entry and exit points for each cell will also affect the total inter-cell path length greatly.

Figure 3 is an example that demonstrates the entry and exit points for each cell and the visiting order of cells should be considered simultaneously for TSP-CPP. In Figure 3, (a) is the optimal solution for the simple 4-cell environment. Either a different visiting order like (b) or a different entry and exit points choice for any cell like (c) can cause an increase in inter-cell path length. This integrated TSP and CPP problem to determine both the visiting order and the entry and exit points choice of each cell is called TSP-CPP (Xie, J. et al, 2018). In the following section, we will describe two algorithms for solving TSP-CPP to find the optimal entry and exit points for each cell and the visiting order of cells with the minimum total coverage path length (under our decomposition and possible choices of intra-cell paths).

4. SOLUTIONS TO THE INTEGRATED TSP-CPP

In the following, we will denote the cell number as N. Furthermore, to make the explanation more concise, we will call a specific cell with its entry and exit points (intra-cell path) choice a *cell-path combination*. Each cell has four possible choices of entry and exit points, which results in four possible cell-path combinations, and there are in total 4N cell-path combinations for all cells.

4.1 Dynamic Programming Approach

Dynamic programming approach for TSP-CPP and proof of its optimality is described in Xie, J. et al, 2018. The main idea of the approach is similar to DP for TSP. We can assign one cell as the starting cell and consider four cell-path combinations of it as starting cell-path to break the route cycle. We will then recursively update shortest path length for a specific visited state (set of cells that have already been visited) and last cell-path, this can be done by checking all possible previous cell-path with corresponding previous visited state. The time complexity of DP approach is $O(2^N \times (4 \times N)^2)$, and the space complexity is $O(2^N \times 4 \times N)$, which is still $O(2^N \times N^2)$ and $O(2^N \times N)$, the same as DP for TSP. In our paper, dynamic programming is used as a baseline of optimality, computation time, and memory usage.

4.2 Genetic Algorithm Approach

Although the dynamic programming approach can find the optimal solution for TSP-CPP, due to the exponential factor in both time and space complexity, it is only suggested to be used for up to N = 17 targets (Bellman, R.E., 1961, Xie, J. et al, 2018). However, in real life application, it is highly possible that there will be more than 20 cells for an environment with many obstacles. For TSP, genetic algorithm is an efficient approach to get a decent solution within limited time and memory. We then came up with the idea to modify GA operators for TSP. To apply on TSP-CPP, the modified operators must consider both the entry and exit points of each cell and the visiting order of cells at the same time.

4.2.1 Encoding and Decoding

In GA for TSP, genes of one individual is usually one permutation of numbers $1 \sim N$, which can be directly viewed as the visiting order. In TSP-CPP, we can instead assign each gene to be one of the 4N cell-path combinations. The cell id and corresponding entry and exit points choice can be decoded by dividing 4 and modulo 4. Genes of one individual cannot have repeated cell id.

4.2.2 Crossover Operator

There are many crossover operators for GA in TSP. One simple but powerful crossover operator is the heuristic crossover operator used in Vahdati, G. et al, 2009. This heuristic crossover operator performs well even when the city number is more than 100 (Ismkhan, H. and Zamanifar, K., 2015). Our modified heuristic crossover operator to consider entry and exit points of each cell can be described by the following steps :

1. Randomly pick one cell as the starting cell, then randomly pick the cell-path combination of starting cell from one parent. Add this to the offspring as the starting cell-path.
2. Find 4 candidates for the next cell : cells of the first left neighbor and the first right neighbor of starting cell in two parents which are still not visited in the offspring. (This can be implemented by keeping four pointers of current position as described in Vahdati, G. et al, 2009)
3. Consider 4 possible choices of entry and exit points for each candidate cell, so there are 16 candidates for next cell-path. For the heuristic to determine the next cell-path of the offspring, in addition to the inter-cell path length between current cell-path of offspring and candidates of next cell-path, we also consider the change of intra-cell path length caused by different entry and exit points choices of a cell. We define cost of one cell-path to be its intra-cell path length minus the minimum intra-cell path length among four possible entry and exit points choices of its cell, which represents the extra cost for not choosing the entry and exit points choice of a cell with the shortest intra-cell path. We will add the cell-path with minimum distance(current cell-path of offspring, cell-path) + cost(cell-path) among 16 candidates to offspring as the next cell-path.
4. Repeat step 2 and 3 until all cells are visited in offspring

4.2.3 Mutation Operator

```
Algorithm : Given the visiting order of cells, find the optimal entry and exit
points for each cell with the shortest coverage path
Input : Offspring after crossover and swap mutation
Output : Individual that has the same visiting order as the input with the
shortest coverage path

Initialize cell array of size N
for each i = 1 ~ N do
    cell[i] = the cell of individual[i]
Initialize optimal solution array of size N
for each starting cell-path of cell[1] do
    Initialize from array of size 4N
    Initialize shortest path length array of size 4N with infinity
    shortest path length[starting cell-path] = intra-cell path length of
    starting cell-path
    for each i = 2 ~ N do
        for each current cell-path of cell[i] do
            for each previous cell-path of cell[i - 1] do
                if shortest path length[previous cell-path] + distance(previous
                cell-path, current cell-path) + intra-cell path length of current
                cell-path is smaller than shortest path length[current cell-path]
                then
                    update shortest path length and from of current cell-path
    for each ending cell-path of cell[N] do
        add distance(ending cell-path, starting cell-path) to shortest path
        length[ending cell-path]
    find the best ending cell-path with the smallest shortest path length[ending
    cell-path]
    if shortest path length[best ending cell-path] is smaller than the coverage
    path length of optimal solution then
        find the whole solution for current starting cell-path by backtracking
        from best ending cell-path with from array
        update optimal solution
return optimal solution
```

In GA for TSP, the swap mutation operator is a commonly used mutation operator. In our GA for TSP-CPP, we also randomly swap the visiting order of two cell-path combinations under the assigned probability. In addition to the swap mutation operator, 2-opt local search is an effective mutation operator that improves an individual solution through local search. Similarly, if we assume the visiting order of cells to be fixed, we can find the optimal entry and exit points for each cell with the shortest coverage path through dynamic programming in linear time (see the Algorithm above). The main idea is that if the visiting order of cells is fixed and we do not have to go back to the starting cell, the only factor that will influence the following path is our choice of exit point for current cell. We can permutate four possible starting cell-path to break the route cycle, and use an array to record the shortest path length (intra-cell path + inter-cell path) to each cell-path combination if we visit cells in the given order. As the shortest path length to current cell-path is only related to the previous cell-path and its shortest path length, we can update the shortest path length array by checking four possible previous cell-path combinations. An additional array is used to record from which previous cell-path can we get the shortest path length to current cell-path for backtracking.

Although the result is a local optimal solution as we consider only the entry and exit points of each cell and the visiting order of cells cannot be modified, this optimal entry and exit points mutation will increase our probability of getting a global optimal solution through evolution.

4.2.4 Overall Flow of GA

The overall flow of our GA approach can be described by the following steps :
1. Initialize population of assigned population size with random permutation of cells by Knuth shuffle, and randomly choose one entry and exit points from four possible choices for each cell.
2. Calculate coverage path length of each individual in the initial population, assign current best solution to be the best solution among them.
3. Randomly choose two parents from the population with roulette wheel selection, and generate offspring with our modified heuristic crossover operator. Repeat this process until there are as many offspring as the original population.
4. For each offspring generated by crossover operator, randomly swap the visiting order of two cells according to the assigned swap mutation probability, and then find the optimal entry and exit points for each cell with the same visiting order of cells (Algorithm 1).
5. Replace the population with the newly generated offspring. Calculate coverage path length of each individual in the new population, update the current best solution if any better solution is found.
6. Repeat step 3 to 5 until number of evolved generations is equal to the assigned generation number. Output the best solution obtained in the evolution process.

4.2.5 Time and Space Complexity

Time complexity :
- Calculate coverage path length : $\mathrm{O}(\mathrm{N})$ for each individual
- Modified heuristic crossover : $\mathrm{O}(16 \times \mathrm{N})$ for each offspring (16 candidates for each position)
- Optimal entry and exit points mutation : $\mathrm{O}(64 \times \mathrm{N})$ for each offspring (Permutate 4 possible starting cell-path combinations and consider 4 possible previous cell-path combinations each time for updating shortest path length of 4N cell-path combinations)
- Overall : $\mathrm{O}(\mathrm{N} \times \text{population size} \times \text{generation number})$

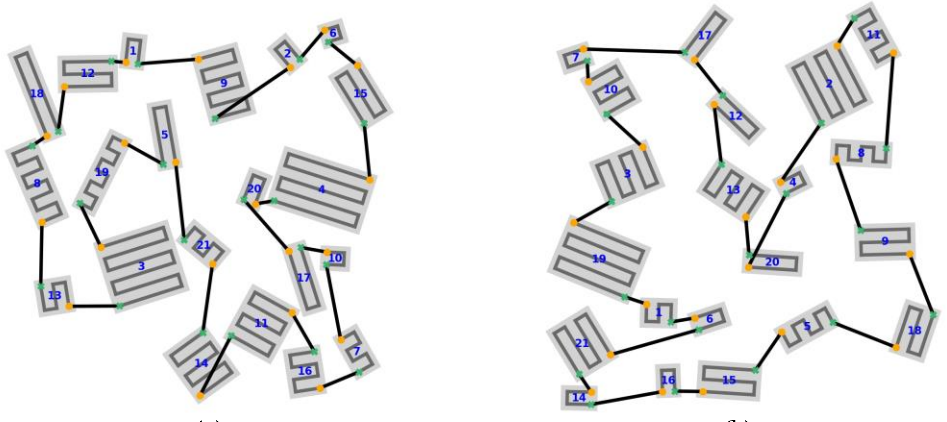
Space complexity : $\mathrm{O}(\text{population size} \times \mathrm{N})$

5. SIMULATION RESULTS

To examine solution quality of our GA approach, we first generate 100 random environments with multiple spatially distributed rectangular regions (cells) to cover and compare the coverage path solution by GA and DP (Xie, J. et al, 2018). Without obstacles, we do not have to consider the decomposition of environment, and visibility graph is not required for finding the shortest path between two points. For a rectangular region, the intra-cell path length is the same for all four choices of a cell. Therefore, the intra-cell path part is common for GA and DP regardless of the solution, we will use only inter-cell path length to calculate the relative error. With population size = 4096, generation number = 16, crossover probability = 0.9, and swap mutation probability = 0.02, our GA approach can find the optimal or near-optimal (within 3%) solution in all generated environments as seen in Table 1. Figure 4 is two example test cases with 21 cells and their corresponding optimal solutions. In Figure 5, we display the test case with maximum relative error 2.778% (25 cells).

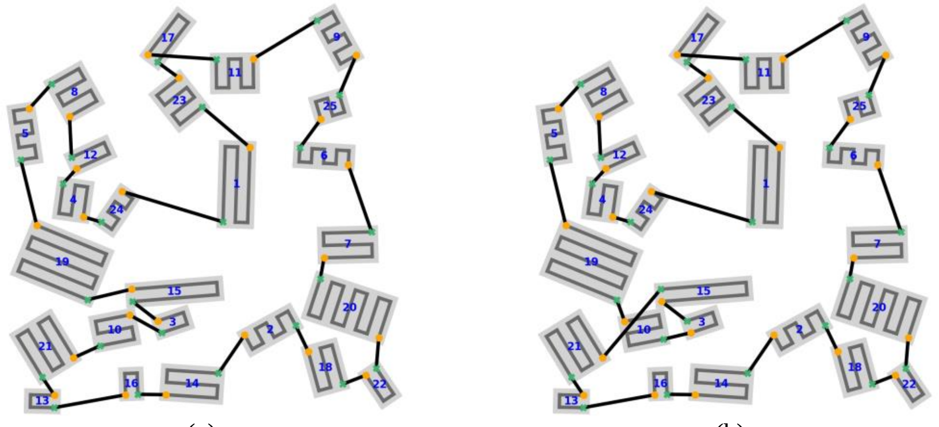
Table 1. Solution quality of GA for environments with multiple spatially distributed rectangular regions to cover. For each cell number, we randomly generate 10 corresponding test cases of same environment size. Number of successes is defined as the number of times that GA successfully find the same optimal solution as DP among those 10 cases. The relative error is calculated between the inter-cell path length of DP and GA

Cell Number	Environment Size	Number of Successes	Relative Error (inter-cell path)	
			Maximum	Average
16	1280×720	10	0.0%	0.0%
17	1080×1080	10	0.0%	0.0%
18	1280×720	10	0.0%	0.0%
19	1080×1080	10	0.0%	0.0%
20	1280×720	9	0.095%	0.01%
21	1080×1080	9	1.063%	0.106%
22	1280×720	8	0.322%	0.057%
23	1080×1080	8	1.217%	0.227%
24	1080×1080	7	1.688%	0.288%
25	1080×1080	4	2.778%	0.750%



(a) (b)

Figure 4. Two example test cases with 21 cells and their corresponding optimal solutions



(a) (b)

Figure 5. The test case with maximum relative error 2.778% (25 cells). (a) is the optimal solution generated by DP, and (b) is the solution generated by our GA approach.

We then design six experiment environments with polygonal obstacles of various shapes and sizes to analyze the quality of solutions and time and memory efficiency of our GA approach in our main problem introduced in previous sections. Those environments have different number of cells from 10 to 26 after decomposition. We use the same GA parameters as previously stated, and assume sensor radius to be 10. In all experiments, GA is able to find the same solution as the optimal solution of DP. Table 1 is the performance comparison of GA and DP. Figure 6, 7, 8, 9, 10, 11 display the optimal solutions of experiments generated by DP and GA. For Experiment 2, 3, 4, 5, 6, as the cell number is large, we only show the inter-cell path and the entry and exit points for each cell, the intra-cell path is not plotted in the figure.

As seen in Table 2, when the population size and generation number are fixed, the computation time of GA is almost linear to the cell number, which makes GA much faster than the DP approach with exponential time complexity when cell number is large. Due to the exponential time and space complexity of DP, experiment environments are limited to cell number no more than 26. Nevertheless, with the complexity linear to cell number, population size, generation number, and the quality of solutions demonstrated, we consider GA a potential approach to find the optimal or near-optimal solution for environments with a large number of decomposed cells that prohibit the application of DP.

Table 2. Performance comparison of GA and DP. GA is able to find the same solution as the optimal solution of DP in all experiments. C++ programs of DP and GA are executed on the same computer (HP Z440 Workstation, CPU : Intel Xeon E5-1630 v3, 64G RAM, OS : Ubuntu 18.04) five times to take the average computation time, and memory usage is measured by maximum resident set size

Experiment #	Environment Size	Cell Number	Computation Time (s)			Memory (Maximum RSS, MB)		
			DP	GA	DP / GA	DP	GA	DP / GA
1 (Fig 6)	640 × 480	10	0.014	1.508	0.009	3.668	4.052	0.905
2 (Fig 7)	1200 × 1200	14	0.289	1.845	0.157	6.836	4.264	1.603
3 (Fig 8)	1080 × 1080	18	7.538	2.210	3.411	78.92	4.284	18.42
4 (Fig 9)	1280 × 720	20	36.53	2.361	15.47	339.1	4.340	78.13
5 (Fig 10)	1080 × 1080	23	383.2	2.629	145.8	3083	4.716	653.8
6 (Fig 11)	1080 × 1080	26	3965	2.900	1367	27790	4.740	5863

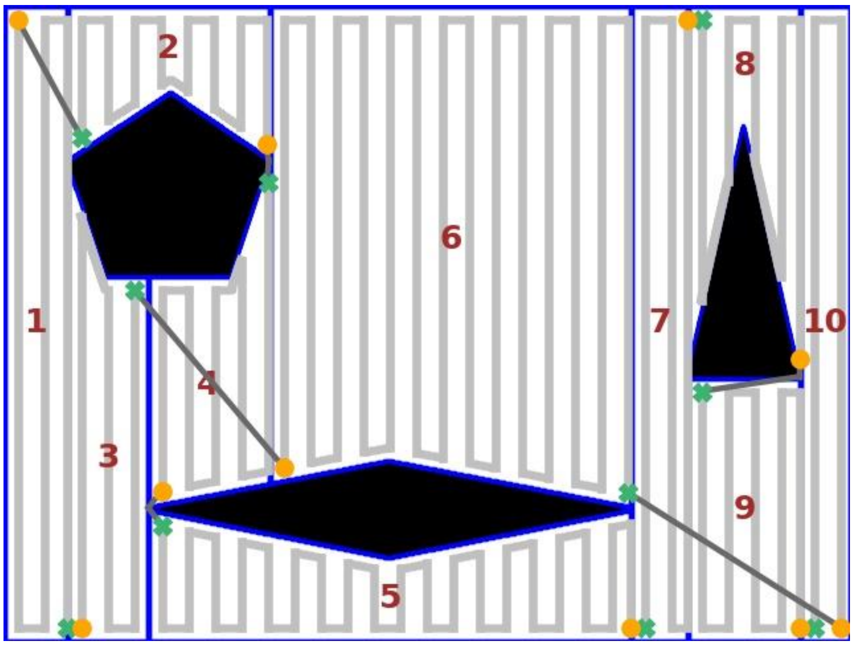


Figure 6. Optimal solution for Experiment 1 with total coverage path length 15582. The light gray path is the intra-cell path (boustrophedon path), and the dark gray path is the inter-cell path. The visiting order of cells is 1, 3, 6, 10, 9, 8, 7, 5, 4, 2, 1

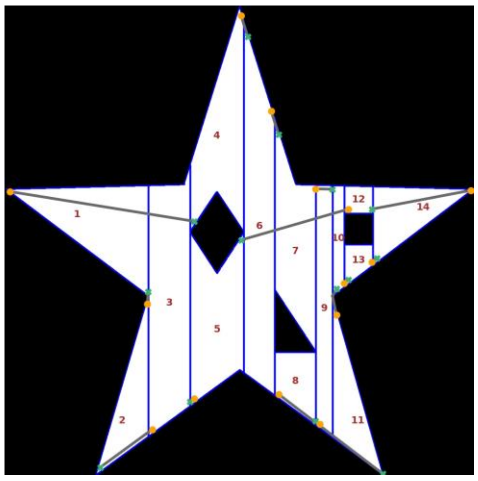


Figure 7. Optimal solution for Experiment 2 with total coverage path length 26977.5. The visiting order is 1, 2, 3, 5, 12, 14, 13, 10, 11, 8, 9, 7, 6, 4

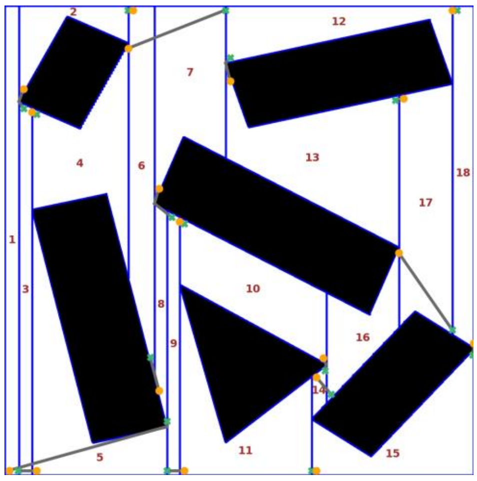


Figure 8. Optimal solution for Experiment 3 with total coverage path length 43682.8. The visiting order is 1, 5, 11, 15, 18, 12, 13, 17, 16, 14, 10, 9, 7, 4, 3, 2, 6, 8, 1

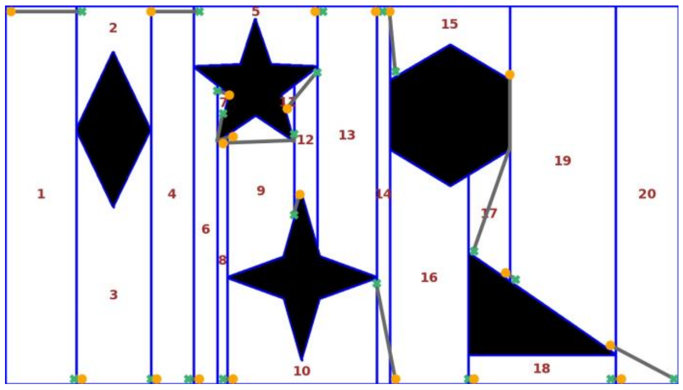
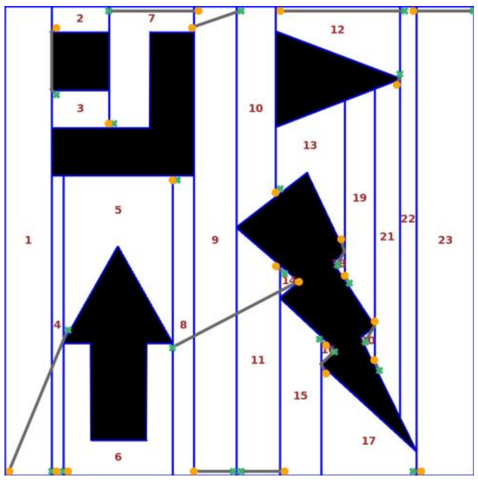


Figure 9. Optimal solution for Experiment 4 with total coverage path length 43640.1. The visiting order is 1, 3, 4, 6, 7, 9, 12, 11, 8, 10, 16, 18, 20, 19, 17, 15, 14, 13, 5, 2, 1



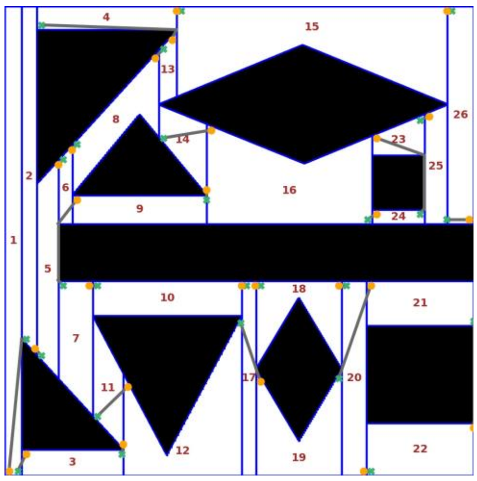


Figure 10. Optimal solution for Experiment 5 with total coverage path length 52065.4. The visiting order is 1, 4, 6, 14, 11, 15, 16, 17, 23, 22, 12, 21, 20, 19, 18, 13, 10, 7, 3, 2, 9, 8, 5, 1

Figure 11. Optimal solution for Experiment 6 with total coverage path length 37717.7. The visiting order is 1, 3, 11, 12, 19, 21, 22, 20, 18, 17, 10, 7, 9, 14, 16, 24, 23, 25, 26, 15, 4, 13, 8, 6, 5, 2, 1

6. CONCLUSION

In this paper, we propose a genetic algorithm approach with modified operators for the TSP-CPP problem. Our GA approach can successfully find the same solution as the optimal solution of DP in all experiments. Furthermore, it is not limited to the exponential time and space complexity, which makes it more than one thousand times faster than the DP approach when cell number is large and a potential solution for more complex environments that DP cannot handle. In this paper, we do not consider the more realistic vehicle model such as size and shape of the vehicle and its kinodynamic constraints. For non-polygonal or arbitrary shape obstacles, convex hull or bounding volume approach can also be implemented. Some possible further improvements include testing more intra-cell path patterns, considering turn cost, choosing different decomposition like MSA decomposition (Huang, W.H., 2001), applying multi-objective genetic algorithm (MOGA). Taking these details and extensions into account will complicate the planning, and can be implemented in the future.

REFERENCES

Alt, H. and Welzl, E., 1988. Visibility graphs and obstacle-avoiding shortest paths. *Zeitschrift für Operations-Research,* 32(3-4), pp.145-164.

Bellman, R.E., 1961. Dynamic programming treatment of the traveling salesman problem.

Choset, H. and Pignon, P., 1998. Coverage path planning: The boustrophedon cellular decomposition. In *Field and service robotics* (pp. 203-209). Springer, London.

Galceran, E. and Carreras, M., 2013. A survey on coverage path planning for robotics. *Robotics and Autonomous systems, 61*(12), pp.1258-1276.

Huang, W.H., 2001. Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)* (Vol. 1, pp. 27-32). IEEE.

Ismkhan, H. and Zamanifar, K., 2015. Study of some recent crossovers effects on speed and accuracy of genetic algorithm, using symmetric travelling salesman problem. *arXiv preprint arXiv:1504.02590.*

Jimenez, P.A. et al, 2007, September. Optimal area covering using genetic algorithms. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics* (pp. 1-5). IEEE.

Khan, A. et al, 2017. On Complete Coverage Path Planning Algorithms for Non-holonomic Mobile Robots: Survey and Challenges. *J. Inf. Sci. Eng., 33*(1), pp.101-121.

Khsheibun, E. et al, 2018, June. Efficient Coverage of Unstructured Environments. In *International Conference on Intelligent Autonomous Systems*(pp. 111-126). Springer, Cham.

Potvin, J.Y., 1996. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research, 63*(3), pp.337-370.

Scholz, J., 2019. Genetic Algorithms and the Traveling Salesman Problem a historical Review. arXiv preprint arXiv:1901.05737.

Vahdati, G. et al, 2009, December. A new approach to solve traveling salesman problem using genetic algorithm based on heuristic crossover and mutation operator. In *2009 International Conference of Soft Computing and Pattern Recognition* (pp. 112-116). IEEE.

Xie, J. et al, 2018. An Integrated Traveling Salesman and Coverage Path Planning Problem for Unmanned Aircraft Systems. *IEEE control systems letters, 3*(1), pp.67-72.