Trade Study - Search Algorithm

Tiger Rescue

## 

Table of Contents

[**Background**](#_gdmkzhz4pzq1) **3**

[**Strategies**](#_d856kx7bdm5l) **3**

[**Scenario Areas**](#_pf2zx2fsnr3t) **4**

[**Metrics**](#_jc23u85t8mlo) **4**

[**Procedure**](#_juno75ywelev) **5**

[**Results Analysis**](#_qkfeqq80td6y) **5**

[Path Planning](#_k3qy6qqor5rs) 5

[Partitioning](#_7gdxcyp7s72n) 6

[**Conclusion**](#_6yd73bbklj98) **6**

[**References**](#_hduiiiu8d4i) **7**

## 

## Background

The system must be able to distribute a search area among the drone swarm to quickly and efficiently find a target. We devised a coverage strategy that breaks the given area polygon down into grid cell points. These points need to be fairly distributed amongst the swarm, and paths through the points should have minimal cost. To accomplish this, we needed to investigate both distribution and path planning algorithms.

## Strategies

For path planning, two primary algorithm classes were examined: [TSP](https://en.wikipedia.org/wiki/Travelling_salesman_problem) (Traveling Salesman Problem) and mTSP (Multiple Traveling Salesman Problem, also referred to as [Vehicle Routing Problem](https://en.wikipedia.org/wiki/Vehicle_routing_problem)). Unlike the TSP algorithms, mTSP/VRP algorithms have multi-agent planning built-in, so they are complete strategies by themselves. For the TSP algorithms, every TSP-partitioning combination will become a strategy.

To introduce turn costs to these algorithms, the cost function from [1] was used.

**mTSP/VRP Algorithms**

* [Google OR-Tools](https://developers.google.com/optimization) - 60s Guided Local Search, Pure Distance (OR-PD)
  + Uses Google OR-Tools VRP solver with a cost function only accounting for euclidean distance between points. The solver attempts to optimize the solution until the 60s time limit is reached. This limit was subjectively decided on based on poor or incomplete solutions observed with smaller time limit values.
* [Google OR-Tools](https://developers.google.com/optimization) - 60s Guided Local Search, Turn Costs (OR-TC)
  + Uses Google OR-Tools VRP solver with a cost function that accounts for turn penalties with 45° of turn precision. This strategy was built by creating a new ‘node’ for each entry angle at all points, and creating a disjunction of 1 cardinality for them. Like the other OR-Tools solution, a 60s time limit was used based on output observations.
* [Nearest Neighbor](https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm) With Turn Costs (NN-TC)
  + Typical iterative nearest neighbor implementation, except distances are based on the turn cost function. At each iteration, the nearest point is selected for *each* drone without duplication.
* Spiral
  + This strategy creates a spiral path through the full area by continuously going forward, and making the cheapest turns when a boundary or already visited point is reached. The spiral is then evenly sliced for distribution among the drones.

**TSP Algorithms**

* [Lin-Kernighan Heuristic](https://en.wikipedia.org/wiki/Lin%E2%80%93Kernighan_heuristic) (LKH)
  + Traditional Lin-Kernighan implementation with a cost function only accounting for euclidean distances.
* [Lin-Kernighan Heuristic](https://en.wikipedia.org/wiki/Lin%E2%80%93Kernighan_heuristic) With Turn Costs (LKH-TC)

**Partitioning Algorithms**

* UB-ANC Load Balancing (UB-ANC LB) [1]
  + [MILP](https://en.wikipedia.org/wiki/Integer_programming) optimization problem solved with [Glop](https://developers.google.com/optimization/lp/glop). The cost function of each drone is the total distance from the drone’s start point to each assigned point (not a path distance). This generates clearly-defined continuous areas for the drones.
* Alternating-3
  + The grid is iterated in row-col order. Each set of three points is assigned to one drone, then the next, and so on.
* Alternating-5
  + The grid is iterated in row-col order. Each set of five points is assigned to one drone, then the next, and so on.
* Random Assignment
  + Each point is randomly assigned to a drone.

## Scenario Areas

To test strategy effectiveness, the following scenario areas are used:

1. 5x5 square
2. 10x10 square
3. 25x25 square
4. 5x10 rectangle
5. 10x25 rectangle
6. 5x10 triangle
7. 10x25 triangle

In each scenario, the drone(s) will be ‘placed’ south of the area.

## Metrics

The following metrics will be collected and averaged over all test trials:

* Average path cost per drone
* Total path cost
* Partitioned path costs standard deviation (σ)
* Average path distance per drone
* Total path distance
* Partitioned path distances standard deviation (σ)
* Computation time
* Percentage of cells with duplicate coverage

## Procedure

For each scenario-strategy combination, 25 simulation trials will be conducted with 1-10 drones to collect our metrics of interest. Each strategy has been assessed to appropriately scale drone coverage with unequal capabilities (e.g. battery level), so our tests assume equal capabilities to simplify analysis. The time-performance of all the partitioning algorithms was deemed negligible for all the area sizes so it is not included in the computation time metric.

## Results Analysis

Unfortunately the collected metrics data is too vast to include in this document. The results for each algorithm are summarized below.

### Path Planning

**OR-PD**

This algorithm was found to perform well with respect to total distance and coverage duplication. Since turns are not accounted for, the total cost in most scenarios was comparably high since there was a significant amount of zig-zagging. A notable deficiency of this algorithm is that it sometimes generates solutions with severely unbalanced path distributions. There were some cases where one drone was assigned only two points. This can be somewhat fixed by tweaking the algorithm parameters, but we were unable to devise input values for these parameters that provided consistently fair distributions.

**OR-TC**

This algorithm was found to perform best in small areas, and very poorly in the larger areas. This is attributable to the time limit not being sufficient to reach an approximately optimal solution. Strangely, increasing this time limit to up to 10 minutes for the problematic areas did not resolve the issue. This difference from OR-PD likely occurs because of the added complexity of having 8 disjunctive nodes (45° angle precision) for every point.

**NN-TC**

This algorithm was found to perform best in small areas. Interestingly, generated paths consisted of multiple spirals. These spirals often ultimately ‘trapped’ the drone into an area, where it would then have to move far away to reach its next point. Coupling this with the fact that end points were often in the middle of an area, there was significant duplicate coverage with this algorithm for the larger areas.

**Spiral**

This algorithm was found to perform somewhat well in the rectangular/square scenarios, and very poorly in the triangle scenarios given the sharp turn angles at corners. While path costs *within* the area were low, the drones had added costs of reaching their start points, and returning from their end points, which were both in the middle of the area (except with the first drone). The placements of these start and end points incurred duplicate coverage.

**LKH**

This algorithm was found to perform well distance-wise in all of the scenarios, with minimal coverage duplication. One aspect of this success is the path’s construction taking into account reaching the end point. However, this algorithm did not perform well with regard to the path cost function, as it was found to frequently zig-zag between points. A major advantage of this algorithm is its computation time, being under 1s per drone in all scenarios.

**LKH-TC**

This algorithm was found to overall perform well in all of the scenarios, sharing the advantages of LKH. Compared to LKH, the modified cost function did not significantly affect the average path distance of each drone or computation time. For both LKH and LKH-TC, it was found that running many instances (~50) of the algorithm could often improve path costs by up to ~10%. This is expected as the algorithm only finds a local minima from an initial random state in each instance.

### Partitioning

**UB-ANC Load Balancing**

This algorithm was found to generate clearly-defined, continuous areas for each drone. Duplicate coverage was typically very low, and even zero in some cases. Since each drone only had one continuous area to cover, path costs were much lower.

**Alternating-\***

Both Alternating algorithms (Alternating-3 and Alternating-5) were found to generate ‘islands’ of points in the larger areas, where each drone would have multiple islands. This resulted in high path costs and some duplicate coverage as the drones had to move between the islands.

**Random Assignment**

Predictably, this algorithm generated large point spreads (e.g. opposite corners) in all scenarios that resulted in both high path costs and significant amounts of duplicate coverage.

## Conclusion

Our assessment results show that the LKH-TC TSP algorithm combined with the UB-ANC LB distribution algorithm performs (in general) the best for our metrics of interest. This strategy scales well, consistently providing reasonably optimal paths and distributed areas within a short computation time for all of the tested scenarios. We identified that running many iterations of this algorithm can improve its output even further, and thus will include that finding in our final implementation.

## References

[1] Modares, J., Ghanei, F., Mastronarde, N., & Dantu, K. (2017). UB-ANC planner: Energy efficient coverage path planning with multiple drones. 2017 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/icra.2017.7989732