Routing with Genetic Search using graph traversal derived operators

Ibrahim Ahmed

January 10, 2024

1 Introduction

Path-finding applications are ubiquitous in many engineering domains [4, 6, 10]. Given a start and a goal in a maze, algorithms exist that can find an optimal path in polynomial time. However, when the route is constrained by additional stopping points, the task of finding an optimal ordering of stops, in the worst case, is a computationally intractable problem. This is canonically known as the Traveling Salesman Problem (TSP) [3].

The naive approach so solving TSP is through simulation. The costs of all permutations of stopping points is calculated, and the ordering with the minimum cost is chosen as optimal. This scales with a complexity of $\mathcal{O}(n!)$, where n is the number of stopping points.

In this work, a heuristic-based search method is used to find a locally optimal solution to TSP. The associated code is available at `https://github.com/hazrmard/genetic-path-planning`.

2 Genetic Algorithms

A brief description of Genetic Algorithms is provided in this section. Genetic Algorithms borrow inspiration from nature to iteratively search for a solution to an optimization problem. The genetic search process can be distilled into a series of sequential steps, each contributing to the exploration and exploitation of potential solutions:

Initialization The optimization process begins with the creation of an initial population of potential solutions, often represented as chromosomes or individuals. These solutions are randomly generated within the search space, forming the starting point for the evolutionary algorithm.

Selection The selection phase mimics the natural process of survival of the fittest. Individuals from the population are probabilistically chosen based on their fitness, with higher fitness conferring a greater chance of being selected.

This step emphasizes the retention of superior solutions for subsequent generations.

Crossover Inspired by genetic recombination, crossover involves combining genetic material from two parent solutions to produce offspring. This step facilitates the exploration of new solution spaces by blending traits from successful individuals, potentially yielding improved solutions.

Mutation Mutation introduces random changes in selected individuals, simulating genetic variation. This stochastic process injects diversity into the population, preventing premature convergence to sub-optimal solutions and facilitating the discovery of novel, potentially superior candidates.

Replacement Newly generated offspring, possibly improved through crossover and mutation, replace less fit individuals in the current population. This step ensures the continuous evolution of the population towards better solutions over successive generations.

3 Existing Work

Before discussing prior work with path planning, mathematical notation is reviewed. The routing problem can be represented as a graph traversal task. A graph, G, is made up of vertices $v \in V$, connected by edges $e \in E$. The edges are directed, indicating the direction of allowed travel between two vertices. The cost of traversal is the summation of traversed edge weights. In this case, weights may represent the distance between vertices, the congestion in the path, or miscellaneous energy expenditures for that segment in the route.

3.1 Path planning algorithms

Readers are directed to established literature in path planning algorithms using graph theory. Dijkstra Algorithm [14] works by greedily solving sub-problems of minimum cost paths in a graph to find the shortest path from the start to the goal vertex. A* algorithm [2] is an informed variant of the former, where candidate vertices in a graph are ordered by a cost heuristic. The D* Algorithm is an incremental heuristic search algorithm that can solve the shortest path problem with dynamically changing arc costs [12].

Adaptive Parallel Arithmetic Optimization Algorithm (APAOA) is a parallel communication strategy used for robot path planning [15]. Rapidly-Exploring Random Trees (RRT) are designed to handle high-dimensional nonholonomic planning problems [8]. Ant Colony Algorithm is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs [1]. Firefly Algorithm is inspired by the flashing behavior of fireflies, used for solving optimization problems [16]. Genetic Algorithms are search heuristics that is inspired by Charles Darwin's theory of natural evolution [7].

3.2 Use of Genetic Search

Novel Knowledge-Based Genetic Algorithm for Robot Path Planning in Complex Environments proposes a knowledge-based genetic algorithm for path planning of a mobile robot in unstructured complex environments. The algorithm incorporates the domain knowledge of robot path planning into its specialized operators, some of which also combine a local search technique. The proposed algorithm is capable of finding a near-optimal robot path in both static and dynamic complex environments [5].

In Genetic Algorithm-Based Trajectory Optimization for Digital Twin Robots, authors use genetic algorithms for path planning of robots, which enables trajectory optimization of mobile robots by reducing the error in the movement trajectory of physical robots through the interaction of virtual and real data. It provides a method to map learning in the virtual domain to the physical robot [9].

Another work is presented as Mobile Robot Path Planning Using Multi-Objective Genetic Algorithm. The MRPS-MOGA is designed with the novelty of genetic algorithm with multiple objective function to solve mobile robot path planning problems. It handles five different objectives such as safety, distance, smoothness, traveling time, and collision-free path to obtain optimal path [13].

3.3 Genetic Search with A*

Global path planning for mobile robot based on A* algorithm and genetic algorithm proposes a hybrid method based on the A* algorithm and genetic algorithm in a grid map to solve the optimum path planning for mobile robots. The A* algorithm is utilized for searching a constrained shortest path quickly, and the global optimal path is obtained by using a genetic algorithm to optimize the path [18].

Another research work uses a hybrid of the A* algorithm and genetic algorithm for global optimal path planning of mobile robots. The MAKLINK graph theory is used to establish the free space model of mobile robots, then the Dijkstra algorithm is utilized for finding a feasible collision-free path, and finally, the global optimal path of mobile robots is obtained based on the hybrid algorithm [17].

Robot path planning based on A* algorithm and genetic algorithm proposes a combination algorithm based on a genetic algorithm and A* algorithm for optimizing the path planning of a medical service robot. The hospital map was modeled in two dimensions using the raster method [11].

In the following proposed research, genetic search is combined with graph search algorithms, while adding performance optimizations for dynamically changing fitness measures for desired routes.

4 Problem description

The path-finding problem is formulated as a routing challenge for autonomous robots inside a two-dimensional floor plan for a warehouse. The floor plan is represented as a directed planar graph of vertices and edges $G = \{V, E\}$. vertices in the graph may represent intersections, turns, and stopping points. Two vertices may be connected by a directed edge. A graph can be represented as an adjacency matrix, where the elements represent the weights of edges connecting vertices at the corresponding indices. For example, $G(v_i, v_j) = \infty$ signifies no connection between vertices v_i and v_j.

For a robot given its starting vertex, the objective is to traverse a set of stop vertices $v \in V_s \subseteq V$ in an optimal order. The path between any two vertices (v_i, v_j) is called a segment $s = S(v_i, v_j) = \{v_i, v_{i+1}, v_n, ..., v_j : i \leq n \leq j, G(v_n, v_{n+1}) \neq \infty)\}$. A segment may traverse multiple vertices, eventually connecting the stop vertices. The path connecting multiple stop points is the route. At any instant, there may be multiple occupants in the warehouse. No two robots may occupy the same edge between vertices, or the same vertex, simultaneously.

A route r is represented as a vector of stop vertices such that there are segments connecting them $r = \{v_i \in V_s, \forall i, j : i \neq j \iff v_i \neq v_j\}$, where the first element is the starting vertex. An optimal solution has the smallest cumulative cost over segments.

5 Approach

The hybrid search method is described in this section.

5.1 Fitness function

Optimality of the overall route is quantified by a fitness function of the following measures.

1. The distance traversed on a route. Distance traveled will affect the wear and tear of mechanical components. Robots traversing longer routes for the same operations will incur a larger maintenance cost for the same productivity.

2. The energy consumption along a route. Routes may have different energy requirements depending on path geometry and ground topology. Energy consumption is a direct material cost for operations.

3. The time taken to traverse the route. Physically shorter routes may take longer to traverse due to congestion. Therefore time to target is an independent measure of optimality.

And optimal route will consist of segments whose accumulated fitness is maximal. Therefore the individual segments will accrue a maximal fitness as well.

$$-f_s(v_i, v_j) = \alpha_d distance(v_i, v_j) + \alpha_e energy(v_i, v_j) + \alpha_t time(v_i, v_j) \quad (1)$$

$$S(v_i, v_j) = \arg\max_{s \in S} f_{segment}(v_i, v_j) \quad (2)$$

$$f_r(\text{route}) = \sum_{i,j \in \text{route}} f_s \quad (3)$$

Finding an optimal segment between two vertices can be achieved in $\mathcal{O}(E \log |V|)$ time using A^* search (algorithm 1).

In the worst case, a route or candidate routes may contain segments that altogether traverse every edge in the graph. That is, pair-wise fitness between all nodes will need to be calculated. This can be done in $\mathcal{O}(V^3)$ using the Floyd-Warshall algorithm. For a planar graph, the number of edges is limited as $|E| \leq 3|V| - 6$. Therefore the complexity of running A^* for each pair will be $\mathcal{O}(V^3 \log |V|)$. In this case, the graph topology is not stationary. Due to evolving traffic and costs of energy, pair-wise optimal segments may need to be calculated repeatedly. Thus, lazily evaluating optimal segments during route search and caching the results will save the computational cost node pairs not relevant to the routing problem at hand. In the worst cast, the computational performance will be slower than a factor of $log|V|$.

5.2 Genetic Search

The search function is provided with graph representing the floor plan (including vertex congestion, vertex distances, and energy costs), the starting vertex, and the desired stopping points. Then, genetic search iteratively evaluates candidate solutions.

5.2.1 Initialization

The search begins by creating an initial population $P_0 = \{r_1, ...r_p\}$ of p candidates. The population is created by randomly permuting the stopping points. For each chromosome in the population, the fitness value is calculated.

5.2.2 Selection

A subset of the next generation is selected from the prior generation. Chromosomes with a higher fitness value are selected with a proportionally larger probability. The probability of selection into the next generation is given my $P(r \in P_g \ \& \ r \in P_{g+1} = f_r(r) / \sum_{r \in P_g} f_r(r)$.

Algorithm 1 A* Search Algorithm

```
1: procedure ASTAR(adjacency matrix G, vertex start, vertex goal, f_r)
2:     openList ← PriorityQueue()                ▷ Priority queue for open vertices
3:     openList.push(startvertex, g(startvertex) + h(startvertex))   ▷ Push start vertex with its f value
4:     closedList ← zeros(1, numvertices) ▷ Closed list to track visited vertices
5:     parent ← zeros(1, numvertices)            ▷ Parent array to reconstruct path
6:     while ¬openList.isEmpty() do
7:         currentvertex ← openList.pop()
8:         path ← reconstructPath(parent, currentvertex)
9:         if currentvertex == goalvertex then
10:            return path
11:        end if
12:        closedList(currentvertex) ← 1
13:        neighbors ← find(0 < G(currentvertex, :) < ∞)
14:        for i ← 1 : length(neighbors) do
15:            neighbor ← neighbors(i)
16:            if closedList(neighbor) == 1 then
17:                continue
18:            end if
19:            tentativeG ← f_r(path)
20:            if ¬openList.contains(neighbor) ∨ tentativeG < g(neighbor) then
21:                parent(neighbor) ← currentvertex
22:                g(neighbor) ← tentativeG
23:                fValue ← g(neighbor) + h(neighbor)
24:                if ¬openList.contains(neighbor) then
25:                    openList.push(neighbor, fValue)
26:                end if
27:            end if
28:        end for
29:    end while
30:    return empty path                          ▷ No path found
31: end procedure
32: procedure RECONSTRUCTPATH(parent, goalvertex)
33:    path ← []
34:    vertex ← goalvertex
35:    while vertex > 0 do
36:        path ← [vertex, path]
37:        vertex ← parent(vertex)
38:    end while
39:    return path
40: end procedure
```

5.2.3 Crossover

For each chromosome, a segment is selected, reversed, and inserted back at the same position. That is, the the ordering of a subset of stopping points is reversed, excepting the start vertex.

5.2.4 Mutation

Mutation simply swaps two genes in a chromosome. That is, the two stopping points exchange places in the order of stops, excepting the starting vertex.

5.3 Post-processing

Crossover and mutation operations, if left unsupervised, may lead to suboptimal routes. This is because when the order of stopping points is changed, new paths are calculated between adjacent stops. It is possible that a later stop point is already traversed in the optimal pairwise path between earlier stop points. Therefore, if the shortest pairwise path between two vertices v_i, v_j contains another stopping point v_k, then the optimal ordering of stops is v_i, v_k, v_j. Thus, any chromosome that contains v_k at an index outside the indices of v_i, v_j is discarded.

Furthermore, repeated calls to the A search function are memoized in an array. Therefore multiple chromosomes with shared orderings of stop points do not accrue additional computational cost of pairwise path calculation. Instead, the optimal pairwise paths are calculated in amortized constant time via simple array lookup.

6 Experiments

The proposed approach is tested in a floor plan for a delivery goods warehouse. The floor plan has contains pathways and constant obstacles. Additionally, closed routes due to congestion may be represented by edge weights between vertices. Figure 1a shows a visualization of the layout, with the relative congestion overlaid in a heat map. Figure 1 shows the internal graph representation of the same layout, showing bi-directional paths between vertices. The cost of traversal, including energy and congestion, is encoded in edge weights.

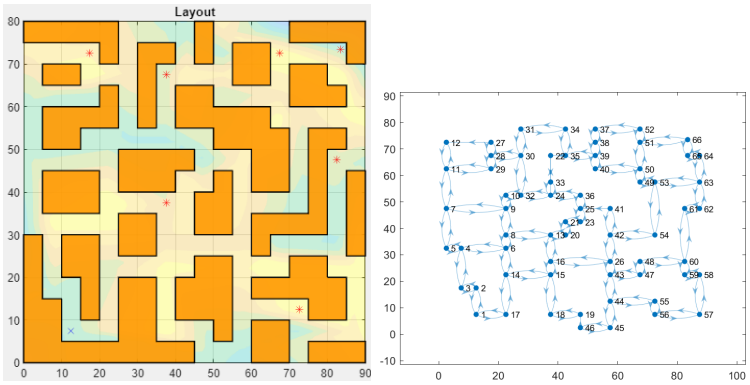
A globally optimal routing can be calculated via simulation. That is, an exhaustive search of all permutation of stopping points, where each pair of adjacent stops is routed via A search. Figure 2 shows the results of an exhaustive A* search versus genetic search informed by A. As the number of stops increases, the time taken for exhaustive search scales exponentially. However, genetics search is able to maintain a constant time cost (figure 2a). Figure 2b shows the fitness distribution of routes between the two approaches. Both are able to find routes with the same cost.

Algorithm 2 Post processing

```
1: procedure PostProcess(population, allPairsPaths,)                    ▷
2:     post = [];
3:     for i=1:length(population) do
4:         ordering = population(i);
5:         nodesRepeat = false;
6:         for j=1:length(ordering)-1 do
7:             v_i = ordering(j); v_j =ordering(j+1);
8:             if isempty(allPairsPaths(v_i,v_j)) then
9:                 path = AStar(G, v_i, v_j, fitnessLocal);
10:                allPairsPaths(v_i, v_j) = path;
11:            else
12:                path = allPairsPaths(v_i, v_j);
13:            end if
14:            nodesRepeat = nodesRepeat || isempty(intersect(path, ordering(j+1:end)));
15:        end for
16:        if ¬ nodesRepeat then
17:            post = [post; ordering];
18:        end if
19:    end for
20: end procedure
```

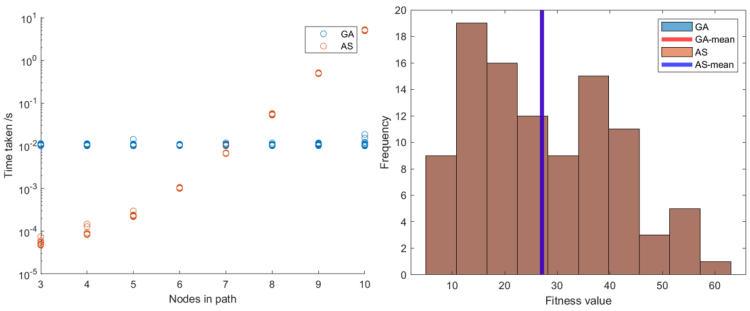
Algorithm 3 Genetic Search Algorithm

```
1: procedure GeneticSearch(a, b, f)        ▷ Optimal path between a, b by f
2:     Input: Population size N, Number of generations G, Crossover probability P_c, Mutation probability P_m
3:     Output: Optimized solution
4:     Initialize population P with N random solutions
5:     for gen ← 1 to G do
6:         Evaluate the fitness of each individual in P
7:         Select individuals for crossover and create offspring
8:         for each pair of parents selected for crossover do
9:             if random number < P_c then
10:                Perform crossover to create offspring
11:            end if
12:        end for
13:        Mutate the offspring with probability P_m
14:        Evaluate the fitness of the offspring
15:        Select individuals for replacement
16:        Replace individuals in the current population with the offspring
17:    end for
18:    return Best solution found in the final population
19: end procedure
```



(a) The floor layout (b) The graph representation

Figure 1: The experimental setup



(a) Genetic search versus exhaustive A* search (b) Genetic search versus exhaustive A* search

Figure 2: Results

7 Conclusion

In this work, a genetic search algorithm informed by A* search was devised for routing planning for robots under dynamic route fitness conditions. Exploiting the optimal nature of A* for pairwise distances, and the constant-time performance of genetic search, a fitness of routing near-identical to exhaustive search is obtained without expensive time scaling. As the fitness function changes, lazy evaluation and memoization ensure that only vertices encountered during the search are re-evaluated, and only once.

References

[1] Christian Blum. "Ant colony optimization: Introduction and recent trends". In: *Physics of Life reviews* 2.4 (2005), pp. 353–373.

[2] František Duchoň et al. "Path planning with modified a star algorithm for a mobile robot". In: *Procedia engineering* 96 (2014), pp. 59–69.

[3] Merrill M Flood. "The traveling-salesman problem". In: *Operations research* 4.1 (1956), pp. 61–75.

[4] Amir Gharehgozli and Nima Zaerpour. "Robot scheduling for pod retrieval in a robotic mobile fulfillment system". In: *Transportation Research Part E: Logistics and Transportation Review* 142 (2020), p. 102087.

[5] Yanrong Hu and Simon X Yang. "A Novel Knowledge-Based Genetic Algorithm for Robot Path Planning in Complex Environments". In: *arXiv preprint arXiv:2209.01482* (2022).

[6] Vu Quang Huy et al. "Design and Develop Optimal Pathfinding Algorithm Applied in Book Accessing and Returning for Autonomous Mobile Robot in Library". In: *2022 6th International Conference on Green Technology and Sustainable Development (GTSD)*. IEEE. 2022, pp. 350–354.

[7] Annu Lambora, Kunal Gupta, and Kriti Chopra. "Genetic algorithm-A literature review". In: *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE. 2019, pp. 380–384.

[8] Steven M LaValle, James J Kuffner, BR Donald, et al. "Rapidly-exploring random trees: Progress and prospects". In: *Algorithmic and computational robotics: new directions* 5 (2001), pp. 293–308.

[9] Xin Liu et al. "Genetic algorithm-based trajectory optimization for digital twin robots". In: *Frontiers in Bioengineering and Biotechnology* 9 (2022), p. 793782.

[10] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. "CBSS: A New Approach for Multiagent Combinatorial Path Finding". In: *IEEE Transactions on Robotics* (2023).

[11] Tang Xiang Rong and Xu Hong Mei. "Robot path planning based on A* algorithm and genetic algorithm". In: *2020 International Conference on Computer Engineering and Intelligent Control (ICCEIC)*. IEEE. 2020, pp. 101–105.

[12] Anthony Stentz et al. "The focussed dˆ* algorithm for real-time replanning". In: *IJCAI*. Vol. 95. 1995, pp. 1652–1659.

[13] KS Suresh, R Venkatesan, and S Venugopal. "Mobile robot path planning using multi-objective genetic algorithm in industrial automation". In: *Soft Computing* 26.15 (2022), pp. 7387–7400.

[14] Huijuan Wang, Yuan Yu, and Quanbo Yuan. "Application of Dijkstra algorithm in robot path-planning". In: *2011 second international conference on mechanic automation and control engineering*. IEEE. 2011, pp. 1067–1069.

[15] Ruo-Bin Wang et al. "An adaptive parallel arithmetic optimization algorithm for robot path planning". In: *Journal of Advanced Transportation* 2021 (2021), pp. 1–22.

[16] Xin-She Yang and Xingshi He. "Firefly algorithm: recent advances and applications". In: *International journal of swarm intelligence* 1.1 (2013), pp. 36–50.

[17] Cen Zeng, Qiang Zhang, and Xiaopeng Wei. "Robotic global path-planning based modified genetic algorithm and a* algorithm". In: *2011 Third International Conference on Measuring Technology and Mechatronics Automation*. Vol. 3. IEEE. 2011, pp. 167–170.

[18] Liang Zhang et al. "Global path planning for mobile robot based on A* algorithm and genetic algorithm". In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2012, pp. 1795–1799.