NASA/TM—20220016356

Exploring Applications of Machine Learning for Wildfire Monitoring and Detection using Unmanned Aerial Vehicles

** All authors contributed equally to this memo. The names are listed alphabetically by first name.*

Aanvi Koolwal
Irvington High School

Aarfan Hussain
Arnold O. Beckman High School

Adityan Vairavel
Dougherty Valley High School

April Zelinski
Washington High School

Iulia Iordanescu
Acton Boxborough Regional High School

Mathew Zheng
Aragon High School

September 2022

NASA/TM—20220016356

Exploring Applications of Machine Learning for Wildfire Monitoring and Detection using Unmanned Aerial Vehicles

** All authors contributed equally to this memo. The names are listed alphabetically by first name.*

Aanvi Koolwal
Irvington High School

Aarfan Hussain
Arnold O. Beckman High School

Adityan Vairavel
Dougherty Valley High School

April Zelinski
Washington High School

Iulia Iordanescu
Acton Boxborough Regional High School

Mathew Zheng
Aragon High School

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, CA 94035-1000

September 2022

Acknowledgments

The authors would like to acknowledge **David Thipphavong** and **Min Xue** for their continual guidance throughout this study.

Abstract

Wildfires are increasing in frequency and severity around the world, including the United States. The losses caused by wildfires could be mitigated if high-risk areas, hotspots, and flare-ups could be monitored continuously, such as through the use of Unmanned Aerial Vehicles (UAVs). This paper documents exploratory efforts using machine learning to determine efficient flight paths for UAVs and to detect wildfires using image classification. On path planning, three machine learning techniques—Genetic Algorithm, Simulated Annealing, and Dynamic Programming—were explored. Genetic Algorithm was found to be an effective approach for path planning for wildfire monitoring and surveillance by UAVs. For a scenario of 25 locations in a circular arrangement, the algorithm was able to return the optimal path. The accuracy and execution time was found to be sensitive to the algorithm hyperparameters selected, which was especially evident in scenarios with hundreds or thousands of locations. Simulated Annealing was also found to be an effective approach for UAV path planning, with a major benefit of avoiding getting trapped in local minima and being straightforward to implement. Like Genetic Algorithm, the performance of Simulated Annealing was also found to be sensitive to the algorithm hyperparameters selected. By comparison, Dynamic Programming guarantees optimality for any number of locations, but it was found to be less practical in terms of execution time for scenarios with more than about a couple dozen locations. On wildfire detection, image classification using deep learning with a convolutional neural network was explored. Transfer learning was found to be a useful technique to efficiently train deep learning models. Also, it was determined that GPU processing can increase training speed by an order of magnitude, which enables significantly faster development. For a validation test set of 500 images, there were only two false negatives and zero false positives. These results demonstrate that detecting wildfires in static cameras using machine learning is feasible and establish a baseline for using images captured by UAVs in flight for wildfire detection.

I. Introduction

Wildfires are a severe and rising problem in the United States. Historically, wildfire season spans July through September or October, the drier months of the year. However, because of climate change, wildfire season is now year round. Low precipitation, dry weather, droughts, and available fuel are all needed for a wildfire to occur naturally. Due to climate change, this particular set of conditions occurs more frequently than in the past. Wildfire "fuel" refers to combustible vegetation, usually dry, over which flames can burn and spread easily, allowing for a wildfire's quick growth. This is especially prevalent over much of the Western United States. For example, California is currently in the midst of a megadrought, with significant portions of the state that are brown, dry, and exposed, which make it easily ignitable by a heat source. Strong prevailing winds can cause wildfires to grow and spread quickly [1].When a wildfire occurs naturally, its heat source is usually in the form of a lightning strike. However, lightning strikes are not common, and wildfires are rarely natural. In fact, humans cause 84% of wildfires, with campfires, sparking power lines, and cigarettes being sources of ignition. Human-caused fires are responsible for the most damaging fires [2].

The losses caused by wildfires could be mitigated if high-risk areas, hotspots, and flare-ups could be monitored continuously, such as through the use of Unmanned Aerial Vehicles (UAVs). Due to size, weight, and power constraints, UAVs need to be able to travel over efficient flight paths for UAVs to detect and monitor wildfires. The contribution of this paper is the exploration of three machine learning techniques—genetic algorithm, simulated annealing, and dynamic programming—to solve the Traveling Salesman Problem (TSP) [3] to find the shortest path to visit each location in a given set of locations exactly once and return to the starting location, which is a good representation of the mission to perform detection, monitoring, and surveillance of wildfires.

The tradeoffs between accuracy and execution time of the solutions generated by the machine learning algorithms were evaluated. In addition, the feasibility of image classification using deep learning with a convolutional neural network to detect wildfires in static images was explored. As part of this, transfer learning to efficiently train deep learning models and using GPU processing to increase training speed were also investigated.

Section II provides background on how the increase in the Wildland Urban Interface (WUI) where humans live in or near wooded areas is a factor in the increase in wildfires. Section III describes the wildfire-related problems and corresponding solutions that were identified in this study. Section IV explains how the UAV path planning problem is modeled as the Traveling Salesman Problem and documents the three approaches that were explored—genetic algorithm, simulated, annealing, and dynamic programming—and the corresponding results. Section V documents complementary exploratory work on image classification using deep learning for wildfire detection. Section VI discusses the findings of the exploratory efforts in this study and details potential follow-up work. Section VII summarizes the work that was performed and what was learned.

II. Background

Humans, living near wooded and fire-susceptible areas and being the root cause of most wildfires, make the growing Wildland Urban Interface (WUI) a great concern. According to a 2022 report by Federal Emergency Management Agency (FEMA) and the U.S. Fire Administration, WUI is defined as "an area where human development meets or intermingles with undeveloped wildland and vegetative fuels that are both fire-dependent and fire-prone" [4]. When wildfire hits the WUI, life, property, and natural resources, like timber, are put at risk and often lost.

From 2005 through 2020, around 90,000 structures have been destroyed by wildfires in the United States [5]. Of these, the most damaging wildfires have occurred over the past few years, accounting for 62% of the damage over those 15 years [4]. On top of that, the WUI is increasing by 2 million acres per year, which is putting more and more property and people at risk [4]. The WUI expanse involves several factors. First, climate change is causing the land to become drier and, thus, more susceptible to fire. In addition, population growth and population migration towards the Western and Southern regions of the United States has resulted in greater demand for more housing to be built, including in fire-susceptible WUI areas. Finally, with the cost of living rising in big cities coupled with the desire to live near nature, people are moving to more rural and wooded spaces. The United States also suffers economic losses due to wildfires that range between 80 and 400 billion dollars lost annually [4]. Wildfires also present a significant health risk to both civilians and firefighters. Short-term exposure to wildfire smoke can affect cardiovascular and respiratory health temporarily. However, long-term exposure greatly increases the likelihood of cancer, heart disease, and chronic respiratory disease, all of which often affect firefighters.

III. Problems & Solutions Identified

During the course of conducting background research on wildfires, four major problem areas were discovered. First, a significant amount of fuel has been building up over the years. Second, firefighter safety and health are constantly at risk as they need to contain fires during more of the year (or even year round). Third, communication in the field can be challenging and could be improved. Fourth, there are large areas of land that are very susceptible to fire that are difficult to patrol from the ground.

Fuel Buildup

Climate change has resulted in the creation of more fuel for wildfires. In increasingly hotter and drier climates, vegetation dries out quicker, increasing the chance for a wildfire to start, grow and spread [6].

Prescribed fires (also known as prescribed burns or controlled burns) are recommended to reduce the fuel buildup. Safely burning potentially dangerous areas decreases the risk of a wildfire breaking out and causing major damage. With fire professionals, safety personnel, and equipment present during a prescribed fire, the flames are much easier and safer to contain and put out. UAVs could be used in the identification of areas with large fuel build up, as well as in monitoring a prescribed fire's progress from above. This would be particularly useful during burns over a great area.

Firefighter Health

Firefighter health is constantly at risk during wildfires. Their health is jeopardized most by the high concentration of carbon monoxide and metalloids found in smoke [7], which firefighters are exposed to often and over long periods of time.

UAVs could be used to observe and detect smoke so that firefighters can avoid unnecessary smoke exposure. For example, if UAVs determined that smoke was blowing towards the North, firefighters could be directed to approach from a different direction. UAVs could also potentially be used to filter smoke if equipped with a HEPA filter and a heavy-activated granule filter. Lastly, UAVs could be used to help extinguish the fire, thus avoiding human risk.

Communications

Communications are critical, especially when under harsh conditions like those of a wildfire. Today, two-way radios are still the primary method of communications between firefighters. On its own, this is not a huge problem. However, in combination with the use of Personal Protective Equipment (PPE) and Self-Contained Breathing Apparatuses (SCBAs), communication can be affected. SCBA can muffle audio or inadvertently create interference [8-9], and PPE gloves make it challenging to operate a radio [8], which can have difficulty transmitting if not oriented properly. There is also a lack of frequencies devoted specifically to wildfire suppression, and the frequencies that are available are high frequency, which do not travel as far as low-frequency radio waves. Weak signal strength can also be a problem, especially in more remote areas where wildfires often occur [8].

To mitigate the problem of communicating via radio when outfitted with PPE and in an SCBA, a radio and/or other communication devices could be integrated into the SCBA. Preferably, these devices would be voice controlled for ease of use by firefighters wearing thick gloves. To mitigate the lack of frequencies, besides allocating more frequencies to wildfire management, multiple transmitters could be deployed at different locations transmitting at different frequencies and at different times. In addition, UAVs operating as remote WiFi hubs or repeaters could be used to boost signal strength. Currently, NASA's Scalable Traffic management for Emergency Response Operations (STEReO) project is working on improving communications through communication networks, remote sensors, and common information distribution [10].

Large Areas at Risk*

The areas of land at risk of major wildfires are so vast that continual monitoring for and of wildfires on the ground is not feasible. The same can be said for the "mopping up" process after a fire has been contained. "Mopping up" refers to the additional surveillance and clean up of an area after flames have been extinguished and includes the removal of burning or hazardous materials [11].

Toward solving this problem, UAVs equipped for surveillance can be utilized to aid in fire detection, monitoring, and mopping up. Developing pathway-finding and fire detection software for UAVs would enable them to efficiently survey areas at risk, detect wildfires early on, and alert local fire departments before the wildfires become fully developed.

**The only problem for which solutions were developed further. Three path planning algorithms—genetic algorithm, simulated annealing, and dynamic programming—were explored and a smoke recognition algorithm developed from static camera images was investigated. The findings from these efforts are documented in the subsequent sections of this report.*

IV. Path Planning

In this section, three path planning algorithms—genetic algorithm, simulated annealing, and dynamic programming—were explored towards efficient wildfire monitoring across multiple sites in a large area. This section will begin with a description of the Traveling Salesman Problem (TSP) that each algorithm was applied to solve. Then, the methodology and results will be presented and discussed for each algorithm in turn.

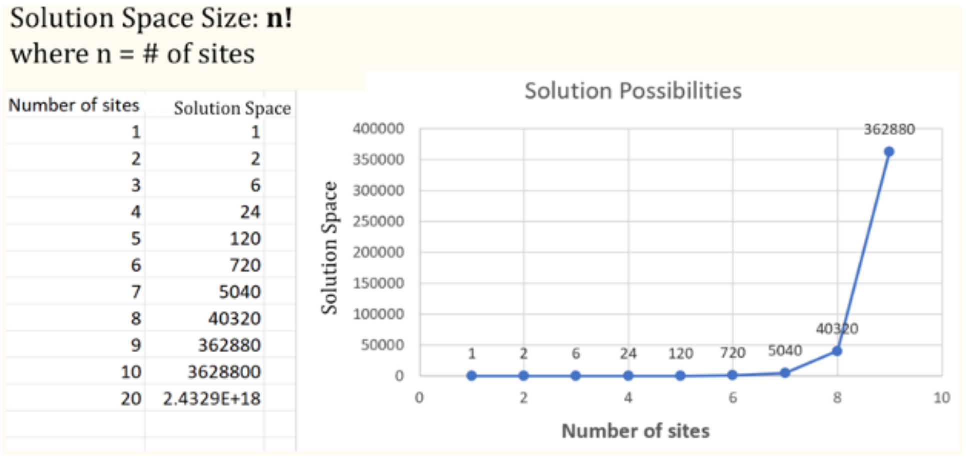
Traveling Salesman Problem

The TSP is a classic optimization problem in theoretical computer science and operations research. It was originally formulated in the 1930s and has been intensely studied, with many algorithms developed and tested to solve it efficiently and accurately. It has applications in logistics, planning, and scheduling. The problem itself is to find the shortest path to visit each node in a given set of nodes exactly once and return to the starting node [3]. It is relevant to the use of UAVs for wildfire management because UAVs have size, weight, and power limitations that constrain flight range. As such, it is important to cover all areas and points of interest in an efficient manner, which solving the TSP would enable.

There are many ways to compute the shortest-distance path. The next subsection discusses exhaustive search and why it is not a feasible approach. Subsequent subsections present the other algorithms explored in this study—genetic algorithm, simulated annealing, and dynamic programming—and the corresponding results of applying them to the TSP.

Exhaustive Search

In exhaustive search, all possibilities are attempted and therefore the global optimum will always be found. However, as illustrated in Figure 1, this method is not always feasible because the size of the potential solution space increases factorially as a function of the number of sites. It is computationally intensive to find the optimal solution for a set of more than 10 locations by this method.


Solution Space Size: **n!**
where n = # of sites

Number of sites	Solution Space
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
20	2.4329E+18

Figure 1. Exhaustive search solution space size for *n* number of sites.

Genetic Algorithm

Genetic Algorithm (GA) can find a near-optimal solution much quicker than exhaustive search. As described in [10], "a genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation." It uses an evolving subset of solutions (GA population size) to find the optimum of that subset. It can be thought of as a guided trial-and-error methodology.

Methodology

As seen in Figure 2, GA has four main processes: initialization, reproduction, selection, and termination testing. Initialization occurs only once at the beginning of execution whereas the other three processes occur once per generational iteration.

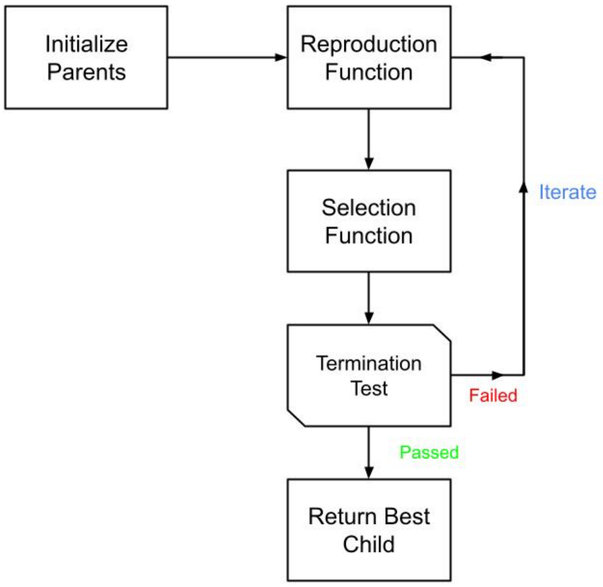


Figure 2. GA decision-making process.

In the initialization step, the initial parent path and several numerical hyperparameters (parent probability weight, maximum mutations per child, chance of mutation, birth size, stagnation, and generation cap) are inputs to the rest of the algorithm. In the implementation of the algorithm for this study, each of the two initial parent paths and subsequent child paths are stored as a 2-D Python array of coordinates in (x, y)-tuple format. In the context of UAV path planning, a gene is a single (x, y)-tuple pair in a path. A path can be visualized by connecting each point in the list to the point directly after it using a line segment. The implementation of the algorithm for this study does not return to the starting point, though this is possible by re-adding the first point to the end of the parent path. The fitness of a path is defined as the total distance calculated by taking the sum of all line segment lengths. After the initialization step using the two initial parent paths, the algorithm begins generational iteration.

The two main functions in the algorithm are the reproduction and selection functions. It begins by taking the two fittest organisms (i.e., those with the shortest path distance), either the original parents (if the first generation) or the two fittest children (if not the first generation and they are fitter than the previous generation's parents). Then, *n* number of children are created using two helper functions, crossover and mutation. Both functions are called once to create each child. As illustrated in Figure 3, the crossover function iterates through the length of the parents' paths, with iteration variable *i* in both parents. A parent is selected at random based on the parent probability weight hyperparameter. If the chosen parent's gene at index *i* is not already present in the child, it appends the gene to the child. If it is already present, a gene in the chosen parent that is not already in the child is appended to the child. When crossover is complete, the child has all genes. The child is then mutated. For the number of mutations specified (and the chance of mutation), two genes are swapped randomly as illustrated in Figure 4.

```
path_1 = [A, E, C, D, B]

path_2 = [A, B, D, C, E]

child_path = [A, B, C, D, E]
```

Figure 3. Example of crossover in reproduction.

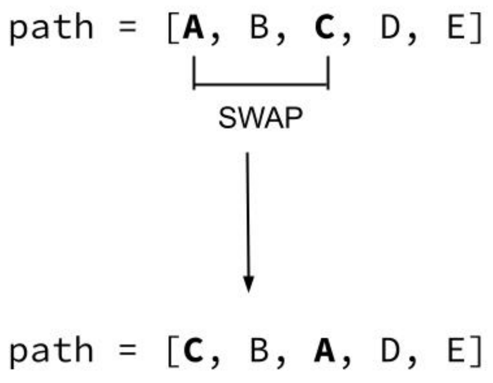


Figure 4. Example of mutation in reproduction.

Every child created in these steps is appended to a list of children and passed to the selection function when the reproduction function is completed. The two fittest children are appended to a list and then compared with the generation's parents. The parents are replaced with the children if the latter are fitter. If not, the parents are left unmodified. The algorithm repeats these processes until the termination condition (either stagnation or generation cap) returns true. When this occurs, the fittest parent is returned as the final path.

GA performance is influenced by the algorithm hyperparameters that are set. Parent probability weight controls the likelihood of the fittest parent transferring its genes to the child. A higher value means the child will inherit more genes from the fittest parent. Maximum mutations per child controls how many genes can be swapped during reproduction. Chance of mutation controls the probability of a mutation occurring. Birth size controls how many children are produced in each generation. Stagnation is the maximum number of children with equivalent *f* fitness that are allowed before termination. If *n* number of children with *f* fitness are found, the algorithm returns the most recent child with *f* fitness. Generation cap controls how many generations are produced before the algorithm terminates and returns a final path.

Results

A 25-location system in a circular arrangement was used to test the algorithm because it is easy to determine the fittest path by eye, but the algorithm operates as though they were any other set of points. This enabled rapid code development (https://github.com/theyhaveGrean/adityan_nasa), testing of the accuracy of the algorithm, and determination of strengths and weaknesses. Figure 5 is an example of the initial path (left) compared to the GA-developed path (right). In this case, the algorithm returned the optimal path with a fitness of 151.01, which is an improvement by a factor of four compared to the initial path that had a fitness of 742.54, with execution time of 10.93 seconds. During performance testing, the algorithm was run five times per set of parameters with randomly generated initial paths each time. The algorithm returned a path that was significantly shorter than the initial path 100% of the time, with an average processing time under 10 seconds.

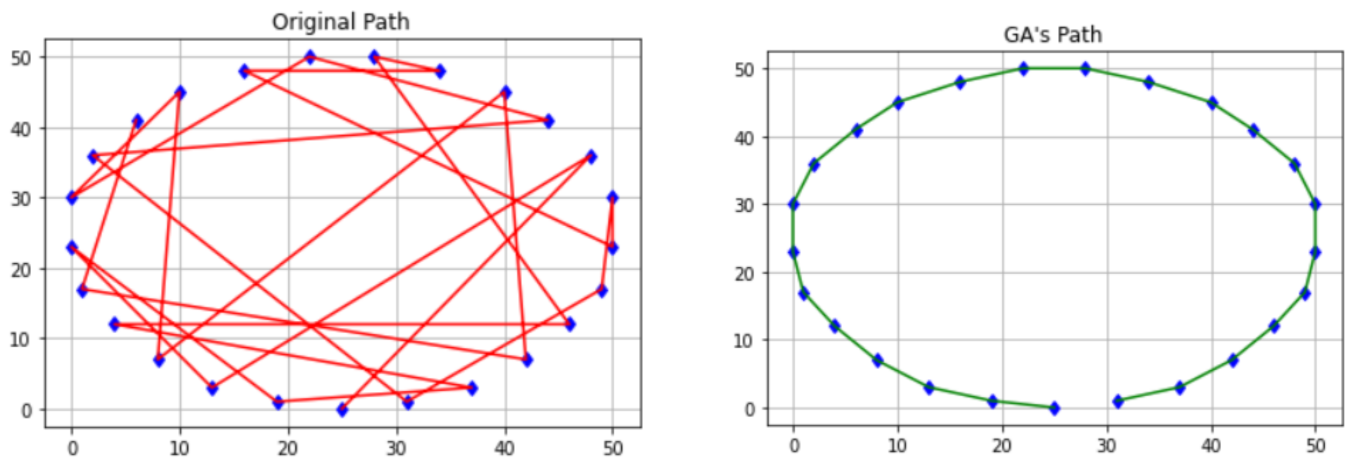


Figure 5. Example 25-location system that was successfully solved with GA. (Left) Initial parent path passed to the GA. (Right) The GA-returned path, which converged on the optimal solution.

Sensitivity Analysis

An extensive sensitivity analysis was performed to investigate how modifying the algorithm's hyperparameters could affect its accuracy and processing time. Modifying the number of points had the expected effect, with more points requiring more processing time and fewer points requiring less processing time.

Increasing the value of some hyperparameters—birth size, stagnation, and generation cap—consistently led to higher accuracy but also higher processing time. However, modifying the values of parent probability weight, maximum mutations per child, and chance of mutation did not have a consistent effect on accuracy. This suggests that there is a "sweet spot" or optimal zone for these hyperparameters for a given number and arrangement of nodes.

To test this, the algorithm was methodically run using different sets of values for these hyperparameters to determine how processing time and accuracy would be affected. For example, increasing the number of mutations per child above the optimal zone increased processing time and decreased fitness. Decreasing it below the optimal zone decreased processing time and fitness. In addition, changing the parent probability weight above 70% and below 30% decreased fitness while maintaining processing time. Through testing, it was determined that the optimal GA hyperparameters that provided the best fitness and processing time for the 25-point system in a circular arrangement was a parent probability weight of 50%, a maximum of three mutations per child, a 100% chance of mutation, a birth size of 500, a stagnation of 10, and a generational cap of 100.

Simulated Annealing

Simulated Annealing (SA) is an algorithm based on the process of physical annealing, which is the heating up of a material, such as a metal or glass, above its recrystallization temperature and then gradually cooling it down to change the material to a desired structure. This can include allowing regions of the material to grow and shrink to increase ductility and reduce stress in the material. As the temperature is lowered, atoms in the material are given time to form a uniform lattice with minimal energy. If the metal is cooled down too quickly, the atoms form a slightly irregular lattice with higher energy due to internal stress [14].

Methodology

The SA algorithm mimics physical annealing. Similar to how the latter seeks to find the best arrangement of atoms for energy minimization, in the case of the TSP, the SA algorithm seeks the route that covers all locations in the shortest distance. In physical annealing, high temperatures are used to allow the atoms to move freely, often moving to positions that temporarily increase the total energy. As the temperature is slowly lowered, the atoms in the material gradually move toward a regular lattice and will only occasionally increase their energy. The same concept is used in SA to find the optimized path that minimizes the cost (i.e., distance) and avoid being trapped in local minima. In terms of complexity, the SA algorithm is $O((n^2 + n)log\ n)$ [14], which is a much quicker heuristic approach compared to exhaustive search which has complexity $O(n!)$.

In a nutshell, the SA algorithm works by randomly perturbing the original path by a gradually decreasing "temperature," which is a parameter that is initially set at a high number and iteratively lowered by multiplying it by a factor close to one. The temperature is a measure of the degree of randomness by which changes are made to the path. A high temperature is utilized at first so that larger random changes occur to reduce the risk of being trapped in local minima. In the implementation of the algorithm in this study, the temperature is decreased exponentially, with each new temperature being 0.99 times the previous temperature.

A Python implementation of the SA algorithm was written for this study and is accessible at https://github.com/KoolAanvi/SimulatedAnnealingAlgorithm. The first step of the algorithm is to order all of the locations into a randomly selected route. Then, at each temperature, one of two transformations takes place: reverse or transport. Reverse is when an alternative path is generated by reversing the locations on a segment of the path. Transport is when a segment is taken out of its position in the path and spliced into a randomly chosen position in the remaining path.

Figure 6 illustrates the first step of a randomly selected path that traverses nine locations. It is represented by the initial path [1, 9, 8, 2, 3, 4, 7, 5, 6].

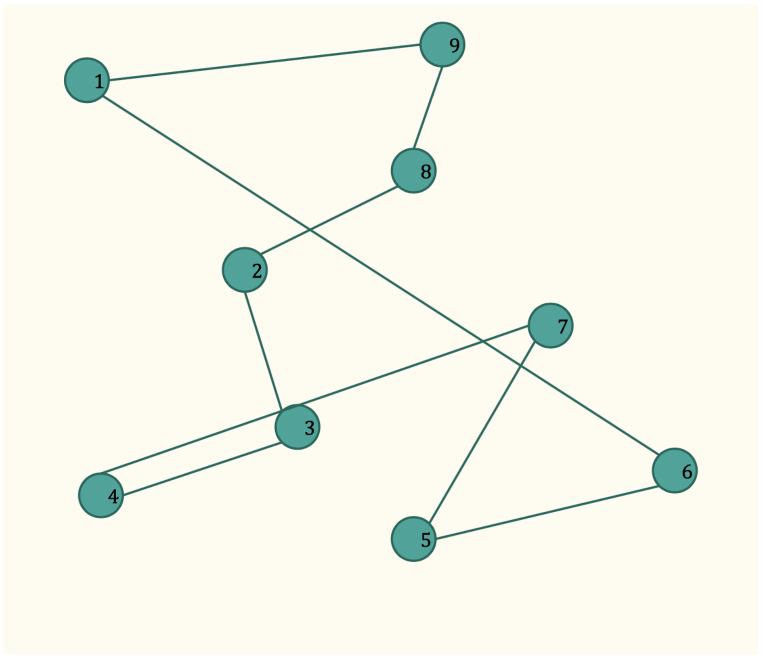


Figure 6. Visual representation of a randomly selected path [1, 9, 8, 2, 3, 4, 7, 5, 6].

Figure 7 shows what happens when the algorithm takes a random segment and uses reverse to transform the route. Here, the segment [8, 2, 3, 4] is selected and reversed to [4, 3, 2, 8]. Then, the original segment is replaced with the reversed segment, resulting in the modified path (drawn in red) of [1, 9, 4, 3, 2, 8, 7, 5, 6] .

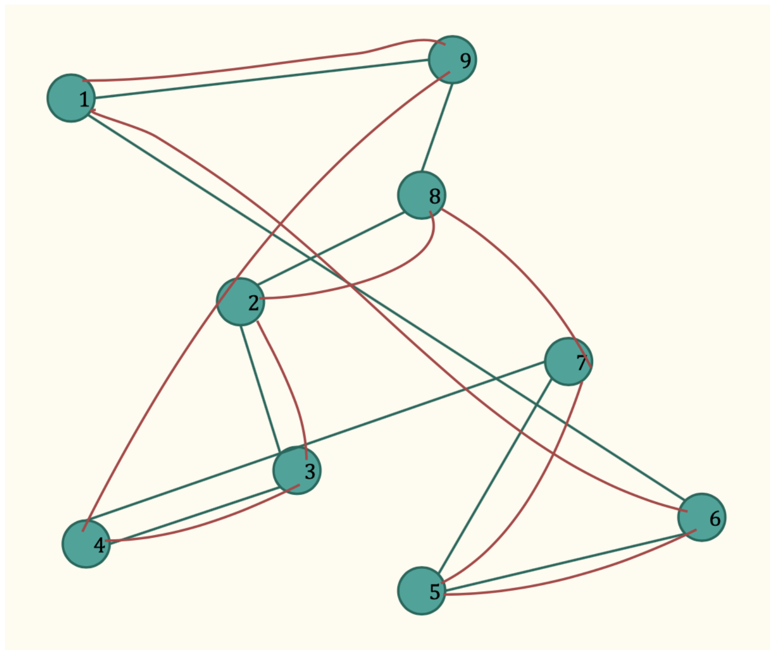


Figure 7. Visual representation of the modified path (drawn in red) resulting from a reverse transformation randomly applied to a segment of the original path.

Figure 8 shows an alternative path to the original path of [1, 9, 8, 2, 3, 4, 7, 5, 6] after being transformed using transport. Here, the segment [8, 2, 3, 4] is selected and spliced into a randomly chosen position, which in this example is between the first two nodes (1 and 9) in the original path. This results in the modified path (drawn in blue) of [1, 8, 2, 3, 4, 9, 7, 5, 6].

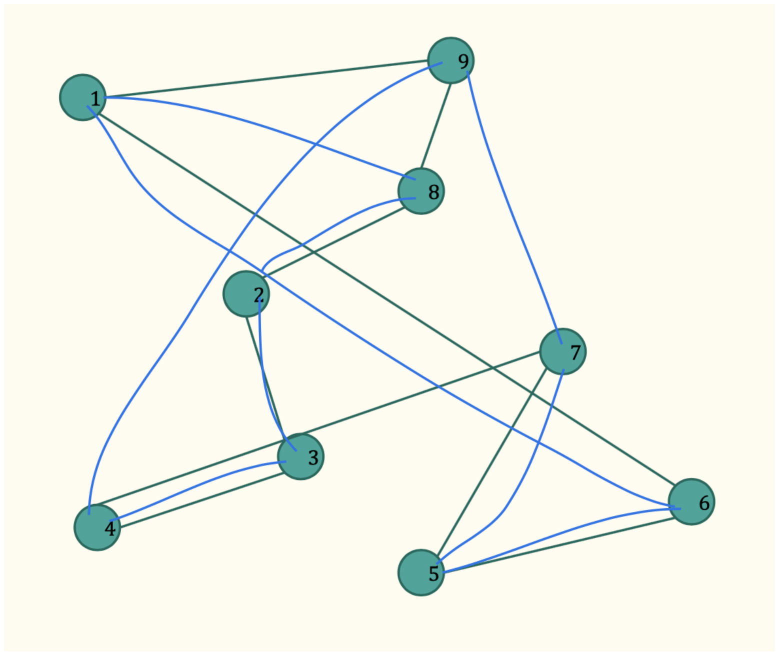


Figure 8. Visual representation of the modified path (drawn in blue) resulting from a transport transformation randomly applied to a segment of the original path.

After each transformation is conducted, the length of the modified path is calculated and compared to the length of the pre-modified path. If the former is less than the latter, then the modified path is utilized in the next iteration of the algorithm. However, if the former is greater than the latter, then the exponential of its negative magnitude divided by the current temperature is compared to 0.99 (the factor by which the temperature is reduced). If this value is greater, then the modified path will be used even though it has a greater length. This is to avoid becoming trapped in a local minimum. This process continues until the length of the path becomes constant.

A higher temperature means a greater probability of making such changes. Therefore, as the temperature falls, only smaller increases in cost are accepted. If no path changes are found after trying all of the possible changes at a given temperature level, the solution is deemed to be "good enough" and the resulting path is displayed.

The process described above is repeated until the length of the path (i.e., cost) does not change. The corresponding path is the optimized path.

Results

Figures 9-11 illustrate the results of running the implementation of the SA algorithm for this study on 10, 50, and 100 locations, respectively. The left graph in each Figure is the initial path. The center graph in each Figure is the optimized path after running the algorithm. The right graph in each Figure is a scatter plot of the cost of the routes calculated at each iteration of the algorithm. As explained in the prior section, the costs do not monotonically decrease because the algorithm accepts paths with higher costs to avoid becoming trapped in local minima. Overall, though, the SA algorithm does tend to decrease the cost to a constant value, for which the corresponding path is the optimized path.

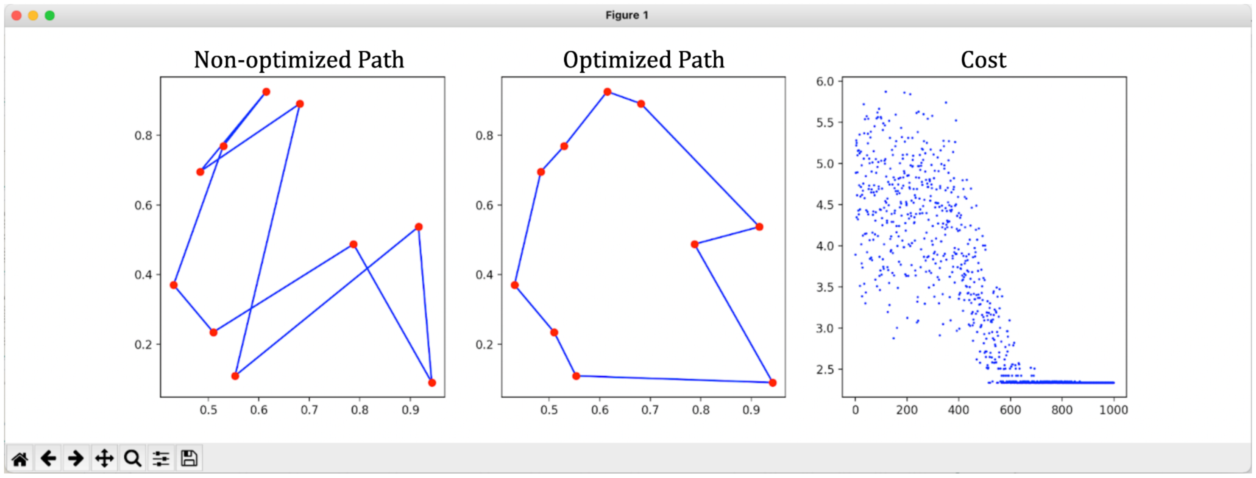


Figure 9. Example 10-location system that was successfully solved with SA. (Left) Non-optimized path. (Center) Optimized path. (Right) Cost of path at each iteration.

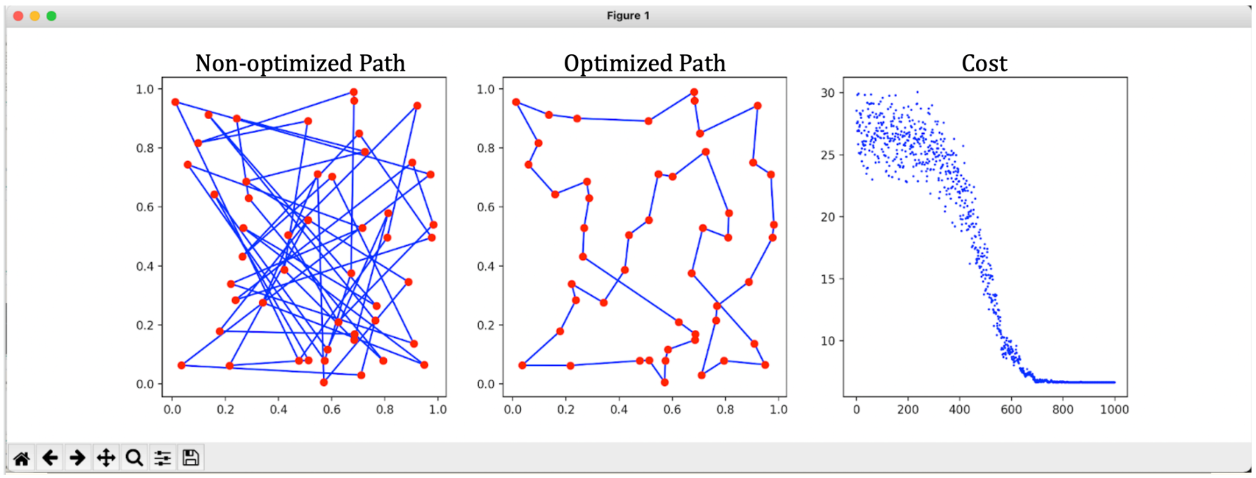


Figure 10. Example 50-location system that was successfully solved with SA. (Left) Non-optimized path. (Center) Optimized path. (Right) Cost of path at each iteration.

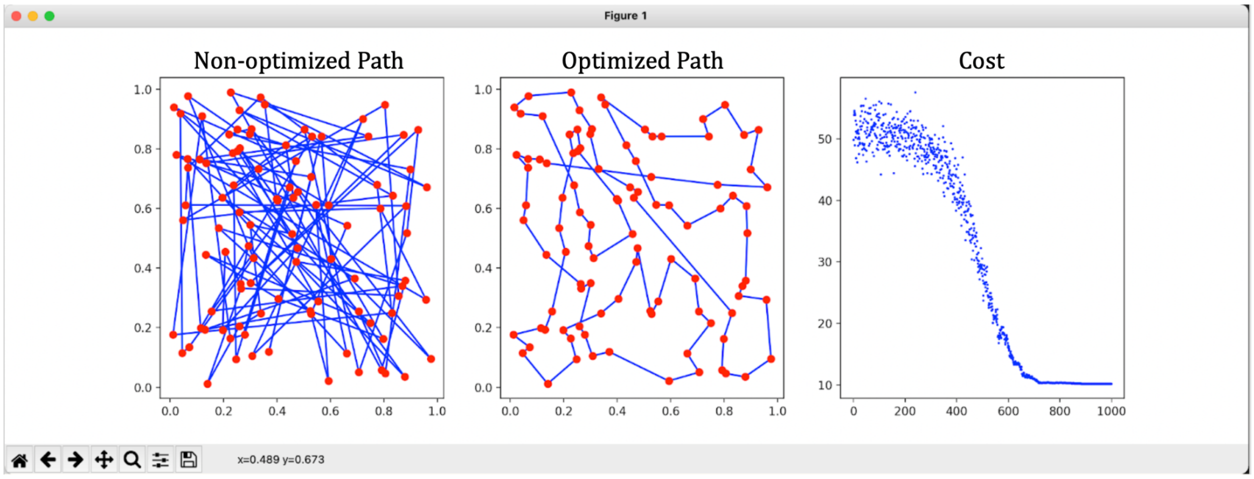


Figure 11. Example 100-location system that was successfully solved with SA. (Left) Non-optimized path. (Center) Optimized path. (Right) Cost of path at each iteration.

Discussion

Overall, the Simulated Annealing algorithm is well-suited for the TSP. It has the ability to avoid becoming trapped in local minima by utilizing intermediate solutions that are moderately worse than the best previous solution. As such, the algorithm is capable of finding the globally optimal solution even after finding a local minimum. In general, it is also relatively easy to implement and provides good solutions after a relatively low number of iterations.

However, the Simulated Annealing algorithm has some drawbacks as well. The algorithm can take a relatively long time to run if a large number of iterations is required. Also, it is not guaranteed that the algorithm will find the globally optimal solution. The algorithm is also dependent on the hyperparameters chosen (e.g., temperature and number of iterations), which can make real-time use challenging.

Dynamic Programming

Dynamic Programming (DP) is an algorithmic technique used to solve optimization problems by breaking them down into smaller and simpler overlapping subproblems [15]. It utilizes the fact that the optimal solution to the overall problem depends on the optimal solutions to its subproblems. The solutions to these subproblems are saved for future reference after they are found to avoid solving a subproblem multiple times, which can significantly reduce runtime from factorial to exponential. More specifically, in terms of complexity, the DP algorithm is $O(n^n + 2^n)$compared to exhaustive search, which is $O(n!)$ [16].

Methodology

For the TSP problem, the DP algorithm reduces runtime by saving prior sub-paths so that they do not need to be computed repeatedly. Dynamic Programming by Memoization is a top-down approach that was explored in this study. It starts by solving the highest level subproblem(s), also known as the base case(s), and works its way down. In the TSP, the base case is the initial node that the DP starts from. From there, the DP by Memoization works its way down to the lower-level subproblems by branching out to other nodes, creating potential solutions or paths. This contrasts with other Dynamic Programming approaches, which use a bottom-up approach, starting with the lowest-level subproblems. In this case, the lowest-level subproblem is the last node in a completed, explored path—the last explored node of the previously defined top-down approach.

The Memoization process starts by identifying the starting node for the problem, which is used both as a reference point to branch out from and as a return point for the TSP at the end of the computation. The next step is the initial computation, in which the optimal value from the starting node to all of the other nodes in the problem is found. By doing so, all TSP problems with two nodes are solved, and problems with three or more nodes now have a base from which to begin to solve.

Figure 12 is a visual example of the first step of the Memoization approach to a four-node/site TSP. As seen, it explores the optimal path from the chosen starting node, 0, to all three other nodes.

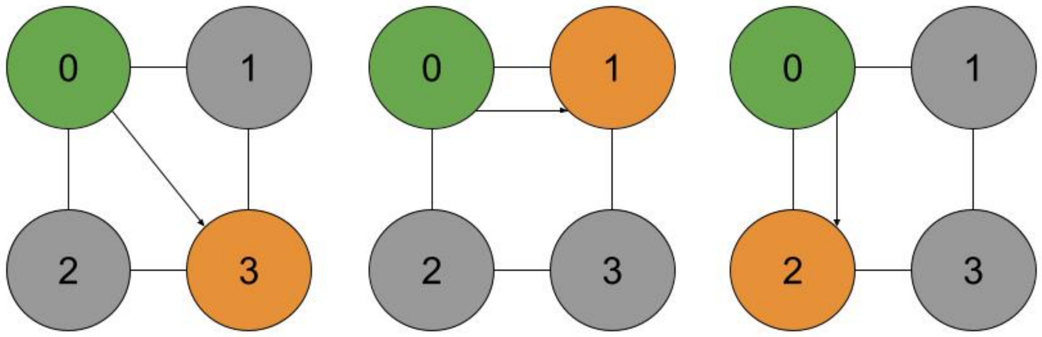


Figure 12. Example of the first step in a four-node TSP. (Left) Initial Exploration 0-3. (Middle) Initial Exploration 0-1. (Right) Initial Exploration 0-2.

The next step is where storing these subproblems begins. In particular, there are two central pieces of information: the set of visited nodes in the current subpath, and the index of the last node visited. The set of visited nodes in the current subpath are saved for the program to save time calculating the subpath in the future, and the index is needed for the program to know from which node to extend to form future pieces of the path. These two pieces of information together form the state of the UAV.

Moving on, the DP enters the expanding stage during which the program takes the previous subpath saved and extends the path to a new, unvisited node. Every time the DP extends to an unvisited node, it will continue to update and save the new set of visited nodes and the index and recursively branch out until all combinations are explored. Continuing from one of the subsets of the four-node TSP, the visual example in Figure 13 illustrates the expansion stage of the DP as it branches out to explore new, unvisited nodes in the problem. The final step is to return to the starting node. This is achieved by looping the end state and minimizing the lookup value and cost of returning from every possible end position.

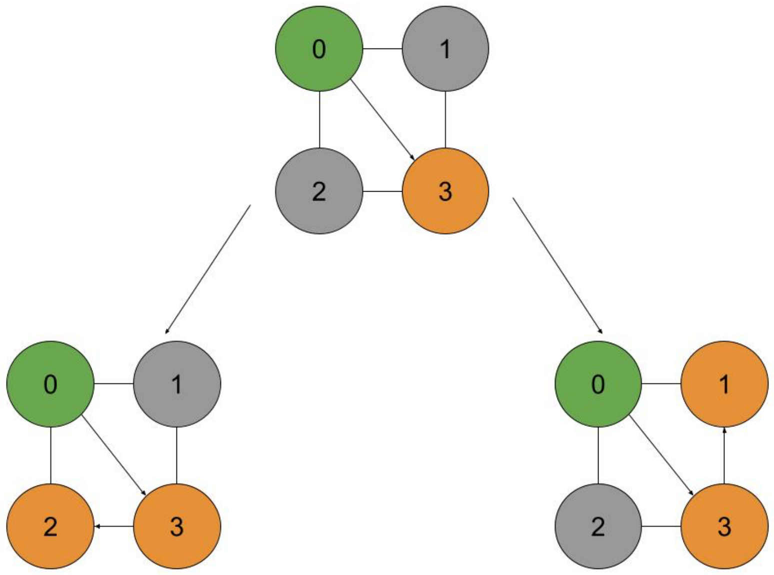


Figure 13. Expansion stage following the initial 0-3 case in a 4 node TSP. (Left) Branch out 0-3-2 (Right) Branch out 0-3-1.

In this research, the TSP was represented as a graph with weighted edges, or as an adjacency matrix. The set of visited nodes was represented as a single 32-bit integer, which is both compact, quick, and allows for easy and efficient caching. More specifically, inside the binary representation, when node i is visited, the i-th (the index of the bit) is flipped from 0 (unvisited node) to 1 (visited node). For example, in the left diagram in Figure 12 in which a path starts from node 0 and travels to node 3, the binary representation of the visited nodes would be 1001, with the last visited node being 3. Just like in the example, these two pieces of information are saved when the DP begins storing the subproblems.

Results

As seen in Table 1, whereas the DP performs well with a lower number of sites, the compute time quickly scales up for more than 20 sites, more than doubling the time needed per site added. For example, at 24 nodes in the TSP, the program takes more than 60 seconds to compute the path. In addition, DP was also found to be limited by its space complexity, which is derived from the values that the algorithm stores during its expansion. Since there are n nodes that could have been visited last and 2^n possible subsets of visited nodes, the space complexity of the DP is $O(n2^n)$ [16], meaning that with each node added to the TSP, there is an exponential growth in the amount of total space the algorithm takes up.

Number of Sites	5	10	20	21	22	23	24	25
Compute Times (ms)	38	51	1760	4213	10226	23418	67611	Java OutOfMemory Exception

Table 1. DP compute times.

Discussion

Dynamic Programming is a powerful tool for optimization and has been shown to be applicable to the TSP. One of the major benefits to Dynamic Programming is its accuracy. Unlike some other methodologies such as Genetic Algorithm and Simulated Annealing, DP guarantees optimality no matter how many sites or nodes are added. However, it has major drawbacks in terms of runtime duration and space complexity. Although an exponential runtime is a significant improvement compared to Exhaustive Search, DP is relatively inefficient compared to methodologies like GA and SA, which are still viable even with thousands of sites. DP scaling was found to increasingly worsen in terms of both runtime and space complexity (exponential as the number of sites increases).

V. Image Classification using Deep Learning for Wildfire Detection

In addition to investigating the use of Genetic Algorithm, Simulated Annealing, and Dynamic Programming to plan efficient flight paths for UAVs to traverse a set of locations for wildfire management, the second challenging problem investigated in this study was the identification of wildfires from deployed monitoring systems such as static optical camera networks.

The solution explored in this study was image classification using deep learning (DL) and transfer learning. The implementation of the model for this study can be found in the Jupyter notebook at https://github.com/iulia-iordanescu/NASA_VIP_2022_transfer_learning. The results of the model can be used to alert human operators of potential wildfires and help them prioritize the deployment of resources (such as UAVs flying efficient flight paths as described in the prior section). The results of this study can serve as a baseline for follow-up research such as using cameras on UAVs to more flexibly collect visual data to detect wildfires.

The dataset utilized in this study is from the public repository of "AI For Mankind," which is a non-profit focused on encouraging the development of AI applications for "wicked problems" in the world, including automatic wildfire detection [17]. The dataset contains 2622 no-smoke images and 999 smoke images from two camera systems—AlertWildfire and High Performance Wireless Research and Education Network—installed around mountaintops in California.

Background

Neural networks (NNs) are a subset of machine learning algorithms. They are a sequence of layers of artificial neurons (basic unit of NNs). There are three types of layers: input layer, hidden layers (large number for DL), and output layer. A neuron loosely models a biological brain neuron. They have weights, which decide how much influence the input will have on the output of that neuron. Neurons are named from their biological counterparts because neural networks learn by adjusting these weights to improve classification performance. The learning process is called training. The weights are adjusted during the training process to minimize the difference between the predicted labels and the true label. In this study, the label is whether an image has smoke or not.

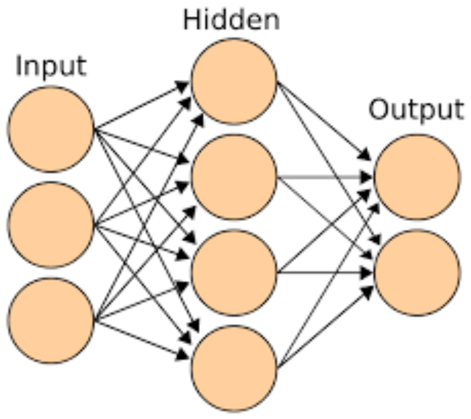


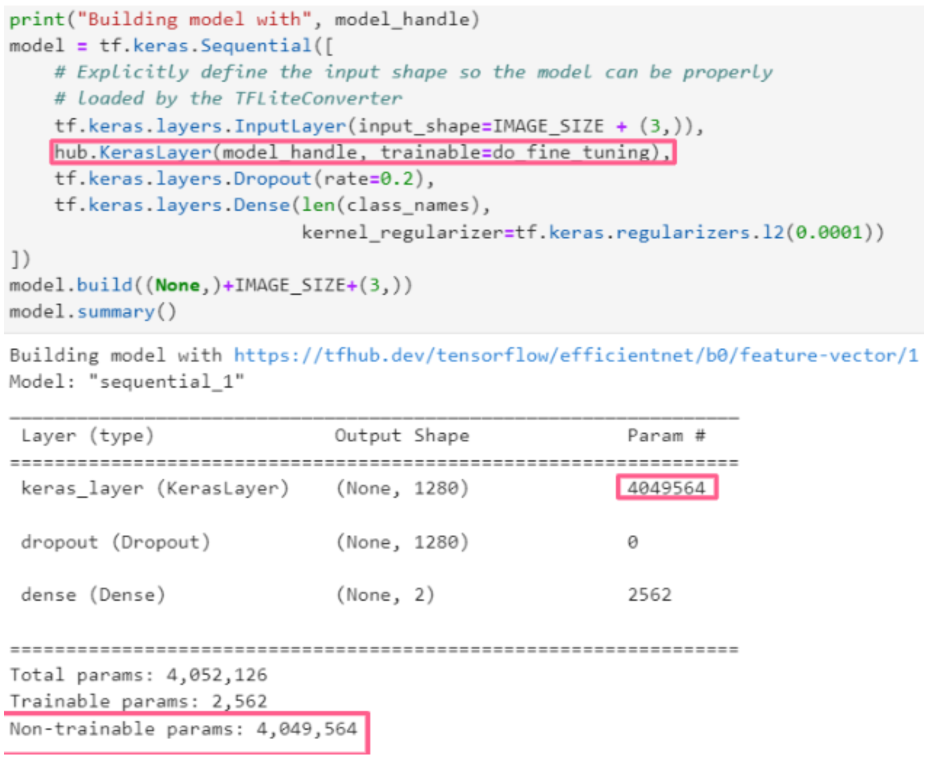
Figure 14. Example of a fully connected shallow neural network with just one hidden layer [18].

Convolutional Neural Networks (CNN), a type of neural network typically used for image processing, was used for the image classification model in this study. In CNNs, the neurons in the hidden layers are not connected to every single pixel in the input image, but rather to a subset (e.g., a three-by-three window). This helps a neuron specialize and focus on a specific area. In this study, the weights of the neurons in a CNN layer are forced to be equal (called parameter sharing) so that the layer “learns” a basic image processing filter. This process is equivalent to doing low-level image processing like edge detection via Sobel operator. Compared to shallow neural networks which have just one hidden layer, deep learning NNs have many layers and are needed in cases with subtle features that are challenging to define specifically. As processing happens at higher and higher levels, the model eventually aggregates features for what is relevant to classifying the image. The only input into the model are the pixels of the image.

Transfer learning is utilized to speed up training time to create the wildfire detection model using the relatively small dataset available (detailed in the next section). In transfer learning, a model developed for another task in the same domain is reused as the starting point for the current model. The advantage of doing this is that the current task requires a smaller dataset because the pretrained model is already effective at extracting features. The closer to the input, the lower the level of image processing that is performed by the layers, which is less task specific. In practice, the only layer that is task specific is the classification layer, which is the last layer and discarded. Everything else is kept to perform a highly discriminative image encoding.

Methodology

TensorFlow Hub [19], a repository with many collections of pre-trained models, was leveraged in this study for transfer learning. The EfficientNet-B0 model [20], from the EfficientNet family [21], was chosen because it requires a smaller input image size (224 x 224) that was similar to the image size (227 x 170) in the “AI For Mankind” wildfire detection dataset to minimize the noise generated during the resizing process, which uses interpolation. As shown in Figure 15, the model developed in this study incorporates the pre-trained EfficientNet B0 model as the second “layer” with four million parameters that are fixed and thus not changed during training, followed by a dropout layer to minimize overfitting, and a dense output layer for class prediction.



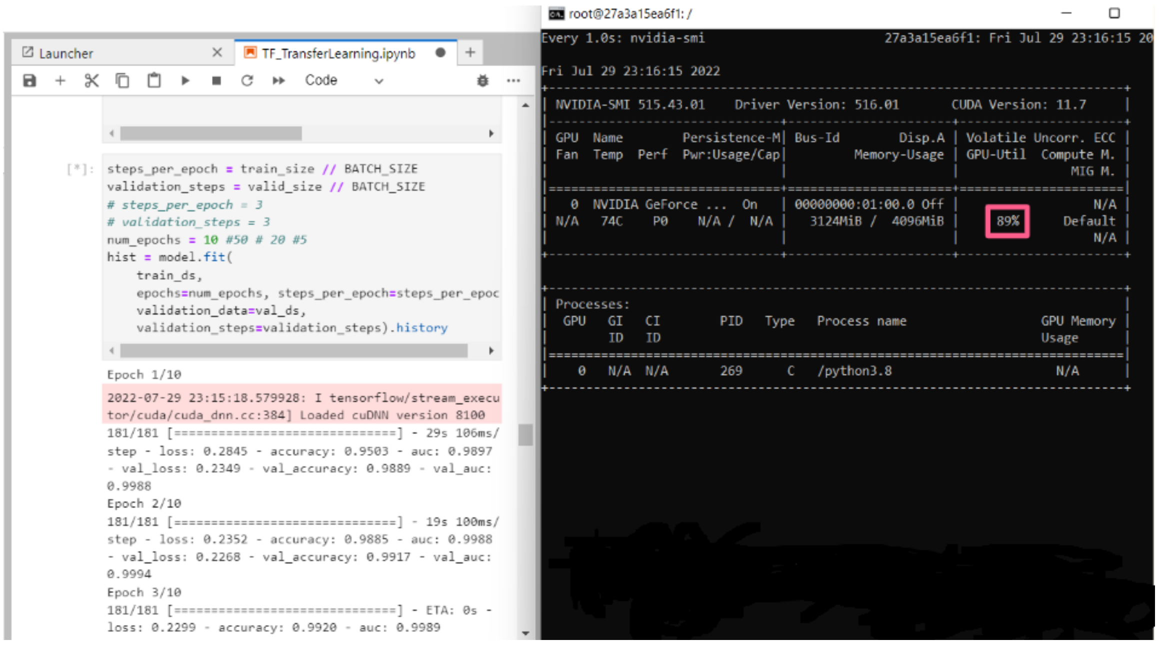
```
print("Building model with", model_handle)
model = tf.keras.Sequential([
    # Explicitly define the input shape so the model can be properly
    # loaded by the TFLiteConverter
    tf.keras.layers.InputLayer(input_shape=IMAGE_SIZE + (3,)),
    hub.KerasLayer(model_handle, trainable=do_fine_tuning),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(len(class_names),
                          kernel_regularizer=tf.keras.regularizers.l2(0.0001))
])
model.build((None,)+IMAGE_SIZE+(3,))
model.summary()
```

```
Building model with https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              4049564

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 4,052,126
Trainable params: 2,562
Non-trainable params: 4,049,564
```

Figure 15. Structure of the DL model for wildfire detection leveraging transfer learning.

GPU-Powered Training

Tensorflow [22] is a modern machine learning framework that leverages GPU processing without having to change the code. Thus, as shown in Figure 16, if the GPU is available on the machine where the model is trained, Tensoflow will recognize it and use it to off-load heavy computations like multiplication to increase the training speed, which in this case was by almost an order of magnitude.



```
[*]: steps_per_epoch = train_size // BATCH_SIZE
     validation_steps = valid_size // BATCH_SIZE
     # steps_per_epoch = 3
     # validation_steps = 3
     num_epochs = 10 #50 # 20 #5
     hist = model.fit(
         train_ds,
         epochs=num_epochs, steps_per_epoch=steps_per_epoc
         validation_data=val_ds,
         validation_steps=validation_steps).history
```

```
Epoch 1/10
2022-07-29 23:15:18.579928: I tensorflow/stream_execu
tor/cuda/cuda_dnn.cc:384] Loaded cuDNN version 8100
181/181 [==============================] - 29s 106ms/
step - loss: 0.2845 - accuracy: 0.9503 - auc: 0.9897
- val_loss: 0.2349 - val_accuracy: 0.9889 - val_auc:
0.9988
Epoch 2/10
181/181 [==============================] - 19s 100ms/
step - loss: 0.2352 - accuracy: 0.9885 - auc: 0.9988
- val_loss: 0.2268 - val_accuracy: 0.9917 - val_auc:
0.9994
Epoch 3/10
181/181 [==============================] - ETA: 0s -
loss: 0.2299 - accuracy: 0.9920 - auc: 0.9989
```

```
Every 1.0s: nvidia-smi                          27a3a15ea6f1: Fri Jul 29 23:16:15 20

Fri Jul 29 23:16:15 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.43.01    Driver Version: 516.01       CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  On   | 00000000:01:00.0 Off |                  N/A |
| N/A   74C    P0    N/A /  N/A |   3124MiB /  4096MiB |     89%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A       269      C   /python3.8                      N/A      |
+-----------------------------------------------------------------------------+
```

[redacted]

Figure 16. Screenshot taken during training session on 10 epochs in the notebook (left) showing GPU utilization at 89% (right).

For training, the Categorical_Crossentropy loss function and stochastic gradient descent optimizer (learning_rate = 0.005) was used. Accuracy and AUC (Area Under the receiver operating characteristic Curve) was monitored for both training and validation data. Categorical cross entropy is a standard loss function for classification models that computes the cross entropy between the true labels and predicted probabilities (for labels) by taking the negative-log of the predicted probability for the true class. It punishes the model when it is confidently wrong by giving a large mismatch value. As illustrated in Figure 17, the loss function during training (left) does not improve much because it already starts low. As expected, AUC (right) rapidly improves for the training data because of the transfer learning approach.

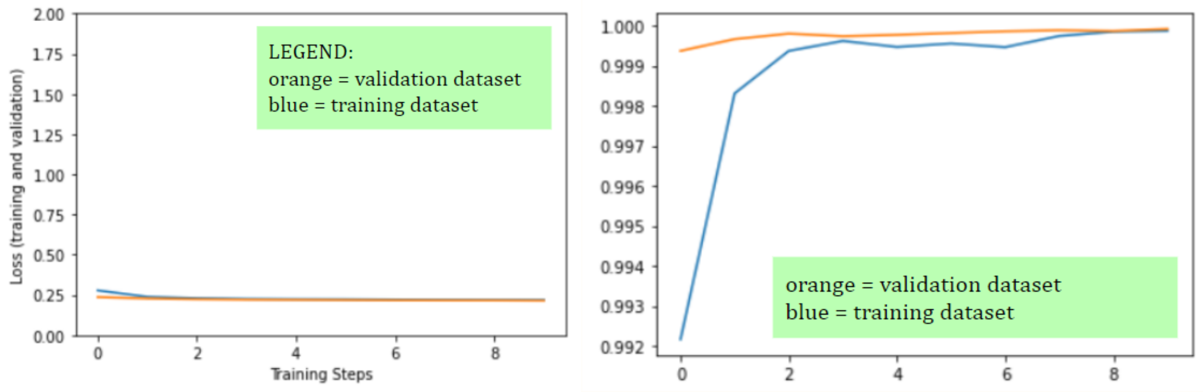


Figure 17. (Left) Loss function during training. (Right) AUC.

Results

Figure 18 is the confusion matrix for the wildfire detection model on a random sample of 500 images from the validation dataset. It shows excellent classification performance, with only two false negatives and no false positives. This is likely due to the static nature of the images, which captured both images with fire and with no fire at the same angle, so the model is able to leverage the highly discriminative image encodings from the pretrained model. Figure 19 are examples of validation images without smoke (left) and with smoke (right) that were correctly classified by the wildfire detection model.

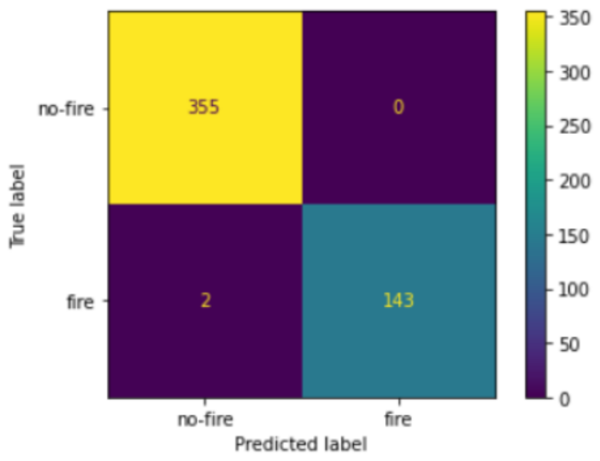
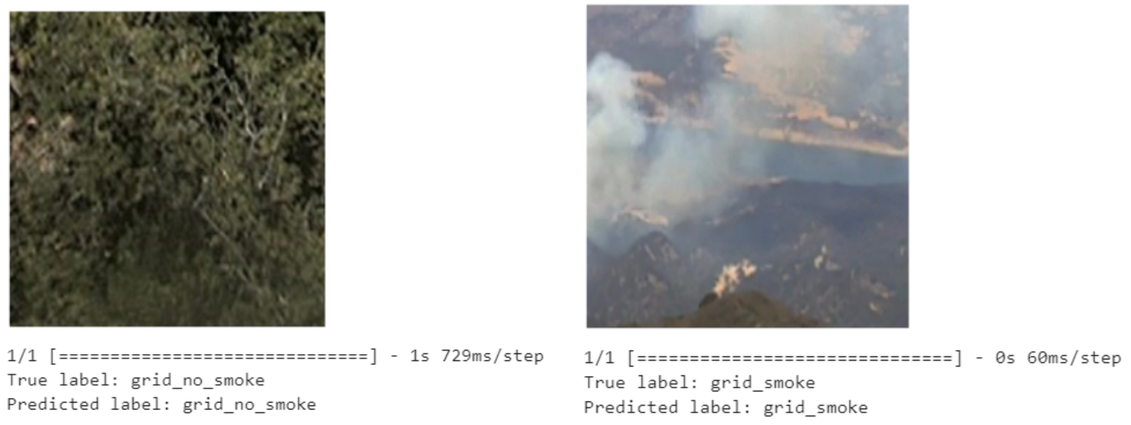


Figure 18. Confusion matrix for the wildfire detection model on a random sample of 500 validation images.



```
1/1 [==============================] - 1s 729ms/step
True label: grid_no_smoke
Predicted label: grid_no_smoke
```

```
1/1 [==============================] - 0s 60ms/step
True label: grid_smoke
Predicted label: grid_smoke
```

Figure 19. Examples of validation images ("True label") that were classified ("Predicted label") by the wildfire detection model. (Left) No smoke. (Right) With smoke.

This confirms the results from the training session when AUC was 1, which shows that this problem is easy for the model. For other datasets, correctly classifying images may not be as easy. Work is underway on a larger dataset with higher-resolution images, specifically 3072 x 2048 compared to 227 x 170 for the results shown in this exploratory study.

VI. Discussion

This study explored several machine learning techniques—genetic algorithm, simulated annealing, and dynamic programming—to solve the TSP to find the shortest path to visit each location in a given set of locations exactly once and return to the starting location, which is a good representation of the mission to perform wildfire monitoring and surveillance. In addition, image classification using deep learning with a convolutional neural network to detect wildfires in static images was explored, including the use of transfer learning to efficiently train deep learning models and the use of GPU processing to increase training speed.

Path Planning

For each of the machine learning techniques explored to solve the TSP, it was observed that accuracy and processing time were affected by the selection of algorithm hyperparameter values. It is important to carefully select the values of these hyperparameters for the type and size of the problem. In this study, a manual grid search was performed. In future work, though, automated hyperparameter optimization approaches such as Bayesian optimization and gradient-based optimization should be explored to more optimally solve the problem.

Image Classification

Initial results for image classification using deep learning for wildfire detection are promising. However, the problem investigated in this study may be easy because fixed cameras always have the same perspective and, thus, wildfires may be easy to detect by contrasting images from the same camera. Future work should be performed to extend the work in this study from static camera images to dynamic images captured by a moving UAV, which will have a larger range of perspectives. If successful, this would allow UAVs to be utilized for fire detection and, thus, enable expanded and more flexible coverage of areas of interest. In addition, utilizing more sophisticated objection detection frameworks such as PyTorch [23], Meta AI's Detectron2 [24], and Tensorflow Object Detection API [25] should be investigated and compared to the simple transfer learning approach used in this study.

UAV Features

It is important to minimize the weight of UAVs for wildfire detection to maximize flying time, reduce inertia, and improve maneuverability. This can be achieved by utilizing low-density materials such as carbon fiber-reinforced composites, thermoplastics (polyester, nylon, and polystyrene), and aluminum.

These UAVs would be equipped with cameras and sensors to detect wildfires—ideally, before the wildfires are fully developed—and monitor progression. As soon as a wildfire is detected (e.g., using image classification as described in Section V), the corresponding GPS coordinates will be sent to the wildfire authorities.

Recommendations

Wildfire Management

Given the substantial potential benefits seen in this exploratory study, it would be valuable to explore other applications of ML/AI for wildfire management using UAVs. This could include but is not limited to expanding communications capabilities and monitoring air quality in the vicinity of firefighters. Furthermore, the exploratory work performed in this study for path planning and image classification could be extended and optimized to more effectively monitor and detect wildfires.

Other Wicked Problems

ML/AI has great potential for wicked problems in other areas like ocean sustainability, health and well-being, and responsible consumption and production. Within ocean sustainability, fishing surveillance could be utilized to prevent overfishing, and eutrophication (excess nutrients due to run-off) cleanup would help to de-pollute the water. In addition, UAVs guided by ML/AI have the potential to help in first aid response and search-and-rescue missions, which would benefit the health and well-being of those in need. Proper cleanup and recycling as well as the ability to safely manage electronic waste are opportunities within the goal of responsible consumption and production.

VII. Conclusion

Wildfires are a growing severe problem that has resulted in substantial losses that may have been prevented. To address the problem of efficiently monitoring multiple sites in a large area, several path planning algorithms were explored and a wildfire image detection method was developed in this exploratory study.

It was found that Genetic Algorithm is an effective approach to solve the Traveling Salesman Problem to plan a near-optimal UAV path across a 25-location system. Simulated Annealing was found to be another effective approach, with an important feature that it avoids being trapped in local minima. Both Genetic Algorithm and Simulated Annealing do not guarantee optimality, though. Dynamic Programming was also found to be an effective method for solving the TSP that can reduce runtime while maintaining accuracy. Although it always finds the optimal solution unlike Genetic Algorithm and Simulated Annealing, it does not scale as well for TSP with more than a couple dozen locations.

In the research on image classification using deep learning that was performed in this study, it was found that wildfires can be detected from deployed monitoring systems such as static optical camera networks. Transfer learning was found to be an effective technique to efficiently train the deep learning model in this study. It was also found that GPU processing significantly increased training speed (by almost an order of magnitude in this study). These results provide a baseline for follow-up research to utilize images captured by UAVs in flight for wildfire detection.

References

[1] Mercury Insurance, “How Wildfires Start and Spread,” Mar. 2022. https://www.mercuryinsurance.com/resources/weather/how-wildfires-start-and-spread.html#:~:text=worth%20of%20destruction.-,How%20fast%20do%20wildfires%20spread%3F,destroying%20everything%20in%20its%20path

[2] Joosse, T., “Human-sparked wildfires are more destructive than those caused by nature,” Dec. 2020. https://www.science.org/content/article/human-sparked-wildfires-are-more-destructive-those-caused-nature#:~:text=Studies%20have%20shown%20human%20ignition,all%20those%20that%20threaten%20homes

[3] “Travelling Salesman Problem,” https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/

[4] U.S. Fire Administration," Wildland Urban Interface: A Look at Issues and Resolutions," Jun 2022. https://www.usfa.fema.gov/downloads/pdf/publications/wui-issues-resolutions-report.pdf

[5] Barrett, K. , "Wildfires destroy thousands of structures each year," https://headwaterseconomics.org/natural-hazards/structures-destroyed-by-wildfire/

[6] U.S. Forest Service, "Prescribed Fire," https://www.usda.gov/media/blog/2019/06/27/wildfires-all-seasons

[7] New York State Department of Health, "Exposure to Smoke from Fires," https://health.ny.gov/environmental/outdoors/air/smoke_from_fire.htm#:~:text=All%20smoke%20contains%20carbon%20monoxide,%2C%20styrene%2C%20metals%20and%20dioxins.

[8] TriData Corporation, "NIOSH Firefighter Radio Communications. Chapter III: Firefighter Communications Issues," https://www.cdc.gov/niosh/fire/pdfs/ffrcsch3.pdf

[9] van der Feyst, M., "Firefighter Training Drill: Face Piece Communications," https://www.fireengineering.com/firefighter-training/firefighter-training-drill-face-piece-communications/#gref

[10] NASA, "What is Scalable Traffic Management for Emergency Response Operations?" https://www.nasa.gov/ames/stereo

[11] U.S. Forest Service, "What does "mop up" mean?" https://inciweb.nwcg.gov/incident/article/7013/54854/

[12] Mallawaarachchi, V., "Introduction to Genetic Algorithms — Including Example Code," https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[13] https://github.com/guofei9987/scikit-opt/blob/master/examples/demo_ga_gpu.py

[14] Hansen, P.B., "Simulated Annealing," *Electrical Engineering and Computer Science - Technical Reports*, 170, 1992. https://surface.syr.edu/eecs_techreports/170/

[15] M, Prajwal, "Demystifying Dynamic Programming" https://www.freecodecamp.org/news/demystifying-dynamic-programming-3efafb8d4296/

[16] "Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming)" https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/

[17] AI For Mankind, "Let's Stop Wildfire Hackathon," https://github.com/aiformankind/lets-stop-wildfires-hackathon/blob/master/Challenge_1B_WildfireSmokeImageClassifierForDemo.ipynb

[18] " Artificial neural network" https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

[19] TensorFlow Hub, https://tfhub.dev/

[20] "EfficientNet-B0," https://tfhub.dev/tensorflow/efficientnet/b0/classification/1

[21] “EfficientNet,” https://tfhub.dev/google/collections/efficientnet/1

[22] TensorFlow, https://www.tensorflow.org/

[23] PyTorch, https://pytorch.org/

[24] Meta AI’s Detectron2, https://ai.facebook.com/tools/detectron2/

[25] Tensorflow Object Detection API,
https://github.com/tensorflow/models/tree/master/research/object_detection