# PATH PLANNING ALGORITHM FOR MULTIPLE UNMANNED AERIAL VEHICLES WITH PARALLEL IMPLEMENTATION ON A GRAPHICS PROCESSING UNIT

# ALGORITHME DE PLANIFICATION DE TRAJECTOIRE POUR MULTIPLES DRONES AVEC IMPLÉMENTATION PARALLÈLE SUR PROCESSEUR GRAPHIQUE

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Shan Mufti, BASc.

In Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in Computer Engineering

April, 2019

# Acknowledgements

I would like to thank the staff and my peers at the Royal Military College of Canada for their support and for welcoming me into the community throughout my time here. In particular my supervisors Dr. Vincent Roberge and Dr. Mohammed Tarbouchi for their excellent mentorship and guidance. I thank you for taking me on as a student and for the opportunities that you have provided me.

I am forever grateful for my parents and the unconditional support they have given me throughout both my life and educational career. I would not be where I am today if it was not for them. Lastly, a thanks to my sister, being someone I have always been able to rely on, and to keep reminding me that anything is possible.

# Abstract

The usage of unmanned aerial vehicles (UAVs) have been growing at increasing rates and primarily for the execution of surveillance and reconnaissance tasks. Motivations for their use pertain to their versatility, low cost, elimination of human risk, and potential autonomous capabilities. Intelligence, surveillance and reconnaissance (ISR) tasks for UAVs generally require the aircraft to overfly specified points of interest in an efficient manner while avoiding terrain and dangerous regions to gather data and information. A step towards autonomous UAV's is a path planning module, capable of solving the complex optimization problem of computing the routing in a reliable and timely manner. This speed is essential to allow for rapid flight path updating due to variabilities in the environment and changing objectives. Deploying multiple UAVs on a mission allows for faster objective completion and provides a higher level of redundancy; however, adds complexity as the problem of resource allocation and coordination must be considered. This thesis work investigates a fast flight planning algorithm for an ISR scenario involving multiple UAV's over a known geographical area. A path planning algorithm was developed, it consists of the setup and formatting of data, solving the routing problem for each POI using Bellman-Ford, and the distribution and assignment of the appropriate paths for multiple UAV's using the Genetic Algorithm. The acceleration of this process was achieved using a Graphics Processing Unit and allowed for significant speed-up enabling rapid path planning and in-mission recalculations. The path planning program was tested and verified on a variety of real-world terrain scenario's simulating an ISR type mission with varying number of UAVs.

# Résumé

L'utilisation de véhicules aériens sans pilote (UAV) augmente de plus en plus rapidement et sert principalement à l'exécution de tâches de surveillance et de reconnaissance. Leur utilisation est motivée par leur polyvalence, leur faible coût, l'élimination des risques pour l'homme et leur autonomie. Les tâches de renseignement, de surveillance et de reconnaissance (ISR) pour les UAV requièrt de survoler des points d'intérêt spécifiés de manière efficace tout en évitant les terrains et les régions dangereuses pour collecter des données et de l'information. En marche vers leur autonomie, les UAV doivent disposer d'un module de planification de trajectoire capable de résoudre le problème complexe d'optimisation de calcul de manière robuste et rapide. Cela permet une mise-à-jour rapide de la trajectoire de vol en raison de la variabilité de l'environnement et des objectifs changeants. Une complexité supplémentaire est introduite lors d'un scénario où plusieurs UAV sont déployés. Cela permet d'accomplir les objectifs plus rapidement et offre un niveau de redondance plus élevé, mais l'affectation des ressources et la coordination des aéronefs doivent être prise en compte. Cette thèse présente une solution de planification de vol pour un scénario ISR où plusieurs UAV survolent une zone géographique donnée. Cela a été accompli grâce à la configuration et au formatage des données d'entrée, à la résolution du problème du chemin le plus court à source unique pour chaque point d'intérêt utilisant Bellman-Ford, ainsi qu'à la distribution et à l'attribution des chemins appropriés pour plusieurs UAV utilisant la métaheuristique de l'algorithme génétique. L'accélération de ce processus, obtenue à l'aide d'un processeur graphique, a produit un gain significatif permettant la planification et le re calcul des trajectoires de manière très rapide. De plus, le programme de planification de trajectoire a été testé et vérifié pour divers scénarios représentant un terrain réel et simulant une mission de type ISR avec un nombre variable d'UAV.

# Contents

# List of Figures

# List of Acronyms

**ACO** Ant Colony Optimization

**BF** Bellman Ford

**CUDA** Compute Unified Device Architecture

**FLOPS** Floating Point Operations per Second

**GA** Genetic Algorithm

**GPGPU** General Purpose Computing on Graphics Processing Units

**GPU** Graphics Processing Unit

**m-TSP** Multiple Travelling Salesman Problem

**POI** Point of Interest

**POMDPs** Partially Observable Markov Decision Processes

**PSO** Particle Swarm Optimization

**SM** Streaming Multiprocessor

**SSSP** Single Source Shortest Point

**TSP** Travelling Salesman Problem

**WPS** Wolf Pack Search

**UAV** Unmanned Aerial Vehicle

# 1 Introduction

UAV is an acronym for Unmanned Aerial Vehicle, which are versatile aircraft platforms used in a variety of applications and has been experiencing significant growths in the military and commercial sectors. Usages vary from performing tasks in intelligence, surveillance and reconnaissance (ISR), targeting, seeding in farms, weather forecasting, and hazardous environment monitoring. A significant benefit of employing a UAV is the lower costs compared to a crewed aircraft, risk mitigation due to the elimination of the onboard pilot and the development of autonomous features.

Due to these lowering costs, deploying multiple UAVs (multi-UAVs) in the same environment to perform a given set of objectives has become a popular choice, providing a further increase in efficiency and redundancy. Multi UAV deployment also allows for larger scale operations in terms of area coverage and the scale of mission environments. More recent developments in technology have brought to light the ability of UAVs to perform tasks requiring lesser amounts of human intervention and interaction, with the eventual goal to have fully autonomous flight capability. A vital step to achieve this is the path planning module, which is the system capable of generating a trajectory for the UAV control system to follow throughout the flight.

## 1.1 UAVs and Path Planning

UAVs form a subset of the unmanned systems category, classified as either a remotely piloted aircraft (RPV), generally flown by a pilot located at a ground control station, or an autonomous platform, control coming from an

autonomous based system usually from a pre-programmed flight plan or may consist of more complex dynamic systems. Additionally, the concept of a Unmanned Aircraft System (UAS) has emerged as in many cases an entire system is required to adequately support the UAV(s). These systems may include ground station aspects such as communication systems, remote pilot stations, and centralized computational resources.

The military role of UAVs have been growing at unprecedented rates, rapid advances in technology are enabling more and more capability to be located on smaller airframes which are spurring a substantial increase in the number of deployments on the battlefield [1]. Research and development in this domain have enabled UAV usage in a multitude of mission categories; however, the ISR mission remains to date the most popular. These UAVs range from thousands to millions of dollars in cost and range in size from a Micro Air Vehicle (MAV) weighing less than one pound to large aircraft over 40,000 pounds.

Even with the development of new capabilities, a diverse collection of UAVs have already been deployed in a variety of mission scenarios. Some of the current bounds of UAV operation are demonstrated by the vehicles shown in Figures 1.1.1 - 1.1.4. The Northrop Grumman RQ-4 Block 10 Global Hawk (Figure 1.1.1) is one of the largest UAVs currently in deployment, with a mass of 12,000 kilograms and a 35-meter wingspan [2]. With a range of over 20,000 kilometers, the Global Hawk UAV can provide real-time imaging for broad mission-level intelligence, surveillance, and reconnaissance (ISR). Figure 1.1.3 is an image of a smaller, launch based fixed-wing UAV for smaller mission areas.

At the other end, more portable and smaller sized UAVs capable of vertical take-off and lift (VTOL) provide squad-level support in local and possibly cluttered environments. One such example is the Indago (Figure 1.1.4, developed by Lockheed Martin and already in operation which weighs in at approximately 2.2 kilograms and is capable of covering an area greater than 3 kilometers [4].

Some early UAVs were referred to as drones due to their lack of sophistication as they were no more than radio controlled aircraft being controlled by a human pilot at all times. More modern UAVs employ systems such as

Figure 1.1.1: The Northrop Grumman RQ-4 Global Hawk. An example of a UAV for long range, high altitude ISR missions [2].


Figure 1.1.2: The C170 Heron is a medium-altitude long-endurance UAV used in Afghanistan [2].

built-in controllers, autopilot, and even simple pre-scripted navigation functions such as waypoint following. Despite having such systems, a UAV cannot be considered autonomous. The field of air vehicle autonomy is an emerging field, primarily driven by the military to develop technology to assist in the modern day and ever-changing battlefield. Compared to the manufacturing of UAV flight hardware, the area of autonomy technology is considered to be a new and developing field. Due to this, autonomy has been and may continue to bottleneck future developments in the field of unmanned systems thus emphasized in terms of research and development.

Autonomy in the context of this field is the ability to make decisions during flight and would allow for mission completion without human intervention. A significant aspect of this is tied in with path planning and navigation, with

Figure 1.1.3: Lockheed Martin Stalker XE is an example of a launcher based UAV for military missions [3].

Figure 1.1.4: Lockheed Martin Indago is a UAV capable of VTOL for short range ISR type missions [4].

technological development mostly following a bottom-up approach correlating to control science instead of computer science. Similarly, autonomy has been and probably will continue to be considered an extension of the controls field [5]. In the foreseeable future, however, the two fields will merge to a much higher degree, and practitioners and researchers from both disciplines will work together to spawn rapid technological development in the area. To some extent, the ultimate goal in the development of autonomous technology is to replace the human pilot. It remains to be seen whether future events, the perception of this technology, and the political climate surrounding the use

of the technology, will be limiting factors in the usage of autonomy for UAV applications.

## 1.2 Motivation

UAV's benefit in terms of use in industrial and military applications due to their low operational costs, low risk, and increased flight time in comparison to crewed aircraft. By having no flight crew on-board, they can be used for long duration flights without fatigue limitations, and in dangerous environments that would otherwise be deemed unsafe to fly. UAV's are versatile platforms, capable of performing a variety of tasks such as surveillance, reconnaissance, targeting, weather forecasting, and hazardous environment monitoring amongst others. For the accomplishment of these tasks, the UAV must possess the ability to navigate and explore the area in which it operates. ISR platforms such as UAVs have been vital and continue to be a growing interest in the military domain. As UAVs do not have the burden of the physiological limitations of human pilots, they can be designed for maximized on-station times. The maximum flight duration of unmanned aerial vehicles varies widely. Internal combustion engine aircraft endurance depends strongly on the percentage of fuel burned as a fraction of total weight, and quadcopter designed rely on battery capacity and efficiency. In accomplishing an ISR mission, the UAV must be capable of overflying specified way-points, avoid obstacles, and do so in the most efficient manner possible. Demand for adequate ISR data to support ongoing military operations is significant as it can detect and provide early warning of enemy threats, also enables forces to increase effectiveness, coordination, and lethality.

Moreover, employing UAV's to work as a group in an ISR setting presents the opportunity for more excellent operational capability. Having multiple UAVs operate together in the same environment to execute a given set of objectives can increase the efficiency in a mission portfolio and decrease the amount of time as the UAVs can gather data from different specified locations. Additionally, it allows for more extensive operations in terms of the number of reachable way-points, or the amount of area to be surveyed. A layer of

redundancy is also added, as the ability to accomplish the objectives is still possible in the event one or multiple vehicles are lost.

Autonomous path planning is a sought after feature for UAVs due to several reasons. An obvious one is the reduction of human workload in terms of creating the appropriate flight plan for aircraft to follow which is generally a tedious task. Rapid calculation of flight paths for multiple UAVs in a scenario expands mission capabilities through live flight path updating. The incorporation of a flight path planning module in UAV systems is a crucial step towards providing UAVs the ability to accomplish ISR missions autonomously. This brings path planning to the forefront in terms of enabling autonomous flight operations [6].

The US Department of Defence Unmanned Systems Roadmap 2005-2030 details a vision for the development of all types of unmanned systems [7]. It is their goal to integrate manned and unmanned systems for optimal mission execution seamlessly. According to the timeline, UAVs will be able to perform a full range of mission tasks by 2030 including surveillance, counter air strikes, penetrating strikes, and airlifts. There will be a need for increased autonomy as these vehicles perform more complex tasks in dynamic environments. Automation will by no means eliminate the need for human involvement in the operation of the unmanned systems but does alter what this interaction entails. Operators will move into supervisory roles, monitoring and guiding the activities of the vehicle through computer interfaces rather than mechanically controlling its movements [8]. The trend of autonomous UAV capabilities has been exponentially growing based on the results found in Figure 1.2.1 of which past and future UAVs have been plotted based on the respective autonomous control level. These control levels range from the basic remote controlled UAV to a fully autonomous swarm (group) of UAVs being the final and highest level sought.

## 1.3 Path Planning and Scenarios

The ISR mission profile considered in this work was formulated as follows. The mission area would be known in terms of geographical and natural features,

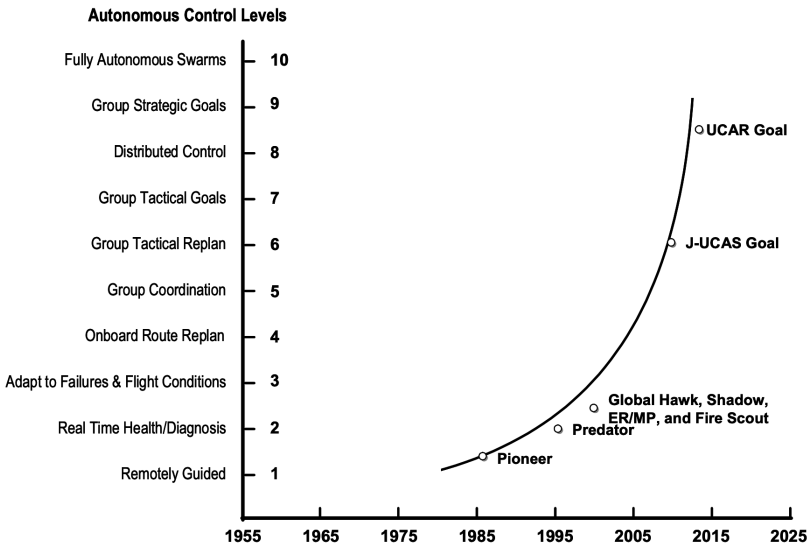


Figure 1.2.1: The trend in UAV autonomous capabilities. [8]

with a defined maximum altitude. There are multiple points of interest (POI) at locations specified by the user and must be directly overflown by at least one UAV to collect observation data. The positions of the POIs can be at various altitude levels above the given geographical terrain, limited to the mission area. A path consisting of waypoints must be generated for each autonomous UAV to define its routing between specified POIs. The generation of these paths would be through a path planning module, which was the focus of this work. In this scenario, multiple UAV's would be deployed at the same time from a single starting location of which they return upon reaching their assigned POI's. For this case, the path planner is a centralized system, calculating the flight paths for all UAV's with the ability to communicate and send updated flight paths if needed throughout the mission.

The most straightforward path would consist of straight lines of point to point routing passing through the POIs. This is inadequate as it is desirable to create more optimized routing for the sake of efficiency and to evaluate the path for safety. In path planning for multi-UAVs, besides optimizing and calculating the routing, the allocation of UAVs must be performed. A simple

example of a path planning scenario shown in Figure 1.3.1 consists of terrain features, a start position, and the desired POIs.

In a given path planning scenario there can be a variety of natural terrain and manufactured features. In ensuring safety, paths must remain free of collisions. Additionally, aspects such as altitude should be minimized to reduce the risk of detection by both enemy observers and radars. In performing this minimization, it is likely that the UAV path will want to follow the terrain closely. As the path closes in on terrain, it becomes imperative that the path has sufficient resolution and remains collision-free.

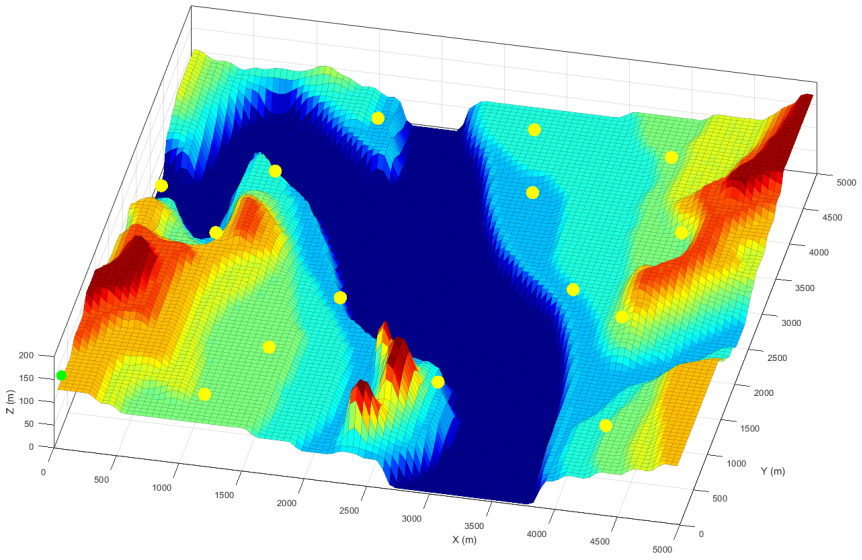


Figure 1.3.1: An example of a 3D environment used for path planning with the starting point located at the bottom left, and POIs located in yellow throughout.

Currently, a large amount of time spent during the operation of unmanned vehicles is directing, planning, and re-planning their desired route [9]. In a changing environment, the planned route may have to be recomputed numerous times hence acceleration and achieving fast runtimes is a topic of interest for researchers. Computing on GPU's has become increasingly popular especially when parallelism can be exploited. Their relatively high thread to cost

ratio allows for a massively parallel system capable of high data throughput and have proven effectiveness in running specific algorithms.

## 1.4 Research Aim

This research aimed to investigate and improve on the current state of multi-UAV path planning. A path planning method was proposed that provides higher quality paths with fast computation times, even with a high number of POIs and UAVs. The research undergone pertained to the usage of an exact algorithm, capable of evaluating the entire map for possible routing solutions. Additionally, it was proposed that the speed-up potential through the utilization of GPU parallel processing could enable fast runtimes. Fast in terms of computation time, allowing for in-mission recalculations, and by using an exact algorithm, it would allow for the reliable generation of solutions. The routing solutions would then be combined with the usage of a meta-heuristic algorithm to enable the deployment and optimization of multi-UAVs in the stated ISR mission.

Research was performed to investigate current path planning approaches and to determine areas of improvement. Such areas were identified as inadequate route optimization, as many solutions do not optimize for the entire routing path and difficulties in solving with a larger number of POIs and UAVs. In researching this hypothesized path planner, the desired output was first determined to be 3D paths for many UAVs directly overflying all specified POI's. The paths should be capable of following terrain and optimized considering the entire route and mission area. The development of a multi-objective cost function was undertaken for optimized path calculations and provided the ability to minimize a variety of parameters. With this information, the Bellman-Ford (BF) pathfinding method was selected for modification to suit the defined problem area. Selection of the BF was based on the algorithms parallel programming potential and reliability to generate a solution. Furthermore, the problem of solving for multi-UAVs involved developing an instance of the Genetic Algorithm to optimize the division and allocation of POIs based on the number of UAVs present.

The proposed planner was to be tested through simulations on a desktop computer using several scenarios on realistic and real-world 3D terrains. These simulations tested for a variety of input parameters and evaluated for runtimes, particularly in terms of achieved speed-up, and for the validity of the generated routes. To adequately demonstrate the program, real-world terrain maps of large areas were presented to ensure feasibility.

### 1.4.1 Constraints and Limitations

In formulating the problem space, various aspects were defined to constrain the bounds of this work. This path planner was to be presented a bounded and known area. The POI locations would be entered at the start of the program, with locations throughout the map space at various altitudes. The POIs were constrained to the resolution of the map, numbering from 1-25. The UAV type remained generic with its flight dynamics not being accounted for, of which general optimization parameters and values were used. This assumed that the UAV could turn and ascend at any rate. The number of UAVs in a scenario would vary from 1-10, under the assumption that they possess the adequate control systems to follow the paths outputted by this program.

### 1.4.2 Thesis Objective

This work contributes to the future development of autonomous UAVs, by expanding in the area of path planning in terms of quick calculations for in-mission use and providing a high-quality solution. The objective of this thesis was to assess the feasibility and speed-up of a centralized flight planning algorithm for multi-UAVs through parallel implementation on a GPU. It was to be shown that adequate speed-up of the BF algorithm can be achieved to generate routing between all the POIs. Additionally, these results were then to be utilized to assign the routes to each UAV adequately. The algorithm would be tested on several scenarios to assess its speed-up and characteristics.

## 1.5 Thesis Outline

The remainder of this document presents the research in greater detail. The next chapter consists of a literature review summarizing relevant past works in this research domain, followed by a background of the appropriate data structures, algorithms, and the GPU processing platform. Following that, the methodology is then presented, outlining the program implemented to test the path planning algorithm. The next chapter consists of the results and program validation. The final chapter contains the discussion with a statement of contributions, ending with the conclusion.

# 2 Literature Review

This chapter presents some of the more popular existing approaches and methods used in solving the UAV path planning problem. In surveying the field, it was found that both deterministic and non-deterministic algorithm types had been used, with non-deterministic tending to be a more popular choice. Map representation techniques are briefly assessed, followed by presenting previous works in solving the single UAV, and then the multi-UAV case.

## 2.1 Map Representation

### 2D & 3D Grid

In this representation, the input planning space was encoded as a 2D matrix in which the elements represent the elevation of the terrain [10]. This was adequately suited for the processing of certain algorithms with the ability to recreate the original map through the process of interpolation. The desired waypoints are overlaid independently of the planning space at elevations greater than the interpolated terrain height. Additionally, other features such as radar keep out zones can be included in this map representation format. A network of closely connected tetrahedrons with nodes was used to represent an entire 3D space. This representation style is known as Delaunay triangulation and can be used to quickly partition an input space[6]. With their method, threatening areas were embedded in the network in which the pathfinding algorithm can then avoid.

### Voronoi diagram

The input space can be converted to a Voronoi diagram to be processed by certain pathfinding algorithms. The Voronoi diagram partitions a plane of points into polygons containing exactly one point. The construction of Voronoi diagrams centers around an input with polygon edges initialized with a weight. It was successfully used to represent an environment with edge weights calculated accounting for fuel costs, threats, terrain, and distances in which a pathfinding algorithm was applied [11]. Voronoi diagrams were used to represent an urban environment [12] in which an automatic building detection method was used and enabled the use of 3D path planning.

## 2.2 Algorithms for Single UAV Path Planning

### Dijkstra's Algorithm

Dijkstra is a traditional graph-based pathfinding algorithm [13], in its original form has a computational complexity of $O(|E|+|V|^2)$. Where E represents the number of edges and V represented the number of vertices in the graph. For sparse graphs and a rearranged data structure this complexity can be reduced to $O(|E| + |V|log|V|)$ [14]. There have been adaptations of this algorithm for the use in UAV path planning [15], [16]. Palossi et. al. proposed a real-time 3D path planning algorithm utilizing Dijkstra's Single Source Shortest Path (SSSP) algorithm [17]. The enabling factor for it to be real-time was its implementation on a GPU; they showed the algorithm to be energy efficient hence applicable for UAVs with limited power resources. A year later in another publication, the same author expanded on the algorithm, [18] highlighting high GPU utilization, additional speed-up, and greater accuracy. It should be noted that this work was applied to a relatively small input space (under $100m^2$) indicative of substantial time increases or failure to solve in larger areas.

### A* Algorithm

The A* pathfinding algorithm has commonalities to Dijkstra, however, has a more heuristic approach [19]. This allows for a better selection of the next node queued in the search and enables the algorithm to reach the specified end node faster. This pathfinding algorithm has lower computational complexity than the Dijkstra, and in conjunction with a cost function, can be used for optimized path planning. Feng et. al. measures the performance of A* against Dijkstra's algorithm in the context of UAV path planning [20]. The A* method was shown to produce results faster than the previously stated algorithms. Their work was limited to solving in 2D domain taking into account obstacles with an unsmoothed path. They tested the algorithm solving for the path of more than one UAV and produced meaningful solutions through the minimization of distance traveled.

### Bellman Ford Algorithm

The Bellman Ford is a deterministic algorithm for the computation of the shortest path on a graph input [21]. A key difference in the BF is its ability to handle negative edge weight cycles. It, however, suffers from a higher computational complexity in comparison being $O(|E|\,|V|)$ [22]. Where again, E and V represented the number of edges and vertices in the graph.

The algorithm was adapted for use in UAV path planning with obstacle avoidance utilizing a multi-objective cost function [23]. The authors provided an extension allowing for 2D path planning with the consideration of a minimum turning radius while still directly overflying the waypoints. In a later publication is was shown that the Bellman-Ford algorithm to find the SSSP is a magnitude faster in terms of execution time when comparing a parallel implementation against a sequential version.

### Ant Colony Optimization

The ant colony optimization (ACO) was a common metaheuristic approach applied to the path planning problem. The algorithm was inspired by the

swarm intelligence seen in the behaviours of ants in the wild. In this case, a number of ants searching for the shortest path to the given waypoints. The ants travel between points based on pheromone lookup table and distances. Each ant completes a tour and updates the pheromone table to be fed to the next generation of ants. Generations of ants are continuously created and seek out paths using past knowledge to ideally find a more optimal path. This is repeated until a termination criterion is met. The ACO was first applied to the UAV path planning problem in 2D [24], building on their work, a more realistic path planner utilizing a hybrid ACO and differential evolution (DE) algorithm [25]. This approach produced an optimized, smoothened 3D trajectory minimizing fuel consumption while preserving the reliableness offered by the original ACO.

Building on the ACO approach, the authors of [26] use the multi-colony ACO algorithm for single UAV path planning. The ACO has been proven to be quite efficient, although it can be prone to stagnation or early convergence. By having multiple ant colonies communicating and working on the same problem each generation of ants can draw from a more comprehensive pheromone table enhancing the potential solution of a given generation. Their experimental results show their method to produce a more feasible solution for UAV path planning in comparison to the single colony ACO. Furthermore, it was shown that the ACO can be parallelized and experienced a speed-up of 82x in comparison to its sequential counterpart [27].

### Genetic Algorithm

There have been promising advances in the application of a parallelized version of the GA to solve the complex UAV pathfinding problem. This version of the traditional algorithm was developed to run on CUDA architecture and was shown to have considerable speed-up and good convergence [28]. The work, however, was limited to solving the 2D TSP in the context of UAV SSSP. The parallel GA was later applied to a 3D environment for single UAV path planning [29]. In their work, the results showed that the parallel version experienced a notable speed-up compared against a sequential version.

Additionally, there were minimal effects on GPU execution time with population size increase as opposed to almost exponential time increase with CPU. The most recent work found consisted of an all-encompassing solution by in which optimal, multi-objective path planning was achieved in a 3D environment for fixed-wing UAVs [30]. Their algorithm which provides a complete, smoothened path, gained an average 290x performance improvement over the sequential version. Another significance of their work is the introduction of a fitness function allowing for a higher quality solution, accounting for no-fly zones, obstacle detection, vertical speed limits, distance, fuel consumption. They also accelerate path smoothing on the GPU which is performed based on the GA solution. This work was promising due to having a 0.647 second execution time feasible for the use of in-mission path recalculation. However, this number was limited to cases in which only a low amount of POIs were solved for, and the algorithm was not suitable for use in complex terrain environments.

### Particle Swarm Optimization

The Particle Swarm Optimization (PSO) is a swarm optimization metaheuristic, it can be related to groups such as flocks of birds attempting to find an optimized solution [31]. The basis of the algorithm is to use a population of candidate solutions, called particles, and moving them toward the best-known positions in the search space based on the calculated results from each iteration. The movement of each particle is dependent on its assigned location and velocity which are influenced by the swarm's best-known position. This method was adapted for UAV path planning [32] in which terrain avoidance and threat evasion were successful. The authors of [33] compared the parallel GA and PSO UAV path planning algorithms both of which were multi-threaded on a CPU. The algorithms accounted for a multi-objective cost function in a 3D environment with path smoothing. The conclusion was that both algorithms experience speed-up through parallel implementations, however it was shown that the GA was more likely to produce a superior result than the PSO.

### Reinforcement Learning

Within the domain of machine learning exists a method known as reinforcement learning. An approach called adaptive and random exploration (ARE) with a Q-learning network was proposed as a solution for UAV path planning. They showed their method was adequate for path planning in a 2D environment with complex obstacles [34]. Limited work has been performed in this domain, although there is a possibility that reinforcement learning could be a viable option for the complex pathfinding problem. The drawback of these methods and similar reinforcement learning based methods are their significantly slower execution times in comparison to heuristics as there is a need for the system to undergo learning before its use.

## 2.3 Algorithms for Multi-UAV Path Planning

### Dijkstra's Algorithm

Representing the environment using a Voronoi diagram allowed for the SSSP to be calculated using Dijkstra's algorithm [12]. In their work, up to six UAVs were used to accomplish a reconnaissance task in the shortest amount of time. The path planning of multi-UAVs was through determining the minimum time taken by a given vehicle over those paths calculated. They recalculated paths based on an update in the environment however failed to indicate if the performance of the algorithm is suited for online path planning.

### Genetic Algorithm

Sahingoz applied the GA to multi UAV path planning and uses path smoothing to ensure flyable path based on UAV dynamics [35]. The same author then later produced a parallel version of this algorithm presented in [36]. Limitations of their work were the assumptions the UAV does not change altitude and flies at a constant speed, this makes it challenging to find an optimal solution given a multi-objective cost function.

More recently, Cekmez et al. implemented the parallel GA for multi-UAV path planning [37]. They used k-partitioning to partition the POIs into $K$ convenient parts to be assigned to a single UAV. This partitioning was sped up significantly by implementing it on the GPU. Next, the parallel GA was run on each partition separately solving for each UAV independently. Their solution resulted in up to 4.20x faster mission times by using four UAVs and executed from 185x to 223x faster than the sequential implementation. The path accounted for and avoided obstacles, however only was solved for a 2D input space optimizing just for distance. Additionally, the paths were only point-to-point between the POIs in which the route was not entirely evaluated. Their solution was not capable of solving adequately when presented with greater numbers of POIs or UAVs.

### Wolf Pack Search

The wolf pack search (WPS) algorithm is a swarm intelligence method in which was modified for use in 3D space for multiple UAVs [10]. The WPS has promise in this domain is due to strong convergence, the ability for global search, and parallel computing capability. The WPS works by performing the following steps: Initialization, Fitness (evaluation of the cost functions of entire pack), Elitism (selecting group from within the pack), Safari (small scale optimization), Update (move wolves closer to best wolf), Replacement (replace weak wolves in pack), and returns to fitness step until reaching a stopping criterion. Using a 2D matrix planning space, each POI was represented as a gene, and each wolf had a unique chromosome being an entire path.

The novelty of the work in this paper is the inclusion of the GA crossover and mutation operations to the original WPS algorithm. It was shown that this modified WPS algorithm retains all the beneficial qualities of the original while offering an improvement. The authors, through simulations, showed the modified WPS to be reliable and outperforms the original WPS, and GA by returning a more optimal solution. However, this algorithm takes significantly longer to run which would be a hindrance in online path planning and suffers similar traits as the GA in terms of larger problem sizes.

### Ant Colony Optimization

An improved ant colony optimization method was used to solve for multi-UAV path planning [11]; the approach used Voronoi graphs to allow for the optimization of fuel, and minimize threats based on the environment. They combined the usage of the ACO with k-degree smoothing to solve for the multi-UAV case. The work pertained more to multi-UAV coordination in which k-degree smoothing was used allowing the UAVs to arrive at the destination simultaneously or in an acceptable time interval. They produced feasible coordination with their method; however, it did not use any terrain collision avoidance evaluation.

### Reinforcement Learning

The Partially Observable Markov Decision Processes (POMDPs) was used for flight path planning for the use of target tracking [38]. Their work enabled the use of multiple UAVs in a dynamic environment and was able to maximize performance, and their calculated trajectory was constrained to the dynamics of the UAV. A policy-based Q-learning method was used; the policy was derived from Bellman's principle of optimality and incorporated coordination amongst multiple UAVs. Zhang et al. introduced a collaborative framework for multi-UAVs to produce a low-risk path [39]. Their method allowed for the collaboration of UAVs observations in path calculation as well as their locations to avoid mid-air collisions.

## 2.4 Consideration of UAV Dynamics

The result of most pathfinding algorithms is a point to point solution which is unlikely to be flyable by a fixed-wing UAV due to its minimum turning radius and flight dynamics. To modify the routing, a common approach was to apply a path smoothing algorithm on the point to point solution. After the application of path smoothing, the path may then be checked to ensure it does not violate any of the constraints such as collision with terrain. The modification

of the path also can result in the UAV no longer directly overflying the desired POI. Depending on the objectives of the flight this may not be acceptable.

The fuel consumption can be quite complex to calculate, formulas simplifying the equations of motion allow for faster computation of approximate but precise solutions [40].

Wind is one of the most substantial disturbances for aircraft during flight, and it can delay or speed-up flight times as well as veer aircraft off course if not corrected for. Weather such as thunderstorms must also be avoided as they are more than capable of causing loss of an aircraft, and at a more minor level cause disturbances in communication and sensor data. Weather is a changing environmental factor and flight planning modules must be capable of recalculating paths based on these changes. In this authors work, the Dubins path vehicle routing problem established for UAV path planning and was extended, accounting for steady state winds [41]. Another example was accounting for winds when planning a path for target tracking [38]. There have been seminal work regarding integrating this aspect in path planning algorithms.

## 2.5 Summary

There has been a significant amount of work published regarding autonomous UAV path planning for both the single and multi-UAV case; however, only a handful of solutions produce results capable of providing a complete path planning solution. The speed-up of the algorithm types used for path planning through parallel computing have been quite promising and continues to be a growing area of exploration in research. The significance of this is the application to in-mission path planning, which is a crucial milestone to allow autonomous UAVs to adapt to changing conditions. Throughout researching this topic, it seems that researchers tend to focus on specific aspects of the problem such as execution time, accounting for danger zones, problem representation, resource allocation with the tendency to miss one or more of these aspects. These limitations likely occur due to the complexity of the problem and the scale of the work needed to address all the possible problem areas.

Some publications, such as [10] stand out by offering an almost entirely complete solution in terms of a realistic and flyable path by multi-UAVs but is still limited by straight lines between POIs.

Many works, although showing promising execution time improvements should be further expanded on to provide a more thorough solution. This expansion could very well result in the executing time not being viable as initially indicated such as for increases in problem size.

Research seems to indicate that parallelized meta-heuristic algorithms are a promising approach to tackling the multi-UAV problem. However, it appears that these algorithms may not be suitable for cases where many POIs are present especially in conjunction with larger and more complex terrain. Additionally, there are gaps in the resource allocation and coordination aspect of employing multi-UAVs in an environment and the integration into the pathfinding algorithm used.

In the approach taken, the aim was to utilize the reliability of an exact algorithm, accelerate it via using the parallel processing capabilities of a GPU. Then use a meta-heuristic to converge on an optimized solution in terms of solving paths for multi-UAVs.

# 3 Background

This chapter provides the reader with information relevant to the path planning problem, optimization algorithms, and GPU programming. The path planning problem is further expanded on and defined for this work. The graph data structure will be presented due to its relation to pathfinding algorithms which in turn the Bellman-Ford (BF) algorithm will be presented. The genetic algorithm is then discussed. The chapter ends with the structure of GPU's in which the programming model and considerations are included.

## 3.1 Path Planning for UAVs

Path planning is a crucial element to any flight, and a functional path planner is essential to enabling autonomous UAV's. A path for UAV's should be provided in 3-dimensional space utilizing a multi-objective cost function to minimize parameters. A simple example of a path planning case for UAV's was illustrated earlier in chapter 1 Figure 1.3.1 in which consisted of the environment and contained multiple POIs at various altitudes. The result of a path planning algorithm should include the 3D trajectory necessary for the UAV to fly and reach a specified set of way-points. The resultant path must account for constraints such as obstacle avoidance, dynamic flight properties, borderlines, and mission goals. It allows the UAV to autonomously compute an optimal or near-optimal flight path between the initial point to the end by checking some specific control points or fulfil some mission specific constraints (path length, obstacle avoidance, fuel consumption, etc.). In the use of UAVs, generally, the path planning process tries to arrange a new path in 3D space

for checking a specific area or specific control points.

Path planning can be defined as determining an optimal path for the vehicle to go while meeting particular objectives and constraints, such as avoiding obstacles. For the case of an ISR mission as focused on in this work, the UAV's will all begin and finish their journey at a single starting location.

Task allocation and scheduling are determining the optimal distribution of tasks amongst a group of agents, with time and equipment constraints, with cooperative tactics being the formulation of an optimal sequence and spatial distribution of activities between agents to maximize the chance of success in any given mission scenario.

### 3.1.1 Path Planning Complexity

Planning an optimized route through a known environment is a computationally expensive task due to a variety of factors which have been previously presented. There are a variety of problem domains that involve path planning, such as routing for robotics and vehicles. These differ however due to the problem space and desired output, mainly due to the freedom of navigation an aircraft has in 3D space.

In the computation of a flight path, it is popular to optimize based on a multi-objective cost function. Based on numerous papers in the subject, standard parameters that have been deemed necessary to minimize include distance, average altitude, radar exposure, and fuel consumption. Throughout the flight, it is critical for the planned route to avoid any terrain or obstacles to avoid collision and the loss of a vehicle. In a changing environment, the planned route may have to be recomputed numerous times bringing important to the resources and time to execute this.

### 3.1.2 Relation to the Travelling Salesman Problem

The UAV path planning problem can be considered a subset of the traditional travelling salesman problem (TSP). The TSP is an NP-hard (non-deterministic, polynomial time, hard) optimization problem. The aim of TSP is merely to route a salesperson through several cities where the locations

are known. The route must be the shortest possible way to pass every given city exactly once and return to back to the starting point. In TSP solving, there are $(N-1)!/2$ single solutions where N is the number of cities. At least one of these solutions is the best one concerning its total distance. As the problem domain increases as does to computational requirements, with large scale problems causing the failure of many search algorithms to converge on a solution. With extensive data sets, and average computers will have severe difficulties solving through an exact algorithm. With such limitations, it is common to apply a heuristic approach to converge on a near-optimal result. Using the optimization algorithms focused on problems such as TSP can be seen as an effective way to study the UAV path planning problem. Additionally, the multiple travelling salesman problem (m-TSP) require more than one travelling body to be computed for and is often applied or related to multi-UAV path planning.

## 3.2 Graphs

Graphs are a versatile and powerful method of which is commonly used to represent a variety of data and situations. A graph contains two main data types; nodes that hold data, and edges representing connections between nodes and may also contain additional data. In an undirected graph, the edges are all bi-directional allowing for traversal to/from any connection, a directed graph has uni-directional edges allowing for one way traversal only. In most cases, including this work, the graph is weighted; containing weights representative of a cost associated with edges between the nodes. The method of graph representation and implementation may vary widely based on the problem domain and decisions made by the developer, and the intended uses of the graph. In graph implementation, it is usually desirable to have the ability to add/delete nodes and edges, along with modifying the weight values. Some methods of graph representation are summarized below. [42]

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph if a valid edge connection exists between two nodes, its respective index will contain a value. For example, a value of 1 at row $i$, column

$j$ would indicate a connection between node $i$ and $j$. In the case of a weighted graph, this value would represent the edge weight of the connection. It can also be noted that the adjacency matrix for undirected graphs are always symmetric. This method is relatively simple to implement and allows for easy and efficient edge modification with queries taking only $O(1)$ time. This method, however, consumes significant amounts of memory $O(V^2)$ despite the connectivity of the graph, and for the case of a sparsely connected graph, most of the memory is wasted space. Adding a vertex is $O(V^2)$ time.

Adjacency List is a method in which an array of lists are used, each list containing the list of vertices adjacent its respective vertex. For the case of a weighted graph, the weights of edges can be represented as lists of pairs. This method has a better memory utilization as the size of the array is equivalent to the number of vertices.

Edge lists are a simple graph representation, being just an array of $|E|$, called an edge list. Each entry in this array contains two vertex numbers that the edges are incident to. For weighted graphs, an additional element is added representing the edge's weight. Since each edge contains just two or three numbers, the total space for an edge list is $O(E)$. Edge lists are simple, but if we want to find whether the graph contains a particular edge, we have to search through the edge list. If the edges appear in the edge list in no specific order, that's a linear search through $|E|$.

Representing graph structures as arrays are desired, particularly when utilizing GPU processing, this can be accomplished through a number of methods, usually based on the graph connectivity (the edge/node ratio). For this work, it was desired to use a lightweight, simple, graph representation. This was to allow for minimal memory overhead, a high number of nodes and edges, and fast indexing. Another critical advantage was due to this works defined problem space being fixed at the beginning of the program; the of nodes and edges could be fixed with their respective connections.

## 3.3 Optimization Algorithms

This thesis work's primary focus was the adaptation and acceleration of two existing algorithms to solve the defined problem. The Bellman-Ford, a single source shortest path (SSSP) finding algorithm was modified to suit the needs of map area processing. Further, an adaptation of the Genetic Algorithm was implemented to solve the task allocation and cooperative problem for multi-UAVs.

### 3.3.1 Bellman Ford

The Bellman-Ford algorithm in its original form, it computes a path based on a weighted graph input, the details of its original implementation can be found in [21]. The algorithm is a single source shortest paths solver for directed graphs defined $G = (V, E)$ with embedded edge lengths $l(u, v)\ \epsilon\ E$ and starting node $s\ \epsilon\ V$. The desired output is all of the shortest distance values $dist(u)$, $u$ being all the vertices reachable from $s$. These values are calculated through an iterative process run for $|V| - 1$ times, each iteration updating all edges $e\ \epsilon\ E$

The time complexity of this algorithm is $O(|V| * |E|)$, this has typically led to other methods, such as the Dijkstra's algorithm $O((E + V)\ log(V))$ to be employed. An advantage of the BF is its ability to handle negative edge weights, which causes algorithms such as Dijkstra to fail. It can also be noted that the BF will perform a check for each vertex, and is not dependent on the edge processing order. This allows for more straightforward implementation in a distributed computing environment such as on a GPU platform.

The pseudo-code of the original BF algorithm (Algorithm 1) details the implementation and steps performed throughout. The initialization takes place from line 1 - 5, the loop initializes two components per vertex. All the separately stored vertex shortest distance values are set to infinite, and an additional list containing the previous node based on the shortest path calculation is initialized. Additionally, the starting node ($s$) is set to 0. The following $for$ loop from line 6 - 12 consists of the iterative process, the critical operation is the edge relaxation found on line 8 and 9. Edge relaxation is performed on

each edge $V - 1$ times, which checks the cost of the edge and current vertex to the next location and updates with the lower value when necessary. This is done by checking the previous node $u$ weight and the edge length $l$ connecting it to the current node $v$, updating as necessary and adding the node to the array containing previous nodes in the path represented as $prev$.

**Algorithm 1** Bellman Ford Pseudo-code

```
1: for all u ϵ V do
2:     dist(u) = ∞
3:     prev(u) = nil
4: end for
5: dist(s) = 0
6:
7: for i = 0 to |V| − 1 do
8:     for all e ϵ E do
9:         if dist(u) + l(u, v) < dist(v) then
10:            dist(v) = dist(u) + l(u, v)
11:            prev(v) = u
12:        end if
13:    end for
14: end for
```

In this work, the algorithm was adapted for use in UAV path planning utilizing the data structure with the embedded cost function previously discussed. Primary reasons for its selection was its reliableness and effectiveness as a highly parallelizable SSSP algorithm [43].

### 3.3.2 Genetic Algorithm

The Genetic Algorithm is a meta-heuristic approach to the optimization of a given problem and is presented in Algorithm 2. Its inspiration was from the evolution process experienced in species over time. Similarly to the natural selection process experienced, the GA aims to improve a set of candidate solutions over the course of 'evolutions' [44]. The aim is to reach an optimized solution based on the set criteria employed, generally in the form of a calculation of a fitness function. In the usage of a meta-heuristic approach, and evaluation of the results can be performed by analyzing how well the potential

candidates converge to a solution. It is desirable to have a fast convergence in terms of runtime, and that the algorithm indeed does converge to a solution. Reasons for an unreliable GA are commonly due to inappropriate initialization in which convergence may never be reached [45].

The first step of the algorithm to initialize a group of chromosomes called the initial population each of which represents a possible solution. In our implementation, a two chromosome technique was used consisting of two arrays of length $n$ each being a list of POIs and UAVs. For scenarios containing POIs outnumbering UAVs, a single UAV will be associated with multiple POIs. The fitness of each solution is evaluated using the weights from the provided weight matrix, and the mating pool is selected favouring fit parents. Children solutions are created from the selected parents using a crossover method in which a section of data from two parent chromosomes is swapped. Mutation then takes place by randomly selecting and swapping two data points. This allows for variability in the children chromosomes and prevents premature termination. These new children chromosomes will replace the old parent chromosomes if they have better fitness values. The process is repeated until it reaches the termination criterion, depending on the situation could be a number of iterations or a certain fitness value threshold. The result of the GA is now an optimized chromosome containing the data, depending on the chromosome encoding to be of the most optimized solution found throughout the iterative process.

**Algorithm 2** Pseudo-code of Genetic Algorithm

```
Initialize Generation of random candidate solutions
while Stopping Criteria is False do
    Compute fitness values
    Selection: Select parent solutions
    Crossover: Generate children solutions using a crossover method
    Mutation: Mutate the children solutions
    Local Optimization: Replace parent solutions with improved children solutions
end while
return Best Solution
```

The GA was implemented in previous works for solving the m-TSP problem [46] [47]. Furthermore, the GA was chosen due to its versatility, existing usage in path planning applications, and speed-up through multi-threading [48].

## 3.4 GPU Background

### 3.4.1 Overview of CUDA Enabled GPU's

Graphics Processing Units (GPUs) are massively parallel processing components which generally contains cores commonly numbering in the thousands. They were originally developed for the gaming industry to deal with the large amounts of data that needs to be processed continually. More recently they have been adapted for use as a general-purpose programming platform for various applications. This architecture differs from the traditional central processing unit (CPU) which contains at most a few amounts of cores, but they have been designed and highly optimized for peak sequential code performance. Recently, the advancement in achieving higher clock frequencies on CPU chips has slowed significantly largely due to the inability to meet the heat dissipation requirements. This has caused the progression of increasing the processing power of CPU's to slow down, resulting in other approaches, particularly the use of GPU's to take the spotlight. GPU's offer an alternative solution for increasing computing power. Instead of increasing clock frequency, they introduce a higher number of threads, allowing for the massive amounts of data to be processed in parallel. A commonly used performance metric known as floating point operations per second (FLOPS) is used to evaluate the capability of a GPU to perform calculations.

GPU's offer qualities which are well suited for the lightweight computation of large amounts of data, they have the potential to perform with execution times magnitudes less than that of traditional CPU's with a much lower per-core cost. To implement a program which effectively uses the features of a GPU generally has a much greater development time as opposed to its sequential implementation. Hence it is critical to ensure that the resulting

program has the speed-up potential if being parallelized.

Nvidia has created an API and parallel computing platform known as Compute Unified Device Architecture (CUDA) in which allows for general purpose computing on graphics processing units (GPGPU). CUDA has been designed to work with C, C++ and Fortran languages; it also contains accelerated libraries, debugging and optimization tools [49]. CUDA has become quite popular in the development of applications for GPU's and was found to be used in most of the work which implemented parallelized path planning algorithms. Many of these algorithms are suited for parallelism, as elements can undergo independent processing in parallel, resulting in a significant overall speed up [27].

### 3.4.2 GPU Programming Model and Architecture

In the development of a massively parallel program, it is imperative to understand the GPU hardware design and architecture to create an efficient and effective program. CUDA enabled GPU hardware designs to have slight variations since its release. However, the base architecture remains the same and will be discussed.

In CUDA, threads are organized into a configurable two-tiered hierarchy of grids, and blocks found within each grid presented in Figure 3.4.1. The maximum number of threads within a single block is 1,024 which is configured in one, two or three dimensions. Additionally, grids can also be launched in up to the third dimension. Threads are launched from the CPU (host) with identical copies of a kernel function, a method explicitly compiled to be run on the GPU. This type of processing is known as single instruction, multiple threads (SIMT).

Another aspect of processing on a GPU is the organization of memory (Figure 3.4.2) and the consideration of its access patterns. The global (device) memory is made up of banks of double data rate random access memory [51] in which is the only location apart from constant memory that data can be copied to and from the CPU before and after launching kernel functions. Global memory large in size but the trade-off is its high latency. Shared memory is

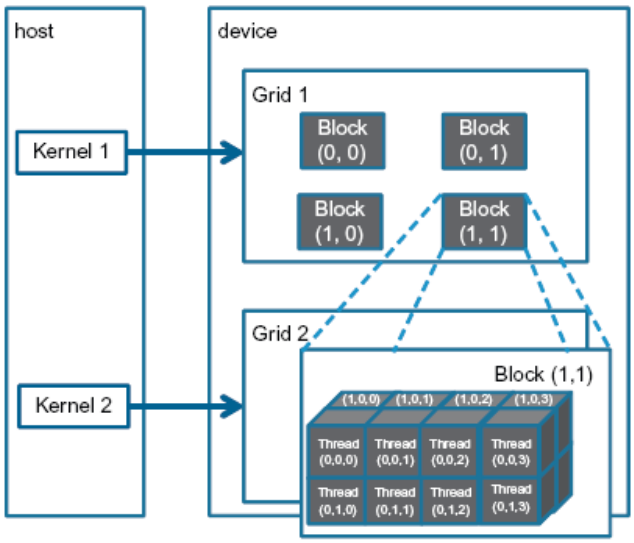


Figure 3.4.1: Launching kernels from the CPU in grids consisting of blocks. In this case each kernel function is launched on a 2D grid consisting of four blocks, each of which had threads arranged in 3D. Reproduced from [50].

much smaller in size and has a scope limited to each individual thread block but offers significantly lower latency in comparison to global memory making it useful for storage of data which experiences a high number of operations. One vital aspect of parallel processing is the coalescing of global memory read and writes. This ensures that the actual throughput of the device can reach values close to its theoretical throughput which is a metric of how effectively the programmer is utilizing the GPU. A memory access instruction is capable of retrieving large segments of aligned data; having misaligned data can result in the program bottle-necking as each of these instructions are relatively slow.

The execution of instructions are performed on a block by block basis by streaming multiprocessors (SM's) found in Figure 3.4.3, with blocks assigned at runtime. The number of SM's as well as the number of blocks assigned to each are limited, often causing the actual number of threads concurrently running instructions to be less than the total number of threads launched, particularly in applications a large number of threads. Each SM runs the same

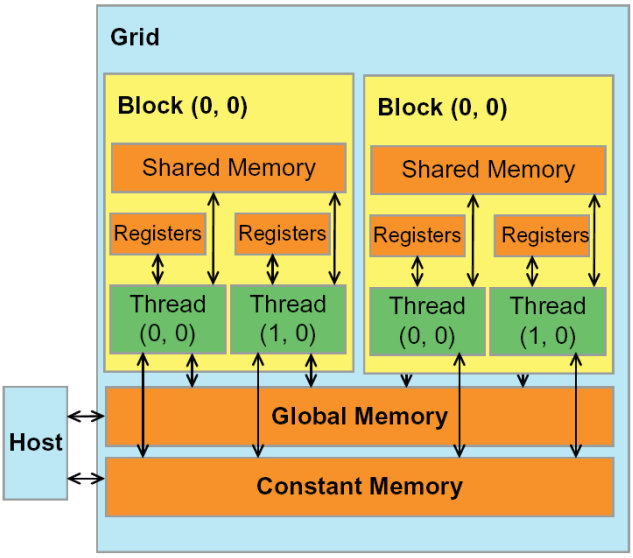


Figure 3.4.2: GPU memory hierarchy illustrating memory scopes. Reproduced from [50].

instruction on a group of 32 threads called a warp; the execution efficiency of the warps is a major consideration when creating and analyzing a parallel program. Reason for this being that SM's have limited scheduling resources, and are designed to fetch and execute one instruction at a time hence making it desirable to have all threads within a warp execute together. A delay of a single thread will cause the entire warp to bottleneck, a common cause of this is thread divergence through the use of conditional statements in the kernel function.

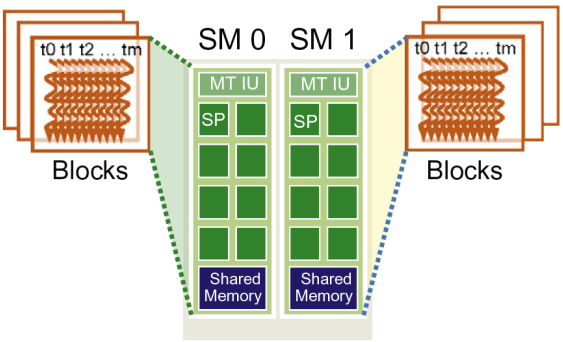


Figure 3.4.3: The architecture of streaming multiprocessors (SM's). Reproduced from [50].

# 4 Method

The proposed flight planning program employed a three-step methodology which, in conjunction proved capable of achieving the desired results. The flow of the program can be explained as follows: Step 1 consists of the raw data processing and cost function calculations, Step 2 processes the formatted map data for each POI using the Bellman-Ford algorithm accelerated on a GPU, Step 3 uses the resultant information to assign each UAV to POI(s) using the Genetic Algorithm and construct complete paths. Figure 4.0.1 is a high-level program flowchart showing the steps performed by the program throughout its execution.


Figure 4.0.1: A flowchart of the program steps illustrating the inputs and outputs

## 4.1 Step 1: Data Formatting and Initialization

For a path planning program, an appropriate design decision must be made to appropriately correlate the data structure and format with the algorithm used. The usage of the BF dictated that it would be necessary to use a graph-based representation. The raw data had to be processed and used in the creation of the path planning *scenario* containing all the required parameters and information for the program. In this work, the term *scenario* referred to

the software object created internally to the program that was subsequently passed through the two algorithms and contained the information for the final reconstruction.

### 4.1.1 Map Representation

The raw data of the terrain map consisted of a 2D matrix of the terrain elevation heights; each value's index represented a position in x-y coordinated based on the specified scale for the map. The dimensions of this matrix along with the scaling factor was used to set two indexing values; the number of rows $n$ and columns $m$, these two values also represent the x-y limits of the map space. Another map parameter was the number of vertical layers which dictated the vertical resolution and spacing.

Based on these inputs, nodes were then generated, each representing a potential location in which a UAV can occupy. The number of nodes was based on two parameters, the map resolution from the terrain matrix, and the desired number of layers vertically. In a partial visual representation found in Figure 4.1.1, they can be seen overlaid as vertically stacked layers of 2D grids on top of the terrain. In determining the vertical spacing of layers, at each point a floor altitude was set as an offset from the respective points terrain elevation. A top altitude was also defined for the entire map, calculated as an offset from the highest terrain feature in the given area. For each x-y location, the nodes were evenly spaced between the floor altitude and the top altitude. This resulted with node elveation changes when advancing towards terrain, and vertical spacing between nodes reduce over elevated terrain. To keep track of these node altitudes, an array of node elevations was stored, later used in cost function calculations and path regeneration. This method was chosen over having consistently spaced nodes vertically as it prevented drastic altitude changes as nodes neighbouring steep terrain features were highly separated.

With this structured node layout, each node has a set of adjacent nodes in each cardinal and diagonal direction on its current layer and the layer above and below it. A corresponding edge array was then generated and used to

track the connections leading from each node to its neighbours; the values contained in this array represented the cost of travel between nodes.

This representation was chosen due to the structured nature of the map 3D grid of neighbouring nodes with a predetermined amount of edges. An advantage of the grid representation was that the edge connections can be determined without additional data, and there was no need to store the nodes' previous information as paths will consist of adjacent nodes. This structure consumed $O(N + E)$ space, growing linearly in size with the map.

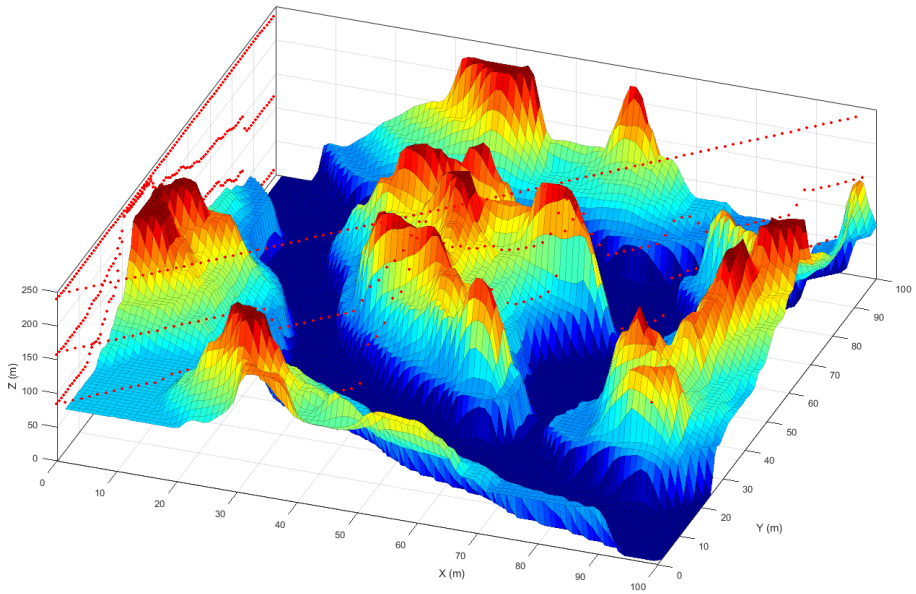


Figure 4.1.1: An example showing the 3D distribution of partial segments of nodes. Note this only shows two lines of nodes at a low resolution for visualization purposes.

### 4.1.2 Data Formatting & Size

As previously stated, there is potential for reduction in the BF computational complexity, this can be achieved through modifications to the graph data structure. Since the graph was considered sparsely connected as each node had only connections to adjacent nodes the data representation was simplified into a list style format. The edges and nodes were represented as two independent lists, and rather than having additional data relating the two for indexing, indexing was based on the map parameters.

The edges were all contained within a single array, edges for node $N(E)$ was accessed through $N * connectivity$. The node array was formatted as a 1D array, based on its indexing however was based on the number of columns, rows, and layers: $N\_index$

Several in-line functions were implemented within the scenario class, each returning the index of the neighbours of a node in a specified direction. Based on the indexing shown in Figure 4.1.2, the index can be found through mathematical operations using the number of rows $n$ and columns $m$ along with layer size. The letters in the array depict the direction the edge leads towards; F, B, L, R, U, D, represent front, back, left, right, up, down.

| Node Array | 1 | | | | | | | | | | | | | | | | | | | | | | | 2 | | | | ... | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Edge Array | F | B | L | R | FR | FL | BR | BL | UF | UB | UL | UR | UFR | UFL | UBR | UBL | DF | DB | DL | DR | DFR | DFL | DBR | DBL | F | B | ... | DBL | ... | n * 24 |

Figure 4.1.2: An example of the data array format created and passed to the next stage of the program in which a modified version of the BF algorithm utilizes the data in this format.

In the implementation, a new boolean array was used as an indicator if a node does indeed exist in the direction being checked. This addressed the edge cases for the map and upper and lower limits. Using this array, simplified the repetitive calling of the inline functions and reduced the number of parameters passed between functions. It was particularly helpful in GPU programming, allowing for a direction validity check before trying to access an array index, otherwise resulting in a segmentation fault if invalid.

A table of the data sizes was created, due to the GPU having a dedicated

and limited amount of memory it was essential to establish the data size limitations prior to proceeding. The table presented in 4.1.3 was parametric with some key parameters such as the map size, resolution, and the number of POIs.

| Bellman Ford Data | | | | | |
|---|---|---|---|---|---|
| Input | | | | | |
| **Data Name** | **Notes** | **Data Type** | **Data Size (byte)** | **# Items** | **Total size (bytes)** |
| Nodes | | int | 4 | 2,500,000 | 10,000,000 |
| Node Elevations | | int | 4 | 2,500,000 | 10,000,000 |
| Edge Weights | temp for reading .txt | float | 4 | 60,000,000 | 240,000,000 |
| Output | | | | | |
| **Data Name** | **Notes** | **Data Type** | **Data Size (byte)** | **# Items** | **Total size (bytes)** |
| POI Node Weights | | float | 4 | 50,000,000 | 200,000,000 |
| POI Connectivity Weight Matrix | | float | 4 | 400 | 1,600 |
| | | | | | |
| GA Data | | | | | |
| Output | | | | | |
| **Data Name** | **Notes** | **Data Type** | **Data Size (byte)** | **# Items** | **Total size (bytes)** |
| POI Chomosomes | | int | 4 | 20 | 80 |
| UAV Chromosomes | matrix for each waypoint | int | 4 | 20 | 80 |
| Path Nodes | | float | 4 | 25000 | 100,000.00 |
| | | | Total Bytes (MB) | | **460.10** |

Figure 4.1.3: An example of the data size calculations undergone during initial stages of program development. A fairly extreme example is shown with a resolution 10 meter resolution, 20 POIs.

### 4.1.3 Cost Function Embedding

Path planning, along with most optimization problems, involve the minimization of parameter(s). As stated, an objective of this work was to implement a solution which allows for the consideration of multiple parameters. Optimization was achieved through the use of a multi-objective cost function parametrize the metrics chosen for minimization such as in [52]. Two objectives were decided upon, the first was minimizing the distance travelled by each UAV, and the second was to maintain a minimum altitude. Optimizing for distance was essential due to limited operating range of aircraft, this also directly affects the overall flight and mission time. Additionally, a significant cost of UAV operation is its fuel consumption, which was minimized through distance, as well as altitude.

Unnecessary changes in altitude are undesirable, due to increases in fuel consumption and add complexity to routing. It is however desired to maintain a minimal altitude as it minimizes the risk of detection from enemy observers and radar [53]. Due to the high map resolution, and the method of path planning it was possible to generate paths capable of close terrain following.

It can also be noted that this cost function was expandable if user requirements demanded. A key aspect in formulating a multi-objective cost function was the integration of the parameters. This was achieved through the usage of reference values and weights for each objective. After the creation of the edges, each connecting two adjacent nodes $E(u, v)$ the cost of each edge $C(u, v)$ was calculated through the function shown in Equation 4.1.

$$C(u,v) = \omega_1 * \frac{d_{(u,v)}}{d_{ref}} + \omega_2 * \frac{a_{(u,v)}}{a_{ref}} \tag{4.1}$$

This equation required finding two reference values used for normalization, $d_{ref}$, $a_{ref}$. These were found using the locations of the two furthest POIs ($i$ and $j$) in the scenario of which the straight line distance and difference in altitude were calculated respectively. For the case of zero altitude change, the reference value would be set to 1 preventing division by zero in the cost calculation.

$$d_{(u,v)} = \begin{cases} horzscale & \text{if } E(u,v) \text{ is Cardinal} \\ \frac{horzscale}{sin(45)} & \text{if } E(u,v) \text{ is Diagonal} \end{cases} \tag{4.2}$$

The distance between each of the selected points $d_{(u,v)}$ and altitude change $a_{(u,v)}$ were the two factors taken into account. The cartesian distance and altitude reference values were simply calculated as:

$$d_{ref} = \sqrt{[i(x) - j(x)]^2 + [i(y) - j(y)]^2} \tag{4.3}$$

$$a_{ref} = |nodeelev(i) - nodeelev(j)| \tag{4.4}$$

The two additional parameters $\omega_1$ and $\omega_2$ were used as weights for the distance and elevation costs respectively. These values could be varied based

on user preference.

### 4.1.4 Results of Step 1

Upon completion of this step, the scenario was now considered initialized, ready for further processing. The data that the scenario contains was then used for the next step in which a BF algorithm was run to calculate the paths of least cost. The essential data for this is contained in the edge array, of which the cost function is now embedded. Additionally, the list of nodes including the start point was organized into an array.

## 4.2 Step 2: POI Solving

This next step utilized an adaptation of the Bellman-Ford algorithm specific for the data in the scenarios and was modified to exploit the massively parallel nature of a GPU.

The BF was run once for each point, calculating an array of node weights for each respective point as the starting location. The result of this was an array of nodes for each POI, with the calculated cost it would take to fly to each point in the map space. The data from these arrays were summarized in a POI connectivity weight matrix consisting of the total path costs from the POI to all the other POIs including the start point. This was run on a pre-allocated set of node arrays kept in order through an additional array of pointers.

### 4.2.1 BF Sequential Implementation

The BF was first implemented in sequential on the CPU to ensure feasibility for solving the given scenario. Algorithm 3 outlines the BF implementation which was a modified take on the original version located in Chapter 2. Notable changes were in collecting the neighbouring node data, and the utilization of a flag to determine when the stopping condition was reached.

Before running the algorithm, the node weight arrays were initialized to 'infinity' except for the source node. This value in the program was the max-

imum value of a float data type. The source node is either the start/end location, or a POI node was initialized to zero, causing iterations to update nodes beginning at this point. Each node is then iterated through, and updated in Line 6 if, based on its weight and associated edge weights a lower weight is found. An advantage of the data format is knowing beforehand the edges associated with the node, along with the neighbouring nodes weight values. These values were preloaded into temporary arrays to check if the node needed weight updating, eliminating the need to check for every edge.

**Algorithm 3** Sequential Implementation of Bellman Ford

```
1:  for all POI do
2:      Node weights V(G) = ∞, POI node V(s) = 0
3:      Flag = False
4:      while Flag = False do
5:          Flag = True
6:          for all Nodes: V(i)εV(G) do
7:              for all Neighbour Nodes V(n) do
8:                  if V(n) + E(n,i) < w(i) then
9:                      w(i) = V(n) + E(n,i)
10:                     Flag = False
11:                 end if
12:             end for
13:         end for
14:     end while
15: end for
16: return Minimal routing costs
```

Each iteration would progressively update nodes further and further away from the starting node until no further changes are required. This is indicated by using a flag in addition to updating node weights.

The results from this step were tested through reconstructing a single path between two points. It became clear an additional method of setting the nodes lying at the map perimeter to a maximum value was needed to prevent the path from leading off course.

Having tested this algorithm on a variety of scenarios including scaled down resolutions, a various number of layers and simple cases it was shown

to be a valid shortest path solving method with the primary downside is the exponentially increasing run-times with increases in data size and POIs.

### 4.2.2 BF Parallel Implementation

As previously discussed, there are a variety of aspects that need to be considered in converting a program for GPU acceleration. The data was formatted in such a way to maximize accessibility to data in memory, and with the nature of the selected BF based search method, parallel processing of nodes was deemed possible. The relatively independent nature of the data structure allowed for parallel implementation as threads could independently work on the calculation of node weights without much concern about run-ons.

Due to the GPU having a separate, dedicated memory, the copy of relevant data was necessary prior to the algorithm starts. Fortunately, only the edge array containing each edge weight needed to be copied. The node arrays were initialized on the GPU using a simple kernel function with the final results then copied back to the CPU for validation and to be used to complete this step by creating the POI connectivity matrix (PCM), and later for path construction.

In calling kernel functions, each GPU thread was individually assigned to a node matching its index, allowing for each node to be independently worked on, and updated if necessary. This method was chosen over running a thread per edge, negating the potential for race conditions. However, due to nodes being updated in parallel the spread of weight information through the graph would likely take more iterations. This was proven and can be found in the following chapter in which a comparison is performed based on the number of iterations required. Despite having the potential for more iterations, it can be expected that the speed-up allowed though using GPU would be substantial. Each iteration required a single kernel launch containing the method with the Bellman-Ford algorithm. The kernel function would first initialize the data required for its specified node, similarly to the sequential implementation and run through the edge relaxation process. To implement the flag check, indicative of algorithm completion, a memory copy from the GPU to CPU of the single boolean had to be called. As these instances of memory copied is

time intensive, it was deemed faster to perform memory copies after a given number of iterations on the GPU, enough to allow for an adequate amount of data spread, however short enough to prevent too many unnecessary iterations of which no updates occur.

**Algorithm 4** Parallel Implementation of Bellman Ford

```
1: for all POI do
2:     Launch: Initialize Kernel
3:     Edge Array Copy: CPU → GPU
4:     Node Direction Array Copy: CPU → GPU
5:     Launch: Flag Reset Kernel
6:     while Flag == False do
7:         Launch: Launch BF Kernel
8:         Flag Copy: CPU ← GPU
9:     end while
10:    Node Array Copy: CPU ← GPU
11: end for
12: return Route costs for all POIs
```

Algorithm 4 shows the program flow and kernel launching from the CPU. Algorithm 5 consists of the kernel implementation launched on each GPU thread.

**Algorithm 5** Bellman Ford CUDA Kernel

```
1: NodeID = ThreadID
2: Flag = True
3: Load Neighbouring Nodes into temp array
4: Load Incoming Edge Weights into temp array
5: Edge Relaxation
6: if Edge Relaxation Occurs then
7:     Flag = False
8: end if
```

In the initial program development, the sequential node array results were compared to the node arrays copied from the GPU program which would then be considered 'correct' if the values matches within a small margin of error. The margin of error was caused due to the precision of the data type (float) and was within acceptable tolerance for the application.

### 4.2.3 BF Results

As stated, there was a significant amount of data from the result of this step. This data was arrays of nodes, now each with its final calculated weight. Based on the array for each point set as the 'source', each node weight within that array represents the least cost to reach that location. These costs were derived from the originally embedded cost function from the edges. An advantage of incorporating distance as a weight is that there was a positive gradient in terms of node values moving further from the start location as shown in Figure 4.2.1. Additionally, the cost of changing altitude varied based on the terrain features. This aided in allowing for quick path construction found in the final stage of the program.

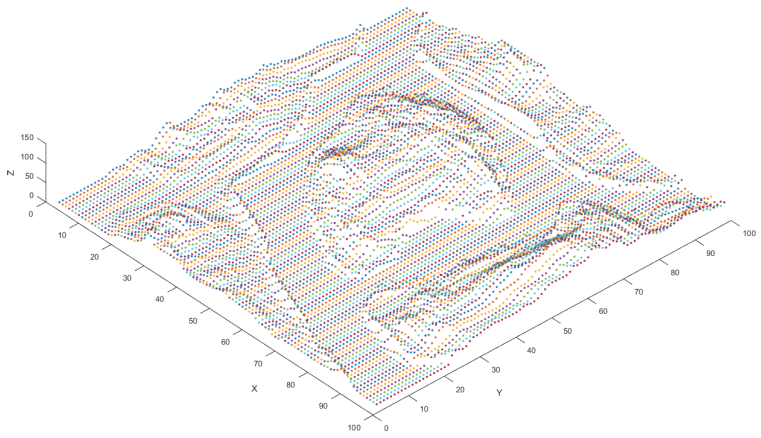


Figure 4.2.1: A visual representation of the resultant node weight array for a single point. The result presented was reconstructed based on the location of each node despite being represented as a 1D array within the program.

The relevant results, the cost for any possible desired path was gathered and stored in a smaller, node weight matrix (Figure4.2.2), representing the interconnectivity of each POI and start point, was then to be passed as an input for the next algorithm/step.

This was achieved by looping through each POI $p$, returning the node weight of every other POI node weight from within the node array belonging to $p$.

| | S | A | B | C |
|---|---|---|---|---|
| S | | # | # | # |
| A | # | | # | # |
| B | # | # | | # |
| C | # | # | # | |

Figure 4.2.2: A visual representation of the resultant matrix from step 2. $S$ represents the starting point, letters $A, B, C$ represent three POIs in which the path cost to/from another point can be found.

## 4.3 Step 3: Multi-UAV Solving

Now having the complete set of weights for any possible path routing between points, the next step was to create the complete routing of a minimal cost by optimizing the order of POIs visited and the allocation of the UAVs to the POIs. This problem becomes less trivial when dealing with larger numbers of POIs and/or UAVs. It was decided to use a meta-heuristic approach to generate an optimized solution rapidly and reliably given the conditions for this step in the program. In this work, the Genetic Algorithm was used to simulate the evolution of UAV and POI pairings through the process of selection, crossover, and mutation allowing for fast convergence to an optimized solution. It can be noted that solving this problem is quite similar to solving the m-TSP problem, although now utilizing the data calculated as a result of the BF solver.

### 4.3.1 Genetic Algorithm

The primary modifications to the GA were identified as: adapt to solve for the multiple UAVs and POIs, accounting for the single depot situation in which

each aircraft must return to the start point, and utilization of the POI weight connectivity matrix.

In this approach, the number of UAVs and POIs were defined as $m$ and $n$ respectively. Each chromosome consisted of two arrays of equal length $n$ one being a list of POIs and the second, its respective assigned UAV number. With this format, UAVs are matched with POI(s) based on its indexing within the arrays. For scenarios containing POIs outnumbering UAVs, a single UAV will be associated with multiple POIs.

The initial population of candidate solutions were generated at random. Each chromosomes POI array consisted of all the POI numbers shuffled at random, without any repeating values. The UAV assignment to the POIs was generated at random. The chromosomes were then evaluated based on the overall path cost for each UAV prior to starting the GA iterative process.

| UAV - chromosome | 2 | 1 | 2 | 4 | 3 | 4 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| POI - chromosome | 6 | 4 | 7 | 1 | 3 | 9 | 2 | 5 | 8 |

Figure 4.3.1: A visual representation of a chromosome used in the GA with 9 POIs and 4 UAVs.

The fitness of each candidate solution was evaluated using the weights from the provided weight matrix. Tournament selection was used to choose parent chromosomes. This was performed by checking a set number of parent chromosomes and selecting the one with best fitness. Children solutions are created from the selected parents using a crossover method in which the POIs from one parent chromosome was copied over and the UAV arrays were merged, favouring each parent equally. This method can be found in Figure 4.3.2.

Mutation then takes place by randomly selecting and swapping two points. This allows for variability in the children chromosomes and prevents premature termination.

With the new generation of candidate solutions, the order of POIs assigned to each UAV from within each chromosome. To ensure the POI order was feasible, and did not cross over itself, the 2-opt TSP solving method was

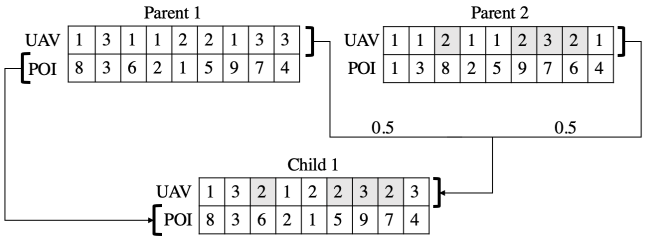


Figure 4.3.2: Illustrating the crossover method for the specified chromosome format. This exmaple contains 9 POIs and 3 UAVs.

applied. These new children chromosomes would then replace the old parent chromosomes if they have better fitness. The algorithm is repeated until it reaches the termination criterion, a set number of iterations. This was deemed adequate due to the speed of each iteration being in the range of a millisecond.

An advantage was the ability to run the 2-opt operation in parallel on the CPU using the openMP library. This resulted in speed-up for a very low development cost.

Another notable advantage to this method was the fitness values being taken from the weight matrix rather than being calculated in each iteration allowing for fast iterations. Additionally, the two chromosome coding method allows for better population evolution enabling faster convergence. The stopping criteria was set to limit the number of generations, through testing it was found that 100 generations and 100 candidate solutions would produce acceptable results with a significantly fast runtime.

Due to having a relatively small amount of input data from the previous step this allowed for the algorithm to a quickly converge to an optimized solution. This was first tested in a standalone environment using Matlab, which randomized inputs were fed to the solver and map results were plotted, along with convergence values. The weight values based on the Cartesian distance at this point which was replaced with the weight values from the POI connectivity matrix.

### 4.3.2 Path Construction

The result of the GA was an optimized chromosome containing the order of POIs with their respective UAV assignment. Based on the best chromosome from the last generation of the GA, a list of POIs per UAV was extracted. Routing cost was calculated under the assumption of a round trip from the start point; however, this point was not included in the chromosome data. This segment was added to the solution enabling the UAV to return upon mission completion. Through using the saved weighted node arrays from Step 2, a detailed path can then be traced between the POI's distributed to each of the UAVs. This path consisted of adjacent nodes for the UAV will travel through. This allowed for a path of high resolution, and that remains true to the calculated cost.

**Algorithm 6** Path Segment Construction

```
    Input : Start Node, End Node
1:  PathArray[0] = End Node
2:  i = 1
3:  while PathArray[i-1] ! = Start Node do
4:      PathArray[i] = nextNode()
5:      i++
6:  end while
7:  Reverse Path Array
8:  return Number of Nodes, Path Array
```

A lightweight function was implemented to construct the paths, performed by descending through the array of node weights (Algorithm 6), and returns the path segment (a list of way-points) between two points. Due to the nature of the node weights, there was a gradient at each point, allowing for precise, and quick construction. Finally, on a per UAV basis, the segments were stitched together resulting in a complete list of desired way-points for each UAV to travel through starting and finishing at the 'start point'.

These lists, combined with the input scenario information was used to reconstruct the map, with paths highlighted for visualization. This was the final desired result of the program, which was then evaluated in terms of

runtimes, and visual inspection further discussed in the following chapter.

# 5 Results

Chapter 1 introduced the path planning problem and highlighted the criteria needed for the implementation of a UAV path planner and provided and overview of the work performed in this thesis. Chapter 3 provided the reader with background of the algorithm types employed in the work, and programming considerations in order to achieve speed-up through GPU utilization. The previous chapter described, in depth, the methodology and development of the path planner presented in this work. This chapter serves to present the reader with the validation of the program and the achievement of the goals set out in Chapter 1. First, the verification of the individual program steps are presented, followed by the overall program output validation performed for a variety of scenarios. This serves to show that the generated paths are indeed feasible, and through scenario reconstruction and visual inspection, can be seen as an adequate results. Next, testing of the program for timing and speed-up was performed to validate the acceleration using the GPU, feasibility for in-mission recalculation capability, and to evaluate program behaviour given a variety of hyper-parameters.

## 5.1 Verification

The program verification began by establishing that the output for each program stage was as desired as per the design requirements. For the first step this was through checking through samples of edge weights, ensuring their values were as expected based on the cost function. In step two, the node weights were verified and plotted to check the behaviour of the BF algorithm.

Lastly the GA was checked by creating simple routing paths with a set number of UAV's. Finally the entire program output was verified by passing multiple scenario's through and rebuilding the final results which included the routing for each UAV's as defined in each scenario. The program was developed in C++ with CUDA 9.1 on Visual Studio. The program was executed on a HP Z820 workstation with 18 GB of DDR3 RAM, an Intel Xeon E5-2650 CPU and Nvidia GeForce GTX 1080 Ti. The CPU contained a total of 16 cores with 32 logical processors. The GPU had 3584 CUDA cores with a clock speed of 1582 MHz and 11GB of memory.

### 5.1.1 Verification of BF

In verifying the output of the BF step for which node weight arrays were calculated by the GPU, the implemented sequential version of the BF was also run. The output node arrays for an individual point generated by each version was compared for correctness. This performed for each scenario tested, however, in final program implementation it would be possible to negate the sequential BF in order to achieve speed-up. Additionally, to evaluate the behaviour of the algorithm and its propagation of the cost function, plots of the node weight values were created on a layer by layer basis. An example of this can be found in Figure 5.1.1 where it can be seen that the weights increase further away from the starting point, and terrain features are accounted for.

### 5.1.2 Verification of GA

The standalone verification of the GA with modifications for use with the POI connectivity matrix with a varying number of UAV's was performed utilizing Matlab. The tests run were: 1) UAV assignments to POI's 2) Solution convergence.

A set of randomly generated POI's each with a set of Cartesian x-y coordinates were used to test the assignment of UAV's with the algorithm evaluating routing based on distance. The resulting point to point routes were plotted for visual inspection to gauge the behaviour of the algorithm.

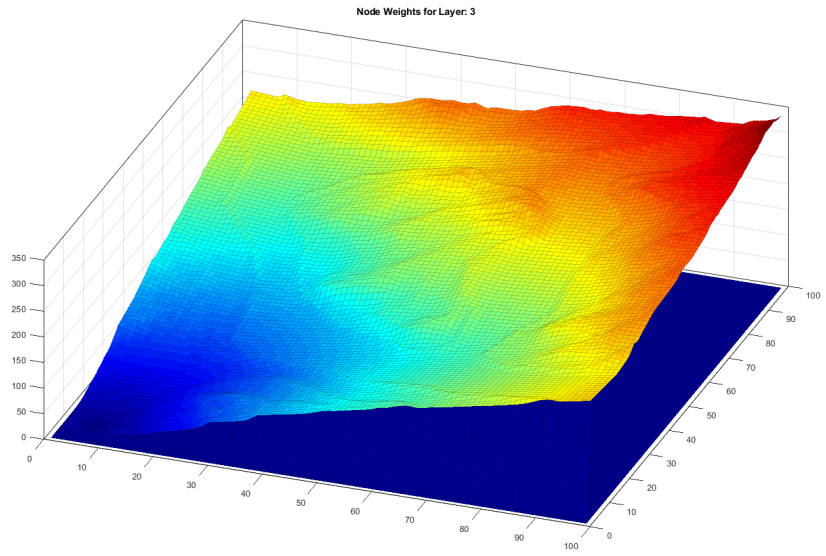


Figure 5.1.1: BF Weight Array verification output

Additionally, it had to be verified that sufficient iterations were undergone as that was the decided the stopping criteria. A method for checking was through plotting the fitness values of each generation of which convergence to a value was an expected outcome. Figure 5.1.2 illustrates an example of a convergence plot, this was performed for many test scenarios, in which it was determined that setting the iteration amount to 100 resulted in adequate convergence.

## 5.2 UAV Path Quality

The first step in UAV path planning verification is through visual inspection of the output path quality. This allows checking of the path, ensuring it remains collision free, has logical routing, and also provides for a deeper understanding of the algorithm behaviour.

It was noted in the development of the path reconstruction, with cases of routing near the edge of the map there may be instances of the program

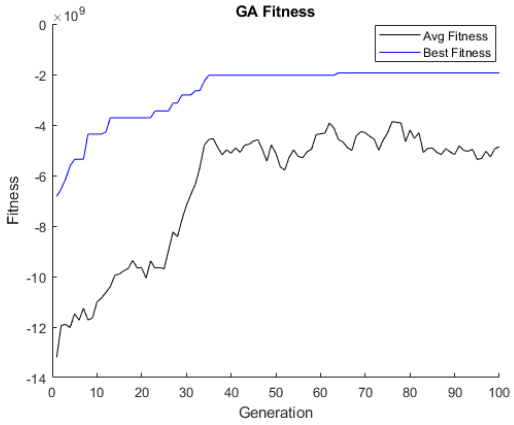


Figure 5.1.2: The average and best fitness values plotted over the course of 100 iterations showing adequate convergence

getting 'lost' in determining the next way-point. This was rectified by adding a padding layer around the map edges.

As the path for each UAV is a list of adjacent nodes, it can be seen that routes remain free of terrain conflicts. This is attributed to the nodes being an offset of terrain elevations, resulting in their valid connecting edges also remaining free of conflict.

Figures 5.2.1 - 5.2.3 present three examples of the scenario reconstruction with the final resultant paths for each UAV. The first scenario consisted of fictitious terrain features which was used for program development and testing. The following two scenarios are examples of two real world terrain maps based on geography from various regions. This was done to ensure program feasibility in complex, real-world scenarios.

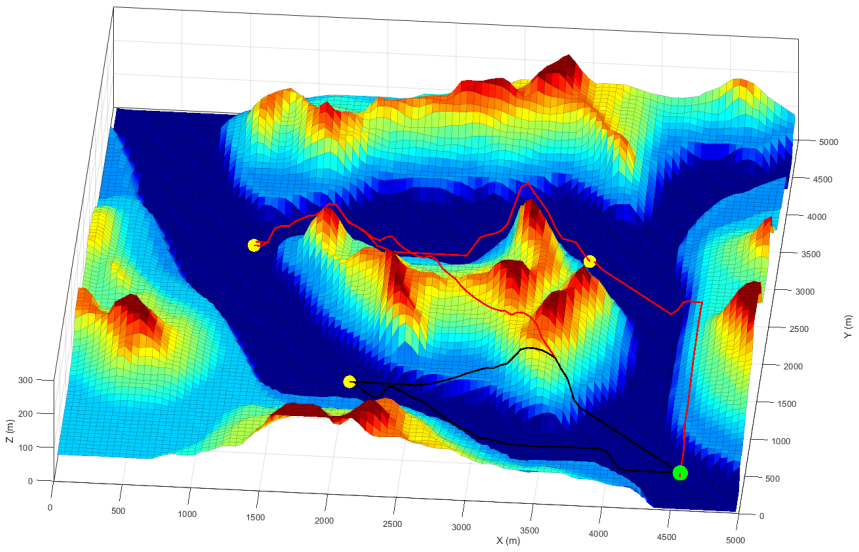


Figure 5.2.1: Scenario reconstruction with two UAV's and four POI's

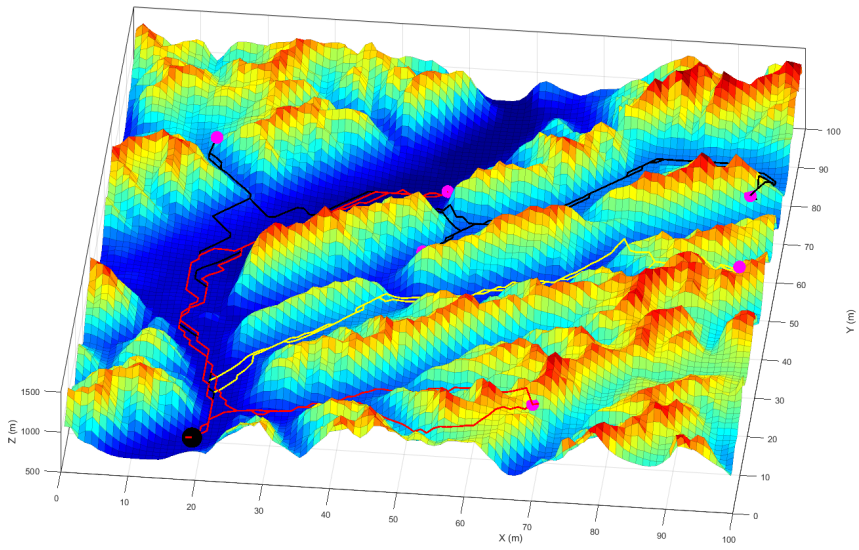


Figure 5.2.2: Scenario reconstruction with three UAV's and six POI's

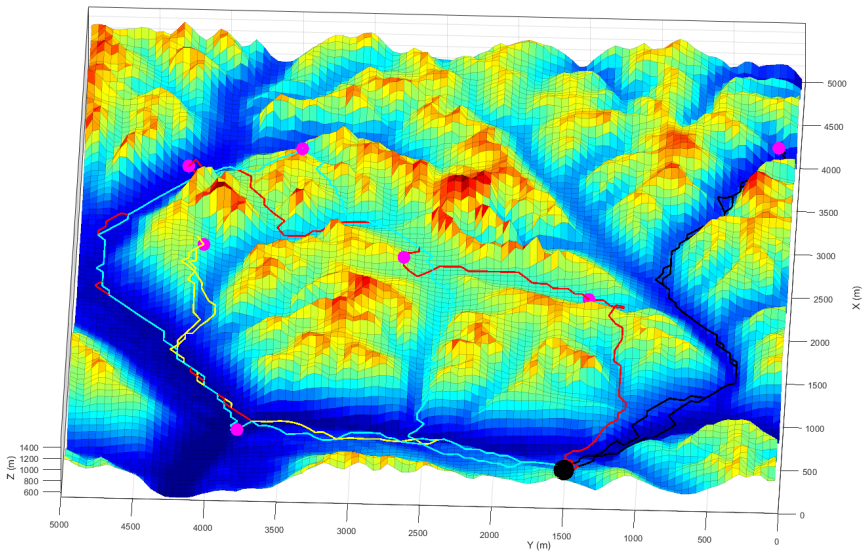


Figure 5.2.3: Scenario reconstruction with two UAV's and seven POI's

It can be seen from the scenarios, the paths are of high resolution, capable of closely following the terrain, minimizing altitude, overflying the desired POI's and return back to the starting point. These paths are considered of high quality as they have high resolution, truly achieve proper path creation with the consideration of terrain and high level of cost minimization, all of which has been shortcomings of many existing methods.

## 5.3 Numerical Results

The second step of program verification for the purpose of this work was ensuring adequate speed-up was achieved. Slight variances in runtimes were expected, hence each timing data point used was an average based on a number of repeated runs.

In overall program evaluation, each step and in turn, function was measured in terms of time consumed. As expected the sequential BF method consumed significantly more time than any other aspect. The timing was

greatly reduced and work balance was evened out as can be seen in Figure 5.3.1.

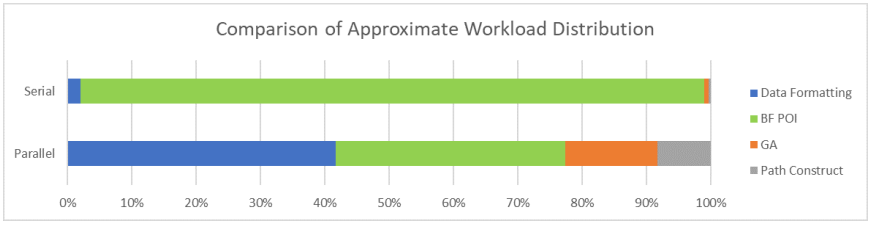


Figure 5.3.1: Chart comparing the amount of work performed at each step. The key aspect is the reduction in the BF processing work.

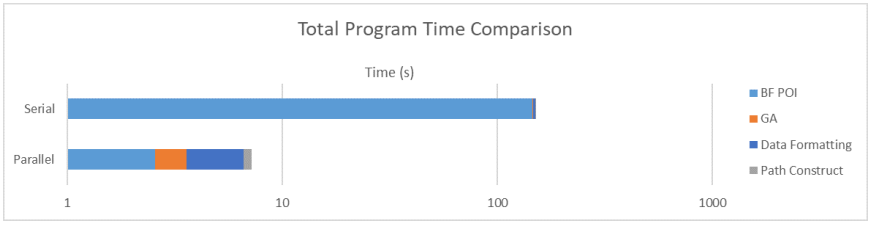


Figure 5.3.2: An average overall program runtime broken into steps. This illustrates the significant advantage of using a GPU in accelerating the BF SSSP algorithm.

### 5.3.1 BF Timing

As the change in hyper-parameters primarily affected the BF algorithm, further timing analysis was performed and can be found in Figures 5.3.5 - 5.3.8. Behaviours such as number of iterations required to variation of GPU parameters were also tested.

As expected the number of iterations required by the BF kernel function outnumbered the CPU version, this was tested for varying the number of POI's in a scenario (Figure 5.3.3). This indicates that the parallel algorithm was not as work efficient in comparison to the sequential. Meaning the propagation across the map nodes occur at a slower pace. The per iteration runtime however was significantly faster resulting in a high amount of speed-up as

found in Figures 5.3.5 and 5.3.6. This speed-up negates the greater number of iterations required. It was also noted that the number of iterations is largely based on the number of POI's, however do vary as each POI required a different number of iterations to solve based on its location.

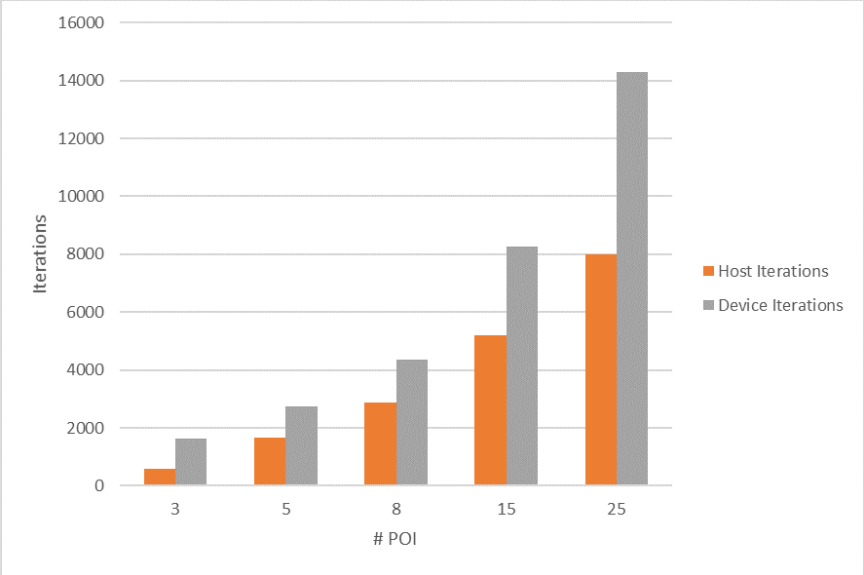


Figure 5.3.3: Comparison of the amount of host and device BF iterations required with varying POI numbers.

An important parameter was the number of threads launched per block on the GPU in running the BF kernel. This parameter was tested at a fixed problem size and it was determined the best speed-up occurred with a block size of 256. At greater numbers the amount of parallelism no longer benefits algorithm speed-up.

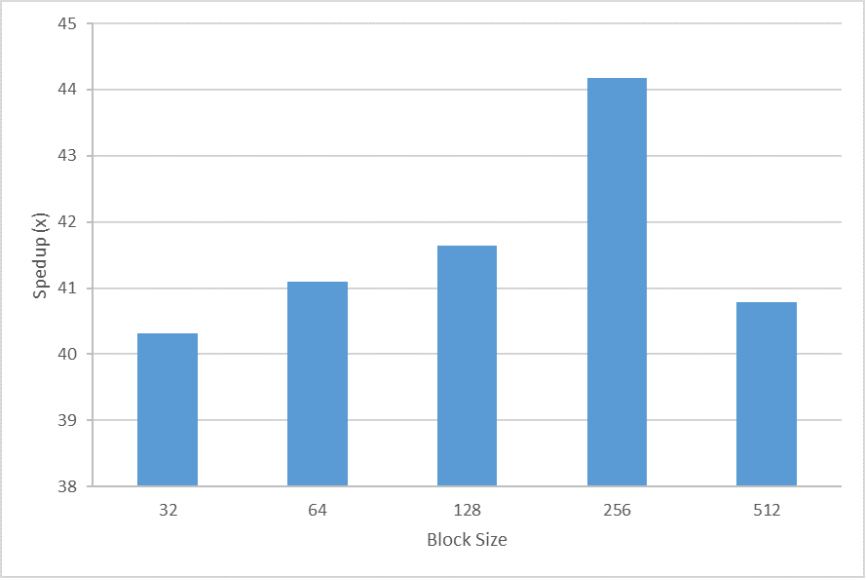


Figure 5.3.4: BF runtime varying the block size GPU parameter

Figure 5.3.5 depicts the significant runtime reduction achieved from using the GPU to run the BF algorithm. Identical problem cases were run with both the sequential and parallel versions with various number of POI's. The results of the algorithm matched, however it can be seen that the sequential version runtime would not allow for rapid recalculations. Using the same data obtained, the speed-up of the BF algorithm is shown in Figure 5.3.6 which peaked at approximately 45x.

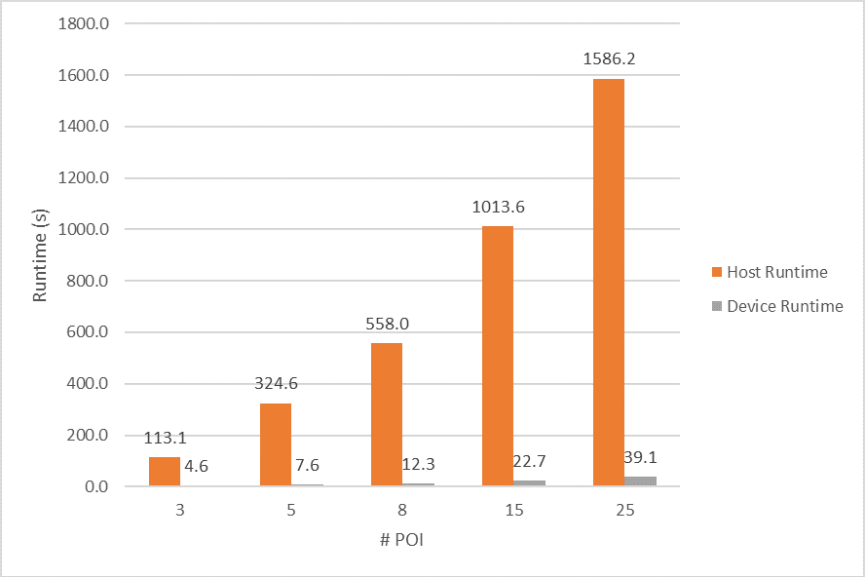


Figure 5.3.5: BF runtime varying the number of POI's.

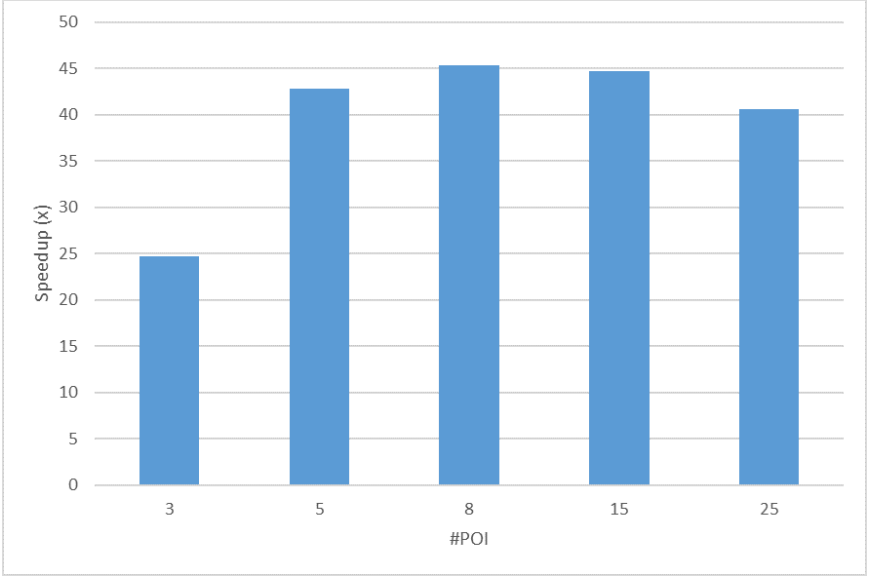


Figure 5.3.6: BF speed-up varying the number of POI's.

The massively parallel nature of GPU and its advantage can be seen

through the increase in data size. Figure 5.3.7 shows the speed-up achieved from running the BF algorithm on an increasing number of nodes based on various map resolutions.

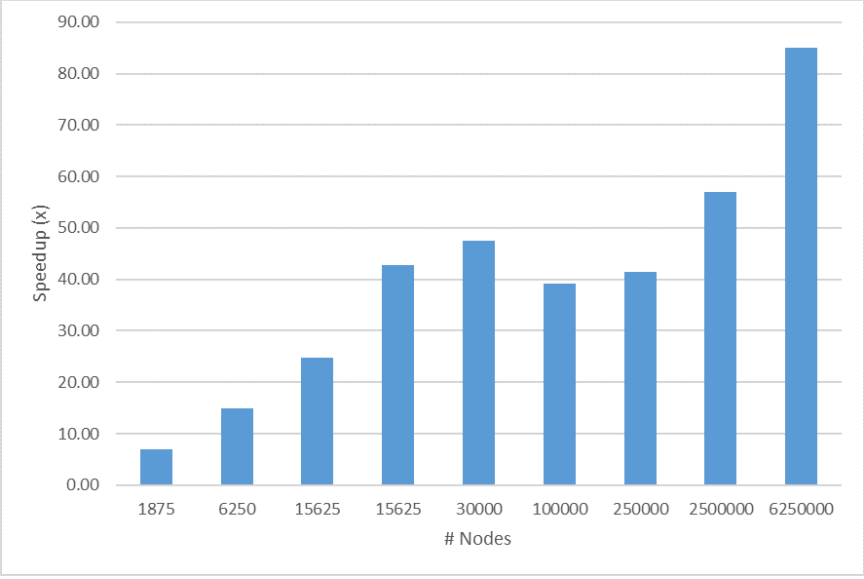


Figure 5.3.7: BF speed-up varying the map number of nodes through changing map resolution and number of layers.

Figure 5.3.8 characterizes the speed-up of the BF given a variety of 3D and 2D resolutions. Similar to the previous figure, this further shows the advantage of the parallel implementation with larger amounts of data. The speed-up continued to significantly increase with the resolution, hence further justifying the use of a GPU for larger scenarios.

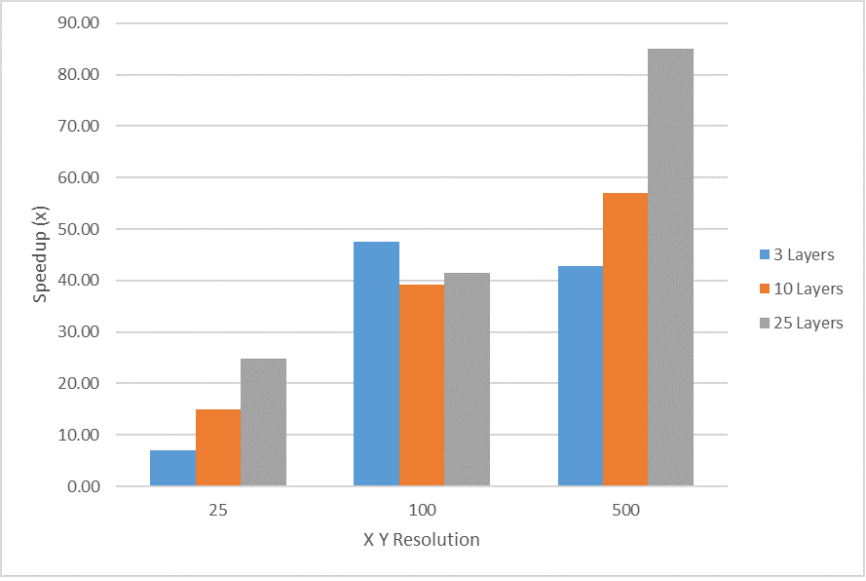


Figure 5.3.8: BF runtime varying the map 2D and 3D resolution.

### 5.3.2 GA Timing

The following two plots were created to characterize the GA timing behaviour given a varying amount of POI's and UAV's. Despite these variances, it can be noted that the runtimes were all short. It can be noted that the GA runs adequately fast given a the range of POI's, however runtimes start to increase significantly beyond 25 as seen in Figure 5.3.9. Figure 5.3.10 shows the runtime for a scenario for two UAV's, which was a poor case in terms of algorithm runtime.

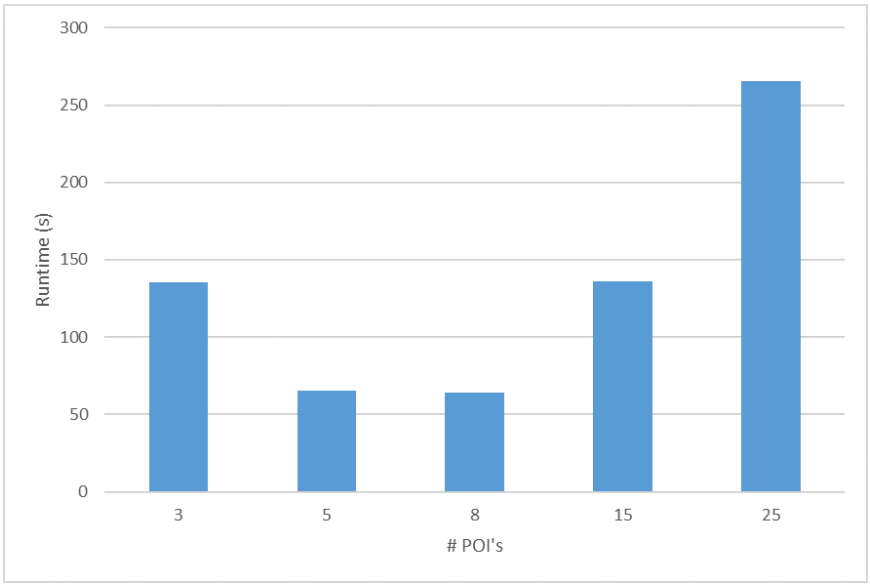


Figure 5.3.9: GA runtime with varying number of POI and two UAV's.

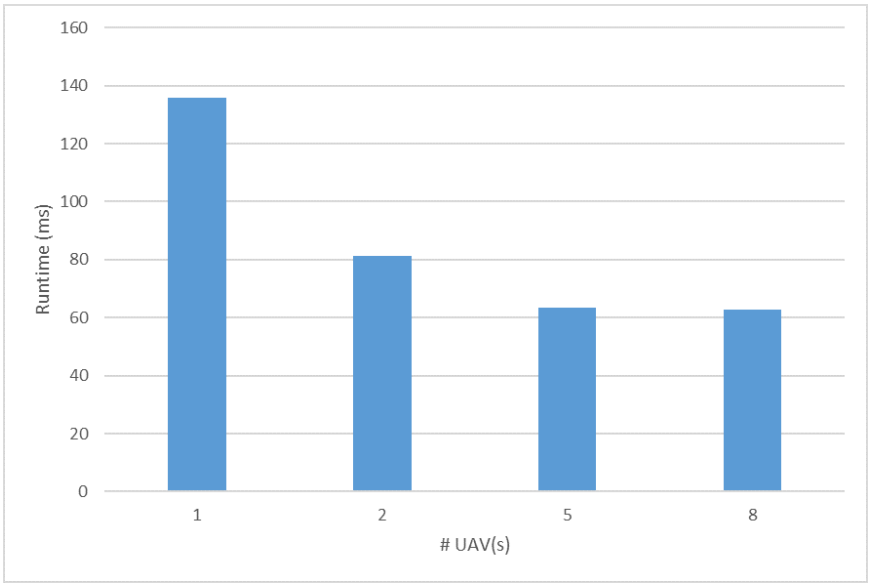


Figure 5.3.10: GA runtime with 15 POI's and a varying number of UAV's.

# 6 Discussion and Conclusion

Previous chapters introduced the multi-UAV path planning problem, provided background on the subject area and presented the method researched and implemented as a solution to this problem. This section serves to provide the reader with a continuation of the previous results chapter, including research contributions, a discussion of limitations, and potential future work. The work is then concluded.

## 6.1 Discussion

### 6.1.1 Research Contributions

This research contributes to the field of multi-UAV path planning computation, which as previously discussed is of growing interest to many. The work presented aims to provide a combined solution in terms of calculating optimized paths and solving for the multi-UAV case. It was desired to have a solution consisting of high-quality paths while achieving fast computation times, even with a high number of POIs and UAVs.

The approach used was two-fold. The usage of the BF to calculate the path costs based on a multi-objective cost function was run. This generated a matrix containing all possible paths and their respective costs between POIs, including the starting point. This data was then utilized by a GA to allocate and solve for the multi-UAV case with the output being a high-resolution path for each UAV. Through the adaptation of the BF algorithm for processing on a CUDA enabled GPU, it was shown to have significant speed-up, allowing in-mission recalculations. The results of this method were then veri-

fied qualitatively through visual inspection of the reconstructed scenarios, and numerically.

It was proven that this method was capable of providing a solution that provided 1) higher quality routing 2) significant overall speed-up through GPU utilization.

### 6.1.2 Limitations

The ability to achieve the demonstrated acceleration was through the usage of a Nvidia GPU, hence the computing platform for this program is limited to one with a CUDA enabled GPU.

Another consideration is that this is algorithm must be run under the assumption that all features are known. Though capabilities for recalculation in-mission are possible given new information, this system is not to be confused with a UAV guidance system which would calculate a trajectory in an unknown environment.

An assumption that was made was that an additional unit responsible for the control of the UAV would be present, and would be capable of following the paths generated by this algorithm.

### 6.1.3 Future Work

Path planning capabilities for autonomous UAVs is ever growing and must address a variety of criteria. The path planner presented in this work includes the necessary components for multi-UAV path planning, however, there is a potential for further development. Due to this planner used for a generic UAV, the cost function was around the concept of reducing factors such as altitude and distance travelled. Future work could involve updating this to reflect a specific UAV model based on flight parameters and characteristics, and this would more accurately balance the cost function components and provide route limitations based on factors such as maximum UAV distance.

Additionally, the inclusion of features such as known danger/keep-out zones could be added to scenarios, allowing for more accurate modelling of the region, and are situations encountered throughout flight navigation. Lastly,

for more practical use, the development of a GUI could be considered allowing for more accessible user interface and input of data such as the desired POIs.

## 6.2 Conclusion

UAV autonomy will continue to be a growing field with advances in technology. This research was undertaken to improve the current state of UAV path planning through the development and testing of an algorithm accelerated on a GPU platform.

This research involved investigating the parallel processing capabilities of a GPU to achieve the necessary runtimes needed by a flight planner algorithm. In this work, the processing of raw data for the use of a path planning algorithm is presented, along with the algorithm itself, consisting of two primary components. By launching the BF SSSP solver using the CUDA platform, it was shown that significant speed-up was achievable due to the nature of the data and algorithm type. The data generated was then used to create the POI connectivity matrix to enable task allocation and distribution for the multi-UAV environment. As a result, this work showed that through the speed-up capabilities of a GPU, an exact algorithm can be used to evaluate and provide highly detailed routing on a known environment, then allocated to multiple UAVs to accomplish an ISR mission scenario with capability for in-mission recalculations.

# Bibliography


[1] "Dilbert at war," *The Economist*, 2014-06-23T18:28:36Z, 2014-06-23T18:28:36Z, ISSN: 0013-0613. [Online]. Available: https://www.economist.com/united-states/2014/06/23/dilbert-at-war (visited on 04/06/2019).

[2] C. Smith. (). On JUSTAS and Justice: Drones in the Canadian Armed Forces, Part One, [Online]. Available: http://natoassociation.ca/on-justas-and-justice-drones-in-the-canadian-armed-forces-part-one/ (visited on 03/27/2019).

[3] (). Stalker XE UAS, [Online]. Available: https://www.lockheedmartin.com/en-us/products/stalker.html (visited on 03/27/2019).

[4] T. Editor. (). Lockheed Martin Indago VTOL Ready for Operations – UAS VISION, [Online]. Available: https://www.uasvision.com/2014/04/30/lockheed-martin-indago-vtol-ready-for-operations/ (visited on 03/27/2019).

[5] David P. Watson and David H. Scheidt, "Autonomous systems," *Johns hopkins ApL TechnicAL DigesT*, vol. 26, no. 4, pp. 368–376, 2005.

[6] Y. Qu, Y. Zhang, and Y. Zhang, "Optimal flight path planning for UAVs in 3-D threat environment," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2014, pp. 149–155. DOI: 10.1109/ICUAS.2014.6842250.

[7] US Department of Defence. (). Unmanned Systems Roadmap 2005-2030, [Online]. Available: https://fas.org/irp/program/collect/uav_roadmap2005.pdf (visited on 03/28/2019).

[8] L. D. Swartzentruber, "Improving path planning of unmanned aerial vehicles in an immersive environment using meta-paths and terrain information," p. 154,

[9] K. P. Valavanis and G. J. Vachtsevanos, "UAV Mission and Path Planning: Introduction," in *Handbook of Unmanned Aerial Vehicles*, K. P. Valavanis and G. J. Vachtsevanos, Eds., Dordrecht: Springer Netherlands, 2015, pp. 1443–1446, ISBN: 978-90-481-9707-1. DOI: 10.1007/978-90-481-9707-1_143. [Online]. Available: https://doi.org/10.1007/978-90-481-9707-1_143 (visited on 04/06/2019).

[10] C. YongBo, M. YueSong, Y. JianQiao, S. XiaoLong, and X. Nuo, "Three-dimensional unmanned aerial vehicle path planning using modified wolf pack search algorithm," *Neurocomputing*, vol. 266, pp. 445–457, Nov. 29, 2017, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.05.059. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217309220 (visited on 01/24/2018).

[11] L. Huang, H. Qu, P. Ji, X. Liu, and Z. Fan, "A novel coordinated path planning method using k-degree smoothing for multi-UAVs," *Applied Soft Computing*, vol. 48, pp. 182–192, Nov. 1, 2016, ISSN: 1568-4946. DOI: 10.1016/j.asoc.2016.06.046. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494616303362 (visited on 01/24/2018).

[12] S. Jung and K. B. Ariyur, "Robustness for large scale UAV autonomous operations," in *2011 IEEE International Systems Conference*, Apr. 2011, pp. 309–314. DOI: 10.1109/SYSCON.2011.5929115.

[13] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, ISSN: 0029-599X. DOI: 10.1007/BF01386390. [Online]. Available: http://dx.doi.org/10.1007/BF01386390 (visited on 03/25/2019).

[14] K. Mehlhorn and P. Sanders, *Algorithms and Data Structures: The Basic Toolbox*, 1st ed. Springer Publishing Company, Incorporated, 2008, ISBN: 978-3-540-77977-3.

[15] F. L. L. Medeiros and J. D. S. da Silva, "A Dijkstra Algorithm for Fixed-Wing UAV Motion Planning Based on Terrain Elevation," in *Advances in Artificial Intelligence – SBIA 2010*, A. C. da Rocha Costa, R. M. Vicari, and F. Tonidandel, Eds., ser. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 213–222, ISBN: 978-3-642-16138-4.

[16] B. M. Sathyaraj, L. C. Jain, A. Finn, and S. Drake, "Multiple UAVs path planning algorithms: A comparative study," *Fuzzy Optimization and Decision Making*, vol. 7, no. 3, p. 257, Jun. 26, 2008, ISSN: 1573-2908. DOI: 10.1007/s10700-008-9035-0. [Online]. Available: https://doi.org/10.1007/s10700-008-9035-0 (visited on 03/25/2019).

[17] D. Palossi, M. Furci, R. Naldi, A. Marongiu, L. Marconi, and L. Benini, "An Energy-efficient Parallel Algorithm for Real-time Near-optimal UAV Path Planning," in *Proceedings of the ACM International Conference on Computing Frontiers*, ser. CF '16, New York, NY, USA: ACM, 2016, pp. 392–397, ISBN: 978-1-4503-4128-8. DOI: 10.1145/2903150.2911712. [Online]. Available: http://doi.acm.org/10.1145/2903150.2911712 (visited on 02/03/2018).

[18] D. Palossi, A. Marongiu, and L. Benini, "On the Accuracy of Near-Optimal GPU-Based Path Planning for UAVs," in *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '17, New York, NY, USA: ACM, 2017, pp. 85–88, ISBN: 978-1-4503-5039-6. DOI: 10.1145/3078659.3079072. [Online]. Available: http://doi.acm.org/10.1145/3078659.3079072 (visited on 02/03/2018).

[19] Peter E. Hart and Bertram Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *ResearchGate*, DOI: http://dx.doi.org/10.1109/TSSC.1968.300136. [Online]. Available: https://www.researchgate.net/publication/3479333_A_Formal_Basis_for_the_Heuristic_Determination_of_Minimum_Cost_Paths (visited on 03/25/2019).

[20] Q. Feng, J. Gao, and X. Deng, "Path planner for UAVs navigation based on A* algorithm incorporating intersection," in *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, Aug. 2016, pp. 2275–2279. DOI: 10.1109/CGNCC.2016.7829147.

[21] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958, ISSN: 0033-569X, 1552-4485. DOI: 10.1090/qam/102435. [Online]. Available: https://www.ams.org/home/page/ (visited on 02/11/2019).

[22] J. C. D. Cruz, G. V. Magwili, J. P. E. Mundo, G. P. B. Gregorio, M. L. L. Lamoca, and J. A. Villaseñor, "Items-mapping and route optimization in a grocery store using Dijkstra's, Bellman-Ford and Floyd-Warshall Algorithms," in *2016 IEEE Region 10 Conference (TENCON)*, Nov. 2016, pp. 243–246. DOI: 10.1109/TENCON.2016.7847998.

[23] W. A. Kamal and R. Samar, "A mission planning approach for UAV applications," in *2008 47th IEEE Conference on Decision and Control*, Dec. 2008, pp. 3101–3106. DOI: 10.1109/CDC.2008.4739187.

[24] C. Zhang, Z. Zhen, D. Wang, and M. Li, "UAV path planning method based on ant colony optimization," in *2010 Chinese Control and Decision Conference*, May 2010, pp. 3790–3792. DOI: 10.1109/CCDC.2010.5498477.

[25] H. Duan, Y. Yu, X. Zhang, and S. Shao, "Three-dimension path planning for UCAV using hybrid meta-heuristic ACO-DE algorithm," *Simulation Modelling Practice and Theory*, Modeling and Simulation for Complex System Development, vol. 18, no. 8, pp. 1104–1115, Sep. 1, 2010, ISSN: 1569-190X. DOI: 10.1016/j.simpat.2009.10.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X09001567 (visited on 04/05/2018).

[26] U. Cekmez, M. Ozsiginan, and O. K. Sahingoz, "Multi colony ant optimization for UAV path planning with obstacle avoidance," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, Jun. 2016, pp. 47–52. DOI: 10.1109/ICUAS.2016.7502621.

[27] L. Dawson and I. Stewart, "Improving Ant Colony Optimization performance on the GPU using CUDA," in *2013 IEEE Congress on Evolutionary Computation*, Jun. 2013, pp. 1901–1908. DOI: 10.1109/CEC.2013.6557791.

[28] S. Sancı and V. İşler, "A Parallel Algorithm for UAV Flight Route Planning on GPU," *International Journal of Parallel Programming*, vol. 39, no. 6, pp. 809–837, Dec. 1, 2011, ISSN: 0885-7458, 1573-7640. DOI: 10.1007/s10766-011-0171-8. [Online]. Available: https://link-springer-com.proxy.queensu.ca/article/10.1007/s10766-011-0171-8 (visited on 02/12/2018).

[29] U. Cekmez, M. OZSIGINAN, M. Aydin, and O. Sahingoz, "UAV Path Planning with Parallel Genetic Algorithms on CUDA Architecture," vol. 1, Jul. 1, 2014.

[30] V. Roberge, M. Tarbouchi, and G. Labonté, "Fast Genetic Algorithm Path Planner for Fixed-Wing Military UAV Using GPU," *IEEE Transactions on Aerospace and Electronic Systems*, vol. PP, no. 99, pp. 1–1, 2018, ISSN: 0018-9251. DOI: 10.1109/TAES.2018.2807558.

[31] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, Nov. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[32] Q. Wang, A. Zhang, and L. Qi, "Three-dimensional path planning for UAV based on improved PSO algorithm," in *The 26th Chinese Control and Decision Conference (2014 CCDC)*, May 2014, pp. 3981–3985. DOI: 10.1109/CCDC.2014.6852877.

[33] V. Roberge, M. Tarbouchi, and G. Labonte, "Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 132–141, Feb. 2013, ISSN: 1551-3203. DOI: 10.1109/TII.2012.2198665.

[34] Y. Choi, H. Jimenez, and D. N. Mavris, "Two-layer obstacle collision avoidance with machine learning for more energy-efficient unmanned aircraft trajectories," *Robotics and Autonomous Systems*, vol. 98, pp. 158–173, Dec. 1, 2017, ISSN: 0921-8890. DOI: 10.1016/j.robot.2017.09.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889015301421 (visited on 02/03/2018).

[35] O. K. Sahingoz, "Flyable path planning for a multi-UAV system with Genetic Algorithms and Bezier curves," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2013, pp. 41–48. DOI: 10.1109/ICUAS.2013.6564672.

[36] O. K. Sahingoz, "Generation of Bezier Curve-Based Flyable Trajectories for Multi-UAV Systems with Parallel Genetic Algorithm," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 1-2, pp. 499–511, Apr. 1, 2014, ISSN: 0921-0296, 1573-0409. DOI: 10.1007/s10846-013-9968-6. [Online]. Available: https://link-springer-com.proxy.queensu.ca/article/10.1007/s10846-013-9968-6 (visited on 02/03/2018).

[37] U. Cekmez, M. Ozsiginan, and O. K. Sahingoz, "Multi-UAV Path Planning with Parallel Genetic Algorithms on CUDA Architecture," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '16 Companion, New York, NY, USA: ACM, 2016, pp. 1079–1086, ISBN: 978-1-4503-4323-7. DOI: 10.1145/2908961.2931679. [Online]. Available: http://doi.acm.org/10.1145/2908961.2931679 (visited on 01/24/2018).

[38] S. Ragi and E. K. P. Chong, "UAV Path Planning in a Dynamic Environment via Partially Observable Markov Decision Process," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 4, pp. 2397–2412, Oct. 2013, ISSN: 0018-9251. DOI: 10.1109/TAES.2013.6621824.

[39] B. Zhang, Z. Mao, W. Liu, J. Liu, and Z. Zheng, "Cooperative and Geometric Learning for path planning of UAVs," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2013, pp. 69–78. DOI: 10.1109/ICUAS.2013.6564675.

[40] G. Labonte, "Simple formulas for the fuel of climbing propeller driven airplanes," *Advances in Aircraft and Spacecraft Science*, vol. 2, pp. 367–389, Oct. 1, 2015. DOI: 10.12989/aas.2015.2.4.367.

[41] H. Luo, Z. Liang, M. Zhu, X. Hu, and G. Wang, "Integrated optimization of unmanned aerial vehicle task allocation and path planning under steady wind," *PLOS ONE*, vol. 13, no. 3, e0194690, 21-Mar-2018, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0194690. [Online]. Available: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0194690 (visited on 04/05/2018).

[42] (). Representing graphs, [Online]. Available: https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs (visited on 03/25/2019).

[43] F. Busato and N. Bombieri, "An Efficient Implementation of the Bellman-Ford Algorithm for Kepler GPU Architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2222–2233, Aug. 2016, ISSN: 1045-9219. DOI: 10.1109/TPDS.2015.2485994.

[44] M. Sedighpour, M. Yousefikhoshbakht, and N. M. Darani, "An Effective Genetic Algorithm for Solving the Multiple Traveling Salesman Problem," p. 8, 2011.

[45] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, ISSN: 1089778X. DOI: 10.1109/4235.996017. [Online]. Available: http://ieeexplore.ieee.org/document/996017/ (visited on 04/08/2019).

[46] J. Li, Q. Sun, M. Zhou, and X. Dai, "A New Multiple Traveling Salesman Problem and Its Genetic Algorithm-Based Solution," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 2013, pp. 627–632. DOI: 10.1109/SMC.2013.112.

[47] M. Kaur, "Particle Swarm Optimization to Solve Multiple Traveling Salesman Problem," vol. 04, no. 07, p. 6, Jul. 2017.

[48] M. d. S. Arantes, J. d. S. Arantes, C. F. M. Toledo, and B. C. Williams, "A Hybrid Multi-Population Genetic Algorithm for UAV Path Planning," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16, New York, NY, USA: ACM, 2016, pp. 853–860, ISBN: 978-1-4503-4206-3. DOI: 10.1145/2908812.2908919. [Online]. Available: http://doi.acm.org/10.1145/2908812.2908919 (visited on 02/03/2018).

[49] (Jul. 2, 2013). CUDA Toolkit, [Online]. Available: https://developer.nvidia.com/cuda-toolkit (visited on 04/06/2018).

[50] D. B. Kirk and W.-m. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Second Edition. Elsevier, Dec. 14, 2012, ISBN: 978-0-12-415992-1.

[51] (). CUDA C Best Practices Guide, [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html (visited on 04/06/2018).

[52] M. Bagherian and A. Alos, "3D UAV trajectory planning using evolutionary algorithms: A comparison study," *The Aeronautical Journal*, vol. 119, no. 1220, pp. 1271–1285, Oct. 2015, ISSN: 0001-9240, 2059-6464. DOI: 10.1017/S0001924000011246. [Online]. Available: https://www.cambridge.org/core/product/identifier/S0001924000011246/type/journal_article (visited on 04/08/2019).

[53] H. Wang, W. Lyu, P. Yao, X. Liang, and C. Liu, "Three-dimensional path planning for unmanned aerial vehicle based on interfered fluid dynamical system," *Chinese Journal of Aeronautics*, vol. 28, no. 1, pp. 229–239, Feb. 1, 2015, ISSN: 1000-9361. DOI: 10.1016/j.cja.2014.12.031. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1000936114002301 (visited on 03/28/2019).