DIFPL
Distributed Drone Flight Path Builder System

Manu Shukla, Ziqian Chen and Chang-Tien Lu
Virginia Tech, Falls Church, Virginia, U.S.A.
{*mashukla, czq, ctlu*}*@vt.edu*

Keywords: Distributed Systems, Drones, Spatial Computing.

Abstract: Drones have become ubiquitous in performing risky and labor intensive areal tasks cheaply and safely. To allow them to be autonomous, their flight plan needs to be pre-built for them. Existing works do not precalculate flight paths but instead focus on navigation through camera based image processing techniques, genetic or geometric algorithms to guide the drone during flight. That makes flight navigation complex and risky. In this paper we present automated flight plan builder *DIFPL* which pre-builds flight plans for drones to survey a large area. The flight plans are built for subregions and fed into drones which allow them to navigate autonomously. DIFPL employs distributed paradigm on Hadoop MapReduce framework. Distribution is achieved by processing sections or subregions in parallel. Experiments performed with network and elevation datasets validate the efficiency of DIFPL in building optimal flight plans.

1 INTRODUCTION

With improvements in technology such as high speed cameras and sensors drones have not only become capable of performing varied tasks but also become increasingly autonomous during flight. Drones have proven very useful in both military battlefield and civilian tasks. Common civilian tasks for drones include education(Krajnik et al., 2011), studying natural phenomena(Williams, 2013), reconnaissance(Segor et al., 2011) and conservation(Koh and Wich, 2012) amongst others. Drones can fly manually through controller or autonomously. If they are controlled manually the cost of operating them increases. Hence it is preferred to operate them autonomously. Autonomous flying presents challenges in terrain navigation. Multitude of flight path scenarios such as variations in altitude and density of objects to survey need to be handled elegantly. Flight planning has to account for drone hardware limitations.

The complexity of covering an area with automated flights increases when there are multiple types of drones available with different capabilities. For this work drones can be one of two types as shown in Figure 1; Conventional drone which performs conventional take off and landing and Quadcopter with vertical take off and landing. The primary challenge encountered in building flight path of drones is in optimizing the use of different types of drones to cover

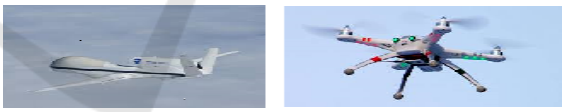

Figure 1: Conventional drone and Quadcopter.

the entire region. Each drone type has its specific limitations. The cost of operating different types of drones is also varied. The optimization problem then becomes multi-pronged. Not only should it cover the entire region with multiple flights of the right drone type that can navigate varying terrain but also use the cheaper drone as frequently as possible to minimize cost. This motivated us to create DIFPL.

DIFPL accounts for terrain scenarios such as network lines length and elevation based climbing angle to optimally divide area into subregions and build the flight plan for each subregion. It generates a set of *Flight Plans* in order to cover the entire power lines network of aviation organization and minimize the number of drone flights and overall cost. This is achieved by optimizing coverage by each flight in a subregion and assigning to the type of drone needed for the subregion. Network lines and elevation of waypoints in each subregion need to satisfy rules that are represented as *autonomy* and *climbing angle* constraints. The constraints determine if the subregion needs to be shrunk or expanded or split between multiple drone types. Our technique uses sectionwise or subregion based distribution of network lines process-

Shukla M., Chen Z. and Lu C..
DIFPL - Distributed Drone Flight Path Builder System.
DOI: 10.5220/0005378700170026
In *Proceedings of the 1st International Conference on Geographical Information Systems Theory, Applications and Management* (GISTAM-2015), pages 17-26
ISBN: 978-989-758-099-4

ing, linear inequalities, and spatial index to query elevation around waypoints. It is powerful as it can automate flight path building with only terrain data and pre-known drone hardware limitations. Focus of this work is on drone flights to take pictures of vegetation over electricity poles network. Image analysis during post-processing determines if the vegetation has overgrown over poles and needs trimming. The application can automate flights for many such tasks such as determine flood damage, deforestation, pollution and agricultural activities. The volume of terrain and network lines data for large areas increases rapidly. In order to scale to the large terrain and networks datasets, DIFPL uses distributed paradigm on Hadoop MapReduce framework. The contributions of the paper are:

- **Create Subregions within Overall Area Dynamically.** DIFPL uses a novel way to divide a large area into subregions that can be covered with a single flight of a drone. Processing terrain data by subregion provides flexibility in deciding which type of drone to assign the subregion.
- **Model Terrain Scenarios with Multiple Drone Types and Their Hardware Limits.** This work combines terrain and network data with drone flight constraints for multiple drone types applied as linear inequalities. It maximally exploits capability of each drone type to determine subregion size and type of drone to cover it.
- **Distribute Flight Plan Creation.** Flight plan construction is distributed with MapReduce framework. Distribution eliminates any limitations on size of spatial index on a single node with novel key-value pair based joins and scales horizontally to larger terrains datasets.
- **Optimize Subregions After Distributed Processing.** DIFPL minimizes subregions by merging as many boundary regions as possible after creating flight plans for subregions. This requires aligning incomplete subregions so adjacent ones can be merged optimizing subregions and its effectiveness is validated by experiments.

Section 2 explores research related to this work. Section 3 presents the preliminary design considerations of flight plan builder. In Section 4 the MapReduce framework based DIFPL implementation is described followed by experiments in Section 5. Conclusions are presented in Section 6.

2 RELATED WORK

Research similar to this work can be broken into two categories, previous approaches to automate flight of drones and distributed platforms for general spatial data processing and specific to drones.

Automated Flights. Several ways to automate the flight of drones exist including using sensors(Visse et al., 2011), camera images(Bills and Saxena, 2011), feeding their waypoints as a file(Babaei and Mortazavi, 2010) or automating from the control(Lugo and Zell, 2014). Genetic algorithms have been used to trace flight paths(De Paula Santos et al., 2013) along with ant colony algorithms in 3D route planning(Deng et al., 2013). Optimization algorithms for multi-objective drone route planning have been explored(Li et al., 2013).

Distributed Spatial Operations. With increase in spatial data distributed approaches are increasingly being explored. Spatial data processing using MapReduce is explored(Cary et al., 2009). Techniques for accelerated processing with MapReduce have been proposed(Wang et al., 2010). Distributed spatial operations on Hadoop and SpatialHadoop as Hadoop extension for spatial operations have been explored(Eldawy and Mokbel, 2013). Computational geometry algorithms have been distributed using SpatialHadoop(Eldawy et al., 2013). Distribution of drone data analysis and multiple drone flight coordination tasks has begun to gather momentum(Chmaj and Selvaraj, 2015). Hadoop based platforms that support spatial queries with MapReduce are proposed(Aji et al., 2013).

As far as we know, Previous works did not consider multiple drone types and variations in terrain together in automating flight paths. DIFPL does not rely on images or video to navigate. Flight paths are built offline with terrain and network lines data and do not need to be adjusted dynamically as all the constraints are applied at the time of running the program. Our distributed technique makes 2 passes on the data using standard hadoop constructs and avoids building a large index on a single node by splitting the data into sections or subregions that can be processed independently.

3 PRELIMINARIES

In this section we describe the background information to DIFPL including input and output data, constraints on the hardware and subregion and waypoint construction.

3.1 Data

DIFPL uses inputs (x,y) geocoordinate position of network lines endpoints provided by aviation orga-

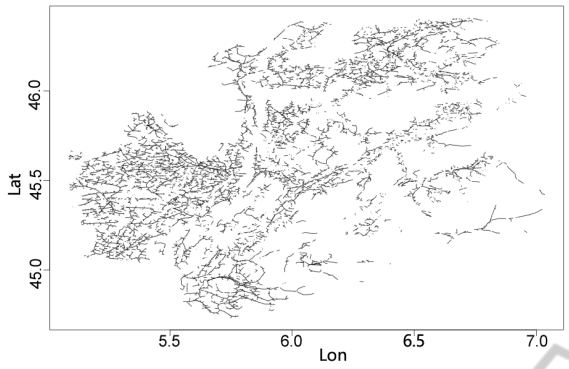

Figure 2: Entire network lines area.

nization and elevation data (x,y,elevation) provided by geographic agency. The elevation points are 25m apart. Output of the program is a set of flight plans, each composed of a set of waypoints (x,y,altitude) and one landing point in KML format to be fed directly into the drone. A separate output file is written for each subregion. The overall 7800 KM^2 power lines network is shown in Figure 2. The elevation points data is filtered for the region for which network lines data is available as part of pre-processing. Every network line had to be covered by one of the available drone types.

3.2 Constraints

The drone characteristics are specific to the type of drone, such as speed, autonomy, turning radius, max slope, and flight height. These characteristics were kept configurable and drone company was given ability to adjust them easily through configuration files. Every pole in network lines need to be photographed at least 4 times. The drone does 2 passes from each side of the lines, one pass in one direction and another pass in the other direction. Drone is equipped with a NEX7 24 Mega pixel camera with 50 mm optical lens. The images are used to perform 3D image reconstruction of each pole to determine if vegetation has overgrown around the pole. Every attempt is made to maximize the use of conventional drones as they are cheaper and more plentiful. Primary limitations of the hardware of drones are:

- **Number of Waypoints.** The hardware of conventional drone can be programmed with up to 200 waypoints and the quadcopter can be programmed with 50 waypoints.
- **Climbing Angle.** Max slope of ascent for conventional drone is 12°while for descent is -16°. For quadcopter the max slope for ascent is 90°and for descent is -90°.
- **Autonomy.** Maximum distance a conventional drone can fly in single flight is 30KM and a quadcopter can fly is 3KM.

The constraints on the flight path of drones are modeled as inequalities. The inequalities are applied for each subregion for the type of drones. The inequalities are defined as follows.

For climbing angle:

$$Max(c_p) \leq C_{type}$$

where c_p is measure of the angle drone has to climb to fly from one waypoint to next along network line and is calculated from recommended drone flying altitude and elevation at the waypoints. The recommended altitude for conventional drone is 100m and for quadcopter is 50m. The maximum weight of the drone can be 2200g. For the waypoints along the sides of the network lines, the elevation is calculated by querying the k nearest neighbor elevation points with a kNN spatial index query and taking their average and ensuring it satisfies the climbing angle constraint.

For autonomy:

$$\sum_{l}(2*d_l+i_l)+3*l*2*\pi*r+t+n \leq A_{type}$$

where d is the distance of each network line, t is the takeoff distance to get to required elevation over first network pole with the climbing angle of each drone type, n is the landing distance with the descent angle for drone type, i distance between two network lines and r is the turn distance for the drone type for l lines. Turning radius of conventional drone is 150m while that of quadcopter is 0m. The distance i is calculated by ordering network lines in the subregion by their x_{start} and calculating the distance between one line to next. Since there are 3 turns for a drone to cover a line segment twice and proceed to the next line segment 3 turning circumferences have to be added to the equation. The requirement of photographing each pole 4 times is satisfied by setting camera to take an image a second. The number of waypoints in output is achieved by collecting waypoints along network lines every 200m for conventional drone and every 100m for quadcopter and increasing it if the number of waypoints exceed the maximum.

4 DISTRIBUTED SYSTEM *DIFPL*

This section describes the distributed system DIFPL that builds flight plans based on distributed paradigm. It gives overview of the architecture of the system, algorithms used and optimized distribution for maximum parallelization.

4.1 Architecture

The architecture of DIFPL is based on distributed paradigm. The distribution approach in DIFPL was implemented using Apache Hadoop MapReduce framework.

Hadoop. Hadoop(Apache and Hadoop, 2014) is an open source framework which facilitates distributed computations on large clusters. A master node orchestrates data storage and computation distribution over multiple slave nodes. Files are uploaded into distributed file storage called HDFS, split into 64MB blocks and then processed. Master node keeps track of all the blocks of a file and where they are stored. MapReduce(Dean and Ghemawat, 2008) allows master node to break down computation tasks into *mappers* and *reducers* distributed over slave nodes. They work on file blocks on slave nodes exploiting colocation of computation with data. Mappers read in input data as key value pairs $<k_1, v_1>$and emit intermediate key value pairs $<k_2, v_2>$. Reducer receive the intermediate key value pairs grouped by k_2 and processes them to generate the final set of key value pairs $<k_3, v_3>$.

The distribution of flight path builder is necessitated due to memory limitations of indexing large elevation and network datasets on a single node. Several opportunities for distribution of the flight plan builder process are available. The identification of quadcopter subregions and the shrinking of quadcopter and conventional drone subregions can be performed in parallel. An overview of the distributed system architecture is shown in Figure 3. The distributed application runs on a cluster on Amazon Web Services (AWS). MapReduce jobs are run on AWS Elastic MapReduce (EMR) and data is read from and written to S3 buckets similar to HDFS. It uses Hadoop 2.5.1 and MapReduce2. The experiments were performed on a 5 node Hadoop cluster with 1 master and 4 slave nodes.

4.2 Algorithms

The algorithms used in DIFPL include a base Build Flight Plans algorithm that performs queries and applies constraints on results within a section or subregion. Two levels of distribution are built, one based on the flight path algorithm and another parallelized on subregions.

4.2.1 Flight Path Building within Section

Algorithm 1 shows the process of building the flight paths. The flight plan building is implemented in 3 steps. The builder reads index file from disk if present else reads the network line and elevation data $\{network_i\}$ and $\{elevation_i\}$ and inserts them into a spatial index Si. The builder then starts querying index with conventional drone default D_c sized subregions as range queries q_i. Objects returned in result set of the query r_i include network lines and elevation points inside the subregion. It calculates waypoints along network lines in query and then elevation of waypoints using kNN query and averaging the elevation of nearest neighbors. If the network lines l_i satisfy the conventional drone autonomy constraint A_c but fail to satisfy the conventional drone climbing angle constraint C_c, then it queries spatial index for the subregion using default size of quadcopter D_q with queries qq_i from left to right. Each consecutive quadcopter size subregion that satisfies the climbing angle constraint is merged with previous one. Ones that do not satisfy C_c are deemed to require quadcopter. If the length of network lines in r_i are too large or less than β% threshold of the autonomy of quadcopter or conventional drone, then the subregion is shrunk or expanded iteratively till it satisfies the autonomy constraint for the respective drone type. The output for each subregion O_i as waypoints and landing point is written out.

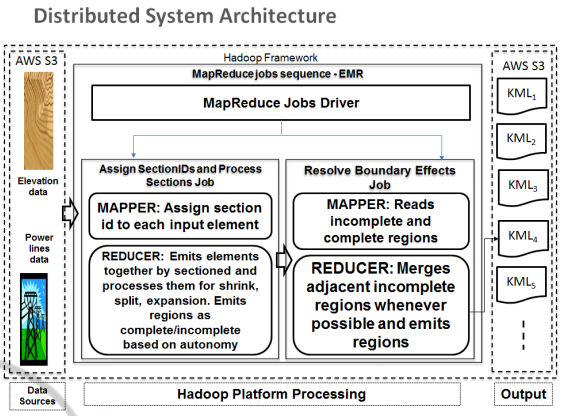

Figure 3: Distributed System Architecture.

4.2.2 Sectionwise Distribution

The first task of distribution is to split the entire area into sections and query for subregions in spatial index built separately for each section. This allows for building smaller sections in spatial index instead of the entire area.

The distribution paradigm is broken down into 2 phases, each translating into a pass over the data. Each pass incrementally identifies subregions for quadcopter or conventional drone and final pass resolves boundary issues and outputs final flight paths for each subregion.

Algorithm 1: Build Fligh Plans.

Input: $\{network_i\}$,$\{elevation_i\}$ {network lines and elevation data}
Output: $\{subregion_i, waypoint_i\}$ {each subregion and waypoints}
1: {**step 0: setup index and subregion iterations**}
2: **if** index spatial index file does not exist **then**
3: Si $\leftarrow$ $\{network_i, elevation_j\}$ {create spatial index with network lines as 2D objects and elevation points}
4: **else**
5: read Si from disk {read in spatial index from disk}
6: **end if**
7: $\forall$ query q_i for subregions $subregion_i$ starting from bottom left of region with size D_c
8: **while** !at top right of region **do**
9: regionDone=false
10: $r_i \leftarrow$ Si(q_i) {query Spatial index Si with query q_i to generate result set}
11: generate $waypoint_i$ in r_i
12: **for all** $waypoint_i$ in r_i **do**
13: $elevation_{wi} \leftarrow \frac{\sum kNN_{waypointi}}{k}$
14: **end for**
15: **if** max C(r_i) $\leq C_c$ **then**
16: {**step 1: Subregion can be covered with conventional drone**}
17: {retrieved objects elevations satisfy conventional drone climbing angle constraint}
18: **while** $\sum\{|\ l_r\ |\}$ does not satisfy A_c **do**
19: {retrieved objects network lines do not satisfy conventional drone autonomy constraints}
20: Si $\leftarrow q_i \pm \gamma$
21: {reduce or expand window size and re-query}
22: **end while**
23: **if** $\sum\{|\ l_r\ |\}$ satisfies A_c **then**
24: {conventional drone constraints satisfied}
25: $\{waypoint_i\} \leftarrow C_q$ {create waypoints for conventional drone subregion}
26: write output O_i
27: $q_i \rightarrow q_j$ {move to next window}
28: **end if**
29: **else**
30: {**step 2: Subregion needs quadcopter**}
31: **for all** $qq_i \in q_i$ **do**
32: $r_i \leftarrow$ Si(qq_i)
33: {requery with default quadcopter subregion sizes}
34: **if** !maxC(r_i) $\leq C_c$ **then**
35: {region fails conventional drone climbing angle constraints}
36: $\{waypoint_i\} \leftarrow Q_q$
37: {build waypoints for quadcopter}
38: $qq_i \rightarrow qq_j$
39: {move to next window}
40: **else**
41: $O_i \cup\ q_{i-1}$ {merge into previous conventional subregion}
42: Write Output O_i
43: **end if**
44: **end for**
45: **end if**
46: **end while**

Assign and Process Section. The first pass labels network and elevation data with the section they belong to. The details are described in Algorithm 2. The data is processed by two mappers. First mapper reads network lines data $\{network_i\}$ as text and calculates section id based on the coordinates of the network line. The key value pair emitted from the mapper are $<sectionID_j, networkline_l>$. The second mapper similarly reads elevation data $\{elevation_j\}$ and calculates section id and emits <K,V>pairs $<sectionID_j, elevation_i>$. The reducer reads the data and aggregates all elevation and network line observations for a section id.

Algorithm 2: Assign and Process Section.

Input: $\{elevation_i, networkline_i\}$
Output: $subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j,$ quadcopter|conventional,$\{network_j\}$,$\{elevation_j\}$, $\{waypoint_j\}$
1: *mapper1*:
2: calculate $sectionID_j$ based on its coordinates
3: emit:$sectionID_j \rightarrow elevation_j$
4: *mapper2*:
5: calculate $sectionID_j$ based on its coordinates
6: emit:$sectionID_j \rightarrow networkline_j$
7: *reducer*:
8: **for all** observations **do**
9: build $section_j$
10: **end for**
11: *APPLY Build Flight Plans Algorithm*
12: emit:$subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j,$ quadcopter|conventional,$\{network_j\}$,$\{elevation_j\}$, $\{waypoint_j\}$

Reducer builds spatial index for elevation points for the subregion with network and elevation data in memory and marks quadcopter and conventional subregion by elevation and length constraints. It shrinks, expands and merges subregions as needed. The final emitted <K,V>pairs in the reducer are the details of each subregion within a section such as subregion id $subregion_j$, the subregion extent xsr_j,ysr_j,xer_j,yer_j, a flag indicating if its a quadcopter or conventional drone subregion, the network lines and elevation points and waypoints for a quadcopter or conventional drone to follow along the network lines in the subregion $waypoint_k$,...,$waypoint_l$.

Resolve Edge Effects. Second pass resolves edge effects between sections as described in Algorithm 3. The input to the mapper are the subregions $subregion_j$ with their extent, the drone type needed, network lines and elevation points in the subregion and the waypoints. For the subregions that are along the vertical edge of the sections associated with their section id, reducer pairs the corresponding left and right subregions. The subregions could be merged together if they are covered by same drone type. The output from

Algorithm 3: Resolve Edge Effects.

Input: $\{subregion_j \rightarrow \{xsr_j,ysr_j,xer_j,yer_j,$ quadcopter|conventional,$\{waypoint_j\},\{network_j\},$ $\{elevation_j\}\}\}$
Output: $subregion_k \rightarrow \{xsr_j,ysr_j,xer_j,yer_j,$quadcopter |conventional, $\{waypoint_k\}\}$
1: *mapper*:
2: emit:"verticalborder|notverticalborder" $\rightarrow subregion_j$, $\{xsr_j,ysr_j,xer_j,yer_j,\{network_j\},\{elevation_j\}$ $\{waypoint_k\},\}$
3: *reducer*:
4: **if** border **then**
5: **if** $subregion_i$ && $subregion_j$ are adjacent **then**
6: **if** they can be merged with combined subregion network length <β% **then**
7: merge subregions
8: emit:$subregion_k \rightarrow \{xsr_k,ysr_k,xer_k,yer_k,$ quadcopter|conventional,$\{waypoint_k\}\}$
9: **else**
10: emit:$subregion_i \rightarrow \{xsr_i,ysr_i,xer_i,yer_i,$ quadcopter|conventional,$\{waypoint_i\}\}$
11: emit:$subregion_j \rightarrow \{xsr_j,ysr_j,xer_j,yer_j,$ quadcopter|conventional,$\{waypoint_j\}\}$
12: **end if**
13: **end if**
14: **else**
15: emit:$subregion_j \rightarrow \{xsr_j,ysr_j,xer_j,yer_j,$ quadcopter|conventional,$\{waypoint_j\}\}$
16: **end if**

the reducer are the entire set of subregions that include merged subregions. All non boundary subregions are emitted as is.

4.3 Optimized Distribution

A more scalable approach which precludes need to build spatial indexes in each reducer in order to build waypoints in a subregion is now discussed with network lines passed in raw form and elevations inserted in spatial index.

Assign and Process Subregion. This algorithm receives as input network and elevation data as $\{elevation_i\},\{networkline_i\}$ and in mapper emits them with the key $subregion_j$. Mapper emitted pairs $subregion_j,networkline_j$ and $subregion_j,elevation_j$ then result in network lines and elevation points landing together in the reducer. In reducer the waypoints are built along the network lines and climbing angle constraint C_c checked by querying elevation points from subregion near waypoint from spatial index using kNN query. It then emits all subregions with waypoints and a flag indicating if they are complete or incomplete. The details are shown in Algorithm 4.

If the autonomy constraint is not satisfied such that the network lines are too long or less than threshold β% of the drone type autonomy, then we mark the subregion at the time of emission from reducer as

Algorithm 4: Assign and Process Subregions.

Input: $\{elevation_i,networkline_i\}$
Output: $subregion_j \rightarrow xsr_j,ysr_j,xer_j,yer_j,$ $\{waypoint_j\},$ quadcopter |conventional, complete |incomplete, $\{elevation_j\},\{networkline_j\}$
1: *mapper1*:
2: emit:$subregion_j \rightarrow elevation_j$
3: *mapper2*:
4: emit:$subregion_j \rightarrow networkline_j$
5: *reducer*:
6: index $\{elevation_i\}$ in spatial index
7: **for all** observations calculate $waypoints_j$ based on its network lines **do**
8: **for all** $waypoint_i$ calculate elevation with kNN query **do**
9: compute climbing angles and check if they satisfy constraint C_c
10: **if** climbing angle fails constraint **then**
11: split subregion into quadcopter subregions
12: progressively apply C_c on each subregion of size D_q
13: **if** satisfied **then**
14: **if** autonomy constraint A_c satisfied **then**
15: merge with previous if autonomy constraint satisfied
16: **else**
17: mark as quadcopter
18: **end if**
19: **end if**
20: emit:$subregion_j \rightarrow xsr_j,ysr_j,xer_j,yer_j,$ $\{waypoint_j\},$ quadcopter |conventional, complete |incomplete, $\{elevation_j\},\{networkline_j\}$
21: **else if** autonomy constraint A_c not satisfied **then**
22: shrink $subregion_j$
23: emit:$subregion_j \rightarrow xsr_j,ysr_j,xer_j,yer_j,$ $\{waypoint_j\},$ quadcopter |conventional, complete |incomplete, $\{elevation_j\},\{networkline_j\}$
24: **end if**
25: **end for**
26: **end for**

incomplete. The task then becomes to merge the adjacent incomplete subregions.

Reconcile Adjacent Subregions. Subregions in sparse area that need to be expanded, or ones generated after shrinking dense subregion are emitted as incomplete. All subregions that are split result in a set of quadcopter and conventional drone subregions. The ones that are not able to satisfy the autonomy β% constraint are also emitted by reducer as incomplete to merge with adjacent incomplete subregions. The details are shown in Algorithm 5. The algorithm accepts all subregions and mapper emits ones that are incomplete. The reducer then aligns the ones that are adjacent. It then checks if the adjacent subregions can be merged together. Every time a merged subregion satisfies β% threshold and autonomy constraint

Algorithm 5: Reconcile Adjacent Subregions.

Input: $subregion_j \rightarrow xsr_j, ysr_j, xer_j, yer_j$, $\{waypoint_j\}$, quadcopter |conventional, complete |incomplete,$\{elevation_j\}$,$\{networkline_j\}$

Output: $subregion_i$,$xer_i, yer_i, xsr_i, xsr_i$,$\{waypoint_i\}$, conventional|quadcopter

1: *mapper*:
2: emit:"incomplete" $\rightarrow$ $subregion_i$,$xer_i, yer_i, xsr_i, xsr_i$, $\{elevation_i\}$,$\{networkline_i\}$,incomplete,$\{waypoint_i\}$, conventional|quadcopter
3: emit:$subregion_j \rightarrow xer_i, yer_i, xsr_i, xsr_i$, $\{elevation_i\}$,$\{networkline_i\}$,complete,$\{waypoint_i\}$, conventional|quadcopter
4: *reducer*:
5: **for all** incomplete subregions calculate new subregions based on its coordinates **do**
6: **if** $subregion_i$ && $subregion_j$ are adjacent **then**
7: **if** they can be merged with combined subregion network length <β% **then**
8: merge subregions
9: emit:$subregion_k \rightarrow \{xsr_k, ysr_k, xer_k, yer_k$, quadcopter|conventional,$\{waypoint_k\}\}$
10: **else**
11: emit:$subregion_i \rightarrow \{xsr_i, ysr_i, xer_i, yer_i$, quadcopter|conventional,$\{waypoint_i\}\}$
12: emit:$subregion_j \rightarrow \{xsr_j, ysr_j, xer_j, yer_j$, quadcopter|conventional,$\{waypoint_j\}\}$
13: **end if**
14: **end if**
15: **for all** complete subregions **do**
16: emit:$subregion_i \rightarrow \{xsr_i, ysr_i, xer_i, yer_i$, quadcopter|conventional,$\{waypoint_k\}\}$
17: **end for**
18: **end for**

it emits it in reducer as $subregion_i \rightarrow \{xsr_i, ysr_i, xer_i, yer_i$,$\{waypoint_i\}$,quadcopter|conventional$\}$ and proceeds to the next.

5 EXPERIMENTS

This section explores the scenarios to determine a subregion and the type of drone flights to cover it. The scenarios can be divided into 3 categories. They can be within a section including the entire area on a single node or distributed by subregion.

5.1 Scenarios within a Single Section

These scenarios occur when applying queries and constraints within a single section which can either be the whole area or a section.

Use of Conventional Drone. A typical flight path as built by Flight Plan Builder is shown in Figure 4. The waypoints for the subregion are highlighted along with the electric pole network lines they have to cover which are shown in red. The distance δ of waypoints

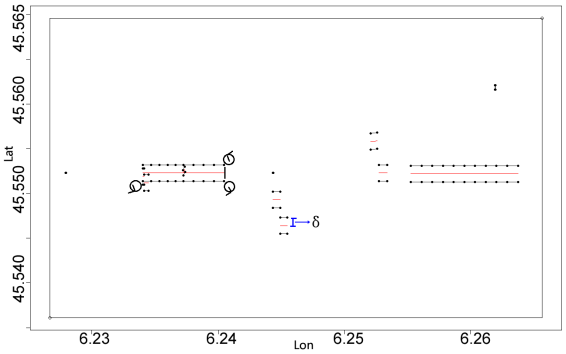

Figure 4: Waypoints along network lines δ distance from line for subregion covered by conventional drone.

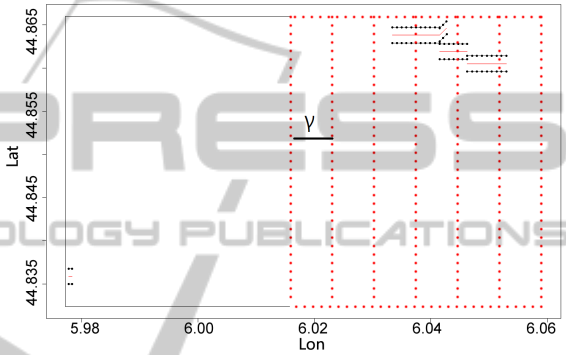

Figure 5: Subregion with network lines length below threshold of conventional drone autonomy and expanded.

path from the network line is constant and configurable. The turning radius of drone along with flight from one network line to another is taken into account in the *autonomy* constraint but not shown in figure or entered as waypoint as drone automatically determines how it will navigate from one point to next. The circles represent the turns conventional drone has to make to fly on both sides of a network line.

Expanding Subregion for Conventional Drone or Quadcopter. Figure 5 shows the subregion which has network lines less than β% autonomy of a conventional drone. The size of the subregion is incrementally expanded by length γ until autonomy reaches β% of constraint or higher. This parameter is kept configurable and defaults to 80%. The dotted lines in figure represent the incremental expanding of subregion.

Splitting Subregion between Conventional Drone and Quadcopter Flights. If a subregion fails to satisfy the climbing angle constraint for a conventional drone it is split to cover segments where constraint fails with a quadcopter. Figure 6 shows the flight path waypoints for a subregion that can not be covered by a conventional drone and requiring a quadcopter. A larger subregion with this split is shown in Figure 7. Quadcopter default size subregions are incre-

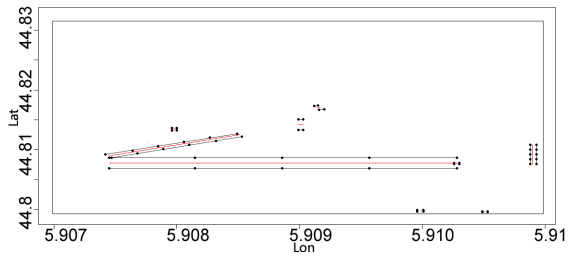

Figure 6: Waypoints along network lines δ distance from line for subregion covered by quadcopter.

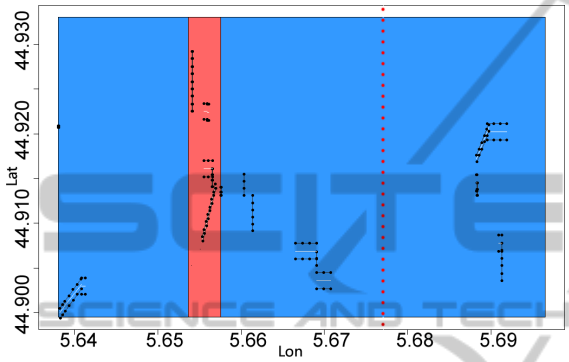

Figure 7: Subregion that fails to satisfy climbing angle constraint and split between conventional drone in blue and quadcopter in red.

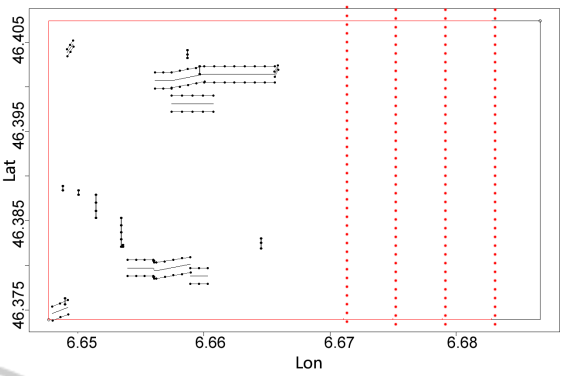

Figure 8: Subregion with network lines too long for conventional drone autonomy and needs to be shrunk.

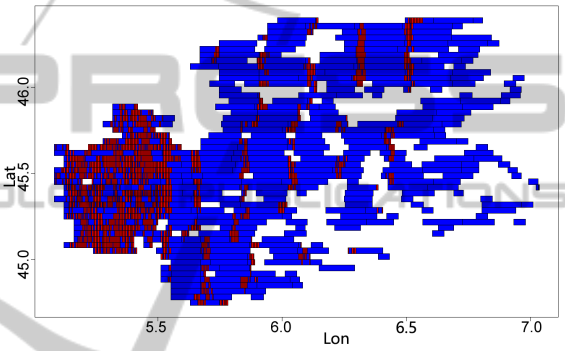

Figure 9: All the subregions in overall area shown in blue for conventional drone and red for quadcopter.

mentally applied to determine where the conventional drone climbing angle constraint fails. If that subregion lies in the middle of the original subregion, it is covered with quadcopter and all the other subregions before it are assigned to minimal number of conventional drone flights. Subregions after the last quadcopter segment are merged with the subsequent conventional drone subregion indicated by dotted line. Only one subregion which required quadcopter coverage was found so after that subregion algorithm resumes querying with default conventional drone subregion size. The quadcopter default sized subregions before the quadcopter assigned subregion are merged together into one conventional drone subregion.

Shrinking Subregion for Conventional Drone or Quadcopter. Subregions which has network lines too long for a conventional drone or quadcopter to cover in one flight necessitates shrinking of subregion. Figure 8 shows incrementally shrinking of subregion by horizontal length γ until a size that satisfies the distance constraint for conventional drone is reached. The shrinking reduces the length of network lines until it can be covered by a conventional drone. The querying then resumes from the end of shrunk subregion. The dotted lines represent incrementally shrunk subregions.

After experimenting with several default subregion sizes for both types of drones starting default subregion sizes of 3.7KMx2.9KM for conventional drone and 3.7KMx0.29KM for quadcopter serve as good default sizes. The defaults are optimal size to utilize the autonomy of the conventional drones and quadcopter and minimize processing time. Figure 9 shows the subregions in the overall area with some subregions covered by conventional drone and some by quadcopter and some subregions being shrunk for conventional drone and some split for quadcopter and the remaining segment covered in the following subregion. The subregions in the figure match well the network lines and their density shown in Figure 2. Dense network lines areas have more default size and shrunk conventional drone subregions. Larger number of quadcopter subregions indicate altitude variations. Sparse network line areas has larger number of expanded conventional drone subregions.

The optimizations allow for far fewer drone subregions overall and minimized number of quadcopter flights which is the goal as conventional drone are cheaper and more plentiful. The incremental optimizations impact is shown in Figure 10. The incremental savings with expansions and splitting of con-

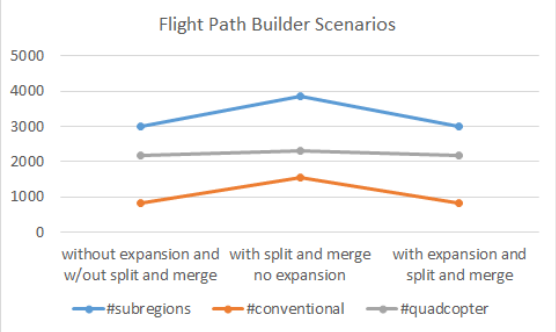

Figure 10: All the subregions for conventional drone and for quadcopter broken down by the analysis type.

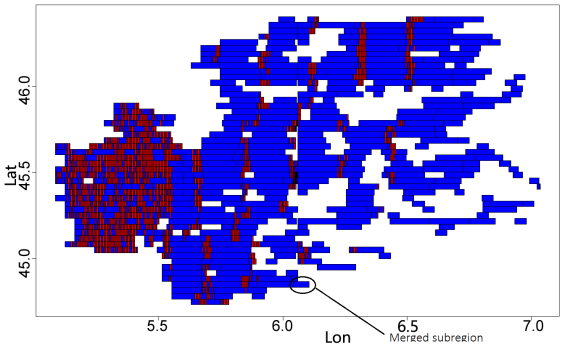

Figure 11: All the subregions in overall area shown in blue for conventional drone and red for quadcopter after merging subregions across the vertical section boundaries.

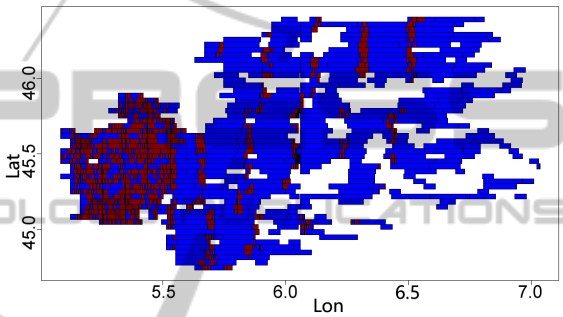

Figure 12: All the subregions in overall area shown in blue for conventional drone and red for quadcopter after subregionwise distributed processing and merge.

ventional default sized subregions into minimal number of quadcopter and conventional drone subregions is very effective in minimizing subregions, in particular quadcopter subregions. The bulk of efficiency comes from expanding subregions reducing the number of conventional drone subregions from 1553 to 809. The splitting of a conventional drone subregion into pinpoint quadcopter and efficiently merged conventional drone subregions minimizes the number of quadcopter subregions.

5.2 Distributed Scenarios

In distributed paradigm, the scenarios change as subregions on the edges of sections can not expand which as then resolved in an additional step.

Shrink, Expand, Merge or Split Subregions. The subregions are shrunk and expanded based on the same threshold β% of autonomy as used in sequential algorithm for each section. This invariably means several subregions on the right edge of each section do not get to meet the threshold as they run out of room to expand. This issue is mitigated to some extent by the edge effect job.

Merge Section Edge Subregions. Edge subregions for each section are merged together if feasible. The result after merging in a 4-section distribution are shown in Figure 11. If the subregions are being covered by the same drone type on either side of an edge and they have not been expanded they are likely to be merged together. Experiments find 4 such subregions.

5.3 Optimized Distributed Scenarios

Optimized distribution focuses on processing each subregion in parallel. The subregions are then reconciled by merging with adjacent subregions.

Splitting or Shrinking Subregion. Since a subregion by itself can not be expanded, it can only be shrunk if its network lines length exceeds the autonomy of a conventional drone or quadcopter. We mark the subregions that do not satisfy the β% network line length constraint after splitting or shrinking as incomplete. Subegions that satisfy the length constraint are emitted as complete.

Merging Subregions. All incomplete subregions are then ordered and adjacent ones that can be merged together are emitted as new subregions. Figure 12 shows the subregions created by the subregion parallelization. The entire set of subregions that can be processed together are merged if possible in a single pass.

Running the scenarios in the distributed paradigm produces results that are similar to the sequential processing results in terms of number of subregions of quadcopter or conventional drone type. Merging reduces quadcopter subregions by 627 and conventional subregions by 385.

Comparisons of single node run with distributed and optimized distributed results are shown in Figure 13. It clearly shows the impact of merge on reducing the count of quadcopter and conventional drone subregions in optimized distribution. After merge there are 1239 conventional drone subregions and 1839

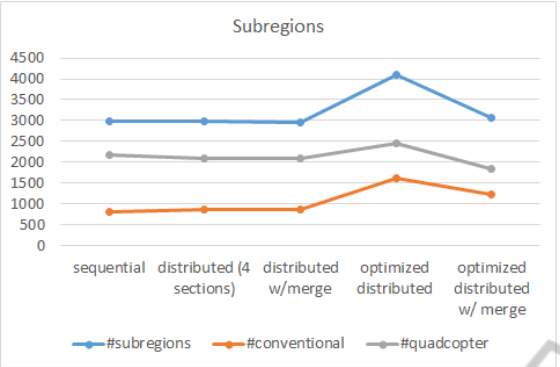

Figure 13: All the subregions for conventional drone and for quadcopter broken down by the analysis type.

quadcopter subregions which compares well with 809 conventional drone subregions and 2181 quadcopter subregions in single node execution.

6 CONCLUSIONS

DIFPL uses a novel approach to flexibly divide a large area into subregions and dynamically adjust them to optimally cover with a single drone flight. It combines spatial data and drone limitations or constraints modeled as linear inequalities to automate flight path of drones. The distributed implementation presents way to handle large datasets which can not be processed on a single node. The subregion level distribution allows horizontal scalability. The flight plans produced by distributed version are similar in numbers to the ones by single node implementation but generated more efficiently. The technique used is not only useful for the task of surveying power lines but extensible to a host of other drone applications.

REFERENCES

Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., and Saltz, J. (2013). Hadoop gis: A high performance spatial data warehousing system over mapreduce. *Proc. VLDB Endow.*, 6(11):1009–1020.

Apache and Hadoop (2014). http://hadoop.apache.org. *Apache Hadoop*.

Babaei, A. R. and Mortazavi, M. (2010). Three-dimensional curvature-constrained trajectory planning based on in-flight waypoints. *Journal of Aircraft*.

Bills, C.and Chen, J. and Saxena, A. (2011). Autonomous mav flight in indoor environments using single image perspective cues. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*.

Cary, A., Sun, Z., Hristidis, V., and Rishe, N. (2009). Experiences on processing spatial data with mapreduce. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, SSDBM 2009, pages 302–319, Berlin, Heidelberg. Springer-Verlag.

Chmaj, G. and Selvaraj, H. (2015). Distributed processing applications for uav/drones: A survey. In Selvaraj, H., Zydek, D., and Chmaj, G., editors, *Progress in Systems Engineering*, volume 1089 of *Advances in Intelligent Systems and Computing*, pages 449–454. Springer International Publishing.

De Paula Santos, G., Garcia Marques, L., Miranda Neto, M., Cardoso, A., Lamounier, E., and Yamanaka, K. (2013). Development of a genetic algorithm to improve a uav route tracer applied to a man-in-the-loop flight simulator. In *Virtual and Augmented Reality (SVR), 2013 XV Symposium on*, pages 284–287.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Deng, T., Xiong, Z. M., and Meng, Y. J. L. Q. Z. (2013). Research on 3d route planning for uav in low-altitude penetration based on improved ant colony algorithm. *Applied Mechanics and Materials*, 442:556–561.

Eldawy, A., Li, Y., Mokbel, M. F., and Janardan, R. (2013). Cg_hadoop: Computational geometry in mapreduce. In *ACM SIGSPATIAL*, pages 294–303.

Eldawy, A. and Mokbel, M. F. (2013). A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. *Proc. VLDB Endow.*, 6(12):1230–1233.

Koh, L. P. and Wich, S. A. (2012). Dawn of drone ecology: lowcost autonomous aerial vehicles for conservation. *Tropical Conservation Science*, 5(2):121–132.

Krajnik, T., Vonasek, V., Fiser, D., and Faigl, J. (2011). *AR* drone as a platform for robotic research and education. *Communications in Computer and Information Science*, 161:172–186.

Li, L., Liu, X., Peng, Z.-R., and Xu, X. (2013). Multi-objective optimization model and evolutionary algorithm to plan uav cruise route for road traffic surveillance. In *Transportation Research Board 92nd Annual Meeting*.

Lugo, J. J. and Zell, A. (2014). Framework for autonomous onboard navigation with the ar.drone. *Journal of Intelligent and Robotic Systems*, 73(1-4):401–412.

Segor, F., Bürkle, A., Kollmann, M., and Schönbein, R. (2011). Instantaneous autonomous aerial reconnaissance for civil applications. In *ICONS 2011, The Sixth International Conference on Systems*, St. Maarten, The Netherlands Antilles.

Visse, A., Dijkshoorn, N., van der Veen, M., and Jurriaans, R. (2011). Closing the gap between simulation and reality in the sensor and motion models of an autonomous ar.drone. In *Proceedings of the International Micro Air Vehicles conference*.

Wang, K., Han, J., Tu, B., Dai, J., Zhou, W., and Song, X. (2010). Accelerating spatial data processing with mapreduce. In *Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems*, ICPADS '10, pages 229–236, Washington, DC, USA. IEEE Computer Society.

Williams, S. C. P. (2013). Studying volcanic eruptions with aerial drones. In *Proc Natl Acad Sci U S A*.