Search for Underutilized Airspace for Extensible Traffic Management Operations Based on Air Traffic Patterns

Jinhua Li[1]
NASA Langley Research Center, New Hampton, Virginia, 23681, U.S.A.

Min Xue[2] and Paul U. Lee[3]
NASA Ames Research Center, Moffett Field, CA, 94035, U.S.A.

Priyank Pradeep[4]
University Space Research Association, Moffett Field, CA, 94035, U.S.A.

This paper presents a new method to facilitate integrating new vehicle traffic operations, operating with existing air traffic operations. The extensible traffic management (xTM) concept assumes that the new vehicles can operate in a dedicated Cooperative Area (CA) with minimal interaction with conventional air traffic and requiring minimal air traffic supervision. Our method assumes that a new xTM CA can be created when an underutilized airspace with little or no traffic can be identified. Our approach involves modeling airspace as a tree data structure and iteratively subdividing it into smaller cells, with underutilized airspace defined as any cells without flight tracks. The benefits of our approach include its applicability to all xTM scenarios, the ability to handle both 2D and 3D space using a unique tree data structure, and computational efficiency for key functions such as space decomposition, labeling of connected cells, and searching of cells containing a given point. By automatically searching for underutilized airspace based on operating air traffic patterns, we can optimize airspace utilization and improve air traffic management. Our proposed approach can quantitatively determine when and where to allow xTM operations in the National Airspace System.

I. Introduction

The anticipated increase in the number of new vehicle types accessing the U.S. airspace, such as small Unmanned Aircraft System (UAS), Vertical Take-off and Landing (VTOL) aircraft, High Altitude Long Endurance (HALE) balloons, solar-powered airplanes, and supersonic transport poses unique challenges for the National Airspace System (NAS). To address these challenges, NASA has been exploring the federated, automated, and cooperative air traffic management concepts under the heading eXtensible Traffic Management (xTM) [1].

Currently, the NAS is categorized into controlled, uncontrolled, special use, and other airspace based on air traffic activities. Initially, xTM operations are expected to be conducted in dedicated Cooperative Area (CA) for new vehicle types, away from conventional aircraft, to minimize the burden on air traffic controllers and ensure safety. However, as traffic density increases, it may become necessary to integrate these operations with existing air traffic operations while minimizing their interactions [2, 3]. As a first step towards integrated airspace operations, this paper proposes

[1] Research Computer Engineer, Crew Systems and Aviation Operations Branch, jinhua.li@nasa.gov
[2] Aerospace Engineer, Aviation Systems Division, min.xue@nasa.gov.
[3] Research Engineer, Human Machine Systems Division, paul.u.lee@nasa.gov.
[4] Research Scientist, Aviation Systems Division, Priyank.pradeep@nasa.gov

a novel approach to dynamically identify regions of underutilized airspace that are commonly assigned to conventional air traffic, to be repurposed and authorized for xTM operations.

The proposed method for the identification of underutilized airspace leverages well-known techniques from computer graphics. First, air traffic data is processed into spatial data points. Second, the airspace is modeled using a unique tree data structure called octree, which recursively subdivides the three-dimensional airspace into small cells. Third, small cells without flight track data points are grouped into blocks of underutilized airspace. The proposed approach is generic in its approach and computationally efficient, and therefore has the potential to be a useful genera-purpose tool that can be applied for all types of vehicles across different xTM operations.

The paper is organized as follows: Section II provides an overview of the quadtree and octree data structures. Section III presents the search algorithm for identifying underutilized airspace. Finally, Section IV presents the results, demonstrating the new model and search algorithm using actual air traffic data.

II. Quadtree and Octree

A quadtree is a tree data structure in which each node has exactly four child nodes. A region quadtree is a type of quadtree that represents a partition of space by dividing the region into four equal quadrants. A point region (PR) quadtree is a region quadtree that stores a list of points that exist within the cell of a leaf. Quadtree has been widely used in computer graphics and image processing. It has also been applied to Urban Air Mobility (UAM) flight planning [4].

Spatial data, also called geospatial data, refers to a type of data that references a specific geographical area or location. Geometric data is a spatial data type that is mapped on a two-dimensional flat surface. Given a set of spatial or geometric data points in a space, a PR quadtree decomposition recursively subdivides the space into four equal-sized sub-cells until each cell contains no more than one point. The stop criteria can be extended by limiting the maximum number of points in a cell, the maximum tree depth, or both. Figure 1 depicts the quadtree data structure and decomposition process. An octree is a three-dimensional analog of a quadtree in which each node has eight child nodes.

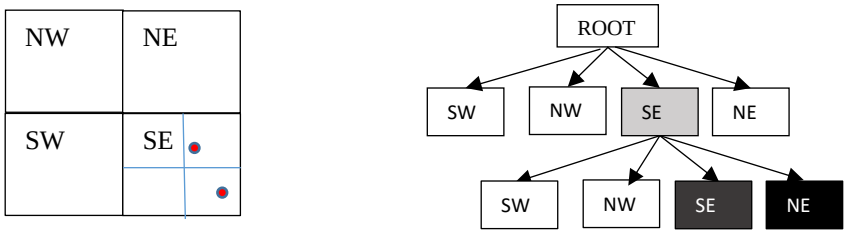


Figure 1. Quadtree node and decomposition

Compared with the other tree data structures, such as point quadtree and R-tree [5], the PR quadtree is chosen in this paper for three reasons:

(1) The airspace partition is equally subdivided into small cuboid cells without overlapping.

(2) The partition location does not depend on the order of data points. Additionally, the partition location is robust against standard data deviations, i.e., small deviations of data point locations generally do not change the partition location.

(3) The quadtree is easy to implement.

However, there are also some known disadvantages of PR quadtree:

(1) Closely spaced points may require many tree levels to split into small cells. It has been proven that the depth of a quadtree is at most $\log_2(s/c) + 3/2$ where c is the smallest distance between any two points and s is the side length of the initial space [5]. To avoid overly splitting the airspace, we set the maximum tree depth as one of the stop criteria.

(2) The volume of a single leaf cell containing a data point can be large, particularly in three dimensions. However, such volume can provide an additional safety buffer from an operational perspective.

(3) Octree can use a lot of memory space. The space complexity of octree is $O(8^D)$ and $O(N)$ in the worst case, where D is the tree depth and N is the number of data points.

Given a quadtree and an octree, there are three basic operations as follows:

***OP1 - SEARCH*:**
GIVEN AN OBJECT, SEARCH FOR THE LEAF NODE THAT CONTAINS THE OBJECT.
OP2 - FIND NEIGHBOR NODES:
GIVEN A NODE, FIND ITS IMMEDIATE NEIGHBOR NODES.
***OP3 - LABEL*:**
GIVEN A NODE, FIND ALL CONNECTED NODES.

The first tree operation OP1 can be implemented using a standard tree depth search. For the second tree operation OP2, each node has neighbor nodes in four directions, as illustrated in Figure 2. We implemented a two-step method to find all the neighbor nodes [7]. Furthermore, we can extend OP2 to find all connected leaf nodes as OP3 using a graph search method. Figure 2 shows an example of searching and labeling using a simple quadtree with randomly generated data points. Please refer to the Appendix for the detailed data structures and algorithms.

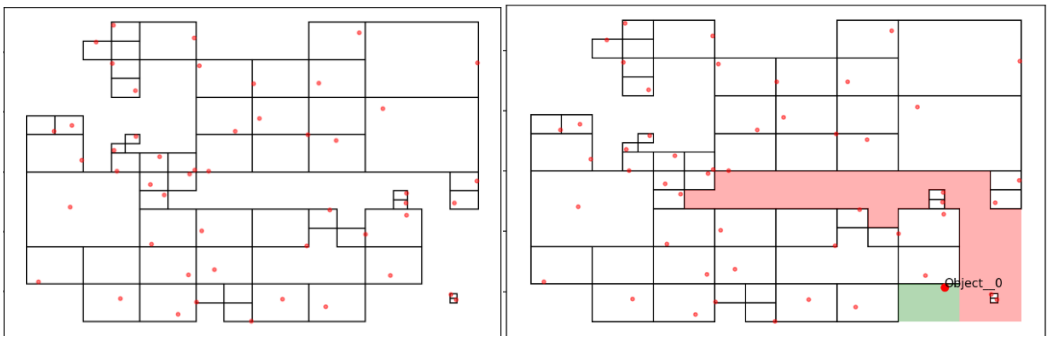


Figure 2. A simple quadtree with randomly generated data points (left). Quadtree searching and labeling, in which the green cell is a leaf cell that contains the object *object_0* and pink cells are all connected unoccupied cells (right).

Finally, given N as the number of spatial data points, and D as tree depth, the time complexity of basic quadtree operations is as follows [3]:

***LEMMA* 1.**
THE AVERAGE AND WORST TIME COMPLEXITY OF CONSTRUCTING A QUADTREE IS $O(N \log_2 N)$ AND $O(N^2)$ USING RECURSIVE SUBDIVIDING METHOD.
***LEMMA* 2.**
THE AVERAGE AND WORST TIME COMPLEXITY OF QUADTREE SEARCH IS $O(D)$ AND $O(N)$
***LEMMA* 3.**
THE WORST-CASE TIME COMPLEXITY OF TWO-STEP FINDING NEIGHBORING NODE METHOD IS $O(D)$

***LEMMA* 4.**
TO LIMIT THE SIDE LENGTH OF ANY LEAF CELL EQUAL OR LESS THAN *A*, THE MAXIMUM TREE DEPTH MUST SATISFY $D \geq \log_2(S/A)$, WHERE D IS TREE DEPTH AND S IS THE SIDE LENGTH OF ROOT CELL OR INITIAL SPACE.

Lemma 4 provides a lower bound of tree depth if we want to limit the cell size to be no greater than a threshold. The proof is given in the Appendix. The quadtree construction and basic operations can be extended to octree in three-dimensional space, as illustrated in Figures 3-5.

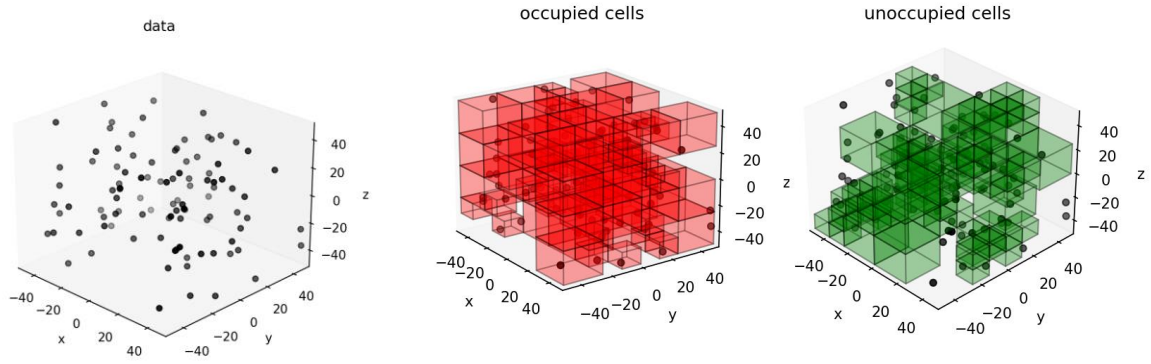


Figure 3. A simple octree example with randomly generated data points. Red denotes occupied cells; green denotes unoccupied cells.

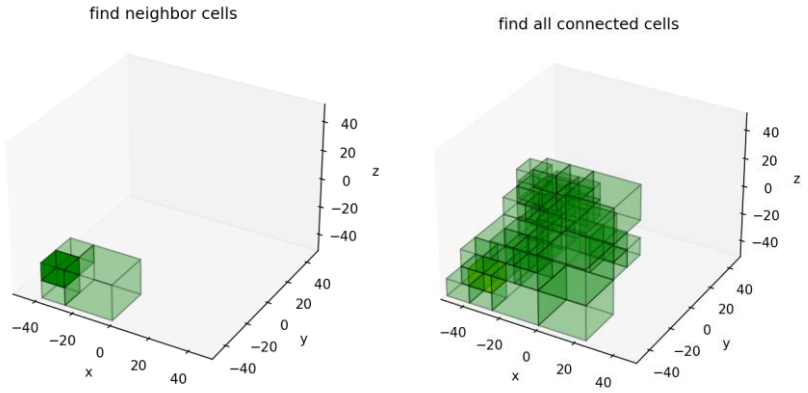


Figure 4. Octree search and find neighbor nodes (left) and labeling (right)

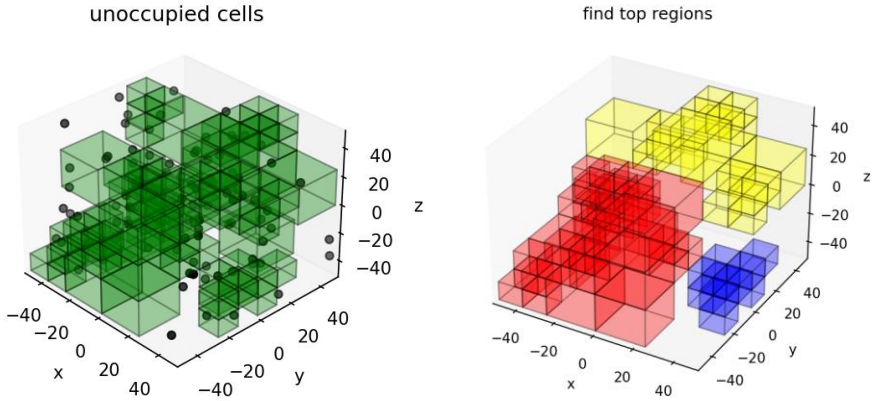


Figure 5. Ranking the connected airspace by size. The three largest connected airspace regions are shown as red, yellow, and blue in descending order of size.

III. Search Algorithm for Underutilized Airspace

Figure 6 shows the search algorithm for underutilized airspace workflow. First, flight track positions are converted into Cartesian coordinates. Next, quadtrees and octrees are constructed by recursively subdividing the space into small cells. Third, the unoccupied cells are labeled, grouped into large cells as underutilized airspace, and ranked. Lastly, the coordinates are converted back into the geospatial coordinates.

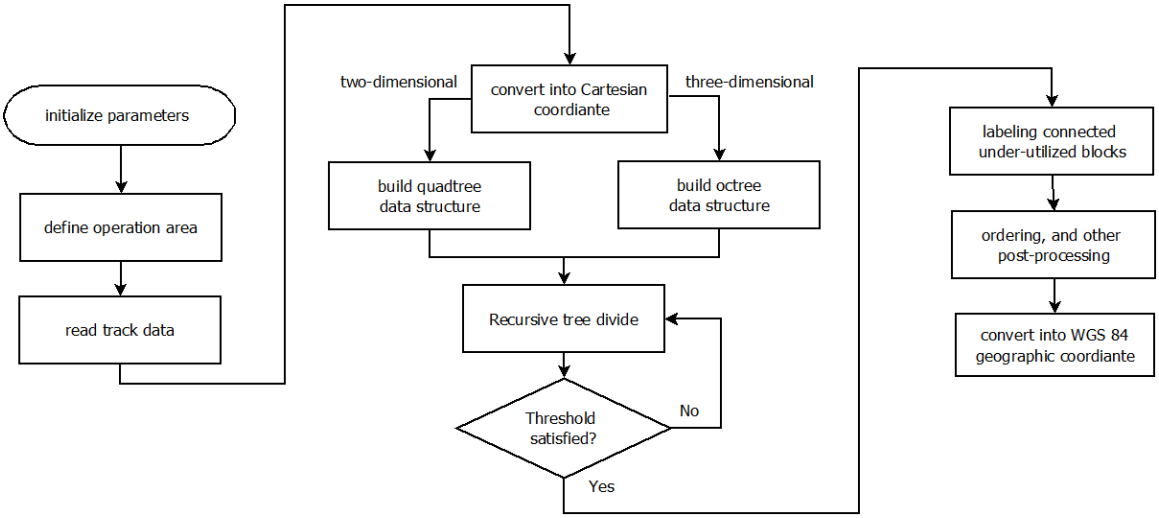


Figure 6. Search algorithm for underutilized airspace workflow

IV.Examples

In this section, we evaluate the effectiveness of the proposed search algorithm for underutilized airspace using actual air traffic data. Our study focused on an area near Fort Sumner Airport (KFSU) in New Mexico, where NASA operates a permanent scientific balloon launch site. The quadtree and octree were constructed using actual flight track data as spatial data points. Our sample data consisted of 24-hour recorded flight tracks from September 15, 2021, covering a 40-mile by 40-mile square region centered on the NASA balloon launch site. Additionally, we divided the airspace vertically into three layers based on the airspace class, as shown in Figure 7. These layers included: (a) Class E airspace from ground level up to 18,000 ft; (b) Class A airspace from 18,000 to 60000 ft; and (c) upper-Class E airspace above 60,000 ft. Depending on the traffic density, each layer may be further subdivided into additional sublayers.

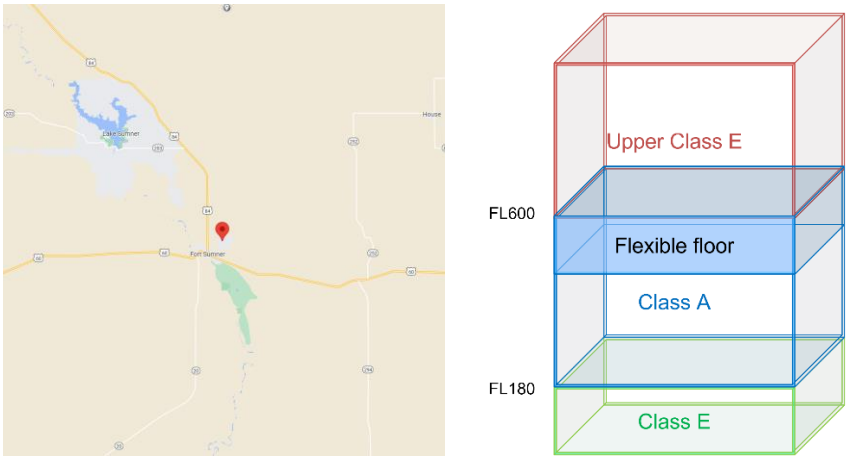


Figure 7. 40-mile by 40-mile study area at Fort Sumner Airport in New Mexico. The flexible floor is a proposed concept by the FAA for aircraft in upper-Class E airspace to use the upper part of Class A airspace

A. Upper Class E airspace

The traffic sample had no recorded air traffic in the upper-Class E airspace. Therefore, the entire upper Class E airspace was identified as underutilized airspace without use of the algorithm. However, it is anticipated that in the 2030 timeframe, this upper Class E airspace may become more active with vehicles like HALE solar airplane, airships, balloons, and supersonic transport [1][6]. In that event, the proposed method would be more relevant for upper Class E airspace.

B. Class E airspace

Figures 8 and 9 shows the 24-hour aggregated flight tracks with the identified underutilized airspace using quadtree in two-dimensional space and octree in three-dimensional space. Only a small amount of air traffic was observed over the 24-hour period because Class E airspace is mainly used by small aircraft flying under Visual Flight Rules (VFR). Class E airspace is not typically used by commercial airline operators. As a result, the algorithm identified a sizable area of Class E airspace as underutilized, as shown in the right plot of Figures 8 and 9. For example, an underutilized airspace of this size could be used for balloon launch, small UAS operations, or Vertical TakeOff and Landing (VTOL) aircraft operations. Note that occupied airspace and unoccupied airspace are complement to each other.

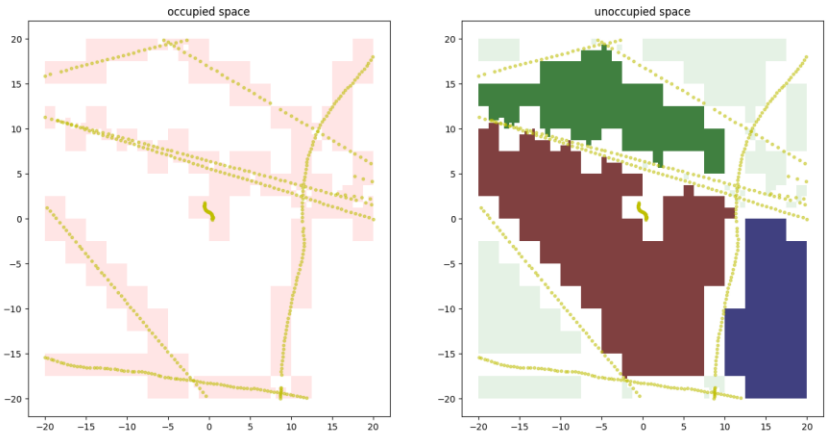
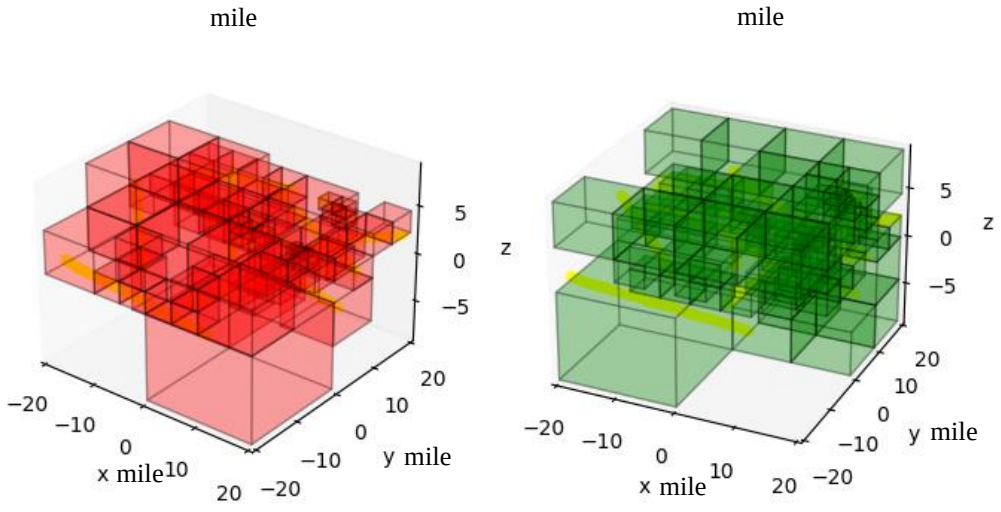


Figure 8. Underutilized airspace identified below 18,000ft using quadtree



(a) occupied airspace (b) unoccupied/underutilized airspace

Figure 9. Underutilized airspace identified below 18,000ft using octree. Red cells represent occupied airspace. Green cells represent unoccupied or underutilized airspace (i.e., the complement of the red cells). Yellow dots represent flight track data over a 24-hour period.

C. Class A airspace

Class A airspace is the primary domain for commercial aviation operations. Figure 10 shows the underutilized airspace identified by the proposed algorithm based on the same 24-hour sample of air traffic data. First, the quadtree method projected all 24 hours of air traffic data from Class A airspace onto a planar space. Due to high air traffic density, the algorithm detected no underutilized airspace using a quadtree. In future work, it may be worthwhile to consider relaxing definition of underutilized airspace to be "fewer than a given number of track data" (greater than one). Either way, identifying such airspace and using it for xTM operations would require closing that airspace to conventional air traffic or allowing mixed operations.

Second, the octree method was applied uses three-dimensional data set. As a result, the algorithm detected under-utilized airspace distributed spatially, as shown in the right plot of Figure 10. In future work, we can validate if any given trajectory may potentially conflict with the existing air traffic by checking if any part of the trajectory crosses the occupied airspace. Moreover, we can build flight paths to a designated flight levels like the upper-Class E airspace from the ground levels for applications such as space launch operations as long as the path stays within underutilized airspace..

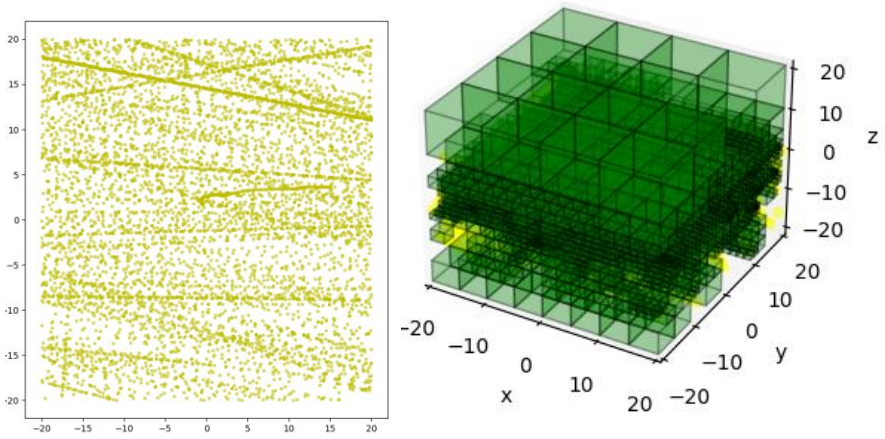


Figure 10. Underutilized airspace identified between 18,000ft and 60,000ft using quadtree (left) and octree (right).

Lastly, as an example, we consider a special use case called flexible floor. Flexible floor is a concept proposed by the FAA to use part of the upper Class A airspace to be accessed by upper-Class E air traffic [6]. For example, a HALE solar airplane may want to descend below 60,000ft at night or in low sunlight conditions to reserve its energy, subsequently ascending out of upper-Class A and back into upper-Class E when daylight returns. The algorithms identify the entire airspace as underutilized because the sample data does not record traffic between 50,000 and 60,000. As a result, the entire Class A airspace between 50,000 and 60,000 ft can be used as the flexible floor. Next, we apply the search algorithm to the airspace between 45,000 and 50,000 ft. We also divide the 24 hours into three 8-hour periods for illustration. The results are shown in Figure 10. We found that the search algorithm was able to dynamically extract underutilized airspace to be used as flexible floor. In future work, we can quantitatively analyze the underutilized airspace regions.

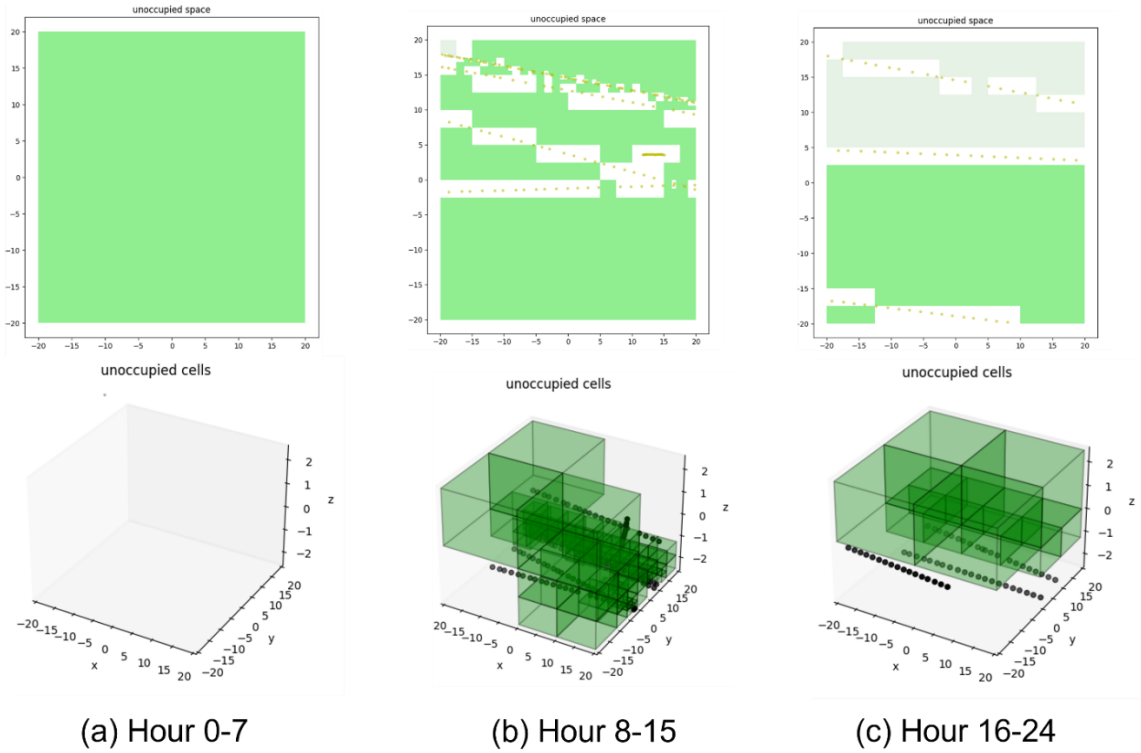


Figure 11. Underutilized airspace identified between 45,000ft and 50,000ft using quadtree (top) and octree (bottom).

V. Conclusion

A novel approach was presented to identify underutilized airspace based on existing air traffic patterns. The method using octree is generic to the data type and computationally efficient. The algorithm was demonstrated using air traffic data gathered over 24 consecutive hours around Fort Sumner Airport in New Mexico. The algorithm successfully identified underutilized airspace that could potentially have been used for extensible traffic management (xTM) operations. Future work may include: (1) developing path planning algorithms for various xTM operations, such as space launches and small UAS and UAM operations near the ground based on the identified underutilized airspace; and (2) identifing underutilized airspace for special use like upper-Class E traffic management's flexible floor, UAS traffic management's low altitude authorization and notification capability (LAANC), and xTM cooperative area.

References

[1] Jung, J., Rios, J., Min, X., Homola, J. and Lee, P. U., "Overview of NASA's Extensible Traffic Management (xTM) Research", AIAA SciTech Forum, San Diego, CA, Jan. 2022

[2] Lee, P.U., Chartrand, R., Oseguera-Hohr, R., Brasil, C., Bakowski, D., Gaberial, C., and Evans, M., "Identifying Common Use Case Across Extensible Traffic Management (xTM) for Interactions with Air Traffic Controllers", AIAA SciTech Forum, San Diego, Jan. 2022

[3] Lee, P.U., Brasil, C., Bakowski, D., Gaberial, C., Evans, M., and Chartrand, R. "Identifying Common Coordination Procedures Across Extensible Traffic Management (xTM) to Integrate xTM Operations into the National Airspace Systems", AIAA Aviation Forum, Chicago, Jun. 2022

[4] Min, X. and Wei M., "Small Unmaned Aerial Vehicle Flight Planning in Urban Environments", Journal of Aerospace Information Systems, Vol.18, No. 10, Oct. 2021

[5] Samet, H., "Foundations of MultiDimensional and Metric Data Structures", Morgan Kaufmann, 2006

[6] Bradford, S., "Upper Class E Traffic Management (ETM) Concept of Operations (version 1.0)", FAA, 2020

[7] Samet, H., "Neighbor finding in images represented by octrees", Computer Vision, Graphics, and Image Processing, volume 46, issue 3, page 367-386, Jun. 1989

Appendix

1. Quadtree and Octree data structure

Table 1. Octree and Quadtree Data Structure

QUADTREE/OCTREE NODE DATA STRUCTURE	QUADTREE/OCTREE DATA STRUCTURE
PROPERTIES – VARIABLE NAME: [VARIABLE TYPE] DESCRIPTION	
1. *centroid*: [float] centroid position of the node 2. *width, length, (height)*: [float] width, length, (height for octree) of the node 3. *objs*: [list] objects stored in the node 4. *depth*: [integer] tree depth of the node 5. *children*: [list] of all child nodes in NW, NE, SE, and SW. (8 child nodes for octree) 6. isLeafNode: [bool] is the node a leaf node? 7. neighbor: [list] all neighbor nodes 8. parent: [node] parent node 9. name: [string] a unique identifier	1. *root*: [node] root node 2. *objs*: [list] objects stored in the tree
METHODS – METHOD NAME (INPUT VARIABLE): DESCRIPTION	
1. *find_neighbors*(): return all immediate neighborhood nodes;	1. *find*(object): return node that contains the object; return None if the object does not exist 2. subdivide (): construct a quadtree or octree by calling recursive subdivide method 3. *_recursive_subdivide*(node): recursively subdivide a tree node

2. Two-step method to find neighbor nodes

Step 1: find the neighbor nodes that are greater or equal in size to the given node

For a given node, it has neighbor nodes in four directions, as shown in Figure 12. In each direction, there can be two cases.

- In the first case, the neighbor node shares the same parent node with the given node.
- In the second case, the neighbor node is a child node of the given node's parent node's neighbor node of greater or equal size in that direction.

For example, in Figure 13,

- If the given node is node 1's SW child, then it is evident that the north neighbor of the given node is node 1's NW child.
- If the given node is node 1's NW child, the north neighbor of the given node can be in two conditions, as illustrated in Figure 13:
    - (a) if the given node's parent (Node 1)'s north neighbor node of greater or equal size (Node 2) has no child, then the north neighbor of greater or equal size of the given node is Node 2.
    - (b) if Node 2 has a child, then the north neighbor of greater or equal size of the given node is Node 2's SW child.

Step 2: find the neighbor nodes that are smaller in sizes to the given node

The candidate neighbor nodes of smaller size are the child nodes of the neighbor node of greater or equal size from Step 1.

The two-step method above can be applied to octree.

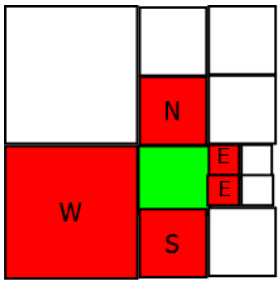
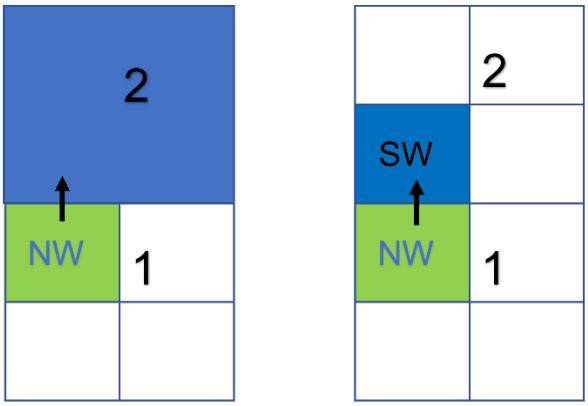


Figure 12. A quadtree node (green) with its immediate neighbor nodes (red). The neighbor nodes in the west, south, and north directions are equal or greater in size than the given node, and the neighbor nodes in the east direction are smaller in size.



(a) Node 2 is a leaf node (b) Node 2 is an internal node

Figure 13. Reference the NW child (green) of node 1, panel (a) illustrates a North neighbor (i.e., node 2 (blue)) that is greater in size; panel (b) illustrates a North neighbor (i.e., SW child of node 2) that is equal in size.

ALGORITHM 1: ALGORITHM TO FIND ALL IMMEDIATE NEIGHBORHOOD NODES OF A QUADTREE NODE

FUNCTION NAME: **FIND_NEIGHBORS**

INPUT: A NODE **NODE**

OUTPUT: A LIST OF NODES THAT IS NEIGHBORED TO THE NODE **ANS**

1. FOR EACH DIRECTION **D** IN NORTH, WEST, SOUTH, EAST:
    a. CALL FUNCTION **_FIND_NEIGHBORS_GERATER_OR_EQUAL_SIZE**(NODE, D), RETURN A LIST OF NEIGHBOR NODES **NEIGHBORS_GE** THAT HAVE A GREATER OR EQUAL SIZE THAN THE NODE IN DIRECTION D,
    b. CALL FUNCTION **_FIND_NEIGHBORS_SMALLER_SIZE**(NODE, D, **NEIGHBORS_GE**) GIVEN THE OUTPUT LIST OF NODES FROM STEP 2, RETURN A LIST OF ALL NEIGHBOR NODES **NEIGHBORS_LG** IN DIRECTION D
    c. APPEND NODES IN **NEIGHBORS_L** TO LIST **ANS**
2. RETURN ANS

ALGORITHM 1.A: ALGORITHM TO FIND IMMEDIATE NEIGHBORHOOD NODES OF A QUADTREE NODE THAT HAVE GREATER OR EQUAL SIZE

FUNCTION NAME: **_FIND_NEIGHBORS_GREATER_OR_EQUAL_SIZE**

INPUT: A NODE **NODE** AND DIRECTION **D**

OUTPUT: A LIST OF NEIGHBOR NODES THAT HAVE A GREATER OR EQUAL SIZE **NEIGHBOR_GE**

1. IF **NODE** IS A ROOT NODE, RETURN **NONE**
2. IF NEIGHBOR NODE **NEIGHBOR_GE** AND **NODE** HAVE THE SAME PARENT, RETURN THE **NODE**'S PARENT'S CHILD NODE AT THE MATCHING DIRECTION (REFER TO FIGURE 1 AND FIGURE 3). FOR EXAMPLE, IF **NODE** IS IN THE SW CELL, THEN THE NEIGHBOR NODE OF GREATER OR EQUAL SIZE IN THE SOUTH DIRECTION IS THE NW CELL OF THE SAME PARENT, I.E. RETURN NODE.PARENT.CHILDREN.NW
3. ELSE: RECURSIVELY CALL FUNCTION **_FIND_NEIGHBORS_GERATER_OR_EQUAL_SIZE**(NODE.PARENT, D), RETURN TO **NEIGHBOR_GE**

ALGORITHM 1.B: ALGORITHM TO FIND IMMEDIATE NEIGHBORHOOD NODES OF A NODE THAT HAVE SMALLER SIZE

FUNCTION NAME: **_FIND_NEIGHBORS_SMALLER_SIZE**

INPUT: A NODE **NODE**, DIRECTION **D** AND A LIST OF NEIGHBOR NODES WITH GREATER OR EQUAL SIZE FROM ALGORITHM 1.A **NEIGHBORS_GE**

OUTPUT: A LIST OF NEIGHBOR NODES **NEIGHBORS**

1. INITIALIZE LIST **CANDIDATES** = [NEIGHBORS_GE], **NEIGHBORS** = []
2. WHILE LIST **CANDIDATES** IS NOT EMPTY:
    a. POP THE FIRST ELEMENT **CANDIDATES**[0]
    b. IF **CANDIDATES**[0] IS A LEAF NODE, APPEND **CANDIDATES**[0] TO **NEIGHBORS**
    c. ELSE: APPEND THE CHILD NODE OF CANDIDATES[0] IN THE OPPOSITE DIRECTION TO **CANDIDATES**. (REFER TO FIGURE 1). FOR EXAMPLE, IF DIRECTION **D** IS NORTH, ADD CELLS IN BOTH SW AND SE TO **CANDIDATES;** IF DIRECTION D IS SOUTH, ADD CELLS IN BOTH NW AND NE TO **CANDIDATES**
3. RETURN **NEIGHBORS**

ALGORITHM 2: ALGORITHM TO FIND ALL NEIGHBORHOOD NODES OF A QUADTREE NODE

FUNCTION NAME: **FIND_CONNECTED_NODES**

INPUT: A NODE **ROOT**

OUTPUT: A LIST OF NODES THAT IS CONNECTED TO THE NODE **ANS**

1. INITIALIZED LIST **ANS** = [], LIST **QUE** = [], AND SET **VISITED** = SET([ROOT.NAME])
2. WHILE QUE IS NOT EMPTY:
    3. POP THE FIRST **QUE** ELEMENT TO **NODE**
    4. CALL **FIND_NEIGHBORS** RETURN TO LIST **NEIGHBORS**
    5. FOR EACH **NODE** IN **NEIGHBORS:**
        6. IF **NODE.NAME** IS NOT IN **VISITED:**
            7. ADD **NODE.NAME** TO **VISITED**
            8. APPEND **NODE** TO **ANS**
            9. APPEND **NODE** TO **QUE**
10. RETURN **ANS**

3. Proof of Lemma 4

LEMMA 4.

TO LIMIT THE SIDE LENGTH OF ANY LEAF CELL EQUAL OR LESS THAN *A*, THE MAXIMUM TREE DEPTH MUST SATISFY $D \geq \log_2(S/A)$, WHERE D IS TREE DEPTH AND S IS THE SIDE LENGTH OF ROOT CELL OR INITIAL SPACE.

Proof: Given the size length of node at depth D is $\frac{S}{2^D}$ [3],

$$\frac{S}{2^D} \leq A \rightarrow 2^D \geq \frac{S}{A} \rightarrow D \geq \log_2 \sqrt{\frac{S}{A}}$$ □