An Efficient Strategic Deconfliction Algorithm for Lane-Based Large-Scale UAV Flight Planning

Thomas C. Henderson, David Sacharny and Michael Cline
University of Utah

UUCS-19-005

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

3 September 2019

Abstract

Given a lane-based airway navigation framework wherein each lane is one-way, and intersections are handled by means of polygonal lane roundabouts, then it is possible to assign flight plans so that the set of all such plans is strategically deconflicted. That is, no two Unmanned Aerial Systems (UAS's) will ever get closer in a lane than the minimum allowed headway time (or distance) of each other. We describe here a method to determine all allowable launch times (i.e., strategically deconflicted) given a requested launch time interval and a set of scheduled flights. Scheduling a new flight is linear complexity in the number of scheduled flights.

1 Introduction

Currently, NASA has proposed a UAS deconfliction strategy that requires service providers (UAS Service Suppliers or USS's) to exchange full flight path information and to mutually find a deconflicted set of flights. This approach has high complexity and sacrifices UAS operator privacy. We propose a lane-based deconfliction strategy which reduces the shared information to be simply lane entry and exit times and UAV speed through the lane. Then given a requested launch time interval, it is possible to determine the set of all allowable (deconflicted) time intervals within the requested interval.

2 Strategic Deconfliction Algorithm

Figure 1 represents the airway corridors (lanes) between two ground locations. Launch and land

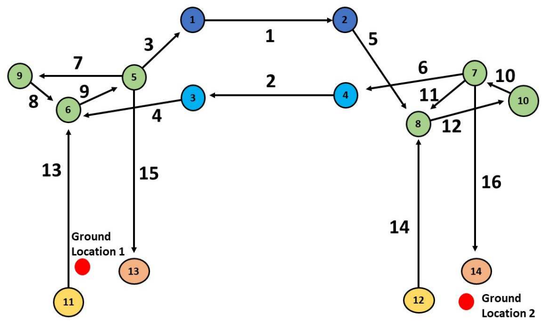

Figure 1: The Lanes (and Vertexes) for the Two Ground Locations Case.

nodes exist for both locations in this example; these are nodes 11 and 13, and 12 and 14, respectively, and vertical lanes exist between these and the roundabout. Nodes 5, 6, and 9 lie on a circle above the first ground location and form a (polygonal) roundabout. Lane 1 (going from node 1 to node 2) provides a way from Ground Location 1 to 2, while Lane 2 provides a return corridor. A flight from Ground Location 1 to 2 follows the sequence of lanes: 13, 9, 3, 1, 5, 12, 10, and 16, and can be viewed as a polyline. In this example, Lane 1 is at altitude 534 feet, Lane 2 is at 467 feet, and the roundabouts are at 500 feet. These may be set to other values as desired by the system designers. An airway lane constrains the trajectory of the UAS to the center-line of the airway, referred to as the longitudinal direction of the aircraft trajectory in prior research (e.g. [3]). The vertical and lateral directions are assumed to be under control to remain inside the lane. Uncertain altitude and lateral movements should be compensated for in the design of the width and height of the airway; this is a subject of ongoing research. The critical aspect of this formulation is that there are no crossing-conflicts. In previous work [6] we gave a discrete time slot algorithm, whereas here the solution is over continuous intervals.

To better utilize intersections, only merging or diverging conflicts should exist because crossing conflicts require that the scheduler manage nodes as well as segments. This would add additional constraints on UASs requesting time within an intersection that would be independent otherwise. Since each segment is defined by exactly one schedule that manages UAS arrivals, organizing the airspace in this way removes the need for intersection management such as the signalized intersections in [2]. In Figure 2, the node labeled “2” is an example of a diverging conflict, where incoming traffic is split into two traffic streams [4]. The node labeled “1” is an example of a merging conflict, where two traffic streams are joined into one [4]. Crossing conflicts may be eliminated by implementing a roundabout, a concept borrowed from ground traffic engineering [4]. Figure 2 displays the graph model for a roundabout, which includes unidirectional edges between eight nodes (each node represents the endpoint of a segment) in a counter-clockwise direction.

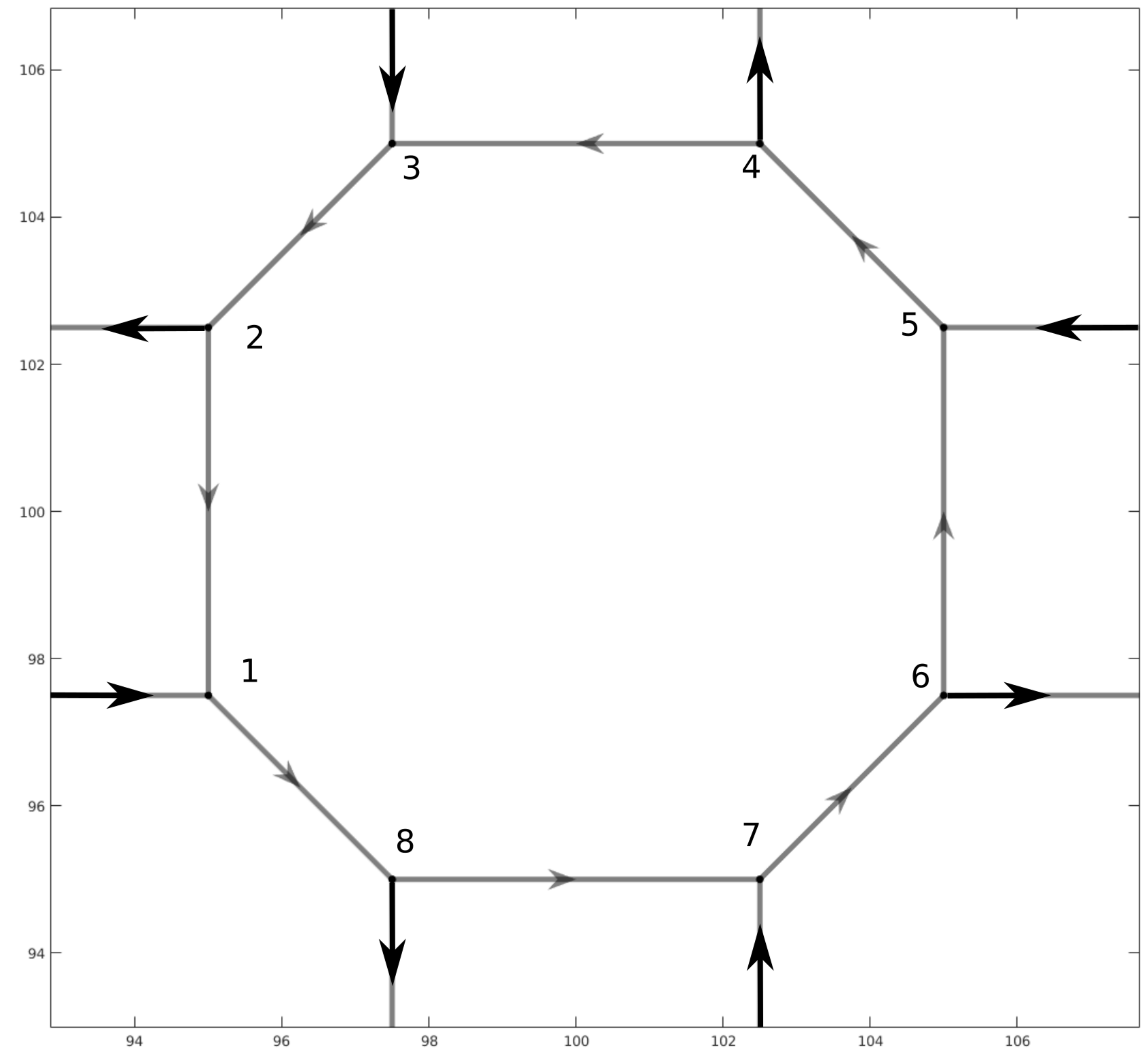

Figure 2: Airway Roundabout.

The primary safety concern is to schedule flights so that no two UAS's are ever closer than the minimal specified headway time (we can also plan using headway distance). On the other hand, optimal resource utilization requires packing as many flights as possible into the lanes. Assume that requested flight launch times are uniformly distributed across a fixed-length time interval, say from $[0, x]$. Then this problem has been studied by Renyi [1, 5] as a parking problem (i.e., cars of unit length are parked in a $[0, x]$ interval at uniformly distributed locations), and it was shown that the parking density, $M(x)/x$, is Renyi's constant, 0.74759, in the limit as x goes to infinity, where $M(x)$ is the mean of a number of trials with sampling from the uniform distribution. This provides a useful tool for analyzing flight densities through the lanes. For example, in our simulations (over 20 trials) on this problem with $x = 100$ (time units), the flight packing density was found to be 0.743, consistent with Renyi's constant.

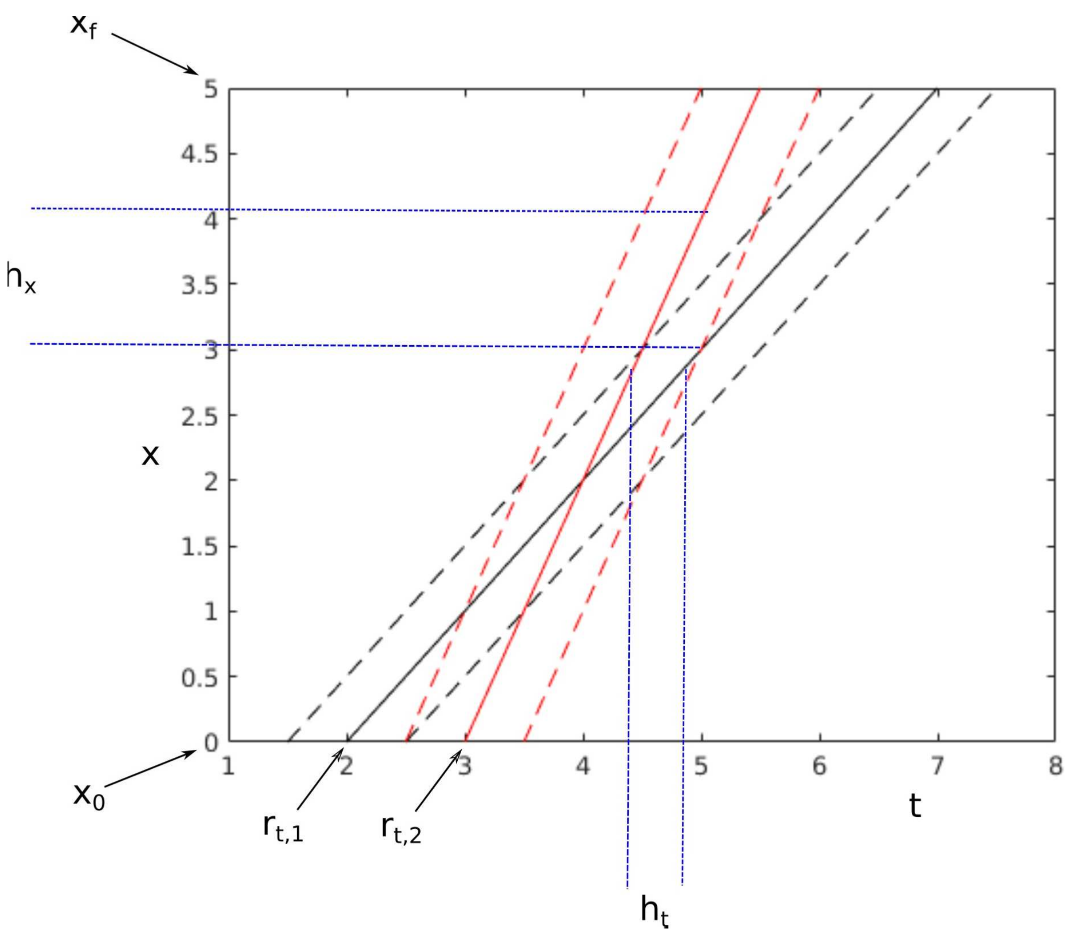

Figure 3: Space-Time Lane Diagram (STLD) for two UASs in a lane. The abscissa is time and the ordinate is distance along the lane. h_t is the time headway (distance between UASs in time in lane), and h_x is the space headway (distance between UASs in lane). Note that h_t and h_x are are linearly related due to the constant speed. The two trajectories in this scenario intersect at $t = 4$ and $x = 2$, however, they violate space-headway before then.

The other main issue is the determination of whether a proposed flight conflicts with any scheduled flight. The Space-Time Lane Diagram (STLD) is used to solve this problem (see Figure 3). Suppose that there exists a set of scheduled flights which are represented in terms of enter-exit times and speed through each lane (the speed of a UAS is assumed constant along a lane, but speeds may differ across UAS's). Let $F(c)$ be the set of scheduled flights through lane c defined as:

$$F(c) \equiv \{t_{1,1}, t_{1,2}, s_1^c; ...; t_{n,1}, t_{n,2}, s_n^c\}$$

where $t_{i,1}$ is the lane entry time for flight i and $t_{i,2}$ is the lane exit time, and s_i^c is the speed of the flight through the lane. Furthermore, let a flight request interval be specified as:

$$R \equiv [q_1, q_2, s^r]$$

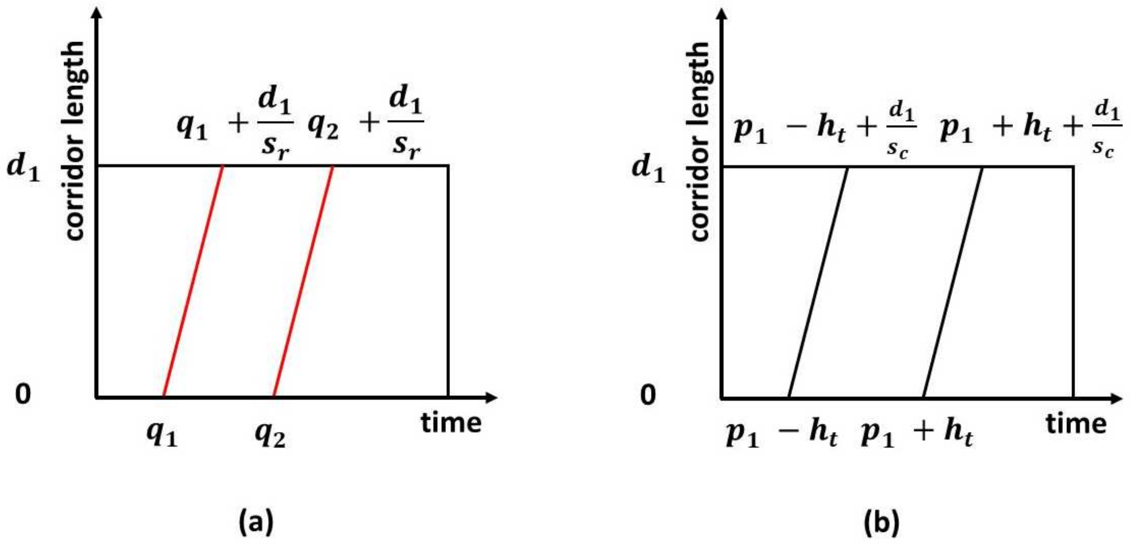

Figure 4: Space-Time Lane Diagrams: (a) trajectory boundaries for requested launch time interval. (b) The headway boundary trajectories for a scheduled flight.

where q_1 is the first possible launch time, q_2 is the latest possible lauch time, and s^r is the proposed speed. What must be determined is the set of (possibly disjoint) intervals in R which are possible launch times (i.e., strategically deconflicted). In order to determine this, the requested launch time interval is put in the Lane 1 STLD as shown in Figure 4(a), where d_1 is the length of Lane 1, q_3 is $q_2 + \frac{d_1}{s^r}$, and q_4 is $q_1 + \frac{d_1}{s^r}$. Each flight in Lane 1 is considered separately to ensure that the time headway, h_t, is respected.

To determine safe launch time intervals, first consider the labeling of the STLD shown in Figure 5. The labels are defined as follows:

- *Label 1*: The interval $[0, q_1)$
- *Label 2*: The point q_1
- *Label 3*: The interval (q_1, q_2)
- *Label 4*: The point q_2
- *Label 5*: The interval (q_2, ∞)
- *Label A*: The interval $[0, q_1 + \frac{d_1}{s^r})$
- *Label B*: The point $q_1 + \frac{d_1}{s^r}$
- *Label C*: The interval $(q_1 + \frac{d_1}{s_r}, q_2 + \frac{d_1}{s^r})$

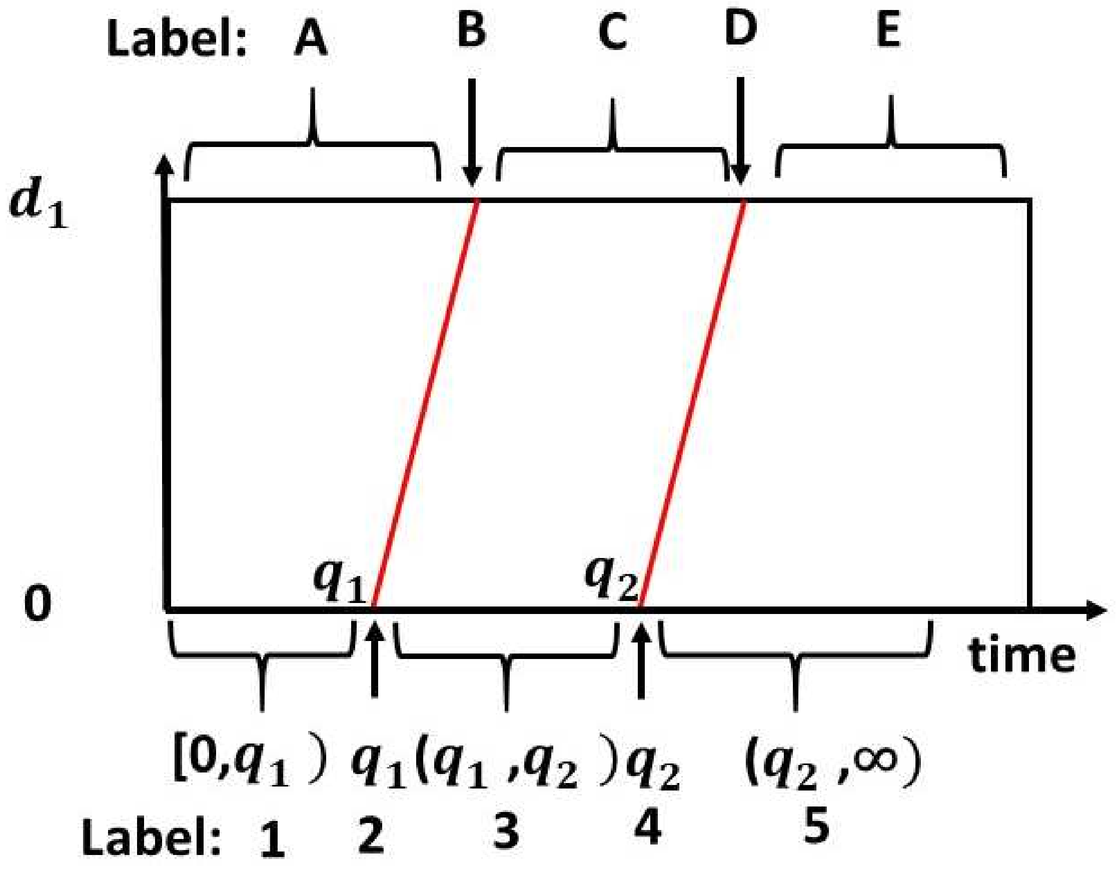

Figure 5: Space-Time Lane Diagram Labels. 1,2,3,4,5 indicate intervals and times at the entry to the lane, and A,B,C,D,E indicate times at the lane exit.

- *Label D*: The point $q_2 + \frac{d_1}{s^r}$
- *Label E*: The interval $(q_2 + \frac{d_1}{s^r}, \infty)$

The two trajectories arising from the scheduled flight are labeled according to where their endpoints lie with respect to the requested launch interval. For example, if $t_{i,2} < q_1$ and $t_{i,2} + d_1/s_i^c < q_1 + d_1/s^r$, then the label for that trajectory is 1A since both start and end points are in the first intervals at distances 0 and d_1, respectively. The relation of a scheduled flight to the requested launch time interval is determined by the labels of the two scheduled flight headway trajectories. Figure 6 shows the first 13 possible combinations. For example, 1A,1A is the case where both headway trajectories are completely to the left (i.e., before in time) the first possible launch time trajectory through the lane. Note that in the figures, p_1 is $t_{i,1} - h_t$, p_2 is $t_{i,1} + h_t$, p_3 is $t_{i,2} + \frac{d_1}{s_i^c}$, and p_4 is $t_{i,1} + \frac{d_1}{s_i^c}$. Also, the square brackets ([]) in the figure indicate the empty interval. Although there are 625 total label combinations, only 139 are physically possible; for example, no start time can be greater than the end time (see Appendix A for an enumeration). For each combination, it is

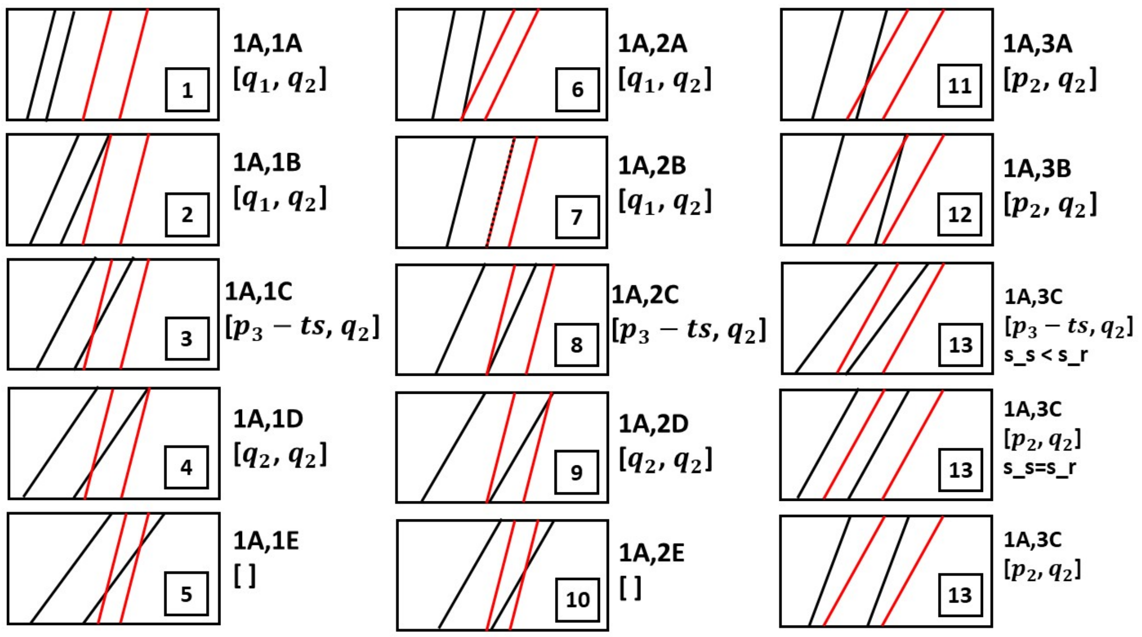

Figure 6: Space-Time Lane Diagrams for the First 13 Possible Label Combinations.

possible to give the safe launch intervals contained in $[t_1^r, t_2^r]$ (see the figure for some examples). In some cases, there is no possible safe launch time (e.g., 1A,1E in the figure). For other combinations, the resulting safe intervals depend on the relative speeds of the two UAS's. An example of this is 1A,3C where a scheduled flight slower than the requested flight has a different interval as when the scheduled flight is equal or greater in speed. It can also happen that multiple intervals result as shown by the 2B,3C case in Figure 7. To determine the viability of a flight through the complete sequence of lanes, each lane is considered in order as described by Algorithm SD (Strategic Deconfliction).

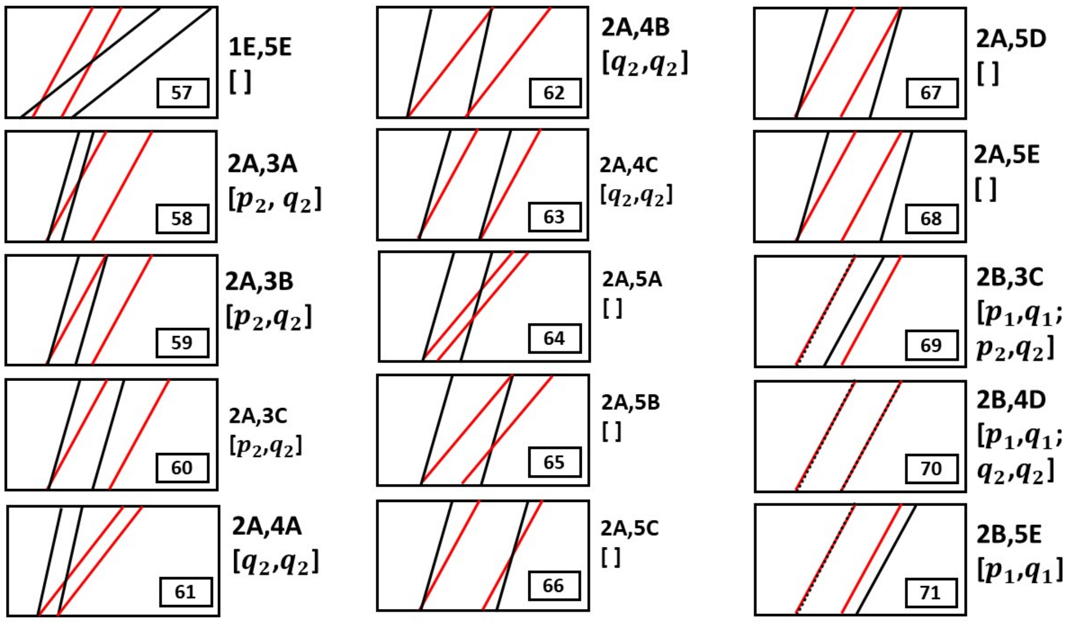

Figure 7: Space-Time Lane Diagrams for Possible Label Combinations 57 through 71.

Algorithm SD (Strategic Deconfliction)

```
On input:
   $n_c$: number of lanes
   flights: flights per lane
   $h_t$: headway time
On output:
   Safe time intervals to launch
begin
possible_intervals ← $[t_1^r, t_2^r]$
for each lane $c$
   possible_intervals ← possible_intervals + time_offset
   for each flight in lane $c$
      new_intervals ← $\emptyset$
      for each interval in possible_intervals
         $[t_1, t_2]$ ← interval $i$
         label ← get_label($t_{i,1}^c, t_{i,2}^c, s_i^c, t_1, t_2, s^r, h_t$)
         f_interval ← get_interval(label,$t_{i,1}^c, t_{i,2}^c, s_i^c, t_1, t_2, s^r, h_t$)
         new_intervals ← merge(new_intervals,f_intervals)
      end
   end
   possible_intervals ← new_intervals
end
```

The key computational cost of this algorithm is the determination of f_interval; however, this is done in constant time for each scheduled flight using the trajectory labels to index to get the appropriate resulting intervals. Appendix B gives the Matlab code for the algorithm.

3 Conclusions

Algorithm SD has been developed to provide strategically deconflicted flight plans for lane-based, large-scale UAS flight management. It has been demonstrated on a number of scenarios without problem. The next objective is to exploit the algorithm to determine optimal Urban Air Mobility paraemters (location of launch/land sites, lane speeds, etc.) which will be compared based on a number of airway lane performance measures (e.g., flow, density, utilization, etc.) determined by large-scale simulations.

References

[1] Matthew P. Clay and Nandor Simanyi. Renyi's Parking Problem Revisited. *Stochastics and Dynamics*, 16:1–12, June 2014.

[2] Dae-Sung Jang, Corey A. Ippolito, Shankar Sankararaman, and Vahram Stepanyan. Concepts of Airspace Structures and System Analysis for UAS Traffic flows for Urban Areas. In *AIAA Information Systems-AIAA Infotech @ Aerospace*, Grapevine, Texas, 1 2017. American Institute of Aeronautics and Astronautics.

[3] Matt R. Jardin. Analytical Relationships Between Conflict Counts and Air-Traffic Density. *Journal of Guidance, Control, and Dynamics*, 28(6):1150–1156, 2005.

[4] Engineering National Academies of Sciences and Medicine. *Roundabouts: An Informational Guide*. Transportation Research Board, Washington, D.C., 2 edition, 12 2016.

[5] A Rényi. On a One-Dimensional Problem Concerning Random Space Filling. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences,*, 3:109–127, 1958.

[6] D. Sacharny and T.C. Henderson. A Lane-Based Approach fo Large-Scale Strategic Conflict Management for UAS Service Suppliers. In *IEEE International Conference on Unmanned Aircraft Systems*, Atlanta, GA, 2019.

4 Appendix A: Space-Time Lane Diagram Enumeration

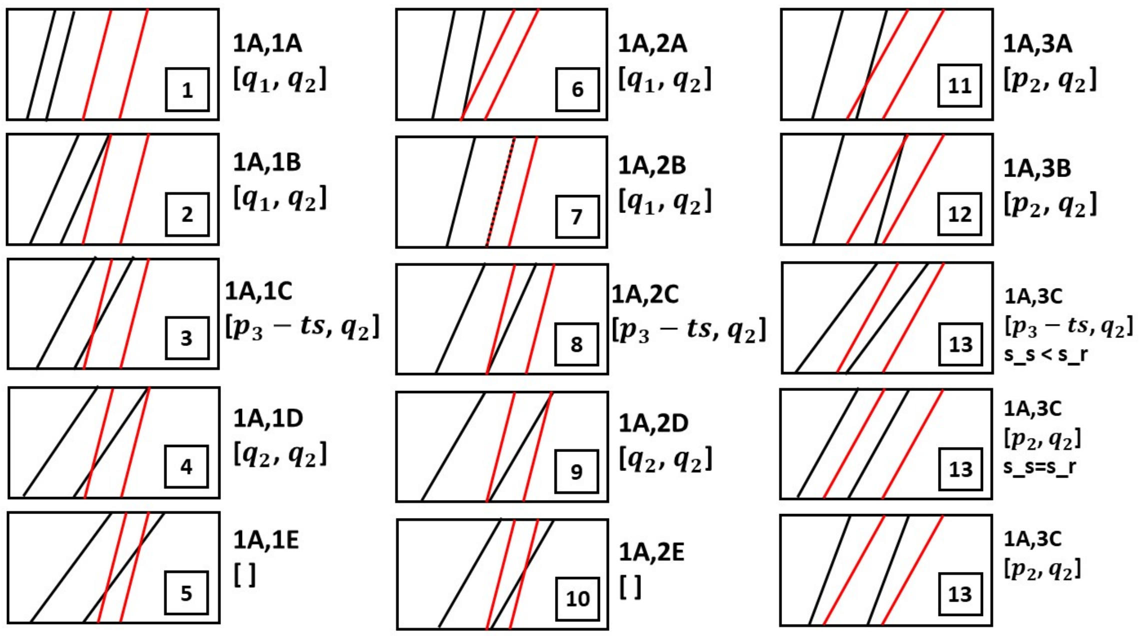

Figure 8: Space-Time Lane Diagrams for Possible Label Combinations 1 through 13.

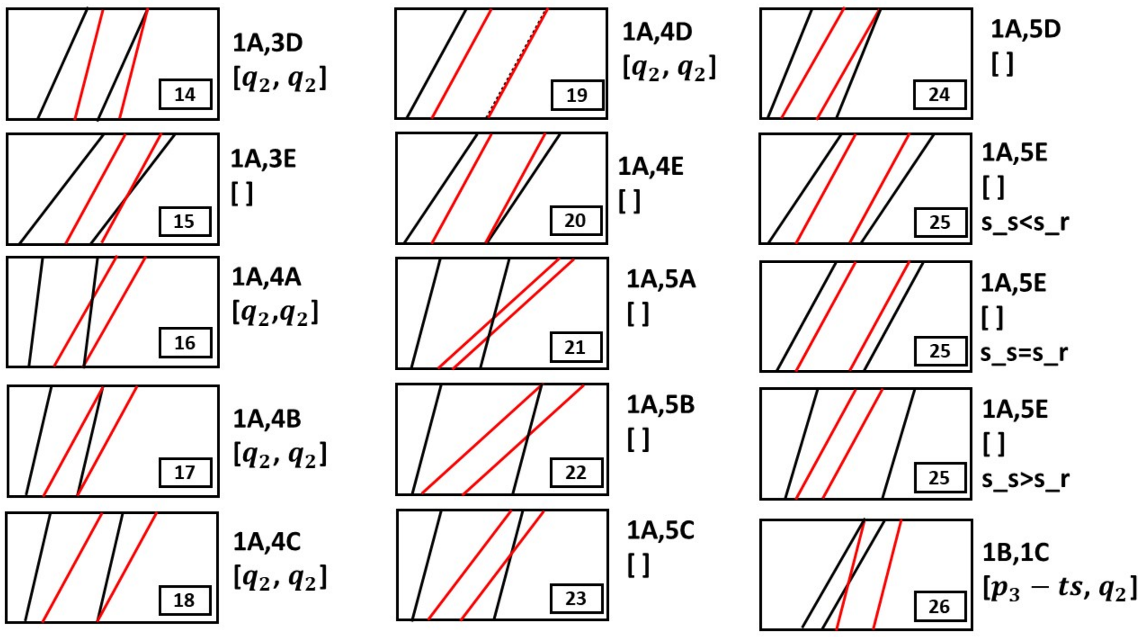

Figure 9: Space-Time Lane Diagrams for Possible Label Combinations 14 through 26.

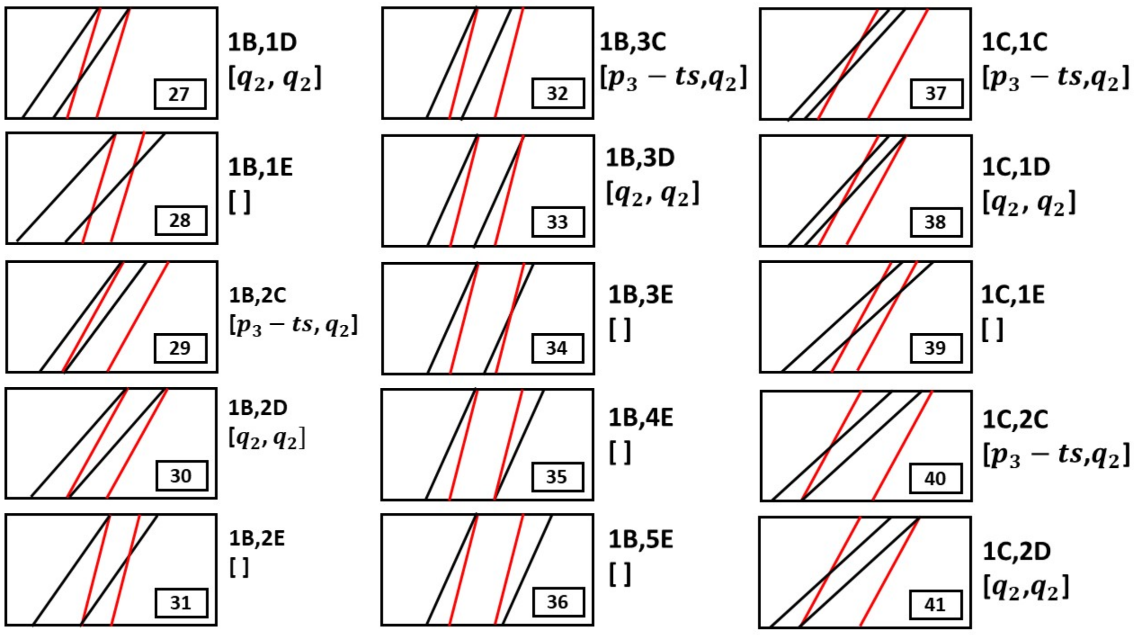

Figure 10: Space-Time Lane Diagrams for Possible Label Combinations 27 through 41.

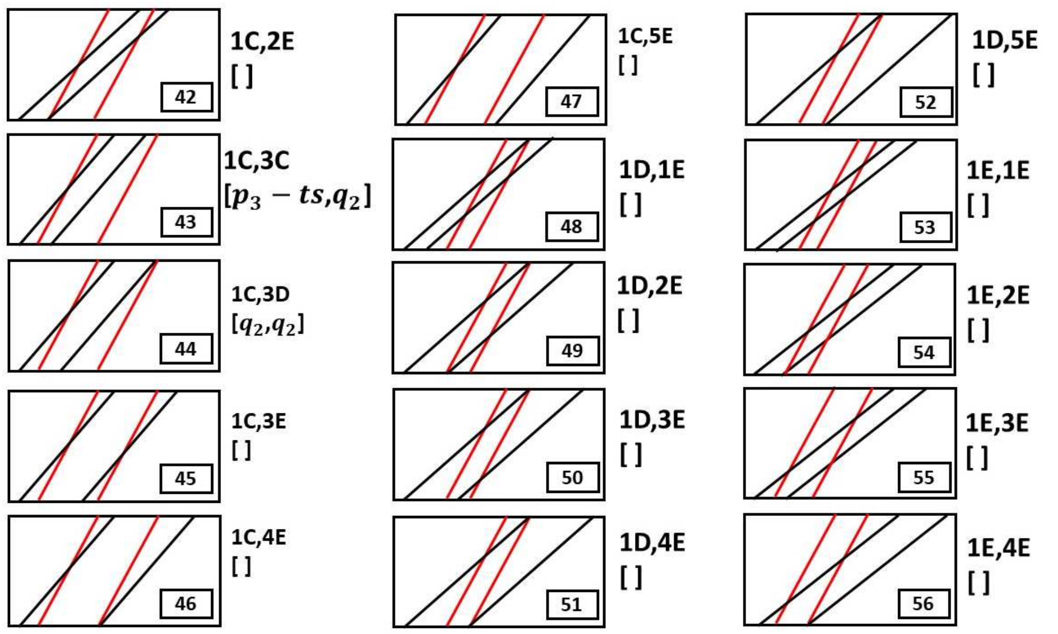

Figure 11: Space-Time Lane Diagrams for Possible Label Combinations 42 through 56.

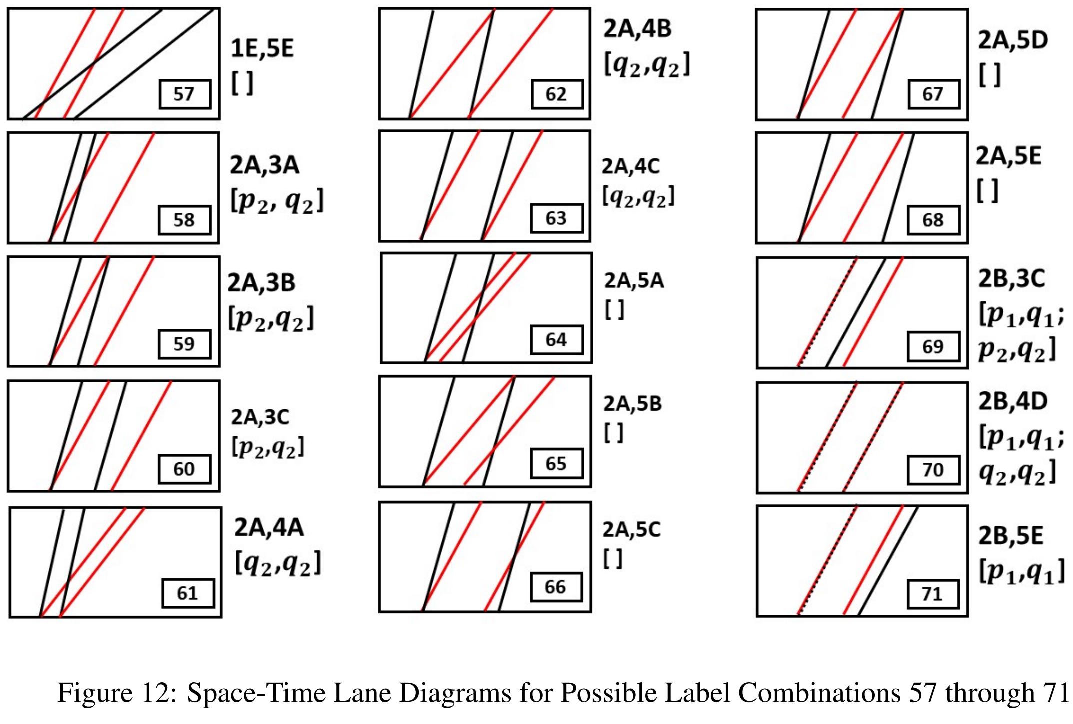

Figure 12: Space-Time Lane Diagrams for Possible Label Combinations 57 through 71.

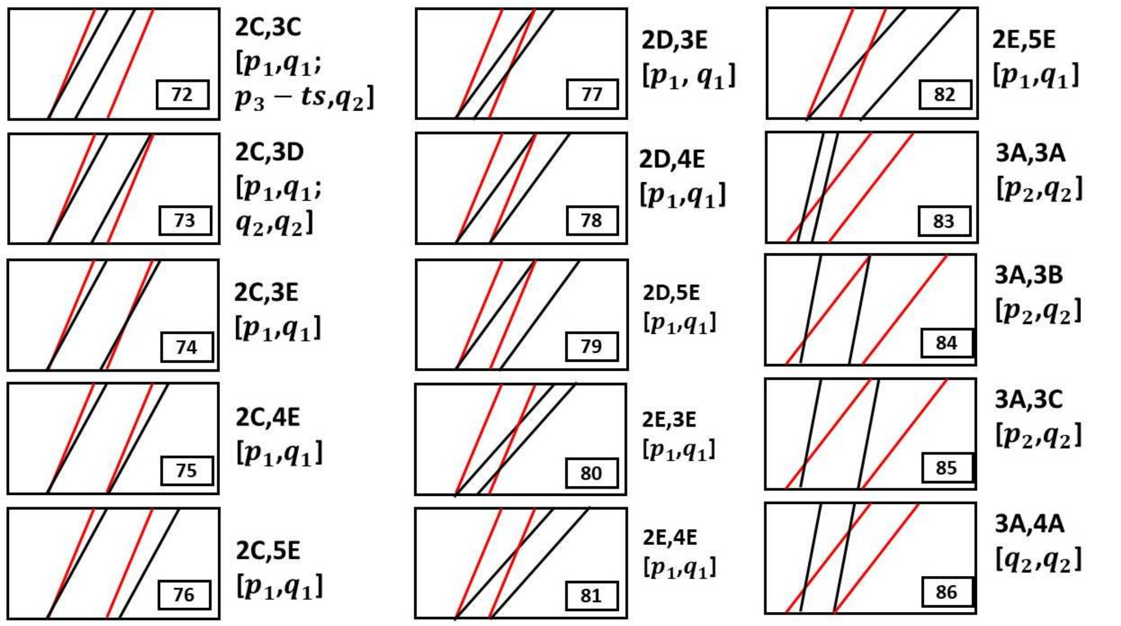

Figure 13: Space-Time Lane Diagrams for Possible Label Combinations 72 through 86.

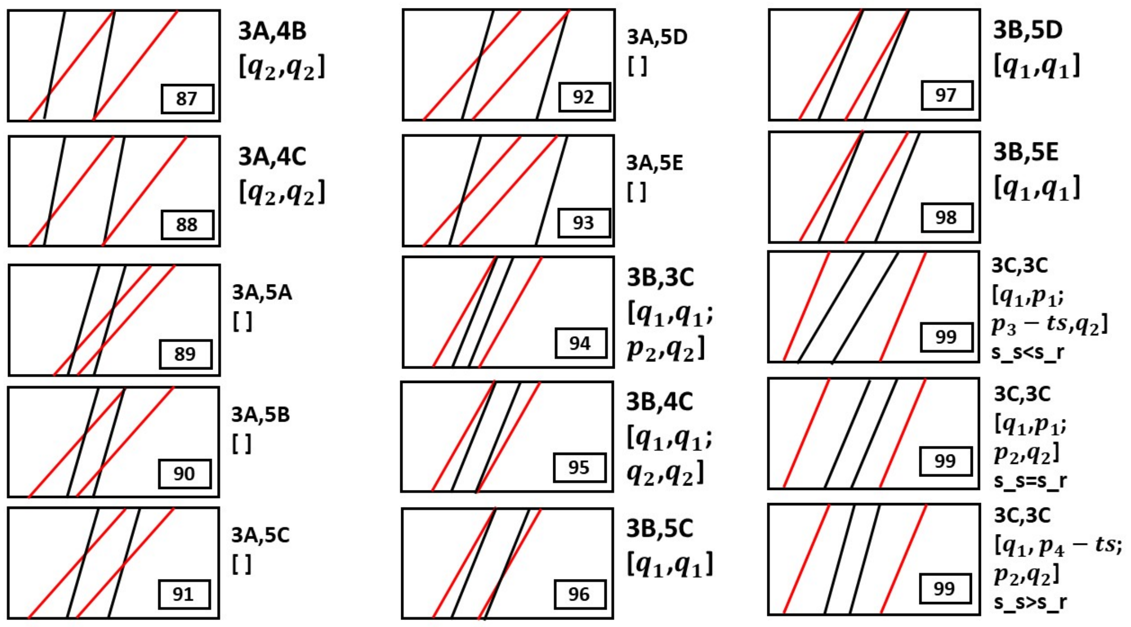
87 3A,4B $[q_2,q_2]$
88 3A,4C $[q_2,q_2]$
89 3A,5A [ ]
90 3A,5B [ ]
91 3A,5C [ ]
92 3A,5D [ ]
93 3A,5E [ ]
94 3B,3C $[q_1,q_1;$ $p_2,q_2]$
95 3B,4C $[q_1,q_1;$ $q_2,q_2]$
96 3B,5C $[q_1,q_1]$
97 3B,5D $[q_1,q_1]$
98 3B,5E $[q_1,q_1]$
99 3C,3C $[q_1,p_1;$ $p_3-ts,q_2]$ s_s<s_r
99 3C,3C $[q_1,p_1;$ $p_2,q_2]$ s_s=s_r
99 3C,3C $[q_1,p_4-ts;$ $p_2,q_2]$ s_s>s_r

Figure 14: Space-Time Lane Diagrams for Possible Label Combinations 87 through 99.

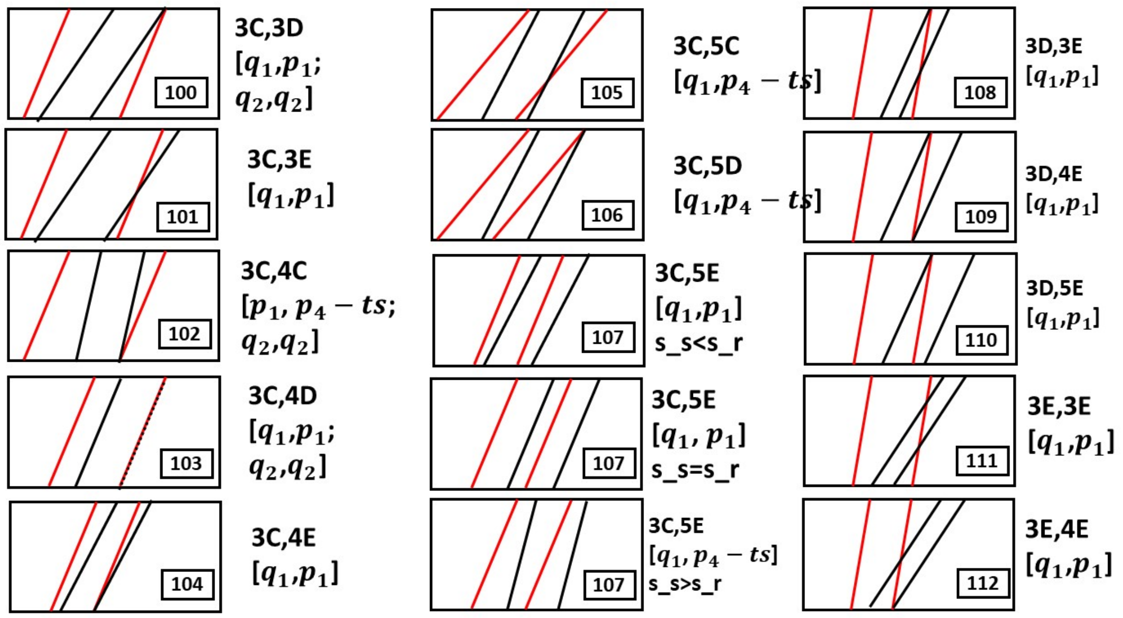

Figure 15: Space-Time Lane Diagrams for Possible Label Combinations 100 through 112.

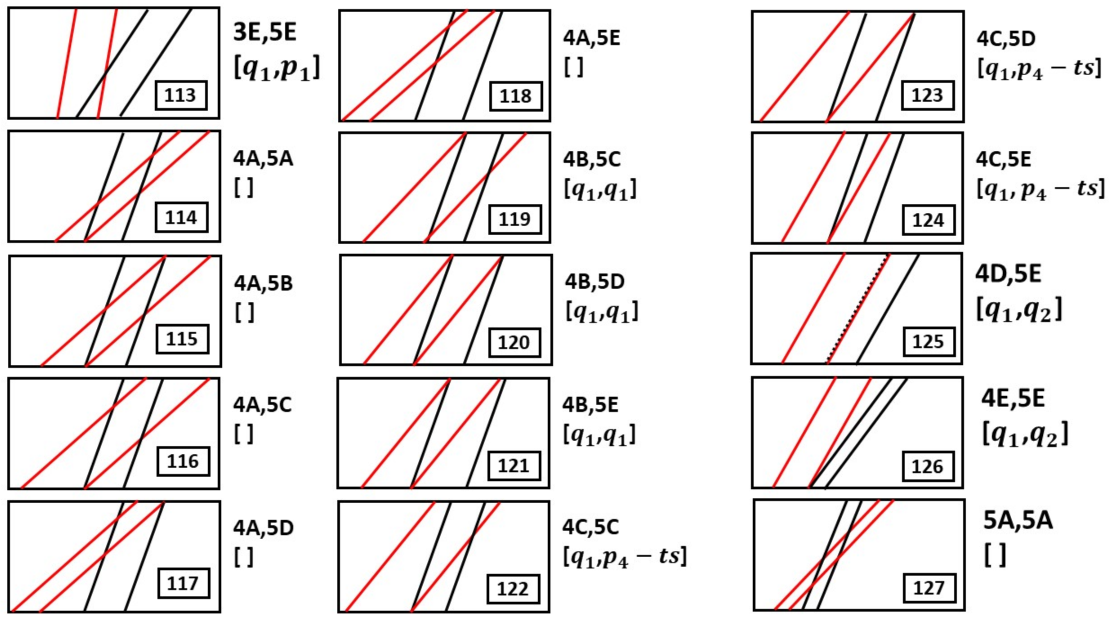

Figure 16: Space-Time Lane Diagrams for Possible Label Combinations 113 through 127.

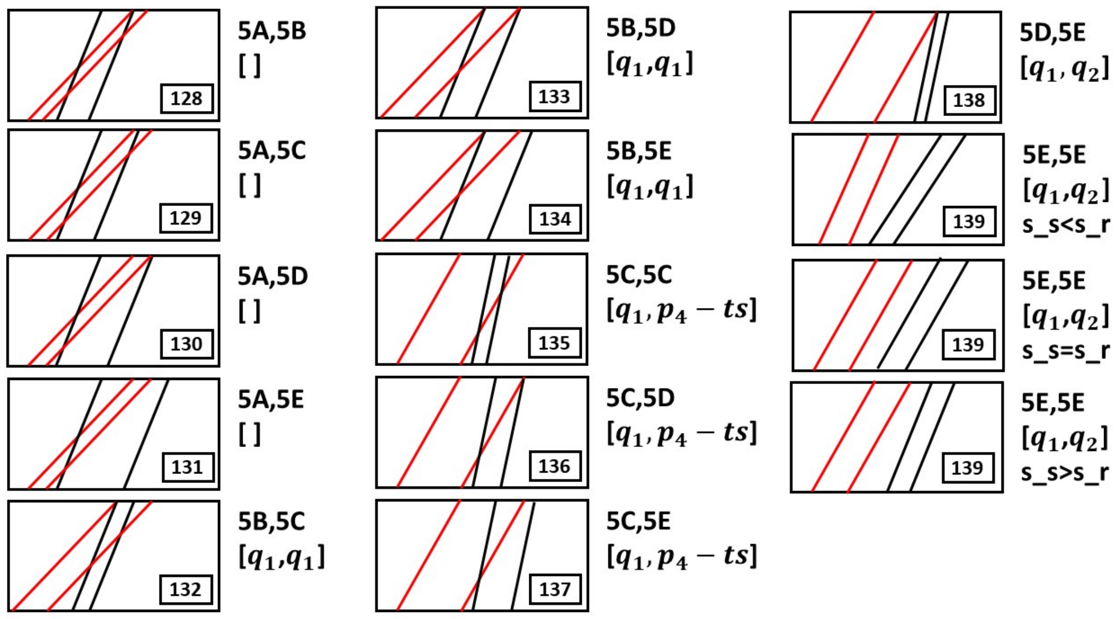

Figure 17: Space-Time Lane Diagrams for Possible Label Combinations 128 through 139.

5 Appendix B: Matlab Code for Algorithm SD

```
function possible = UR_possible_times_int(possible0,speed,cor_list,...
    cor_lengths,flights,ht)
% UR_possible_times_int - provide possible strategically deconflicted
%      launch time intervals given a requested interval and the
%      scheduled flights
% On input:
%     possible0 (1x2 vector): first and last possible launch times
%     speed (float): speed to requesting UAS
%     cor_list (kx1 vector): list of corridors to be traversed (in
%     order)
%     cor_lengths (kx1 vector): lengths of corridors to be traversed
%     flights (vector struct): scheduled flights (given per corridor)
%     ht (float): headway time
% On output:
%     possible (nx2 array): each row is a continuous interval of
%     possible
%        starting flight times
% Call:
%     inters =
%     UR_possible_times_int([4,35],2,[13,6,14],[500,6,500],fl,5);
% Author:
%     T. Henderson
%     UU
%     Summer 2019
%

len_cor_list = length(cor_list);
intervals = possible0;
offset = 0;
c = 0;
total_time = 0;
while ~isempty(intervals)&c<len_cor_list
    c = c + 1;
    dc = cor_lengths(c);
    cor = cor_list(c);
    ts = dc/speed;
    [num_intervals,dummy] = size(intervals);
    for k = 1:num_intervals
        intervals(k,:) = intervals(k,:) + [offset,offset];
    end
    c_flights = flights(cor).flights;
    if ~isempty(c_flights)
        [num_c_flights,dummy] = size(c_flights);
        f = 0;
        [num_intervals,dummy] = size(intervals);
        while f<num_c_flights&~isempty(intervals)
            f = f + 1;
            tr1 = min(intervals(:,1));
            tr2 = max(intervals(:,2));
            ts1 = c_flights(f,1);
            ts2 = c_flights(f,2);
            tr1e = tr1 + dc/speed;
            tr2e = tr2 + dc/speed;
            if ~((ts1+ht<=tr1&ts2+ht<=tr1e)|(ts1-ht>=tr2&ts2-ht>=tr2e))
                new_intervals = [];
                for k = 1:num_intervals
                    k_intervals = UR_OK_sched_req_enum(c_flights(f,1),...
                        c_flights(f,2),c_flights(f,3),intervals(k,1),...
                        intervals(k,2),speed,dc,ht);
                    new_intervals = UR_merge_intervals(k_intervals,...
                        new_intervals);
                end
                intervals = new_intervals;
                if isempty(intervals)
                    num_intervals = 0;
                else
                    num_intervals = length(intervals(:,1));
                end
            end
        end
    end
    offset = ts;
    total_time = total_time + ts;
end
total_time = total_time - ts;
[num_intervals,dummy] = size(intervals);
for k = 1:num_intervals
    intervals(k,:) = intervals(k,:) - [total_time,total_time];
end
if ~isempty(intervals)
    t1 = intervals(1,1);
    offset = 0;
    for c = 1:len_cor_list
        cor = cor_list(c);
        if ~isempty(flights(cor).flights)...
                &abs(t1-flights(cor).flights(1,1))<7
            tch = 0;
        end
        t1 = t1 + cor_lengths(c)/speed;
    end
end

%  return all intervals
possible = intervals;
return

function intervals =
UR_OK_sched_req_enum(ts1,ts2,s_s,tr1,tr2,s_r,d,ht)
% UR_OK_sched_req_enum - determine OK intervals for proposed flight in
%                        specific corridor
% On input:
%     ts1 (float): start of scheduled flight
%     ts2 (float): end of scheduled flight
%     s_s (float): speed of scheduled flight
%     tr1 (float): min start time requested
%     tr2 (float): max start time requested
%     s_r (float): speed of requested flight
%     d (float): corridor length
%     ht (float): headway time
% On output:
%     intervals (nx2 array): possible start time intervals
% Call:
%     int1 = UR_OK_sched_req_enum(23,51,5,8,40,3,49,5);
% Author:
%     T. Henderson
%     UU
%     Summer 2019
%

persistent first itable

if isempty(first)
    first = 0;
    itable = [...
        1 1 1 1;...  % Case   1
        1 1 1 2;...  % Case   2
        1 1 1 3;...  % Case   3
        1 1 1 4;...  % Case   4
        1 1 1 5;...  % Case   5
        1 1 2 1;...  % Case   6
        1 1 2 2;...  % Case   7
        1 1 2 3;...  % Case   8
        1 1 2 4;...  % Case   9
        1 1 2 5;...  % Case  10
        1 1 3 1;...  % Case  11
        1 1 3 2;...  % Case  12
        1 1 3 3;...  % Case  13
        1 1 3 4;...  % Case  14
        1 1 3 5;...  % Case  15
        1 1 4 1;...  % Case  16
        1 1 4 2;...  % Case  17
        1 1 4 3;...  % Case  18
        1 1 4 4;...  % Case  19
        1 1 4 5;...  % Case  20
        1 1 5 1;...  % Case  21
        1 1 5 2;...  % Case  22
        1 1 5 3;...  % Case  23
        1 1 5 4;...  % Case  24
        1 1 5 5;...  % Case  25
        1 2 1 3;...  % Case  26
        1 2 1 4;...  % Case  27
```

```
1 2 1 5;...  % Case  28
1 2 2 3;...  % Case  29
1 2 2 4;...  % Case  30
1 2 2 5;...  % Case  31
1 2 3 3;...  % Case  32
1 2 3 4;...  % Case  33
1 2 3 5;...  % Case  34
1 2 4 5;...  % Case  35
1 2 5 5;...  % Case  36
1 3 1 3;...  % Case  37
1 3 1 4;...  % Case  38
1 3 1 5;...  % Case  39
1 3 2 3;...  % Case  40
1 3 2 4;...  % Case  41
1 3 2 5;...  % Case  42
1 3 3 3;...  % Case  43
1 3 3 4;...  % Case  44
1 3 3 5;...  % Case  45
1 3 4 5;...  % Case  46
1 3 5 5;...  % Case  47
1 4 1 5;...  % Case  48
1 4 2 5;...  % Case  49
1 4 3 5;...  % Case  50
1 4 4 5;...  % Case  51
1 4 5 5;...  % Case  52
1 5 1 5;...  % Case  53
1 5 2 5;...  % Case  54
1 5 3 5;...  % Case  55
1 5 4 5;...  % Case  56
1 5 5 5;...  % Case  57
2 1 3 1;...  % Case  58
2 1 3 2;...  % Case  59
2 1 3 3;...  % Case  60
2 1 4 1;...  % Case  61
2 1 4 2;...  % Case  62
2 1 4 3;...  % Case  63
2 1 5 1;...  % Case  64
2 1 5 2;...  % Case  65
2 1 5 3;...  % Case  66
2 1 5 4;...  % Case  67
2 1 5 5;...  % Case  68
2 2 3 3;...  % Case  69
2 2 4 4;...  % Case  70
2 2 5 5;...  % Case  71
```

```
2 3 3 3;...  % Case  72
2 3 3 4;...  % Case  73
2 3 3 5;...  % Case  74
2 3 4 5;...  % Case  75
2 3 5 5;...  % Case  76
2 4 3 5;...  % Case  77
2 4 4 5;...  % Case  78
2 4 5 5;...  % Case  79
2 5 3 5;...  % Case  80
2 5 4 5;...  % Case  81
2 5 5 5;...  % Case  82
3 1 3 1;...  % Case  83
3 1 3 2;...  % Case  84
3 1 3 3;...  % Case  85
3 1 4 1;...  % Case  86
3 1 4 2;...  % Case  87
3 1 4 3;...  % Case  88
3 1 5 1;...  % Case  89
3 1 5 2;...  % Case  90
3 1 5 3;...  % Case  91
3 1 5 4;...  % Case  92
3 1 5 5;...  % Case  93
3 2 3 3;...  % Case  94
3 2 4 3;...  % Case  95
3 2 5 3;...  % Case  96
3 2 5 4;...  % Case  97
3 2 5 5;...  % Case  98
3 3 3 3;...  % Case  99
3 3 3 4;...  % Case 100
3 3 3 5;...  % Case 101
3 3 4 3;...  % Case 102
3 3 4 4;...  % Case 103
3 3 4 5;...  % Case 104
3 3 5 3;...  % Case 105
3 3 5 4;...  % Case 106
3 3 5 5;...  % Case 107
3 4 3 5;...  % Case 108
3 4 4 5;...  % Case 109
3 4 5 5;...  % Case 110
3 5 3 5;...  % Case 111
3 5 4 5;...  % Case 112
3 5 5 5;...  % Case 113
4 1 5 1;...  % Case 114
4 1 5 2;...  % Case 115
```

```
        4 1 5 3;...  % Case 116
        4 1 5 4;...  % Case 117
        4 1 5 5;...  % Case 118
        4 2 5 3;...  % Case 119
        4 2 5 4;...  % Case 120
        4 2 5 5;...  % Case 121
        4 3 5 3;...  % Case 122
        4 3 5 4;...  % Case 123
        4 3 5 5;...  % Case 124
        4 4 5 5;...  % Case 125
        4 5 5 5;...  % Case 126
        5 1 5 1;...  % Case 127
        5 1 5 2;...  % Case 128
        5 1 5 3;...  % Case 129
        5 1 5 4;...  % Case 130
        5 1 5 5;...  % Case 131
        5 2 5 3;...  % Case 132
        5 2 5 4;...  % Case 133
        5 2 5 5;...  % Case 134
        5 3 5 3;...  % Case 135
        5 3 5 4;...  % Case 136
        5 3 5 5;...  % Case 137
        5 4 5 5;...  % Case 138
        5 5 5 5];    % Case 139
end

intervals = [];

t_across = d/s_r;
%t_across = ceil(d/s_r);
p1 = ts1 - ht;
p2 = ts1 + ht;
p3 = ts2 + ht;
p4 = ts2 - ht;
q1 = tr1;
q2 = tr2;
q3 = tr2 + t_across;
q4 = tr1 + t_across;

if p1<q1
    i1 = 1;
elseif p1==q1
    i1 = 2;
elseif p1>q1&p1<q2
```

```
    i1 = 3;
elseif p1==q2
    i1 = 4;
else
    i1 = 5;
end
if p2<q1
    i3 = 1;
elseif p2==q1
    i3 = 2;
elseif p2>q1&p2<q2
    i3 = 3;
elseif p2==q2
    i3 = 4;
else
    i3 = 5;
end

if p3<q4
    i4 = 1;
elseif p3==q4
    i4 = 2;
elseif p3>q4&p3<q3
    i4 = 3;
elseif p3==q3
    i4 = 4;
else
    i4 = 5;
end
if p4<q4
    i2 = 1;
elseif p4==q4
    i2 = 2;
elseif p4>q4&p4<q3
    i2 = 3;
elseif p4==q3
    i2 = 4;
else
    i2 = 5;
end

index = find(itable(:,1)==i1&itable(:,2)==i2&itable(:,3)==i3...
    &itable(:,4)==i4);
switch index
```

```
case {1,2,6,7,125,126,138,139}
    intervals = [q1,q2];
case {3,8,26,29,32,37,40,43}
    intervals = [p3-t_across,q2];
case {4,9,14,16,17,18,19,27,30,33,38,41,44,61,62,63,86,87,88}
    intervals = [q2,q2];
case {11,12,58,59,60,83,84,85}
    intervals = [p2,q2];
case 13
    if s_s<s_r
        intervals = [p3-t_across,q2];
    else
        intervals = [p2,q2];
    end
case 69
    intervals = [p1,q1; p2,q2];
case {70,73}
    intervals = [p1,q1; q2,q2];
case {71,74,75,76,77,78,79,80,81,82}
    intervals = [p1,q1];
case 72
    intervals = [p1,q1; p3-t_across,q2];
case {94,95}
    intervals = [q1,q1;q2,q2];
case 99
    if s_s<s_r
        intervals = [q1,p1; p3-t_across,q2];
    elseif s_s==s_r
        intervals = [q1,p1; p2,q2];
    else
        intervals = [q1,p4-t_across; p2,q2];
    end
case {96,97,98,119,120,121,132,133,134}
    intervals = [q1,q1];
case {100,103}
    intervals = [q1,p1; q2,q2];
case 102
    intervals = [p1,p4-t_across; q2,q2];
case {101,104,108,109,110,111,112,113}
    intervals = [q1,p1];
case 107
    if s_s<=s_r
        intervals = [q1,p1];
    else
```

```
            intervals = [q1,p4-t_across];
        end
    case {105,106,122,123,124,135,136,137}
        intervals = [q1,p4-t_across];
end

function new_int = UR_merge_intervals(int1,int2)
% UR_merge_intervals - given two sets of intervals, merge them
% On input:
%     int1 (n1x2 array): first set of intervals
%     int2 (n2x2 array): second set of intervals
% On output:
%     new_int (px2 array): intersection of two interval sets
% Call:
%     new_int = UR_merge_intervals (int1,int2);
% Author:
%     T. Henderson
%     UU
%     Summer 2019
%

if isempty(int1)&isempty(int2)
    new_int = [];
    return
end

new_int = [int1;int2];
[vals,indexes] = sort(new_int(:,1));
new_int = new_int(indexes,:);
change = 1;
while change==1
    change = 0;
    len_new_int = length(new_int(:,1));
    for k = 1:len_new_int-1
        if new_int(k,1)==new_int(k+1,1)
            v_min = min(new_int(k,2),new_int(k+1,2));
            new_int(k+1,2) = v_min;
            new_int(k,:) = [];
            change = 1;
            break
        end
    end
end
```