Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal

Volume 9 | Issue 1 Article 3

March 2022

Scheduling Aircraft Departures to Avoid Enroute Congestion

Johannes Martinez
University of Minnesota - Morris

Scheduling Aircraft Departures to Avoid Enroute Congestion

Johannes Martinez
mart4695@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

When scheduled flights are forecast to overcrowd sections of enroute airspace, an air traffic control authority may need to delay departures. Mixed integer linear programming can be used to compute a schedule that resolves the congestion while bringing the sum of all delays to a minimum. Standard linear programming constraint formulations for such scheduling problems, however, have poor run times for instances of realistic size. A new constraint formulation based on cycles and paths through a route graph reduces run times in computational experiments. It shows particularly strong performance for schedules that approach the worst-case solution times in standard formulations.

Keywords: air traffic management, hotspot problem, mixed integer linear programming, scheduling, enroute airspace.

1 Introduction

Airlines submit flight plans to an air traffic control authority (ATC) to access high-altitude enroute airspace and to fly when the visibility is poor. Flights request routes and departure times, allowing ATC to forecast traffic conditions. Once airborne, every aircraft imposes a workload on controllers assigned to the airspace's sectors. Although sectors are staffed according to typical traffic conditions, traffic spikes and increasing airspace use can create capacity overloads known as *hotspots*. Hotspots may increase errors, worsen congestion and delays, and lead to accidents.

In this paper, sector capacity is defined as a limit on the number of aircraft allowed in a sector at once. In practice, however, capacity may be defined according to the number of tasks controllers perform over a period, a measure associated with aircraft entries per sector per controller shift. The scheduling techniques discussed can be extended to a variety of hotspot definitions [4].

A central ATC authority may receive flight plans early enough to simulate various traffic scenarios [4]. If these simulations indicate that the flight plans create hotspots, the authority may request aircraft to submit altered plans or else accept route changes. But if such measures do not succeed in resolving the hotspot, the authority must begin delaying flights. Especially for widespread predicted congestion, the question then is which aircraft to delay and for how long. Problems of this type can be solved using *linear programming*, introduced in Section 2. The particular manner in which the problem is written into a linear program is known as a *formulation*. In Section 3 we describe a standard formulation of the hotspot problem, while in Section 4 we describe an improved formulation introduced by Mannino and Sartor [5]. Finally, in Section 5 we discuss computational experiments comparing the two formulations.

2 Background

2.1 Linear Programming

In a linear program, we try to minimize or maximize a linear function subject to a set of linear constraints. Our objective, for example, may be to minimize the sum of two variables, x_1 and x_2. If our variables represent coordinates on a Cartesian plane, we can imagine that our constraints outline a *feasible region*, an area of the plane that corresponds to permitted solutions to our problem (not every feasible solution is necessarily optimal, though). Our problem, for instance, may have three constraints, each of which requires that a linear expression composed of our variables be of at least or at most a certain value. We can write a linear program as follows:

$$\begin{array}{lll} \text{minimize} & x_1 + x_2 & (i) \\ \text{subject to} & -x_1 + 2x_2 \leq 2 & (ii) \\ & x_1 + 2x_2 \geq 3 & (iii) \\ & 2x_1 - x_2 \leq 4. & (iv) \end{array}$$

Our objective function (i) is written first, followed by our constraints (ii) to (iv). If we plot the constraints, we can see that our problem's feasible region is the triangle seen in Figure 1. Any point on the edges or interior of this triangle is a feasible solution. While there are an infinite number of feasible coordinate pair solutions in this case, we are interested in the pair whose sum is the lowest.

All points (x_1, x_2) with the same sum, z, form a line with slope -1 [2]. The line is described by the linear objective function itself, and is therefore known as the *objective line*. In Figure 1, if we translate the objective line from the origin towards the feasible region, the sum $x_1 + x_2$ will grow and the line will eventually reach either an edge or a vertex of the feasible region. In this case, point (0.5, 1.25) represents the optimal solution, as any objective line beyond that will have a greater sum z. It is sometimes useful to know the

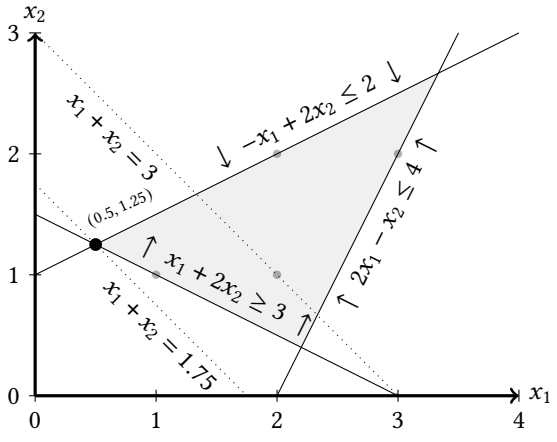

Figure 1. Graph showing constraint inequalities. Each decision variable x_1 and x_2 adds a dimension. In the shaded feasible region, any point $(x_1,\ x_2)$ satisfies all three constraints. The dotted lines show the points at which the objective value is 1.75 and 3. The minimum feasible solution occurs on the labeled vertex. Light dots show the integer values in the feasible region.

optimal integer solution, shown in the figure as the point (1, 1).

2.2 Linear Programming Solvers

The task of a linear programming solver is to find an optimal solution to an objective function subject to constraints or else to determine that no solution exists. If we consider a single constraint in the previous example, we can see that it divides the plane into half-planes and that the intersection of all constraint half-planes produces the feasible region. The objective line, meanwhile, relates all points in the feasible region that share the same value of the objective function. Because an enclosed feasible region contains an infinite number of points, it is not possible for a computer to evaluate the objective function at every single point in a finite amount of time and to determine from that what the optimal solution is. Because of the mathematical properties of feasible regions produced by linear constraints, however, the objective line will always find an optimal solution at a vertex [1]. Even if the region were bordered by a segment parallel with the objective line, we can still make the case that optimal solutions always exist at vertices because the segment is bounded by vertices that have solutions as optimal as any of the infinite solutions along the line [2].

When we add another variable, we add another dimension to our space. With three variables, each constraint creates half-spaces, and the intersection of these half-spaces creates a feasible region. All points in this region that result in the same z value are linked by an objective plane. This plane can encounter a single vertex at a three-dimensional feasible region's corner. As with the parallel line segment case in two dimensions, if a plane parallel to the objective plane is encountered, that plane will still be bounded by vertices that contain a solution as optimal as any of the infinite number of points along the plane. In general, a feasible region in any number of dimensions is known as a *simplex*.

The *simplex algorithm* forms the basis of many solvers. The algorithm starts at a vertex, and, in the case of minimizing, moves to a neighboring vertex whose objective output is equal or lower than the output of the current vertex. Because the simplex is convex, it has no local dip that is not also global. Thus, if the simplex algorithm finds no neighboring vertex with a lower objective output, the current vertex can be considered the optimal solution, and other vertices can be safely ignored. The simplex algorithm usually avoids visiting many if not most nodes, and its behavior can thus be described as polynomial on average. In the worst case, however, feasible regions can be shaped in a manner that forces the algorithm to visit 2^n vertices, where n is the number of variables in the problem.

2.3 Mixed Integer Linear Programming

We will see in Section 3.3 that modeling the hotspot problem requires discrete variables to represent binary conditions. A flight, for example, either is or is not in a sector. To accommodate this, the simplex method must be extended. A *mixed integer linear program* (MILP) accepts continuous and discrete variables. When variables are limited to integers, a convex polygon feasible region no longer has infinite solutions. It also, however, no longer has the convex properties of the feasible region of a continuous linear program: a local maximum or local minimum no longer necessarily implies a global maximum or minimum. Many integer solvers compute a continuous solution first. Then, they search around that continuous solution for the most optimal solution that also satisfies the integer constraint [1]. The search process makes integer solvers visit far more vertices on average than linear solvers. Any integer constraint formulation that does not tightly confine the feasible region may thus be prone to an exponential growth in solution times with the addition of each program variable.

3 A Standard MILP Model

Mannino and Sartor [5] describe a standard approach to modeling the hotspot problem for a mixed integer linear programming solver. As we shall see in Section 3.4, this formulation grows quickly and may be impractical for realistic scenarios. Nevertheless, it serves both as an introduction to the modeling techniques used in an improved formulation the authors introduce (Section 4) and as a point of comparison.

3.1 Relating Flights and Sectors

The aircraft scheduling problem consists of three main phases. First, we collect a set of proposed flights. Then, we estimate

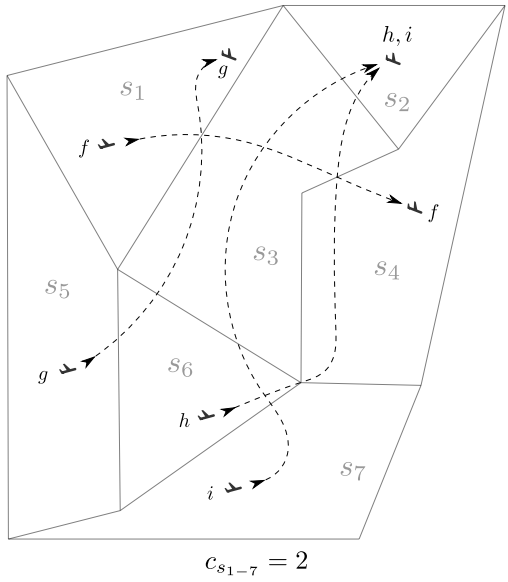

Figure 2. Sector map including planned routes.

sector traversal times for each flight given the route, aircraft type, departure time, weather, and other relevant data. Finally, we build constraints from this data to form a MILP. Throughout this section, we will use an example batch of four flights: f, g, h, and i. We can superimpose the routes these flights submit to ATC on a sector map (Figure 2). Then, we can begin to extract the information required by the scheduling model.

Consider our set of sectors $\{s_1,\ s_2,\ \ldots,\ s_7\} \in S$ and our set of flights $\{f,\ g,\ h,\ i\} \in F$. If a flight passes through a sector, we create a route node $(f, s) \in R$. A flight can thus be represented as a sequence of route nodes. For flight f in Figure 2, for example, we have the sequence

$$(f,\ s_1),\ (f,\ s_3),\ (f,\ s_4), \text{ where each node } (f,\ s) \in R.$$

We store the time at which flight f enters sector s as a schedule node $t_{(f,\,s)}$. All schedule nodes belong to the set $\mathbb{R}$. For the sake of illustration, we will use the infeasible schedule in Figure 3 to demonstrate how the various constraints work and also to store departure times and traversal times.

3.2 Flight-Delaying Constraints

We assume that a flight cannot leave before its scheduled departure time. For any flight f, where the planned departure time is Γ_f and where D is the set of departure route nodes, we must satisfy the inequality

$$t_{(f,\,s)} \geq \Gamma_f, \text{ where } (f, s) \in D.^{1} \tag{1}$$

We can find the minimum departure time for a flight by reading the planned entry time into its departure sector. In Figure 3, we find that $\Gamma_{f,\,g} = 20$, $\Gamma_h = 0$, and $\Gamma_i = 5$.

[1]We assume departure times are measured relative to the start time of the model. A time offset constant can be added if this is not the case.

$$\begin{aligned}
&t_{(f,\,s_1)} = 20, \quad t_{(f,\,s_3)} = 30, \quad t_{(f,\,s_4)} = 45 \\
&t_{(g,\,s_5)} = 20, \quad t_{(g,\,s_6)} = 25, \quad t_{(g,\,s_3)} = 35, \quad t_{(g,\,s_1)} = 55 \\
&t_{(h,\,s_6)} = 0, \quad t_{(h,\,s_4)} = 15, \quad t_{(h,\,s_3)} = 50, \quad t_{(h,\,s_)} = 60 \\
&t_{(i,\,s_7)} = 5, \quad t_{(i,\,s_6)} = 10, \quad t_{(i,\,s_3)} = 20, \quad t_{(i,\,s_2)} = 40
\end{aligned}$$

Figure 3. An unfeasible schedule used to illustrate how the model constraints work. This schedule also stores estimated sector traversal times and filed departure times.

Considering any flight f, we define the time to traverse a sector s as $\Lambda_{(f,\,s)}$, and we represent the route node that follows node (f, s) as $(f, s + 1)$. Given that the scheduling algorithm does not adjust an aircraft's planned speed and route, both the sector traversal times and the order of the route nodes must be preserved. We call this constraint the *precedence constraint*, and it can be written as

$$t_{(f,\,s+1)} - t_{(f,\,s)} = \Lambda_{(f,\,s)}. \tag{2}$$

We see in Figure 3 that our initial schedule implies that $t_{(f,\,s_3)} - t_{(f,\,s_1)} = 10$. This ten minute traversal time for sector s_1 (written as $\Lambda_{(f,\,s_1)} = 10$) must be retained in any schedule revision.

3.3 The Hotspot Constraint

While constraints (1) and (2) ensure that we limit solutions to delaying flights, we have yet to constrain against hotspots. Each sector s has an associated capacity c_s. The workload imposed on air traffic controllers is considered excessive if the number of flights in s at the same time is greater than c_s. We let the set F_s [2] contain all flights that pass through sector s. From this set, we consider each possible pairing (order does not matter). If any pair f and g share s for at least part of their sector traversal time, we set binary quantity x^s_{fg} to 1. In Figure 4, for instance, we see that flights h and i share sector s_6 for five minutes. Hence, $x^{s_6}_{hi} = 1$.

Let K be a subset of F_s and let K have $c_s + 1$ flights. We consider only $c_s + 1$ flights at a time because we want to identify a particular set of flights that violates the constraint, while allowing combinations that do not (see Example 3.3.2). Note that only sectors s_3 and s_6 are traversed by more than two flights over the schedule period. If we assume that the capacity of all our sectors $c_{s_{1-7}}$ is two, only these two sectors have this subset K. The hotspot constraint can be written as follows:

$$\sum_{\{f,\,g\} \subseteq K} x^s_{fg} \leq \binom{c_s + 1}{2} - 1, \quad K \subseteq F_s, \quad |K| = c_s + 1, \quad s \in S \tag{3}$$

3.3.1 Example. Using Figure 4 as an aid, we can check whether a hotspot will form in sector s_6. We find that K has a cardinality of $c_{s_6} + 1 = 3$. This implies that in the inequality

[2]Mannino and Sartor [5] use $\overline{F}$. F_s, however, makes the sector these flights are associated with more apparent.

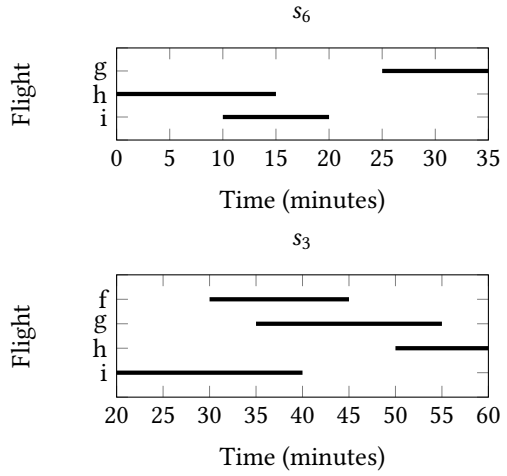

Figure 4. Flight entry and exit times for sectors traversed by more than $c_{s_{1-7}} = 2$ flights over the batch period. Based on data from Figure 3.

in (3) we can consider three flights at a time. We can see in Figures 2 and 4 that F_{s_6} contains three flights over our period. We therefore have that subset K exists for F_{s_6} and that the inequality (3) will be performed for only one group of pairings.

$$x^{s_6}_{gh} + x^{s_6}_{gi} + x^{s_6}_{hi} \leq \binom{c_{s_6}+1}{2} - 1$$
$$0 + 0 + 1 \leq \binom{3}{2} - 1$$
$$1 \leq 2$$

Given that constraint (3) is satisfied, we conclude that there is no hotspot in sector s_6

3.3.2 Example. For sector s_3, $|K| = 3$ but $|F_{s_3}| = 4$. This means that we need to consider three pairings at a time, in four sums (each sum excludes one flight).

$$\begin{aligned} x^{s_3}_{gh} + x^{s_3}_{gi} + x^{s_3}_{hi} &\leq 2 \quad \text{(excluding } f\text{)} \\ x^{s_3}_{fh} + x^{s_3}_{fi} + x^{s_3}_{hi} &\leq 2 \quad \text{(excluding } g\text{)} \\ x^{s_3}_{fg} + x^{s_3}_{fi} + x^{s_3}_{gi} &\leq 2 \quad \text{(excluding } h\text{)} \\ x^{s_3}_{fg} + x^{s_3}_{fh} + x^{s_3}_{gh} &\leq 2 \quad \text{(excluding } i\text{)} \end{aligned} \quad \rightarrow \quad \begin{aligned} 1 + 1 + 0 &\leq 2 \\ 0 + 1 + 0 &\leq 2 \\ 1 + 1 + 1 &\not\leq 2 \\ 1 + 0 + 1 &\leq 2 \end{aligned}$$

Given that constraint (3) is not satisfied, we conclude that there is a hotspot in sector s_6. Note, however, that only the combination of flights f, g, and i results in a conflict. The other three combinations will not be blocked.

3.4 The Disjunctive Constraints

If we take a pair of distinct flights f and g that traverse sector s at some point, only one of the following three conditions may be true:

- Flight f leaves s before g enters (denoted as $y^s_{fg} = 1$).
- Flight g leaves s before f enters (denoted as $y^s_{gf} = 1$).
- Flight f shares s with g (denoted as $x^s_{fg} = 1$).

We can summarize this list and the condition that exactly one statement be true in a new constraint:

$$y^s_{fg} + y^s_{gf} + x^s_{fg} = 1, \quad \{f,\ g\} \subseteq F_s \quad s \in S \tag{4}$$

Referring to flights h and i as they traverse sector s_3 in Figure 4, for example, we have

$$y^{s_6}_{hi} + y^{s_6}_{ih} + x^{s_6}_{hi} = 1$$
$$0 + 1 + 0 = 1$$

For every pair of flights $\{f,\ g\} \subseteq F_s$, the associated parts of the schedule t (Figure 3) need to satisfy the disjunctive constraint (4) as modeled in a conjunction of constraints.

$$\begin{aligned} &\text{(i)} \quad t_{(g,\,s)} - t_{(f,\,s+1)} \geq -M(1 - y^s_{fg}) \\ &\text{(ii)} \quad t_{(f,\,s)} - t_{(g,\,s+1)} \geq -M(1 - y^s_{gf}) \\ &\text{(iii)} \quad t_{(g,\,s+1)} - t_{(f,\,s)} \geq -M(1 - x^s_{fg}) \\ &\text{(iv)} \quad t_{(f,\,s+1)} - t_{(g,\,s)} \geq -M(1 - x^s_{fg}) \\ &\text{where} \quad y^s_{fg},\ y^s_{gf},\ x^s_{fg} \in \{0,\ 1\} \end{aligned} \tag{5}$$

The constant M, known as Big-M, is a large number introduced to simulate $-\infty$. Because this is a conjunction of *constraints*, the purpose of Big-M is to make the inequalities hold for any entry and exit times suggested, except those that do not satisfy the properties the set Boolean represents. If we want to be assured that the times suggested are feasible in a solution where $y^s_{fg} = 1$, the entry time of g must be greater than the exit time of f. As the right-hand side of (i) is zero in this case, the constraint will not allow the time as f enters the sector following s to be greater than the time g enters sector s.

If we consider inequality (ii) with the same pair of flights and the same sector, even though subtracting the entry time of the first flight from the exit time of the second flight yields a negative number, with a suitably large constant M, the inequality will still hold. This means that (ii) cannot change the solution: any times that satisfy (i) can be considered in the conjunction, but times that do not satisfy it (those that make the left-hand side negative), will invalidate the entire conjunction and clip off any other inequalities in it that might still hold. Similar cases can be made for (iii) and (iv).

This concludes the standard MILP formulation. Any schedule allowed by the constraints 1 - 5 will ensure flights leave at of after their departure time, traverse sectors at the speeds planned, do not cause hotspots, maintain a logical sequence, and have sector entry times consistent with all prior constraints. The objective, then, is to return the feasible schedule in which the sum of all delays is the lowest. The MILP solver achieves this by considering the sets of t, x, and y variables satisfying the constraints, and by finding the set in which the sum of all times t is smallest.

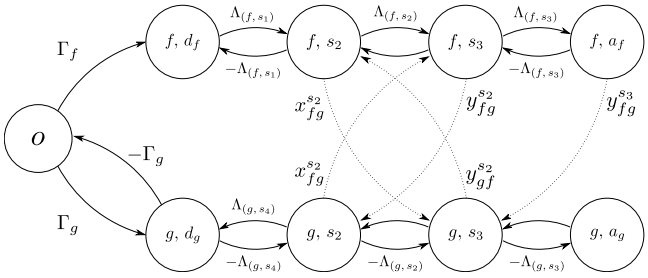

Figure 5. A disjunctive graph model of the hotspot problem, based on Mannino et al. [4], [5]. Activation edges and notation modified to suit the examples used in this paper.

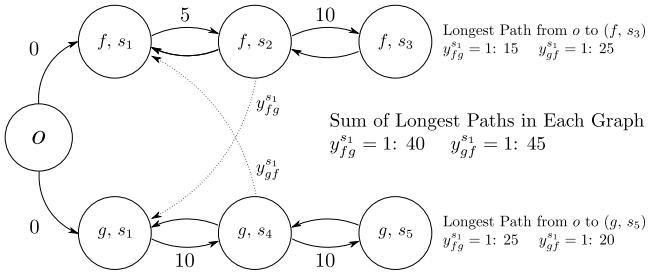

Figure 6. A disjunctive graph with a pair of flights f and g that both traverse a sector s_1 with capacity $c_{s_1} = 1$. The sum of the longest traversal times for each y edge is given to the right. The optimal solution occurs when this sum is lowest, in this case when edge $y^{s_1}_{fg}$ is active. $-\Lambda$ values omitted for readability.

4 The Path&Cycle Formulation

While the standard formulation in Section 3 correctly models and constrains the hotspot problem, the use of Big-M coefficients poorly bounds the integer search trees, making solution times slow for realistic scenarios. To eliminate Big-M constraints, Mannino and Sartor [5] introduce a formulation based on paths and cycles formed in a disjunctive graph that models the hotspot problem. The graph observations can be written as a MILP problem that can be solved by any solver capable of computing the standard MILP formulation in the previous section.

4.1 Disjunctive Graphs

We can draw the Big-M formulation in the previous section as a directed graph (Figure 5). Each route node becomes a vertex $u \in R$, and we add an origin vertex o as well. For a flight f, the directed edge from o to the departure vertex $(f,\ d_f)$ has a weight corresponding to the minimum departure time Γ_f. The precedence constraint (2) is represented by a pair of directed *precedence edges* between two route nodes u and v. The edge from u to v has weight Λ_u, while the edge from v to u has weight $-\Lambda_u$. As in the previous section, a route node is written as a flight and sector pair $(f,\ s)$. Edges with weight 0 link each pair of flights that at some point traverse the same sector: $(f,\ g) \in F_{s_2}$, for instance. These edges are associated with Big-M constraint (5). If (5.i) is active ($y^s_{fg} = 1$), we can draw a *conflict edge* $(u,\ v)$, where u and v are vertices related to $t_{(f,\ s+1)}$ and $t_{(g,\ s)}$ respectively. Figure 5 illustrates all possible conflict edges for sector s_2. Constraint (4) is used in the new formulation as well, however, ensuring that a feasible solution has only either a pair of x conflict edges active or else a single y edge active per mutually visited sector.

Rather than constrain entry and exit times for every sector, as in Big-M constraint (5), the new formulation only varies conflict edge activation. Rules regarding these activations make only feasible combinations of active edges possible. In other words, contradictions such as a flight entering a sector both before and at the same time as another flight are not permitted. In a graph, such contradictions manifest as positive directed cycles. In Figure 5, flights f and g both traverse sectors s_2 and s_3. If we attempt to make flight g traverse s_2 first ($y^{s_2}_{gf} = 1$), we expect flight f to enter s_2 only after g enters s_3. In this case, it is not possible to make flight f traverse s_3 before g because flight f is behind and not allowed to enter until g leaves. (The node sequence shows that flight g is not allowed to exit the sector pair to wait for flight f.) In the graph, this situation is modeled by activating conflict edges $y^{s_2}_{gf}$ and $y^{s_3}_{fg}$. Note that we get a cycle from node $(g,\ s_3)$ through $(f,\ s_2)$, $(f,\ s_3)$, and $(f,\ a_f)$ back to $(g,\ s_3)$. All precedence edges taken on this cycle are positive, and the cycle therefore has a positive traversal length and is not permitted by the cycle constraint.

Net negative cycles, however, are allowed. In Figure 5, for example, flight g has both a positive and negative departure edge. If conflict edge $y^{s_2}_{fg}$ is active, a cycle can be formed from the origin o through the first three f nodes and through two g nodes back to o. The cycle is only permitted if the positive path through the f nodes is shorter than the negative path through the g nodes, a condition that depends on the minimum departure times and precedence edge values of f and g. In effect, f is only allowed to traverse s_2 first if g arrives to s_2 later than f. The existence of a negative departure time edge for g thus gives the flight priority over other aircraft in the system. This is useful for accomodating a flight from an external jurisdiction, whose departure time is not controlable by the local authority. Flight f, in contrast, has only a positive departure edge and can be rescheduled as needed.

Mannino and Sartor [5] state that the objective function of the Path& Cycle formulation is to minimize the sum of the longest traversal paths from the origin o to every arrival node. The graphs evaluated must satisfy a hotspot constraint, constraint (4) from the previous formulation, and a constraint against positive directed cycles. Figure 6 considers a pair of flights f and g that both traverse sector s_1. If $c_{s_1} = 1$, no x edges are possible, and because both flights request to enter

s_1 at the same time ($\Gamma_{f,g} = 0$), one flight must be delayed. If conflict edge $y^{s_1}_{gf}$ is active, the longest path from o to arrival node (f, s_3) proceeds through the first sector of g, the active conflict edge, and all f nodes. The longest path from o to (g, g_3) traverses all flight g's nodes.

Figure 6 shows a similar process for when conflict edge $y^{s_1}_{fg}$ is active. The smallest longest path sum occurs when $y^{s_1}_{gf}$ is active, a result that is consistent with the fact that flight f traverses s_1 more quickly than g. The delay information for each flight can be extracted from the length of the longest path to the arrival node of each flight for the optimal conflict edge solution. In Figure 6 we can see that the longest path to the arrival node of flight f is unmodified from the original departure time proposal. The longest path to the arrival node of flight g, however, is five minutes longer than the shortest traversal, implying that flight g must be delayed by five minutes to avoid sharing sector s_1 with flight f.

For a derivation of the graph formulation and proofs of the properties used, see [3]. For MILP formulation of the graph constrains, see [5]. An expansion of the Path&Cycle formulation that includes more definitions of sector capacity is found in [4].

5 Results and Conclusions

Mannino and Sartor [5] implemented the standard and Path&Cycle formulations. They used the CPLEX 12.8 integer programming solver and ran their formulations on a single thread. The authors generated 30 scheduling scenarios in a simulated world containing 400 sectors and 20 airports. For each run, a pair of airports was randomly selected and flights were randomly scheduled from one airport to the other.

In Table 1 we can see how the Path&Cycle formulation significantly reduces the number of branch and bound nodes evaluated by the algorithm before reaching an optimal solution. These reductions become particularly important as the size of set the of schedules submitted increases, and they suggest that the new formulation markedly tightens the solution space. While we can see consistent improvements in run times, it is worth noting that the largest performance differences occur between the worst instances of each formulation, where the Path&Cycle approach takes only 1.35 seconds to compute an optimal schedule, while the standard formulation takes over 50 seconds. This strongly suggests that the Path&Cycle formulation produces a feasible region that is less complex and avoids large search trees in the mixed integer linear programming solver.

ID	|F|	c_s	Solved hotspots		Visited nodes		Time (s)		Speed up
			PC	BF	PC	BF	PC	BF	
ATM1	122	3	13	13	1016	19175	1.06	4.99	4.7x
ATM2	137	3	13	13	3062	36806	1.35	10.38	7.7x
ATM3	131	3	8	8	109	774	0.18	0.28	1.5x
ATM4	142	3	13	13	833	40482	0.73	6.43	8.8x
ATM5	110	3	12	12	795	31117	0.39	7.38	18.8x
ATM6	127	3	5	5	79	570	0.16	0.17	1.1x
ATM7	115	3	1	1	0	5	0.05	0.05	1.0x
ATM8	120	3	4	4	2	97	0.05	0.10	1.8x
ATM9	131	3	4	4	42	554	0.08	0.16	2.1x
ATM10	143	3	8	8	76	2313	0.18	0.48	2.6x
ATM11	136	3	15	15	371	39300	0.31	14.90	47.3x
ATM12	142	3	9	9	274	1974	0.22	0.57	2.6x
ATM13	139	3	11	11	118	2217	0.19	0.94	5.1x
ATM14	126	3	7	7	60	2182	0.13	0.59	4.6x
ATM15	139	3	9	9	3625	172950	0.88	50.61	57.4x
ATM16	288	5	4	4	47	1579	0.27	0.71	2.6x
ATM17	289	5	9	9	113	12503	0.38	6.05	16.0x
ATM18	278	5	6	6	183	2188	0.37	1.23	3.3x
ATM19	259	5	3	3	0	296	0.15	0.20	1.4x
ATM20	254	5	5	5	55	1977	0.23	1.01	4.3x
ATM21	279	5	6	6	255	4175	0.32	2.10	6.6x
ATM22	287	5	3	3	0	985	0.11	0.32	2.8x
ATM23	259	5	6	6	37	2452	0.25	1.00	4.0x
ATM24	281	5	4	4	161	1350	0.47	0.60	1.3x
ATM25	296	5	4	4	71	1518	0.22	0.60	2.7x
ATM26	275	5	7	7	50	2872	0.41	1.32	3.2x
ATM27	256	5	5	5	464	5042	0.46	1.58	3.5x
ATM28	273	5	6	6	298	1542	0.60	0.91	1.5x
ATM29	274	5	6	6	193	104627	0.53	35.09	66.7x
ATM30	287	5	7	7	1306	9129	0.75	3.10	4.1x

Table 1. Mannino and Sartor [5]. A comparison of the Path&Cycle formulation (PC) and the standard Big-M formulation (BF), in run times and visited nodes, for randomly generated flights in a 400 sector world.

6 Acknowledgments

I would like to thank Dr. Elena Machkasova for her feedback on drafts and for the many hours of support and advice she offered. I would also like to thank Brian Goslinga for his detailed feedback on an earlier draft.

References

[1] Stephen P. Bradley. 1977. *Applied mathematical programming.* Addison-Wesley Pub. Co., Reading, Mass. http://web.mit.edu/15.053/www/AMP-Chapter-09.pdf

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[3] Leonardo Lamorgese and Carlo Mannino. 2019. A Noncompact Formulation for Job-Shop Scheduling Problems in Traffic Management. *Operations Research* 67, 6 (November 2019), 1586–1609. https://doi.org/10.1287/opre.2018.1837

[4] Carlo Mannino, Andreas Nakkerud, and Giorgio Sartor. 2021. Air Traffic Flow Management with Layered Workload Constraints. *Computers and Operations Research* 127 (03 2021), 105159. https://doi.org/10.1016/j.cor.2020.105159

[5] Carlo Mannino and Giorgio Sartor. 2018. The Path&Cycle Formulation for the Hotspot Problem in Air Traffic Management. In *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018) (OpenAccess Series in Informatics (OASIcs), Vol. 65)*, Ralf Borndörfer and Sabine Storandt (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 14:1–14:11. https://doi.org/10.4230/OASIcs.ATMOS.2018.14