Branch & Bound Global-Search Algorithm for Aircraft Ground Movement Optimization

Pushkar J. Godbole[*], Abhiram G. Ranade [†], Rajkumar S. Pant [‡]

Indian Institute of Technology - Bombay, Powai, Mumbai, 400076, India

Optimal aircraft ground scheduling is a well known NP-Hard problem and hence many heuristics are used to generate schedules within realistic run-times. These heuristics are designed to run fast, but often do not promise any guarantee about the solution quality. Inspired from the railway scheduling algorithms by D'Ariano et al.[1] and Mannino et al.[2], this paper introduces a Branch & Bound based aircraft routing and scheduling approach with guaranteed global optimality; as a real-time decision support tool for Air Traffic Controllers.

The performance of the algorithm has been benchmarked against various others such as the Mixed Integer Linear Programming (MILP) approach by Roling et al.[3], Bacterial Foraging heuristic by Baijal et al.[4]. The configuration agnostic design of the algorithm makes it suitable for applications to unconventional airport designs, such as the cross-runway map of the Mumbai International Airport. The globally optimal nature of the solution exhibits a distinct improvement over the respective solutions while maintaining minimal run-times.

I. Introduction

The throughput of an airport in terms of the the number of aircraft handled, depends on the airport's infrastructure capacity and operational efficiency. Due to heavy increase in air traffic, many of the airports today work at the brink of their infrastructural capacity thus making efficient operations a significant determining factor.

[*]Department of Aerospace Engineering, IIT Bombay, Email: pushkar.godbole@iitb.ac.in, AIAA member
[†]Department of Computer science and Engineering, IIT Bombay, Email: ranade@cse.iitb.ac.in, AIAA non-member
[‡]Department of Aerospace Engineering, IIT Bombay, Email: rkpant@aero.iitb.ac.in, AIAA member

Delays in departures and arrivals for individual aircraft, can be caused by a plethora of factors such as weather, visibility, ground crew availability, aircraft readiness, passengers and so on. But in an ideal scenario, where these delays do not exist, some aircraft would still have to run late. This is because unlike trains, aircraft do not follow a fixed schedule. For instance, if 9:00pm happens to be a high demand time for the majority of passengers, all airlines would try to schedule their aircraft at 9:00pm, irrespective of the airport capacity; so that when a customer looks for aircraft departing at 9:00pm, all of them may get listed. However, an airport can physically handle only a fixed number of take-offs/landings at a time, which inevitably leads to push-backs for some aircraft. Thus it can be seen that delay in this problem is *systemic* and the objective is only to minimize it.

Although substantially efficient, the process of Air Traffic Control (ATC) remains majorly manual and empirical in nature. The downside of this is particularly observed in heavy traffic hours which make the manual decision making significantly difficult to handle, leading to myopic solutions. The objective of this work is to develop an algorithm that could generate optimal schedules on the fly, for given aircraft within a limited time window and serve as a real-time assistance tool in the ATC's decision making process.

Aircraft scheduling poses itself as an NP-Hard problem with a large solution space. This makes deterministically finding a globally optimal solution within an acceptable run-time challenging. Hence various stochastic and semi-deterministic algorithms have been developed and investigated to tackle the problem in a realistic run-time. Smeltink et al.[5] use a Mixed Integer Linear Programming (MILP) model to generate a locally optimal schedule on a predetermined path allocation for aircraft. They use the concept of *Rolling Horizons* to subdivide the problem into briefly solvable sub-problems. The disadvantage of this algorithm is that it uses fixed paths. Therefore re-routing in case of conflicts is not possible. Roling et al. [3] also use MILP similar to Smeltink for scheduling. Although rerouting of aircraft is possible in this model, it leads to steep increase in the number of decision variables thus making it difficult to be solved on the fly. Montoya et al.[6] also follow an MILP based method with limited rerouting, however they use a mesoscopic approach to scheduling which may lead to coarser solutions. Baik et al.[7] use Time Dependent Shortest Path (TDSP) to schedule each aircraft individually. The aircraft are ordered based on a predetermined priority and scheduled one by one. The one by one scheduling leads to myopic schedules just as in case of manual scheduling. Gupta et al. [8] use a combination of the two methods by Smeltink and Baik.

In this approach, TDSP is used for routing of the aircraft which is then fed to the MILP model similar to that by Smeltink.

Amongst the stochastic approaches, Gottland et al.[9] use a combination of Genetic algorithm and A* search to generate the schedule. The Genetic algorithm is used for aircraft prioritization and path allocation. A* is used to compute the fitness function of the Genetic algorithm while resolving conflicts. Baijal et al.[4] use Bacterial Foraging on a fixed set of path allocations generated using Dijkstra's shortest path algorithm to resolve conflicts and generate the schedule.

All the algorithms attempt to minimize the total taxi time including the waiting time of all aircraft. Different strategies have thus been investigated to determine an optimal schedule although all schemes have an evident trade-off between run-time and solution quality; and none provides a testable exact solution.

A very similar problem to aircraft ground scheduling is that of railway routing and scheduling. These problems are complementary to each other in many respects and belong to a class of optimization problems known as job-shop scheduling. In this class of problems, certain jobs (flights in this case) have to be scheduled across a limited set of machines/resources (runways and taxiways) so as to maximize the throughput and minimize the delay. Branch & bound is one of the standard methods used to tackle such problems.

In this paper, we present an adaptive Branch & Bound based algorithm inspired by the papers on railway scheduling by D'Ariano et al.[1] and Mannino et al.[2]. Their algorithms however do not allow re-routing of trains which divests them from exploring the entire solution space. This algorithm facilitates re-routing of aircraft thus ensuring global optimality. The next section describes in detail the problem setup, inputs and assumptions. Section III explains the design of the optimization model while section IV illustrates the results and performance of the algorithm.

II. Problem Setup

Given the information about the entry-time, origin and destination points for a set of aircraft on an airport, the objective of the optimizer is to generate a conflict free schedule so as to minimize the overall taxi and waiting time while respecting the safety regulations.

A. Modeling

The two key elements to be modeled in the problem are the Airport and the Aircraft. While developing a rigorous and fast approach to the aircraft ground scheduling problem, care has been taken to maintain adequate realism in the modeling.

a. Airport

As is commonly done in this class of problems, the network of runways and taxiways on the airport is modeled as a directed graph with runways and taxiways discretized into segments. The segments are called 'arcs' and links between the arcs are the 'nodes'. Every segment has four attributes namely, the two end nodes, length and arc type (based on whether the arc belongs to a runway or a taxiway). An aircraft at any point on an arc is said to be occupying that arc. Also, if an aircraft waits during its taxi operation, the waiting is defined on the arc and not at a point on the map/graph.

b. Aircraft

The aircraft is modeled as a point object traveling through the graph. The aircraft has seven attributes as follows:

1. **Start time:** Time when the aircraft enters the map.
2. **Origin:** Node from where the aircraft enters the map.
3. **Destination:** Node from where the aircraft exits the map.
4. **Aircraft speed:** Speed of the aircraft as it traverses its path. In the current implementation, the speed of the aircraft is assumed constant all across the map. Also, the aircraft is assumed to have infinite acceleration (which is to say that the aircraft is either moving at a constant speed or waiting).
5. **Trailing separation:** The minimum distance that has to be maintained by a trailing aircraft following this aircraft in order to mitigate the effects of turbulence. This distance is larger for larger aircraft.

6. **Landing/Take-off distance:** The minimum distance an aircraft has to traverse on the runway (before take-off/after landing). While choosing paths from the pool of possible choices for an aircraft, only the paths with sufficient runway length (based on this parameter) are considered.

7. **Priority:** The weight of the aircraft in the cost function. Aircraft with higher priority are preferred over those with a lower priority. Thus a high priority aircraft would tend to have lower delays. Landing aircraft are generally assigned higher priority (as circling in the sky is significantly costlier as compared to waiting on ground).

B. Constraints

The constraints to be imposed on the aircraft for a conflict free feasible schedule can be categorized into two types: Conjunctive and Disjunctive constraints. The conjunctive constraints are *fixed* LP (Linear Programming) type constraints that represent physical path traversals for an aircraft.The disjunctive constraints are *either-or* type constraints that establish precedence relations between occurrence of two events. These constraints exist in pairs and are resolved when exactly one of the the two conditions is satisfied. In the following section, the first (Travel time) constraint is conjunctive while the subsequent ones are disjunctive.

t_{ik} : Time when aircraft i reaches node k

a. Travel time

An aircraft should not traverse an arc at speed greater than its speed limit. Travel time on an arc should be greater than or equal to ${}^{(arc\ length)}/_{(aircraft\ speed)}$.

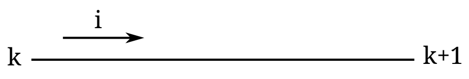

Figure 1: Travel time

$$t_{i(k+1)} - t_{ik} \geqslant \frac{length_{(k,k+1)}}{speed_i}$$

b. Head-on collision

Two aircrafts entering an arc from opposite directions must have no temporal overlap. i.e. if two aircraft i and j enter an arc from opposite directions, either i should enter only after j has completely exited or visa-versa.

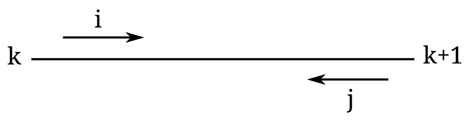

Figure 2: Head-on collision

$$If\ i\ leads:\ \ t_{j(k+1)} - t_{i(k+1)} \geqslant 0$$

or

$$If\ j\ leads:\ \ t_{ik} - t_{jk} \geqslant 0$$

c. Common node 1

Let sep_i and sep_j be the trailing separations for two aircraft i and j respectively. If aircraft i and j share the first node k of their succeeding arcs, and say i follows j.

Then if the length of the arc of flight j (i.e. arc (k,m)) is less than sep_j, then i must enter its arc (i.e. arc (k,l)) only after j has exited its arc (i.e. arc (k,m)).

If length of arc of flight j (i.e. arc (k,m)) is greater than or equal to sep_j, i may enter its arc (i.e. arc (k,l)) after j is at least at a distance of sep_j from node k and visa-versa.

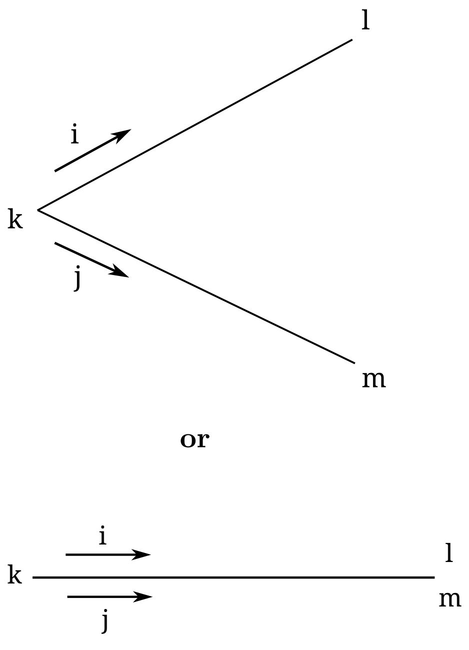

Figure 3: Common node 1

If i leads:

$$If\ length_{(k,l)} < sep_i:\ \ t_{jk} - t_{il} \geqslant 0$$

or

$$If\ length_{(k,l)} \geqslant sep_i:\ \ t_{jk} - t_{ik} \geqslant \frac{sep_i}{speed_i}$$

or

If j leads:

$$If\ length_{(k,m)} < sep_j:\ \ t_{ik} - t_{jm} \geqslant 0$$

or

$$If\ length_{(k,m)} \geqslant sep_j:\ \ t_{ik} - t_{jk} \geqslant \frac{sep_j}{speed_j}$$

d. Common node 2

Let sep_i and sep_j be the trailing separations for two aircraft i and j respectively. If aircraft i and j share the second node k of their succeeding arcs, and say i follows j.

Then if the length of the arc of flight i (i.e. arc (l, k)) is less than sep_j, then i must enter its arc (i.e. arc (l, k)) only after j has exited its arc (i.e. arc (m, k)).

If length of arc of flight i (i.e. arc (l, k)) is greater than or equal to sep_j, i must be at least sep_j units away from node k on its arc (i.e. arc (l, k)) when j exits its arc (i.e. arc (m, k)) and visa-versa.

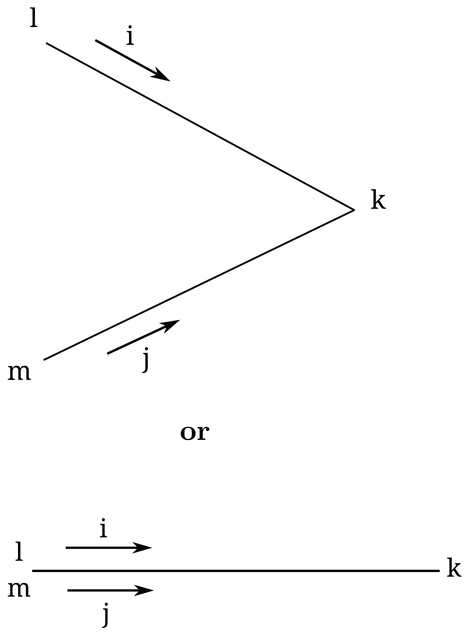

Figure 4: Common node 1

If i leads:

$$If\ length_{(m,k)} < sep_i: \quad t_{jm} - t_{ik} \geqslant 0$$

or

$$If\ length_{(m,k)} \geqslant sep_i: \quad t_{jk} - t_{ik} \geqslant \frac{sep_i}{speed_j}$$

or

If j leads:

$$If\ length_{(l,k)} < sep_j: \quad t_{il} - t_{jk} \geqslant 0$$

or

$$If\ length_{(l,k)} \geqslant sep_j: \quad t_{ik} - t_{jk} \geqslant \frac{sep_j}{speed_i}$$

e. Runway

No two aircraft can simultaneously occupy the same active runway. Either one has to exit before the other enters. This constraint is particularly crucial from a safety point of view as flight speeds reach their maximum on runways.

Say for two aircraft i, j the horizontal path is the runway. i enters the runway at k and exits at l while j enters the runway at k and exits at m. If j leads i, then i can enter node k only after j exits node m and if i leads j, then j can enter node k only after i exits node l.

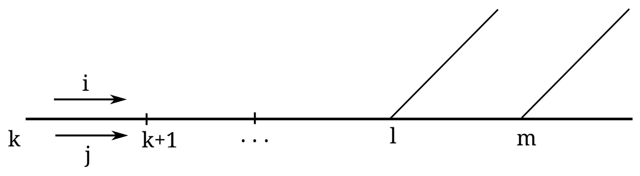

Figure 5: Runway

$$If\ i\ leads: \quad t_{jk} - t_{il} \geqslant 0$$

or

$$If\ j\ leads: \quad t_{ik} - t_{jm} \geqslant 0$$

Constraint (a) is applied to every arc in the flight-path while constraints (b), (c) and (d) are applied to taxiway arcs. Constraint (e) is applied to runways. Since this constraint supersedes (b), (c) and (d), these constraints need not be applied to runways.

C. Objective

The objective of the optimization is to generate a schedule while satisfying all the aforementioned constraints, such that it minimizes the summation over the destination times of all aircraft, weighted by priority.

$$Minimize: \sum_{i=0}^{N} P_i t_{in}$$

The output would be, values of t_{ik} for $i \in (0, N)$ & $k \in (0, n)$.

t_{ik} : Time when aircraft i reaches node k

P_i : Priority level of aircraft i.

N : Total no. of flights

n : Last / Destination node in the path of flight i

III. Optimization model

The aircraft schedule is a function of two criteria: The path taken by the aircraft and its precedence relation with other aircraft. The optimizer therefore has two fundamental tasks towards generating the globally optimal schedule: Routing and scheduling. The approach used here for optimization does not progress in time but rather in solution cost. The algorithm starts-off from a potentially feasible conflicted solution and resolves the conflicts while simultaneously evaluating the partial cost, until a feasible solution is reached. Thus the path generation and scheduling processes are delinked from each other and hence can be implemented as such.

A. Routing

The routing routine is responsible for generating all valid paths for each flight and feeding them to the scheduler.

a. Route generation

The route generation routine uses Depth First Search (DFS) to search the graph for all simple paths (paths without cycles) between the aircraft's origin and destination nodes. Looking only for simple paths ensures that the DFS algorithm does not get trapped in a loop and generates finite number of paths. Besides, paths containing cycles can not be considered to be valid paths for aircraft routing. The DFS simple path search algorithm is run for every flight.

Only those paths containing sufficient runway length (w.r.t. the aircraft's landing/take-off distance) are added to the valid path pool. The runway constraint (e) renders the runway blocked for other aircraft, whenever an aircraft enters it. Hence an aircraft is only allowed to use the runway once during its entire path. That is, a runway can not be used as a taxiway. Not only does this rule ensure realism of the solution, but also shrinks the path pool of an aircraft to valid choices thus reducing the run-time.

b. Combination generation

Once all valid routes have been identified for each aircraft, the combination generation routine generates all possible flight-path combinations in order to span the entire feasible solution domain. These combinations are then fed to the scheduler in order of increasing path lengths (This ensures that the global optimum be identified early on, as shorter paths *in general* imply better solutions).

B. Scheduling

In order to identify the global optimum, the scheduler has to evaluate every flight-path combination for optimality and deliver the best solution. For a particular flight-path combination, the scheduler has to resolve all conflicts to generate a feasible solution with least cost. The global optimum would be a schedule (with a particular flight-path allocation) that gives the minimum cost of the objective function.

a. MILP & Motivation

Initially, an MILP (Mixed Integer Linear Programming) based brute force approach was tried for scheduling; in which, for every flight-path combination, the overlaps (conflicts) between aircraft were identified. An MILP was generated for these conflicts and fed to a LP (Linear Program) solver (CPLEX)[10]. The MILP solver delivered an optimal (exact) solution for that particular flight-path allocation. The minimum cost over all such combinations was finally identified as the global optimum. Although this approach delivered the requisite global optimum, the run-time of the optimization was very high and increased exponentially with the problem size, rendering it unsuitable for real-time usage. It was however observed that merely feeding the flight-path combinations to the scheduler in order of increasing path lengths, led to early identification of the global optimum. Rest of the run-time was consumed in proving the global optimality of this solution. In this brute-force approach, every flight-path combination was evaluated independently by running a complete MILP solver on each, despite the global optimum being identified at an early stage. Whilst MILP is a good approach for fixed path problems, it is not suitable for multiple path allocations due to its memorylessness w.r.t. previous computations (at least in the context of this approach). The need to customize the solver and utilize the early identified global optimum to reduce subsequent computational effort led to the development of the Branch & Bound approach.

b. Branch & Bound

In this approach, aircraft scheduling is treated as a job-shop scheduling problem. This job-shop problem is solved for every flight path combination and the cost of best solution found yet is shared across all as the upper bound. The Upper bound is initially set at infinity and then updated through the scheduling process as new and better solutions are identified. For a given flight-path combination, overlap constraints (based on section 2-B) are generated and resolved using Branch & Bound. Similar to D'Ariano et al. [1] and Mannino et al.[2], the constraint resolution problem is modeled as a disjunctive graph and solved as such.

1. Disjunctive graph

A disjunctive graph is constructed from the path choices and overlaps. Every node numbered (i, j) in the disjunctive graph represents the event of aircraft i reaching node j. Every edge in this graph represents a constraint and the edge length represents time consumed in executing the process. A dummy origin node represents the starting point for all aircraft. Conjunctive branches corresponding to each aircraft originate from the dummy origin node and end at the respective destination nodes. The edge connecting the dummy origin node to every flight's starting node has a length equal to that flight's starting time. The subsequent conjunctive edges represent the corresponding travel time constraints (2-B-a). Disjunctive edge pairs (for every flight pair) connect overlapping nodes to establish precedence constraints (2-B-b,c,d,e).

Constraint resolution:

For two flights i and j starting at node 1 and reaching node 3 with trailing separations of 10 units, starting times of 0 and 4 units and speeds of 1 and 2 units respectively, the disjunctive graph looks as follows:

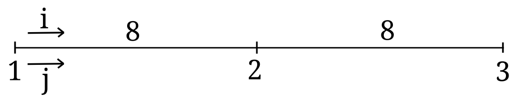

Figure 6: Physical Path

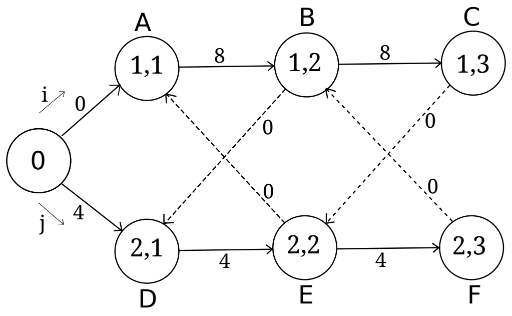

Figure 7: Disjunctive Graph
(Trailing separation $\geqslant$ Arc lengths)

Continuous edges represent conjunctive constraints and dashed edges represent pairs of disjunctive constraints. Edge (B,D) represents that aircraft j can enter node 1 only after aircraft i exits node 2. Edge (E,A) represents that aircraft i can enter node 1 only after aircraft j exits node 2. These edges are paired and represent one disjunctive pair. Only one of these two edges can be retained for a feasible solution to be generated. The scheduler makes the choice of an edge from every disjunctive pair in order to minimize the cost.

The length of a disjunctive edge represents the buffer time to be maintained between the two events. A length of zero implies that the succeeding event can start immediately after the preceding one. If in the above problem, the trailing separations of flights i and j are reduced from 10 units to 3 and 6 units respectively, the disjunctive graph (based on the constraints in 2-B) will change as follows:

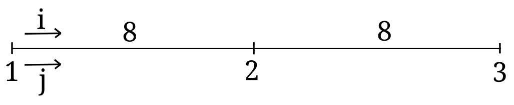

Figure 8: Physical Path

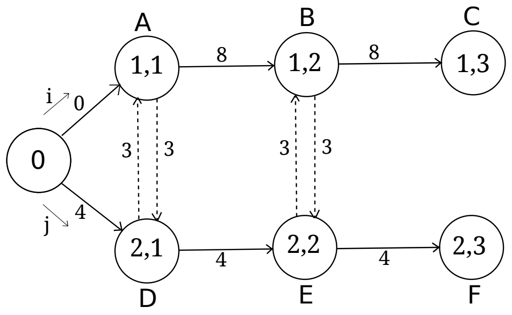

Figure 9: Disjunctive Graph
(Trailing separation < Arc lengths)

The *maximum distance* between two nodes on the disjunctive graph represents the *minimum time difference* to be maintained between the two events for feasibility. A matrix of these nodes represents the distances. Whenever a new disjunctive edge is added, an algorithm (of $O(n^2)$ per edge, n being the number of disjunctive nodes), on the lines of the Floyd-Warshall algorithm is used to update the distance matrix. A feasible solution is the one in which every disjunctive constraint has been appropriately resolved with a choice of an edge from the pair. An optimal solution is a feasible solution in which the objective (summation of the *maximum distance* between the dummy origin node and destination nodes of each aircraft weighted by priority) is *minimum.*

Feasibility:

Not every combination of disjunctive arc choices implies a feasible solution. In figure 7 Choosing arcs (B,D) and (F,B) for the two constraints respectively implies aircraft i leads aircraft j on arc (1,2) while j leads i on arc (2,3) which is infeasible. The disjunctive graph essentially establishes relations between a series of events. The above choice of arcs implies that event B has to occur before event D and after event F which in turn occurs after event D. Infeasibility in a disjunctive graph manifests itself as a cycle (BDEFB in this case) and hence, any solution graph containing a cycle implies infeasibility. Because the distance matrix stores the longest paths between any two nodes, a cycle containing any node i is detected when the the matrix shows a non-zero distance between nodes (i, i).

Constraint grouping:

Although a cycle in the graph implies infeasibility, the converse is not true. Consider the case in figure 9. Choice of edges (A,D) and (E,B) is an infeasible solution although no cycle is formed in this situation. Thus this infeasibility goes unnoticed in the cycle-detection scheme. However, what can be contended is that, the choice of certain disjunctive edges precludes some other choices.[1] Like in this case, choice of edge (A,D) forces the choice of edge (B,E) and the choice of edge (D,A) forces the choice of edge (E,B). If such interconnected constraints are grouped, infeasibility would be avoided. For a pair of aircraft, interconnected constraints occur between consecutive (adjacent) events. The following example exhibits this relation. Aircraft i takes the path 1-2-3-4-5-6-7-8 while aircraft j takes the path 1-2-3-9-10-6-7-8:

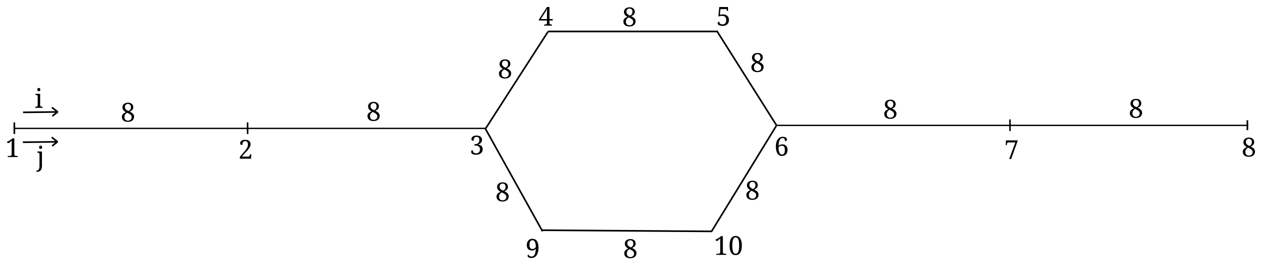

Figure 10: Physical Path

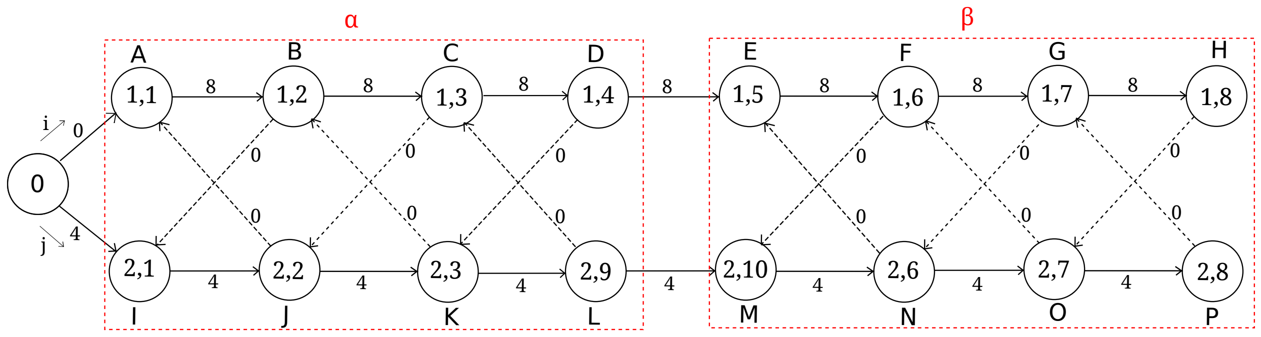

Figure 11: Disjunctive Graph (Constraint grouping)

The adjacency of disjunctive constraints is broken at the nodes D-E and L-M due to split of paths between the two aircraft. Thus the two aircraft have a chance to change their precedence only at nodes 1 and 6 in their path. The break in adjacency of disjunctive constraints implies the formation of two constraint groups marked by *red boxes* in the figure. The problem of feasibility detection in cases such as figure 9 is thus resolved. Besides, because constraint grouping facilities simultaneous resolution of multiple constraints, it also helps in speeding up the optimization.

2. B&B tree

The routing routine feeds the flight path combinations to the Branch & Bound scheduler in order of increasing path lengths. A Branch & Bound tree is constructed for every flight-path combination. After the constraints are identified and grouped, they are fed to the B&B tree for resolution. Every node of this tree represents a partial resolution of the constraints.

Branching:

The root-node of the tree is a disjunctive graph with only conjunctive constraints. The root-node physically implies that, every aircraft would be scheduled independently of others (due to absence of any precedence relations/disjunctive constraints). The partial cost corresponding to the root-node is essentially the value of the objective function without any waiting for any aircraft. Beyond this, the tree is fed the disjunctive constraint groups, one at a time, for resolution. The resolution of a constraint group implies two precedence choices (based on the choice of disjunctive arcs). Thus at every node, a new constraint group is resolved leading to formation of two branches at that node. The partial costs of the two newly formed nodes represent the the updated value of the objective function with the corresponding choice of disjunctive arcs. The B&B tree is thus branched at every node with every level representing a particular constraint group.

Pruning:

A feasible solution is generated only after all the constraint groups have been resolved. Therefore all nodes of the B&B tree (except the leaf-nodes) only represent potentially feasible solutions with partially resolved conflicts. The root-node has the minimum value of cost function due to unresolved conflicts. Resolution of conflicts leads to rise in value of the cost function. Therefore the partial cost of any node represents the lower bound of the schedule for that choice of precedence relations. The upper bound of the cost function is the value of the objective function corresponding to the best solution found yet. Hence if the value of the lower bound of a particular node is found to be greater than the current upper bound, that node is discarded. That node is discarded, because any further exploration of the node would only cause a further rise in the cost function ultimately leading to a worse solution than the best one found yet.

Prioritization:

The performance of a Branch & Bound algorithm entirely depends upon the appropriate balance between branching and pruning; which in turn depends on two criteria: *the order in which the tree-nodes are explored* and *the order in which constraint groups are resolved.*

The ability of a B&B tree to prune itself depends upon the value of the upper bound. Lower the value of upper bound, higher the pruning and hence better the performance. Therefore early identification of good solutions is necessary. Two criteria have been used to prioritize the tree-nodes. Initially when the upper bound is infinite, pruning does not occur as any lower bound is always less than the upper bound (infinity). Hence at this stage, early identification of a feasible solution with a finite cost is necessary. Therefore initially the tree-nodes are prioritized based on their level in the tree, which is same as the number of constraint groups resolved yet. Once a feasible solution is identified and the upper bound lowered to a finite value, the tree-node prioritization criterion is set to the partial cost or the lower bound of that node. Because, a lower value of the lower bound implies a higher potential to lead to a better solution.

Closer the pruning occurs to the root-node, better is the performance of the algorithm. Since pruning occurs when lower bound of a node exceeds the upper bound, prioritizing the constraint groups that lead to maximum increase in partial cost would lead to early pruning. In this problem, since runways are the most utilized and least available resource, they act as bottle-necks in scheduling. Therefore constraint groups containing the maximum runway constraints are prioritized.

The tree is thus branched and pruned based on the relation between the lower and upper bounds. If a leaf-node is reached whose cost is found to be lower than the upper bound, it implies that a better solution has been identified. The upper bound is then updated, which leads to better pruning for the subsequent trees.

IV. Results

The algorithm has been implemented in C++ and Python has been used for solution visualization. The results have been generated on an Intel Xeon Quad Core processor. In the following two sections, the results of the algorithm on problems by Roling et al.[3] and Baijal et al.[4] have been illustrated. The runway constraints (2-B-e) and Landing/Take-off distance criteria have been relaxed in this run to match the constraint model with that mentioned in the corresponding paper.

A. Roling et al.[3] problem solution

Fno.	**O**	**D**	**t_0**	**Sp**	**T_{sep}**	**P**
1	26	15	7	1	4	1
2	24	15	6	2	4	1
3	25	6	10	2	4	1
4	25	6	8	1	4	1
5	25	6	16	2	4	1
6	24	6	14	1	4	1
7	28	26	0	1	4	1
8	28	26	3	1	4	1

Table 1: Roling et al.[3]: Initial conditions

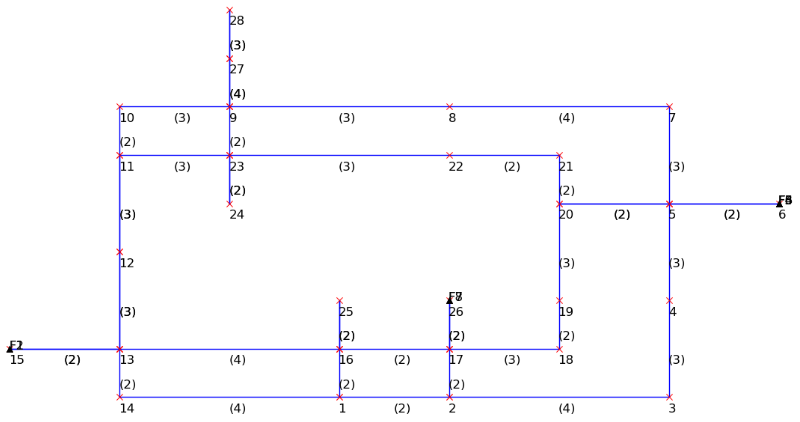

Figure 12: Roling et al.[3]: map

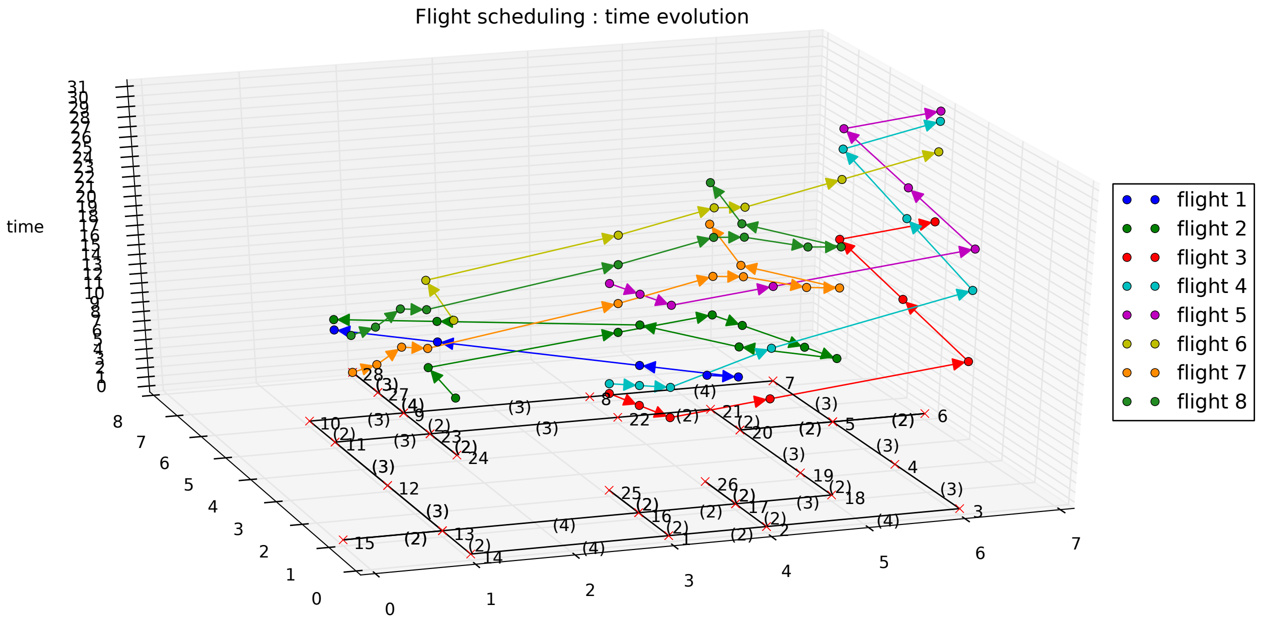

Figure 13: Roling et al.[3]: Time evolution of globally optimal schedule
XY plane: Airport map; Z axis: Time

Roling et al.[3] solution		B&B solution (Global optimum)				
Cost	Runtime	Cost	Time to find optimum	Total Runtime	Flight-path combinations	Improvement in cost
362	NA	204	1 sec	7 sec	7776	75%

Table 2: Roling et al.[3]: Performance

B. Baijal et al.[4] problem solution

Fno.	O	D	t_0	Sp	T_{sep}	P
1	1	8	13	1	3	1
2	1	13	12	1	3	1
3	5	12	6	1	3	1
4	9	2	6	1	3	1
5	1	15	0	1	3	1
6	15	1	0	1	3	1
7	1	8	0	1	3	1

Table 3: Baijal et al.[4]: Initial conditions

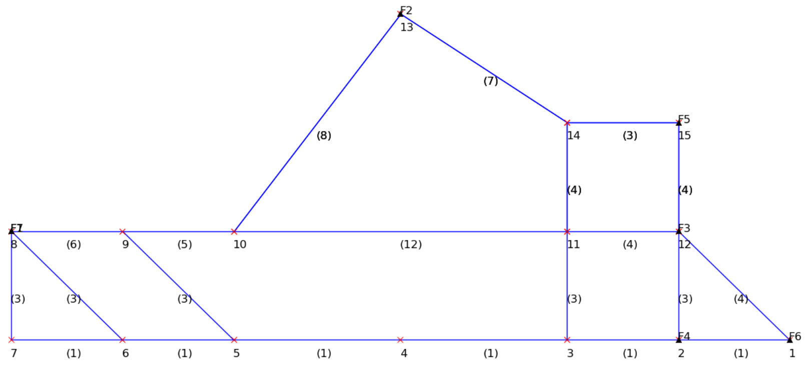
Figure 14: Baijal et al.[4]: Map

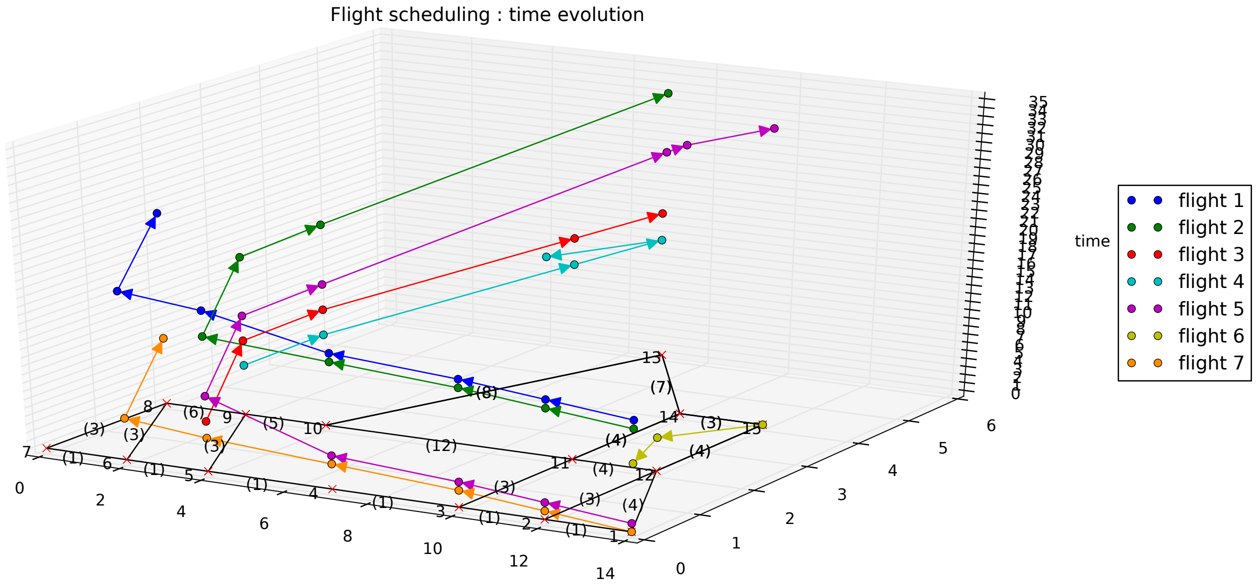

Figure 15: Baijal et al.[4]: Time evolution of globally optimal schedule
XY plane: Airport map; Z axis: Time

Baijal et al.[3] solution		B&B solution (Global optimum)				
Cost	Runtime	Cost	Time to find optimum	Total Runtime	Flight-path combinations	Improvement in cost
186	NA	166	2 sec	18 sec	124416	12%

Table 4: Baijal et al.[4]: Performance

C. Full Mumbai International Airport

A problem on the full map of the Mumbai International airport with crisscross runways was constructed and solved with increasing number of aircraft (without constraint relaxation) using the B&B approach as follows.

Fno.	**O**	**D**	**t_0**	**Sp**	**T_{sep}**	**R_{dist}**	**P**
1	1	14	0	1	3	20	1
2	11	22	10	1	3	20	1
3	36	40	20	1	3	20	1
4	24	36	30	1	3	20	1
5	24	36	40	1	3	20	1
6	36	16	50	1	3	15	1
7	36	40	60	1	3	20	1

Table 5: Mumbai International Airport: Initial conditions

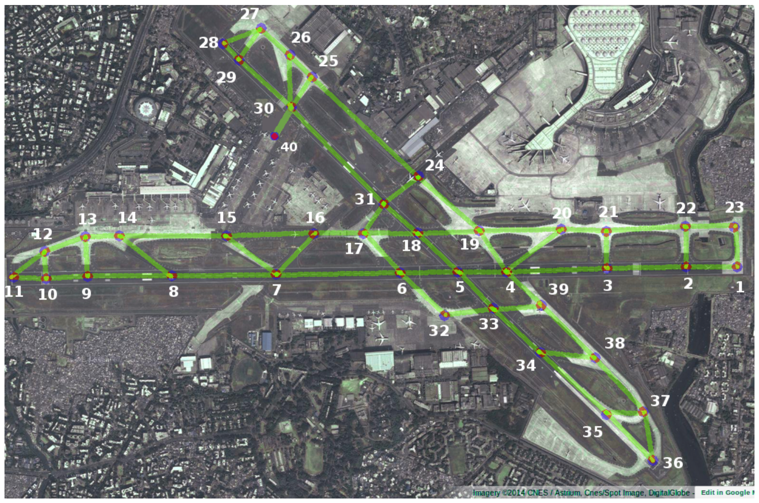
Figure 16: Mumbai International Airport: Map (Google Maps)

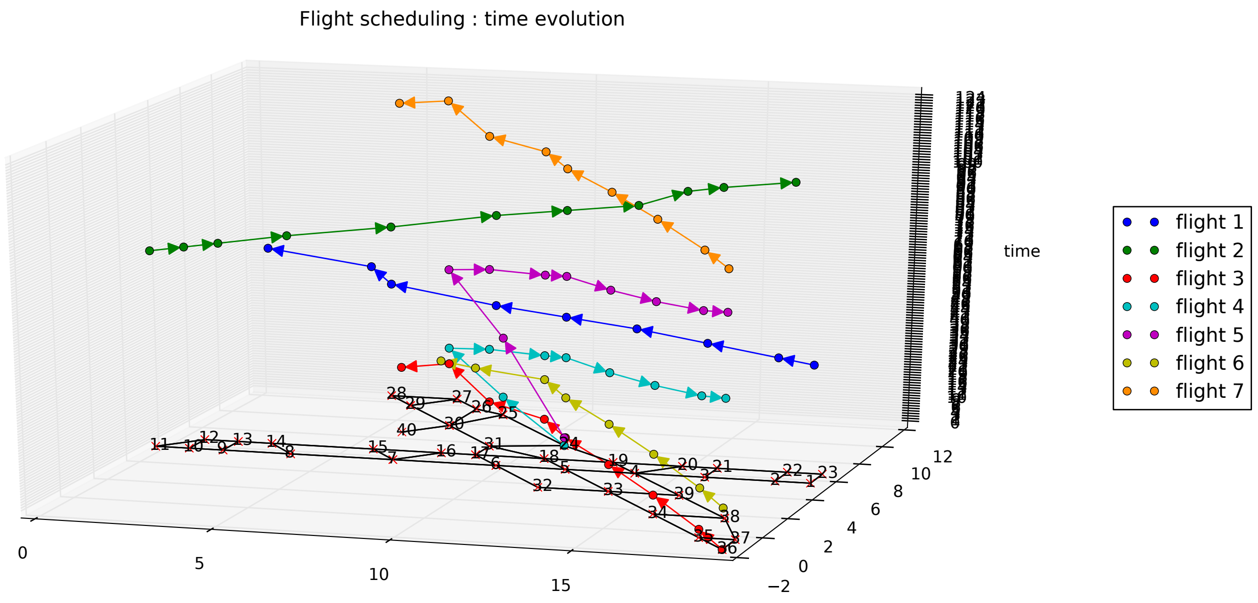

Figure 17: Mumbai International Airport: Time evolution of globally optimal schedule XY plane: Airport map; Z axis: Time

No. of flights	Optimal cost	Flight-path combinations	Time to find optimum	Runtime
4	227	120	0 sec	1 sec
5	335	480	0 sec	5 sec
6	445	1440	0 sec	1 min 14 sec
7	575	1440	2 sec	6 min 45 sec

Table 6: Mumbai International Airport: Performance

Tolerance level:

It can be observed that the total runtime of the optimizer increases rapidly with the number of aircraft. It can also be seen that the global optimum is identified at a very early stage in the run (within a few seconds as desired). Rest of the computation is only performed to prove the global optimality of this solution. The steep rise in runtime with increase in the number of aircraft is attributed to the high amount of branching that occurs at every flight-path combination. This implies that many of the feasible solutions are closely spaced, leading to low pruning and high branching. In order to speed-up the optimality check, a provision to incorporate a *tolerance value* in the optimality validation has been introduced. Running the algorithm on multiple problems with various airport topologies vindicates that the global optimum almost always lies in the shorter path allocations which are evaluated initially. Therefore based on the observations, we postulate that the global optimum most generally is found in the first 1% of the flight-path combinations. Hence the solution space for the first 1% of the flight-path combinations is completely explored after which an optional provision is made to improve the pruning within the bounds of the user defined tolerance value. For problems with less number of flight-path combinations, a lower bound of 10 combinations is set on this 1%. The tolerance is defined in terms of time units per flight.

For instance, if the user defines a tolerance value of two time units per flight for the above problem with seven flights. The potentially optimal solution with value 575 units is identified within the first 1% of the optimization. The upper bound is now set to 575 - 14 = 561. Although 14 time units tolerance is quite small w.r.t. the value of the optimum, it leads to better pruning of the subsequent trees given that many of the solutions are closely spaced, thus reducing the validation time without compromising on the optimality and testability of the solution. In this problem, a total tolerance of 14 units implies that, given the solution with a cost of 575 units, a better solution could lie within the range of 575-561. However non-identification of a solution with a cost lower than 561 implies that such a solution does not exist. The following table for the above problem with 7 flights illustrates the effect of tolerance on the speed of the optimality check.

Tolerance/flight	Optimal cost	Relaxed upper bound	Runtime
0	575	575	6 min 45 sec
2	575	561	4 min 35 sec
4	575	547	2 min 54 sec
6	575	533	1 min 53 sec
8	575	519	1 min 21 sec
10	575	505	58 sec

Table 7: Mumbai International Airport: Performance

V. Conclusions and Future work

Aircraft Ground Movement Optimization has been modeled as a job-shop scheduling problem and solved using a Branch & Bound based approach inspired from the railway scheduling methods used by D'Ariano et al.[1] and Mannino et al.[2]. The primary advantage of this approach over all other approaches investigated is its guaranteed optimality. The optimizer has been developed as a real-time decision support tool for Air Traffic Controllers. In that context, the algorithm is capable of identifying globally optimal schedules on the fly for typical problem sizes. For larger problem sizes, the algorithm is able to generate potentially optimal or near optimal solutions with predefined tolerance limits within few seconds.

Although substantially realistic, the algorithm still lacks in aircraft speed profiling. All aircraft speeds in the current model are assumed to be binary, thus creating a gap between ideal and real operations. Hence introduction of continuously variable aircraft speeds is necessary to improve the veracity of schedules. Secondly, whilst the algorithm is capable of optimally scheduling aircraft within realistic problem sizes, the incorporation of the concept of rolling horizons on lines of Smeltink et al.[5] is crucial. This would enable the algorithm to continuously generate optimized schedules for every aircraft subject to other flights within its spacial and temporal vicinity. The cost based nature of the B&B tree would make this implementation quite straightforward.

References

[1] D'Ariano A., Pacciarelli D., and Pranzo M. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 2006.

[2] Mannino C. and Mascis A. Optimal real-time traffic control in metro stations. *Institute for Operations Research and the Management Sciences (INFORMS), Maryland, US*, 2009.

[3] Roling P. C. and Visser H. G. Optimal airport surface traffic planning using mixed-integer linear programming. *International Journal of Aerospace Engineering*, Volume 2008(Article ID 732828), 2008.

[4] Baijal A. and Pant R. S. Airport ground movement optimization using bacterial foraging algorithm. *Proceedings of International Symposium on Combinatorial Optimization, University of Oxford*, 2012.

[5] Smeltink J. W., Soomer M. J., Waal P. R., and Van der Mei R. D. An optimization model for airport taxi scheduling. *Proceedings of the INFORMS Annual Meeting, Denver, Colo, USA.*, Oct 2004.

[6] Montoya J., Wood Z., Rathinam S., and Malik W. A mixed integer linear program for solving a multiple route taxi scheduling problem. *AIAA Guidance, Navigation, and Control Conference, Toronto, Ontario, Canada*, 2010.

[7] Baik H, Sherali H. D., and Trani A. A. Time dependent network assignment strategy for taxiway routing at airports. *Transport Research Record*, (Paper 02-3660), Oct 2006.

[8] Gupta P., Subramanian H., and Pant R. S. Generation of optimized routes and schedules for surface movement of aircraft on taxiways. *AIAA ATIO/ISSMO Conference, Fort Worth, TX, USA*, 2010.

[9] Gotteland J. B., Durand N., Alliot J. M., and Page E. Aircraft ground traffic optimization. *Proceedings of the 4th International Air Traffic Management R & D Seminar, Santa Fe, NM, USA*, Dec 2001.

[10] CPLEX release 12.5 ILOG. 2012, 2012.