Analaysing The Effect of Uncertainty in Airport Surface Operations

Heron Yang[1] Robert Morris[2] Corina S. Păsăreanu[3]

Abstract— **Designing efficient and robust automated planning and scheduling systems for complex logistics applications presents many challenges, especially if the executing environment is changing unpredictably. In this paper, we study the problem of planning and scheduling airport surface movement at large airports. We describe a fast-time simulation tool that is easily extensible to different scenarios and airports, and is capable of simulating uncertainty in the form of unexpected delays in aircraft movement at different locations. We also describe an approach to scheduling surface movement which is comprised of a deterministic 'rolling horizon' scheduler, paired with heuristic search based on conflict detection and resolution. We summarize a set of experiments designed to quantify the effects of uncertainty on the robustness of schedules and the incurred delays. The system developed is open sourced at `https://github.com/heronyang/airport-simulation`.**

I. Introduction

Airport surface operations present a difficult, large-scale logistics problem with a wide range of sub-problems requiring multi-criteria optimization, including: runway sequencing and scheduling; spot or gate release scheduling; gate allocation and taxi route planning and scheduling [1] [2], [3], [4].

The optimization of airport surface movement is necessary to avoid unnecessary delays and to improve the overall throughput at the airports. However, airport surface movement optimization is known to be hard (NP-hard [5]). Surface movement planning and scheduling is dynamic, with aircraft continuously entering and leaving the operating space. Furthermore, surface movement is unpredictable and prone to unexpected changes in operating conditions due to external factors such as weather. In general, efficiency and safety are difficult objectives to achieve in practice, due to the challenges posed by the presence of uncertainties, human factors, and competing stakeholder interests.

Current practice handles the dual problems of complexity and uncertainty in two ways: continuous planning and scheduling and sub-optimal heuristic scheduling [1]. Furthermore, virtually all planning and scheduling decisions are currently made by human operators. Typically, airport surveillance data and scheduling tasks (for departures and arrivals) provide inputs to planning and scheduling. To handle the dynamics of the operations, schedules are revised continuously (for example, every 15 minutes at busy airports). Second, the complexities of the planning and scheduling problem have until now forced the utilization of heuristic approaches to optimization that require reduced computational overhead while achieving useful results.

In this paper we describe a fast-time simulation tool for studying airport surface operations. The tool is easily extensible to different scenarios and airports, and is capable of simulating uncertainty in the form of unexpected delays in aircraft movement at different locations. In the tool we maintain the general architecture of continuous logistics planning and scheduling for an uncertain executing environment, and investigate automating aspects of the planning and scheduling. The latter have two main components: models that allow for accurate anticipation of unexpected events; and fast algorithms to generate plans and schedules as needed for execution. We study the effects of uncertainty on the decisions made by the scheduler within our simulation environment, based on the data from the San Francisco Airport (SFO).

This work intersects previous efforts in at least three areas. First, the problem to be solved requires the coordinated planning and scheduling for multiple agents [6]. Second, this work expands upon work on so-called 'rolling horizon' approaches to solve complex scheduling problems under uncertainty [7]. Finally, this work contributes to recent work at building models of uncertainty to improve the robustness of solutions to planning and scheduling problems (for example [8]).

The contributions of this work are as follows:

- We describe a general simulation tool that is easily extensible with different scenarios and airports, and can incorporate different scheduling policies.
- We develop realistic models for simulating and visualizing uncertainties in airport surfaces.
- We analyze the performance of systems combining deterministic planners and schedulers with continuous re-planning as a means to handle uncertainty.

In the next section we describe the model and algorithms used for planning and scheduling. In section three we discuss the simulation system and visualizer. In section four we describe experiments conducted with a deterministic scheduler using the simulator. We conclude with a discussion of future work.

[1] Heron Yang `heronyang@cmu.edu`
[2] Robert Morris `robert.a.morris@nasa.gov`
[3] Corina S. Păsăreanu `corina.pasareanu@west.cmu.edu`

II. Modeling Airport Surface Operations

A. Multi-agent Path Planning

Aircraft moving along the airport surface can be viewed as a set of agents and the planning and scheduling problem for them can be defined in terms of a *multi-agent path planning problem (MAPPP)* [9].

In general, the airport surface can be defined as a node-link model, where the nodes represent various important points on the surface (gates, runway) and links represent the connections (roads) between them.

The first step in this formulation therefore a transformation of a node-link model of an airport into a unit-distance graph $G = \langle V, E \rangle$ comprised of a set of vertices V and a set of edges E representing a unit distance between vertices, i.e., a distance related to discrete time-steps in the execution of a path. To handle holding actions, we assume that each vertex has an edge directed to itself; i.e., $\{\langle v_i, v_i \rangle\} \in E$ for all $v_i \in V$.

Given a set $A = \{a_1, \ldots, a_k\}$ of agents, and a time horizon $H = 0 \ldots T$, we define the *state* of an agent as a triple $\langle a_i, v_j, t_k \rangle$ of agent, vertex in G and time in H. An *itinerary* I is a set of tuples $\langle a_i, v_{s,i}, v_{g,i}, e_t, d_t \rangle$ consisting of an agent, a start and goal vertex, a time e_t representing the time the agent enters the planning system, and its scheduled arrival or departure time. Given I, a *feasible path* for an agent a_i is a sequence of vertices $P = \langle v_0, \ldots, v_n \rangle$, where $v_0 = v_{s,i}$, $v_n = v_{e,i}$, and there are edges $\langle v_k, v_{k+1} \rangle \in E$ for $k = 1 \ldots n-1$. The length of a path is the number of vertices in the path. A *route* between two vertices in G is a path with no duplicated vertices. A feasible path for an agent corresponds to a sequence of states $S_a = [\langle a, v_j, t_k \rangle]$. We write $P(S_a) = \langle v_0, \ldots, v_n \rangle$ to represent the path defined by the sequence of vertices in S_a.

An *action model* defines the allowable transitions between states, i.e. a set of actions $A_i : S \rightarrow S'$, where $A_i(S) = S'$. The action model used here consists of two actions: *hold* and *move*, defined by $hold(\langle a_i, v_j, t_k \rangle) = \langle a_i, v_j, t_{k+1} \rangle$ and $move(\langle a_i, v_j, t_k \rangle) = \langle a_i, v_l, t_{k+1} \rangle$, where $\langle v_j, v_l \rangle \in E$[1]. A feasible transition sequence is a sequence $TS = S_0, A_0, S_1, A_1, \ldots A_n, S_n$, where $P(S_0, \ldots S_n)$ is a feasible path, and $A_i(S_i) = S_{i+1}$ for all $i = 0, \ldots n-1$. S_0 (S_n) will be called the initial (terminal) state. Similarly, given an initial state S_s and goal state S_g, a *plan* is a sequence of actions $A_0, A_1, \ldots A_n$ such that $S_s, A_0, S_1, A_1, \ldots A_n, S_g$ is a feasible transition sequence from the start to goal states. The action model $M = \{hold, move\}$ is used by the planner to generate a plan sequence. Notice that if a path is a route, a corresponding transition sequence has the form $S_0, move, S_1, move, \ldots, move, S_n$. This special case will be called the *unimpeded* transition sequence, with associated unimpeded plan abbreviated by $move_{0,\ldots n-1}$. Thus, without fear of ambiguity, we use feasibility interchangeably to define paths, state sequences and transition sequences.

[1]Of course, if there are multiple vertices to which an agent can move, an argument should be added to $move$ to distinguish among the different destinations. But as we'll see later, for this MAPPP this argument won't be required.

In addition, because we are solving a continuous planning problem, we extend the action model to include actions not assigned by the planner; specifically we define two actions $exit, enter$ that define when an agent enters or exits the planning problem. The passing of time and the itinerary I determine when an agent enters the system: specifically given $\langle a_i, v_{s,i}, v_{g,i}, e_t, d_t \rangle \in I$, agent a_i enters the problem at time e_t. Conversely, given a transition sequence with terminal state $S_n = \langle a, v_j, t_k \rangle$, agent a leaves the problem at time t_{k+1}. Finally, we say an agent is *active* if it entered the problem but has not exited.

Given two transition sequences TS_{a_0}, TS_{a_1} a *conflict* is a pair of states $\langle a_0, v_j, t_k \rangle \in TS_{a_0}, \langle a_1, v_j, t_k \rangle \in TS_{a_1}$. There are two kinds of conflict: a move-move conflict is the result of two (or more) agents converging to a single vertex; a move-hold conflict is the result of one agents moving to a vertex in which another is holding. All conflicts are resolved by introducing delays (hold actions) on agent plans. Resolving either kind of conflict might result in the creation of one or more new move-hold or move-move conflicts. The total number of hold insertions needed to create a conflict-free feasible set of transition sequences is bounded by the number of agents, as well as the topology of G; specifically, the length of the longest route between start and goal states.

A MAPPP is a tuple $\langle G, I, A \rangle$, where G is a unit-distance graph, I an itinerary, and A a set of agents. A solution to a MAPPP is a collection of conflict-free feasible Transition Sequences $TS = \{TS_{a_1}, \ldots, TS_{a_n}\}$ for each $a_i \in A$. Solving an MAPPP consists of finding a path plan for each agent: a sequence of actions that defines a feasible transition sequence from a start state to a goal state that does not contain states that conflict with states of other agents.

B. Continuous Planning

The interest here is in representing the airport surface scheduling problem by casting it as a MAPPP, where the agents are aircraft arriving or departing. Drawing upon previous work on solving the same problem (e.g. [7]), and on characteristics of real airport operations, we first make the simplifying assumption that given a start and goal vertex v_s, v_g, e.g. corresponding to a gate and runway node for departing aircraft, there is a single predetermined route R_{v_s, v_g} assigned to any aircraft. This simplifies the problem while making it more realistic for airport surface planning, but the assumption can be removed without fundamentally changing the overall approach adopted here.

The second characteristic of airport surface planning is uncertainty. Specifically, a solution to a MAPPP may lead to conflicts when executed, due to unexpected delays. In general, there are two ways to handle uncertainty: by anticipating them at planning time by applying uncertainty models, or by responding to unanticipated changes to the operational environment by continuous re-planning. The interest of the research effort here is in both approaches to handling uncertainty.

We define an approach to continuous re-planning based on the idea of a *rolling horizon* (RH) [7]. An *RH variable* defines a moving window for solving a subproblem of the complete planning problem. Given a rolling horizon of N, the inputs to a RH planner is a set of current states $\{\langle a_i, v_j, t_k\rangle\}$ for each a_i in the set of active aircraft. The output of the planner is a set of feasible conflict-free transition sequences of the form $\langle a_i, v_j, t_k\rangle, A_k, \langle a_i, v_{j+1}, t_{k+1}\rangle A_{k+1}, \ldots, A_{k+N}$, $\langle a_i, v_{j+N}, t_{k+N}\rangle$. For uniformity, if an aircraft enters or exits the plan within the planning horizon, the states and actions before entering or after exiting consist are designated as NULL actions and NULL states.

In an environment with no delay uncertainty, the result of executing any RH-plan $P = A_{k,\ldots,k+N}$ from a set of initial states of the form $\{\langle a_i, v_j, t_k\rangle\}$ would be a set of states $S_{rh} = \{\langle a_i, v_{j+N}, t_{k+N}\rangle\}$ for each $a_i \in A$. Equivalently, the degree of delay uncertainty determines the extent to which the the set of states S resulting from executing P from the set of initial states differs from S_{rh}. To avoid conflicts from unexpected delays, we introduce a re-planning variable RP that designates the time between RH-planning, thus controlling the effects of uncertainty. Specifically, a smaller RP allows for more frequent sampling of the current active states, thus allowing for adjustment to unexpected delays in the execution of plans.

Algorithm 1 RH Planner

```
Inputs: Initial states $S = \{\langle a_i, v_j, t_K\rangle\}$; Itinerary $I$; Routes $\{R_i = v_0 \ldots v_M\}$ for all $a_i$ in the set of active aircraft; A Rolling Horizon $N$
Output: A set of conflict-free feasible Transition Sequences $TS$ and associated plans $A$ for each initial state in $S$
    1) Form the set $TS_u$ from generating the unimpeded transition sequence $S_j, move, S_{j+1} \ldots, move, S_{j+N}$ for each initial condition $S_j = \langle a_i, v_j, t_K\rangle \in S$, where $P(S_j, \ldots, S_{j+N}) = v_j, \ldots v_{j+N}$ taken from $R_i$, the preassigned route for $a_i$.
    2) Initialize the set $C$ of conflicts in $TS_u$
    3) Let $TS_u$ and $C$ provide the seed to local search to find a conflict-free feasible transition sequence $TS$
    4) return $TS$
```

The approach to scheduling is a variation of those found in AI Planning, for example the approach underlying Local Search on a Planning Graph (LPG) [10], and involves a two stage process. In the first stage, a "relaxed" version of the problem is solved quickly, where the relaxed version ignores conflicts between paths generated for different aircraft. The result of this stage is then used to guide the search for a conflict-free solution to the original problem during the second phase, using local search.

The RH-planner pseudo-code is found in Figure 1. There are three main computational components of the RH-planner: first, generating the initial unimpeded transition sequences, one for each initial state in the input of states ($O(N)$, step 1); second, detecting the set of conflicts in a set of transition sequences ($O(N^2)$, steps 2 and 3); finally, in the application of local search to repair a conflicted set of transition sequences.

Local search [11] is an family of algorithms that enables search through the space of solutions to find ones that are optimal with respect to an evaluation function. Local search requires a neighborhood function that, given a solution, returns a set of solutions that "are close to" the original. In one version of local search, the algorithm greedily selects the first neighbor that scores higher with respect to the evaluation function to continue the search. A termination condition is required to halt the search.

In our problem, local search is applied initially to a solution to the relaxed problem, in which all aircraft are assigned their unimpeded paths. This induces a set of conflicts. At each iteration, the set of neighbors consists of the result of removing a single conflict to the current schedule by inserting a delay. In the approach here, the conflicts are resolved in a temporal order, earliest conflict first. This heuristic corresponds to the intuition that assigning conflicts to those aircraft earliest in the sequence will not tend to induce new conflicts involving trailing aircraft.

III. Simulation System

The Airport Surface Simulation and Evaluation Tool-2 (ASSET2), was developed to evaluate our proposed planner under uncertainties. ASSET2, a revision of the original ASSET [12], is a generic tool for building models of airport surfaces and running fast-time simulations based on customized experiments.

An aircraft departure consists of four stages: pushback, spot release, taxi and takeoff, where the 'spot' is a location that provides the boundary between the ramp area and the taxiway. We define the link between a gate and a spot position as a push-back way, and the link between a spot position and the start node of the runway as a taxiway. These surface components are be presented in a link-node model in our simulation.

The simulator is composed of static objects containing immutable data of the simulation, dynamic objects representing run-time states, and the stateless object for computations, primarily the scheduler. Examples of static objects are *Surface*, *Scenario* and *Routing Table*; and dynamic object examples are *Airport* (contains a list of *Aircraft*) and *Clock*. A detailed description of the above objects is included in Table I.

Two kinds of data are required for executing a simulation: the airport surface data and the Itinerary data. The surface data is presented in a KML file format, which is parsed into several JSON files for comprising the input for the simulation. Surface data of an airport includes the name and geolocation of the gates, spot positions, taxiways, push-back ways, and the runways. For the experiments reported here, itineraries are randomly generated [2]. A time gap between two flights in an itinerary is based on a normal distribution, with the mean and the deviation value provided.

[2]The itinerary data can also be provided and be parsed into the format recognized by the simulation in the case of testing real-world scenarios.

	Description
Surface	Contains airport surface data of the gates, spot positions, taxiways, runways, etc.
Scenario	Contains a list of departure and arrival flight information include their designated gates and runways.
Routing Table	Used by *Routing Expert* object for giving the shortest route between two nodes.
Airport	Contains a list of active aircrafts, queues at gates, and the *Surface* object
Clock	Contains the simulated time of a day.
Scheduler	Generates itineraries per active aircraft every reschedule cycle.

TABLE I

DESCRIPTION OF OBJECTS IN *Simulation*

For each simulation run, a configuration file is given which describes the parameters. A *Routing Expert* calculates the shortest path between any two nodes using the Floyd Warshall algorithm and caches the routing table for future use.

Within a simulation, we keep a virtual time t_v representing the time of the day and define a constant time value T_S as the simulation time unit. The *Simulation* simulates the states at $t_v = n * T_S, n = 0, 1, ...$; therefore, T_S can be viewed as the resolution of the simulation. For active states, `tick()` function updates the states from t_v to $t_v + T_S$. The simulator calls its own `tick()` function in a loop until *Clock* object reports the simulated day has reached the end. Inside the `tick()` function, the simulator first checks if it is time to reschedule, and if so, it passes the delegate of the simulation to the scheduler and retrieves the sceanrios for each active aircraft. After executing the scenarios, it injects uncertainty delays, adds or deletes aircraft from the active list, calls the `tick()` function of the *Clock* object for all the active aircraft. Eventually, after the loop is finished, the simulator saves the output metrics into files or plots.

The scheduler is called once every reschedule cycle. It defines a function, `schedule(simulation)`, that reads the current simulation state and returns a list of itineraries for each active aircraft. The scheduler use the run-time states within the simulator to make scheduling decisions, or to make a copy of the simulation and predict the future states by calling the `tick()` function of the cloned simulation object.

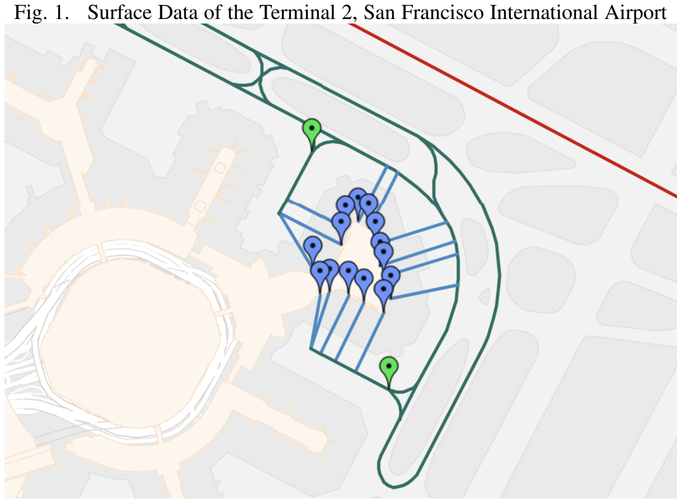
Finally, a web-based visualization tool is provided for viewing a simulation result or for debugging. The tool reads the output metrics of a single simulation, and visualized them onto using Google Map API. A portion of the visualization is shown in Fig. 1.

IV. EXPERIMENTS

A. Dataset

To evaluate our proposed model under a real-world environment and to avoid starting with an overly complex dataset, we picked a segment of San Francisco International Airport, Terminal 2, as our dataset. In this airport, there are 14 gates, 2 spot positions, 14 push-back ways, 25 taxiways, and 1 runway (see Fig. 1). Before running each simulation, the itinerary data is randomly generated where the time gap between two aircrafts is a Gaussian random number with a given mean value 90 seconds and a deviation 30 seconds.

Fig. 1. Surface Data of the Terminal 2, San Francisco International Airport



B. Environment Setup

1) Control Variables: We selected 30 seconds as *the time unit* of our simulation representing the real-world time period passed by per tick, and use 5 as the number of ticks that each uncertainty delay lasts.

Delays can occur anywhere along the airport surface, but in particular, common delays occur at gates, spot positions and runways. The simulator allows for delays to be inserted at these three positions. To better illustrate the effects of uncertainty on deterministic schedulers, in these experiments we restrict delays to occur at spot positions.

2) Experimental Variables: We define the *reschedule time cycle* as the time period between two schedule operations done by the scheduler, and the *uncertainty duration* as the length of the delay injected to an aircraft at a spot position. We launch two experiments for testing diffeent reschedule time cycles and different uncertainty durations while other parameters are fixed.

3) Output Metrics: For each simulation, we collect several output metrics including the number of delays injected by the uncertainty model and the scheduler, the queue size at the gate, the execute time of rescheduling, and the failure rate. The details of each metric is described below in Table II.

C. Reschedule Time Cycle Experiment

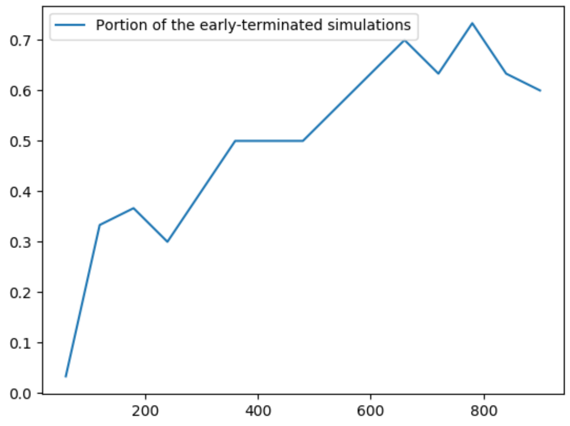
For our first experiment, we executed a batch run with reschedule time cycle ranging from 60 to 900 seconds with step 60 seconds. The uncertainty amount at the spot positions is set to 0.01. We sample 30 runs and use the average output metrics value of the 30 runs as the final result.

In Fig. 2, we plot the early termination rate for different reschedule times. The plot indicates that a large reschedule time leads to a high failure rate. This matches our expectation where a larger reschedule cycle decreases the chance for the scheduler to resolve future conflicts before they happen. For

	Description
Number of delay injected	Per tick, we count the number of aircrafts delayed by the uncertainty model and the scheduler separately.
Queue size at gate	The later flights are be added into a queue at the gate if the gate is occupied by a previous flight. Per tick, we log the total size of the queues at gates.
Reschedule execution time	We log the actual execution time for each scheduling operation.
Failure rate or early termination rate	If a conflict occurs during simulation, we abort the simulation and count it as a failure run. The failure rate is calculated by the number of failures divided by the total number of simulation runs.

TABLE II
DESCRIPTION OF THE OUTPUT METRICS

Fig. 2. Early-terminated Simulation Rate with Different Reschedule Cycle Time



example, if a conflict C is predicted to be happened in the third tick from now, C can be resolved by adding delays within the next two ticks. However, if the scheduler only reschedule after the third tick, the conflict is not able to be prevented, which leads to an early termination.

D. Uncertainty Experiment

In this second experiment, we executed two batch runs. In the first batch run, we use uncertainty amount ranging from 0 to 0.3 with step 0.03 and set the reschedule time cycle to 60 seconds indicating that scheduling happens in every two ticks. Each of the result we collect is an average over 30 samples. In Fig. 3, we plot the rate of early-terminated simulations, illustrating how a higher degree of uncertainty leads to a corresponding higher failure rate. When we assign uncertainties rate above 0.15, virtually all the simulations lead to early termination because of conflict.

In the second batch run of this experiment, we use an uncertainty amount ranging from 0 to 0.3 with step 0.05. Rescheduling happens in each tick and therefore all conflicts generated by the uncertainty can be solved immediately. For each data record, we average over 10 samples.

Fig. 3. Early-terminated Simulation Rate with Different Reschedule Cycle Time

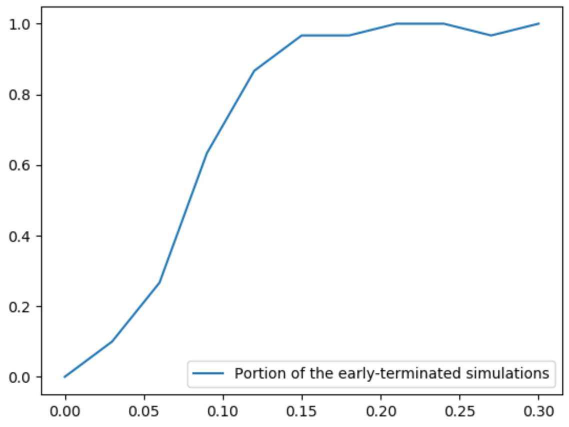
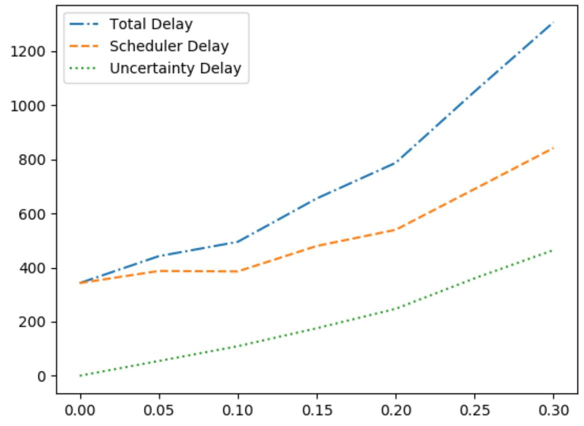


Fig. 4. Average Delay Added Per Tick with Different Uncertainties



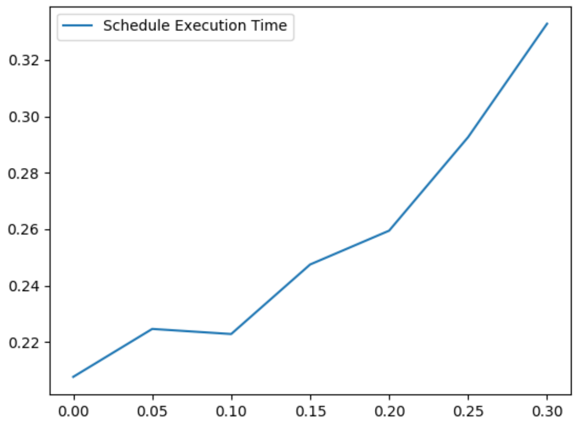
In Fig 4, we plot the accumulated amount of scheduler delays and uncertainty delays observed in each tick with different level of uncertainties. As expected, the amount of scheduler delays increases as more uncertainties are injected. By visualizing the simulation result, we reveal that the aircrafts are lining up at the spot position if the leading aircraft is delayed there.

In Fig 5, we plot the execution time of the scheduling process. As the uncertainty amount increases, more aircraft are expected to encounter a conflict, more conflicts needed to be resolved by the scheduler, thus increasing the scheduler execution time.

V. Summary and Future Work

This paper has introduced a framework that allows for the quantification of the effects of uncertainty on the robustness of solutions generated by deterministic planners and schedulers. The framework is composed of a fast-time

Fig. 5. Average Execution Time (Seconds) Per Reschedule with Different Uncertainties



simulator and visualization tool of the gate and taxi areas of San Francisco International Airport, and a deterministic scheduler based on a heuristic conflict-directed rolling horizon approach. Results documented in this paper are a first step in identifying the rescheduling time required to ensure robustness of schedules.

The goals of future work include higher fidelity, higher resolution simulation of aircraft movement, as well improvements to planning and scheduling to address the dual objectives of robustness and efficiency.

On the simulator side, we would like to expand the geometric model of SFO into a complete graphical model of the airport. We would also like to improve the fidelity of dynamic models, including the modeling of velocity and of pilot-induced delay. We're also interested in more realistic models of delays at gates, runways, and other bottleneck locations, including models based on real data. Also planned is a more thorough investigation of the effects of delays at all the criticial locations on the surface.

On the scheduler side, we will study the integration of path planning into the system for greater optimality. We will incorporate arrivals as well as departures into the problem space. Finally, we will integrate probabilistic models of delay into the decision-making of the planner and scheduler, thus allowing a more proactive response to delay during planning.

References

[1] W. Malik, G. Gupta, and Y. . Jung, "Spot release planner: Efficient solution for detailed airport surface traffic optimization," *Proceedings of the 12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, 2012.

[2] S. Rathinam, Z. Wood, B. Sridhar, and Y. Jung, "A generalized dynamic programming approach for a departure scheduling problem," *AIAA Guidance, Navigation, and Control Conference*, 2009.

[3] J. Ravizza, S. amd Atkin and E. Burke, "A more realistic approach for airport ground movement optimisation with stand holding," *Journal of Scheduling*, vol. 17, pp. 507–520, 2014.

[4] P. C. Roling and H. G. Visser, "Optimal airport surface traffic planning using mixed-integer linear programming," *International Journal of Aeronautical Engineering*, vol. 1, pp. 1–11, 2008.

[5] J. Reif, "Complexity of the movers problem and generalizations," *Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science*, p. 421427, 1979.

[6] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, 2016.

[7] G. L. Clare and A. G. Richards, "Optimization of taxiway routing and runway scheduling," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, vol. 12, 2011.

[8] G. Wagner and H. Choset, "Path planning for multiple agents under uncertainty," *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 2017.

[9] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," *CoRR*, vol. abs1204.5717, 2012.

[10] A. Gerevini and I. Serina, "LPG: A planner based on local search for planning graphs with action costs," in *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, 2002, pp. 13–22. [Online]. Available: http://www.aaai.org/Library/AIPS/2002/aips02-002.php

[11] E. Aarts and J. K. Lenstra, *Local Search in Combinatorial Optimization*. John Wiley and Sons, 1997.

[12] R. Morris, M. L. Chang, R. Archer, E. V. C. II, S. Thompson, J. L. Franke, R. C. Garrett, W. Malik, K. McGuire, and G. Hemann, "Self-driving towing vehicles: A preliminary report," in *Proceedings of the Workshop on AI for Transportation (WAIT)*, 2014, pp. 35–42.