Embedded Fast-Time Simulation to Support Airport Surface Operation Optimization

Victor H. L. Cheng¹, Anthony Y. Seo², and Paul P. Lin³
Optimal Synthesis Inc., Los Altos, CA, 94022

Wenbin Wei⁴
San Jose State University, San Jose, CA, 95192

and

David Davis⁵
VGO Associates LLC, Newbury, MA, 01951

A key operational capability specified in the Joint Program Development Office (JPDO) Next-Generation Air Transportation System (NextGen) Concept of Operations (ConOps) is Trajectory-Based Operations (TBO). TBO refers to operations based on 4-dimensional (4D) trajectories, each of which contains a specified route and a set of required times of arrival (RTAs) specified along the route. When this concept is applied to the airport surface domain, automation for the control tower can plan Trajectory-Based Surface Operations (TBSO) to optimize performance of the airport surface traffic. A recent TBSO research involves the development of a Surface Traffic Planner to optimize the 4D surface trajectories. The planner considers all the flights within a given time window, and systematically generates and evaluates different sets of possible 4D trajectories for the flights to arrive at an optimal solution. For the planner to operate in real time assuming a reasonable level of computational power, the planner needs a fast way to evaluate these sets of 4D trajectories, where each set comprising one 4D trajectory for each flight within the time window represents a “candidate solution.” To this end, an event-based fast-time simulation has been developed for evaluating these candidate solutions to produce the necessary performance metrics such as taxi time or taxi delay. Design of this simulation is based on a link/node graph-theoretic queuing model. The simulation also provides the capability to detect unacceptable events such as conflicts and incursions which render the candidate solution infeasible.

I. Introduction

A key operational capability specified in the Joint Program Development Office (JPDO) Next-Generation Air Transportation System (NextGen) Concept of Operations (ConOps) is Trajectory-Based Operations (TBO). TBO refers to operations based on 4-dimensional (4D) trajectories, each of which contains a specified route and a set of required times of arrival (RTAs) specified along the route. Operationally for TBO concepts, it is implicitly assumed that there is already agreement between the Air Navigation Service Provider (ANSP) and the flight deck (FD) on a 4D trajectory to be executed; otherwise the notion of 4D trajectories is meaningless. The 4D trajectory can be proposed by the FD and approved by the ANSP. However, to take full advantage of TBO, automation to help the ANSP plan the 4D trajectories can serve to optimize performance of the whole traffic. When this concept is applied to the airport surface domain, automation for the control tower can plan Trajectory-Based Surface Operations

---

¹ Vice President, Optimal Synthesis Inc., Los Altos, CA 94022, AIAA Associate Fellow.
² Research Engineer, Optimal Synthesis Inc., Los Altos, CA 94022.
³ Research Engineer, Optimal Synthesis Inc., Los Altos, CA 94022.
⁴ Associate Professor, Aviation, San Jose State University, San Jose, CA 95192.
⁵ President, VGO Associates LLC, Newbury, MA 01951.

American Institute of Aeronautics and Astronautics
(TBSO) to optimize performance of the airport surface traffic\textsuperscript{2–6}. Figure 1 illustrates a collaborative TBSO concept that counts on a tower automation to plan 4D-trajectory operations and send the planned data to the flight deck via digital datalink, where the flight deck is expected to have the necessary automation to execute the cleared trajectory.

Mathematically, optimization involves maximizing or minimizing some performance index, subject to one or more constraints. For TBSO, the performance index typically involves some function of operational delays, for which common metrics include pushback delay, taxi delay, and departure delay. Typical constraints involve safety concerns such as requirements for conflict-free and incursion-free trajectories, separation minima for takeoff and landing, etc.

As illustrated in Figure 2, the Tower Automation serves to assist the ANSP at the Controller Station to plan efficient and safe traffic on the surface. The planning process can count on the ability of the flight-deck automation to execute 4D surface trajectories with precise RTAs at selected points along the trajectories to assure conflict-free operations. The Tower Automation has various functions for managing the routes and clearances, which are sent to the Flight Deck via data link for the pilots to execute. The Surface Route Planner in the Tower Automation has the responsibility to generate the 4D taxi routes for the whole traffic, based on information on the flights, the airport layout model, and runway configurations. The Optimizer in the Surface Route Planner implements the optimization algorithm. Many optimization approaches need a function to evaluate the metrics used to determine the performance index and the constraints. This paper describes the development of an embedded fast-time simulation to achieve this function, shown as the Fast-Time Simulator in Figure 2.
In the Surface Route Planner, the Optimizer generates candidate solutions and sends them to the Fast-Time Simulator to evaluate. The objective of the Fast-Time Simulator is to evaluate the candidate solutions to determine the metrics used for computation of the performance index and the functions used to define the constraints. For the Surface Route Planner, a single candidate solution entails specification of complete taxi trajectories for all the flights included for planning within a time window, typically between half-an-hour to an hour. Some optimization techniques may consider one candidate solution at a time, while others such as Genetic Algorithms may consider a large population of candidate solutions at a time, perhaps in the order of hundreds. For this reason, the Fast-Time Simulator needs to be able to efficiently evaluate the candidate solutions. The embedded Fast-Time Simulator discussed here uses a link/node graph-theoretic queuing model to simulate the traffic over the airport. The simulation is event-based in order to speed up the computation relative to time-based options.

Section II discusses the relationship between the Optimizer and the Fast-Time Simulator within the Surface Operation Planner, and the requirements this relationship imposes on the Fast-Time Simulator. Section III describes the design of the Fast-Time Simulator, and Section IV discusses the interface requirements between the Optimizer and the Fast-Time Simulator. Some evaluation results are provided in Section V, followed by concluding remarks in Section VI.

II. Overview of Fast-Time Simulator’s Role in Surface Operation Planner

A typical optimization problem involves the maximization or minimization of a performance index, subject to a set of constraints, which may include both equality and inequality constraints. For the Surface Operation Planner, examples of performance index may involve maximizing throughput or minimizing delay, while constraints will include at least conflict-free surface operations which are also free of runway incursions from arrival and departure traffic. Typically, the Optimizer in Figure 2 will generate candidate solutions, and it will be the Fast-Time Simulator’s role to evaluate the performance index and the functions defining the constraints for these solutions. Based on performance index values and whether the constraints are satisfied, the Optimizer will be able to adjust the set of candidate solutions to arrive at an optimal solution. For the Surface Operation Planner, a candidate solution may involve a set of 4D taxi routes for all the flights expected within the planning time horizon, e.g., 30 min or an hour.

Regardless of the optimization approach adopted to implement the Optimizer, the Fast-Time Simulator is an integral part of the optimization process. Considering the conflict-free condition as a constraint, some of the candidate 4D trajectories may conflict with one another due to insufficient separation, e.g., a candidate solution involving two aircraft occupying an intersection of a taxiway/Runway at the same time, or one aircraft overtaking another along a taxiway. Conditions for runway incursion are more complicated as no other traffic is allowed on the runway ahead of the landing or takeoff flights. Other typical constraints include wake-safe separation between flights for landing and taking off, and safe taxi speeds for surface operations. In order to allow the Optimizer to test all its candidate solutions to support real-time operations, the Fast-Time Simulator has to be able to evaluate all these conditions within a sufficiently short time. For this reason, the Fast-Time Simulator is developed based on a graph-theoretic queuing model approach, where the airport surface layout is modeled as a link-node model. The links constitute the taxi paths, and the nodes are intersections or critical hold points.
For this development effort, Dallas/Fort Worth International Airport (DFW) has been selected as the airport. Figure 3 shows the layout of DFW, with its five terminals A, B, C, D and E and seven runways. Figure 4 shows the complete link/node model with over 1000 nodes and 1300 links. Figure 5 shows the zoomed-in view of the link/node model for Dallas/Fort Worth International Airport (DFW) overlaid on Google Earth satellite imagery, focusing on the gates around the terminal buildings.

With this queuing model approach, the Fast-Time Simulator can simply inform the Optimizer that a candidate solution is infeasible if any constraint is violated, e.g., a conflict between two flights exists among the flights included in the candidate solution. If a significant fraction of the candidate solutions turn out to be infeasible, valuable computation time will be wasted for their evaluation. It would be beneficial if there is an option to allow the Fast-Time Simulator to adjust an infeasible solution and turn it into a feasible one that can contribute towards the optimal solution. To achieve this, the Fast-Time Simulator is developed to detect all possible conflicts hidden in the set of 4D trajectories, and it also has the option to de-conflict the 4D trajectories by adjusting the RTAs of individual aircraft crossing the nodes of the respective routes, but otherwise keeping the routes unmodified as provided by the candidate solution.

The Fast-Time Simulator software is intended to efficiently evaluate candidate solutions. Certain optimization approaches such as Genetic Algorithm can consider multiple candidate solutions in parallel. Such approaches can benefit from the use of a collection of Fast-Time Simulators evaluating multiple candidate solutions in parallel, reducing time in the computation cycles. The application topology illustrated in Figure 6 allows multiple candidate solutions to be evaluated simultaneously, where candidate solutions and evaluation results are communicated over TCP/IP among distributed processors that implement the Optimizer and the Fast-Time Simulators. The Fast-Time Simulator Host coordinates the traffic of the candidate solutions.

Figure 3. DFW Layout
Figure 4. Link/Node Model of DFW

Figure 5. Example of Link/Node Model Overlaid on Google Earth Satellite Imagery
III. Design of Fast-Time Airport Surface Operation Simulator

A. Overview of the Fast-Time Simulator

1. Requirements for the Fast-Time Simulator
   Discussions from the preceding section suggest a few desired properties of the Fast-Time Simulator that have been adopted as requirements for its design.
   • The Fast-Time Simulator should be able to run sufficiently fast to support optimization models.
   • The Fast-Time Simulator should be able to detect and report conflicts that must be avoided under 4D-trajectory surface operations, such as violation of minimum taxi separation and runway incursions.
   • It is desirable for the Fast-Time Simulator to be able to adjust infeasible routes and turn them into feasible routes to the extent possible.
   • The Fast-Time Simulator should be able to create stochastic events to simulate airport surface traffic operations in the presence of uncertainties.
   A version of the Fast-Time Simulator has been developed with the capability to recognize some critical conflicts of airport surface operations. It can adjust the timing of events to resolve some of the conflicts. However, as its responsibility in resolving the conflicts is limited to adjusting the timing, it may not be able to recognize or resolve all possible conflicts along the given routes; e.g., if a trial candidate solution involves a trajectory that takes off from a closed runway, the Fast-Time Simulator would not be expected to re-route the flight, because doing so would expand its role to that of the Surface Operation Planner.

2. Assumptions
   The Fast-Time Simulator is implemented based on certain assumptions as follows:
   • Uncertainties: There are many sources of uncertainties. Examples of the sources include push back times for departing flights, taxi speeds, runway threshold arrival times, runway occupancy times, etc. It is assumed that, to ensure separation of the vehicles, the tolerance for each RTA will have to be padded with a sufficient amount of time buffer to account for uncertainties. (Simulation of the stochastic RTA is based on the assumed distribution of the variable.)
   • Pilot characteristics: Uniform behaviors of pilots are assumed.
   • Aircraft dimension: In the initial implementation of the Fast-Time Airport Surface Operation Simulator, a default length of 140 feet is assumed for the aircraft. This assumption can be alleviated with aircraft length dependent on the aircraft type.
3. Architecture of the Fast-Time Simulator

The Fast-Time Simulator is designed as an event-driven simulation system. In event-driven simulations, the system updates the variables when “something interesting” occurs, which is referred to as an “event.” In the context of airport surface operations, events include pushback, node-crossing, takeoff, touchdown, etc.

As shown in Figure 2, the Fast-Time Simulator collaborates with the Optimizer by evaluating the routes comprising the candidate solution. Figure 7 represents a high-level architecture of the Fast-Time Simulator, and shows how events and information flow among system components. Input from the Optimizer to the Fast-Time Simulator includes the candidate solution in the form of pre-optimized or trial routes. The Fast-Time Simulator decomposes the routes into events and evaluates them through the Traffic Controller function and the Surface Graph. It also checks the routes for feasibility relative to some of the typical constraints based on conflict-free requirements. In order to speed up the optimization process, the Fast-Time Simulator has been developed to correct some of the conflicts, so that an infeasible candidate solution may be made feasible and the final recommended solution sent back to the Optimizer upon successful evaluation. If the candidate solution includes constraint violations that the Fast-Time Solution cannot resolve, information on the infeasible route is returned to the Optimizer.

![Figure 7. High-Level Architecture Showing Information Flow of Fast-Time Simulator](image)

Each flight in the Flight Manager receives one pre-optimized route from the optimization model. Events are created in increasing order of time, and sent to the Event Queue managed by the Event Manager. The Event Manager pops one event at a time, and sends it to the Traffic Controller. The Traffic Controller pre-analyzes and processes the event if the event is feasible. The Traffic Controller determines whether the simulation can continue or not based on the type of conflicts detected.

The Fast-Time Simulator evaluates one complete set of routes at a time. Typically, the Optimizer may send a successive candidate set of routes to evaluate based on the evaluation result of a previous set until the optimization process converges.

B. Software Design of Fast-Time Simulator

The Fast-Time Simulator is implemented using C++ to take advantage of the programming language’s superior performance. The high-level architecture of the Fast-Time Simulator complies with the object-oriented programming (OOP), but some foundational data structures and algorithms are implemented using rudimentary C programming codes to further improve its performance.
Figure 8 shows the sequence of the processes for evaluating sets of routes, where the sequence contains an initialization, an inner loop and an outer loop of processes. During initialization, it reads in files that contain information of an airport link/node model. An object of SurfaceGraph class is then instantiated based on information of the airport link/node model. It also reads in types of aircraft, and instantiates EventManager, TrafficController and CommAgent. EventTimeRandomizer is instantiated to randomize event timing if the simulation is part of a stochastic simulation run. The initialization process ends with a set of flights to be received from the Optimizer.

![Figure 8. Sequence of Processes](image)

For each outer-loop iteration, the outer loop processes a set of routes for the given set of flights: it receives a set of routes, one route for each flight. A series of events is created, and is put into an event queue in the order of event times. The TrafficController and the EventManager collaborate to process the events, and the results of the evaluation are reported back to the Optimizer. If there is another set of routes to be evaluated, the SurfaceGraph is re-initialized; otherwise the simulation terminates.

The inner loop deals with an individual event. The EventManager gets the next event and sends it to the TrafficController, which pre-analyzes the event, and sends the results to the TrafficController. If the event is determined feasible, the TrafficController processes the event and updated the SurfaceGraph. If the event is determined to be infeasible for continuing the evaluation of the given set of routes, the TrafficController reports it to the optimization model. The following subsections describe the various aspects considered in the design of the Fast-Time Simulator.

1. Optimizer

The Optimizer feeds the Fast-Time Airport Surface Operation Simulator with a set of flights with 4D routes that are not necessarily feasible. These 4D routes may have routes that contain conflicts with one another. Along with the set of routes, the Optimizer may send over additional parameters related to constraints, objectives, stochastic behaviors, etc.
2. Layout

Airport surface modeling determines the level of accuracy of the simulation result. In the Fast-Time Simulator, a link-node model is used to model the flow of surface traffic at the airport being studied. The Layout class is defined for instantiation of such link/node airport models. Layout contains a set of links, nodes, and runways, which constitute a graphical abstraction of the airport surface system. The target airport model for this study is Dallas/Ft. Worth International Airport (DFW), which has 1041 nodes and 1340 links to model the airport surface pavement system. Figure 9 contains a graphical representation of the DFW layout overlaid with the link-node model.

Figure 9. Dallas/Ft. Worth International Airport (DFW) Link-Node Model

Figure 10 shows the class relationships among Layout, Link, Node, and Runway in a UML class diagram. As shown in the UML class diagram, the Runway class is introduced as separate from the Link and Node classes to represent runways as special resources in airport operations.
Node Properties

Nodes are defined at specific locations on the airport surface. They include intersections of runways and/or taxiways, gates, ramp spots, hold points, parking spots, deicing locations, etc. Actions of flights at a node depend upon its location in the runway-taxiway system and upon the configuration of the airport. This information is essential for determining the types of airport surface operations that the Fast-Time Simulator needs to process. To support the simulation processing, six node attributes are defined: Domain, RefName, Type, Dependency, Jurisdiction, and Comments. Their significance is as follows:

- **Domain** specifies the usage of the node. Possible values are: Runway, Taxiway, Ramp, Gate, Parking, and Deice. The attributes RefName, Type, and Dependency are dependent on the Domain attribute.

- **RefName** defines the specific object to which the Domain attribute refers. For the Runway domain, it is the runway name. For an example, “13R” means that the node is on runway 13R. If the node is a crossing point of two intersecting runways, it would have two values such as “13R/22L”; note the forward slash separator for the two values. If the node is the intersection of more than two taxiways, the names of all of the intersecting taxiways will all be included as values for this attribute.

- **Type** represents the function, or purpose, of each node. Possible values include:
  - HighSpeed: High-speed runway exit, i.e., taxiway centerline is 30°–45° from the runway centerline.
  - Exit: Low-speed runway exit, i.e., a taxiway that is approximately at right angle to the runway.
  - Entry: Runway entry point for taking off.
  - RwyCrossing: Crossing node of two intersecting runways.
  - TxyCrossing: Node that serves as a runway crossing point for a taxiway.
  - Hold: Point along a taxiway where an aircraft may hold before crossing an active runway or entering an intersection (i.e., on a hold line).
- **RampSpot**: A ramp node—normally at the boundary of the ramp (also referred to in the US as the non-movement area)—where a departure flight may hold while waiting for clearance from the tower controller to enter the movement area, or where an arrival flight enters the ramp area.
- **Normal**: Applies to Runway, Taxiway, or Ramp domains, for a node that does not have any special purpose covered by the other types.
- For other domains such as gates, parking, and deicing, it represents the type of facility yet to be defined.

- **Dependency** lists other objects that may be used simultaneously with the node. In some airport models, there are places where nodes are so close to each other that they are operationally dependent. In other words, if an aircraft is on top of one node, there would not be enough room for another aircraft to pass by adjacent nodes, e.g., in the DFW model, when an aircraft occupies Txy_Y_004, the adjacent nodes Txy_G_Y and Txy_Y_002 are considered to be occupied as well.
- **Jurisdiction** specifies the controller position that has jurisdiction over the node.
- **Comments** contains any special information the user may want to include.

If a particular attribute does not apply, the value should be set to None. The six node attributes are summarized in Table 1. In the current Fast-Time Simulator implementation, not all of the defined attributes are used.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Runway</td>
<td>The area to which the node belongs: Runway and Taxiway domains are in the movement area.</td>
</tr>
<tr>
<td></td>
<td>Taxiway</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ramp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Park</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deice</td>
<td></td>
</tr>
<tr>
<td>RefName</td>
<td>For Runway domain</td>
<td>Runway name</td>
</tr>
<tr>
<td></td>
<td>For Taxiway domain</td>
<td>Taxiway name</td>
</tr>
<tr>
<td></td>
<td>For Ramp domain</td>
<td>Terminal name</td>
</tr>
<tr>
<td></td>
<td>For Gate domain</td>
<td>Gate name</td>
</tr>
<tr>
<td></td>
<td>For Park domain</td>
<td>Parking lot name</td>
</tr>
<tr>
<td></td>
<td>For Deice domain</td>
<td>Deicing station name</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>For Runway domain</td>
<td>HighSpeed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RwyCrossing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TxyCrossing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Normal</td>
</tr>
<tr>
<td></td>
<td>For Taxiway domain</td>
<td>Hold</td>
</tr>
<tr>
<td></td>
<td>For Ramp domain</td>
<td>RampSpot</td>
</tr>
<tr>
<td></td>
<td>For Gate domain</td>
<td>Type of gate</td>
</tr>
<tr>
<td></td>
<td>For Park domain</td>
<td>Type of parking</td>
</tr>
<tr>
<td></td>
<td>For Deice domain</td>
<td>Type of facility</td>
</tr>
<tr>
<td>Dependency</td>
<td>List of dependent nodes</td>
<td>Indicates nodes of which the use may be affected by aircraft at other nodes.</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Jurisdiction</td>
<td>Controller Id</td>
<td>Which ATC controller has the jurisdiction over it</td>
</tr>
<tr>
<td>Comment</td>
<td>Any comments on the node can be put this attribute.</td>
<td></td>
</tr>
</tbody>
</table>
Link Properties

Links are segments between nodes on the airport surface where surface operations are possible. Each link definition is associated with a begin-node (sometimes referred to as nodeOne) and an end-node (or nodeTwo). As with nodes, different links have different uses. A set of standardized attributes has been developed for links, and these attributes need to be defined in the airport model to support the Fast-Time Simulator’s processes. The link attributes are Domain, RefName, Width, Direction, Cost, and Comments. Their meanings are as follows:

- **Domain** specifies the usage of the link. Possible values are: Runway, Taxiway, Ramp, Gate, Parking, and Deice. If a link is directly connected a gate, the domain attribute must have the value Gate. The set of valid attribute values for RefName is a function of the Domain attribute.

- **RefName** defines the specific object to which the domain attribute refers. If the domain is Taxiway, for example, the value of RefName would be a taxiway name.

- **Width** contains the width of the pavement in feet. Some runways and taxiways are not constructed to handle larger aircraft and often the width is related to the load criterion.

- **Direction** is an integer flag for specifying the direction along the link permitted for use by the aircraft. The values are:
  - 0: The link is closed to traffic.
  - 1: The direction of travel is from nodeOne to nodeTwo.
  - 2: The direction of travel is from nodeTwo to nodeOne.
  - 3: Traffic can travel in either direction.

- **Cost** is composed of a pair of numbers separated by a forward slash, where the first number is the cost for traversing from nodeOne to nodeTwo (i.e., Direction = 1), and the other is the cost for traversing from nodeTwo to nodeOne (i.e., Direction = 2).

- **Comments** contains any special information the user may want to include.

Table 2 summarizes the attributes. However, not all the defined attributes are used in the current Fast-Time Simulator implementation.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Runway</td>
<td>Same as nodes. If a link crosses two different domains, see Table 3 for which domain name to use.</td>
</tr>
<tr>
<td></td>
<td>Taxiway</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ramp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Park</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deice</td>
<td></td>
</tr>
<tr>
<td>RefName</td>
<td>Runway</td>
<td>Same as nodes.</td>
</tr>
<tr>
<td></td>
<td>Taxiway</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ramp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Park</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deice</td>
<td></td>
</tr>
<tr>
<td>Width</td>
<td>Width of the link in feet</td>
<td>Distance between the two end nodes.</td>
</tr>
<tr>
<td>Direction</td>
<td>Blocked 0</td>
<td>Under specific airport configurations, links that geographically connected are blocked, one way, or bi-directional.</td>
</tr>
<tr>
<td></td>
<td>One way from nodeOne to nodeTwo</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>One way from nodeTwo to nodeOne</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Bi-directional</td>
<td>3</td>
</tr>
<tr>
<td>Cost</td>
<td>Cost from node one to node two, and cost from node two to node one</td>
<td>To represent operational preference, researchers may adjust link costs.</td>
</tr>
<tr>
<td>Comment</td>
<td>Any comments on the link can be put this attribute.</td>
<td></td>
</tr>
</tbody>
</table>
3. **CommAgent**

This component exclusively deals with communication with the Optimizer. It has been designed to isolate the main component from auxiliary components. It performs functions that send and receive any messages from and to the Optimizer. It also has the responsibility to analyze the received information for integrity of the received messages. This function has been implemented to support different modes of integration with the Optimizer: TCP/IP, and direct integration with the Optimizer process.

4. **FlightRepository, Flight, and Route**

During the initialization phase, a set of flights is sent to the Fast-Time Simulator from the Optimizer. The Fast-Time Simulator simulates the given set of flights in a simulation run. It has the capability to simulate multiple candidate solutions corresponding to different sets of routes for the same set of flights for different simulation runs, in which case the given set of flights does not need to be updated in between runs. Figure 11 illustrates in a UML class diagram showing the relationships among the FlightRepository, Flight, and Route classes, which are the classes used by EventManager to control a simulation.

![Figure 11. Relationships Among FlightRepository, Flight, and Route](image)

The FlightRepository is a container of points of flight, which is implemented as a Standard Template Library (STL) vector.

**Flight** objects are models of flights in airport surface operation, each representing the functions of a pilot and the aircraft. Airline pilots maneuver the aircraft to taxi along the given route. In real-world operations, pilots are autonomous agents; however, pilots and aircraft are modeled together in the Flight class in the Fast-Time Simulator.

A Flight can contain zero Route or multiple instances of Route. A Route consists of a set of RouteSegment objects, each of which comprises an inbound link identifier, crossing node identifier, and crossing time. The crossing time corresponds to the exit time from the inbound link and the entry times of the outbound link, under the assumption that no flight is allowed to stop and block a node. The inbound link of the first segment is NULL if a route starts at a terminating node such as a gate. Figure 12 depicts a Route and its segments. Figure 13 illustrates the C++ implementation of the Route class and the RouteSegment structure.
As mentioned in Section III.A.2, there is a high level of uncertainties in airport surface operations. To account for the uncertainties, events need to have some stochastic features. The \textbf{EventTimeRandomizer} class implements such event-related stochastic features, with the capability to generate random numbers according to the defined distributions of variables such as duration of push back time, speed, etc.

6. \textbf{EventManager}

The EventManager creates events derived from routes, and arranges for the TrafficController to analyze and process the events. It determines the types of successive event and creates them. EventQueue is a simple priority queue that contains future events and automatically sorts the events in the order of event times. Each event is composed of the event time, the flight associated with the event, and where the event is happening. The events reside in the EventQueue and are sorted by event time with the help of an automatic sorting function. In the initial implementation of the Fast-Time Simulator, the following events are defined: Pushback, TaxiCrossing, Takeoff, Landing, Rollout, RunwayCrossing, ParkAtGate, ReleaseFunway, and ReleaseGate. These events are illustrated in Figure 14.

As the Fast-Time Simulator evolves, some events may need to be decomposed into more-detailed events. For example, the \textbf{Landing} event may be broken into events representing touchdown, deceleration, and runway turnoff.
TrafficController has two main responsibilities: pre-analyze events, and process feasible events. The TrafficController analyzes an event to determine if the event is feasible or not in advance of processing the events and updating the Surface Graph. In this context, a feasible event is one where the event is conflict-free without any adjustments or with some acceptable level of adjustment such as adjusting node-crossing time.

If the TrafficController determines an event to be feasible, it will process the event. At this point, the TrafficController can derive the consequences of the event and update the Surface Graph accordingly.

In the pre-analysis phase, the TrafficController is responsible for recognizing the types of conflicts shown in Figure 15: runway incursion, head-on, overtake, wrong direction, occupied, and taxiing on runway.

Link transit and node transit are the fundamental operations modeled for airport surface traffic. These two operations are the basic building blocks of events. In the initial Fast-Time Simulator implementation, a node transit implicitly represents two link transits composed of exiting an inbound link and entering an outbound link. The variable crossingTime in RouteSegment represents the required time to cross the node. The required time is related to both the inbound-link exit time and the outbound-link entry time. It also concludes the delay for taxiing through the inbound link.

\[
\text{Required time (crossingTime)} = \frac{(\text{inbound-link exit time} + \text{outbound-link entry time})}{2}
\]

\[
\text{Delay} = (\text{current-node inbound-link exit time} - \text{previous-node outbound-link entry time}) - \text{unimpeded inbound-link traversal time}
\]

Figure 16 illustrates the timing relationships for node transits and implicit link transits for two sample flights.
Figure 15. Defined Types of Conflict

8. **Surface Graph**

The Surface Graph is a set of queues making up the queuing model to represent the state of the runways, nodes, and links, implemented as RunwayQueue, Intersection, and LinkQueue, respectively. Figure 17 illustrates the implementation of these classes.

American Institute of Aeronautics and Astronautics
A RunwayQueue can hold only one arriving or departing flight at any given time, modeled by the binary states of “free to use” and “occupied by only one flight.” When the queue is “free to use,” then other flights can cross the runway and these crossings may be performed in parallel. An Intersection holds only one flight at any given time. A LinkQueue can hold multiple flights based on the length of the link. A LinkQueue is made up of two unidirectional queues, each modeling a one-way link. These two component queues are used together to model the link transits of every link in the airport surface system.

Figure 18 and Table 4 describe the concept. A blocked link has zero capacity for both component queues. A unidirectional link has nonzero capacity for the direction and zero capacity for the other. A bidirectional link allows both queues to have nonzero capacities; however, these two component queues are mutually exclusive in the sense that whenever one is occupied, the other will have zero capacity.

<table>
<thead>
<tr>
<th>Link Type</th>
<th>Capacity of queueOne</th>
<th>Capacity of queueTwo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocked</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>One way from nodeOne to nodeTwo</td>
<td>≥ 0</td>
<td>0</td>
</tr>
<tr>
<td>One way from nodeTwo to nodeOne</td>
<td>0</td>
<td>≥ 0</td>
</tr>
<tr>
<td>Bidirectional</td>
<td>≥ 0</td>
<td>≥ 0</td>
</tr>
</tbody>
</table>
IV. Interface Requirements with Optimizer

Figure 19 illustrates the modeling of the interface between the Optimization Model and the Fast-Time Surface Traffic Simulator, as well as the relationship with the input files. Details of the implementation are discussed in the following subsections.

A. Nodes

Node data are passed with the following format to describe the information and attributes of the nodes:

- Id, Node_Name, Domain, Latitude/Longitude, X/Y(feet), RefName, Type, Dependency, Jurisdiction

The following is an example of a list of nodes:

1. Txy_M6_002 Taxiway 32.8870/97.0273 4471.9114/-3023.0413 M6 Normal Txy_M6_001 Unknown
2. Txy_R1_001 Taxiway 32.9080/97.0177 7403.3364/4655.2346 R1 Hold None Unknown
3. Txy_P_002 Taxiway 32.9125/97.0195 6860.54172/6264.1782 P Hold None Unknown
...

B. Links

Link data are passed with the following format to describe the information and attributes of the links:

- Id, Name, Domain, RefName, Direction, Length (feet)

The following is an example of a list of links:

1. Rwy_01_001-Rwy_01_002 Runway 13R 1 2825.554016903462
2. Rwy_01_002-Txy_A1_001 Taxiway A1 2 681.1466271134084
3. Rwy_01_002-Rwy_01_003 Runway 13R 1 971.3022638407037
4. Rwy_01_003-Rwy_01_004 Runway 13R 1 1806.4626990511688
...

C. Runways

Runway data are specified with the following format to pass the necessary link and node information:

```
// Runway_Name length width
link idx => node idx
link idx => node idx
...
```
The following is an example involving of two runways:

// 13R 9301.0 150.0 7
null => Rwy_01_001
Rwy_01_001-Rwy_01_002 => Rwy_01_002
Rwy_01_002-Rwy_01_003 => Rwy_01_003
Rwy_01_003-Rwy_01_004 => Rwy_01_004
Rwy_01_004-Rwy_01_005 => Rwy_01_005
Rwy_01_005-Rwy_01_006 => Rwy_01_006
Rwy_01_006-Rwy_01_007 => Rwy_01_007

// 18R 13400.0 150.0 17
null => Rwy_02_001
Rwy_02_001-Rwy_02_002 => Rwy_02_002
Rwy_02_002-Rwy_02_003 => Rwy_02_003

D. Aircraft Characteristics

Each aircraft type can have distinct dimensions and performance characteristics in the implementation of a realistic simulation. Although the Fast-Time Simulator has been designed to accept these data, the initial implementation does not yet use all these data. The format of the aircraft type data is as follows:

- FAA/PAS ID
- Formal type and how described verbally by controller
- Threshold speed
- Deceleration at threshold (fps/s)
- Takeoff acceleration (fps/s)
- Liftoff speed
- Default taxi speed
- Min taxi speed
- Max taxi speed
- AC length
- Wheelbase length
- CG to wheels length
- Tau (time const)
- Min landing distance

The following are examples of aircraft characteristic data:

- A306,"Airbus A300 B4 "A300"",134,0,6,144,15,5,40,177,5,5,49.95,1,5900
- B731,"Boeing 737-100 "737"",126,0,6,111.3692308,15,5,40,94,47,42.3,1,4900
- B732,"Boeing 737-200 "737"",130,0,6,114,15,5,40,100.2,50.1,45.09,1,4900

E. Route Set

A 4D route that originates from any point on the surface to either a gate (for an arrival flight) or a runway entry (for a departure flight) is composed of nodes, links, and times at the nodes along the route. The 4D route is modeled with the RouteSegment class in the Fast-Time Simulator. The format of a route is given by:

callsign and a list of route segments (link idx/node idx/time)

The following is an example of a route set:

- EGF387 0/434/27 1005/301/34 1003/300/39 ...1185/59/326 1186/958/339
- AAL1726 0/618/27 637/357/42 634/358/50 632/359/57 ... 299/934/362 1185/59/375 1186/958/388

F. Route Evaluation Results

The result of evaluation given a candidate set of routes is sent to the Optimizer when the Fast-Time Simulator completes the simulation for all given routes without finding any infeasible routes, or when the Fast-Time Simulator detects any infeasible routes that it cannot handle by itself such as head-on conflicts. The format of the evaluation result data is as follows:

callsign
Link_idx – node_idx, required_time, actual_time, conflict_type

The following is an example of the returned data for one flight:

- EGF387 0-131 requiredTime: 20 actualTime: 14 conflictType: 1
- 1116-144 requiredTime: 40 actualTime: 34 conflictType: 1
- 287-795 requiredTime: 130 actualTime: 132 conflictType: 1
- 366-818 requiredTime: 150 actualTime: 151 conflictType: 1
V. Evaluation of Fast-Time Simulator

The initial implementation of the Fast-Time Simulator has been subjected to two types of testing: (i) computational speed; and (ii) ability to deliver the functionality expected for integration with the Optimizer.

A. Computational Speed Test

The Fast-Time Simulator was subjected to a sequence of tests using a range of traffic demands for the DFW airport model. The scenario was only for the four runways on the east side of DFW, with the number of flights ranging from 20 to 220 scheduled for operation in an hour. The computation times are tabulated in Table 5 and plotted in Figure 20. It can be seen that the computation time tends to increase more than linearly since the number of conflicts compounds as the traffic demand increases. A run with 240 flights basically broke down as the traffic was causing gridlocks on the surface. It can be seen that, for reasonably heavy traffic of 140 flights/hr, the simulation would take less than 3 sec. This computational testing was performed on a consumer-level workstation with the following specifications:

- Processor: AMD Phenom 9550 Quad-Core Processor 2.20 GHz
- Memory (RAM): 6 GB
- Operating system: Windows Vista Home Premium 64-bit

To speed up the overall computation when dealing with multiple candidate solution sets simultaneously, the parallel architecture in Figure 6 can be employed using faster computer servers.

<table>
<thead>
<tr>
<th>Number Of Flights</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>120</th>
<th>140</th>
<th>160</th>
<th>180</th>
<th>200</th>
<th>220</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation Time (sec)</td>
<td>0.250</td>
<td>0.463</td>
<td>0.733</td>
<td>1.075</td>
<td>1.419</td>
<td>1.856</td>
<td>2.590</td>
<td>3.146</td>
<td>4.555</td>
<td>6.880</td>
<td>8.502</td>
</tr>
</tbody>
</table>

Figure 20. Computation Time of Fast-Time Simulator for Range of Traffic Demand

B. Integrated Testing with Optimizer

The Fast-Time Simulator has been integrated with an Optimizer to verify the interface between the two processes. In this test, the Optimizer implements an optimization algorithm based on Genetic Algorithms (GA) technology. The implementation is based on Steady-State GA. The algorithm begins with an initial population of 15 candidate solutions, where each candidate solution contains a complete set of 4D routes for all the flights being considered. During the optimization process, the Optimizer sends the 15 candidate solutions to the Fast-Time Simulator, one at a time, for evaluation, and the Fast-Time Simulator would return the fitness values to the Optimizer. The fitness value in this case is the total delay from all the flights. After the initial population has all been evaluated, the Optimizer would begin the genetic operations of creating a new offspring in each iteration from the
existing population, and removing one old candidate solution to keep the population size constant. The offspring would be sent to the Fast-Time Simulator for evaluation.

Figure 21 shows the evaluation results for 100 flights within a 1-hour time window. A total of over 50 candidate solutions (each containing 100 4D routes for the 100 flights) have been evaluated. It is unclear which 15 candidate solutions are kept in the final population, but the algorithm would always keep the one with the best fitness value (i.e., smallest total delay) and tends to keep those with good fitness values (in a randomized sense) to support genetic reproduction. The one with the lowest fitness value is highlighted in Figure 21, producing an average delay of less than 3.5 min.

![Figure 21. Fast-Time Simulator Evaluation Candidate Solutions Sent by Genetic-Algorithm Optimizer](image)

**VI. Concluding Remarks**

This paper describes the development of a fast-time surface operation simulation to support implementation of the surface traffic optimization solution to realize Trajectory-Based Surface Operations (TBSO). Specifically, the fast-time simulation allows the evaluation of candidate routes supplied by the optimizer for a specified set of flights.

The fast-time simulation uses a graph-theoretic link-node model of the airport layout to implement a queuing model for efficient simulation of the surface traffic. It has the capability to perform limited detection of conflicts inherent in the candidate routes, and the capability to resolve certain types of these conflicts when adjusting the timing of the routes is adequate. Since an optimization approach based on Genetic Algorithms (GA) may create many infeasible solution sets within the population of solution sets it generates through genetic evolution, these capabilities allow the fast-time simulation to turn an otherwise infeasible solution set into a feasible one to be inserted back into the population, while providing an evaluation of its fitness function (or performance index).

An initial implementation of this fast-time simulation has been developed for the South-Flow configuration of the Dallas/Ft. Worth International Airport (DFW). This model has been integrated with a GA-based optimization algorithm for planning surface 4D routes as part of the effort to realize TBSO. Initial tests have been performed to establish the computational speed of the fast-time simulator and verify its interface with the optimization process. The resulting Surface Operation Planner formed by integrating the optimization algorithm and the fast-time simulation will be subjected to evaluation in another simulation environment which offers a higher level of fidelity than the fast-time simulation.
Acknowledgments

This development effort was performed under support from NASA Contract Number NNX07AU49A. The authors thank Mr. Ty Hoang, Ms. Sandra Lozito, and Dr. Yoon Jung from NASA Ames Research Center for their support.

References