A Closed-Form Solution to Multi-Point Scheduling Problems

Larry Meyn[*]
NASA Ames Research Center, Moffett Field, CA, 94035-0001

A closed-form algorithm is proposed as a tool by which aircraft can be scheduled through an arbitrary number of scheduling points within allowable time constraints. The algorithm supports the specification of one or more time constraints at each point, along with minimum and maximum allowable travel-time constraints between points. A constraint algebra that simplifies the expression of time constraint calculations is used to formulate the algorithm. An example is then presented of an automated scheduler that uses constraint algebra and the new algorithm. The scheduler implements prospective operations for San Francisco International airport that is able to accommodate formation landings of aircraft pairs on closely spaced parallel runways for specified speed and timing constraints. A simulation of the scheduler shows that allowing flights to land in formation on two closely spaced parallel runways can reduce flight delays by 72% to 98% by enabling use of a second runway under adverse weather conditions when compared with the current single runway operation.

I. Introduction

THE Federal Aviation Administration has a goal to improve the utilization of available airport capacity without decreasing safety or increasing controller workload.[1] One way to address this goal is by extending and improving the assignment of scheduled times of arrival for flights to a series of specific points along their routes. These points can include fixes, arcs, runway thresholds and boundary crossings. More scheduling points than are used in the current system will be needed. The scheduled times of arrival at each point will need to satisfy multiple scheduling constraints and meet physically realizable minimum and maximum travel-time constraints between points.

Several approaches for determining viable schedules have been investigated. One approach is to calculate them directly from the constraints, by use of a closed-form method. While closed-form solutions have been used for two-point schedules,[2-5] they have not been used to determine schedules with more than two points. The multi-center scheduler presented by Landry[6] breaks up the scheduling problem into loosely coupled sub-problems to reduce its complexity. A scheduler developed for airborne merging and spacing solves the problem by iteratively trying possible scheduling times.[7] Others have utilized optimization routines[8,9] and commercial constraint solvers.[10] One possible reason why closed-form algorithms are not used for multi-point scheduling is the expectation that they might be too complex to implement. Such a

[*]Aerospace Engineer, Systems Modeling and Optimization Branch, MS 210-15, and AIAA Associate Fellow.

possibility could easily be inferred from the apparent complexity of the closed-form, two-point scheduling algorithms, such as those developed by Neuman[2,3] and Wong.[4]

One way to reduce the complexity of scheduling calculations is to recast the equations by use of a constraint algebra, as was done previously for Neuman's two-point scheduler.[5] Because the constraint algebra simplifies the expression constraint calculations in a manner similar to matrix algebra, the perceived complexity of Neuman's algorithm is significantly reduced. The resulting implementation of the software for scheduling calculations was more concise and easier to follow than Neuman's original code. In this paper, the same constraint algebra is used to develop a concise, closed-form scheduling algorithm that is applicable to scheduling problems with any number of points.

The next section of this paper presents the development of the closed-form, multi-point scheduling algorithm in three subsections. First, an example problem is posed and the steps required to solve the problem are described. An overview of the constraint algebra is then presented, followed by a detailed description of the new, closed-form, multi-point scheduling algorithm. Section III describes a simulation that uses the new algorithm for scheduling very closely spaced parallel approach operations at San Francisco International Airport. The results from the simulation are then presented in section IV, followed by some concluding remarks in section V.

II. Multi-Point Scheduling

A. Example Multi-Point Scheduling Problem

Multi-point scheduling problems can be solved by propagating scheduling constraints to all points along a route in both the downstream and upstream directions. The steps required to propagate the constraints are described for a notional scheduling problem. Figure 1 depicts a "node-link" diagram for a scheduling problem with four scheduling points, designated **A**, **B**, **C** and **D**. The links represent paths between the points. The links are directional, as shown by the arrowheads, and the flights passing through **B**, **C** and **D** may have origin and destination points that are not shown. A master schedule is maintained, to reserve blocks of time at each point for previously scheduled flights. These blocked schedule times are constraints on the schedule times available at each point. The problem is to generate viable scheduling times for a flight arriving at **A** at time 0, and then continuing on through points **B**, **C** and **D**. In addition to the blocked schedule times at each point, the minimum and maximum travel-times between sequential points further constrain the solution.

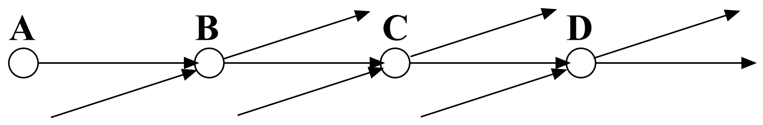
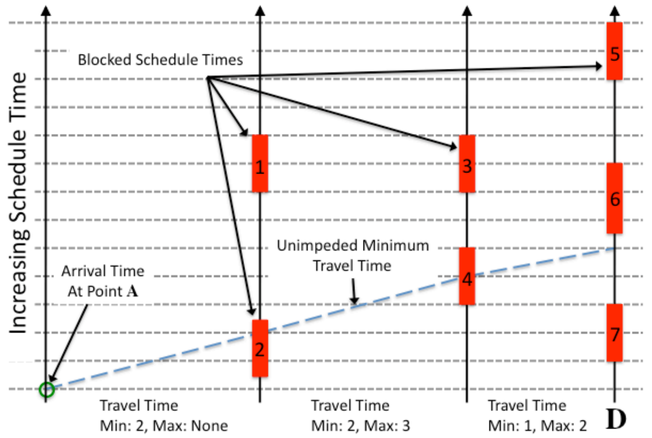


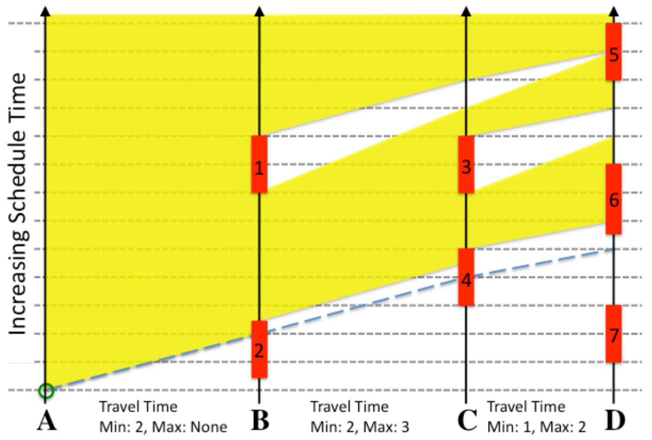
Figure 1. Node-link diagram for the example.

The constraints for this example are depicted in Fig. 2(a). Each point has a schedule time-line shown here with vertical grid markings for a time scale with arbitrary units. The red blocks represent schedule times that are, for example, unavailable due to previously scheduled flights. The green circle represents the arrival time of the flight at point **A**. This flight has a minimum travel-time of two time units between points **A** and **B**, and a maximum travel-time of "None," which indicates that indefinite holding is permitted. The minimum and maximum travel-times between points **B** and **C** are two and three time units, respectively. The minimum and maximum



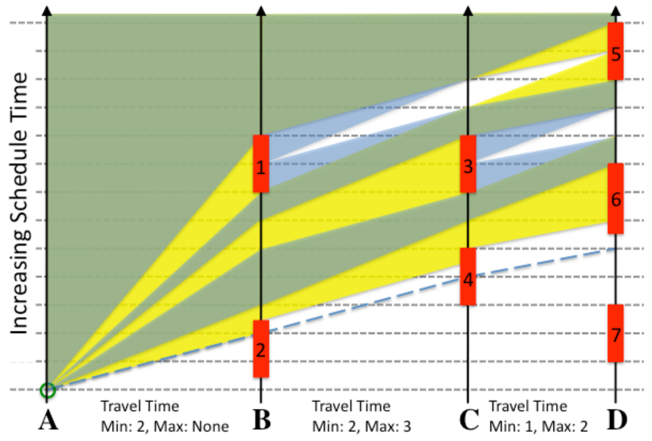
Increasing Schedule Time
Blocked Schedule Times
Unimpeded Minimum Travel Time
Arrival Time At Point A
1
2
3
4
5
6
7
Travel Time Min: 2, Max: None
Travel Time Min: 2, Max: 3
Travel Time Min: 1, Max: 2
D

(a)



Increasing Schedule Time
1
2
3
4
5
6
7
A
B
C
D
Travel Time Min: 2, Max: None
Travel Time Min: 2, Max: 3
Travel Time Min: 1, Max: 2

(b)



Increasing Schedule Time
1
2
3
4
5
6
7
A
B
C
D
Travel Time Min: 2, Max: None
Travel Time Min: 2, Max: 3
Travel Time Min: 1, Max: 2

(c)

Figure 2. Schedule constraint propagation. (a) *The dashed line shows the earliest estimated time of arrival based solely on minimum travel-time between points **A**, **B**, **C** and **D**. Minimum and maximum travel-times between points are specified. A maximum of "None" means that holding is permitted. The red rectangles represent unavailable schedule times at each point.* **(b)** *The yellow shaded regions represent feasible scheduling windows as travel-time and point schedule constraints are propagated downstream.* **(c)** *The blue shaded regions show the propagation of constraints upstream. The regions where the yellow and blue regions overlap represent viable scheduling windows that satisfy all constraints.*

travel-times between points **C** and **D** are one and two time units, respectively. The blue dashed line represents the schedule "path" for the earliest, unimpeded time of arrival at each point. This path has an available schedule time at point **D**, but this path violates scheduling restrictions at points **B** and **C**.

The propagation of scheduling constraints downstream, along the route from **A** to **D** is shown in Fig. 2(b). The yellow shaded region between points **A** and **B** is the "window" of potentially viable schedule times between **A** and **B**, given the arrival time constraint at **A**. The green circle identifies the arrival time at **A**. The minimum travel-time between **A** and **B** defines the lower bound of the scheduling window between **A** and **B**. The upper bound of the window is defined by the maximum travel-time, which in this case is unbounded. At point **B**, the reachable times of the scheduling window are constrained by the blocked schedule times #1 and #2. Based on these constraints, there are two scheduling windows between **B** and **C**. Again, the lower bounds are defined by the minimum travel-time and the upper bounds are defined by the maximum travel-time. It should be noted that schedule block #1 creates an unreachable range of schedule times at point **C**. Next, the reachable times at point **C** are constrained by the blocked schedule times #3 and #4, which results in three scheduling windows. These are then propagated to point **D** and constrained by the blocked schedule times #5 and #6. This completes the downstream propagation of constraints, which results in three viable scheduling ranges at point **D**.

To complete the identification of feasible scheduling windows, the viable scheduling ranges at point **D** are propagated back upstream, as shown in Fig. 2(c). The blue shaded regions between adjacent points illustrate how downstream constraints are propagated upstream. Areas where these blue regions overlap the yellow, downstream propagation regions are shown as green. In propagating schedule constraints upstream, the lower bounds of the reachable upstream schedule time windows are defined by the maximum travel-time, and the upper bounds are defined by the minimum travel-times. For a schedule to be viable, it must satisfy both downstream and upstream constraints. In Fig. 2(c), the viable scheduling windows are the green areas, where the downstream propagation regions, shown in yellow, and upstream propagation regions, shown in blue, overlap.

B. Application of Constraint Algebra to the Example

The foregoing example shows how constraint propagation can be performed manually. The process can also be programmed by explicitly testing each individual constraint in the sequence described. However, the constraint algebra described in the appendix of Ref. 5 can be used to represent these calculations more concisely. The algebra represents time constraints as ranges of allowable schedule times at points or travel-times between points. The simplest constraint is a single pair of values that represent the minimum and maximum limits of a viable time range. In addition to numerical values, either limit can be set to a value of "*None*" to indicate that the limit has no constraining value. Constraints representing scheduling ranges at a point can consist of multiple time range pairs, while constraints representing the travel-time range between points should consist of a single time range pair.

The constraint algebra defines the "&" operator to represent the logical "*and*" operation for combining two sets of scheduling constraints, as well as the "|" operator to represent logical "*or*" operation. A logical "*not*" operation is also defined. The "+" operator in the expression $\left(A + \overrightarrow{ab}\right)$, is used to propagate the schedule-time constraints at A downstream using the travel-time constraint given by $\overrightarrow{ab}$. This is defined as adding the minimum travel-time constraint of $\overrightarrow{ab}$ to each minimum schedule-time constraint in *A*, and adding the maximum travel-time constraint of $\overrightarrow{ab}$ to each maximum schedule-time constraint in *A*. The "-" operator in the expression $\left(B-\overrightarrow{ab}\right)$, is used to propagate the schedule-time constraints at *B* upstream using the travel-time constraint given by $\overrightarrow{ab}$. This is defined as subtracting the maximum travel-time constraint of $\overrightarrow{ab}$ from each minimum schedule-time constraint in *B*, and subtracting the minimum travel-time constraint of $\overrightarrow{ab}$ from each maximum schedule-time constraint in *B*. The "+" and "-" operators also support the addition of scalar values to constraints, as defined in Ref. 5.

Equation (1) prescribes the initial constraint at point **A**, which is represented by the variable A_i. The subscript "*i*" indicates that it is an initial constraint value. The value (0,0) indicates that the arrival time at A is precisely zero, which is the start time for this example. Equations (2)-(4) provide the initial constraint values for points **B**, **C** and **D**. For convenience, these are defined in terms of the blocked scheduled range pairs for the blocked ranges shown in Fig. 2. The "*not*" operator is used to convert them from blocked schedule range pairs to allowable schedule range pairs. For example, the initial constraint for point **B**, given by Eq. (2), would be [(*None*,0.5),(2.5,7),(9,*None*)] after the application of the *not* operator. This indicates that an allowable schedule time would be before 0.5, between 2.5 and 7, or after 9.

$$A_i = \left[(0,0)\right] \tag{1}$$

$$B_i = not\left[(0.5,2.5),(7,9)\right] \tag{2}$$

$$C_i = not\left[(3,5),(7,9)\right] \tag{3}$$

$$D_i = not\left[(1,3),(5.5,8),(11,13)\right] \tag{4}$$

Equation (5) defines variable, $\overrightarrow{ab}$, as the travel-time constraint between points **A** and **B**. Travel-time constraints are defined as a single pair of constraint values representing the minimum and maximum travel-times. In this case, the minimum travel-time is two units, and the maximum travel time is *None*, which means the flight can be delayed indefinitely along this link. Equations (6) and (7) give the travel-time constraints between points **B** and **C**, and between points **C** and **D**.

$$\overrightarrow{ab} = \left[(2,None)\right] \tag{5}$$

$$\overrightarrow{bc} = \left[(2,3)\right] \tag{6}$$

$$\overrightarrow{cd} = \left[(1,2)\right] \tag{7}$$

The point constraints in Fig. 2(b) are propagated downstream by use of the travel-time constraints between points. The propagation of the constraint at point **A** to point **B**, via the travel-time constraint, $\overrightarrow{ab}$, is given by the term ($A_i + \overrightarrow{ab}$) in Eq. (8). The result is combined with the initial constraint at point **B** using the logical "&" operator. The resulting constraint value is assigned to a new, temporary, constraint value for point **B**, designated B_t. The constraint value is subscripted with the letter *"t"* because it is considered temporary as it does not include the influence of downstream constraints. In a similar fashion, Eq. (9) is used to calculate a temporary value for the constraint at **C** using constraints propagated from upstream. Equation (10) completes the downstream propagation of constraints. Because there are no downstream points, the result is subscripted with the letter *"f "* to indicate that it is the final constraint value for point **D**.

$$B_t = \left(A_i + \overrightarrow{ab}\right) \& B_i \tag{8}$$

$$C_t = \left(B_t + \overrightarrow{bc}\right) \& C_i \tag{9}$$

$$D_f = \left(C_t + \overrightarrow{cd}\right) \& D_i \tag{10}$$

In Fig. 2(c), the point constraints are propagated upstream by use of the travel-time constraints between points. In Eq. (11), the final constraint at **D** is propagated upstream to **C** by the term (D_f- $\overrightarrow{cd}$). The "-" operator represents the calculations needed for upstream propagation of the constraints. By use of the logical "&" operator, the result is combined with the previously calculated constraint value, C_t. The result is the final constraint value for point **C**, designated C_f. In a similar fashion, Eqs. (12) and (13) are used to propagate constraints upstream and calculate the final constraint values for points **B** and **A**.

$$C_f = \left(D_f - \overrightarrow{cd}\right) \& C_t \tag{11}$$

$$B_f = \left(C_f - \overrightarrow{bc}\right) \& B_t \tag{12}$$

$$A_f = \left(B_f - \overrightarrow{ab}\right) \& A_i \tag{13}$$

Using software that implements the constraint algebra, the time constraint values given in Equations (1)-(7) were entered into the propagation equations, Eqs. (8)-(13). Evaluation of these equations produces the following values for the feasible schedule ranges at each point. These values agree with those that were determined manually in Fig. 2(c).

$$A_f = \left[(0,0)\right] \tag{14}$$

$$B_f = \left[(3,5),(6,7),(9,None)\right] \tag{15}$$

$$C_f = \left[(6,7),(9,10),(11,None)\right] \tag{16}$$

$$D_f = \left[(8,9),(10,11),(13,None)\right] \tag{17}$$

C. The Multi-point Scheduling Algorithm

Equations (8)-(13) provide a solution for a four-point schedule. Similar sets of equations can be easily developed for other scheduling point counts. For *N* points, 2(*N*-1) equations are needed. To avoid having to generate customized equation sets for different numbers of scheduling points, a recursive algorithm was developed, which is depicted in Fig. 3.

```
1    def getConstraints(path):
2        begin, link, end = path[:3]
3        next = path[3:]
4        newend = (begin+link)&end
5        if next == []: # End of the list
6             return [(newend-link)&begin,newend]
7        else:
8            conlist = getConstraints([newend]+next)
9            newend = conlist[0]
10           return [(newend-link)&begin] + conlist
```

Figure 3. Multi-point constraint propagation algorithm. *The algorithm, as implemented in the Python programming language, is presented here with line numbers added. The algorithm takes a list, named* `path`*, consisting of alternating point constraints and travel-time constraints, ending with a final reference point constraint. It returns a list of constraint objects, one for each reference point, that represent the viable scheduling times.*

The code was written in the object-oriented, programming language Python, which supports the creation of data structures called "objects" with specialized methods and operators. It utilizes previously developed code[5] that implements constraint objects that support logical operators for combining constraints and the previously mentioned "+" and "-" operators for propagating constraints in both the downstream and upstream directions. The "`getConstraints`" function listed in Fig. 3, takes an input named "`path`," which is an alternating list of point constraints and travel-time constraints, ending with a final point constraint. Line 2 of the algorithm takes the first three items of the list and assigns them to the variables "`begin`," "`link`," and "`end`." The variables "`begin`" and "`end`" are the constraint objects for the first two sequential points of the list and "`link`" is the constraint object for the travel-time between them. In line 2, the remainder of the "`path`" list is assigned to the variable "`next`." Line 3 propagates the "`begin`" constraint downstream by adding the "`link`" constraint and is then combined with the "`end`" constraint using the "`&`" operator. The result is assigned to the variable, "`newend`." Line 4 tests to see if "`next`" is empty, which means the constraints have been propagated to the end of the original constraints list given by "`path`." If "`next`" is empty, then it returns a two-item list, which is the seed value for the process used to propagate constraints to upstream points.

The remainder of the upstream propagation process, if needed, takes place if "`next`" is not empty and begins in line 8. Here "`newend`" is pre-pended to the list "`next`" and is passed in a recursive call to "`getConstraints`," the result is assigned to the variable "`conlist`." This result is the list of the final downstream constraints including "`newend`." In line 9, the current value of "`newend`" is replaced by the first item in "`conlist`," which has downstream constraints included. In line 10, the constraints for "`newend`" are propagated upstream by subtracting "`link`" and then combined with "`begin`" using the "`&`" operator. This result is pre-pended to "`conlist`" and then returned.

This ten-line algorithm is valid for any number of points, and the number of calculations varies linearly with the number of points. Fig. 4 shows the feasible scheduling windows for an example with nine scheduling points, eight travel-time constraints, and twenty-six blocked schedule-times. This calculation only took 5ms on a single 3Ghz processor core. Computation time for the entire calculation could likely be reduced by at least an order-of-magnitude if a multi-threaded, compiled programming language were used instead of Python.

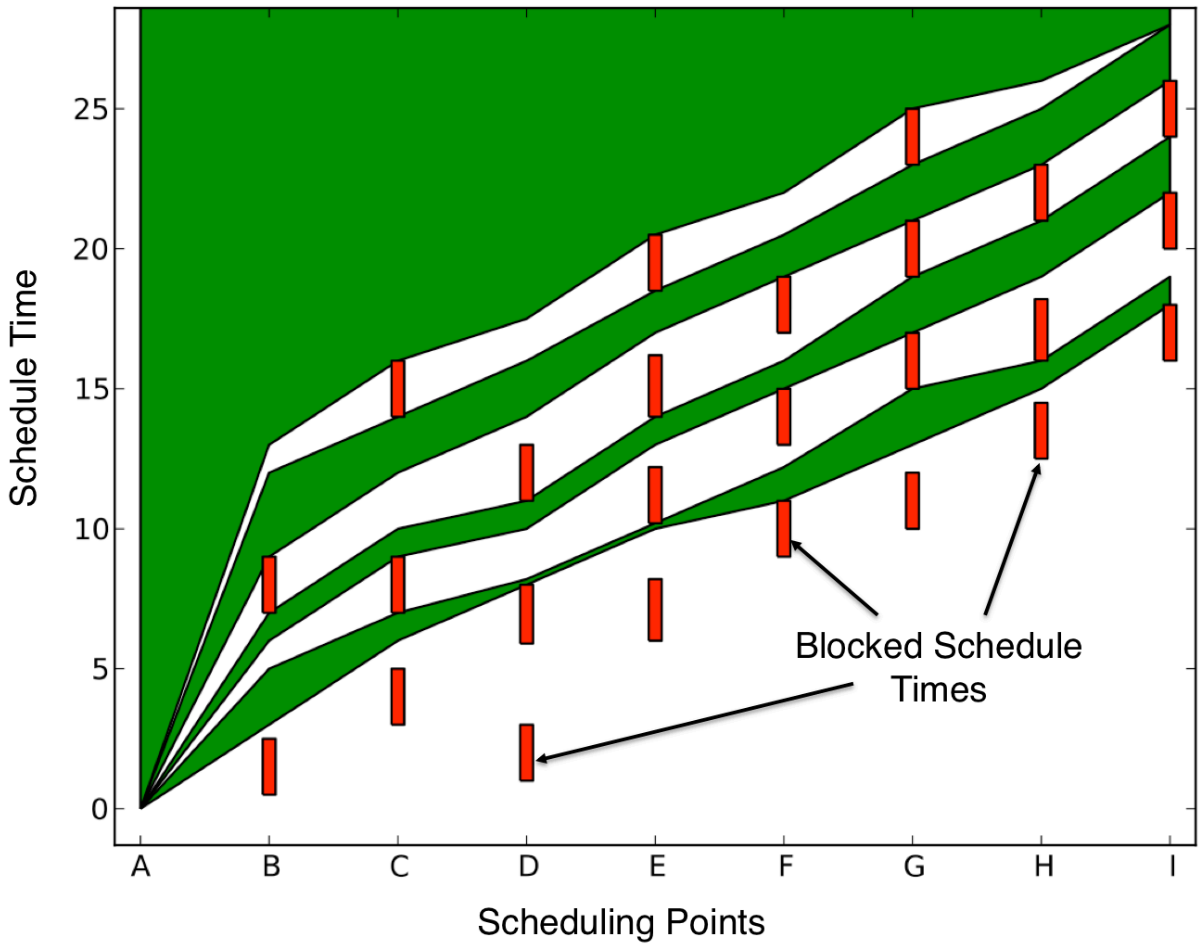


Figure 4. Plot of viable scheduling windows for a nine-point example problem. *Blocked schedule times are depicted as red bars for each scheduling point. The viable scheduling windows are shown in green.*

III. A Multi-Point Scheduling Simulation

To demonstrate the algorithm with a more relevant multi-point scheduling scenario than the foregoing examples, a simple terminal area arrival simulation was developed. The scenario chosen was adapted from a scenario developed for ongoing research on very closely spaced parallel approach (VCSPA) operations.[9,11-14] To study potential new procedures, a modified set of Standard Terminal Arrival Routes for San Francisco International Airport was developed for instrument approaches that allow the pairing of some flights for formation landings on runways

28L and 28R. Because these two runways are only 750 feet apart, simultaneous use is currently only allowed under visual flight rules. Currently, when visibility is poor, only one runway is used, which reduces the airport's arrival capacity by one-half.

The basic idea for VCSPA operations is presented in Fig. 5, which has two aircraft on paired approach paths. The leading aircraft has a straight-in approach to runway 28L. The following aircraft flies along the path shown. When the lead aircraft crosses the coupling point, the autopilot of the following aircraft couples with the leader in order to maintain constant spacing within a safe zone. The safe zone is specified as between five and twenty-five seconds in trail, which is far enough behind the leading aircraft to avoid a blunder by the leader and far enough forward to avoid the hazard posed by the leader's wake vortices.[13,14]

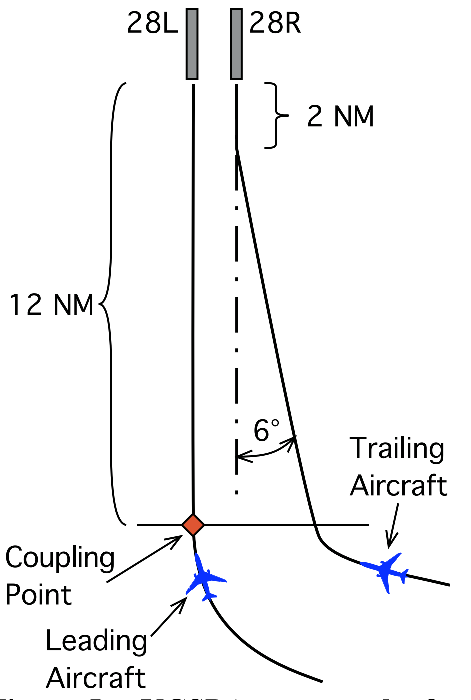


Figure 5. VCSPA approach for San Francisco

The six arrival routes into San Francisco that were developed for evaluation of VCSPA operations are depicted in Fig. 6. Each route has a "split point," where a flight is either directed to runway 28L or it is designated as a follower in a paired approach into to runway 28R. For the scheduling simulation, there are four reference points along each route where time-based separation constraints are enforced. These are the route entry points, the split points, the coupling points and the runway thresholds. At the route entry points and the split points, the time separation constraints were set to enforce five nautical mile separations for the expected aircraft speeds at these locations. At the coupling points and at the runway thresholds, the time separation constraints are set to enforce standard wake separation distances,[15] for the expected approach and landing speeds. All of these separation constraints were represented as constraint pairs and combined, where necessary, by use of constraint algebra operations.

The minimum travel-time constraints between the points were based on nominal speeds and the maximum travel-time constraints were assumed to be 10% greater than the nominal

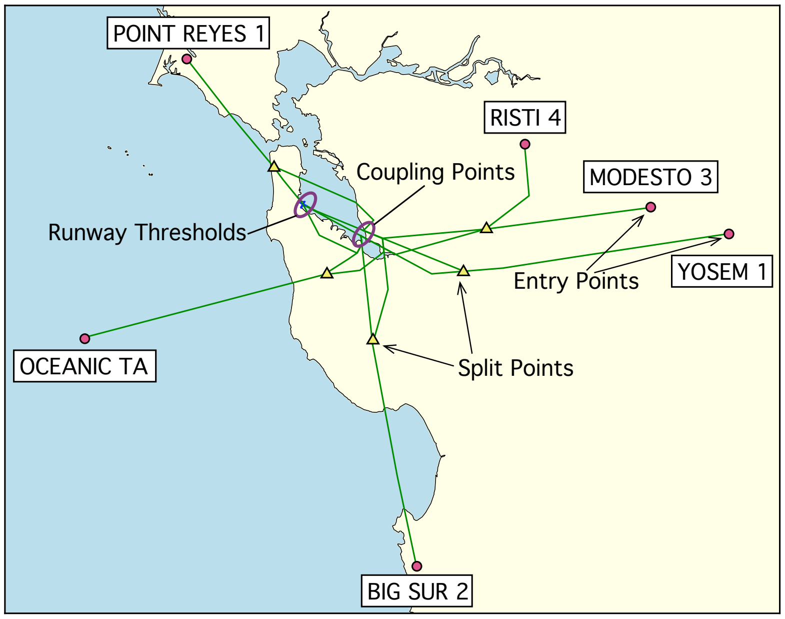


Figure 6. Simulated terminal arrival routes for San Francisco International Airport.

travel-times. Flights designated as follower and directed to runway 28R were further constrained to be within five and twenty-five seconds of the leader's scheduled arrival time at the coupling point and at the runway threshold. It is also assumed that flights could be delayed any amount of time prior to their arrival at the route entry points. To determine the allowable scheduling times for each possible route, the point constraints and the travel-time constraints were combined into a list for processing by the multi-point algorithm described in section II.C.

A simple event-based, first-come-first-served scheduler was also developed for the simulation. The scheduler maintained a queue of flights that were available for assignment of scheduling times. When a flight bound for San Francisco crosses the planning-horizon, shown in Fig. 7, the flight is added to the scheduling queue. The flight is first assigned an arrival route. Then the unimpeded estimated time of arrival (ETA) to the route entry point is calculated, along with the unimpeded runway threshold ETA. When any flight in the scheduling queue is determined to be within ten minutes of it's route entry point ETA, a scheduling event is triggered. Then the flights in the scheduling queue are sorted by the order of their runway threshold ETAs, and the flight with the earliest ETA is chosen for scheduling. Blocked schedule-times at each point along a flight's route are determined from the scheduled arrival times of previously scheduled flights along with separation times that provided the minimum required separation distances at the expected flight speeds. The point constraints and travel-time constraints are evaluated for routes to both runways 28L and 28R by use of the multi-point scheduling algorithm. Flights are only assigned to 28R if they can be follower aircraft in a paired approach. To enforce this rule, the route to 28R is only assigned if the flight can meet the follower constraints when paired with a preceding flight scheduled for 28L. If those constraints cannot be met, then there is no viable schedule to 28R and the flight would be assigned the route to 28L. The earliest viable time of arrival for each point is entered into the schedule and the flight is removed from the scheduling queue. Because flights are scheduled in the order of the their runway arrival ETAs, it is possible that the flight that triggered the scheduling event is not the first flight to be scheduled. If this is the case, then additional flights are scheduled in the sorted order until the flight that triggered the scheduling event is scheduled.

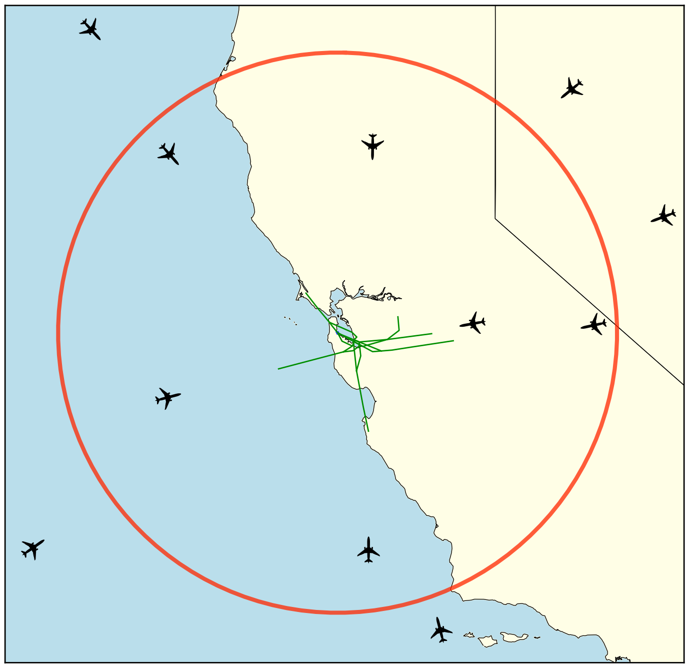


Figure 7. The 200 nmi radius planning-horizon centered on San Francisco International Airport.

To simplify the problem, only scheduled arrivals into San Francisco were included in the simulation and this list of flights is further reduced to only include aircraft equipped with jet engines. A total of 362 flights, derived from scheduled arrivals for April 3, 2006 were simulated in a scenario representing current day traffic. To simulate a future scenario with twice the traffic, a second scenario was created where each flight in the first scenario was entered twice, but with the nominal arrival time of the second flight entry is offset by five minutes from the first. Both of these scenarios were simulated with and without VCSPA operations enabled.

IV. Simulation Results

The simulation provided scheduled arrival times at each reference point for each flight along an available route. For each scenario simulation, the schedule times were individually evaluated to ensure that proper separation times were enforced. These evaluations found no separation violations, which is an indication that the scheduling constraints were properly enforced. Table 1 presents a summary of total and average delay times predicted by the simulation for each scenario. For the current day, "1x Traffic," traffic load, when VCSPA formation landings were enabled, the total and mean delays are reduced by 72% when compared to the single runway arrival operations that are currently allowed when visibility low. When the traffic load is doubled, the delay reduction is 98% when VCSPA operations are enabled. The 1x traffic scenario, with VCSPA formation landings, has 44 formation landings, while the 2x traffic scenario has 194 formation landings.

Table 1. Total and average delay for scenarios with and without VCSPA operations.

	Without VCSPA		With VCSPA	
Traffic Load	Total Delay (hr)	Mean Delay (min)	Total Delay (hr)	Mean Delay (min)
1x Traffic	4.7	0.8	1.3	0.2
2x Traffic	902	75	13.5	1.1

V. Concluding Remarks

It was demonstrated that multi-point, time-based scheduling problems, for a series of points with both scheduling time constraints and travel-time constraints, can be solved by constraint propagation. A previously developed constraint algebra, that represents the combination and propagation of constraints, is used to develop a new, closed-form, multi-point scheduling algorithm. The algorithm provides all potential scheduling windows for any number of points with multiple scheduling constraints at each point. The algorithm is fast and computation time scales linearly with the number of reference points.

This algorithm was used to develop a scheduling simulation for flights into San Francisco International Airport for very closely spaced parallel approach operations that allow formation landings. The first-come, first-served scheduler developed is able to identify numerous formation-landing pairs for a network of approach routes, each having four scheduling points. This preliminary study showed that delay reductions of 72% to 98% might be feasible by enabling the use of a second parallel runway under adverse weather conditions.

Use of constraint algebra and the multi-point scheduling algorithm made it very easy to develop a first-come, first-served scheduler for a relatively complex route structure. The ability to quickly calculate closed-form scheduling solutions for a given sequence of flights should also be useful in developing more optimal schedulers. One application would be schedulers that implement constrained position shifting,[16] which allow limited modifications to a first-come, first-served sequence. However, any optimization scheme that primarily operates by permutation of scheduling order could potentially benefit from rapid, closed-form, scheduling calculations.

References

[1]Federal Aviation Administration, "Time-Based Flow Management (TBFM): SYSTEM SPECIFICATION DOCUMENT (SSD)," URL: https://faaco.faa.gov/attachments/Section_J-3.pdf [cited 2 June 2010].

[2]Neuman, F., and Erzberger, H., "Analysis of Sequencing and Scheduling Methods for Arrival Traffic," NASA Technical Memorandum 102795, Ames Research Center, 1990.

[3]Neuman, F., and Erzberger, H., "Analysis of Delay Reducing and Fuel Saving Sequencing and Spacing Algorithms for Arrival Traffic," NASA Technical Memorandum 103880, 1991.

[4]Wong, G. L., "The Dynamic Planner: The Sequencer, Scheduler, and Runway Allocator for Air Traffic Control Automation," NASA TM-2000-209586, April 2000.

[5]Meyn, L. and Erzberger, H, "Airport Arrival Capacity Benefits Due to Improved Scheduling Accuracy," *Proceedings of the AIAA 5th Aviation, Technology, Integration, and Operations Conference*, AIAA, Washington, DC, 2005.

[6]Landry, S., Farley, T., Foster, J., Green, S., Hoang, T., and Wong, G., "Distributed Scheduling Architecture for Multi-Center Time-Based Metering," *AIAA's 3rd Annual Aviation Technology, Integration, and Operations (ATIO) Forum*, AIAA, Washington, DC, 2003.

[7]Santos. M, Feinberg, A., Zhang, Y., Teng, Y., Chen, J., Nigam, N., and Smith, J., "Scheduling the Use of Airborne Merging and Spacing Along Multiple Converging Routes to an Airport," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, AIAA, Washington, DC, 2010 (to be published).

[8]Anagnostakis, I., Clarke, J-P., Bohme, D., and Volckers, U., "Runway operations planning and control sequencing and scheduling," *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference*, 3-6 Jan. 2001.

[9]Kupfer, M., "Scheduling Aircraft Landings to Closely Spaced Parallel Runways," *Eighth USA/Europe Air Traffic Management Research and Development Seminar (ATM2009)*, Napa, CA, 2009.

[10]van Leeuwen, P., Hesselink, H., & Rohling, J., "Scheduling Aircraft Using Constraint Satisfaction," *National Aerospace Laboratory NLR*, NLR-TP-2002-299, Amsterdam, 2002.

[11]Rossow, V. J, "Use of Individual Flight Corridors to Avoid Vortex Wakes," *AIAA Atmospheric Flight Mechanics Conference*, AIAA, Washington, DC, 2002.

[12]Rossow, V. J., Hardy, G. H., and Meyn, L. A., "Models of Wake-Vortex Spreading Mechanisms and Their Estimated Uncertainties", *5th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, AIAA, Washington D.C., 2005.

[13]Rossow, V. and Meyn, L, "Guidelines for Avoiding Vortex Wakes During Use of Closely-Spaced Parallel Runways," *26th AIAA Applied Aerodynamics Conference*, AIAA, Washington D.C., 2008.

[14]Verma, S., Lozito, S., and Trott, G., "Preliminary Guidelines on Flight Deck Procedures for Very Closely Spaced Parallel Approaches," *International Council for the Aeronautical Sciences (ICAS) 2008 Congress*, Anchorage, AK, 2008.

[15]Federal Aviation Administration, "Air Traffic Control Order 7110.65T," URL: http://www.faa.gov/documentLibrary/media/Order/7110.65TBasic.pdf [cited 2 June 2010].

[16]Balakrishnan, H. and Chandran, B., "Scheduling Aircraft Landings Under Constrained Position Shifting," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA, Washington, DC, 2006.