SCENARIO-FREE SEQUENTIAL DECISION MODEL FOR THE SINGLE AIRPORT GROUND HOLDING PROBLEM

Pei-chen Barry Liu, Mark Hansen, University of California, Berkeley, CA

Abstract

This paper aims to advance the support of decision-making in air traffic flow management under uncertainty with a focus on the single airport ground holding problem (SAGHP). Learning from the shortcomings of the scenario-based models for SAGHP that address uncertainty using probabilistic capacity scenarios, we develop a sequential decision model that is not limited by a small set of scenarios. We present computational strategies and demonstrate the computational feasibility of the model.

Introduction

Ground Delay Programs (GDPs) mitigate destination airport capacity-demand imbalances by assigning ground delays to flights at their origin airports. The ground-holding problem (GHP) is that of ground-holding flights to minimize the overall cost of ground and airborne delays. The ground-holding problem for a single destination airport is termed the Single Airport Ground Holding Problem (SAGHP). The SAGHP has been modeled using scenario-based stochastic programming models in several studies [1][2][3]. In this context, a scenario is a representative time series of arrival capacity values for an airport over a day or some part of a day, as illustrated in Figure 1.

One of the primary shortcomings of the scenario-based models is that they assume a limited number of capacity scenarios when in reality there is a much larger set of possibilities for capacity evolution. The number of scenarios is limited by model solver capacity and data availability. In addition, scenario-tree-based models impose a scenario tree structure when in reality improved information about future capacity is obtained continually rather than at a few discrete branching points. On the implementation side, a previous empirical study [4] has shown that the costs incurred from applying the output of scenario-based optimization models to airports is considerably higher than the theoretical optimization results suggest. The reasons for this discrepancy include a) the actual capacities vary around the nominal values assumed in the optimization; b) there is uncertainty in correctly identifying the scenario that matches best with the condition as more information of the capacity evolution is obtained over time. Hence, in this paper, we consider a "scenario-free" sequential decision making model to avoid these inadequacies.

The use of dynamic programs for the GHP was explored by Andreatta and Romanin-Jacur [5] for a simplified single-time period SAGHP and Terrab and Odoni [6] for SAGHP with multiple time periods. Terrab and Odoni formulated a dynamic program based on the assumption that a fixed landing priority rule has been specified for the flights, and solved for ground-holds for each individual flight. The stochasticity in the problem was accounted for using a set of capacity scenarios. However, their model is a static one, since it does not allow for recourse actions. Since then, the use of dynamic programming for the SAGHP has received little attention because of concerns about the curse of dimensionality. A decade after the initial attempt of using dynamic programs to solve SAGHP, we investigate the applicability of a true "dynamic" program for SAGHP in light of the limitations of the scenario-based stochastic programs, and ever-increasing computing power.

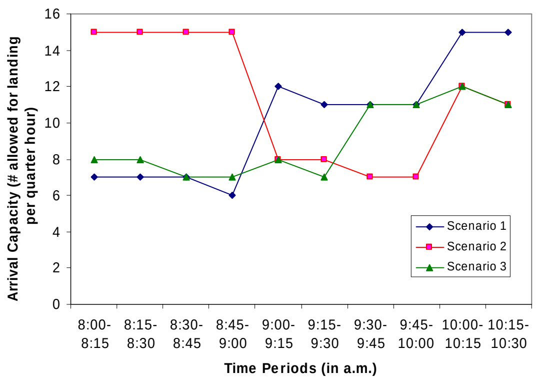

Figure 1. Illustration of Capacity Scenarios

Sequential Decision Model for SAGHP

The process of making ground-holding decisions in a SAGHP can naturally be described as sequential decision making. The ground holding decisions are responses to the capacity uncertainties at the destination airport, and we want to react optimally to the most updated information. We formulate this sequential decision making model using a dynamic program. In this model, the states we react to are the arrival capacity levels at the destination airport; the actions (or decision variables) are the flights to hold on the ground at each decision period; the objective function is to minimize total expected delay cost; and the optimal policies are the actions to take at any state-action trajectory. We assume the availability of transition probability matrices of airport arrival capacities at each time period. We also assume that both the queue on the ground and the queue in the air will clear at the end of the planning horizon. As such, no delay cost would be incurred after the planning horizon. We introduce the following notations:

F = the set of all the flights considered in the planning horizon.

T = the last time period in the planning horizon.

$$X_f^t = \begin{cases} 1 & \text{if flight } f \text{ stays on the ground during period } t \\ 0 & \text{otherwise} \end{cases} \quad \forall f, \forall t.$$

$$Y_f^t = \begin{cases} 1 & \text{if flight } f \text{ is planned by the model to arrive in } t \\ 0 & \text{otherwise} \end{cases} \quad \forall f, \forall t.$$

$$S_f^t = \begin{cases} 0 & \text{if flight } f \text{ is scheduled to depart in or before } t \\ 1 & \text{otherwise} \end{cases} \quad \forall f, \forall t.$$

τ_f = the duration of flight of flight *f*.

A_t = the number of flights planned by the model to arrive in time period *t*.

L_t = number of flights intending to land at time *t*.

W_t = number of flights experiencing airborne delay at time *t*.

K_t = arrival capacity at the destination airport at time *t*.

$K_{\max}(t)$ = maximum arrival capacity at the destination airport at time *t*.

$K_{\min}(t)$ = minimum arrival capacity at the destination airport at time *t*.

$P_{kk'}$ = transition probability from arrival capacity *k* to *k'* in the next period.

c_g = cost of ground delay for one time period per flight.

c_a = cost of airborne delay for one time period per flight.

Among the above notations, only X_f^t is a decision variable. S_f^t and τ_f are given from the flight schedule and it is assumed that the duration of flight is deterministic. Y_f^t and A_t are auxiliary variables to help describe the dynamics. The optimality equation can be stated using the notation defined above as:

$$f_t(K_t) = \min_{X_f^t, f \in F} \left\{ c_g \sum_{f \in F} (X_f^t - S_f^t) + c_a W_t + \sum_{K_{t+1}=K_{\min}(t+1)}^{K_{\max}(t+1)} P_{K_t K_{t+1}} \cdot f_{t+1}(K_{t+1}) \right\}$$

for $1 \le t \le T-1$ (1a)

s.t.

$$Y_f^{t+1+\tau_f} = X_f^t - X_f^{t+1} \quad \forall f \in F, t = 0, \ldots, T-1; \quad \text{(1b)}$$

$$A_t = \sum_{f \in F} Y_f^t \quad t = 0, \ldots, T; \quad \text{(1c)}$$

$$W_t = (L_t - K_t)^+ \quad t = 1, \ldots, T; \quad W_0 = 0; \quad \text{(1d)}$$

$$L_t = A_t + W_{t-1} \quad t = 1, \ldots, T; \quad \text{(1e)}$$

$$X_f^t \ge S_f^t \quad \forall f \in F, t = 0, \ldots, T; \quad \text{(1f)}$$

$$X_f^t \ge X_f^{t+1} \quad \forall f \in F, t = 0, \ldots, T-1; \quad \text{(1g)}$$

$$X_f^t \in \{0,1\}, \quad Y_f^t \in \{0,1\} \quad \forall f \in F, t = 0, \ldots, T; \quad \text{(1h)}$$

and $f_T(K_T) = c_a W_T$. (1i)

When the difference $X_f^t - S_f^t$ equals to one, the flight *f* experiences ground holding during period *t*. Therefore, the number of flights to hold in period *t* is the sum of such differences over all the flights. Constraint set (1b) ties the planned arrival time of flight ($t + 1 + \tau_f$) to its departure time ($t + 1$) and its duration (τ_f) through the auxiliary variable Y_f^t. Equation set (1c) calculates the number of flights planned to arrive in time *t*, A_t. Constraint set (1e) states that the number of flights desiring to land in period *t* includes those planned to arrive in period *t* and those queued in the air in period $t - 1$. Constraint set (1f) ensures that a flight does not take off until its scheduled departure time.

In this formulation, the total delay cost is optimized on the basis of ground holding decisions for each individual flight. Clearly, the action space (to hold or not to hold each flight) is combinatorial with worst case of $2^{|F|}$ per period and the computation is likely to be intractable. The computational load can be greatly reduced when we consider flights in groups classified by flight durations. When a group of flights of the same duration are candidates for release in a given period, it is clear that the objective function depends only on the number of flights from this group to release, not on which individuals to release. We thus reformulate the problem by grouping the flights in F according to their durations. We introduce the following notations for the reformulation.

Γ = the set of groups that represent the classification of flights.

Z_γ^t = number of group γ flights to hold on the ground in time period *t*.

ξ_γ^t = number of group γ flights planned by the model to arrive in time period *t*.

S^t_γ = number of group γ flights scheduled for departure at time t.

G^t_γ = number of group γ flights queued on ground from previous periods by time t.

The reformulated optimality equation can be expressed as:

$$f_t(K_t) = \min_{\substack{0 \le Z^t_\gamma \le G^t_\gamma + S^t_\gamma \\ \gamma \in \Gamma}} \left\{ c_g \sum_{\gamma \in \Gamma} Z^t_\gamma + c_a W_t + \sum_{K_{t+1} = K_{\min}(t+1)}^{K_{\max}(t+1)} P_{K_t K_{t+1}} \cdot f_{t+1}(K_{t+1}) \right\}$$

for $1 \le t \le T-1$ (2a)

s.t.

$$G^t_\gamma = Z^{t-1}_\gamma \quad \forall \gamma \in \Gamma, t = 1, \ldots, T; \quad G^0_\gamma = 0, \forall \gamma \in \Gamma; \quad (2b)$$

$$\xi^{t+\tau_\gamma}_\gamma = G^t_\gamma + S^t_\gamma - Z^t_\gamma \quad \forall \gamma \in \Gamma, \quad t = 0, \ldots, T-1; \quad (2c)$$

$$A_t = \sum_{\gamma \in \Gamma} \xi^t_\gamma \qquad t = 1, \ldots, T; \quad (2d)$$

$$W_t = (L_t - K_t)^+ \quad t = 1, \ldots, T; \quad W_0 = 0; \quad (2e)$$

$$L_t = A_t + W_{t-1} \quad t = 1, \ldots, T; \quad (2f)$$

$$Z^t_\gamma \in Z^+ \qquad \forall \gamma \in \Gamma, \quad t = 0, \ldots, T-1; \quad (2g)$$

and $f_T(K_T) = c_a W_T$. (2h)

In the optimality equation (2a), the decision variable is the number of flights to hold for each duration group γ in time period t, Z^t_γ. Constraint set (2b) associates the action in period t – 1 to its implication for the queue on the ground in period t. Constraint set (2c) links the departure and planned arrival of flights with their flight durations. Constraint set (2d) calculates the number of flights planned to land in time t, A_t, by summing over the number of flights planned to land in period t from all the duration groups.

The computational advantage of the reformulation can be illustrated using a simple example. Consider nine flights waiting for departure in the current time period. Suppose these nine flights belong to three groups of sizes two, three, and four. Using formulation (1a), there are 512 (2^9) candidate holding action combinations to evaluate. When solving for the number of flights to hold per group as in formulation (2a), only 60 (3 × 4 × 5) action combinations need to be evaluated. The model (2a) reduces the action space significantly; at the same time, the model yields the same optimal policy.

Before we discuss the mathematical properties of the model, let us consider a simple numerical example and illustrate the mechanics of the decision process. Figure 2 depicts the Sequential Decision Model (SDM) for a very simple SAGHP. In this example, there is one flight scheduled to depart at time 0 with flight duration of one time period. Destination capacity levels are 0 and 1 with a stationary transition probability matrix of $\begin{array}{c|cc} & 0 & 1 \\ 0 & 0.7 & 0.3 \\ 1 & 0.3 & 0.7 \end{array}$. The capacity in the initial period is 0. Unit delay costs are c_g = 1 and c_a = 3. There is a planning horizon of two periods, and it is guaranteed that the flight can land in the third period. The decision process is described by an acyclic directed graph with two types of nodes: decision nodes and action nodes. We refer to such a graph as an *SDM tree*. The decision node with capacity level k at decision period t is marked by "*Kt* = *k*." The action node with the number of flights to hold h at decision period t is marked by "*Ht* = *h*." The associated number of flights in the ground queue *Gt* and number of flights in the airborne queue *Wt* are listed next to each of the nodes. Also listed are the costs evaluated at each step. At a decision node, the decision maker selects the least costly admissible action. The value of the least-cost action chosen, *ft**(*Kt*), is the cost associated with this decision node at period t with capacity level *Kt*. At an action node, the decision maker evaluates the expected cost-to-go of taking this particular action. The expected cost-to-go associated with each actions is listed as "V = cost." In this example, the optimal policy is to hold the flight at period 0. As this example illustrates that there are quite a few calculations required even for a very small problem, this suggests that the dimension of the SDM tree could be huge for problems with realistic size.

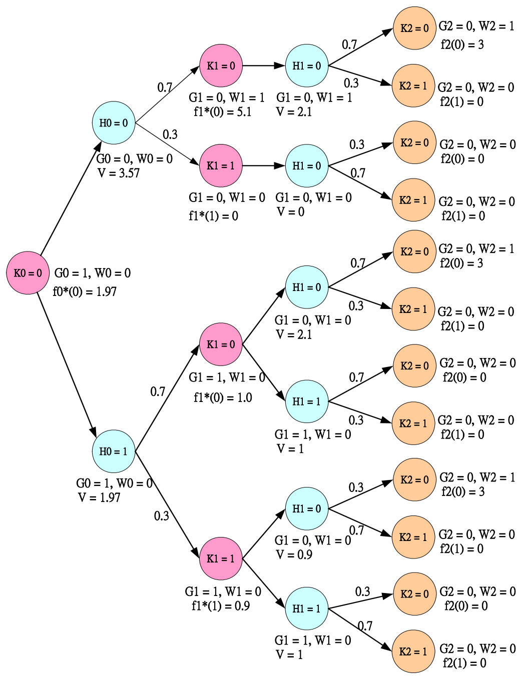

Figure 2. Simple Example of Sequential Decision Model of SAGHP Rendered as an SDM tree

Algorithmic Complexity

We compute the optimal policy for the dynamic program using the *value iteration* algorithm. Traditionally, two algorithms—policy iteration and value iteration—are considered for such problem. In our case, each of the iterations in the policy iteration algorithm involves a protracted iterative computation requiring sweeps through the state set. For that reason, we draw on the value iteration algorithm for our dynamic program. The value iteration is obtained by taking the optimality equation as an update rule. The backup operation includes policy improvement and truncated policy evaluation steps. While the state-action space is traversed recursively, the total number of truncated policy evaluation operations equals to the number of action nodes and the total number of policy improvement operations equals to the number of decision nodes, in the language used in Figure 2. As a result, the worst-case complexity of this algorithm can be derived via counting the number of nodes in the tree. Let us define:

T = the number of decision epochs in the planning horizon;

N = the number of capacity levels considered in a period;

M = the number of actions possible in a decision stage;

F = the number of flights to release in a decision stage;

G = the number of groups of flights.

It can be derived that the value iteration algorithm solves the SDM described by model (2a) in $O(((\frac{F}{G})^G N)^T)$ time.

Consider the existence of a priority ordering among the flights. Such ordering is summoned in current practice to ensure the equity in assigning delays to flights. From an optimization point of view, a priority ordering imposes constraints for making the optimal ground-holding decisions. Unlike the original problem that solves for the optimal combination of flights to hold from each group, with a priority ordering, the flights to hold are determined based on the ordering and we only need to solve for the optimal number of flights to hold. In this setup, the problem reduces to the special case of $G = 1$ and the algorithm can solve the problem in $O((FN)^T)$ time.

The complexity presented above show that the value iteration algorithm for this problem is an exponential-time algorithm. Note that this worst-case complexity is an upper bound on the running time for sufficiently large values of the parameters inside the big O notation. The running time of a problem depends also on the actual magnitude of these parameters and use of problem-specific computational strategies. In addition, heuristics that have potential to cut down redundant computations can be helpful. Several computational strategies are proposed in the next section to make a typical real-world problem solvable within reasonable time.

Computational Solution Strategies

The algorithmic complexity results in the previous section indicate the need for computational solution strategies to reduce the time complexity for problems of realistic size. In this section, we describe the use of memoization, priority ordering, and heuristics to alleviate the computational load.

Memoization

Dynamic programming is particularly efficient in problems in which the same subproblems occur over and over again. Fortunately, this is the case for our problem. *Memoization* is an algorithmic technique used to accelerate computation by storing the results of subproblems for later reuse, rather than re-solving them. Memoizing the top-down recursive algorithm is a variation of dynamic programming which traditionally solves the problems from bottom up. The bottom-up approach starts the computation from identifying and solving all the subproblems at the bottom level. To give an inkling of the predicament in the bottom-up approach, consider a SAGHP of hundreds of flights with various flight durations. The subproblems are characterized by the attributes of the flights waiting in the ground queue and the flights released but yet to land, in addition to the time and the capacity level at the destination airport. The combinatorial effect of these parameters can produce a huge number of subproblems. However, some of the subproblems need not be solved at all because they would arise only from clearly suboptimal upstream decisions that would never be made. Therefore, for the dynamic program for SAGHP, the top-down memoized algorithm has the advantage of only solving those subproblems that are definitely required, and solving them only once.

Priority Ordering

In Subsection 2.1 we discussed the value of priority ordering among flights in the reduction of algorithmic complexity. We showed that priority ordering of flights reduces the time complexity of the algorithm from $O(((\frac{F}{G})^G N)^T)$ to $O((FN)^T)$. Though the use of memoization changes the complexity developed previously, the savings from transforming an exponential component to polynomial should nevertheless be considerable. Therefore, we consider priority ordering as a second computational solution strategy.

One sensible way to prioritize flights is to release the longer duration flights first. This is justified for two reasons. First, delays to flights with longer duration typically have higher economic cost. For long haul flights, airlines usually use larger aircraft, and carry more passengers. Moreover, holding long haul flights on the ground incurs unrecoverable delay costs which may prove unnecessary if destination airport capacity evolves favorably. Short haul flights, in contrast, can be managed in a more responsive manner as capacity evolves over time. For these reasons, it is sensible to give flights with longer flight durations the priority to take off. This is recognized, albeit somewhat bluntly, in the current practice by "exempting" long-haul flights from GDPs. Here, we do not exempt such flights, but rather allow them to take-off ahead of shorter flights. We term this prioritization scheme *Longest Goes First* or LGF.

A second prioritization scheme, *Ration by Schedule* (RBS), is employed in current practice for non-exempt flights. As its name suggests, RBS gives flights with earlier scheduled arrival time the priority. The airline community considers RBS the most equitable method of allocating capacity. In contrast to LGF, RBS protects short-haul carriers and flights from bearing the brunt of GDPs. Thus, while LGF may be a more efficient scheme, RBS is a more equitable one, at least from airlines' point of view. (One could also employ a weighted priority scheme that considers both the schedule and flight duration, but this will not be investigated here.)

Heuristics for Searching for Best Action

The optimality equation (2a), i.e. the cost-to-go function at time *t*, is convex in $Z_\gamma^t, \forall \gamma \in \Gamma$ [4]. One adverse consequence of using LGF or RBS priority is that the cost-to-go function is no longer convex in the number of flights to hold. Given a set of flights ready for departure, F_p, one must evaluate the cost-to-go function for every integer from 0 to $|F_p|$ to guarantee that the true optimum is found. To further reduce the computation time, we investigate two heuristics, which we term H1 and H2, that ignore the non-convexity.

To describe these heuristics, we introduce the following notation. Let *n* be the number of flights ready to be released at period *t*. Denote the cost-to-go at decision node *p* when holding *x* flights by f_p (x). Also, let x^* denote the best number of flights to hold determined by a given algorithm. The heuristic algorithm H1 is described as follows.

Step 1: At decision node *p*, compute the cost-to-go for the actions at the two ends of the cost function; that is, releasing none of the flights, f_p (n), and releasing all of the flights, $f_p(0)$.

Step 2: If $f_p(0) \le f_p(n)$, set $x^* := 0$ and go to Step 3; otherwise, set $x^* := n$ and go to Step 4.

Step 3: FOR $x = 1$ to n-1 DO:

If $f_p(x) > f_p(x-1)$, then $f_p(x-1)$ is a local minimum. Set $x^* := x - 1$ and stop. Else continue.

Step 4: FOR $x = n$-1 to 1 DO:

If $f_p(x) > f_p(x+1)$, then $f_p(x+1)$ is a local minimum. Set $x^* := x + 1$ and stop. Else continue.

Step 5: x^* is the best number of flights to hold at decision node *p* and $f_p(x)$ is the lowest cost found by this algorithm.

Steps 1 and 2 are devised to cut down the chase and pick one direction to search from. If we were to search from both sides, we would often evaluate the function over the entire range and thus defeat the purpose of reducing computations. This algorithm finds the optimal action when the cost-to-go function is convex in the number of flights to hold. When the cost-to-go function is not convex, the solution it yields may be only a local optimum.

To further reduce the computations and arrive at suboptimal but probably good solutions, we propose heuristic algorithm H2. In this heuristic, we always begin our search from zero. This approach is motivated by the observation that in most cases the optimum number of flights to hold will be closer to 0 than *n*. Thus heuristic algorithm H2 is as follows.

Step 1: At decision node *p*, compute the cost-to-go $f_p(0)$.

Step 2: FOR $x = 1$ to n-1 DO:

If $f_p(x) > f_p(x-1)$, then $f_p(x-1)$ is a local minimum. Set $x^* := x - 1$ and stop. Else continue.

Step 3: x^* is the best number of flights to hold at decision node *p* and $f_p(x)$ is the lowest cost found by this algorithm.

While H2 may seem to reduce computational burden only slightly compared to H1, it can make a considerable difference in computation time. Considering that these heuristics are applied at each decision node in the SDM tree, cumulatively H2 can save a sizable number of computations for a large problem. The effect is multiplicative with the

	Case 1		Case 2		Case 3		Case 4		Case 5		Case 6	
Flight Index	Departure Period	Arrival Period	Departure Period	Arrival Period	Departure Period	Arrival Period	Departure Period	Arrival Period	Departure Period	Arrival Period	Departure Period	Arrival Period
1	0	4	0	2	0	1	0	1	0	1	0	1
2	0	4	0	2	0	1	0	2	0	3	0	2
3	0	4	0	4	0	2	0	2	0	3	0	3
4	0	4	0	4	0	4	0	4	0	4	0	4
5	1	5	1	3	1	2	1	2	1	2	1	2
6	1	5	1	3	1	2	1	3	1	4	1	3
7	1	5	1	5	1	3	1	3	1	4	1	4
8	1	5	1	5	1	5	1	5	1	5	1	5

Table 1. Flight Schedule for the Test Cases

number of time periods, so that a 10% saving over one period becomes a 2/3 saving over 10 periods.

Computational Experiments

We programmed the model in Java JDK 5.0 and used it to perform a series of experiments on a Linux server. The aim of the experiments was to compare the exact model with models incorporating flight prioritization and heuristic search. The latter models are expected to have a considerable advantage in computation time, but to yield solutions that are somewhat suboptimal. We sought to gauge the relative magnitude of these differences, and thereby assess the feasibility of the scenario-free approach to the SAGHP for real-world scale problems.

Experimental Design

We experiment with six test cases. The test cases all have eight flights to release but each test case has a different flight duration mix, as described in Table 1. The arrival capacity transition matrix (Table 2) is fixed throughout the planning horizon of six periods and the initial capacity level is two flights per period.

The flight duration is the same for all the flights in Case 1. Under this condition, the exact model and flight priority models are essentially the same. In this case, all models solve in about the same time (Figure 2), and arrive at the same optimal solution (Figure 3). As the mix of flight durations becomes more heterogeneous, computation time for the exact model increases dramatically, while remaining essentially constant for the priority models. The use of priority ordering reduces computation time 87% with two flight durations (Case 2), and 97% when there are four durations (Case 6). Not surprisingly, the LGF and RBS priority scheme exhibit similar computation times for each case, although RBS is consistently somewhat faster to solve.

from \ to	2	3	4
2	0.4	0.4	0.2
3	0.2	0.6	0.2
4	0.2	0.4	0.4

Table 2. Capacity Transition Matrix for Experiments

On the other hand, optimal total delay costs computed by the priority models are not much higher than that obtained using the exact model. In fact, in every case one of the priority models arrives at virtually the same optimal cost as the exact algorithm. Among the six test cases, the biggest deviations from the exact optimal cost due to LGF and RBS are 29% (RBS, Case 3), 33% (RBS, Case 4), and 32% (LGF, Case 5). There is no clear pattern concerning the relative sub-optimality of LGF and RBS.

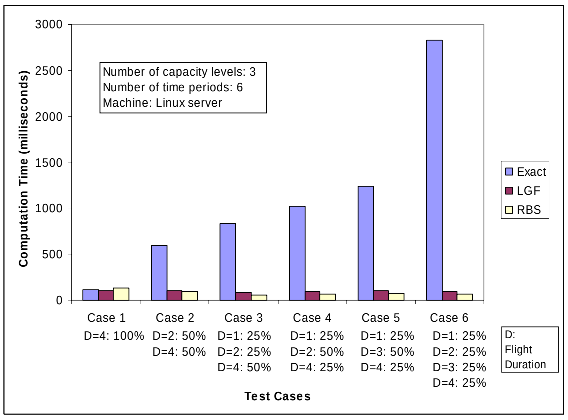

Figure 2. Computation Time with and without Priority Orderings of Flights

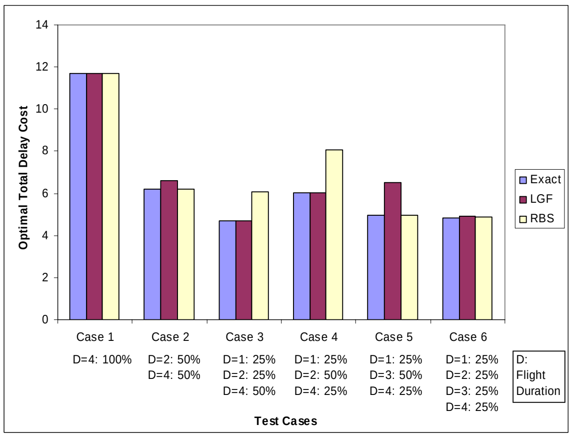

Figure 3. Optimal Total Delay Cost with and without Priority Ordering of Flights

In brief, we experimentally confirmed the effectiveness of priority ordering in reducing computation time as the heterogeneity of the flight durations increases. At the same time, we found that the loss in the quality of solution was never very large, and negligible for at least one priority scheme, in every one of our six test cases.

Heuristics

In Section 3, we proposed two heuristics, H1 and H2, to further reduce the computation required to implement priority ordering schemes. These heuristics are devised to find the optimal number of flights to hold when there is a prioritization scheme in effect. In this subsection, we investigate the effect of these heuristics when used in conjunction with LGF and RBS prioritization.

We continue the use of the six cases introduced in the previous section. Figure 4 shows the computation time results from using heuristics H1 and H2 with each of the two prioritization schemes. Consistently, heuristic H2 takes less time than heuristic H1, and heuristic H1 takes less time than the original priority model. Heuristic H1 reduces computation time 58-86% for LGF, 59-82% for RBS. Savings using H2 increase to 75-99% and 75-94% respectively. The effects of the heuristics are least under Case 1. This is probably because the cost-to-go function is convex in this case, and it takes longer, on average, for the heuristics to find an optimum when only one such optimum exists.

The total delay cost obtained from the heuristics is exhibited in Figure 5, which presents this cost as a percentage of the optimal cost computed by the exact algorithm. As expected, all the algorithms arrive at the same solution for Case 1. For other cases, the costs obtained from the heuristics exceed the optimal cost to varying degrees. Generally, however, the difference is slight. Moreover, the larger differences (in Cases 3, 4, and 5) are mostly caused by the adoption of the priority ordering scheme rather than the heuristics per se. The only appreciable penalty from using H1 is in RBS Case 4, in which the prioritization causes a 33% cost increase which H1 takes up to 40%. H2 does slightly worse in this case, yielding a cost 44% over the true optimum. H2 also performs relatively poorly in RBS Case 3, increasing the cost 49% over the original solution, as compared to the 29% increase with priority ordering without heuristic. On the other hand, across these six cases, the heuristics do not lead to any cost overage when used with LGF prioritization.

In sum, the heuristics reduce computation time by 40% to 99% in our numerical experiments. Between heuristics H1 and H2, H2 reduces the computation time more dramatically but is somewhat more likely to incur a cost penalty. In these test cases, the heuristics lead to the same optimal solution when the LGF priority rule is used, but sometimes lead to actions that are more costly when the RBS priority rule is used. Another pattern suggested, albeit far from conclusively, from these results, is that the cost penalties from heuristic use are highest in the cases where the penalties from using priority ordering are also highest. While these initial results are promising, more extensive testing and analysis is required before the heuristics can be recommended for real-world application.

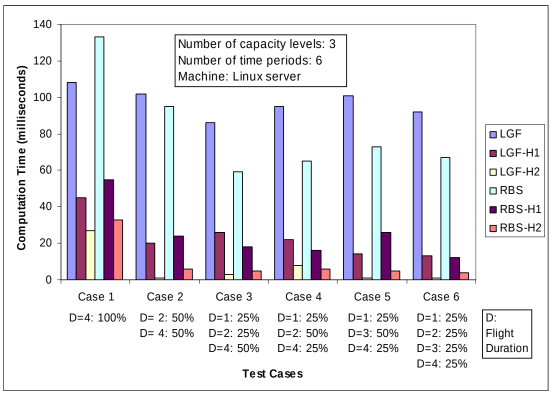

Figure 4. Computation Time with and without the Heuristic

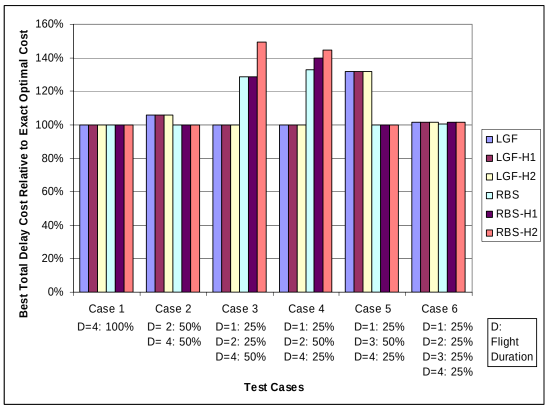

Figure 5. Impact of the Heuristics on the Best Total Delay Cost

Real-world Application

The six cases above are small scale problems formulated so that they cane be solved fairly quickly by all models, including the exact model. In this subsection, we investigate the computational feasibility of solving a real-world problem using our model and algorithms. For this investigation, we first identify a real-world problem as our target problem and then use our algorithm to determine the optimal policy for the SAGHP.

Our case is based on San Francisco International Airport (SFO) on the day of March 2, 2006. In general, SFO has a relatively high proportion of arrivals subject to ground holds, and March 2006 was a month in which ground holding at SFO was particularly common (Table 3). We chose March 2 as a representative day within that month, because it had the median proportion of ground-held flights (Table 4).

Month	Number of Arrivals	Arrivals with EDCT	% of Arrivals with EDCT
January	13484	1419	10.52%
February	11513	1556	13.52%
March	13102	2864	21.86%
April	12889	2520	19.55%
May	12872	824	6.40%

Table 3. Arrival Data at SFO for year 2006 from FAA ASPM Database

Date	Number of Arrivals	Number of Arrival with EDCT	% of Arrivals with EDCT
3/7/2006	427	8	1.87
3/6/2006	426	9	2.11
3/14/2006	423	9	2.13
3/9/2006	437	12	2.75
3/25/2006	395	24	6.08
3/22/2006	439	60	13.67
3/13/2006	429	60	13.99
3/15/2006	429	68	15.85
3/17/2006	442	85	19.23
3/21/2006	427	101	23.65
3/2/2006	427	115	26.93
3/27/2006	425	123	28.94
3/12/2006	413	120	29.06
3/20/2006	430	149	34.65
3/29/2006	430	158	36.74
3/10/2006	432	182	42.13
3/30/2006	425	183	43.06
3/5/2006	393	248	63.1
3/28/2006	408	264	64.71
3/31/2006	440	285	64.77
3/16/2006	414	293	70.77
3/24/2006	425	308	72.47

Table 4. Arrivals with EDCT in March, 2006 at SFO

From the airport data on March 2nd, we observe nearly continuous capacity shortage (arrival demand greater than AAR) from 8:45am to 1:15pm. In addition, most of the flights scheduled to arrive in this period have scheduled departure time from 7am to 12pm. Hence, for this target problem, we include flights with scheduled departure time from 7am to 12pm inbound to SFO. The planning horizon is set for seven hours in total (7am to 2pm). The airport data indicate that the arrival capacity at SFO on that day was around six to seven aircraft per quarter hour across the entire planning horizon. According to the flight schedule, 28 of the flights arriving in between 9am to 2pm originate from foreign airports. To account for the exemption of the international flights in the SAGHP, we simplify by reducing the arrival capacity level and assuming a constant capacity transition matrix throughout the planning horizon (Table 5) with initial arrival capacity level at 5. Though the matrix is a simplification, a 3-by-3 capacity transition matrix nevertheless reflects the real airport situation. The weather condition at an airport is categorized as VFR, MVFR (marginal VFR), or IFR based on the visibility and ceiling conditions. For the same runway configuration, the capacity level is a function of the weather condition. Therefore, it is reasonable to consider three possible capacity levels due to the three primary weather conditions. Excluding the international flights, there are 116 flights departing between 7am and 12pm. We consider these 116 flights in the target problem (Table 6).

from \ to	5	6	7
5	0.4	0.5	0.1
6	0.3	0.4	0.3
7	0.2	0.3	0.5

Table 5. Transition Matrix for the Target Problem

Departure Hour	Departure Quarter	Number of Flights	Flight Duration
7	1	6	3 5 8 15 17 17
7	2	6	5 6 7 7 11 5
7	3	8	4 6 6 8 9 11 16 21
7	4	8	4 5 5 5 5 8 8 18
8	1	6	3 5 17 17 17 17
8	2	6	5 5 5 6 9 26
8	3	2	12 25
8	4	3	7 7 25
9	1	6	2 3 4 5 5 6
9	2	10	4 6 6 6 7 7 8 11 20 23
9	3	5	2 3 16 17 17
9	4	3	5 6 6
10	1	12	4 4 4 5 6 7 7 7 8 8 18 20
10	2	2	6 6
10	3	5	5 6 6 10 21
10	4	2	2 4
11	1	8	5 5 5 6 6 7 18 18
11	2	4	8 11 25 25
11	3	8	4 4 5 6 6 7 13 24
11	4	6	3 5 5 6 8 8

Table 6. Scheduled Flights in the Target Problem

We solve the target problem with heuristic H2 on the Linux server and present the computational results in Table 7. The algorithm reached a slightly lower optimal cost but had a much longer computation time with RBS priority ordering. But even with RBS, the computation is completed in 20 seconds—well within the acceptable range for real-world application.

On the other hand, it took more than two hours for the algorithm with heuristic H1 to solve this target problem using LGF. This is not surprising since, as discussed above, the modest savings from using time heuristic H1 in a given time period grow exponentially with the number of time periods. For a realistic scale problem, this difference is sufficient to make H2 the only feasible heuristic for our computing resources.

	Priority Ordering	
	LGF	RBS
Number of Decision Nodes	156301	415228
Optimal Expected Total Delay Cost	26.43	25.87
Computation Time (milliseconds)	2324	19741

Table 7. Computational Results for the Target Problem

Conclusion

In this paper, we have investigated the use of sequential decision models to optimize ground holding decisions in the context of the single airport ground holding problem. We formulated the model using a dynamic program and solved it with a value iteration algorithm. As far as we know, this is the first scenario-free model that can provide a dynamic optimum for the SAGHP.

From complexity analysis, we recognized the need to explore computational strategies so as to manage the curse of dimensionality of dynamic programming. Firstly, we observed numerous overlapping subproblems in the dynamic program, and chose to memoize the top-down recursive algorithm instead of using the traditional bottom-up approach. Memoization cuts down computation time dramatically and provides the same optimal solution.

Next, we explored several strategies that can reduce computation time but may result in suboptimal solutions. From the algorithmic complexity, we observed that priority ordering among flights could reduce the computational load significantly. We proposed two priority ordering schemes, and found that they could help reduce the computation time greatly while providing solutions that were nearly optimal in our test cases. Additionally, the structural property of the cost-to-go function inspired us to devise heuristics for limited search. We found the heuristics helpful in further reducing computation time while only slightly reducing solution quality. Schematically, our exploration of computational strategies for this computationally intense problem can be summarized by Figure 6.

As a proof of concept, we demonstrated the computational feasibility of our model and algorithm for solving a typical real-world SAGHP. We found that the problem can be solved within an acceptable time for real-world application. In this real-world example, the model, in fact, planned for trillions of capacity scenarios. This quantity is clearly out of the capacity of the integer programming solvers today, and demonstrated an advantage of the scenario-free modeling approach. However, it is not clear by how much the computational strategies compromise the quality of solution. The comparison of the performance of the scenario-based model and the scenario-free model in a real-world setting is the topic of our on-going research.

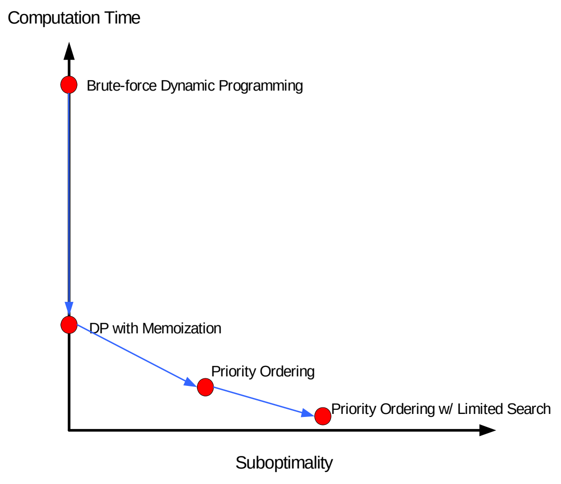
Computation Time
Brute-force Dynamic Programming
DP with Memoization
Priority Ordering
Priority Ordering w/ Limited Search
Suboptimality

Figure 6. Effect of Computational Strategies

References

[1] Richetta, Octavio, Amedeo Odoni, 1994, *Dynamic solution to the ground-holding problem in air traffic control*, Transportation Research Part A, **28** (3), pp. 167–185.

[2] Ball, Michael, Robert Hoffman, Amedeo Odoni, Ryan Rifkin, 2003, *A stochastic integer program with dual network structure and its application to the ground-holding problem*, Operations Research, **51**, pp. 167–171.

[3] Mukherjee, Avijit, Mark Hansen, 2005, *Dynamic stochastic optimization models for air traffic flow management with en route and airport capacity constraints*, Proceedings of the 6th USA/Europe Air Traffic Management R&D Seminar, Baltimore, MD.

[4] Liu, Pei-Chen, 2006, *Managing Uncertainty in the Single Airport Ground Holding Problem using Scenario-based and Scenario-free Approaches*, Ph.D. dissertation, University of California, Berkeley.

[5] Andreatta, Giovanni, Giorgio Romanin-Jacur, 1987, *Aircraft flow management under congestion*, Transportation Science, **21**(4), pp. 249–253.

[6] Terrab, Mostafa, Amedeo Odoni, 1993, *Strategic flow management for air traffic control*, Operations Research, **41**(1), pp. 138–152.

Key Words

Air Traffic Flow Optimization; Ground Delay Program; Ground Holding Problem; Dynamic Programming; Stochastic Optimization.

Biography

Pei-chen Barry Liu is currently a Ph.D. student at the University of California at Berkeley. He received a BS (1997, Civil Engineering) from National Taiwan University, MS (1999, Structural Engineering) from Stanford University, and MS (2006, Industrial Engineering and Operations Research) from UC Berkeley. He works at IBM on business intelligence applications as a software developer prior to and during his doctoral studies. His research interests include air traffic management, machine learning, and stochastic optimization.

Mark Hansen is a Professor of Civil and Environmental Engineering at UC Berkeley and co-director of the National Center of Excellence in Aviation Operations Research. He has undergraduate degrees in Physics and Philosophy from Yale University, and a Master's in City and Regional Planning and a Ph.D. in Engineering Science from UC Berkeley.