AIAA Guidance, Navigation, and Control Conference
10 - 13 August 2009, Chicago, Illinois

AIAA 2009-6010

A Comparison of Optimization Approaches for Nationwide Traffic Flow Management

Joseph Rios*
NASA Ames Research Center, Moffett Field, CA 94035

Jason Lohn†
Carnegie Mellon University, Silicon Valley Campus, Mountain View, CA 94035

Given thousands of flights in a capacity-limited airspace, finding optimal scheduling strategies that minimize delay costs is a computationally difficult task. In this paper, stochastic search techniques are applied to this fundamental Traffic Flow Management problem. Genetic algorithms and simulated annealing rely on searching the solution space in a manner much different than traditional optimization methods. These stochastic search techniques are compared to an integer linear programming model previously described in the literature. The runtime and aircraft schedules resulting from each model are analyzed. Results indicate that the integer programming model finds the optimal schedule faster than the next best technique, simulated annealing. In the presented experiments, solving the same problem with different simulated annealing solvers in parallel, a quality solution (within 5% of optimality) can be found in most cases. With roughly the same number of computations, simulated annealing reached a better solution than the genetic algorithm in 6 out of the 9 scenarios tested.

I. Introduction

Computational issues are an important factor in determining the feasibility of tools for solving large-scale traffic flow management problems in real-time. At any one time, there are thousands of flights in the air above the continental United States and thousands more scheduled to depart within the coming hours. Deciding which flights to delay and where to delay them in the presence of weather issues or excess demand is a computationally difficult task. The problem becomes even more demanding when including flight rerouting, connectivity, and equity.

When modeling the National Airspace System for Traffic Flow Management there are several parameters that might be considered and design choices to make. One of the major choices is whether to aggregate flights into flows[1–3] or consider each flight individually.[4–6] Other issues include which resources (airports, sectors, centers, etc.) to consider, what planning horizon to examine, what level of fidelity is required, and whether to consider some form of equity in a solution.[7,8] To perform a valid comparison of models and techniques, a set of parameters must be agreed upon.

In this study, individual flights are considered with respect to sector and airport capacity restrictions at a medium planning horizon (1-3 hours) with high fidelity (i.e., a solution in which the position of each flight is determined minute-by-minute). The fundamental question within this framework is as follows: how long should any flights be held in order to satisfy the capacity and scheduling restrictions of the system? Given costs of delaying flights, this particular problem has been solved optimally using an integer programming approach described by Bertsimas and Stock-Patterson.[4] In this paper, the performance of that model is compared with two "stochastic search" approaches in terms of runtime and solution quality. A genetic algorithm[9] approach solves the problem by creating "populations" of solutions which "evolve" over a series

*Aerospace Engineer, Automation Concepts Research Branch, Mail Stop 210-10, Joseph.L.Rios@nasa.gov. Member AIAA.
†Research Faculty, MS 23-11, NASA Research Park, Jason.Lohn@sv.cmu.edu.

of iterations. Genetic Algorithms have been applied in the domain of Traffic Flow Management,[10–12] but not in a way amenable to comparison with the integer programming approach since previous work solved more specific Traffic Flow Management problems versus global optimality problems as Bertsimas and Stock Patterson's model does. The same problems were solved with a simulated annealing approach,[13] wherein a single solution is slightly perturbed over many iterations and each perturbed solution is kept or discarded according to a "cooling schedule." To the authors' knowledge, simulated annealing has not been attempted on large-scale Traffic Flow Management problems. All approaches are generally well-understood and provide a varied set of benefits and liabilities. By comparing all of them in a unified manner, the applicability of each method with respect to this particular Traffic Flow Management problem will be objectively compared.

This paper is organized as follows. In Section II details on all three approaches are provided. Next in Section III a description of the data used in the experiments and how they were acquired is presented along with an overview of the necessary tools. Following the presentation of the data, Section V describes the experiments performed and their respective results. Finally, Section VI offers concluding remarks including potential future research directions.

II. Models

The integer programming approach is presented here followed by a full description of the genetic algorithm and simulated annealing approaches.

II.A. Binary Integer Programming Approach

This study uses the BSP model[4] to perform scheduling which minimizes delay costs. The model as presented by Bertsimas and Stock-Patterson is given here:

$$\text{Minimize:} \quad \sum_{f} \left[c_f^g g_f + c_f^a a_f \right]$$

$$\text{Subject to:} \quad \sum_{f:P(f,1)=k} (w_{ft}^k - w_{f,t-1}^k) \leq D_k(t), \qquad \forall\, k \in \text{Airports}, t \in \text{Time} \quad (1)$$

$$\sum_{f:P(f,\text{last})=k} (w_{ft}^k - w_{f,t-1}^k) \leq A_k(t), \qquad \forall\, k \in \text{Airports}, t \in \text{Time} \quad (2)$$

$$\sum_{f:P(f,i)=j, P(f,i+1)=j'} (w_{ft}^j - w_{ft}^{j'}) \leq S_j(t), \qquad \forall\, j \in \text{Sectors}, t \in \text{Time} \quad (3)$$

$$w_{f,t+\min(f,\,j)}^{j'} - w_{ft}^j \leq 0 \qquad \forall\, f \in \mathcal{F}, j = P(f,i), j' = P(f,i+1) \quad (4)$$

$$w_{ft}^j - w_{f,t-1}^j \geq 0 \qquad \forall\, f \in \mathcal{F}, j \in (f\text{'s flight path}) \quad (5)$$

$$\text{Decision variables } w_{ft}^j = \begin{cases} 1, & \text{if flight } f \text{ arrives at sector } j \text{ by time } t, \\ 0, & \text{otherwise.} \end{cases}$$

The air and ground delay (a_f and g_f, respectively) for each flight f are ultimately expressed in terms of the binary variables, w, through a substitution for a_f and g_f, which is omitted here. For each flight the model is able to decide if the flight needs to be held anywhere (and for how long) in order to satisfy the airport and sector capacity constraints presented in Constraints (1), (2), and (3). Within those constraints, $D_k(t)$, $A_k(t)$, and $S_k(t)$ represent the departure, arrival, and sector capacities at a given time t for a given resource k. Airports are denoted by k and sectors by j. Each of the capacities are a function of time, t. Each flight in the set of flights, $\mathcal{F}$, is described as an ordered list of distinct j's from a set of sectors, J, with earliest and latest feasible entry times for each of those sectors. Sectors in the flight path are denoted by $P(f, y)$ where f is the flight and y is the ordinal representing the sector or airport in the flight path. For ease of notation, $P(f, \text{last})$ is used to represent the last sector in f's path. The parameters c_f^g and c_f^a are the costs of holding flight f on the ground or in the air, respectively, for one unit of time.

The problem is also constrained by the physical/temporal limitations of the flights. Specifically, each flight spends at least the specified minimum amount of time in each of its sectors, $\min(f, j)$, as described by Constraints (4). Finally, a set of constraints enforces the temporal logic of the decision variables (Constraints (5)).

II.B. Genetic Algorithm Approach

Genetic algorithms are a type of generate-and-test search technique that are guided by principles of Darwinian evolution. Just as the genetic material of two living organisms can intermix to produce offspring that are better adapted to their environment, GAs expose their "genetic material," frequently strings of 1s and 0s (chromosomes), to the forces of artificial evolution: selection pressure, reproduction, and genetic mutation and recombination. GAs start with a pool of randomly-generated candidate solutions which are then tested and scored with respect to their performance, often termed "fitness evaluation". Solutions are then bred by probabilistically selecting high quality parents and recombining their genetic representations to produce offspring solutions. Offspring are typically subjected to a small amount of random mutation. After a pool of offspring is produced, this process iterates until a satisfactory solution is found or an iteration limit is reached. The details as they apply to this study are supplied below in Section II.B.2, but the overall idea will be to create lists of flights (candidate solutions) for scheduling on a first-come, first-served basis and then manipulating those lists through GA operations to find the best ordering of flights.

II.B.1. Parallel Genetic Algorithms

Parallelized versions of genetic algorithms are popular primarily for three interconnected reasons. First, the GA is an inherently parallel algorithm as the fitness evaluation of the individual solutions are completely independent. Second, typical GA applications are computationally intensive. Finally, powerful computing platforms, from multi-core workstations to computing clusters, are affordable and available. In addition, the low communication bandwidth required allows the use of inexpensive networking hardware such as standard office ethernet.

There are many styles used in implementing parallel GAs.[14] For the work described in this paper, a master-slave[9] architecture is used. Also called a "processor farm," this style automatically balances the computational load among processors within each generation. The master processor maintains the population, performs the genetic operations, and distributes the fitness evaluations to the nodes. Each node processor runs a loop consisting of two main functions: ask for a set of individuals to process, and then process those individuals as described below. A computing cluster consisting of 98 cpu cores was used to carry out the experiments described below. A simple shell script is used to launch the parallel GA run on however many processors the user desires to use.

II.B.2. Genetic Algorithm Details

The genetic algorithm manipulates ordered lists of flights (represented as aircraft IDs (ACIDs)) which are passed to a greedy scheduling algorithm (described below in Section II.B.3) to produce feasible schedules. An overview of this process is shown in figure (1). The GA reads in configuration files that setup the TFM problem to be optimized and parameter settings for the GA and computing cluster. GA parameters include population size, number of generations, crossover rate, and others. These parameters are tuned through experimentation. Cluster parameters include the names of the nodes available to the run, and how many cores per node are available to run on.

The greedy scheduler (GS) processes the GA-produced permutation and uses that ordering to contruct a valid schedule (i.e., one which satisfies all capacity constraints). Once a valid schedule is generated, the overall delay is calculated which is then passed back to the GA. The GS is the "fitness evaluation" for the GA. The GA subsequently uses overall delay costs as the fitness score to select permutations to breed into its next generation of permutations.

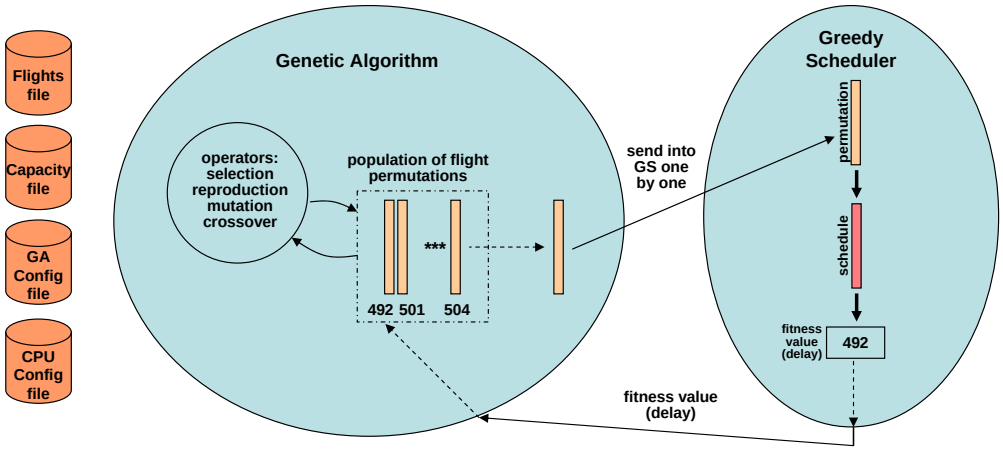


Figure 1. Genetic algorithm and greedy scheduler overview. The main interaction between the GA and GS is seen as loop where permutations generated by the GA are fed into the GS which sends delay values back to the GA. Input files (left) are accessed by both processes.

The GA uses four main operators to build and maintain the population of ACID permutations. The selection operator performs a biased selection of parents for breeding using a roulette wheel model. A roulette wheel model statistically biases selection towards fitter individuals. This selection method preserves diversity by occasionally selecting poor-performing individuals. Fitness scores are used to rank the individuals, the ranking determines the selection probability.

The reproduction operator determines how individuals are reproduced and placed into the population. A simple generational model is used whereby the entire population is replaced with each generation. There are two special cases to how new generations are produced. The first is that the two elitist (i.e., highest performance) individuals are copied into the new generation. The second case is that the two elitist individuals are always bred with each other and inserted into the new generation. Mutation is accomplished by probabilistically swapping a randomly-chosen pair of ACIDs. The crossover operator used is Syswerda's position-based crossover.[15] This operator works as follows:

1. Given two permutations, parent1 and parent2, randomly select a set of flight list positions as determined by the crossover rate (typically 70%).
2. For the selected positions, child1 receives the ACIDs of parent2, and child2 receives the ACIDs of parent1.
3. For the unselected positions, the ACIDs missing from child1 are filled in child1 based on the order they appear in parent1. The same is done for child2 using parent2's order.

Both the mutation and crossover operators retain the integrity of the permutation so that all individuals are valid permutations. Note that this does not guarantee that it will generate a valid schedule by the GS – indeed, in the early generations of the GA, it is common to see permutations that cannot be scheduled.

II.B.3. Greedy Scheduler

The greedy scheduler (GS) was based on an algorithm used in a previous computational study on TFM scheduling.[16] The GS is designed to satisfy three basic criteria: to be efficient, simple and realistic. It has the same delay-cost-minimization goal as the BP model, but it does not take into account the relationship between any of the flights. It schedules on a first-come, first-served basis by finding the first available departure time for each flight in turn that will not violate sector capacities when combined with previously scheduled flights. The GS requires an ordering of flights and capacity values for sectors and airports. The GS then takes each flight, in order, and performs the following steps:

1. Calculate the necessary departure time for the target arrival time and set equal to t.

2. Check each time slice starting from t in the flight path to verify that no capacity constraints are violated.

3. If no violations, commit the flight's departure time to t and decrement all appropriate sector and airport capacities during the time that the flight used the resource to designate accommodation of this flight. Else there is a violation, increment t by 1 and goto step 2.

This algorithm represents a naive, but fast, approach to scheduling. It roughly follows the same philosophy of a "ration by schedule" ground delay program wherein flights are assigned arrival slots based on their original schedule despite any other delays that might be imposed upon that flight.

II.C. Simulated Annealing Approach

The simulated annealing (SA) algorithm, like the genetic algorithm, is a generate-and-test search strategy. The main difference is that SA operates on a single candidate solution instead of a set of candidate solutions. In addition, there is no concept of recombining candidate solutions – SA relies solely on a mutation operator (or "perturbation" in the SA literature) to find higher-performing solutions. In this study, the same mutation operator (described in section II.B) is used for both algorithms.

Simulated annealing starts with a single randomly generated permutation of flights. This permutation (the parent) is mutated to produce a new permutation (a child) which, if it produces a better (more fit) schedule than the parent, will replace the parent. Less fit children can replace the parent with probability $p = e^{\frac{-\Delta F}{T}}$ where ΔF is how much less fit the child is. The temperature T starts at 100 and is multiplied by 0.92 every 1000 children. The same greedy scheduler described in Section II.B.3 that was used by the GA is also used by the simulated annealer to determine the fitness of a given permutation.

This is a basic SA implementation. There are some speed-up improvements presented in the literature[17,18] that could be included in future studies. There are methods available for parallelizing SA. For this implementation, parallelization will be achieved by running several SA instances of the same problem on different processors with different pseudo-random number seeds as will be described further in Section V.

III. Data

The Aircraft Situation Display to Industry[19] data for Thursday, August 24th, 2005 starting 9:15 AM EDT was used for all experiments. This was a benign day in the NAS in terms of weather, thus there were few flow controls implemented. This yields a cleaner data set from which to start. The time chosen represents the fairly dense mid-to-late morning traffic on the east coast, as well as the earlier morning rush on the west coast. All domestic flights were kept, with the exception of some "noisy" flights. After removing the international flights and the noisy flights, over 90% remained.

Any sector or airport that was used by any flight in the system was included in the data set. This resulted in 974 sectors being included along with 905 airports. Many of the constraints generated by these resources were not near their maximum capacity levels and could be simplified out of the problem formulation. For example, if only a few flights are using Moffett Federal Airfield, then its capacities could never be violated and, thus, could be removed from the formulation. There was no distinction made between low, high or superhigh en route sectors.

To extract the flight data from the ASDI file, the Future ATM Concepts Evaluation Tool (FACET)[20] was used. FACET is capable of many functions, but for this study, its capabilities of playing back and simulating historical data and recording statistics were most relevant. The default sector capacities (known as Monitor Alert Parameters (MAP)) and airport capacities as understood by FACET were used for the experiments. If ever an airport was without explicit capacities in FACET, a default value of 10 was used for arrival and departure capacity per 15 minutes.

IV. Tools

Due to the various algorithms being tested and the hardware and software that was available, there are different setups for the integer programming approach and the stochastic search approaches.

IV.A. Integer Programming Tools

The CPLEX solver from IBM-ILOG was used to solve the integer programming model. This software was run on an 8-core Xeon processor running at 2.66 GHz with 32 GB of RAM. IBM-ILOG does offer a parallel version of CPLEX, but that tool was not used in this study and it is not clear how much improvement would be gained from its use. For this study, the model was implemented in the CPLEX LP format. This allows for more efficient runtimes[21] than might be achieved with a modeling language like AMPL or GAMS, but is generally more difficult to implement and maintain.

IV.B. Computing Cluster

For the stochastic search methods, the computing hardware used was a Beowulf Linux cluster consisting of 35 computing nodes: 14 having four AMD Opteron cores, and 21 having two AMD Athlon cores. Thus a total of 98 CPU cores were available. The cores ranged from 1.8-2.1 GHz and were older technology, mainly purchased four to six years ago. For a more modern comparison, a few runs for simulated annealing were completed on a cloud-computing platform with Xeon processors.

V. Experiments and Results

The general framework for comparing the optimization approaches is illustrated in figure 2. Each "problem solver" (e.g. the GA or integer program) is provided the same input files and they each provide the same solution format allowing for clean comparisons. In the following subsections, results from the individual solvers are provided. The final subsection completely describes the combined results.

For all experiments a realistic[22] cost ratio of 2:1 for air to ground holding is used. Three planning horizons are used: 60, 120, and 180 minutes. For each of those planning horizons, the capacity of the en route sectors was varied from 100% of nominal MAP to 90% of nominal MAP and then to 80% of nominal MAP. This method of reducing capacities is not meant to mimic actual disruptions to the NAS, rather it stresses the solvers by forcing them to work on more constrained problem instances as they would need to do during an actual disruption due to, say, weather.

For each scenario, a set of 13 GA runs were executed using the same parameters except with different pseudo-random number generator seeds. The GA ran for 200 generations, with a population size of 1000 individuals, yielding $200,000$ greedy schedule evaluations. The crossover rate was 70% with no mutation. In order to provide an equitable comparison between the GA and SA runs, the same number of runs and the same number of greedy schedule evaluations,13 runs consisting $200,000$ schedule evaluations, were used. In SA, each evaluation is also called an iteration. Like the GA, each of the 13 runs was identical except for the seed for the pseudo-random number generator. While the GA uses multiple cores, the SA uses a single core. For each scenario, the integer program approach was run only once as there is little variation in runtime and no variation in solution quality. The number of runs was simply limited by time and computing availability. The other parameters were determined through experimental tuning.

For the GA, experiments were setup in such a way as to sample across independent GA population runs. Due to the stochastic nature of the GA, run sampling is required in order to collect meaningful performance statistics. Due to computing time limits, only a relatively small number of runs were sampled. As an example, an experiment might consist of 30 runs executing simultaneously using the same parameters, with the exception of having different pseudo-random number streams. Statistical sampling also aids in tuning the GA parameters, although in this study, tuning was held to a minimum due to computing time constraints.

Run times are given in wall clock time, as opposed to CPU time (the time a GA spends running on a CPU).

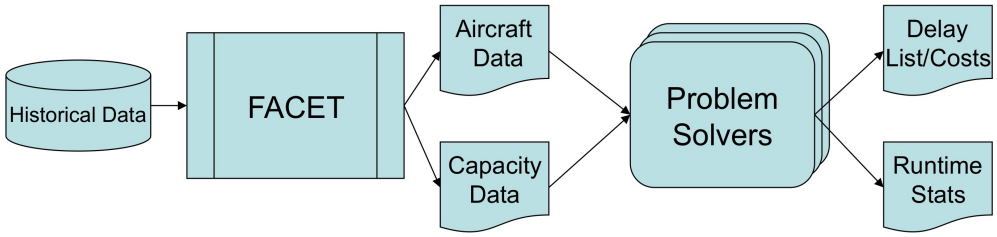


Figure 2. General experimental flow.

V.A. Optimal Results

The BSP model was used to discover optimal solutions for each scenario. The solutions provided by this solver provide a soft lower-bound on the solutions that any other solver is able to obtain. The reason this is a soft lower-bound due to a "maximum lateness" parameter that is necessary in the BSP model. For this study, the maximum lateness for each flight was set to 20 minutes. It is possible that lower delay costs can be found if more delay is allowed per flight. The optimal results and associated runtimes are presented in table 1.

Table 1. Results provided by the Bertsimas-Stock Patterson model.

Scenario (Plan Horizon/ % Capacity)	Delay Costs	Wall Clock Runtime (s)
60min/100	458	48
60min/90	682	85
60min/80	1127	148
120min/100	987	350
120min/90	1410	691
120min/80	2399	1130
180min/100	1270	923
180min/90	1943	2086
180min/80	3748	15322

V.B. Stochastic Search Results

Figure 3 shows individual run performance for a set of runs for two SA scenarios. As can be seen, the runs converge fairly quickly, with most runs being within a few percent of their converged value by 100,000 iterations, as shown in table 2.

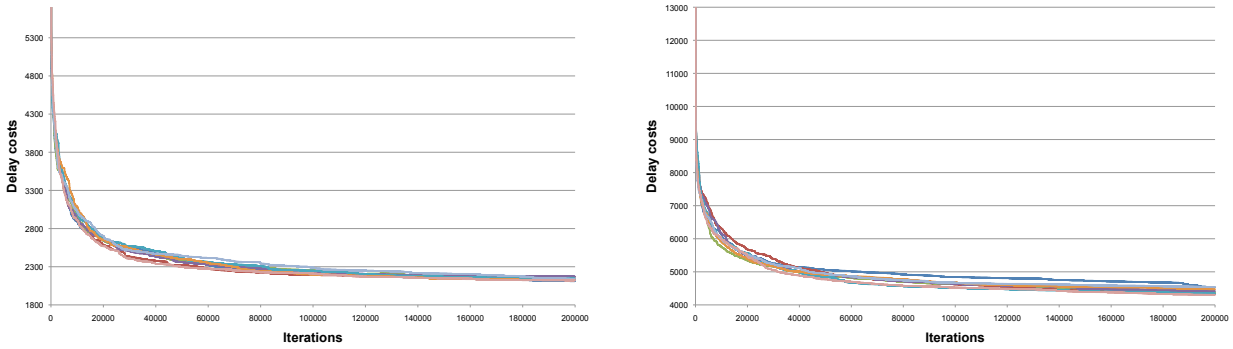
For the same number of fitness evaluations (as described at the beginning of Section V), the GA wasn't able to converge as convincingly. Figure 4 illustrates the same two scenarios as presented in figure 3. The GA converges more slowly and doesn't seem to complete convergence after its 200th iteration. This may be evidence that mutation is the best operator that was tested as the SA uses this operator exclusively and converged more quickly (in terms of fitness evaluations) than the GA.

V.C. Combined Results

The comparative results are now presented. The key measurement in this study is the delay cost found by each method. Recall that the delay cost is a weighted sum of delay minutes where each minute of air holding is equivalent to two minutes of ground holding. Figure 5 shows the comparative delay cost results. For the

Table 2. Simulated annealing achieves near its best solution after half the allowed iterations. Increase over LP solution provided for reference.

	60min 100%	60min 90%	60min 80%	120min 100%	120min 90%	120min 80%	180min 100%	180min 90%	180min 80%
Result at iteration 100,000	459	699	1156	1019	1515	2744	1240	2188	4512
Increase over LP result	0.2%	2.5%	2.6%	3.2%	7.4%	14.4%	-2.4%	12.6%	20.4%
Result at iteration 200,000	458	696	1151	994	1481	2659	1216	2072	4304
Increase over LP result	0.0%	2.0%	2.1%	0.7%	5.0%	10.8%	-4.3%	6.6%	14.8%
Difference between iterations 100,000 and 200,000	0.2%	0.4%	0.4%	3.4%	2.2%	3.1%	1.9%	5.3%	4.6%



(a) 180-minute, 90% capacity runs

(b) 180-minute, 80% capacity runs

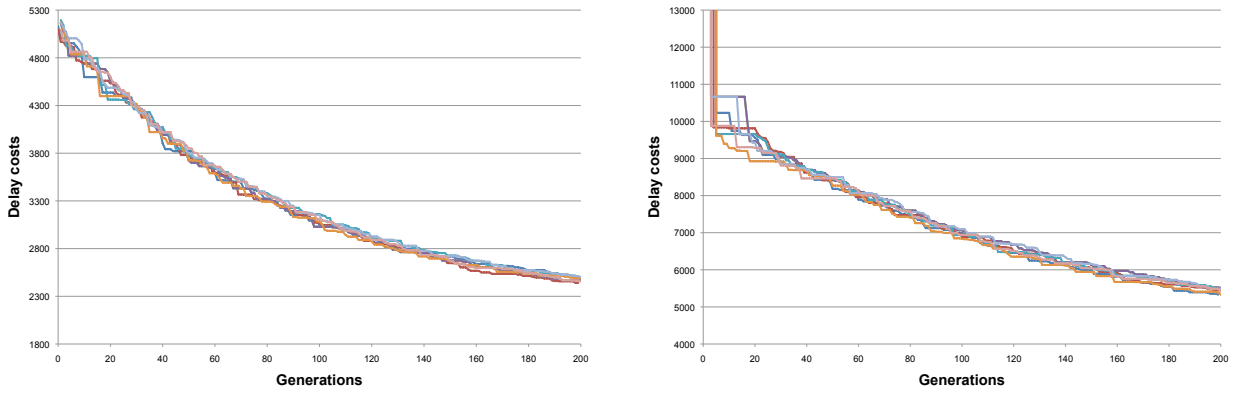
Figure 3. Examples of convergence for simulated annealing.

LP, this is the result from a single run of the CPLEX solver. For the SA and GA methods, the best solution found over the course of all parallel runs is shown.

An important illustrative point is that the SA actually found a better solution for the 180 minute-100% capacity scenario than the integer programming method. This has to do with the "maximum lateness" parameter discussed in Section V.A. While the integer program is limited by how long it can delay a given flight, the SA and the GA have no such restriction. The lateness limitation in the integer programming model is a result of needing to limit the number of decision variables.

An important secondary measure is the relative runtimes to achieve each algorithm's best solution. Figure 6 shows that the LP solver is still much faster than the stochastic search methods. There are many factors influencing the difference in runtimes (hardware differences, parameter choices, fitness functions, etc), but it is clear that, in general, the LP software is able to produce solutions much more quickly than the GA and SA implementations presented in this paper. The one exception occurs in the 180min/80% capacity scenario. The GA runtime is lower than the LP, though referring back to figure 5, the solution quality is better with the LP. Also, recall that SA didn't improve much after roughly 50% of its runtime was complete, offering the potential to cut it runtimes in half for a small increase in delay costs (see table 2).

Despite being slower, the stochastic search methods exhibit an important quality. The runtime for the LP varies greatly with the difficulty of the scenario being solved (especially on the 180 minute scenarios), whereas the GA and SA were indifferent to the difficulty of the problem. This may be an important criterion for certain applications where a solution (optimal or not) must be provided in a guaranteed timeframe, though with the current implementation, that timeframe would likely have to be prohibitively long. The computation time's indifference to the difficulty of the scenario is related to the fact that each fitness evaluation takes roughly the same amount of time regardless of how difficult the scenario might be.



(a) 180-minute, 90% capacity runs

(b) 180-minute, 80% capacity runs

Figure 4. Examples of convergence for the genetic algorithm.

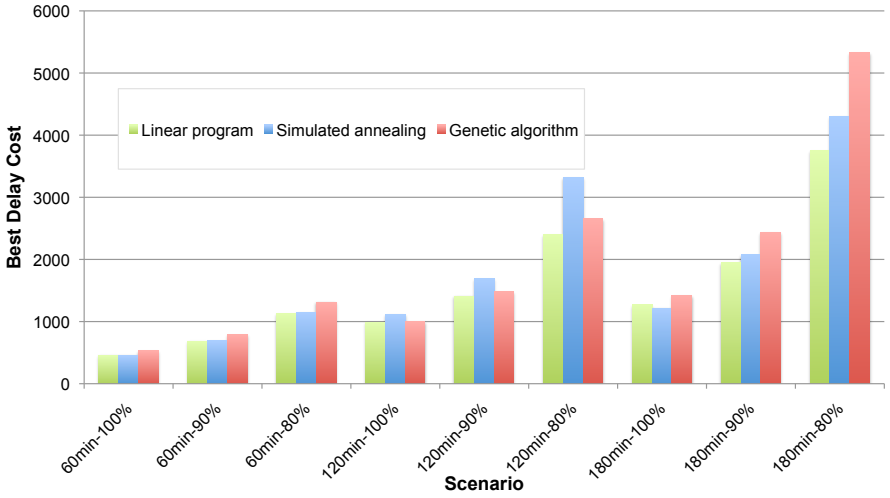


Figure 5. Comparison of the best delay cost found by each method.

Recall that the computing platform for the linear programming method is more modern than that for the stochastic search methods (see Section IV). To get a sense for the performance of the stochastic search methods on more modern hardware, several runs of the 60min-100% capacity and 180min-80% capacity scenarios were sent to a cloud computing service that offered Xeon processors comparable to that which was used for the linear programming approach. The 60min-100% runs completed in parallel and took, on average, 2781 seconds versus the 9881 seconds required on the experiment hardware while the 180min-80% scenarios completed in 5877 seconds versus 20965 seconds on the experiment hardware. This is a 72% reduction in runtime for both sets of scenarios. While this 72% reduction in runtime cannot necessarily be extrapolated to all scenarios, it does illustrate that large improvements in runtime are possible for the stochastic search methods by simply using better hardware. It is also worth noting that the reduction in time for the 180min-80% scenario offers solutions 60% faster than the linear programming approach for the same scenario. This implies that for difficult scenarios, the SA approach may be more operationally useful than the linear programming approach.

VI. Concluding Remarks

This study presented three techniques applied to a challenging TFM problem. Integer programming provided the best results in terms of both solution quality and runtime and is, thus, likely the tool of choice. However, the stochastic search methods of simulated annealing and genetic algorithms exhibit several positive qualities.

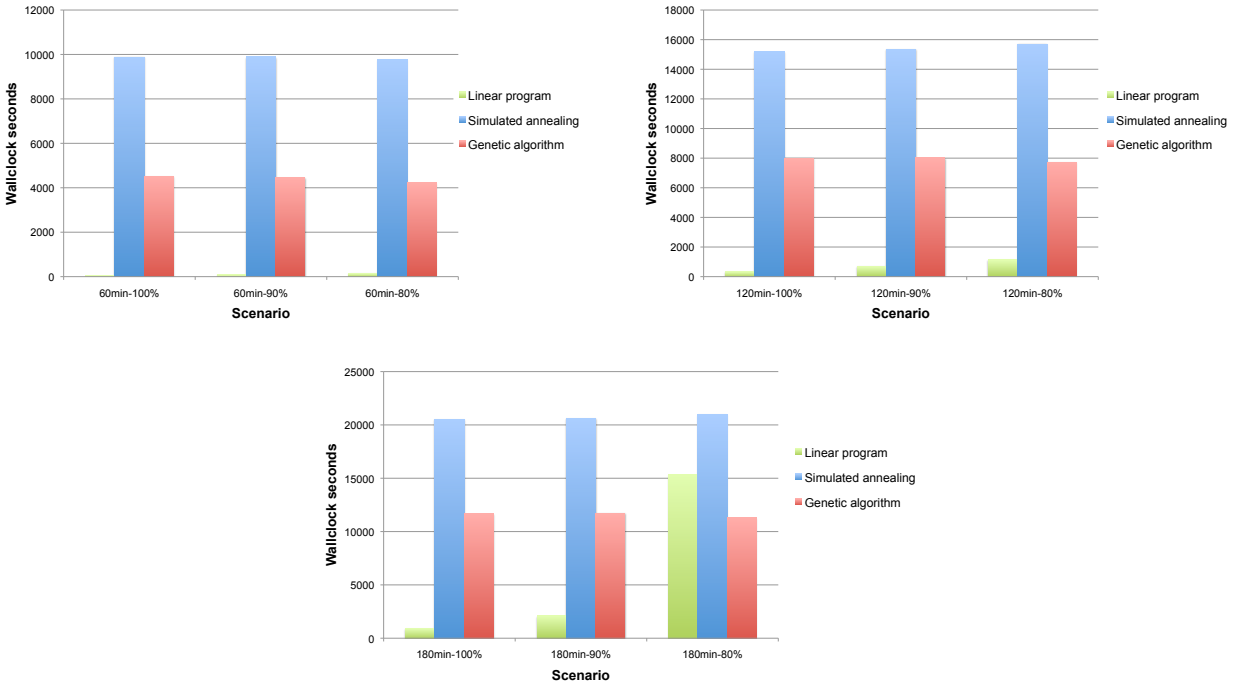


Figure 6. Runtime comparisons for all tools over all scenarios. Different graphs used due to different vertical axes.

Namely, these approaches are indifferent in terms of runtime to the difficulty of any given scenario, whereas the integer programming method is quite sensitive to the difficulty. This feature must be tempered with the fact that the runtimes for the stochastic methods are quite long no matter the difficulty of the scenario. Another positive feature of the the stochastic search methods is potential for parallelization as demonstrated with the implementations presented here. Being able to take advantage of several computers and/or cores to solve a single problem is an important aspect of scalability for a problem solving method.

Now that the stochastic search methods have been tested directly against a well-studied integer programming approach, several new questions can now be posed regarding the utility of such methods. Specifically, future TFM decision support tools will likely need to operate in an "online" manner wherein the tools are continually solving, and the current best solution may be pulled from the tool at any time. It seems that stochastic search methods would be more amenable to such a system than an integer programming approach. In addition, once research on TFM problems drifts away from deterministic data, models can become more cumbersome. There is hope that stochastic search methods will be better equipped to handle uncertainty than traditional optimization approaches. Linear programming approaches, like the one used here, are by definition limited to linear objective functions. The stochastic search methods can more easily accommodate more complicated, non-linear objectives due to the way in which fitness is evaluated.

References

[1]Sridhar, B., Soni, T., Sheth, K., and Chatterji, G., "An Aggregate Flow Model for Air Traffic Management," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island, August 2004.

[2]Menon, P., Sweridul, G., and Bilimoria, K., "New Approach for Modeling, Analysis, and Control of Air Traffic Flow," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 5, 2004, pp. 737–744.

[3]Bayen, A. M., Raffard, R. L., and Tomlin, C. J., "Adjoint-based control of a new Eulerian network model of air traffic flow," *IEEE Transactions on Control Systems Technology*, Vol. 14, No. 5, September 2006, pp. 804–818.

[4]Bertsimas, D. and Patterson, S. S., "The Air traffic Flow Management Problem with Enroute Capacities," *Operations Research*, Vol. 46, No. 3, May-June 1998, pp. 406–422.

[5]Bayen, A. M., Grieder, P., Meyer, G., and Tomlin, C. J., "Lagrangian Delay Predictive Model for Sector-Based Air Traffic Flow," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 5, 2005, pp. 1015–1026.

[6]Lulli, G. and Odoni, A. R., "The European Air Traffic Flow Management Problem," *Transportation Science*, Vol. 41, No. 4,

November 2007, pp. 431–443.

[7]Ball, M. O. and Lulli, G., "Ground Delay Programs: Optimizing Over the Included Flight Set Based on Distance," *Air Traffic Control Quarterly*, Vol. 12, 2004, pp. 1–25.

[8]Vossen, T., Ball, M. O., and Hoffman, R., "A General Approach to Equity in Traffic Flow Management and its Application to Mitigating Exemption Bias in Ground Delay Programs," *Air Traffic Control Quarterly*, Vol. 11, 2003, pp. 277–292.

[9]Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Maching Learning*, Addison-Wesley, Reading, Mass., 1989.

[10]Oussedik, S. and Delahaye, D., "Reduction of Air Traffic Congestion by Genetic Algorithms," *Lecture Notes in Computer Science*, Vol. 1498, 1998, pp. 855–864.

[11]Sood, N., Mulgund, S., Wanke, C., and Greenbaum, D., "Multi-Objective Genetic Algorithm for Solving Airspace Congestion Problems," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA, August 2007.

[12]Tandale, M. and Menon, P., "Genetic Algorithm Based Ground Delay Program Computations for Sector Density Control," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA, August 2007.

[13]Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, 1983, pp. 671–680.

[14]Cantu-Paz, E., "A Survey of Parallel Genetic Algorithms," *Calclateurs Paralleles*, Vol. 10, No. 2, 1998.

[15]Syswerda, G. and Palmucci, J., "The Application of Genetic Algorithms to Resource Scheduling," *Proc. of the Fourth International Conference on Genetic Algorithms*, edited by R. Belew and L. Booker, San Diego, CA, 1991, pp. 502–508.

[16]Rios, J. and Ross, K., "Delay Optimization for Airspace Capacity Management with Runtime and Equity Considerations," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Hilton Head, South Carolina, August 2007.

[17]Szu, H., "Fast Simulated Annealing," *American Institute of Physics Conference*, Vol. 151, August 1986, pp. 420–425.

[18]Ingber, L., "Very fast simulated re-annealing," *Mathematical and Computer Modelling*, Vol. 12, 1989, pp. 967–973.

[19]"Aircraft Situation Display To Industry: Functional Description and Interface Control Document," Tech. Rep. ASDI-FD-001, Volpe National Transportation Center, U.S. Department of Transportation, June 2005.

[20]Bilimoria, K., Sridhar, B., Chatterji, G., Sheth, K., and Grabbe, S., "FACET: Future ATM Concepts Evaluation Tool," *ATM2000*, Napoli, Italy, June 2000.

[21]Rios, J. and Ross, K., "Massively Parallel Dantzig-Wolfe Decomposition Applied to Traffic Flow Scheduling," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Chicago, IL, August 2009.

[22]DeArmon, J., Hoffman, J., Holden, T., Mayo, J., Solomos, G., Kuzminski, P., and Chambliss, A., "An Estimation of the Benefits of Air Traffic Flow Management," *AIAA Aviation Technology, Integration and Operations Conference*, Anchorage, Alaska, September 2008.