Cost Minimization Through Flight Scheduling
Category A

New Mexico High School
Supercomputing Challenge
Final Report
April 4, 2001

027
Albuquerque Academy

Team Members
Lizzie Brown
Ryan Davies
Kurt Diegert
Sarah Oman
Tom Widland

Teacher
Jim Mims

Project Mentor
Jim Mims

Table of Contents

EXECUTIVE SUMMARY

The hub and spoke system of flight scheduling, in which passengers are routed out of their way through certain large hubs, was first implemented in the 1980s to increase airlines' profit margins. Since then, congested hubs have become a burden on air traffic controllers and an inconvenience to passengers. We suspect that, given increasing numbers of travelers and increasing airline size, the hub and spoke system, in addition to being inconvenient, is no longer the most profitable one. Our project tests this hypothesis.

To do this, we designed our project to generate an efficient system of flights between the 56 largest metropolitan areas while accommodating all passengers who want to travel and never forcing them to make more than one connection. In order to make our project as thorough as possible, we experimented with different algorithms and then observed which method best created the least expensive flight plans.

The most simplistic of the algorithms is hill-climbing, in which the program starts with a randomly generated flight plan, changes parts of it at random, and then tests whether the change improved the plan. If so, it retains the changes; otherwise, the old plan is kept. Another approach, linear programming, required too much processing power because of the enormous number of constraints that had to be processed.

Our most successful approach involves a heuristic that allows us to apply our human insight to computational methods. The heuristic builds flight plans one city at a time while trying to schedule flights as inexpensively as possible. The input to this heuristic is a permutation of the cities, which we generated with hill-climbing and also with a genetic algorithm. The hill-climbing approach produced flight plans that were 1% less expensive than a system in which every passenger flies on a direct flight. The genetic algorithm begins with a population of random permutations and determines which of them created the best flight plans. It then "breeds" the plans, combining the best permutations to generate successive generations of even better ones. The best flight plans this approach generated were 2.3% less expensive than a system of direct flights, which translates to savings of over $100 million per day.

The flight plans our program generated did not rely as intensively on hubs as today's flight plans. We found that travelers making connections accounted for no more than 13% of the traffic at any airport, far less than the percentages in today's major hubs. Based on these results, we expect to see a gradual trend among airlines towards a less hub-based system.

INTRODUCTION

Many changes have been taking place in the airline industry recently. Each year, more and more people travel by plane, and the number of companies controlling US domestic flights diminishes. The recent mergers of several major airlines, including TWA with American and United Airlines with USAir produced larger individual airlines, with more resources and more customers.

In the past, airlines depended heavily on hubs in a few major cities. However, the growing congestion at major hub airports during the 1980s created opportunities for alternative services such as direct flights.[1] Routing flights between smaller cities through Chicago, Atlanta, or Dallas used to be a profitable system for airlines in the 1980s when fewer people traveled. Today, on the other hand, many more people travel on a regular basis, and individual airlines serve more customers. The hub system is no longer necessarily the most efficient method of flight scheduling. Huge increases in landings and takeoffs at hub airports also put enormous stress on the air traffic controllers.

We question the optimality of the hub system. If each airport shared some of the burden of directing connecting flights, air traffic could be more easily coordinated, reducing the amount of delayed flights due to lack of free runways. With more possible cities to route flights through, connections could be made through more optimally located cities. This would save fuel because the airplanes are not flying extra unnecessary miles, and save time for the passengers because they would not have to fly far out of their way. A system without hubs may be cheaper for the airline to implement. Although we cannot guarantee that lower costs to the airline would immediately translate to lower ticket prices, in the deregulated industry competition could eventually drive prices down for the consumer.

In addition to these advantages for consumers, many politicians have expressed concern over the effects on consumer service of the near-monopolies possessed by major airlines at certain hub airports. If airlines did not control massive portions of certain airports, it would be more difficult for one airline to dictate over others. Therefore the effects of mergers and monopolies would be mitigated substantially. USAir and United together control 91% of all air traffic in and out of Charlotte, NC. They control 89% at Pittsburgh and 73% at both Philadelphia and Denver. Senator John McCain sponsored legislation that would have lowered the control that these airlines have over air traffic at each individual airport. The kind of control that USAir and United Airlines exert in Denver or Charlotte virtually eliminates competition from smaller companies and leaves passengers at the mercy of the policies and practices of very few corporations. Although Congress may pass legislation that would change this situation, change through legal avenues might take a long time and enforcement wouldn’t be guaranteed to be effective. Change through profit motivation, on the other hand, would probably be substantially faster. A system with less dependence on large hubs would allow for more competition from smaller airlines because the larger would be less likely to control substantial portions of certain airports, allowing more room for smaller competitors.

Southwest Airlines, at the dawn of deregulation, shifted away from the system of congested hubs towards a “point-to-point.” The success of this move is evident in that in the 1990s Southwest grew to be among the top 10 ranking airlines in the United States. Southwest’s low prices also stimulate air travel. For example, in 1996, before Southwest's arrival, daily passenger traffic to Providence was 1,471. One year later, with Southwest having cut the average fare from $291 to $137, the daily passenger count had increased to 5,100.

Based on the success of Southwest Airlines and the other disadvantages of hubs outlined above , we projected that a move away from a hub-based system of scheduling would benefit airlines and consumers.

HYPOTHESIS

Our hypothesis was that, with increased passenger volume and resources, airlines would find it more profitable to rely less on a few hubs. The alternatives would be to route flights through connecting cities that are more convenient, or even route many flight directly. For instance, there might be enough passengers headed from Phoenix to Washington, DC to not have to route them through Atlanta before continuing on to Washington; they could either be routed directly, or through a more convenient city. We hope to show that a system of fewer major hubs with a maximum of one stop between any two cities could save money for the airlines. To do this, we wrote a program in C++ that generates a system of flights that an airline might schedule for a given day. We experimented with various algorithms by incorporating them into the program and observing which one performed the best, that is, which one is able to generate the least expensive flight plan. Of the algorithms we tested, we predicted that genetic

[1] Poole Jr., Robert W., and Butler, Viggo. “Airline Deregulation: The Unfinished Revolution”. April 4, 2001. <

algorithms with heuristics would be the most successful. The results of our program are most directly applicable to the airlines as they try to schedule flights, but they could have other indirect effects on separate aspects of air travel.

In short, our program is a response to concern over the future of air travel in the US. We tested several methods for finding the most efficient system and we suspected that our results would show that the airline industry should make significant changes in the way it schedules flights. The results favored a system that is beneficial both to the consumer and to the airlines.

DESCRIPTION

In order to solve our problem, we first had to prepare some preliminary research. We gathered and projected the input data necessary to run the program. In order to make the problem solvable, we had to make a number of assumptions to limit its scope. Even with these limitations, the problem is still very realistic and applicable to the current real-world problem of flight scheduling. We then wrote a core program that sets up the data structures necessary to generate a system of flights for an airline. Our program consists of over 1500 lines of code written in C++. At first we implemented certain data structures that are common to all of our algorithms, including the matrices that stored the cost of scheduling a flight between any pair of cities and the number of people wanting to fly between the pairs of cities. Next, we included different individual algorithms into the program to generate better results. By comparing the results we determined which algorithm was the most effective. We judged the efficiency of an algorithm based on how cheaply it could accommodate all of the passengers. We experimented with:

- hill-climbing
- hill-climbing with heuristics
- linear/integer programming
- genetic algorithm with heuristics

Once the strongest algorithms had produced several flight plans, we analyzed how much the plans relied on hubs. To do this, we created a Hub Index and compared the values to ascertain the presence of hubs or the lack thereof. From

http://www.cato.org/pubs/regulation/regv22n1/airline.pdf/>.

the results we drew conclusions about the nature of airline flight scheduling and the types of algorithms that are the most useful in solving this and similar problems.

Input Data Collection

Running the various algorithms requires the input of passenger and cost data, as well as geographic coordinates and populations of the 56 largest metropolitan areas in the U.S. The program needs to know the number of passengers traveling between each pair of cities in order to accommodate all of them. We originally believed that we could collect information regarding the numbers of passengers from airline companies. Unfortunately, the airlines were not forthcoming, and accumulating the more than 3,000 necessary data points proved unfeasible. Because of this we wrote a program that would generate the data based on a few numbers that we obtained from Southwest Airlines. The program, written in C++, calculates the number of travelers between a given pair of cities based on the population and distance between those two cities. Due to the simplicity of the function with which we calculated how many passengers would want to fly between a given pair of cities, the data is symmetrical. Thus the same number of people would want to fly from city A to city B as from City B to City A. We derived the following formula to calculate the distance d between cities based on their longitude A and latitude O, where the subscripts D and A represent departure and arrival cities, respectively:

$$d = \cos^{-1}\left[\sin(A_D)*\sin(A_A)+\cos(A_D)*\cos(A_A)*\cos(O_D - O_A)\right]*6250$$

For the cost data, we used this formula and then divided by the cruise speed of a 767-400ER to find the time of a flight so that we could obtain values for total fuel cost based on the fact that fuel costs about $3000 per hour. We calculated the total cost as a first order function of distance. The costs of fuel and flight crew labor vary proportionally to distance and other costs such as landing fees and labor of landing crew are added as constants. We made more assumptions when generating the cost data in order to simplify the process. For instance, we assume that every airport has equal landing fees, which, of course, is not accurate. Pilots are paid more for overtime. The cost of passenger food may vary according to length of flight and time of day. Theoretical considerations aside, the roughly ten actual data points we gathered from airlines were consistent when compared to our generated values. Furthermore, the absolute accuracy of our data is not entirely necessary to ensure that the program works well. If the program is reasonably successful finding cheap flight schedules with a given data set, it will probably be reasonably successful finding cheap flight schedules for similar data sets.

Assumptions

We figured some costs, such as like labor and fuel, on an hourly basis, and presumed other costs to be constants. In short, the cost is determined by a first-order function of the distance between cities. We utilized only one type of airplane, because multiple types would have added an extra, seemingly unnecessary dimension to our simulation. We based our estimations on a 767-400ER, which has a capacity of 235 passengers and a general cruise speed of 540 mph; our airplanes have a capacity of 200. These simplifications freed us from concern with differing fuel costs and passenger capacities.

The program is limited to scheduling only domestic, national-scale flights. Between the 56 cities, a passenger would never have to make more than one connecting flight in our system. As our model only covers a single day, we didn't take into account the location of the airplanes after they've been flown from one city to another. We also do not take into account the number of airplanes owned by the airline.

We feel that these assumptions are reasonable, as they expedite our task without compromising the integrity of our results in any major way.

Hill-climbing

The first method for generating a flight plan that we implemented was hill climbing. The general idea behind this method is that if a blind man were standing at the bottom of a hill, a slow but dependable method of increasing his altitude would be to take a step in a random direction and if that step put him at a higher elevation than his initial elevation he should stay there. If he ended up at a lower elevation he would step back to his initial location and continue this process until no direction led to a higher point. While easy to code, this method has two distinct disadvantages; first, it is inefficient, and second, it offers no guarantee that the maximum it finds is the global maximum, rather than a local maximum. A local maximum in the program would be like the blind man reaching a small peak that is not as high as the highest point in the hill.

One way to address this concern is to implement a technique called "simulated annealing." Annealing is a method of casting metals wherein the molten metal is heated up in small amounts many times as it slowly cools. This heating allows the molecules of the metal to align themselves in a stronger arrangement. In the hill-climbing version of simulated annealing, when the blind man reaches a peak, he commits his position to memory. We then move him randomly to another, lower location on the hill. From his new position he then begins to hill-climb in the manner previously described until he finds another peak.

Hill climbing, in the context of our project, begins with a randomly generated flight plan. The algorithm then randomly selects five city pairs and reassigns the connecting cities of all passengers traveling between those pairs of cities. This is the equivalent of a step in the blind man analogy. If the cost of the modified flight plan is higher than the initial flight plan, the algorithm will retain the original flight plan and repeat the process. On the other hand, if the cost of the modified flight plan is lower than that of the original, we begin the process from the new starting point. If the hill-climb takes too many steps without finding a cheaper flight plan, a greater number of city pairs is modified than in a standard step, without the option of returning to the previous plan. This constitutes annealing. Unfortunately, hill-climbing, even with simulated annealing, turned out to be a rather poor approach to so complex a problem and the most efficient flight plans it generated were not as cost-effective as simply putting all passengers on direct flights.

Linear programming

Two other related methods for solving our problem are linear programming and integer programming. Linear/integer programming reduces our problem to the problem of minimizing or maximizing a certain function subject to a large number of linear constraint equations. Integer programming takes the further step of constraining the solutions to be integers. We attempted to minimize the cost of our flight plan subject to constraints that specified that the flight plan accommodate all passengers. Linear programming has been used effectively to solve problems such as facilities placement and labor scheduling, but flight scheduling for 56 cities, which generated over 100,000 constraint equations, proved to be too difficult for the technique. Linear programming requires a great deal of memory to keep track of the enormous number of variables. We attempted to generate results using LP-Optimizer by Markus Weidenauer, but insufficient memory prevented us from executing the program satisfactorily.

Genetic Algorithm with Heuristics

A more sophisticated method for generating a flight plan is a combination of genetic algorithms and heuristics. Genetic algorithms begin with a population of possible solutions to a problem. Then, using a fitness function that rates the suitability of each individual in the population, the genetic algorithm generates a new population modeled after the fittest individuals. An algorithm known as a crossover operator creates the new population by "breeding" the individuals with the highest fitness in the existing population. This new population has a high probability of being fitter than the last and the computer repeats this process, improving or "evolving" the solutions.

A heuristic is a rule of thumb that the computer uses to approximate an optimal solution within a reasonable amount of time. An example of a heuristic can be given in the context of the well-known "traveling salesman" problem in which the goal is to generate an efficient way for a salesman to travel through a given number of cities before returning to his original city. He could simply travel to a random available city and then eventually return to his starting point, but this wouldn't be a very effective way to find an efficient schedule. A heuristic could be implemented that dictated, for instance, that the salesman should travel to whichever city is closest to his current location. This isn't guaranteed to generate the most efficient schedule, but it is probably better than selecting cities randomly. Removal of randomness from a program, especially one with a large number of data points, can make a program much more efficient.

In our program, the genetic algorithm generates a permutation, or random order, of cities. Then, a heuristic generates a flight plan using each order of cities and evaluates the cost of each flight plan. The cost function is the fitness function. The genetic algorithm takes the cheapest flight plans (the most "fit" individuals) in the population and breeds them using "edge recombination". One way to select these parent orders is "tournament selection", in which the program randomly selects from the tournament size and picks the fittest one to be the first parent. It repeats the process to select the second parent. In edge recombination, the program generates "offspring" by trying to keep cities that are next to each other in the parent orders next to each other in the offspring. A detailed description of this process can be found in an article by D. Whitley, T. Starkweather, and D. Shaner entitled "The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Edge Recombination."

Hill-climbing with Heuristics

Advanced hill-climbing combines the previously discussed techniques of hill-climbing and heuristics to achieve a more sophisticated system which produces better results more efficiently than either of the two separately. In advanced hill-climbing, the hill-climbing process is applied to the permutations from which our heuristic generates a flight plan, rather than to an entire flight plan. This approach allows the search to find lower-cost flight plans more quickly. The advanced hill-climb algorithm works in much the same way as the standard hill-climb algorithm. A step takes the current permutation and randomly swaps a number of the cities in this permutation. If the step is cheaper, it takes another step from there, if not, it reverts to the previous permutation and takes a step from there. If too many steps are taken without finding a cheaper permutation, annealing is performed by randomly swapping a greater number of cities in the permutation. The cost of a permutation is calculated by running a heuristic, which we called "Algorithm O", on it. Algorithm O works as follows:

1. Take the first city off the top of the list
2. Take another city off the list. Put everyone who wants to go from this city to the first city on a direct flight.
3. Take the next city off the top of the list. Call this city A. If no more cities remain on the list, STOP.
4. Select city B in the following manner: find the closest city to A that has already been pulled off the list and has not already been exhausted as a city B for this particular city A. If no more candidates for city B remain, go to step 3.
5. Schedule enough flights from A to B to accommodate everyone who wants to fly from A to B. No more flights will be scheduled from A to B.
6. If no space remains on the last flight from A to B, go to step 4.
7. Execute the following sub-algorithm to fill the last flight from A to B:

   I. Call the first city in the list the City X

   II. Continue adding passengers whose eventual destination is X to the flight from A to B until the flight from A to B fills up or the flight from B to X fills up. Do not schedule any additional flights.

   III. If there is any space left on the flight from A to B, let the next city on the list not yet used as City X be

City X. If City X is City A, go to step 4. Otherwise, go to step II. If no more candidates remain for City X, go to step 4. If there is no more space on the flight from A to B, go to step 4.

Parallelization

Our genetic algorithm took hours to run and we determined that parallelization would be worthwhile. In order to achieve multiprocessing while minimizing inter-processor communication time, we chose to parallelize using “demes” or isolated sub-populations. Each one of sixteen processors runs a genetic algorithm almost independently of all of the others; however, every five generations the processors pause to communicate with each other. The best individual from each processing element’s population is chosen to be a “migrant” and is sent to another processor. We use a ring structure, which is the most common choice for genetic algorithms of the scale on which we run ours. In a ring structure each processor has two neighbors as if the processors were arranged in a ring as shown. Each processor sends it best individual to the next processor in the ring and receives a migrant from the previous processor.

Isolated demes have another benefit besides reducing inter-processor communication time. They actually change the nature of the genetic algorithm process to produce better results. The process is nearly equivalent to performing several separate runs of a genetic algorithm and choosing the best result except that the migrants allow more successful results generated by the different processors to be combined. Therefore, because our parallelized algorithm is not exactly equivalent to our single processor algorithms, we are not surprised to see super-linear speedups and improved convergence values as detailed in the results section.

We parallelized using MPI, the Message Passing Interface. We used blocking sends, although much of our inter-processor communication was done using MPI’s combined send and receive function, which is more intelligent than separate sends and receives and therefore may reduce blocking time. Very large systems of genetic algorithms are often implemented using asynchronous communication, but such complicated functionality proved unnecessary for our program, as we show below.

RESULTS

As a baseline from which to judge our results, we calculated the cost of sending everyone to their destination on direct flights with no connecting flights at all, which was about $4.545 billion per day. This simplistic solution is obviously not the most effective way of solving the problem, but it was still less expensive than the results of our worst algorithms. The simplest algorithms which employed hill climbing as a way of reaching a minimum cost turned out to be exceedingly inefficient and couldn't outperform direct flights even when we added "Algorithm O" as a heuristic to increase the method's efficiency. Even our best methods for finding a flight plan only outperformed the flight plan based on direct flights by a few percentage points. Given the magnitude of the costs involved, however, those several percent represent hundreds of millions of dollars.

First, we ran our algorithms on some smaller test cases to assure ourselves that it was working properly. We then tackled the passenger data that we extrapolated as described in the "Input Data Collection" section.

The least effective approach to our problem was hill-climbing by itself, which yielded flight plans costing approximately 175% of the cost of direct flights. Due to the simplistic way hill-climbing runs, it cannot possibly compete with more advanced algorithms. It is slow and imprecise, choosing random paths and then checking them to see if they were better or worse than the temporarily held best one. This is clearly unsatisfactory.

We were also unable to produce a viable result with either integer programming or its simpler cousin, linear programming. When we first tried to solve our massive set of equations with a borrowed integer programming solver, the program crashed before memory was completely allocated. This alerted us to the fact that the integer/linear programming requires vast resources if approached simplistically. Changing our approach to linear programming, which is less resource intensive, and trying to take advantage of the sparsity of our matrices, we were able to produce a loose lower bound of $4.302 billion, or about 95% of the cost of direct flights. This number errs on the low side because the solution counted unfilled flights as costing less than filled ones, whereas in our model the cost of a flight is independent of whether or not it is filled to capacity. The absolute lower bound of 95% that it establishes, however, shows that the improvement of a few percent that we achieved with later algorithms was not trivial.

Our first successful result was produced by the combination of our heuristic with hill-climbing. This produced flight plans that cost around 99% of direct flights. Applied to the budget of approximately $4.5 billion, this results in savings of around $50 million per day. However, because the algorithm does not necessarily produce steadily cheaper flight plans, it is not as worthy a candidate as those that do.

The combination of genetic algorithms and heuristics proved very successful. Using the genetic algorithm to evolve permutations for the heuristic, we produced flight plans that cost approximately 98% of the cost of direct flights. This results in savings of more than $100 million per day. These savings come from having to run fewer or better-chosen flights in a given day. In addition to saving money, this saves on fuel consumption (the cost of fuel for an airplane is about $3,000 per hour). The benefits of a more efficient flight plan are both economical and environmental.

Our best flight plan, generated with the parallelized genetic algorithm, produced a cost of $4.443 billion, 97.7% of the cost of direct flights and savings of almost $102 million. When compared with the lower bound of 95% produced by linear programming, which we know to be lower than the global maximum, our result is clearly meaningful.

Hub Indices

We can calculate a Hub Index for each city by dividing the number of passengers traveling through the city by the number of passengers for whom the city is an origin or a destination:

Hub Index = (# of passengers traveling through city) **/** (# of passengers for whom the city is an origin or a destination)

A city where no passengers make connections will have a Hub Index of 1; a major hub, such as Philadelphia for United Airlines, will be much higher, perhaps 2 or 3; and most of the airports for the 56 largest metropolitan areas are somewhere in between.

An analysis of a flight plan produced by the genetic algorithm/heuristic combination revealed a very smooth distribution of our calculated Hub Index. The city with the highest average Hub Index, Salt Lake City, had an average Hub Index of only 1.14. Boston had the lowest, at 1.00, and the rest were rather evenly distributed between those extremes. This indicates an absence of major hubs, confirming our hypothesis.

Interestingly, a city's Hub Index varied only slightly throughout the most efficient flight plans. We used Minitab for Windows to perform a one-way ANOVA test on the Hub Indices of each city from our five best flight plans. The ANOVA test indicates the likelihood of different sample means pertaining to the same population; in other words, we performed the test to determine whether or not the Hub Indices were random. Our ANOVA test (see Appendix E) showed that the samples came from different populations at the 99.9% confidence level, ruling out randomness. Therefore, *something* clearly plays a part in determining the degree to which a certain city is a hub.

We attempted to discover whether population and distance from the center of the country were factors that determined a city's Hub Index, but plotting was more or less inconclusive. We have included scatter plots of Hub Index vs. Population, Latitude Squared, Longitude Squared, and Distance from Center (see Appendix B). However, these plots do not suggest even a weak correlation between any of these factors and the hub indices, which nevertheless varied very little. The degree to which a city acts as a hub may be determined by a quirk in our particular solution strategy.

Statistical Analysis of Parallelization

In this section, our goal is to statistically verify two elements of our hypothesis that relate to the parallelization of our genetic algorithm: 1), that the parallelized code produces better solutions (that is, flight schedules with a lower cost), and 2), that the parallelized code is more efficient. We measure efficiency by the amount of time it takes one processor to complete one generation of the genetic algorithm.

We will do this by employing the statistical technique of hypothesis testing. Hypothesis testing is performed to investigate preconceived assumptions about some condition in the (statistical) population."[2] The steps of hypothesis testing are as follows:

1. State the null hypothesis, which is the statement to be tested.
2. State the alternative hypothesis, which is based on the belief of the investigator relative to the relationship between the parameters.

3. Specify the values, namely sample size, sample mean, and standard deviation.
4. Determine the appropriate statistical distribution for the analysis.
5. Compute the test statistic using the appropriate equation.
6. By applying the test statistic to the chosen distribution, draw a conclusion as to whether or not the null hypothesis should be rejected. If the null hypothesis is rejected, the alternative hypothesis is acceptable and the investigator's assumption has been verified to a certain confidence level, taken from the distribution table.[3]

In this section, n indicates sample size, $\bar{x}$ indicates sample mean, s indicates standard deviation, and μ indicates population mean.

With the comparison of solution quality, we have two small samples (n_1 = 10 and n_2 = 5) with unequal standard deviations (s_1 = 3221 and s_2 = 1732). We must use the t-distribution in hypothesis testing for comparing the two population means, μ_1 and μ_2.

Null hypothesis: $\mu_1 = \mu_2$

Alternative hypothesis: $\mu_1 > \mu_2$

$n_1 = 10$

$n_2 = 5$

$\bar{x}_1 = 4455035$

$\bar{x}_2 = 4447041$

$s_1 = 3221$

$s_2 = 1732$

[2] Research and Education Association, Statistics, © 1997, p 179

* We calculate the degrees of freedom, , using the following formula, and rounding the answer down:

$$df = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)}{\frac{\left(\frac{s_1^2}{n_1}\right)}{n_1 - 1} + \frac{\left(\frac{s_2^2}{n_2}\right)}{n_2 - 1}}$$

$$df = \frac{\left(\frac{3221^2}{10} + \frac{1732^2}{5}\right)}{\frac{\left(\frac{3221^2}{10}\right)}{10 - 1} + \frac{\left(\frac{1732^2}{5}\right)}{5 - 1}} = 6.17 \approx 6$$

We then calculate the test statistic, or t-score, which is done using the following formula:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{s_{\bar{x}_1 - \bar{x}_2}}$$

where

$$s_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

so

$$s_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{3221^2}{10} + \frac{1732^2}{5}}$$

and

$$t = \frac{(4455035 - 4447041) - 0}{1279.63} = 6.247$$

[3] ibid., p 179-182

The t-score of 6.247, when applied to the t-distribution indicates a tail probability of less than .0005. We can therefore reject the null hypothesis at the 99.95% confidence level and can conclude with certainty that the program generates better solutions when parallelized.

Similarly, when comparing the runtimes per processor per generation, we have two small samples (n_1 = 10 and n_2 = 5) with unequal standard deviations (s_1 = 0.025 and s_2 = 0.352).

Null hypothesis: $\mu_1 = \mu_2$

Alternative hypothesis: $\mu_1 > \mu_2$

$n_1 = 10$

$n_2 = 5$

$\overline{x}_1 = 8.60$

$\overline{x}_2 = 7.82$

$s_1 = 0.025$

$s_2 = 0.352$

We again calculate the degrees of freedom, and round the answer down:

$$df = \frac{\left(\frac{.025^2}{10} + \frac{.352^2}{5}\right)}{\frac{\left(\frac{.025^2}{10}\right)}{10-1} + \frac{\left(\frac{.352^2}{5}\right)}{5-1}} = 4.00 \approx 4$$

We then calculate the test statistic, or t-score:

$$s_{\bar{x}_1-\bar{x}_2} = \sqrt{\frac{.025^2}{10} + \frac{.352^2}{5}}$$

and

$$t = \frac{(8.60 - 7.82) - 0}{.1576} = 4.948$$

The t-score of 4.948 falls between the tail probabilities of .005 and .0025. We can therefore reject the null hypothesis at the 99.5% confidence level and can conclude with reasonable certainty that the program runs faster per processor per generation when parallelized on 16 processors.

As these tests show, the time spent implementing the parallel version of the algorithm was justified by both the improved runtime per processor per generation and superior solution quality it yielded. We were able to generate better solutions in a shorter period of time with the parallelized code.

Mergers and Closures

We simulated a merger between two identical airlines by doubling the number of passengers to be scheduled. We found that a set of passengers twice as large can be scheduled to fly to their destinations for less than twice the cost. This supports the obvious: mergers enable airlines, with larger amounts of resources at their disposal and more flexibility, to fly passengers more efficiently than two separate companies would be able to. We can soundly make the following generalization: the problem becomes less and less challenging as the number of passengers increases with respect to the capacity of a flight. Also, more sophisticated algorithms outperform direct flights more significantly when the number of passengers is smaller with respect to flight capacity. This reflects common sense – when there are more than enough people to fill any given flight, a direct flight will be the most profitable scheduling solution, whereas when direct flights are likely to be closer to empty a plan that combines groups of passengers at centralized locations, to then be flown to their destinations together, will be more efficient.

Our algorithm can also be used to redirect passengers in the event of an airport closure due to weather or technical problems. Using a pre-computed table or a fast algorithm a new flight system an airline could quickly obtain a new flight plan that would allow as efficient a redirection as possible of all of all remaining passengers.

CONCLUSION

Both aspects of our hypothesis proved to be correct. We predicted that of the algorithms we tested, the combination of genetic algorithms with heuristics would produce the best solution, and we predicted that the least expensive system of flights that our program produced would have minimal reliance on hubs.

The genetic algorithm did indeed prove to be the most effective, consistently generating the least expensive flight plan on several different test cases. It provided flight schedules that cost about 98% of the cost of direct flights, which translates to savings of about $100 million per day. Of course, real airlines schedules are presumably also more efficient than schedules that use direct flights exclusively, but we still believe that the possible savings justify the additional resource use required to run the genetic algorithm with heuristic program. The hill-climbing with heuristic had a slightly quicker runtime, meaning it produced reasonable schedules slightly faster than the genetic algorithm with heuristic, but its schedules were only 99% as costly as pure direct flights, a difference of around $50 million per day.

Parallelization of our genetic algorithm was vindicated by the quicker, better solutions that it produced. The time spent parallelizing and the additional runtime was clearly worth it: in economic terms its benefits can be measured in the millions of dollars per day.

One very important application of our successful algorithm is flight rescheduling in response to airport closures. With our algorithm, airlines could react quickly to closures in order to minimize passenger inconvenience and unexpected costs.

We also surmised that the most efficient flight schedules would minimize reliance on hubs. From the smoothness of the distribution of the hub indices we can draw the conclusion that hubs are absent from the most efficient flight schedules that we were able to produce. We have not proved rigorously that hubs limit efficiency, but it is certainly indicated by our experimental results. It is also logical that less dependence on hubs would provide several benefits for consumers since direct flights are clearly the most efficient way for passengers to travel from one city to another. An efficient flight plan, however, cannot consist entirely of direct flights. In a less hub-reliant plan, connecting flights are likely to connect in cities that are more convenient than the previous major hubs. This combination of direct flights and more convenient connecting flights not only saves the passengers time but also translates to lower ticket prices because there would be less money needed for fuel, crew and passenger food.

We concede that hubs have benefits that are not taken into account by our simulation. One example is their function as centralized repair centers. A hub system ensures that planes will fly through the hubs frequently, where they are repaired at sites specialized for the repair process. This advantage is lost in our more equitable flight distribution. However, technological advances may render repair and similar, hub-specific issues more and more insignificant.

Switching to a general system where airline schedule flights without large, fixed hubs in mind is not a change that can be made quickly. The main obstacle to overcome when implementing this plan is the fact that hubs have developed because they have the capacity to facilitate large numbers of connecting flights through them. The transition from our current system of large, fixed hubs to a system where each airport can facilitate connections easily cannot be made overnight. Certain airports have been expanded to accommodate hub status and will need be reduced in size. Other airports will need to expand to handle traffic that large hubs handled previously. If airports can make this transition, in the long run it will be well worth it. The projected daily saving would, in time, pay for the expenses of expanding smaller airports and limiting expansion of large airports. Once implemented, the plan could save vast amounts of money for the airlines, and perhaps time and money for consumers as well.

ACKNOWLEDGMENTS

We would like to thank David Bader and Stephanie Forrest of the University of New Mexico for a wonderful lunch conversation, from which we got a lot of substantive advice on the programming specifics of our project. They were most forthcoming both that day and in response to our subsequent inquiries. Mr. Bader aided us with conceptualizing the problem, as well as the heuristics; Ms. Forrest, a specialist in genetic algorithms, provided very useful words of wisdom in her area of expertise. We would also like to thank Barak Pearlmutter of UNM for his contributions to our linear programming algorithm and Toby Meek of Albuquerque Academy for his proofreading of our statistical analysis.

Our faculty sponsor, Jim Mims, has been most supportive of our efforts, providing us with logistical assistance, transportation, and anything else that we asked for.

We appreciate the New Mexico Supercomputing Challenge and its administrators for getting us started on this project, and providing us with a forum in which to develop it.

Thanks to the Brown and Widland households for hosting our meetings, and to Mrs. Neerken for a steady supply of delicious baked goods.

REFERENCES

Poole Jr., Robert W., and Butler, Viggo. "Airline Deregulation: The Unfinished Revolution". April 4, 2001. < http://www.cato.org/pubs/regulation/regv22n1/airline.pdf/>.

Sedgewick, Robert. Algorithms in C++: Fundamentals, Data Structures, Sorting, Searching. Atlanta: Addison-Wesley Publishing Company, 1999.

Sipser, Michael. Introduction to the Theory of Computation. Boston: PWS Publishing Company, 1997.

Abelson, Harold, and Sussman, Gerald Jay. Structure and Interpretation of Computer Programs. Cambridge: The MIT Press, 1996.

"Federal Aviation Administration." FAA. Feb. 12, 2001. <http://www.faa.gov/>.

"Department of Transportation." DOT. Feb. 12, 2001. <http://www.dot.gov/>.

"Bureau of Tranportation Statistics." BTS. Feb. 12, 2001. <http://www.bts.gov/>.

LIST OF APPENDICES