Modeling Tools for Airline Crew Scheduling and Fleet Assignment Problems

R.L. Ho
E.L. Johnson
T.L. Shaw

School of Industrial and Systems Engineering
Georgia Institute of Technology

Abstract: Because of recent technology improvements, it has become practical to solve airline planning problems in the PC environment. Here, we use PC tools for modeling and data manipulation. For crew scheduling, we have developed a Microsoft Access[1] interface that can also display various aspects of the schedule, parameters for the problem, and solution. Parameter default values for legality rules and for costs are provided along with helps to explain the rules and costs. Parameters can be changed and new schedules reflecting the new values generated. CPLEX[2] is used as the optimizer. However, the optimum solution is indicated in the original list so that the optimizer is hidden from the user. For the fleet assignment problem, the model is given both in AMPL[3] and Access. The Access version allows the schedule as input and generates CPLEX input without leaving Access.

Models are available at http://tli.isye.gatech.edu/airlines/models/

1. Introduction

The PC and PC network environments continue to experience rapid technological advances pushing the bounds of both computing speed and memory size. Both of these factors have constrained us, in the past, from being able to solve practical optimization problems of any large size in this environment. This was especially true for problems, such as airline crew scheduling and fleet assignment, whose problem size grows rapidly with the size of the input data. As the PC and PC network environments become more powerful, new tools are emerging that can harness and enable users to easily access and exploit that power.

We developed a set of PC-based modeling tools for the airline crew scheduling and fleet assignment problems. Initially, these tools were developed to support an Airline Operations Research class being taught at the Georgia Institute of Technology. The tools were designed for students and researchers to help them better understand the crew scheduling and fleet assignment problems. As such, they allow users to easily view and manipulate the model and data and see the results of the various changes they make. This usage of the tool makes it valuable in what-if studies such as affects of increasing overnight rest times.

For crew scheduling the changes that can be made center on the legality rules and cost formulas used to generate the pairings. It is also easy to alter the schedule, change which stations are the crew bases, and set station specific overrides for the minimum and maximum sit and overnight rest at a particular station. For fleet assignment the changes that can be made include altering the schedule, adding or removing fleet types, changing the cost and revenue data, and restricting a fleet type from flying certain legs.

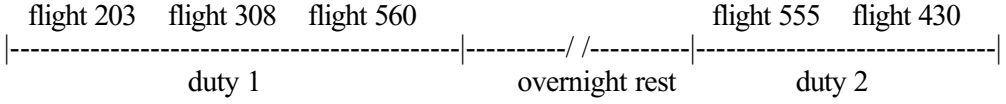
We developed some special purpose code but we extensively use the PC-based applications Microsoft Access (for data management and user interface) and CPLEX (for optimization). For the fleet assignment problem, we also developed a PC-based AMPL model.

In section 2, we discuss the modeling tools for crew scheduling. We describe our basic crew scheduling model, the applications we chose to use, the user interface we developed for entering the data and problem parameters and viewing the output, data structures, and application integration. In section 3, we discuss the modeling tools for fleet assignment. We describe the underlying AMPL model, the conversion to Access/CPLEX, the data structures required, and application integration issues.

2. Crew Scheduling

2.1 Basic Model: The basic model that we use for the crew scheduling problem is the set partitioning model. The columns or sets are called pairings. A pairing is a sequence of flight legs that represents a possible itinerary that a crew might be assigned to fly. It starts at a crew base and ends at the same crew base. It is made up of some number of duties with periods of overnight rest between the duties. Duties can be thought of as a day's work or a day's worth of flying.

The following represents a typical pairing



There are a significant set of rules and regulations that govern what constitutes valid duties and pairings. Such things as the time between flights, number of flights in a duty, total amount of flying time in a duty, length of duties, length of overnight rest, number of duties in a pairing, and length of pairings are regulated. In addition, the cost of a pairing depends on several factors and, in general, can only be computed after the entire pairing has been determined. For these reasons, special purpose algorithms are often used to generate and compute the cost of valid pairings separately from the optimizer.

In a straightforward application of the set partitioning model to crew scheduling, all of the valid pairings are generated up front. For each pairing, a leg incident vector (a vector such that each component j is a 1 if leg j is contained in the pairing and a 0 otherwise) is generated and the cost of the pairing is computed. Let a^i be the leg incident vector for pairing i and let c_i be the cost of pairing i. We define x_i as a binary variable that will be 1 if we select pairing i and 0 otherwise. The resulting set partitioning problem is

$$\begin{aligned} \text{Min} \quad & c^T x \\ \text{s.t} \quad & Ax = 1 \\ & x \in B^n \end{aligned}$$

where the optimal solution is a set of pairings such that each leg is flown exactly once at the least cost. Note that with this straightforward approach, all of the complexities of the problem are dealt with in the specialized pairing generation algorithm. The optimization model is essentially fixed and quite simple conceptually. The focus of our modeling tools, for crew scheduling, is to

provide the user with the ability to easily modify the various rules and regulations that drive the pairing generation algorithm and quickly see the effects on both the pairings that were generated and the ultimate solution.

Computationally, the difficulty in solving this basic crew scheduling problem is the number of columns (pairings) that may be present. For this paper, we assume the problem is not so large that it cannot be passed to CPLEX and solved. See [ATJ] for a global approach to solving the larger crew scheduling problems. This work expanded on the earlier sub-problem optimization discussed in [AGPT]. See also [Hu] for a discussion of quasi-explicit matrices for compact storage and primal-dual methods to solve the lager problems.

2.2 Applications: Our aim was to develop a fairly graphical user interface for modifying the problem parameters. We wanted the user to be able to turn on and off various legality and cost rules and we wanted to provide easy access to default values for the parameters and help. We also knew that for the presentation of the results, especially the pairing generation results, it would be meaningful to provide a "drill down" capability. That is, we wanted to provide the user with the ability to view a list of the generated pairings and then from any pairing "drill down" to see the duties (or flight legs) that made up that pairing and from any duty "drill down" to see the flight legs that made up that duty. Microsoft Access provides both the user interface and the relational tables needed to display the results.

For the pairing generation, we developed special purpose C++ code. The pairing generation code retrieves the parameter data that governs the legality checking and cost computations for the pairings. The code incorporates the parameter data into its routine for generating first the duties and then the pairings. It uses a depth first search scheme over the flight legs to enumerate all valid duties and then a depth first search scheme over the duties to enumerate all valid pairings. It outputs the results in a format that can be used by Access to display the results and provide the "drill down" capability described above.

We chose CPLEX, which can be called from C++ programs in a PC environment quite easily, as the optimization application. We developed special purpose C++ code that retrieves the pairings that were generated, sets up the CPLEX environment, loads the problem, and calls CPLEX to solve it. CPLEX solves the problem as a mixed integer optimization problem. Afterwards, the special purpose code retrieves the results from CPLEX and outputs them in a format that can be used by Access.

2.3 User Interface: The primary input data is the schedule. Our modeling tools assume the user wants to solve a daily problem. That is, we assume that every flight leg in the schedule provided by the user is flown every day of the week. Often, especially for large problems, a daily problem is solved first and then weekly exceptions are incorporated into the solution. Since our primary intent was to provide users with an understanding of the basic crew scheduling problem and a forum to interactively alter various data and problem parameters, we did not feel allowing for weekly exceptions to be a high priority feature.

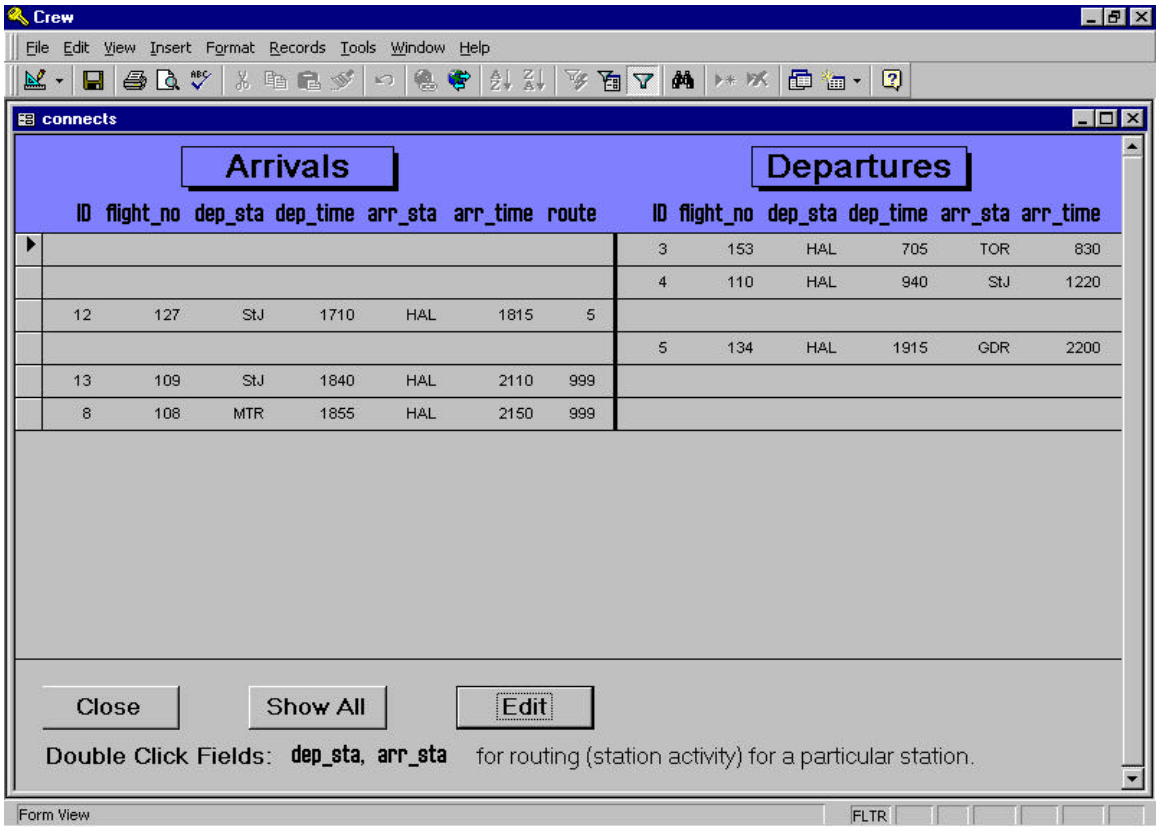
Therefore, the schedule data consists of only a flight leg identifier, the departure station and time, the arrival station and time and, optionally, a plane routing. The flight leg identifier need not be unique and is not used by the model. Each line item in the schedule table is assigned a unique ID and is taken as a unique leg flyable by any crew. That is, there is no advantage enjoyed when the same crew flies contiguous sections of a sequence of legs that share the same flight leg identifier (unless the plane is necessarily routed to the next flight leg in the sequence, in which case, the minimum sit is not enforced).

The departure and arrival stations should be the unique three character airport codes. The departure and arrival times should be numbers from 0 to 2359 representing 00:00am to 23:59pm and they should be given as local times to the departure or arrival stations respectively. The plane routing is optional. It should be a number that represents the line item in the schedule table for the leg the plane will be routed to after this leg. The tool provides an easy means to input the routing by displaying a view of the schedule showing all station activity. That is, all arrivals and departures for a station sorted by times. The tool handles all renumbering required as a result of deleting or adding to the schedule. The default value of 999 indicates that there is no routing (either no routing data was entered or the plane overnights after this leg).

An example of a line of schedule data is shown below. This is flight 1155 from Boston to Chicago. It leaves Boston at 11:30am and arrives in Chicago at 1:08pm. No plane is routing is provided

ID	Flight	Dep Sta	Dep Time	Arr Sta	Arr Time	Route
7	1155	BOS	1130	ORD	1308	999

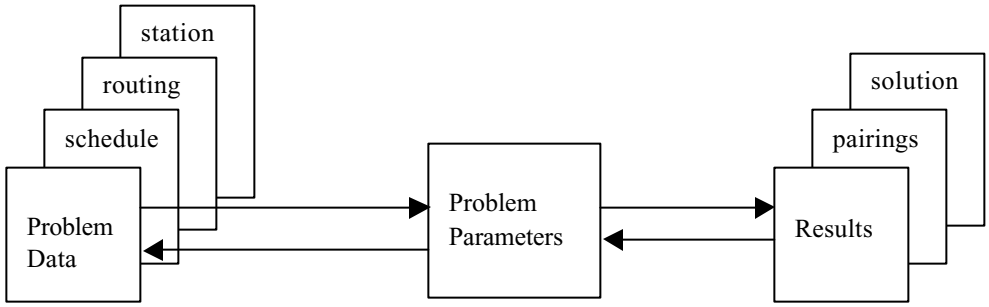
The station activity view (shown below) is a very useful way to display the schedule data. In addition to making it easier to enter the routing data, it allows users an easy way to ensure that the schedule is balanced (that the number of inbound flights equals the number of outbound flights at each station). Also, it helps the user see how various connections within the pairings, and the optimal set of pairings, were made and where some schedule or parameters changes might make a difference in the pairings generated and in the ultimate optimal solution. For these reasons, the station activity view is automatically produced by the tool and made available from various screens by simply double clicking on a station code.



In addition to schedule data, the user provides two important pieces of information about the stations: the time zone each station is in and whether or not the station is a crew base. These data are required by the pairing generation algorithm since it must convert all times to a global standard and it must only start and end pairings at crew bases. Some station specific overrides for minimum and maximum sit and overnight rest can also be specified for any station. These values will override the parameter input screen settings.

The schedule and station data can be entered directly into the Access tables. We developed a set of easy to use input forms for this purpose. Schedule and station data can also be saved in a special format to text files and then later retrieved using the tool. This allows users to switch between multiple problem instances or versions of problem instances. Also, if schedule and/or station data is available from another source, it can be imported using an ASCII delimited file format.

Central to the user interface is the notion of interactivity. The tool is designed to allow users to easily change not only the problem data but also the problem parameters and run the pairing generation and/or the solution code again. In this way, the user can see the affects of different choices of rules and various data instances on the pairings that are generated and on the ultimate solution. The tool is set up so that the user can move easily between problem data, problem parameter, and result screens. The diagram below shows the basic user interface flow.



The parameter input form is at the heart of the user interface. From this form the user specifies all legality rules and cost factors that will be used by the pairing generation algorithms. All rules and cost factors can be turned on or off with check boxes. If they are turned on, a parameter can be specified (note: sometimes more than one parameter is required and sometimes none are). For example, minimum overnight rest, which is the minimum amount of time required between duties, can be turned off completely or turned on and, if turned on, the user can specify the number of hours that should be used as the minimum amount of time between duties.

Default on/off settings and default parameter values are established for each installation of the tool. The user can return the entire parameter input form to the default settings or can reset any specific rule or cost factor to its default. Each rule and cost factor also has an information form associated with it. The user can request this pop up form to receive specific information about the purpose of any rule or cost factor and how it is used by the pairing generation algorithm.

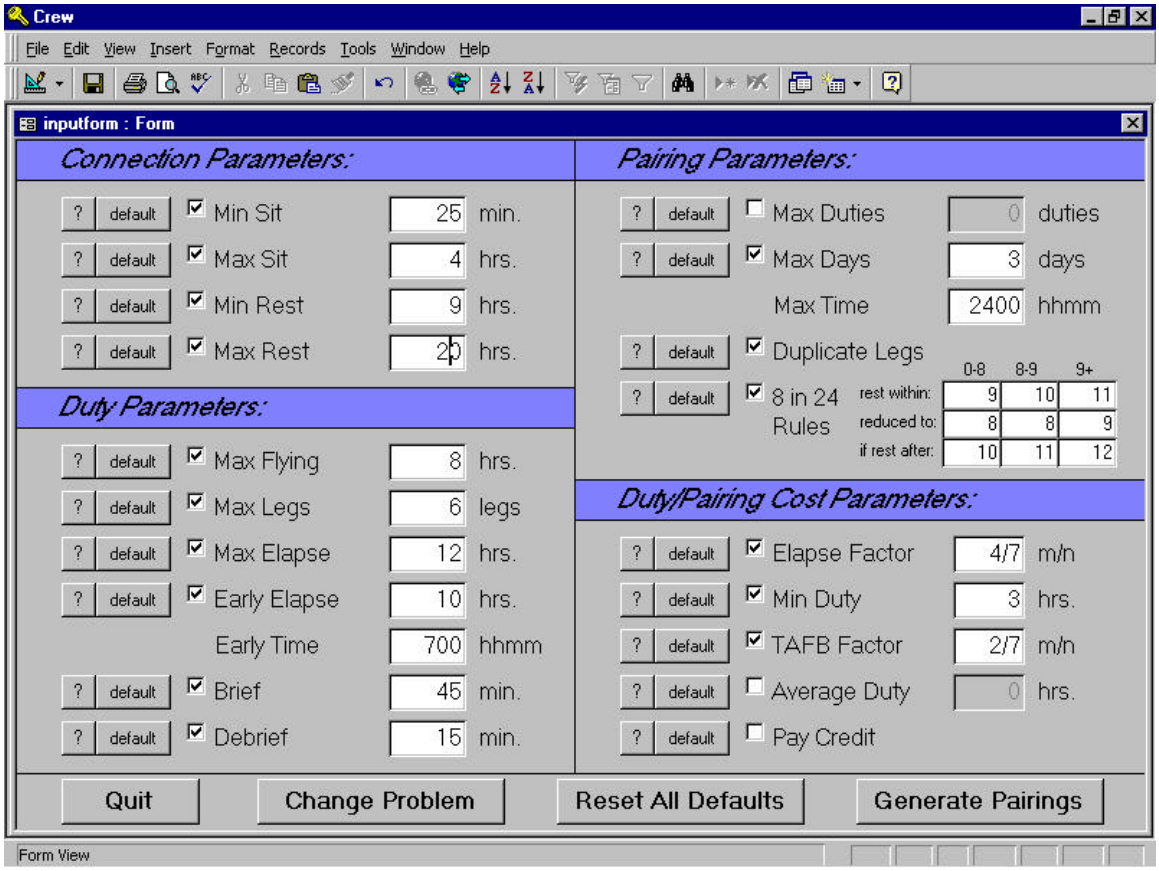
On the next page, we show the parameter input form. Notice that it is divided into four sections.

1. The upper left corner contains the connection parameters. These include minimum and maximum times for the sit between flight legs and minimum and maximum times for the overnight rest between duties.

2. The lower left section contains the duty parameters. These include maximum flying time and maximum number of legs per duty, maximum elapse time of a duty (and maximum elapse time of a duty beginning earlier than a certain early start time), and required brief and debrief periods before and after a duty.

3. The upper right section contains the pairing parameters. These include maximum number of duties in a pairing and maximum number of days in a pairing (i.e. the requirement to return by a certain time on the specified day). Also included are an indicator for whether to eliminate duplicate legs in a pairing and an indicator and related parameters for whether to impose 8 in 24 rules.

4. Lastly, the lower right corner contains cost parameters. These include the duty elapse time factor and minimum duty guarantee used in the duty cost formula, the time away from base (TAFB) factor and the average duty guarantee used in the pairing cost formula, and an indicator for whether to compute the paring cost as pay-in-credit or actual cost.



Once the problem data has been entered and the problem parameters specified, the user can run the pairing generation code with the click of a button. After the code executes, a form showing each station with the numbers of legs, duties, and pairings that originate from that station is displayed. From this form, users can double click on any number to see the respective legs, duties, or pairings originating from a particular station or they can click on buttons at the bottom of the screen to see all the duties or all the pairings that were generated.

The following data are included for pairings

- crew base (recall the departure and arrival station is the same and must be a crew base)
- departure day and time and arrival day and time
- TAFB (time away from base, elapse time of the pairing)
- Amount of flying time in the pairing
- cost
- number of legs in the pairing and number of duties in the pairing*
- indicator for whether the pairing is in the optimal solution (included only after the solution code is run)

The following data are included for duties

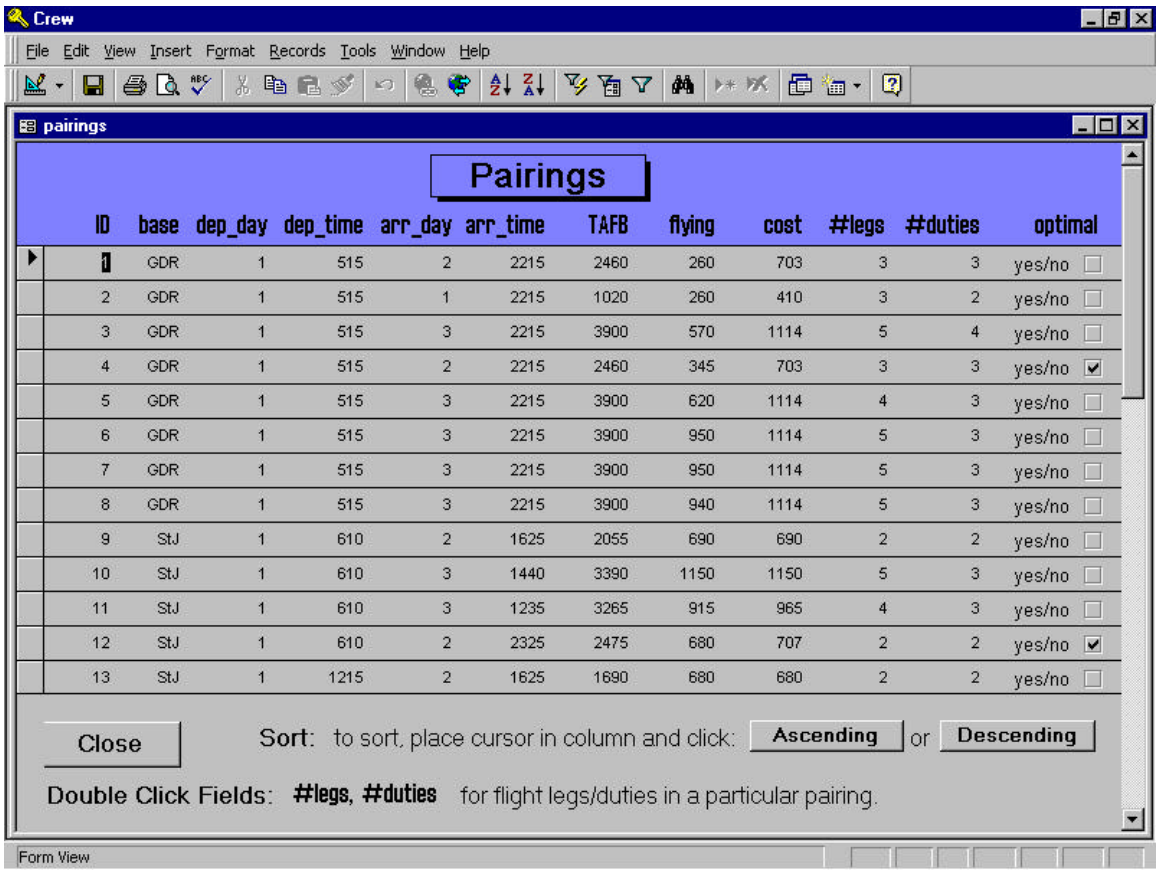
- departure station, day and time and arrival station, day and time
- elapse time of the duty
- amount of flying time in the duty
- cost
- number of legs in the duty*

*By double clicking on the number of duties in a pairing, the user can view only those duties that are contained in the pairing. Similarly, by double clicking on the number of legs in a pairing or duty, the user can view only those legs (from the schedule) that are contained in the pairing or duty. We refer to this as being able to "drill down" to lower levels of related information.

This "drill down" capability enhances the interface to the results by providing users an easy method for dissecting a pairing into its duties (or legs) and dissecting a duty into its legs. This technique provides a good visualization of the duties and pairings and how they are constructed. It also can yield valuable insights into the cost computations, the enumeration procedure, and even the data itself. And by altering the rules and rerunning the pairing generation code, users can see their changes in action (i.e. how the pairings that are generated are effected -- which new ones get generated, and which current ones become illegal).

When the user is satisfied with the pairings that are generated, the problem solution code can be run. The same result screens are displayed as before but now, if an optimal integer solution was found, the pairings in the optimal solution will be identified with a check in the optimal checkbox. By sorting on this field, the user can bring all pairings in the optimal solution to the top of the list.

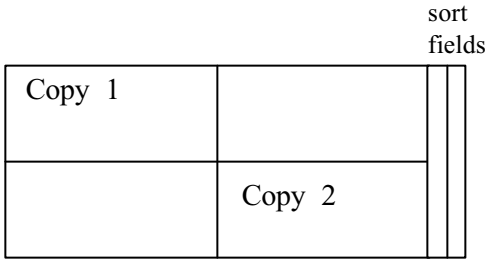
Below is an example of the pairing data produced by the tool.



2.4 Data Structures: The schedule table is the main table used to store the problem data. Each record in the table represents a single flight leg. The tool maintains a unique leg ID, which is re-sequenced whenever the pairing generation code is run, as the key for the table. This re-sequencing keeps the indexes in line with the result tables that relate duties and pairings to legs. We discuss the result tables and their relationships in more detail later in this section.

The stations table stores station specific information. It is constrained by the schedule table (i.e. all stations listed as either an arrival or departure station in the schedule must be included in the stations table and only those stations). The tool ensures this by controlling both the stations view to the user and the stations outputted to the pairing generation code. A unique station ID, which is re-sequenced whenever the pairing generation code is run, is used as the key for the table.

The connects table supplies the information for the station activity view. It is derived directly from the schedule table. A sequence of queries places two copies of the schedule in one table, as shown in the diagram below, and loads the sort fields. The sort fields for the first half of the table are loaded with the arrival station and time while the sort fields for the second half of the table are loaded with the departure station and time. When the entire table is sorted by the sort fields, the result is exactly the station activity view described earlier.



Whenever changes are made to the schedule table, the tool runs various queries to keep the stations and connects tables current.

There are two single record tables for the problem parameters. One stores the current parameter settings and the other stores the default settings. It is a simple matter to copy a single field from one table to the other (to restore the default setting for one parameter) or to copy the entire record from one table to the other (to restore all the default settings).

The result tables contain the results of the pairing generation code and the problem solution code. The pairing generation code produces four ASCII files that Access links to so that they become accessible "tables". These tables are duties, pairings, p2d, and d2l. The problem solution code produces one ASCII file that Access imports into a table called optimal. We discuss more about importing/exporting versus linking in section 2.5.

The result tables and related queries are designed to exploit the power of relational databases. This part of the relational data model is shown in figure 1. The pairings and duties tables contain all the information (listed in section 2.3) about the pairings and duties that were generated. As each duty was generated, it was assigned a number starting from 1. This duty number becomes the unique ID used as the key for the duties table. Similarly, each pairing was assigned a unique ID that is used as the key for the pairings table.

The p2d table is a junction table between pairings and duties. Thus, there is a one-to-many relationship between pairings and p2d and a one-to-many relationship between duties and p2d. Each record in p2d has a unique pairing ID (P:ID) and duty ID (D:ID) combination. A particular combination appearing in this junction table indicates that pairing number P:ID contains duty number D:ID. Similarly, the d2l table is a junction table between duties and legs and each record has a unique duty ID (D:ID) and leg ID (L:ID) combination. Here, the L:ID corresponds to the leg ID in the schedule table.

In addition to identifying which duties are contained in which pairings and which legs are contained in which duties, the p2d and d2l junction tables also include data associated with each pairing-duty and duty-leg combination. These data provide sequencing and day information that are necessary to ensure that the legs/duties are displayed to the user in the proper order and with the appropriate day during a "drill down" request. Departures and arrivals for legs, duties, and pairings are displayed to the user as both day and time with the time being in the local time zone.

The optimal table includes two fields: a pairing ID and the optimal "yes/no" checkbox. There is a record included in the table for only those pairings that are in the current optimal solution and the checkbox is marked "yes" for each one. In a sense, we are storing information that applies to a subset of pairings and thus we have a one-to-one relationship between pairings and optimal. A simple join, including all records from pairings and only those records from optimal that match, is all that is needed to correctly include the optimal checkbox field in the pairings query.

Figure 1: Relational Data Model for Crew Scheduling

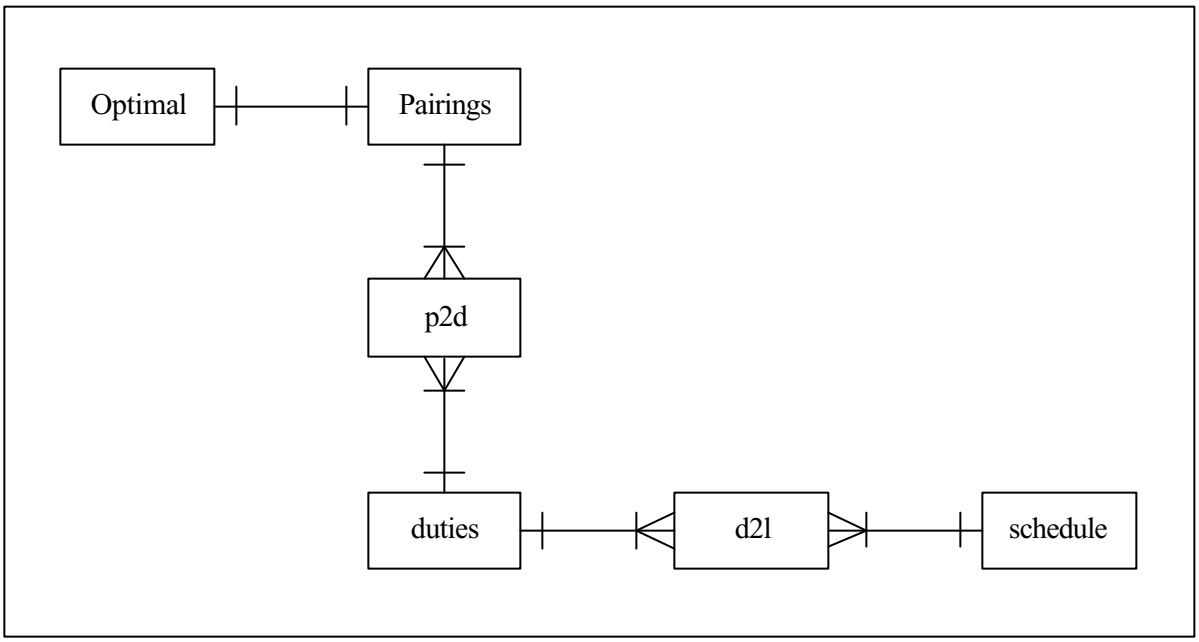
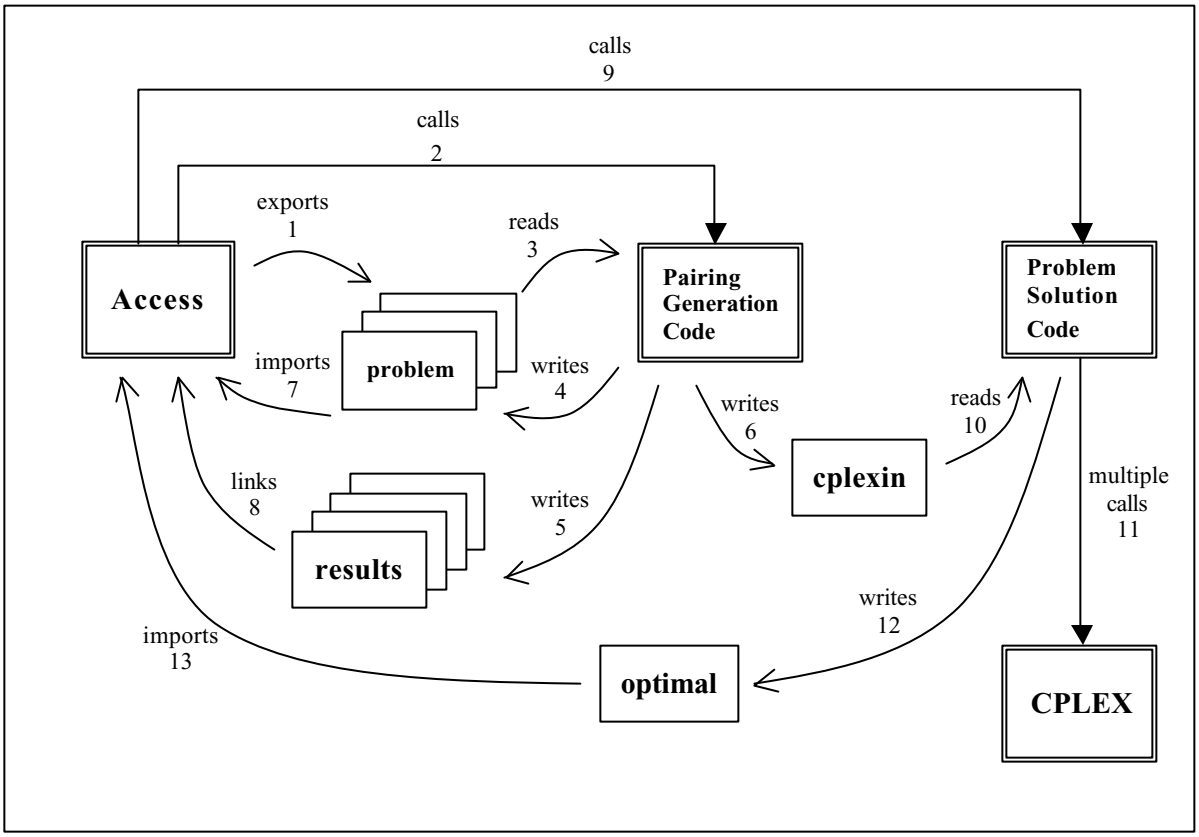


Figure 2: Data Flow Diagram for Crew Scheduling



2.5 Application Integration: We now discuss the integration among the various applications we used to develop the crew scheduling modeling tools. Recall that we chose Microsoft Access for the user interface. As such, Access became the primary application. It calls the special purpose pairing generation and problem solution codes. The problem solution code, in turn, calls CPLEX.

The calls to the pairing generation code and the problem solution code are fairly straightforward. A text file is used to store the current status of the execution. Access sets the text file initially and then calls the appropriate special purpose code with a run application routine. The special purpose code runs with Access, asynchronously, as a console application in a command prompt window. It updates the contents of the status file as it runs and, when it completes its execution, the window closes. The now visible Access status window instructs the user how to proceed. Access reads the status file into a form that is displayed to the user. The status file for the problem solution code includes the optimal solution value when an optimal integer solution is found.

The calls to interface with CPLEX are accomplished through the CPLEX dynamic link library routines. There are routines to setup the environment, load the problem, instruct CPLEX to solve the problem, and retrieve status/results. We use standard CPLEX calls, so we will not be discussed them in detail here.

There are data that must be shared among the applications. We chose to pass this data among the applications by exporting, importing and, in some cases, linking to ASCII files. Specifications in Access allow the exports, imports, and links to be executed from macros, completely transparent to the user. The advantages of using ASCII files over reading/updating the Access tables directly from applications outside of Access are:

1. We can export only the data we need and in the exact format required.
2. We did not need to alter existing Unix-based code that reads data from files.
3. For the larger tables, where we only link to the ASCII file, it is faster and requires less memory to simply write data out to a file as the program executes.
4. The re-sequencing and index control is easier -- it becomes a direct consequence of exporting and importing when the original table is emptied prior to importing.

The data flow diagram in figure 2 shows the primary data flows. The double lined boxes are the applications. The single lined boxes represent ASCII files.

When the user wants to generate pairings, Access exports the problem data: the schedule, station information, and parameters (step 1) and calls the pairing generation code (step 2). The pairing generation code reads the problem data (step 3). When it completes its execution, it writes back out some of the problem data (step 4). This is because the data may have required re-sequencing, and new data such as the number of duties and pairings originating from each station are needed.

The pairing generation code also writes out the four results files that Access needs to display the results (step 5). However, these data are not arranged in a format easily used by the problem solution code, so the pairing generation code produces a version of the generated pairings that are specifically designed for input into CPLEX (step 6). This file is called cplexin. It is a variable length file and has the following format.

Pairing ID Cost Number of Legs LegID LegID LegID …

Access imports the new problem data into the schedule and stations tables (step 7). However, due to the potential size of the result files and the fact that the user only accesses the results in read mode, Access only links to the result files (step 8).

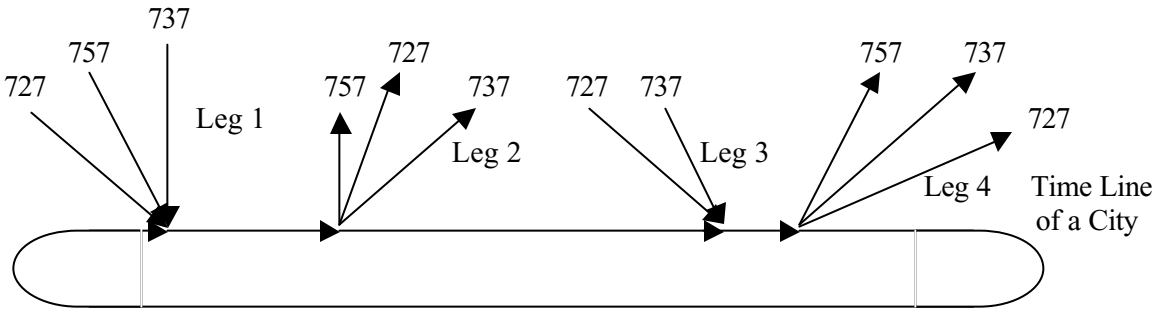
These first 8 steps may be repeated multiple times as the user tries different problem data and parameters to see the resulting duties and pairings that were generated. When the user is satisfied with the results and wants to solve the problem, Access calls the problem solution code (step 9). The problem solution code reads the cplexin file (step 10) and handles all the required CPLEX calls to setup, load, solve, and retrieve the solution to the problem (step 11). Then it writes the optimal data (step 12) which Access imports into the optimal table (step 13).

3. Fleet Assignment

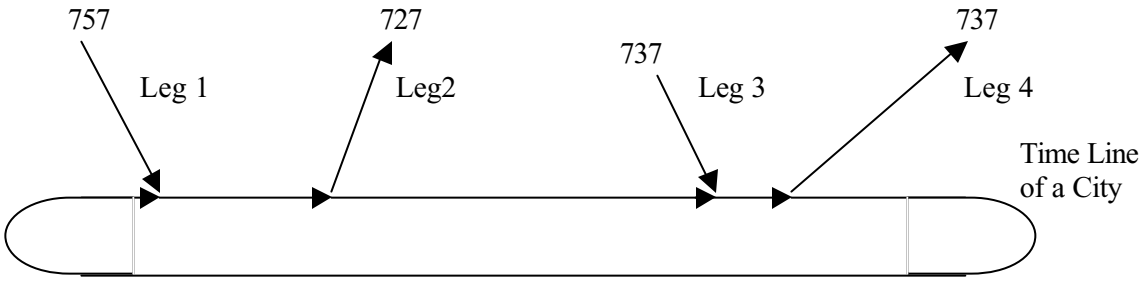
3.1 AMPL Model: We first formulated an AMPL model for the fleet assignment problem [CHJN]. AMPL is a modeling tool that can run in the PC environment. Once a model is entered, users can load data according to the model specifications and call various optimizers to solve problems. We first describe in details the AMPL model.

In the AMPL model, SCHED is a schedule which contains the following information: Leg ID, Departure Station, Departure Time, Arrival Station, Arrival Time, and Equipment. Although we named it as Arrival Time in SCHED, it is actually the Ready Time, which is the arrival time plus turn time. All the time records range from 0000 to 2359. ALLTIME is a circular list of time for any combinations of stations and equipment. REDEYE, which is a subset of SCHED, has departure time greater than arrival time. Fassign is a set of binary variables. Each variable in Fassign is a record of SCHED. The solution to Fassign is 1 when we select such leg and 0 otherwise. Garc is a set of integer variables. Each variable in Garc represents a ground arc, which contains Station, Equipment, and Time. The solution to Garc is the number of planes in such ground arc.

There are three most important types of constraints in the AMPL. They are Cover, Pcount, and Balance. Cover is to make sure each leg is covered by exactly one fleet type. Pcount is for a given time. We need to make sure we don't use more planes than we have. Balance is to ensure whenever there is an in-flow, there must be a corresponding out-flow.



The above represents the possible fleet types for each leg. Below is a feasible solution after we apply the constraints.



For the complete AMPL model we used, please see Appendix A.

3.2 FAM in Access: AMPL is designed to be used not only as a model builder but also as a means to enter a problem instance and call an optimizer. However, we want to prove a more versatile environment for users to manipulate the model and enter and modify problem data. Once again we selected Access to provide this environment.

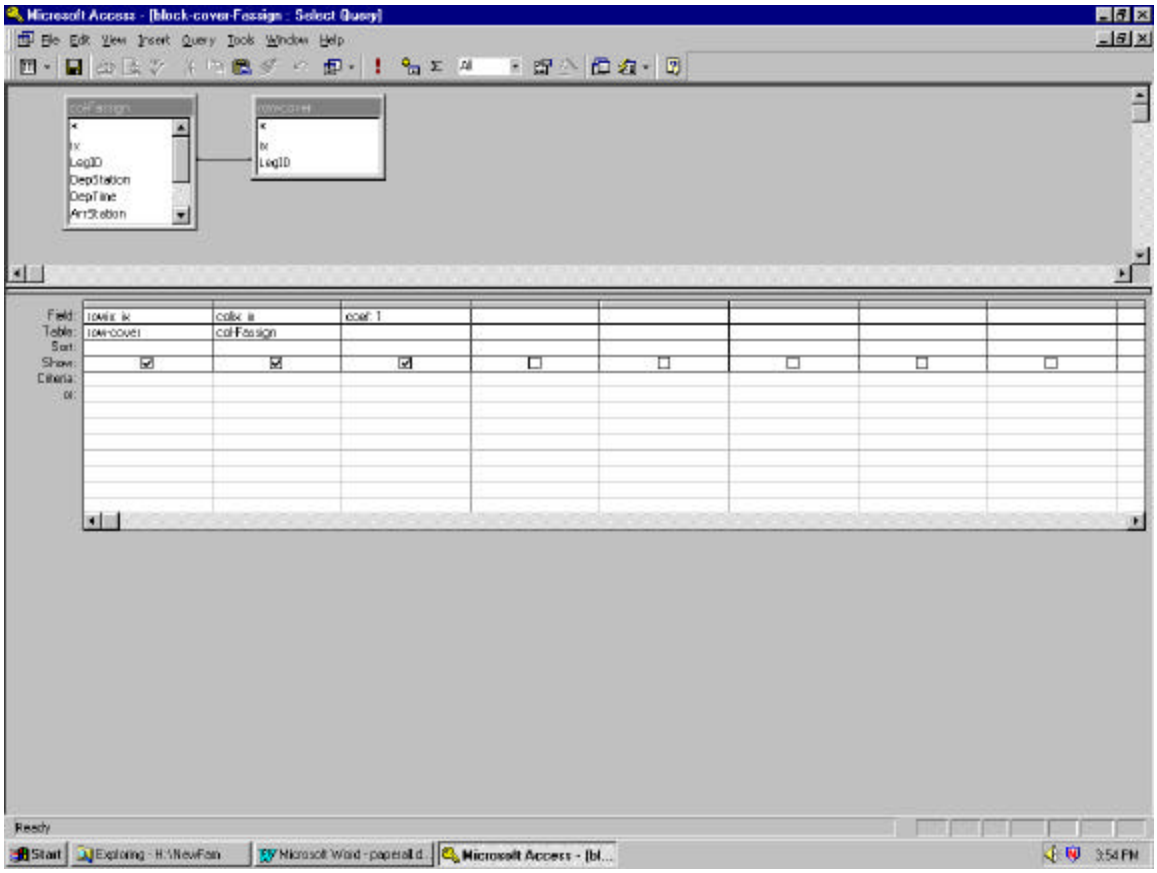
We took advantage of the fact that Access is a relational database. Therefore, we designed the model in the sense of relational modeling. For a detailed description of modeling in this environment, see [AJLS]. First we grouped variables in column strips. As mentioned above, Fassign and Garc are the two main sets of variables. Thus, our two column strips are Fassign and Garc. Col-Fassign is a table which contains the following information: Index, Leg ID, Departure Station, Departure Time, Arrival Station, Arrival Time, Fleet Type and Net Revenue. Again, Arrival Time here is really Ready Time. Garc is a table named alltime in Access. Alltime contains the following information: Index, Station, Time, and Fleet Type. There is a slight difference between the AMPL model and the Access model. In the Access model, we added another constraint, maintenance feasibility. If the length of a ground arc is more than 5 hours, we called it a maintenance feasible ground arc. Each type of constraint is a row strip. Therefore, the row strips are Cover, Pcount, Balance, and Maintenance. Row-Cover is a table with two fields, Index and Leg ID. Row-Pcount is a table with three fields, Index, Fleet Type and Number of Fleets. As for Balance, it is also a table name alltime. Row-Main-Sta-Eqp has three fields: Index, Station, Fleet Type. All the tables mentioned above do not allow duplicated records and all Index fields start with 1.

	Fassign	Garc	
Cover			= 1
Pcount			≤ Number of Planes
Balance			= 0
Maintenance			≥ (1/2) Number of Planes

The above rectangle represents the problem matrix. We divided up the matrix into eight blocks (sub-matrices). Each block represents the intersection between a row strip and a column strip. All the "block tables" have three fields, which are Row Index, Column Index and Coefficient. Usually, we use queries instead of tables (with exception of some more complicated blocks) for the "block tables". It is more dynamic this way because whenever we change something in a "row table" or "column table", queries will automatically incorporate the changes into the appropriate "block table". Otherwise, we would have to run a macro to make the changes every time we change something in the "row tables" or "column tables".

For each block, we find the join of a row strip and a column strip. Then, we get the Row Index from the Index of a row strip, Column Index from the Index of a column strip, and set the coefficient. For example, we get Block-Cover-Fassign when the field, Leg ID, in Col-Fassign and Row-Cover are equal. We set the coefficient to be 1. Thus, we got a table with all non-zero entries in the sub-matrix block. These are the basic techniques on how to get blocks in Access. Some of the blocks are not as easily obtainable. In that case, we break it down into different steps and then merge the tables we created together. Note that there exists no intersection between the column strip Garc and the row strip Cover. Nor does there exist an intersection between the column strip Fassign and the row strip Maintenance. Therefore, Block-Cover-Garc and Block-Main-Fassign are both zero sub-matrices.

The following figure shows what Block-Cover-Fassign looks like in Access.



After getting all the blocks, we need to offset the blocks. For example, we add the number of Fassign columns to the Column Index of Block-Pcount-Fassign, and we also add the number of Cover rows to the Row Index of the same block. After offsetting all the blocks, we merge them together into the table Result, which represents the big problem matrix we are going to pass to the solver.

Here is how we got the objective function. We used the Index in Col-Fassign as the Column Index and the corresponding Net Revenue as Coefficient for the objective function. After completing the objective function and the problem matrix, it's time to work on the right hand side of the equations. Table Rhs has four fields: From Row Index, To Row Index, Sense, Value. When we have a record like

FROM ROW INDEX	TO ROW INDEX	SENSE	VALUE
1	66	E	1

It means that from row 1 to row 66 we have equation sense equal and the right hand side value is 1. We get Rhs table by counting up the number of rows in each row strips and sum them up in a way that will match the offset of the problem matrix.

As before, we developed a special purpose problem solution code to provide the interface required between the data in Access and an optimizer. And as before, we selected CPLEX as the optimizer.

3.3 Data Structures: One of the main differences between the crew scheduling model and the fleet assignment model is that the crew scheduling model uses Access to store its data, but uses a C program to manipulate its data while the fleet assign model uses Access to store and manipulate its data.

There are five different sets of tables/queries in the fleet assignment model. The first set of the tables is instance tables which are provided by the user on a specific instance. These include a schedule table, a table with stations and their corresponding time zone information, a table with equipment and its turn time information, a table with stations and their turn time information, and a table with fleet types and their maintenance stations. The schedule table should contain the following information: Leg ID, Departure Station, Departure Time, Arrival Station, Arrival Time, Fleet Type, and Net Revenue. Here, all the time fields are the local time to its station and Arrival Time is the real arrival time. We don't change the departure time or arrival time until later when we run the time zone and turn time adjustment queries.

The second set of tables is the model tables. The tables' structures are identical to the instance tables' structures. The reason for having both instance tables and model tables is that we can create another instance and run the model again without changing anything in the model. Also, we can modify the records in the model tables without altering the instance tables.

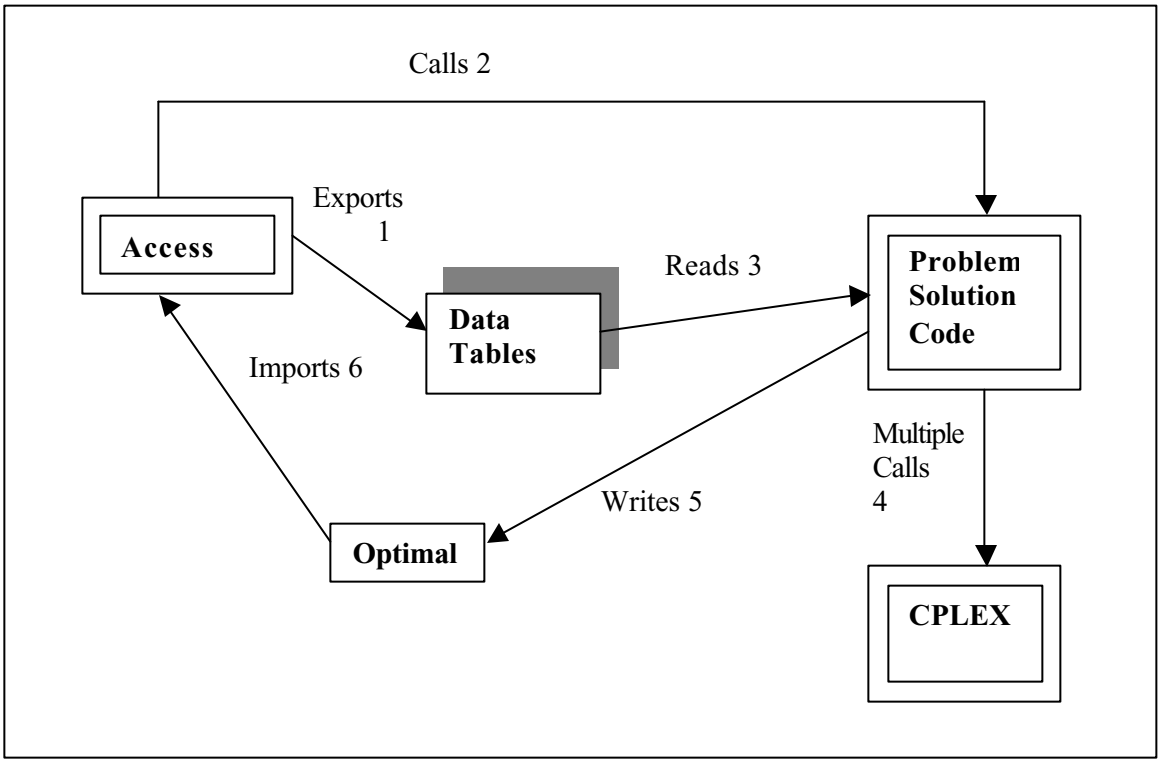
The third set consists of column strips, row strips, and block tables/queries. The fourth set is the operations queries. They consist of append queries, delete queries, make table queries, and update queries. We use these queries to manipulate data and then store the revised information into the third set of tables/queries. The third and fourth set of table/queries is the essential part of relational modeling. By using them, we are able to turn the instance into a solvable form.

The last set of tables is the result tables. They include the ASCII format table we pass to the solver and the ASCII format table we retrieve from as the optimal solution.

3.4 Application Integration: The application Integration for fleet assignment model is similar to crew scheduling model but yet simpler. Like crew scheduling model, we used Access as the primary application. It calls the problems solution codes. Then the problem solution code, in turn, calls CPLEX. Unlike crewing scheduling model, fleet assignment model manipulate data in Access. This made the application integration simpler.

As we see in Figure 3, Access exports data tables including the problem matrix, right hand side, objective function, and etc. (step 1) and calls problem solution code (step 2). The problem solution code read from data tables (step 3) and calls CPLEX which will set up, load, and solve the problem, and then retrieve the solution to the problem (step 4). Then, the problem solution code will writes the optimal data (step 5) which Access imports into the optimal table (step 6).

Figure 3: Data Flow Diagram for Fleet Assignment Model



Appendix A: AMPL Model

```
#leg fam

set SCHED dimension 6;
set LEGID := setof {(l, ds, dt, as. at, eqp) in SCHED} l;
set STATION := setof {(l, ds, dt, as. at, eqp) in SCHED} ds;
set EQUIP := setof {(l, ds, dt, as. at, eqp) in SCHED} eqp;
set STA_EQP := setof {(l, ds, dt, as. at, eqp) in SCHED} (ds, eqp);
set LEG_EQP := setof {(l, ds, dt, as. at, eqp) in SCHED} (l, eqp);
set REDEYE within SCHED;
set ALLTIME {STA_EQP} circular;

param cost {LEG_EQP};
param leg_rev {LEG_EQP};
param num_plane {EQUIP};

var Fassign {SCHED} >=  0;
var Garc ( (s, e) in STA_EQP, t in ALLTIME[s, e] } >= 0;
var Revenue;
var Totcost;

maximize Profit:
        Revenue – Totcost;

subject to Addrev:
        Revenue – sum {(l,ds,dt,as,at,e) in SCHED } leg_rev[l,e] * Fassign[l,ds,dt,as,at,e] = 0;

subject to Addcost:
        Totcost - sum {(l,ds,dt,as,at,e) in SCHED } cost[l,e] * Fassign[l,ds,dt,as,at,e] = 0;

subject to Cover {l in LEGID}:
        sum {(l,ds,dt,as,at,e) in SCHED } Fassign[l,ds,dt,as,at,e] = 1;

subject to Pcount {e in EQUIP}:
        sum {(r,ds,dt,as,at,e) in REDEYE } Fassign[r,ds,dt,as,at,e]
        + sum { (s,e) in STA_EQP} Garc [s, e, last(ALLTIME[s,e])]
        <= num_plane[e];

subject to Balance { (s,e) in STA_EQP, t in ALLTIME[s,e]}:
        Garc[s,e,t] = Garc[s,e,prev(t)] + sum {(l,s,t,as,at,e) in SCHED } Fassign[l,s,t,as,at,e]
        - sum {(l,ds,dt,s,t,e) in SCHED } Fassign[l,ds,dt,s,t,e] = 0;
```

References:

[AGPT] Anbil, R., E. Gelman, B. Patty, and R. Tanga, "Recent Advances in Crew-Pairing Optimization at American Airlines," *Interfaces* **21**, No. 1, 62-74 (1991).

[AJLS] Atamturk, A., E. Johnson, J. Linderoth, and M. Savelsbergh, "ARMOS: A Relational Modeling System", working paper, School of Industrial and Systems Engineering, Georgia Institute of Technology (1996).

[ATJ] Anbil, R., R. Tanga, and E. Johnson, "A global approach to crew-pairing optimization," *IBM Systems Journal* **31**, No. 1, 71-78 (1992).

[CHJN] Clarke, L., C. Hane, E. Johnson, and G. Nemhauser, "Maintenance and crew considerations in fleet assignment," *Transportation Science* **30**, No. 3, 249-260 (1996).

[Hu] Hu, J. "Solving linear programs using primal-dual subproblem simplex method and quasi-explicit matrices," Ph.D. Thesis, School of Industrial Systems Engineering, Georgia Institute of Technology (1997).

---

[1] Microsoft® Access™ is a registered trademark of Microsoft Corporation.
[2] CPLEX® is a registered trademark of ILOG, Inc.
[3] AMPL was developed at AT&T Bell Laboratories and is licensed through Scientific Press.