12:08:09 OCA PAD AMENDMENT - PROJECT HEADER INFORMATION 12/17/93

Active

Project #: E-24-698 Cost share #: Rev #: 2
Center # : 10/24-6-R7641-0A0 Center shr #: OCA file #:
Work type : RES
Contract#: AGREEMENT SIGNED 9/11/92 Mod #: LTR DT 10/14/93 Document : AGR
Prime #: Contract entity: GTRC

Subprojects ? : N CFDA: N/A
Main project #: PE #: N/A

Project unit: ISYE Unit code: 02.010.124
Project director(s):
NEMHAUSER G L ISYE (404)894-2306

Sponsor/division names: NORTHWEST AIRLINES / ST PAUL, MN
Sponsor/division codes: 213 / 038

Award period: 921005 to 940331 (performance) 940331 (reports)

Sponsor amount	New this change	Total to date
Contract value	0.00	25,000.00
Funded	0.00	25,000.00
Cost sharing amount		0.00

Does subcontracting plan apply ?: N

Title: COLUMN GENERATION FOR AIRLINE PROBLEMS

PROJECT ADMINISTRATION DATA

OCA contact: Ina R. Lashley 894-4820

Sponsor technical contact

THERESA HULL OR JANE BECKEN
(000)000-0000

PROCESS IMPROVEMENT & PERFORMANCE
N7140
NORTHWEST AIRLINES
5101 NORTHWEST DRIVE
ST. PAUL, MN 55111-3034

Sponsor issuing office

ELROY M. OLSON
(612)726-6849

DIRECTOR, PIP N7140
NORTHWEST AIRLINES
5101 NORTHWEST DRIVE
ST PAUL, MN 55111-3034

Security class (U,C,S,TS) : U ONR resident rep. is ACO (Y/N): N
Defense priority rating : N/A N/A supplemental sheet
Equipment title vests with: Sponsor X GIT
NONE PROPOSED.
Administrative comments -
LTR DTD 10/14/93 AUTHORIZES AN EXTENSION TO 3/31/94.

SR 197

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 09/30/94

Project No. E-24-698 Center No. 10/24-6-R7641-0A0

Project Director NEMHAUSER G L School/Lab ISYE

Sponsor NORTHWEST AIRLINES/ST PAUL, MN

Contract/Grant No. AGREEMENT SIGNED 9/11/92 Contract Entity GTRC

Prime Contract No.

Title COLUMN GENERATION FOR AIRLINE PROBLEMS

Effective Completion Date 940331 (Performance) 940331 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	
Final Report of Inventions and/or Subcontracts	Y	
Government Property Inventory & Related Certificate	N	
Classified Material Certificate	N	
Release and Assignment	N	
Other	N	

Comments

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Managment	Y
Research Security Services	N
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other	N
	N

NOTE: Final Patent Questionnaire sent to PDPI.

Georgia Tech

School of Industrial and Systems Engineering

Georgia Institute of Technology
Atlanta, Georgia 30332-0205
USA
404•894•2300
404•894•2301 FAX

August 12, 1994

Mr. Elroy M. Olson, Director
Northwest Airlines
Dept. E7140
5101 Northwest Drive
St. Paul, MN 55111-3034

Dear Oley:

Enclosed is the official final report on the Crew Recovery project as required by our contractual agreement. We found this project to be very stimulating and we appreciated your cooperation and support. We hope that you found the results to be beneficial.

As you know, we are working on further developments of the software and we hope to continue the collaboration even though no funding is available now. The current prototype software needs to be improved in several ways to improve its reliability and robustness. Our goal is to achieve this by the end of the year.

Ellis tells me that a visit is planned for late August or early September. Perhaps I'll see you then.

Sincerely,

George L. Nemhauser
Institute Professor and Chandler Chair

GLN/yk
Enclosure
cc E.Johnson
T. Wise
P. Mydell
OCA ✓

Final Report to Northwest Airlines

on

The Crew Recovery Problem

Ellis L. Johnson, Ladislav Lettovsky, George L. Nemhauser,

Ram Pandit, and Steven Querido

Logistics Engineering Center

Georgia Institute of Technology

August 12, 1994

1 Introduction

A crew pairing is a sequence of flight legs flown by a particular crew subject to a set of legal and contractual restrictions. Crew pairings must start and end at the same airport, called the crew's "domicile" and, on U. S. domestic routes, they typically last up to five days. The objective of the crew pairing problem is to find a set of pairings that fly each flight segment once and do so at a minimum cost. After this set of pairings have been chosen, crews are assigned to pairings via a bidding process. When a crew has been assigned to fly a pairing, it will usually receive full credit for having flown that pairing, whether or not this is the case.

A solution to the crew pairing problem gives a plan for the airline to follow. However, unforeseen events, such as mechanical and weather problems, force the airline to deviate from the plan. These deviations cost the airline in a variety of ways, in both "hard" and "soft" dollars.

Consider the following example. A crew arrives late at an airport and fails to make a connection on time. In response to this situation, the airline has the following possible responses. First, it can delay the departure of the connection. There is a cost associated with doing so. This may include extra crew costs since the crew is flying longer than originally scheduled, added fuel costs to try to catch up to the schedule, and customer dissatisfaction. The airline may try to find a different crew that can make the departure. This may involve using a reserve crew or a crew in the middle of a trip. In either case, the airline may have to pay for the extra work. Finally, the airline may choose to cancel the flight. This is usually the most expensive option since now the crew and the aircraft are out of position to continue their trip and there will be more customer dissatisfaction.

The problem we consider in this report is how to repair small deviations with minimum additional cost. The cost of a pairing is measured in hours, i.e. a pilot will be paid x hours for flying some pairing. "Pay-and-credit" or "flight time credit" is the difference between the number of hours a pilot is paid versus the number of hours flown and is usually expressed as a percentage. In some months pay-and-credit can be as high as 7%, which translates into millions of dollars. Our objective is to decrease pay-and-credit by providing better recovery

solutions.

The methodology we describe is not applicable to cases where there are a large number of delayed flights. In addition, our definition of crews is limited to pilots. The problem is somewhat different for flight attendants since they are more flexible with respect to fleet type. However, the same general methodology is applicable to flight attendant recovery.

Section 2 describes methods for recovering from a single delayed flight at a single airport. The basic idea is to generate new pairings that involve the segments of the delayed crew and any other crews that might be available to handle the missed connection. Reserve crews or deadheading are also considered. A "small" covering problem is then solved to find a minimum cost set of new pairings. Section 3 gives some examples to illustrate the results. Section 4 includes our recommendations for further research. The appendix contains the data structures used in this implementation.

2 Outline of Crew Recovery Procedure

We first give a brief overview of each section of the procedure, followed by a more complete algorithmic description.

1. *Read Problem Data.* The first section of the program prompts the user for the data required to run the problem. These data include the airport where the delay occurs, the flight number of the delay, the length of the delay, whether the delay is an arrival or departure and the additional block time in the delay. In addition, since the prototype runs in isolation from any other developed system, the user needs to enter the day and time that a crew coordinator would find out about the delay.

2. *Initialize.* This section of the program requires the complete schedule for all fleet types for a given month, the crew pairings for a particular fleet for that month and the airports in the schedule for that month. It reads these data and creates a "timeline" network of the schedule. A timeline for an airport marks all of the departures and arrivals at that airport.

3. *Update Network.* In this step we update the timeline network to account for the delay.

4. *Identify Potential Crews.* We use a simple approach to identify crews that may be used to help solve the problem. We build a time window starting from when the crew coordinator learns about the delay and ending at a user specified time. Any crew that flies the same fleet type as the delayed crew and falls into this time window at the airport where the delay occurs is used to help recover the delay.

5. *Find Reserve Crews.* If the delay is not at a crewbase, we look for a deadhead that will get a reserve crew from a crewbase to the airport where the delay occurs by the time the delayed crew's flight is supposed to depart.

6. *Generate Pairings.* Each crew we have selected to help recover the delay is in the middle of a pairing. The flight legs remaining from each crew's pairing, including the delayed crew, are used to generate pairings to get each crew back to its crewbase. The pairings are generated using complete enumeration.

7. *Check Legality and Calculate Cost.* Although each pairing generated in the previous step is legal in isolation, each crew has a different flight history over the past few days. In this section, we create for each crew - pairing combination, a new pairing consisting of what the crew flew before the delay and the pairing generated above. We first check if this new pairing is legal. If so, we calculate its cost.

8. *Deadheading the Delayed Crew.* Another way to rectify the delay is to use a reserve crew to fly the legs that the delayed crew will miss and to deadhead the delayed crew to a point where it can meet the reserve crew and finish its pairing. In this section, we generate pairings of this type.

9. *Set Up and Solve Set Covering Model.* In this step, we write an MPS file of a set covering model to cover all of the flights by a set of pairings. We call MINTO to solve the set covering problem. If the problem is infeasible and the delayed crew can legally

fly its next flight, we assume that the flight is delayed and is flown by the delayed crew. We then try to recover at the destination airport of this flight.

2.1 Read Problem Data

The user is prompted for information and information is received via standard output/input. At this part of the program we allow the user to choose any or all of the following scenarios:

1. Find a solution that does not allow the use of reserve crews and allows crews to deadhead only on other flights originally flown by other crews in the problem.

2. Find a solution that allows the use of reserve crews and allows crews to deadhead only on other flights originally flown by other crews in the problem.

3. Find a solution that allows the use of reserve crews and allows crews to deadhead on any flight in the schedule.

The user's choices are kept as an integer whose binary representation contains a 1 in the i^{th} place if scenario i is chosen.

2.2 Initialize

Currently, there are three data files supplied by Northwest Airlines. These are solution.320, the crew pairings for the A320 fleet, schedule.oct, the schedule for October, 1991, and stations.oct, the airports served by Northwest Airlines in October, 1991. There is also a parameter file which gives the values of various legality and cost parameters. The crew pairing file denotes each pairing by listing the sequence of flight numbers in each duty period. This information is stored in the "pairing" array. The schedule file lists all of the flights by flight number with its origin, departure time, destination, arrival time, and the previous and next flight in the aircraft rotation. This information is stored in the "flt-leg" array. The airport file lists each airport with its time zone as an offset from Greenwich Mean Time (GMT). This information is stored in the "airport" array. In addition to the values of cost and legality

rule parameters, the parameter file lists which airports are crewbases and co-terminals, and which airports require extra connect time for crews between flights.

Once these data have been read, we develop for each airport a timeline marking all of the arrivals and departures into that airport. The entries on these timelines are stored in the "tm-line" array. This array is sorted primarily with respect to the airport and secondly with respect to the time of the event, arrival or departure. Next we create a daily network of these timeline entries by marking the predecessor and successor of each entry. The predecessors and successors are marked by their index in the "tm-line" array. We find predecessors and successors using the pairing array to determine the flight number of the next flight in the pairing. This does not always work, however, due to the fact that some through flights are listed only by their origin and final destination and do not show any intermediate points. To determine which is the right predecessor/successor we try to find both through flights and flights with the flight number of the next flight in the pairing. If both are found, we assign the through flight to be the predecessor/successor.

2.3 Update Network

The timeline network created in the previous step is based on the original schedule, which is no longer accurate. In this step we change the time of the delayed timeline entry and then resort the "tm-line" array by time. After this has been done, we determine if recovery is necessary, e.g., if a delayed arrival is the last flight segment of a pairing, recovery is not needed. We also check if a delayed crew can catch its next flight or if the delayed crew's flight is the last flight of a duty period. If the flight is the last of a duty period, we first check if the overnight rest is sufficient. If it is not, we store details of that problem scenario and run it after we reinitialize the timeline network. It is necessary to rebuild the timeline network because we have a created a daily model. Since the beginning of a new duty period starts after the delay, we rebuild the network so the first delay will not affect this new problem.

2.4 Identify Potential Crews

In this section we have decided to use a simple approach to identify crews. We create a time window at the airport where the delay occurs. All crews of the same fleet type that arrive or depart in that time window will be used to help solve the problem. Care must be taken so as not to allow the same crew to be chosen twice. For example, a crew that arrives and departs within this time window will appear twice, however, we want one only record of this crew.

As we choose crews, we gather statistics on that crew's pairing history, such as the crew's base, its pairing and current duty period and the amount of flight and elapsed time so far in this duty period. We also mark the day and time this crew is supposed to finish its pairing. This information is stored in the "crew" data structure. While gathering these data, we calculate the cost of the crew's original pairing and store each duty period or partial duty period flown in the "duty period" data structure.

2.5 Find Reserve Crews

We assume that reserve crews are available as needed. We realize that this assumption is unrealistic, however we do not have information about the availability of reserves. If the delay occurs at a domicile, we assume that there will be reserves available. If the delay is not at a crewbase, we try to find a deadhead that will bring in a reserve crew early enough to cover any missed departures. If we cannot deadhead a reserve crew in time, we cannot generate pairings by deadheading the delayed crew as in Step 8.

2.6 Generate Pairings

In general, at this stage of the procedure, we are not generating true pairings because the crews have different crewbases and the airport where the delay occurs may not be a crewbase. However, we are generating sequences of flights that will return a crew to its domicile. We call these "half-pairings".

We have a list of flight segments to be covered. These flights are the flights left to be flown by the crews we have selected to help us solve the problem. The flight segments are dated, i.e., each flight has to be flown on a particular day, which cuts down the number of possible connections. This allows us to use complete enumeration to generate pairings. If the flight segments were not dated, complete enumeration would generate too many pairings and increase CPU time needed to solve the problem. In the complete enumeration, if we have a sequence of flights that does not end at a domicile, we attempt to find a deadhead that brings the pairing to an end at a crewbase. When complete, we have a set of legal half-pairings that go from the airport where the delay occurred to all of the crewbases.

2.7 Check Legality and Calculate Cost

If each crew were to start fresh and fly one of the half-pairings generated above, no legal restrictions would be violated. However, this is not the case. In this part of the procedure, we combine the part of the original pairing that each crew has already flown with each half-pairing and check the new pairing to make sure all legalities are upheld. The following rules are checked:

- The crew must arrive at its domicile.
- The crew must arrive at the station where the delay occurs before the departure time of the half-pairing under consideration.
- The crew must arrive at its domicile before midnight of its last day or eight hours after its scheduled arrival time, whichever is greater.
- If the crew is taken from the middle of a duty period, the duty period made from its original duty period and the first duty period of the half-pairing must not violate any duty period restrictions.
- All overnight rests must be sufficient.

If all of these conditions are met, we say that the crew can fly this half-pairing and we calculate the cost at which it can do so. We do so by figuring the cost of the pairing comprised of what the crew originally flew up to the delay together with the half-pairing. We compare this cost to the cost of the crew's original pairing. If the cost of the new pairing is greater than that of the original pairing, the cost of the half-pairing is the difference, otherwise the cost is zero. In an effort to make the optimizer reassign crews to their original pairings, when a half-pairing is the end of the crew's original pairing, we assign it a small negative cost.

2.8 Deadheading the Delayed Crew

A quicker and possibly cheaper way to recover the delay is to assign a reserve crew, if one is available, to fly the flights that the delayed crew will miss while at the same time deadheading the delayed crew to a point where it can continue it pairing. This has the added benefit of not affecting any other crews. However, it may be desirable to avoid assigning reserves to this type of trip.

To generate pairings this way, we look for deadheads for the delayed crew from the airport where it is delayed to each other airport on the rest of its pairing. If the delayed crew can be deadheaded to an airport where it can legally continue its pairing, we generate two pairings, one for the delayed crew and one for the reserve. In general, these pairings are relatively cheap; the delayed crew arrives back at its base at the same time it was expected to and there is no additional cost. The cost of the reserve crew's pairing depends on the length of its trip, but in some cases it is not much more than the daily minimum guarantee.

2.9 Set up and Solve Set Covering Model

To choose the minimum cost set of half-pairings that cover all of the flights, we generate and solve a set covering problem. We define the following parameters and variables:

$I \equiv$ the set of flight legs to be covered,

$J \equiv$ the set of generated pairings,

$K \equiv$ the set of crews available,

$K_1 \equiv$ the set of crews such that the delay occurs at their crewbase,

$K_2 = K \backslash K_1 \equiv$ all other crews in K,

$F_j \equiv$ set of crews that can fly pairing j,

$H_k \equiv$ set of pairings that can be flown by crew k,

$c_{kj} \equiv$ cost of assigning crew k to pairing j,

$d_i \equiv$ displacement cost of using flight i for deadheading,

$g_k \equiv$ cost of keeping crew k at its base, for $k \in K_1$,

$a_{ij} = 1$ if flight i is in pairing j, 0 otherwise.

$y_{kj} = 1$ if crew k is assigned to pairing j, 0 otherwise,

$r_k = 1$ if crew $k \in K_1$ stays at its base, 0 otherwise,

$s_i \equiv$ number of crews deadheading on flight segment i.

$t_i \equiv 1$ if flight i is contained in no pairing, 0 otherwise.

$M \equiv$ high cost to indicate infeasibility.

We formulate the crew recovery problem:

$$
\begin{aligned}
\min \quad & \sum_{k\in K}\sum_{j\in J} c_{kj}y_{kj} + \sum_{k\in K_1} g_k r_k + \textstyle\sum_{i\in I} d_i s_i + \sum_{i\in I} M t_i & \\
& \sum_{j\in J}\sum_{k\in F_j} a_{ij}y_{kj} - s_i + t_i = 1 & \forall i \in I, \\
& \sum_{j\in H_k} y_{kj} + r_k = 1 & \forall k \in K_1, \\
& \sum_{j\in H_k} y_{kj} = 1 & \forall k \in K_2, \\
& y_{kj} \in \{0,1\} & \forall k \in K, \forall j \in J, \\
& s_i \geq 0 & \forall i \in I, \\
& r_k \geq 0 & \forall k \in K_1.
\end{aligned}
$$

The first set of constraints guarantees that all of the flight legs are covered at least once if it is feasible to do so. The second set of constraints says that for each crew at its crewbase, we have the option of sending that crew home early. For crews not at their crewbase, the third set of constraints forces us to assign that crew to a pairing, thus sending that crew back to its crewbase.

We solve this set covering problem using MINTO. If any of the t_i variables are positive, the current solution is infeasible. If this is the case, we delay the departure of the delayed crew's next flight and try to find a recovery at the next airport following the same procedure described above.

All of the steps listed above are easily generalized to include more than one delay at an airport. The only difficulty that can arise with more than one delay is if in the optimal solution to the set covering problem there exists some nondelayed flight i such that $t_i > 0$. In this case it is not clear how to determine which delays have been recovered and which have not.

3 Sample Scenarios

The Crew Recovery System was tested on different scenarios. A sample of four tested scenarios and their solutions are presented along with the output provided be the program:

Scenario 1: *Flight 111 delayed on arrival to MSP, on Monday, by 1:00 hour*

No feasible solution without a reserve crew. Reserve crew is used for the rest of the pairing (MSP 662 PHL 663 MSP).

Scenario 2: *Flight 190 delayed on departure from LAX, on Tuesday, by 1:00 hour:*

SOLUTION 1

There is no solution to the problem that covered all of the flight legs and that allowed both the use of reserve crews and deadheading on any flight. A legal option at this point is to delay the departure of flight 154 by 35 minutes and attempt to recover the delay at MSP. We will run this option ....This is the last flight segment of a pairing, no changes are necessary.

Scenario 3: *Flight 1852 delayed on arrival to BOS, on Friday, by 1:00 hour:*

SOLUTION 2

This solution allows the use of reserve crews. It allows deadheading only on the other flights in the problem. Note that the use of reserve crews when the delay does not occur at a crewbase includes the use of any deadhead to get the reserve crew to the delay.

This solution costs 5246.450000 and makes the following assignments:

Crew 1: 105 -1 316 315

Crew 2*: 351 368 -1 579 351 -1 928 -1 118 285

Crew 3*: 3 259 -1 790 583 12 -1 111 662 663

Crew 4*: 1865 1870 1875 -1 1850 351 368 -1 579 351 -1 928 -1 118 285

Crew 5*: 105 457 -1 553 553 -1 986 550 -1 420 579 -1 158

Crew 6: 385 330 -1 103 -1 118 285

An asterik (*) denotes that a crew stays on its original pairing.

Scenario 4: *Flight 722 delayed on arrival to BOS, on Thursday, by 0:50 hour:*

SOLUTION I - delay the departure of 105 (next flight), the crew has time to connect to flight 457 in MSP.

SOLUTION 2 - no delay, no reserve crew, no deadhead: this solution does not allow the use of reserve crews. It allows deadheading only on the other flights in the problem. This solution is infeasible. Below is the solution with the fewest infeasibilities. Below that are the missing flights.

Crew 1: 1875 -1 1850 351 368 -1 579 -1 579 -1 158

Crew 2: 105 457 -1 553 553 -1 986 550 -1 420

An asterisk denotes that a crew stays on its original pairing.

These flights are not covered by the above solution:

Day 4 Flight 1865 BOS 14:00 DCA 15:36

Day 1 Flight 285 DCA 6:55 DTW 8:30

Day 4 Flight 1870 DCA 16:50 BOS 18:18

Day 6 Flight 351 DTW 12:20 SFO 14:36

Day 7 Flight 118 MSP 20:30 DCA 23:43

Day 7 Flight 928 SFO 1:20 MSP 6:51

SOLUTION 3 - this solution allows the use of reserve crews and allows deadheading on all flights in the schedule.

This solution costs 1480.70 and makes the following assignments:

Crew 1: -1 553 553 -1 986 550 -1 420 579 -1 158

Crew 2*: 767 197

Crew 3*: 198 289 -1 285 -1 198 767 197

Crew 4*: 136 475 20 -1 555 555 -1 190 154

Crew 5*: 153 -1 20 636 -1 345 30 361

Crew 6*: 105 -1 260 260 -1 153 153 -1 20 636 -1 345 30 361

Crew 7*: 45 -1 330 390 -1 3 259 -1 790 583 12 -1 111 662 663

Crew 8*: 315 -1 1282 189 -1 550 986 174

Crew 9: 197 280

Crew 10*: 118 285

Crew 11*: 772 771

Crew 12: 457

An asterisk denotes that a crew stays on its original pairing.

3.1 Remarks

The computational testing shows that for an optimized schedule there is little, if any, scope for swapping pairings or finding an "idle" crew with a sufficient time window in which a misconnected flight could be covered. When deadheading is allowed and r eserve crews are available a number of solutions are generated and the optimal solution is displayed. The produced solution, however, often requires major changes in pairings and may be unacceptable from operational point of view. An estimate of the cost of disrupting a pairing and/or a threshold on the maximum number of disrupted pairings would allow a tradeoff between the optimal cost and fewer alterations.

4 Recommendations for Further Research

We have two major recommendations on how this system can be improved. First, the timeline network needs to be expanded from a daily network to one that starts the day the delay occurred and moves out about five days, the length of the longest pairing. This will allow the system to think of the same flight on different days as different flights. This is important because once a delay is noted, the timeline network is changed. For example, if flight 123 out of MSP is delayed on Monday, the system will think that flight 123 is delayed every day. This is a result of using a daily timeline network. Changing to a weekly network will also simplify many of the legality checks, since the day and time of the flight are specified by referring to a timeline index number. Next, the logic for determining which delays have been recovered in the multi-delay case should be worked out fully and implemented. This will make the cancellation case easier since a cancellation is similar to delays at two airports.

The biggest problem with the current system is that the aircraft problem is ignored completely. If the a crew is delayed arriving into an airport, its plane is also delayed and there may not be anything a reserve crew could do because there is no airplane for them to fly. Throughout our study, we have focused on the crew side assuming that the aircraft will be worked out. It may be a better idea to try to work out both simultaneously to recover more efficiently.

Appendix: Data Structures

We have implemented the system described above in UNIX/C. Below are the data structures used.

```
typedef struct FLIGHTS
{
   int number;            /* flight number */
   char dep_st[4];        /* departure station */
   int dep_tm;            /* departure time */
   char arr_st[4];        /* arrival station */
   int arr_st_no;         /* number of the arrival station */
   int arr_tm;            /* arrival time */
   int time;              /* elapsed time of flight */
   int pairing;           /* pairing number in which this appears */
   int day[8];            /* 1 => day the flight is on; 0 => not available */
   char equipment[4];  .../* fleet type used */
   int tail_num;          /* tail number of the plane */
   int duty_period;       /* duty period in the pairing this leg appears */
} FLT_LEG;

typedef struct TIMELINES
{
   int number;            /* flight number */
   int index;             /* index of flight in flt-leg array*/
   int time;              /* time of departure or arrival */
   int pairing;           /* pairing it belongs to */
   int duty;              /* duty period it belongs to */
   int code;              /* +1 for arrival, -1 for departure */
```

```
   int prev;              /* the previous tm-line no */
   int succ;              /* the next tm-line no */
   char equipment[4];     /* fleet type used */
} TMLINE;

typedef struct DUTY_PERIOD
{
   char dep_st[4];        /* departure station */
   int dep_tm;            /* departure time */
   char arr_st[4];        /* arrival station */
   int arr_tm;            /* arrival time */
   int day;               /* day the duty period in on */
   double cost;           /* cost, in dollars */
   int num_segs;          /* number of flights in this duty period */
   int segs[10];          /* list of flights in the duty period */
} DUTY;

struct PAIRINGS
{
   int number;            /* pattern number */
   char base[4];          /* crewbase */
   int start_date;        /* date on which the pairing started */
   int end_date;          /* date on which the pairing ends */
   int tafb;              /* time away from base */
   double sum;            /* sum of duty period costs */
   int num_duty;          /* number of duty periods */
   int cost;              /* cost of the pairing */
   int complement;        /* size of crew */
```

```
   int first;              /* the index of the first tm-line in this pairing */
} pairing[MAX_PAIRING];

typedef struct NEW_PAIRINGS
{
   int no_dps;             /* the number of duty periods in this pairing */
   int dps[9];             /* a list of the duty periods in this pairing */
   int deadhead;           /* 1 if this pairing requires deadheading, 0 otherwise */
   double cost[MAX_CREW]; /* the cost of the pairing, in dollars, for each crew.
                             cost = -1 => crew cannot fly this pairing  */
   int bonus[MAX_CREW];    /* 1 if this pairing is the remainder of a crew's original
                             pairings */
} PAIR;

struct PAIR_FLT_SEGS
{
   int number;             /* list of flt segs */
   int duty_period;        /* duty period in which it appears */
} pair_flt-leg[MAX_SEG];

typedef struct CREW_INFO
{
   int prev_pairing;
                           /* the number of the original pairing that this crew flew
   double orig_pair_cost; /* cost of original pairing */
   int flt_hrs;            /* number of hours flown today */
   int elap_hrs;           /* number of duty hours today */
   int start_time;         /* time of last event */
```

```
   int start_day;          /* day of last event */
   int last_day;           /* last day of current pairing */
   int last_time;          /* time at which the original pairing ends */
   char base_st[4];        /* name of the base */
   int at_base;            /* 1 if disruption occurs at this crew's base */
   int reserve;            /* 1 if reserve crew */
   int deadhead[2];        /* the flight index of the crew's deadhead */
   int last_tmline;        /* the last tm-line this crew flew */
   int first;              /* Index of this crew's first duty period */
   int last;               /* Index of this crew's last duty period */
} CREW;

typedef struct STATIONS *APTR;

typedef struct STATIONS
{
   char name[4];           /* name of airport */
   int off_hr;             /* time_zone */
   int off_mnt;
   int extra_connect;      /* addidtional connect time required */
   int num_coterm;         /* number of coterminals */
   APTR coterm1;           /* pointers to those coterminals, if any */
   APTR coterm2;
} AIRPRTS;

typedef struct OTHER_PROBLEMS
{
   int stat_no;
```

```
    int delay_no;
    int delay_len;
    int day;
    int scenarios;
} SAVED_PROB;
```