

MIT

FTL REPORT R85-4

**FTL COPY, DON'T REMOVE
33-412, MIT 02139**

A.I.S.E. A HYBRID SCHEDULING SYSTEM

DEPARTMENT
OF
AERONAUTICS
&
ASTRONAUTICS

Herve J. Duchesne de Lamotte

FLIGHT TRANSPORTATION
LABORATORY

Cambridge, Mass. 02139

May 1985

FTL REPORT R85-4

A.I.S.E. A HYBRID SCHEDULING SYSTEM

Herve J. Duchesne de Lamotte

May 1985

A.I.S.E
A HYBRID SCHEDULING SYSTEM
by
HERVE J. DUCHESNE de LAMOTTE

Submitted to the Alfred P. Sloan of Management
on May 17, 1985 in partial fulfillment of the requirements
for the Degree of Master of Science in Management

ABSTRACT

A.I.S.E. (Advanced Interactive Scheduling Environment) is an interactive computer system designed to schedule efficiently a fleet of vehicles accordingly to a set of requests made in advance or at the last minute. The system consists of both an electronic drawing-board with which the schedule can be manipulated visually, and a support algorithm that helps the user to build the schedule efficiently.

This thesis begins by discussing the evolution that lead to the design of the A.I.S.E. planning tool together with the environment in which A.I.S.E. will be used. It then gives a general analysis of the support algorithm used in A.I.S.E.. The analysis describes the "insertion heuristic" that is being used, and shows how it can generate an efficient schedule from a list of requests, or insert an individual request into an already existing schedule with minimum disruptions. Since the heuristic was initially designed to solve the "Dial-a-Ride" problem, necessary modifications are also reviewed.

The various aspects of the schedule electronic drawing-board are then presented. The description is based on the version of A.I.S.E. that will be delivered to the U.S. Air Force Operational Support Airlift. Graphics displays are reviewed; the links between the support heuristic and the graphic interface are analysed. This presentation is followed by a review of other areas in which such a scheduling tool can be used.

Thesis Supervisors: Professor Thomas L. Magnanti
Professor of Operations Research and Management

Professor Robert W. Simpson
Professor of Aeronautics and Astronautics

ACKNOWLEDGEMENTS

I am indebted to both Professor Thomas L. Magnanti and Professor Robert W. Simpson as my thesis supervisors on this research. They guided me through the A.I.S.E. project and provided me with invaluable help to avoid the numerous pitfalls of English grammar.

I would like to thank all the staff and friends from Flight Transportation Laboratory for their support, especially Doctor Dennis F.X. Mathaisel who assisted me all along this research and provided me with many suggestions .

I would also like to thank my friends Remy Fasquelle who convinced me to start these studies, and John Tylko who very kindly gave me access to the laser printer of his company.

Finally, I would like to dedicate this thesis to my wife Catherine. Without her love, support and typing skills(!), these years of study would have never been possible.

TABLE OF CONTENTS

	<u>Page</u>
1. GENERAL PRESENTATION	
1.1 Background	8
1.2 Strategy for scheduling systems	9
1.3 O.S.A. scheduling problem	11
2. THE SCHEDULING HEURISTIC	
2.1 Presentation	12
2.2 The Insertion Algorithm	12
2.2.1 Some definitions	12
2.2.2 Purpose of the insertion algorithm	13
2.2.3 The selection criterion	15
2.2.4 The insertion procedure	15
2.2.5 Data structure associated with the schedule	20
2.2.6 General comments	22
2.2.7 Performance of the algorithm	24
2.3 Modifications and extensions over the original insertion algorithm for the Dial-a-Ride problem	26
2.3.1 Description of operational requirements	26
2.3.2 Adressing the operational requirements	29
3. THE GRAPHICS INTERFACE	
3.1 Presentation	50
3.2 The new generation of personal computers	50
3.2.1 The mouse	50
3.2.2 The desktop	51
3.2.3 The Macintosh environment	53
3.3 The graphic scheduling environment	62
3.3.1 O.S.A. scheduling organization	62
3.3.2 The schedule electronic drawing-board	66

	<u>Page</u>
4. FUTURE DEVELOPMENTS AND APPLICATIONS	
4.1 Support Algorithms	81
4.2 Schedule drawing-board	82
4.3 Other applications, Scheduling activities of non-Air Transport operations	83
5. CONCLUSION	
5.1 On the insertion heuristic	86
5.2 On the graphics interface	86
5.3 On the future	87
6. APPENDIX	
6.1 Sample input and output from the insertion algorithm	89
6.1.1 Aircraft data	89
6.1.2 Station data	90
6.1.3 Request data	90
6.1.4 Resulting mission assignment	91
6.2 Effect of different pool sizes in the algorithm	96
BIBLIOGRAPHY	97

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
2.1 Extract of a typical schedule	14
2.2 Simple insertion mechanism	14
2.3 The insertion algorithm as a scheduling tool	16
2.4 Selection of best insertions	19
2.5 Mission list	20
2.6 Data structure of schedules	20
2.7 Current data structure	21
2.8 Sequential insertion (1)	22
2.9 Sequential insertion (2)	22
2.10 Non-sequential insertion (1)	23
2.11 Non-sequential insertion (2)	23
2.12 Algorithm speed, fixed fleet & variable number of requests	25
2.13 Algorithm speed, fixed number of requests & variable fleet	25
2.14 Mission list with initial requirements	30
2.15 Mission with time wasted at a station	31
2.16 Efficient schedule	32
2.17 Infeasible insertion	32
2.18 Beginning of mission list	33
2.19 Insertion into the schedule	34
2.20 Handling of intermediate stops	37
2.21 Typical activity at a station	38
2.22 Handling of station activity in the M.A.C. system	38
2.23 Process to modify the schedule when an insertion is not feasible	42
2.24 Effect of a cutoff time on the schedule	46

<u>Figure</u>	<u>Page</u>
3.1 Relationship between mouse and cursor movements	51
3.2 A typical desktop display with three windows, one folder and one closed document	52
3.3 The Apple Macintosh desktop	54
3.4 Selection of a command from the menu bar	55
3.5 A typical Macintosh window	55
3.6 Relationship between the "elevator" position and the visible area of a document	56
3.7 Typical event driven system	58
3.8 Graphic object manipulation cycle	61
3.9 Graphic interface organization	62
3.10 O.S.A. scheduling organization	64
3.11 Physical set-up of the new scheduling organization	65
3.12 A.I.S.E.desktop display	66
3.13 The mission display window	67
3.14 Image of the "flight" object	67
3.15 Moving a flight object over the display	68
3.16 Schedule information window	71
3.17 Mission information window	71
3.18 Flight information window	72
3.19 Mission display with cut-off time	73
3.20 The station display window	75
3.21 Organization of flights in the station display	75
3.22 Arriving flight graphic object	76
3.23 Departing flight graphic object	76
3.24 Moving a flight on the station display	77
3.25 Station information window	78
3.26 Dialog window in the mission display	80
3.27 Dialog window in the station display	80

1 GENERAL PRESENTATION.

1.1 Background.

Two years ago, the U.S. Air Force Military Airlift Command (M.A.C.) started a major overhaul program to improve its Command and control procedures (C² upgrade) [ref. 15]. M.I.T's Flight Transportation Laboratory was selected as part of a task force to investigate new ways to support M.A.C. scheduling activities.

The basic recommendations made after one year of research were [ref. 9, 16]:

- *avoid* large -scale optimization algorithms,
- build *scheduling graphic workstations* to help human schedulers, and
- support parts of the scheduling activity with *heuristic algorithms*.

Based on these recommendations, M.A.C. asked the Flight Transportation Laboratory to put together a small test scheduling system for the Operational Support Airlift (O.S.A.) which is a small subset of the Airlift Command itself.

This thesis describes A.I.S.E. (Advanced Interactive Scheduling Environment), a scheduling tool which is based on the test system delivered to O.S.A.. The system consists of both a heuristic support algorithm that automatizes some aspects of the scheduler activities and an extensive graphic interface that can be viewed as a sophisticated electronic drawing board and spreadsheet, entirely devoted to the scheduling task.

This chapter will review the main ideas that lead to Flight Transportation Laboratory's recommendations to the Air Force. In order to set up the environment A.I.S.E. is designed to

work in, the chapter will then briefly describe O.S.A. operations as well as describe the type of problem that must be solved by the support algorithm.

Chapter 2 describes the support "insertion heuristic" that was adapted from the "dial-a-ride" environment. The modifications needed to adapt this heuristic to O.S.A.'s operational environment are extensively studied.

Chapter 3 presents the graphics interactive environment that is proposed to the scheduler and shows how it is linked to the support heuristic. It also reviews special programming considerations that are needed to implement such a system.

Finally, the last chapter discusses possible directions for future implementations of interactive scheduling tools.

1.2. Strategy for scheduling systems.

During the past two decades, most of the efforts made to solve practical scheduling problems resulted in large-scale optimization algorithms running on large mainframe computers [ref. 13, 14]. This approach became dominant because most available computers during that time were essentially number processing machines with very few interactive capabilities. Scheduling was also considered as an activity quantitative enough to be entirely solveable by pure mathematical techniques.

Overall, the results have been mixed [ref. 3]. The mathematical optimization approach was discovered to have the following drawbacks:

- the size of problems quickly became enormous and required extensive computation time,
- the assumptions or simplifications needed to integrate real-life scheduling problems into the appropriate mathematical framework lead very often to worthless solutions, and
- the underlying instability of optimal solutions was unacceptable for real-life operations.

For these reasons, large-scale models were found to be appropriate for schedule "*planning*" but not adapted to practical, every-day scheduling. In their analysis of Aircraft Scheduling [ref. 3],

Etschmaier and Mathaisel conclude:

It was thought for quite some time that scheduling airlines was relatively easy and could be carried out by a large linear model [...]. Although many closed form endeavors were begun, none of them were ever completed, but it was hoped that eventually the size and efficiency of computers could catch up and permit some sort of closed form [of the scheduling problem] to be solved.

The consensus today is that the problem of aircraft scheduling is unsolvable, at least by quantitative techniques.

In the last three years, because computers have become more used as general processors of not only numbers but also *text* and *pictures*, researchers have begun to think differently about the scheduling process. After all, some aspects of manual scheduling such as the types of charts used, appeared to be efficient ways to handle information. Humans were also found to be quite efficient at certain kinds of processing. Finally, the recent arrival of a new generation of graphic oriented microcomputers lead us to investigate a very different approach to solve the scheduling problem. Rather than use a machine to produce an excessively optimal schedule, why not use it to replicate electronically the "paper" environment of schedulers; let them create the schedule, and have the computer check for errors and violations of constraints, manage all the data and display graphically appropriate information when needed.

A first experimental prototype was built and successfully tested by Flight Transportation Laboratory [ref. 1]. It quickly demonstrated that such an approach took advantage of the processing power of human schedulers. It also showed that some *parts* of the scheduling activity were indeed quite quantitative and could certainly be handled by heuristic support algorithms that would produce reasonably good and stable answers to the scheduler.

These ideas resulted in our proposal for a hybrid scheduling environment that would exploit the processing capabilities of both the computer and the human scheduler.

1.3 O.S.A. scheduling problem.

The Operational Support Airlift, a division of the Military Airlift Command, has responsibility for the rapid delivery of Air Force personnel and small cargo to any point within the continental United States. O.S.A. uses a fleet of 60 to 80 small business airplanes and satisfies about 200 out of 300 daily requests for transportation. At this time, rejected requests are either accommodated later or simply dropped.

Its activity cycle can be viewed as follows:

- 1) The requests (see 2.1.1) are filed from 0 to 15 days in advance.
- 2) A first schedule plan is prepared two days in advance.
- 3) The schedule is fixed one day in advance.
- 4) Subsequently, last minute requests can be handled only if their insertion does not modify the schedule.
- 5) On the day of service, the schedule is monitored constantly.

A.I.S.E. completes these requirements in the following way:

- Step 1 is handled by a data base manager that maintains the list of requests.
- Steps 2 and 3 are executed by both the support algorithm that proposes an initial schedule, and the scheduler who alters it with the graphic schedule editor in order to account for non-quantifiable constraints.
- The decision support algorithm will also suggest good ways to insert last minute requests into the schedule (step 4); again the solution can be modified by the scheduler.
- Finally, schedule monitoring (step 5) is handled by graphic tools with one exception: after airplanes failures, the decision support algorithm is used to reschedule requests.

The relatively small size of this system made it ideal for a test demonstration that has been implemented on Apple Macintosh microcomputers.

Let us now begin our presentation of A.I.S.E. by describing its support algorithm.

2 THE SCHEDULING HEURISTIC.

2.1 Presentation.

The heuristic selected for the M.A.C. scheduling system is an "Insertion" algorithm initially developed by Dr. J. J. Jaw in order to solve the "Dial-a-Ride" problem. Many similar heuristic algorithms have been developed over the years to solve the Travelling Salesman and the Dial-a-Ride problem [ref. 6, 10, 17] but they usually consider a single pick-up or delivery point. Jaw's heuristic was chosen because it allows multiple pick-up and delivery points and this characteristics made it more appropriate for O.S.A.'s operational environment. In his thesis [ref. 8], Dr. Jaw extensively presents the properties of this algorithm as well as its behavior under various scenarios. We will review here the most fundamental aspects of the heuristic: how it tries to produce an economically good answer and the computational efficiency of the algorithm. We will then investigate the modifications and extensions to this algorithm that are required by the Military Airlift Command specialized operations. Some of these modifications have already been implemented and tested.

2.2 The Insertion Algorithm.

2.2.1 Some definitions.

Let us first review definitions of terms which will appear often in this thesis.

2.2.1.1 Fleet, stations and requests.

There are three sets of data which completely define the scheduling problem.

The **Fleet** represents the resources available to the scheduler. It consists of different **vehicles** each with its own physical characteristics (speed, capacity) as well as economical characteristics (operating costs...).

Vehicles with similar characteristics are said to be of the same **type**.

The **Stations** are the different Cities, Airports, warehouses etc..., the vehicles must serve. Their geographical locations define some of the problem constraints such as interstation distances. Other constraints will be defined by more general characteristics of these stations such as curfews, fuel availability, length of runways etc...

Last, the **Requests** are demands for transportation of cargo, passengers etc... between two stations. They mainly specify the volume to be carried and in our case, the earliest time at which the load will be available as well as the latest time before which it must be delivered. Thus, there is a "time window" constraint associated with each request.

2.2.1.2 Missions and Events.

For each vehicle, a **Mission** is a sequence of events occurring sequentially at specific times during the scheduling period. These events will usually be of three types:

- **Pick-up** a specific load at a given station.
- **Deliver** a specific load at a given station.
- **Go** from one station to another. (In the air transport environment, this will be a flight.)

2.2.1.3 Schedule.

A **Schedule** consists of a series of missions which are individually assigned to specific vehicles from the fleet. An extract of a typical schedule is shown in figure 2.1.

2.2.2 Purpose of the Insertion Algorithm.

As its name indicates, the purpose of this algorithm is to insert a specific "request" for transportation into an already established schedule (see figure 2.2). If the request can be

<u>SCHEDULE</u>		
MISSION #1 (vehicle 1)	MISSION #2 (vehicle 2)	MISSION #3 (vehicle 3)
Pickup 3 passengers in AAA, at 8:30	Relocate from CCC to LLL, leave @ 6:00	Pickup 1 passenger in MMM, at 6:00
Fly from AAA to BBB leaving at 8:45	Pickup 1 passenger in LLL, at 8:15	Fly from MMM to NNN, leaving at 6:30
Deliver 3 passengers in BBB, at 10:15	Fly from LLL to III leave at 9:00	Deliver 1 passenger in NNN, at 6:45
Pickup 5 passengers in BBB at 11:00	...	Deliver 1 passenger in OOO, at 7:00
...		

Figure 2.1: Extract of a typical schedule

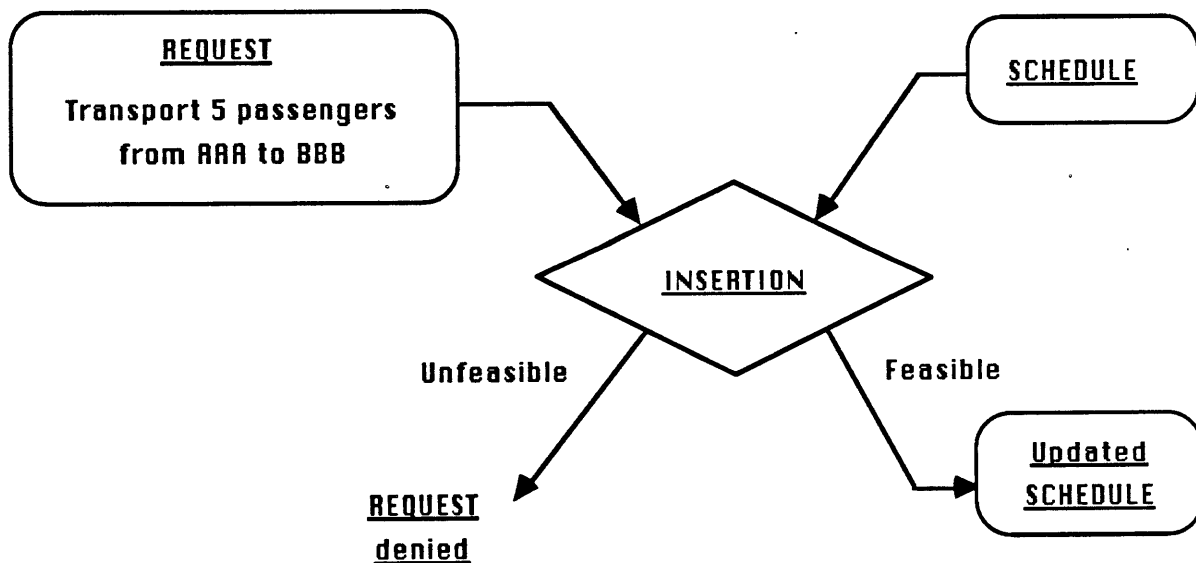


Figure 2.2: Simple insertion mechanism

inserted in many different ways, some criterion will be used to select the insertion to be retained.

The use of the algorithm can be generalized:

- if the initial schedule is "empty" (no request has been served yet), the sequential insertion of multiple requests will be equivalent to the creation of a new schedule. This turns the heuristic into a complete *scheduling algorithm* (see figure 2.3.).
- if the schedule is not empty, the algorithm will simply try to insert with minimum disruptions; it will only modify the existing schedule.

In general, the algorithm will be used in a three-step process:

- 1) Accumulate a certain number of requests and create an initial schedule to satisfy them as much as possible (scheduling function).
- 2) Insert last minute requests into this schedule (insertion function).
- 3) Revise parts of the schedule to accommodate last minute requests of high priority which cannot be inserted otherwise (mixture of both functions).

2.2.3 The selection criterion.

As we mentioned in the last section, when a request can be satisfied in many different ways a selection must be made. In the current implementation of the heuristic, the selection criterion retained is minimum incremental travel time. Among various insertions, the selected candidate will be the one that increases the duration of a mission by the least amount of time. The best possible insertions are therefore those that accommodate requests with already scheduled flights since they add no incremental time to missions.

2.2.4 The insertion procedure.

2.2.4.1 Insertion of single last minute requests.

The algorithm uses three phases to select the best way to satisfy the new request:

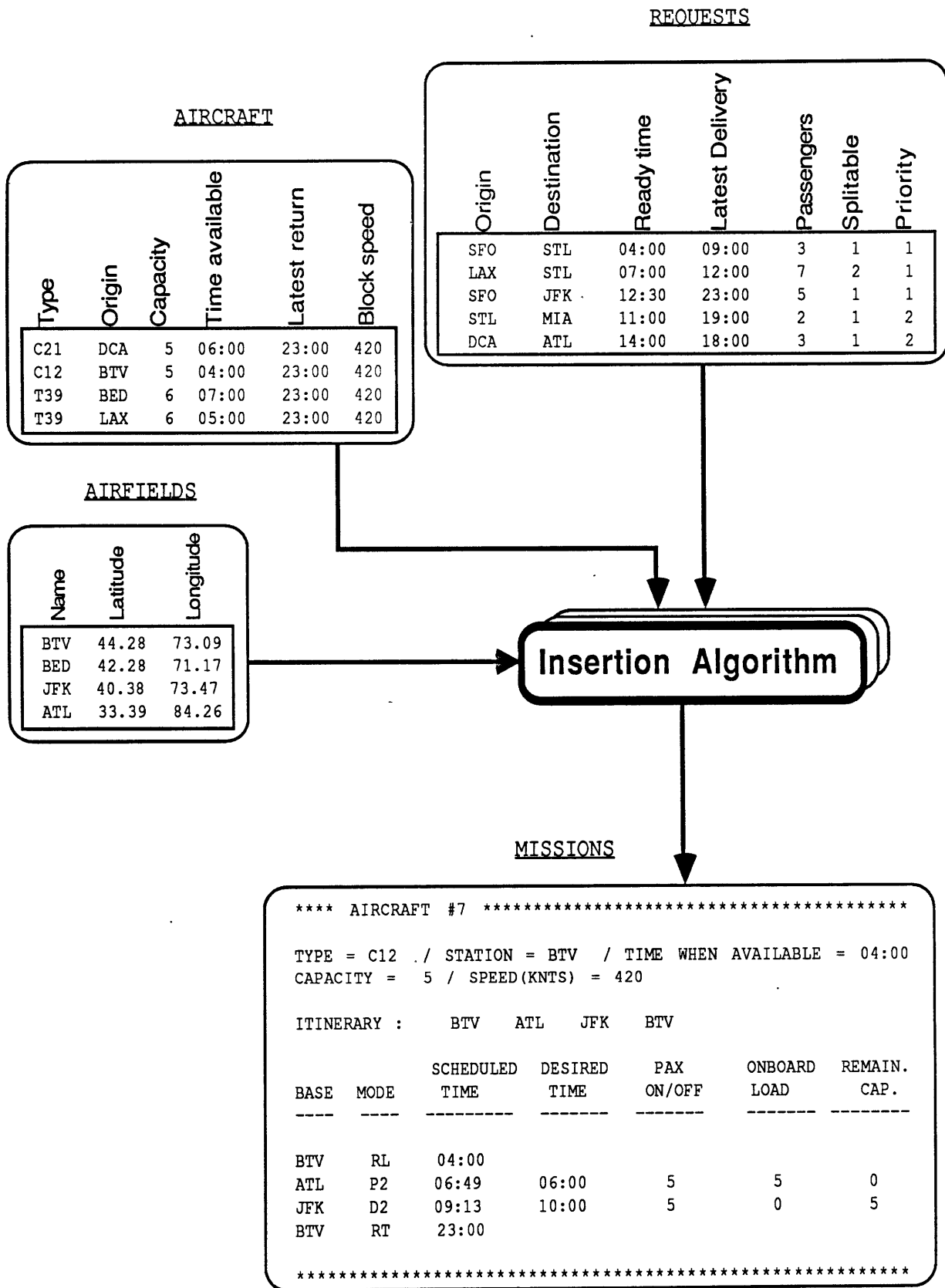


Figure 2.3: The insertion algorithm as a scheduling tool

Phase 1:

For each vehicle in the fleet , establish all possible ways the request can be inserted into the current schedule.

At this point, account only for the time constraints.

Compare all possible solutions and select:

- the best one that can carry the full load
- the best one that can only carry part of the load (due to limitations on the remaining space available).

Phase 2:

- For the "full load" solutions, compare the best answers among all vehicles and select the best one.
- For the "split load" solutions, look at every possible combination of two best answers among all vehicles and select the best pair that can accommodate the full load. (In the current implementation a pair's travel time is the sum of the travel times with each vehicle.)

Phase 3:

Compare the "full load" and "split load" solutions and select the best one. If both solutions add the same incremental time, choose the "full load" answer. If no feasible insertion can be found at the end of phase 2, the request is rejected.

2.2.4.2 Sequential insertion of multiple requests (initial scheduling).

- 1) The initial processing of each request is executed with the 3 phase process described above.
- 2) The best solution is then inserted into a "pool" of n answers which already contains the best solutions corresponding to n-1 previous requests from the request list.
- 3) The best answer among these n will finally be inserted into the schedule. If the requests

have different priorities, higher priorities will always be chosen first.

- 4) At this point, the remaining $n-1$ solutions will be re-calculated, taking into account the new insertion, and a new request will be pulled from the list in order to keep the pool always full.

This procedure is summarized in figure 2.4.

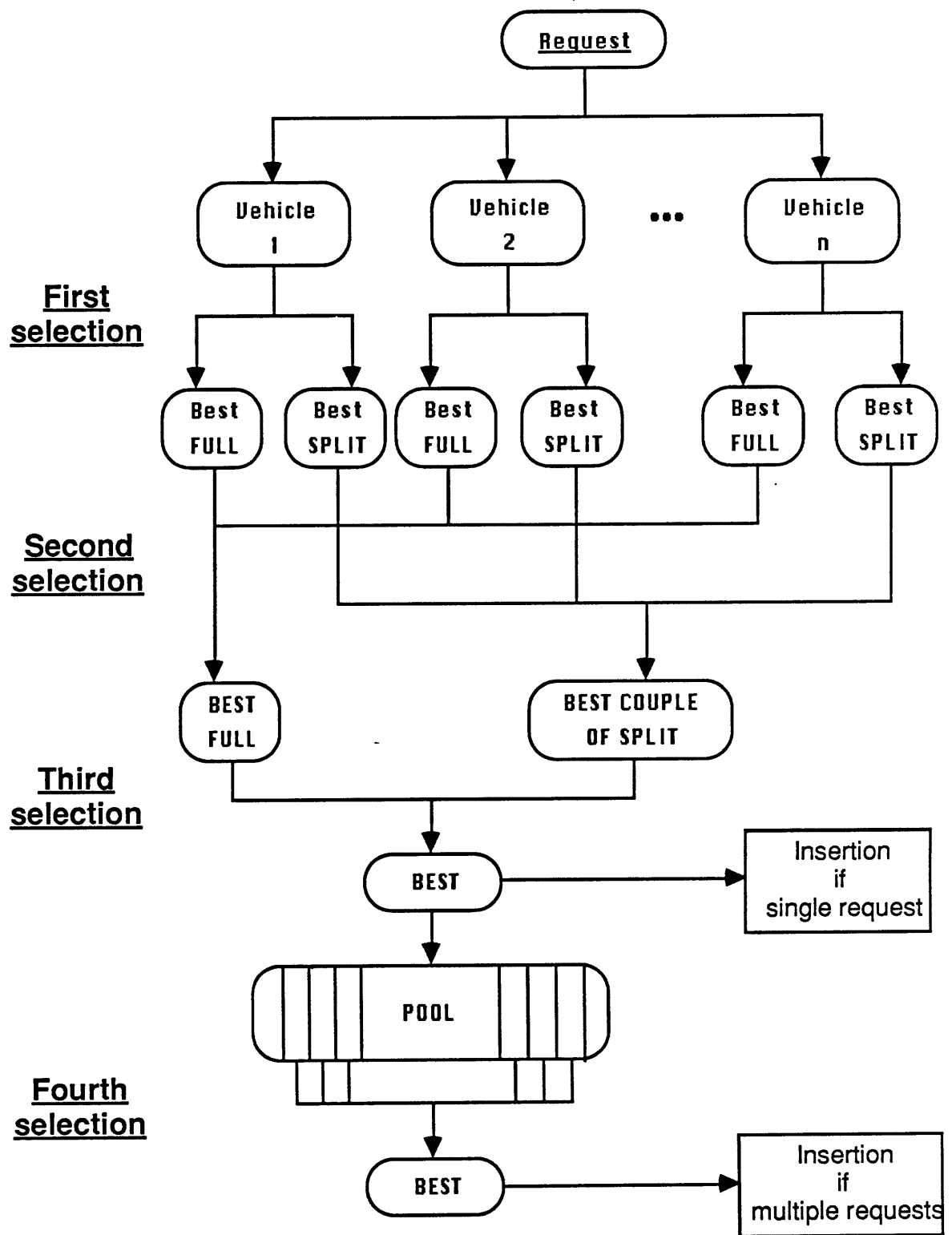


Figure 2.4: Selection of best insertions

2.2.5 Data structure associated with the schedule.

A very flexible internal data structure has been chosen to represent each vehicle's mission. It is based on the events described in 2.1.2.

Considering that each mission is a succession of events during the scheduling period, it is represented as a linked list of event data records:

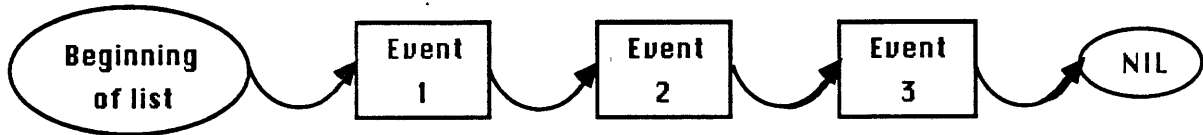


Figure 2.5: Mission list

Adding in the vehicle data itself, the global schedule is represented in computer memory as follows:

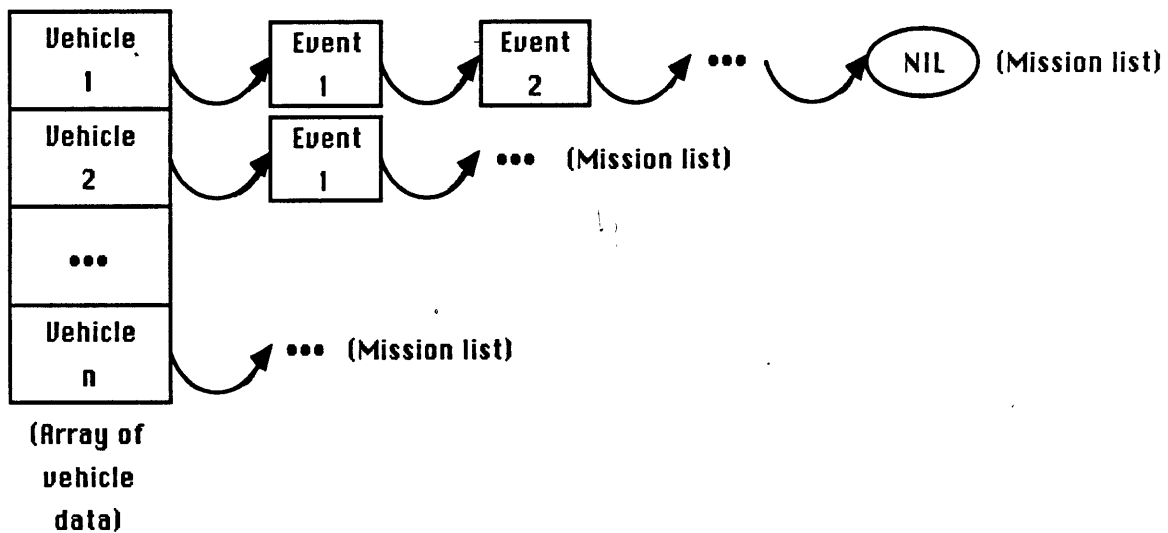


Figure 2.6: Data structure of schedules

Figure 2.7 gives a full description of the structure with details of the content of the event record.

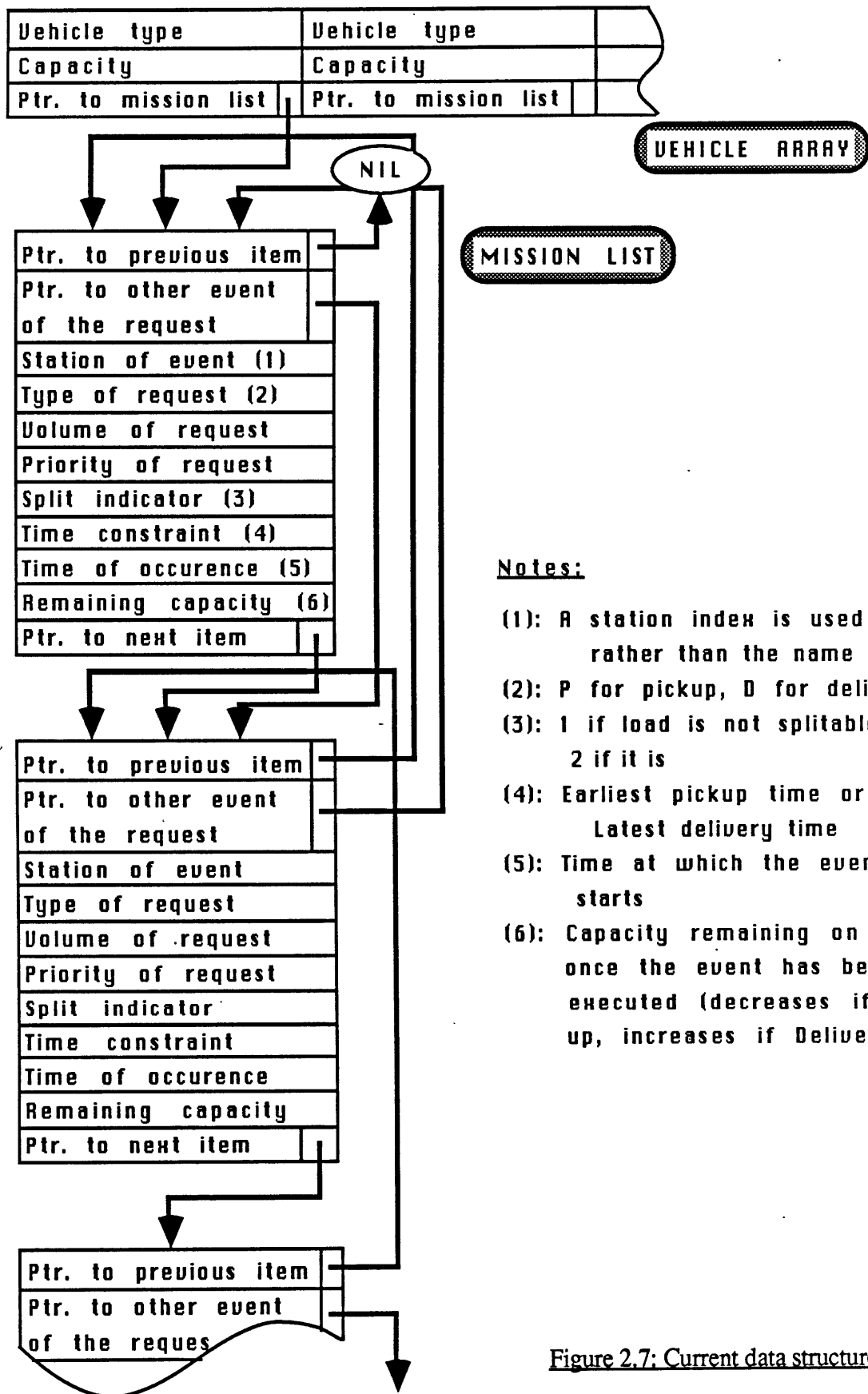


Figure 2.7: Current data structure

2.2.6 General comments.

In light of the above descriptions, a few comments can now be made.

- 1) The handling of split loads was not part of the original "Dial a ride" insertion algorithm.
- 2) Even though three types of events were described earlier, only two -pick up and delivery- will appear in a mission list since it is assumed that a vehicle has to move between two successive events which occur at two different stations.
- 3) This algorithm is a "forward insertion" heuristic: it will always insert a new event after an old one. For this reason, only four types of insertions will be possible for a request, (remember that an inserted request will consist of two events, a pick up and a delivery):
 - Sequential insertion at end of schedule:

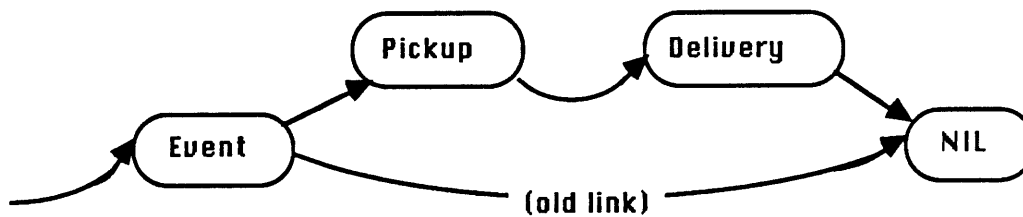


Figure 2.8: Sequential insertion (1)

- Sequential insertion between two events from the mission list:

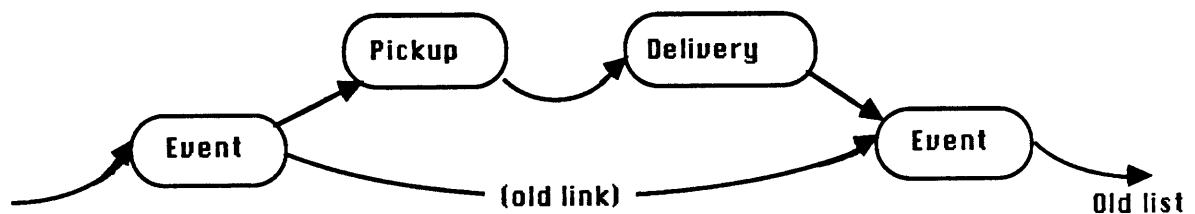


Figure 2.9: Sequential insertion (2)

- Non-sequential insertion with delivery at end of schedule:

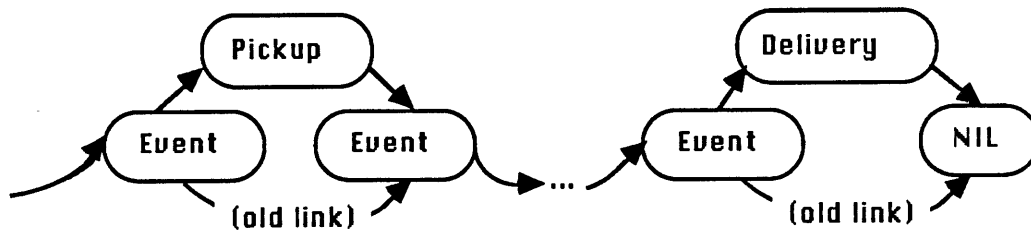


Figure 2.10: Non-sequential insertion (1)

- Non-sequential insertion anywhere in the schedule:

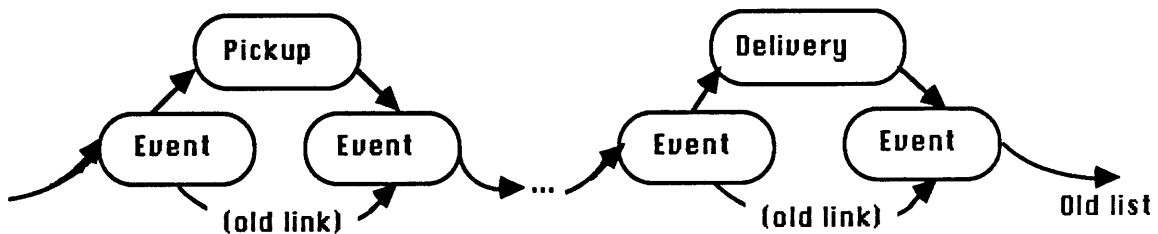


Figure 2.11: Non-sequential insertion (2)

Practically, when the heuristic tries to insert a new request into a mission list, it will skip over the events which occur before the earliest time at which the load will be available. Then, in order to find a feasible insertion, the algorithm will try each of the four types just described, using the remaining events of the mission list.

The last three types of insertion may require the algorithm to push forward (i.e. later) in time some already scheduled events. For instance, let us suppose that a vehicle went from station AAA to BBB in its original mission; all pick-up's and deliveries occurring in BBB and after must be delayed to a later time if we want the vehicle to go from AAA to CCC and only then to BBB. The incremental trip time will be equal to the trip time from AAA to CCC plus the trip time from CCC to BBB minus the old trip time from AAA to BBB. If this process does not violate any of the latest delivery constraints, the insertion will be considered as valid.

This behavior triggered the name "forward insertion" since events can be inserted only *after other events*, and once inserted can be moved only *forward in time* by the algorithm.

2.2.7 Performance of the algorithm.

2.2.7.1 Size of the pool.

The pool (see 2.2.4.2) is the area where a certain number of feasible insertions are kept for comparison before the fourth selection process (Figure 2.4). With a larger number of slots in the pool more insertions can be compared and the resulting schedule should be better. Different sizes, ranging from 1 to 5, have been experimented with, using various test cases. The elements retained to evaluate the schedule produced in each case were the number of passenger-miles flown, the average system load-factor and the total travel time required by the schedule. The results are shown in Appendix 6.2.

As Jaw showed in ref. 8, increasing the pool size *does not* produce any specific improvement. In our example from the appendix, we can even observe a deterioration of the schedule with larger sizes. However, in other cases which were tested, pool sizes of 1 and 2 gave obviously bad schedules with many rejected requests. Improvements came only with a size of 3. Overall, there is no systematic trend and the "goodness" of the solution seems to depend only on the order in which requests are presented for insertion.

After some experiments it was decided that a pool size of 5 would be a possible compromise between producing a good schedule and not increasing substantially execution time. For the time being, results produced by the insertion heuristic are at least as good or better than those produced by human schedulers. However, further research is needed; a better optimization objective-function must be defined, and the procedure to select best insertions must be improved.

2.2.7.2 Execution time.

The timings shown on figures 2.12 and 2.13 were derived, using an Apple Macintosh computer. The time is measured from the beginning of the insertion process to its end. Set up time is not counted, neither is the time it takes to produce the results report.

Surprisingly, when one parameter (fleet size or number of requests) is fixed, calculation times seem to increase linearly with the other parameter. This is not true. In fact, the

problem solved is so small that its real calculation time is hidden by a substantial computer operating-system overhead. Jaw has shown [ref. 8] that computation times needed by the insertion heuristic are proportional to $V*(R/V)^3$ with V vehicles and R requests.

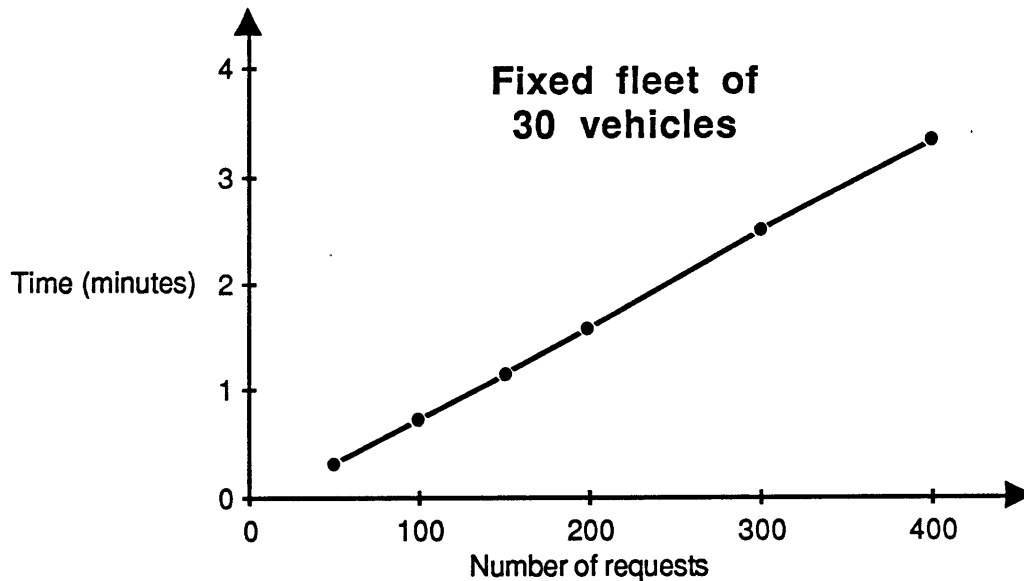


Figure 2.12: Algorithm speed, fixed fleet & variable number of requests

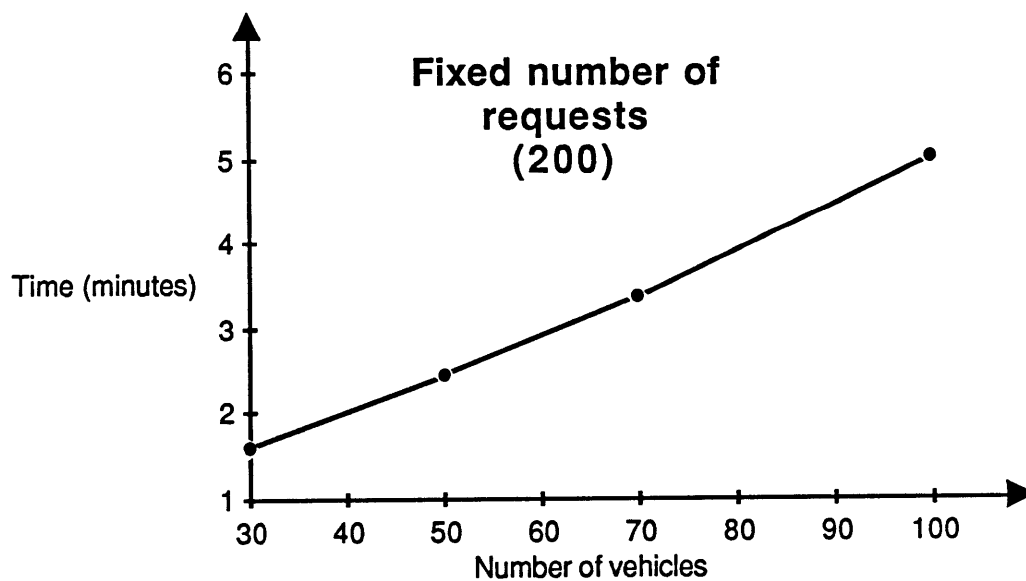


Figure 2.13: Algorithm speed, fixed number of requests & variable fleet

2.3 Modifications and extensions over the original insertion algorithm for the dial-a-ride problem.

Most of the extensions which will be discussed here, were needed in order to satisfy the Military Airlift Command operational requirements. Even though some of them may be very specific to M.A.C. type of activity, most extensions would be needed in a general purpose scheduling system.

When discussing the graphics aspect of A.I.S.E., other requirements will be studied which do not involve the insertion heuristic.

Let us first review the M.A.C. requirements which necessitate modifications or extensions to the algorithm. Their actual implementation will be discussed afterwards.

2.3.1 Description of operational requirements.

2.3.1.1 Duration of events.

The current implementation of the algorithm assumes that each event is instantaneous. For instance, if a pick-up occurs at 10:00 the vehicle can leave the station at the same time. This assumption is obviously unrealistic and must be modified.

2.3.1.2 Return to base, return to predefined station.

A vehicle located at a station at the beginning of the scheduling period may be forced to return to that original base no later than a specified time.

A natural extension of this assumption is to be able to force the vehicle to return to any specified station.

2.3.1.3 Maximum Crew duty.

Each mission is served by one crew and there can be some constraints on how long a crew can remain operational during a scheduling period. This limits the duration of all missions. M.A.C crews have a maximum duty time of 16 hours per day in peace time; their duty day starts one hour before the first departure.

2.3.1.4 Operational range of vehicles.

There usually are some restrictions on how far a vehicle can go without refueling. This means mainly that certain stations cannot be linked by a direct, non-stop service.

2.3.1.5 Minimum time on ground.

In the case of passenger transportation, boarding and disembarking events have a very short duration and an efficient schedule could become very "tight" and quite sensitive to unexpected events such as weather delays, minor breakdowns etc... In order to introduce some slack in its operations and give some margins for small delays, M.A.C. requires a minimum of one hour on the ground at any station.

2.3.1.6 Stations restrictions.

There may be some constraints at specific stations, which should be taken into account when the schedule is established.

The most important ones will be:

- curfews : vehicles may not leave or arrive at a station during a certain period of time.
- maximum number of vehicles at a station at any given time: a station may have a limited number of parking slots.
- maximum handling capacity : there may be enough parking space but not enough equipment to unload more than a certain number of vehicles at the same time.
- runway length: in the case of airplanes there may be some physical constraints limiting the access to certain stations.

2.3.1.7 Force a vehicle to stay at a station.

Again, in the case of passenger transportation, many requests will really consist of two requests:

- Pick up somebody at a station and take him to another station (for a meeting).
- Take him back to the first station (after the meeting).

Very often, however, these requests are left open-ended: the return travel earliest time will depend on the meeting itself. Under these circumstances M.A.C. ensures the availability

of a vehicle for its V.I.P.'s by keeping it ready on the ground as long as necessary.

During that time the vehicle *cannot* be used to satisfy any other request.

2.3.1.8 Force a vehicle to provide a non-stop service.

Even though the service will be provided within the desired time constraints, some individuals might require service in which they are transported from one station to another without many stops in between to pick up or unload other passengers.

2.3.1.9 Keep the departure times unchanged.

When the insertion process has reached the second stage where last minute requests are handled, the algorithm might find a feasible solution by pushing forward in time some of the services provided by a vehicle.

Unfortunately, if a scheduler has already transmitted the schedule to the individuals who placed the requests, it might be desirable not to touch the departure times at all in order to avoid too many late changes.

2.3.1.10 Force an infeasible request to be inserted and/or replace other requests of lower priority.

Once an initial schedule has been established, all subsequent requests will be inserted only on top of it. Prior requests already inserted will never be removed. It is possible, therefore, that at a certain point the schedule becomes so dense that no further request can be served, whatever its priority.

If a last minute request of extreme urgency is presented to the scheduler, it may not be acceptable to have the system simply reject that request if its insertion into the old schedule is infeasible. A way should be provided to allow this request to replace any other request of *lower* priority or, if this is not enough, to be inserted into the schedule *before* any request of *same* priority.

2.3.1.11 Fix schedule up to a cut-off time and reschedule afterwards.

If a long series of last minute requests is presented to a scheduler who has already released the schedule to operations, he (she) might want to modify it only after a certain time in order

to minimize short term disruptions and changes.

2.3.1.12 Reschedule missions following a vehicle failure.

This is certainly one of the most difficult task for the scheduler. For this reason it has usually been handled in the most straightforward way: leave the other missions untouched, relocate a spare vehicle to the station were the failure occurred and resume the mission with a substantial delay...

However, an optimized rescheduling will often decrease the negative impact of vehicle failures by re-routing more than one vehicle and spread the effect of these failures over more missions. Such a capability should be added to our system.

Having reviewed these requirements, let us now return to the algorithm and see how it can be modified or extended to account for them.

2.3.2 Addressing the operational requirements.

2.3.2.1 Duration of events.

A very simple modification was made:

- 1) Add an event duration field to each event record of the mission list.
- 2) Modify all the time and schedule calculations in order to account for this term.

2.3.2.2 Return to base.

Here, rather than altering the insertion algorithm, we decided to make use of "dummy events". These events consist of pick-ups and deliveries of "null" loads (0 passenger, 0 pallet of cargo...) which are inserted into mission lists in order to force vehicles to be at specific stations at specific times.

Suppose that our vehicle must return to its base no later than time **T**. The schedule is first initialized with a dummy delivery that makes the vehicle start from the morning base. The time at which this delivery occurs will be the beginning of the scheduling period (when our

vehicle becomes available). Following this initial delivery, the algorithm inserts a dummy pick-up followed by another dummy delivery. All three events occur at the same station.

- a) The second dummy delivery has a latest delivery time set equal to T in order to ensure that the vehicle returns no later than T .

The event duration of this delivery is set to infinity to avoid the insertion of requests after the return to the base.

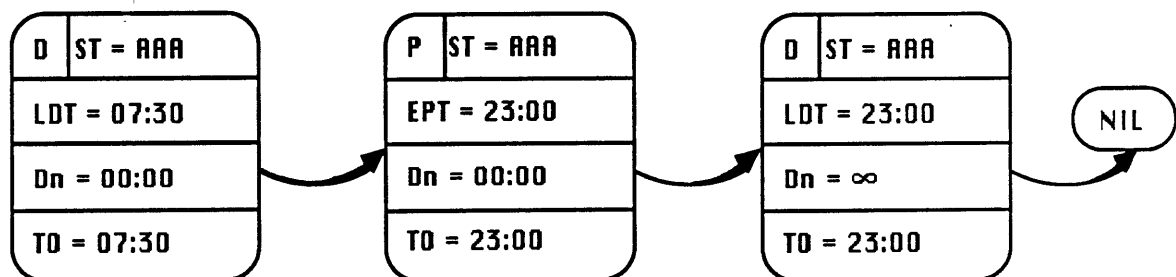
- b) The dummy pick-up has an earliest available time set to T and a duration equal to zero.

This dummy pick-up is necessary to keep the pick-ups and deliveries matched. This requirement is not crucial in the current algorithm, but may become so in future extensions. Another reason for including this activity is algorithm efficiency since certain tests of feasibility are done using pick-up events.

Finally, for overall consistency, the time of occurrence of both events is set to T .

Figure 2.14 shows how initial requirements will be translated into the initial "empty" schedule. Suppose these requirements were: Vehicle V will be ready in AAA at time $T = 07:30$ and will have to return no later than $T = 23:00$.

The initial mission list for vehicle V will be:



Legend:

P = pickup, D = delivery

ST = station name

EPT = earliest pickup time, LDT = latest delivery time

Dn = duration of event

TO = time of occurrence for event

Figure 2.14: Mission list with initial requirements

If the vehicle must return to a station different from its base of origin, the station of both dummy events can simply be changed accordingly.

Finally, if the scheduler wants the vehicle to return to a specific station without any time limit, the latest delivery time of the dummy delivery can be set to infinity.

2.3.2.3 Maximum Crew duty.

This constraint can be checked by keeping track of the total mission time for each vehicle.

This time includes both moving time as well as time spent at each station.

At the first and last stations of the mission, the time to be considered should be respectively vehicle preparation and "deactivation" time.

When a new insertion is considered, the algorithm should simply add to the current mission time the extra time required and check that the sum remains within the limit. If not, that particular insertion will be dropped from further considerations.

Since each request is served as early as possible, it is possible that a vehicle leaves at the beginning of the schedule period in order to serve an early request or relocate itself to another station where service will be needed. Often, this early move will be followed by an extensive time on the ground in order to wait for the next load to be available for pick-up.

For instance, we will have:

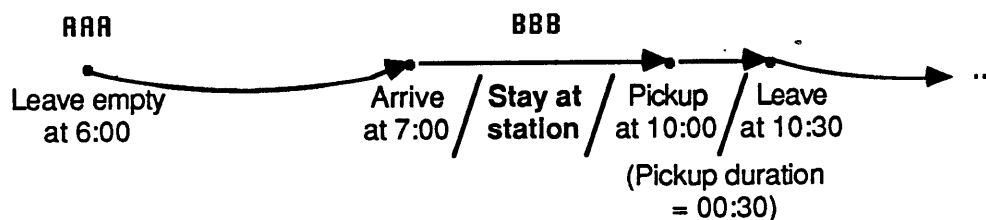


Figure 2.15: Mission with time wasted at a station

Because of the limit on mission time, such a schedule should really be:

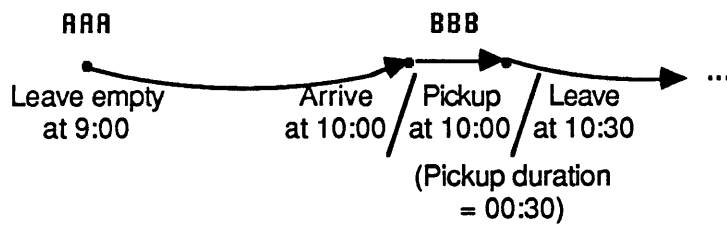


Figure 2.16: Efficient schedule

However, if the first leg is pushed forward in time, we have seen that the algorithm has no ability to move it back. The following new insertion will therefore become impossible:

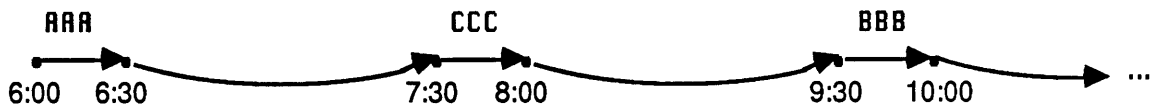
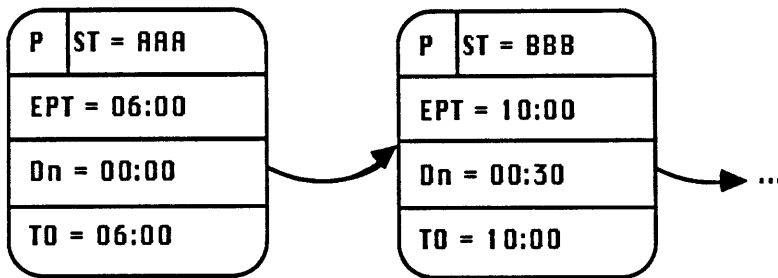


Figure 2.17: Unfeasible insertion

For this reason, the dummy delivery event simulating the time at which the vehicle becomes available should never be moved forward. Rather, when computing the mission duration, the algorithm should assume that the vehicle will leave on its first trip as late as is compatible with the first latest delivery constraint. When evaluating new possible insertions, the mission time will be increased as described above only if the insertion does not involve the first trip. If it does, the duration will be recalculated not using the time of the first dummy delivery but assuming that the new first request is served as late as possible.

Let us clarify this approach using the previous example. Before evaluating insertions, the beginning of the mission list for our vehicle was:

(We have assumed that time to move from AAA to BBB was one hour)



Legend: See figure 2.14

Figure 2.18: Beginning of mission list

This list must be interpreted as:

- The vehicle is available at 6:00 in AAA.
- It will go from AAA to BBB totally empty (relocation).
- Finally, it will pick up a load in BBB at 10:00.

According to the principles discussed above, mission length will be calculated by adding the following terms:

- **One hour** for vehicle preparation.
- **One hour** to relocate the vehicle from AAA to BBB (this move is assumed to occur as late as possible from 9:00 to 10:00).
- **Thirty minutes** of ground/loading time in BBB.
- Etc...

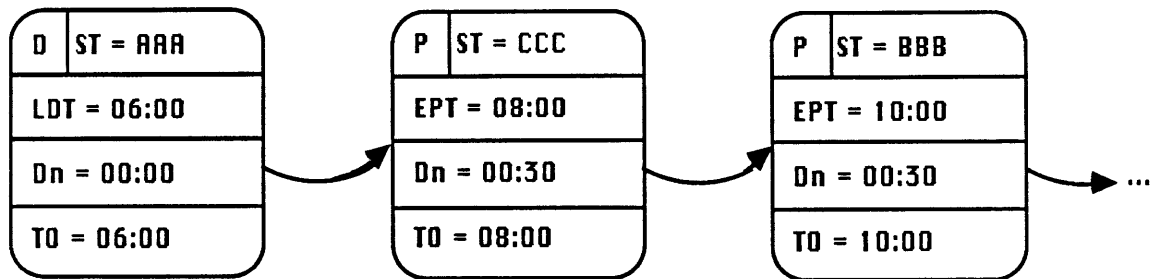
If no "special treatment" was given to the first leg of the mission, the time calculation would assume that the relocation occurs between 6:00 and 7:00 and add a three hour stay on the ground (entirely wasted) in BBB.

Let us now suppose that part of a new request to be inserted is:

Pick up a load in CCC with EPT=8:00 and Dn=0:30.

(Moving times are respectively: **one hour** from AAA to CCC and **one hour** from CCC to BBB).

A possible insertion to be evaluated will be:



Legend: See figure 2.14

Figure 2.19: Insertion into the schedule

Mission length should now be calculated by adding the following terms:

- **One hour** for vehicle preparation.
- **One hour** to relocate from AAA to CCC (again assume a departure as late as possible at 7:00 and not 6:00).
- **30 minutes** for loading in CCC.
- **One hour** to go to BBB (from 8:30 to 9:30).
- **30 minutes** waiting on the ground at BBB.
- **30 minutes** to load in BBB.
- Etc...

As one can see, if the minimum ground time requirement is ignored, there are still 30 minutes lost in BBB. One solution could be to move flight AAA to CCC forward by 30 minutes.

Unfortunately, this tactic would not be compatible with the algorithm implicit "policy": serve each request as early as possible. There is, however a definite tradeoff. Experience only will show whether it is better to serve each request as early as possible or implement a "compaction algorithm" which would push events around in order to minimize time wasted on the ground.

At the end of the mission, the same problem will occur *only if a "return to base" constraint is implemented*. In other cases, the algorithm will schedule events as early as possible insuring the earliest possible arrival at the last station of the mission.

In the "return to base" case, the problem can be easily fixed by changing the constraint implementation previously described (see 2.3.3). The "earliest pick up time" of the dummy pick up event should be changed to the time at which the vehicle is available at the beginning of the scheduling period.

The algorithm will try to serve this dummy request as early as possible but because of the infinite duration of the delivery all requests will be inserted before it.

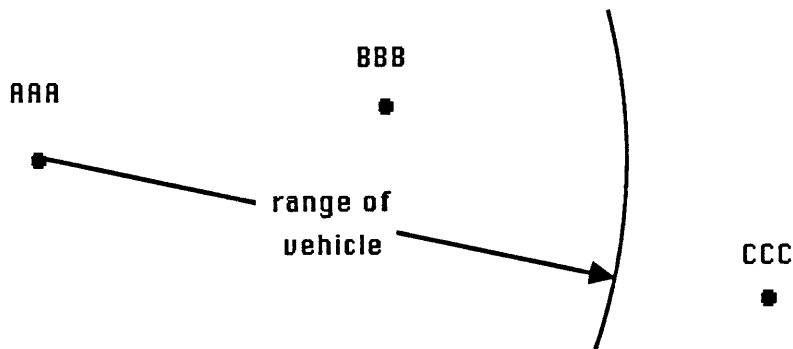
With this new set up, we can expect to see the two dummy events separated by other pick up or deliveries. Very likely, the dummy pick up will be the first event served in the morning if the return has to be at the base (since the vehicle is already at the base, it does not cost anything to insert the dummy pick up at this point). In fact, if it appears that any matching of events (one pick up for one delivery) is not crucial, the dummy pick up could be removed.

2.3.2.4 Range of vehicles.

When the insertion of a request necessitates a new movement from one station to another, the algorithm should check that the distance is less than the maximum range of the concerned vehicle. If the range is not sufficient, however it is not possible to simply reject the insertion.

Suppose for instance that we have:

- 3 stations:



- one vehicle whose range is not enough to go directly from AAA to CCC.

If the first request is for service between AAA and CCC, the range constraint would force the algorithm in its current state to reject it. AAA to CCC would never be feasible; this is obviously wrong. On the other hand, if the first request is between AAA and BBB and if this trip leaves enough space for the AAA-CCC load, then the service from AAA to CCC will become perfectly feasible with a stop in BBB. The feasibility of "out of range" requests depends on the order in which they appear in the request list.

For a specific request, if the best insertion appears to be infeasible due to range restrictions, the following strategy could be used:

(1) Find an appropriate feasible multi-stop itinerary.

- by asking the scheduler, or
- by looking into a data base where possible itineraries could be stored in advance or upon request.

(2) Calculate the "cost" of providing the service with that itinerary.

(3) Compare it to the best entirely feasible insertion (if any).

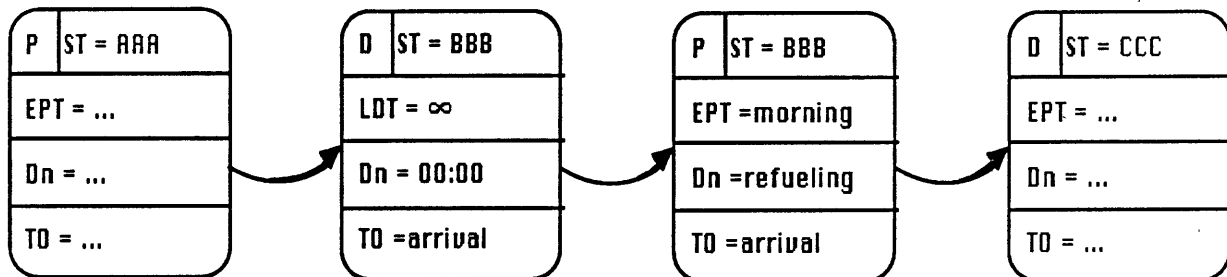
(4) Take the cheapest one.

The insertion of a multi-stop itinerary will be done with two dummy events inserted at the intermediate station:

- a dummy delivery of the load, with an event duration of 0 minutes followed by:
- a dummy pick up of the load, with an event duration equal to the refueling time of the

vehicle.

For a load to be carried from AAA to CCC with an intermediate stop in BBB, the mission list will be:



Legend: See figure 2.14

Figure 2.20: Handling of intermediate stops

The latest delivery time and earliest pick up time of the dummy events are set respectively to infinity and to the start of the scheduling period. They should not interfere with the other events.

2.3.2.5 Minimum time at stations.

In every case, the time spent at the station can be divided as follows:

- 1) Just after the arrival, a first time period is needed to *unload* the vehicle.
- 2) Then there is a *waiting period* until a first load is available for pick-up.
- 3) The third period is the *loading period* that may include other waiting periods if the loads to be picked-up are not quite ready.
- 4) A final *waiting period* will be required only if the minimum time at the station is more than the length of all previous periods.

The duration of the loading and unloading periods will depend on the following factors:

- Duration of individual pick up and delivery events.
- Whether these events are parallel (more than one piece of cargo can be unloaded at the same time) or sequential (if the vehicle has only one door for instance) .

- Earliest pick up times.

All these elements are summarized in the following figure.

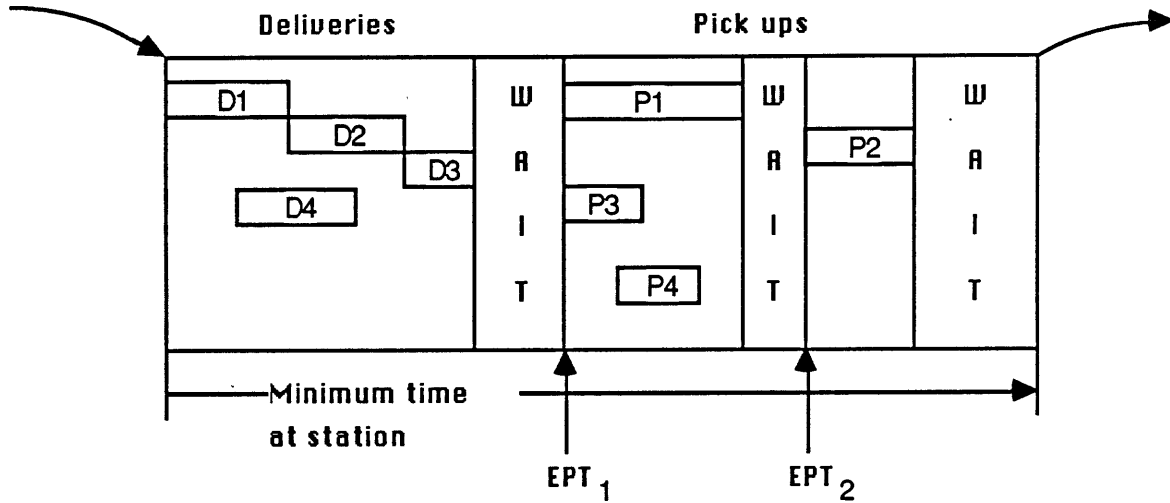


Figure 2.21: Typical activity at a station

In the case of our system, only passengers will be transported. Consequently, the loading and unloading times will always be very short and altogether less than M.A.C. required minimum time at a station.

Also, because passengers do not like to wait inside a vehicle that is not moving, we suggest handling the constraint as follows:

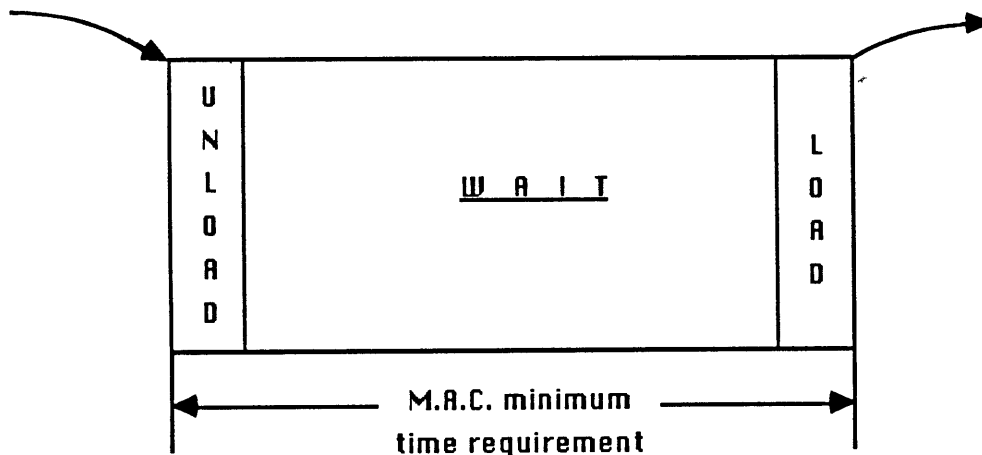


Figure 2.22: Handling of station activity in the M.A.C. system

- 1) Assume that each pick up or delivery event has a duration of say 15 minutes and that they all occur in parallel.
- 2) Thus:
 - set the start time of delivery events to the landing time, their duration to 15 minutes.
 - set the start time of pick up events to (**Landing Time + Minimum Time At Station -15 Minutes**) and their duration to 15 minutes.

2.3.2.6 Station Restrictions.

Most of the restrictions listed in 2.3.1.6 can be handled with a simple check by the algorithm when a new insertion is being evaluated.

- If a station has any constraint on the vehicle types, insertions will be tried only on appropriate types.
- If any curfew exists, insertions with inappropriate pick up or delivery times will be rejected.
- etc...

The limit on loading/unloading equipment will require a different processing. It will mainly affect the length of time spent at a station since the vehicle will have to wait for the equipment to be available.

2.3.2.7 Force a vehicle to stay at a station.

This constraint can be handled by setting the delivery duration to infinity. The system will not schedule any event after that delivery and the vehicle will stay at the station indefinitely. Unfortunately, this setting would also lead to an inconsistent state of the system since ultimately the vehicle will return to another station. If that station is known in advance, the "return to station" dummy events should be added at the end of the mission list.

In the following chapter we will see how the system should interpret the mission list to derive the schedule as well as the list of stations where overnight stays will occur.

2.3.2.8 Force a vehicle to provide a non-stop service.

The scheduler will have the ability to "attach" the non-stop constraint to a request. If he does, the algorithm will:

- 1) Evaluate only the non-stop insertions, if any.
- 2) Set the earliest pick up time and latest delivery time such that

$$\mathbf{LDT - EPT = flight\ time}$$

This choice will insure that no other request be served through an intermediate stop which would require some extra time over the original non-stop service time.

2.3.2.9 Keep the departure times unchanged.

This constraint will be handled by the algorithm when checking the feasibility of new insertions. If the insertion requires to move forward in time any event already scheduled, it will be eliminated and considered infeasible.

If the scheduler wants such a constraint to be active, it can be expected that feasible insertions will become much more difficult to find.

2.3.2.10 Force an infeasible request to be inserted and/or replace other requests of lower priority.

This requirement is one of the most difficult to handle because it requires a change to the initial schedule rather than an addition to it.

If a last minute request of priority p cannot be inserted into the current schedule, the scheduler has two ways to deal with it:

- 1) Consider that it was presented to the schedule *after* all previous p requests. In that case, all requests of lower priority ($p+1, p+2, \dots$) should be removed from the schedule, and the insertion should be retried. If it remains infeasible, the scheduler can justify the rejection of the request by its lateness with respect to other requests of same priority.

2) Consider that it was presented to the schedule *before* all previous priority p requests. Here, all requests of priority p and less should be removed before the insertion is retried. If it is rejected once more, the justification will simply be that there are already too many requests of higher priority being served. In short, the priority is not high enough.

Let us now investigate how requests should be removed from the schedule.

If the schedule is kept unchanged and the appropriate events (priority $p+1, p+2$, etc...) simply removed, the final result will not be very good. During the previous scheduling process, the insertion of these lower priority events requires pushing forward in time some of the higher priority events already inserted. If the low priority events are removed, the remaining events will still be "pushed forward" and as we have seen, the algorithm will not be able to move them back in time.

It is therefore more appropriate to "remove" lower priority events by returning to the schedule obtained before these events were inserted. The algorithm should save the schedule it has produced in a file, each time a request with a new (lower) priority is about to be inserted.

The new process is described in figure 2.23.

Let us now look at two problems associated with this process.

1) Computation time and number of changes.

As shown in figure 2.23 each time a last minute request needs to be inserted, all lower priority requests have to be re-inserted afterwards. The insertion will therefore become quite long if the request is of high priority; basically the whole schedule would have to be recalculated. Moreover, even though the new request will be served by one or two vehicles at most, the new insertion could affect the missions of other vehicles as well.

If execution speed is critical or if it is important to keep the schedule intact as much as possible, the previous strategy could be changed as follows:

If the insertion in the schedule with priorities 1 and 2 is feasible then

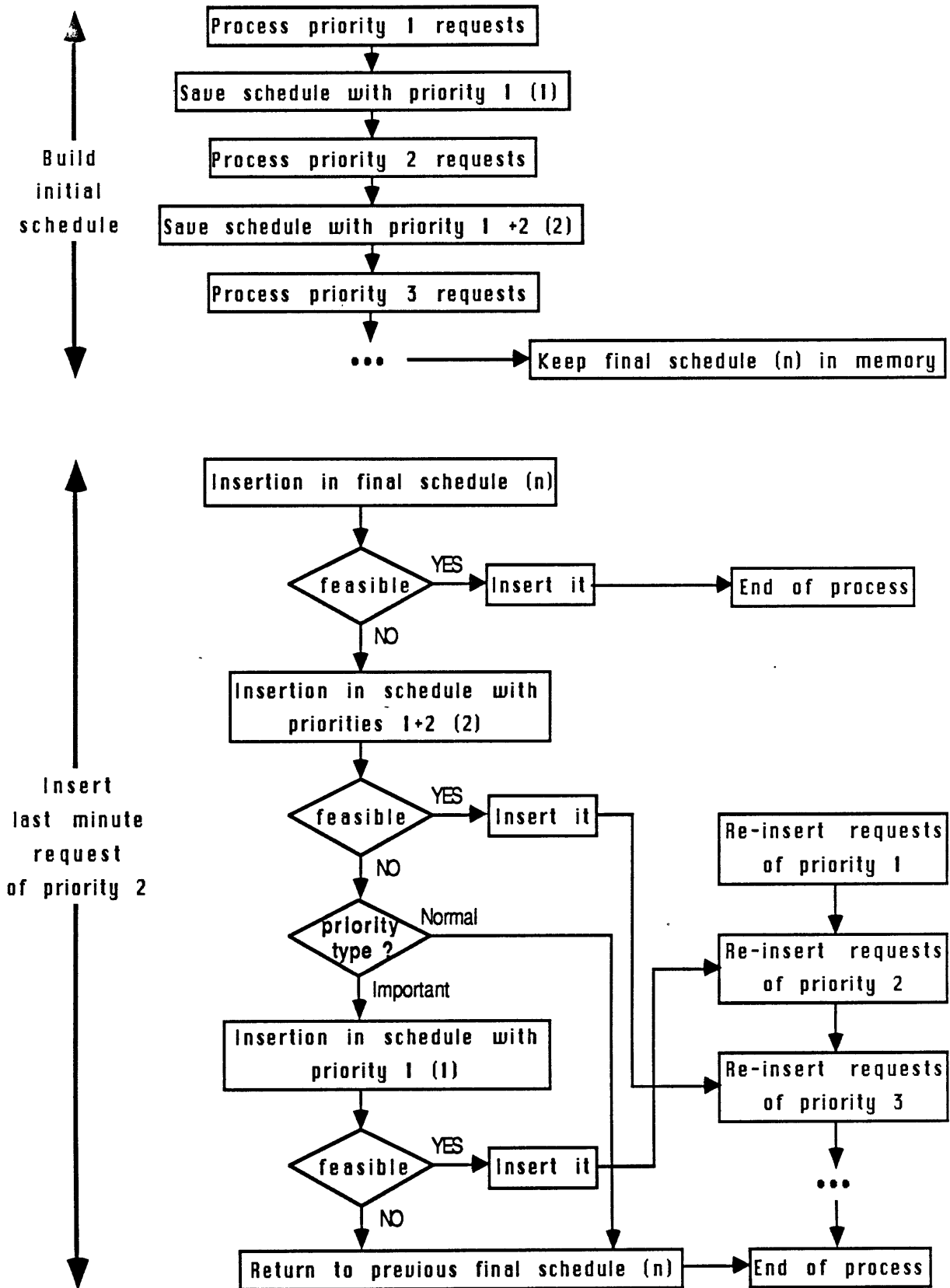


Figure 2.23: Process to modify the schedule when an insertion is not feasible

- keep the new schedule *only* for the affected vehicles,
- reinstall the previous final schedule for all other vehicles,
- re-insert the requests displaced by the new insertion *only*. Modify all intermediate schedules as needed.

Experiments only will show if a schedule produced in this manner is much worse than one obtained by rescheduling all lower priority requests.

Depending on the results there may be some tradeoffs involved between keeping the schedule intact and making it as efficient as possible.

2) Feasible last minute requests.

Let us suppose that before an infeasible priority 2 request was presented to the scheduler, two other priority 1 last minute requests had been successfully inserted. Since these 2 requests were not part of the original list, they will not appear in any of the intermediate schedules saved in data files. If the new process described in figure 2.23 is used, these requests will not be served any more since the system restarts from one of the intermediate schedules. To avoid this undesirable effect, the following steps should be added to the process:

- a) If a last minute request is successfully inserted, add it to a stack.
(each priority should have its own stack).
- b) If a rejected last minute request must be inserted, rebuild the schedule by inserting at each priority level
 - the requests from the appropriate stack if it is not empty,
 - the rejected request when its priority has been reached.
- c) Empty all the stacks and add their requests to the main request list.

Finally, depending on how the fleet is set up, it may be appropriate for the system to give

the scheduler the capability to serve an infeasible request by adding a spare vehicle to the active fleet.

2.3.2.11 Fix schedule up to a cutoff time and reschedule afterwards.

The major difficulty with this feature will be the selection of events that should be kept or even fixed in the schedule once the cutoff time has been selected.

Practically, around that time a typical vehicle will either:

- be at a station, totally empty [case I], or
- be at a station, but with a load in "transit" [case II], or
- be moving from one station to the next with a load on board [case III].

Case I is the most simple to handle. All requests that were served prior to cutoff time are left unchanged into the old schedule; the others are put back into the request list. A new schedule is then initialized in which the vehicle is simply available at cutoff time at the station where it stopped.

Case II requires the scheduler to make some decisions. As in case I, all the requests *entirely* served (pick-up and delivery) before cutoff time will be kept in the old schedule; all those for which the pick-up will occur after cutoff time are put back into the request list. The requests in transit (pick up has been done but not delivery) cannot be handled as easily: two strategies are possible.

- Cancel the pick-up event of the transit request and put the request back into the request list. This makes the old schedule less efficient, but keeps the insertion process simple and free to build the new schedule optimally.
- Keep the pick up event and force the delivery to occur in the new schedule. The old schedule remains untouched but the new one will not be as efficient as possible (within the limits of the algorithm) since it has to deliver the requests in transit .

If the second strategy is selected, the new schedule will have to be prepared with all necessary events appropriately placed:

- the dummy delivery at the beginning for the availability information,
- the deliveries of requests in transit as early as possible (since the algorithm can only push them forward in time),
- the dummy pick-up and delivery at the end, if any "return to station" constraint is active.

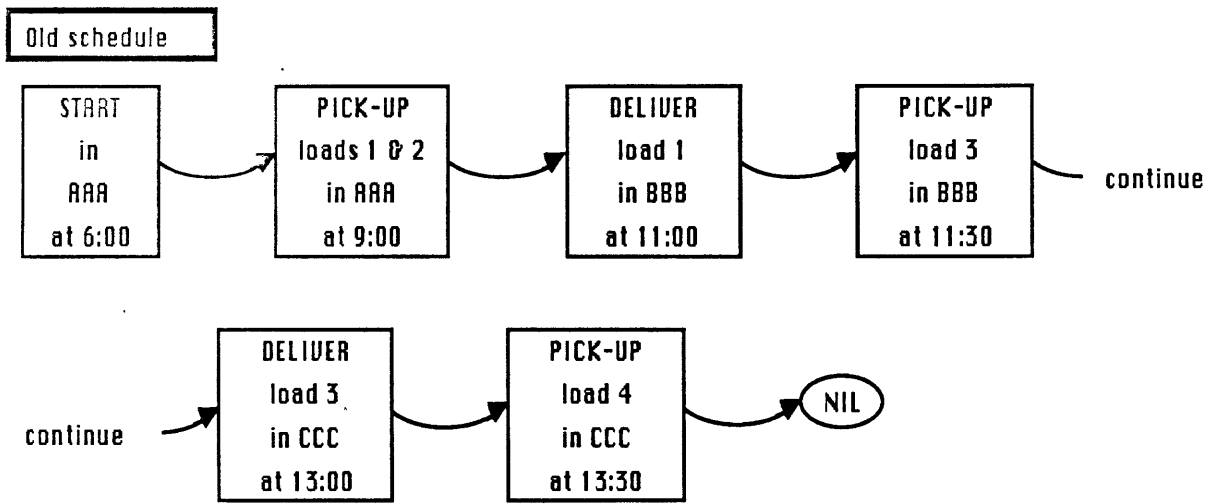
Again if it is important to have a match between pickups and deliveries, the deliveries from requests in transit could be balanced with dummy pickups of the appropriate load at the new station of origin.

Case III will be turned into case I or II as soon as a station is selected as the origin of the new schedule. This station can either be the one the vehicle is coming from or the one it is going to. If the system has to be automatic, it could select the station

- which is after cutoff time (in order to keep the old schedule as untouched as possible),
- or
- which has the least number of requests in transit (in order to allow for more optimality),
- or
- which is closest to cutoff time.

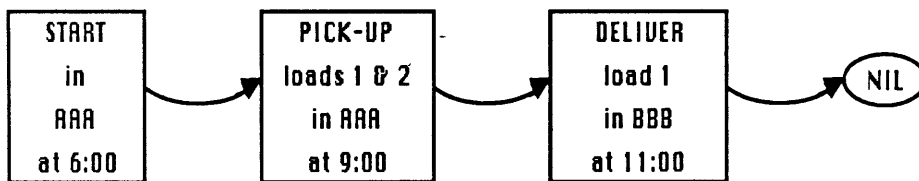
It may be appropriate, however, to leave the choice to the scheduler.

Figure 2.24 summarizes the handling of cutoff time by an example.

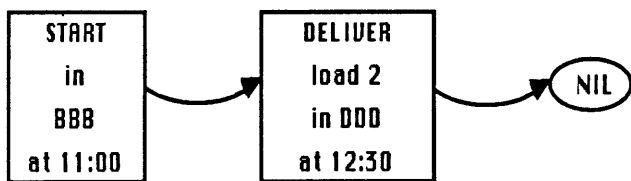


CUTOFF TIME IS 12:00 → **BBB IS SELECTED AS CUTOFF STATION**

Old schedule is changed to:



New schedule



Requests for transportation of loads 3 and 4 are put back into REQUEST LIST.

Figure 2.24: Effect of a cutoff time on the schedule

2.3.2.12 Reschedule missions following a vehicle failure.

This last extension to the insertion algorithm can be handled in two different ways.

1) Re-insertion of affected requests.

With this technique, the rescheduling process is automatic. First, the scheduler makes all spare vehicles available to the system. Then, the insertion algorithm is used to reinsert all the requests affected by the failure:

- requests not yet served (pick-up not done); the request specifications remain unchanged.
- requests not yet delivered; the specifications are changed to reflect the fact that these requests must now be picked-up at the station where the failure occurred.

The advantage of this approach is that it will try to reschedule with minimum incremental time. The solution, however, may not be acceptable to the scheduler if some requests are rejected or if some missions are substantially modified. In order to give the user more control, we suggest also the following heuristic.

2) Manual rescheduling.

Even though we consider this modification as an extension, it may be more appropriate to view it as another decision support tool which can be used in parallel to the insertion algorithm.

Since a failure is a dynamic event which becomes known only when it occurs, the task of the scheduler is to find out which vehicle are, or will be, available after failure time, and figure out if some of these vehicles can be used to solve the problem.

This process can be decomposed in the following way:

1) Get failure information:

- Current time: **CT**
- Vehicle type: **V**
- Estimated repair time: **RT**
- Station at which failure occurred: **AAA**

2) Check if there is a spare vehicle of type **V** at **AAA**. If yes, the problem is solved:

transfer loads to spare and leave with it. If no, find out where the nearest spare of type V is (BBB) and continue with step 3).

- 3) Check if the spare can be moved to AAA and be loaded before the originally scheduled departure time from AAA. If yes, move it. If no, continue with step 4).
- 4) At this point, the maximum delay (MD) in AAA will be:

$$\text{MD} = \min(\text{RT}, \text{trip time from BBB to AAA}).$$

This delay in AAA can be reduced if an appropriate combination of vehicles and stations is used.

If for instance, a regularly scheduled vehicle of type V is at station CCC at time CT, and if CCC lies halfway between AAA and BBB, then it is possible to send the spare vehicle from BBB to CCC and reroute the scheduled vehicle from CCC to AAA. With this scheme, the delay in AAA could be reduced to MD/2 (If RT is greater than trip time from BBB to AAA) at the cost of introducing another MD/2 delay in CCC.

However, if enough slack ground time is available (see 2.3.6), two delays of MD/2 may be much easier to absorb than one delay of MD.

In more detail, the intermediate station CCC should be selected with the following conditions (it is assumed that all spares are available at time CT and that a scheduled vehicles of type V is available in CCC at time CT+WT, [Waiting Time] since it will land in CCC after CT and must be unloaded first):

- trip time from CCC to AAA should be less than MD,
 - trip time from BBB to CCC should be less than MD,
 - [WT+ trip time from CCC to AAA] should be inferior *enough* to MD
(It is not worth relocating vehicles all around in order to reduce MD by 5 or 10 minutes),
 - the delay introduced in CCC should also be inferior *enough* to MD.
- 5) The final station CCC will be the one producing the smallest overall delay (delay in AAA + delay in CCC). If no intermediate station can be found, the nearest spare will be used unless the disabled vehicle can be repaired within relocation time.

Three final remarks can be made about this process.

- a) During the selection of an intermediate station (CCC), the resulting delays have to be inferior enough to MD. How much enough should either be decided by the scheduler or based on what type of delays can be absorbed by the slack waiting time at stations.
- b) The use of one intermediate station was not an arbitrary decision. In reality, a delay could be reduced into smaller pieces, spread over more than two missions but the processing involved was considered unnecessarily complex and lengthy.
- c) The advantage of this approach is that the scheduler has full control over the station CCC that will be selected. If the best choice (timewise) corresponds to a mission that should not be changed (no delay added), it can simply be rejected.

3 THE GRAPHICS INTERFACE.

3.1. Presentation.

As we explained at the beginning of the thesis, A.I.S.E. is a hybrid system that provides both a decision support tool composed of an insertion heuristic, and a graphics environment through which the user can create, control and manipulate a schedule.

This chapter will first review the new generation of computer hardware and software that made the implementation of these ideas possible. It will then describe the principles that underlie the graphics interface as well as the characteristics of the scheduling environment. A specific section will look at how the insertion algorithm and the graphics tool should interact in order to complement each other.

3.2 The new generation of personal computers.

Specific developments have occurred over the past 20 years and lead to the graphics oriented systems that flourish everywhere nowadays.

3.2.1 The mouse.

In hardware, the "revolution" started in 1964 with the invention of a pointing device, called "a mouse", by Douglas Engelbart from the Stanford Research Institute. Typically a mouse is a little box equipped with a system that detects in which direction it is moved. Modern mice have either:

- a ball that rotates as the mouse is moved over a flat surface, or
- an optical scanner that "reads" a grid engraved on a flat platen.

The movement information is interpreted by the computer and used to move in a similar manner a graphic arrow on the C.R.T.. Since the arrow follows exactly the movements imposed on the mouse, the user has the impression of moving the mouse (see figure 3.1). With this system, it is possible to locate the arrow very precisely over any area of the computer screen.

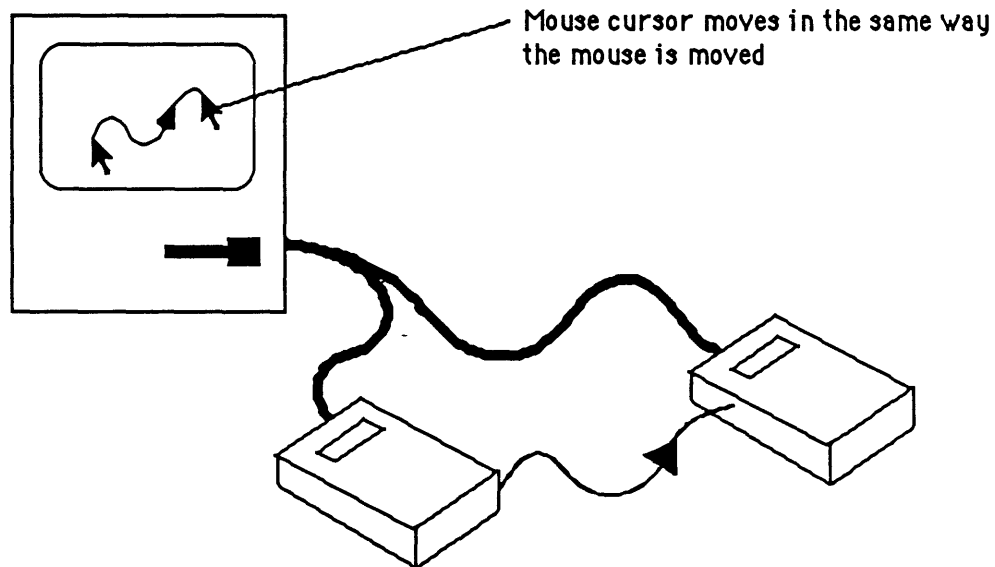


Figure 3.1: Relationship between mouse and cursor movements

A mouse is also equipped with 1,2 or even 3 buttons that are used to trigger actions from the system. These actions will depend on:

- which button was depressed, and
- over which area of the screen the arrow was positioned.

3.2.2. The desktop.

The second major idea came in Software. In 1970, Xerox Corporation formed its Palo Alto

Research Center (PARC) that investigated new ideas on how to make computers more "user friendly"

From this research emerged the idea of desk top environment. The computer C.R.T. is considered as an image of a typical desk; on top of it, lies a certain number of items:

- document folders,
- closed documents represented by graphic symbols (for the type) together with names (symbols are called "Icons"),
- opened documents that appear inside "windows".

A window typically is a partial rectangular view of a document sheet of paper. It can be resized and moved around. The user can look at any part of a document by scrolling it sideways or up and down inside its window.

Finally as on a real desk, many windows will be allowed to overlap but only the top most one will be "active", the document it represents being modifiable by the user.

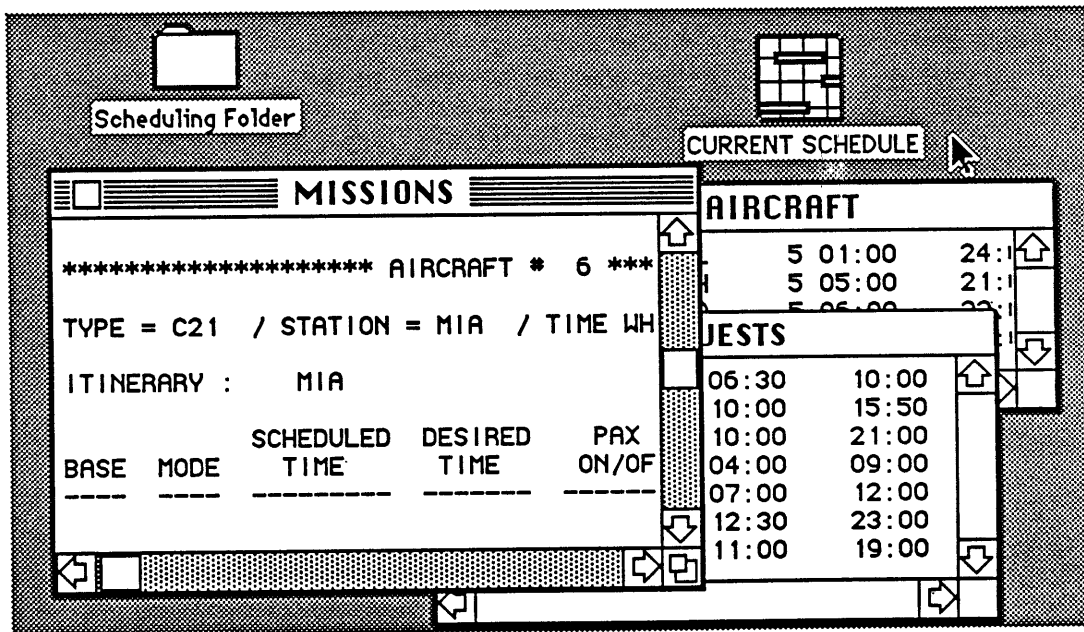


Figure 3.2: A typical desktop display with three windows, one folder and one closed document.

Obviously, the desktop type of display requires a lot of graphics operations. Its feasibility is therefore also due to the dramatic increase in speed of new microprocessors that allows for real time modifications of desktop items.

Both the mouse and the desktop ideas have been incorporated by Apple computer into its new line of machines. This line started in 1983 with the expensive Lisa, followed in 1984 by the Macintosh. A.I.S.E. originally developed on the Lisa, is now implemented on the Macintosh.

3.2.3 The Macintosh environment.

Let us now turn to the Macintosh environment more specifically. We will review in turn the user interface (how the user and the computer communicate) and the special type of programming needed for each application to fit within this interface.

3.2.3.1 The User Interface.

Basically this interface is a close replica of Xerox P.A.R.C. desktop idea. When the machine is started, the desktop appears on the screen with all sorts of icons representing:

- the disk drives connected to the system
- the document folders
- the documents themselves that can either be applications (programs) or data files to be used by these applications.

When a disk drive is opened, the icons corresponding to its content appear in a window or in a folder.

At the top of the screen, Apple has added a new idea: the menu bar. Through it, we will see how the mouse is used to interact with the system.

Looking at figure 3.3, the menu bar appears as a white band containing a certain number of menu names. If the user moves the arrow over a name, presses and holds down the mouse button, an appropriate "pull down menu" appears right under the corresponding name. This pull down menu is a rectangle containing a list of commands that can be executed (Print, Copy, Duplicate, etc...). Holding down the mouse button the user will then move the arrow

down the menu rectangle, highlighting each command as the arrow passes over it. Once the arrow is positioned over the appropriate command, the mouse button is released and the command is executed by the system. This process is described in figure 3.4..

Figure 3.5 describes the elements of windows used in the Macintosh environment. On top of each window a bar is displayed that contains a name of the document presented. If the window is active that name is highlighted. By moving the arrow over the left hand side square and clicking the mouse button the user will close the window.

The window can be dragged around the desktop by moving the arrow anywhere else on the top bar, pressing and holding the mouse button, moving the arrow (a shape of the window follows) and releasing the button. The window can also be resized with the same sequence repeated over the square in the lower right corner. Finally, since each window is a partial view of a document, "controls" are provided to scroll the view in any direction. They appear on the right and bottom side of the window.

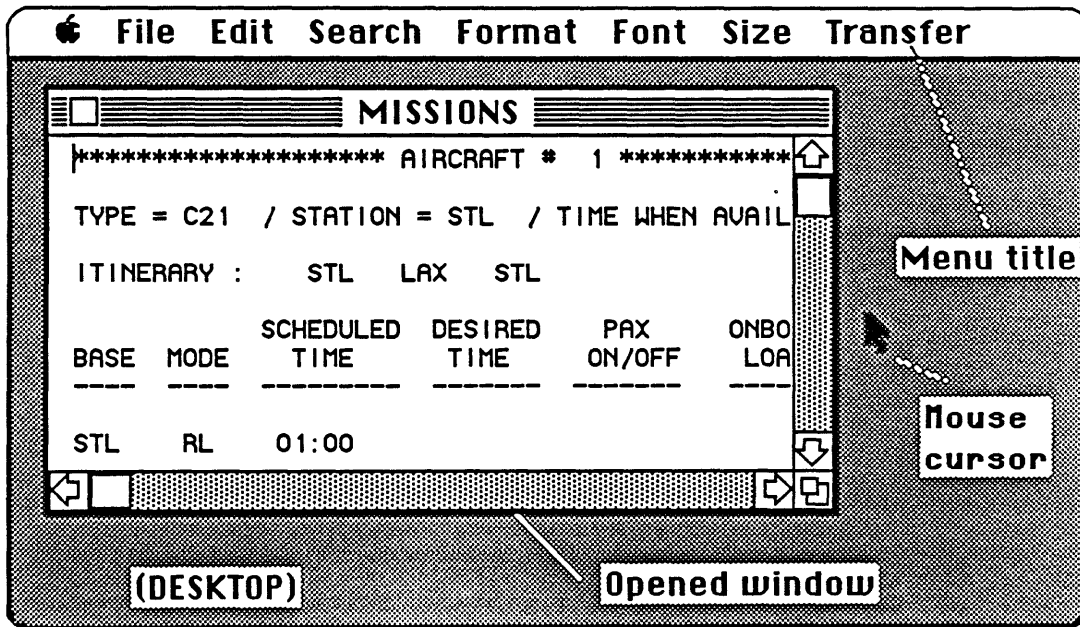


Figure 3.3: The Apple Macintosh desktop

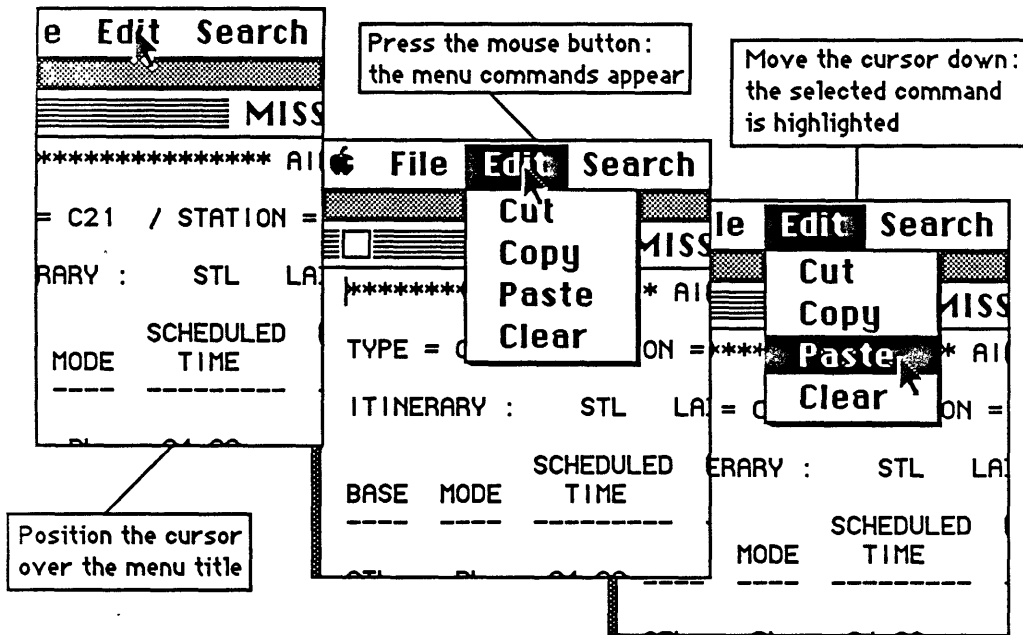


Figure 3.4: Selection of a command from the menu bar

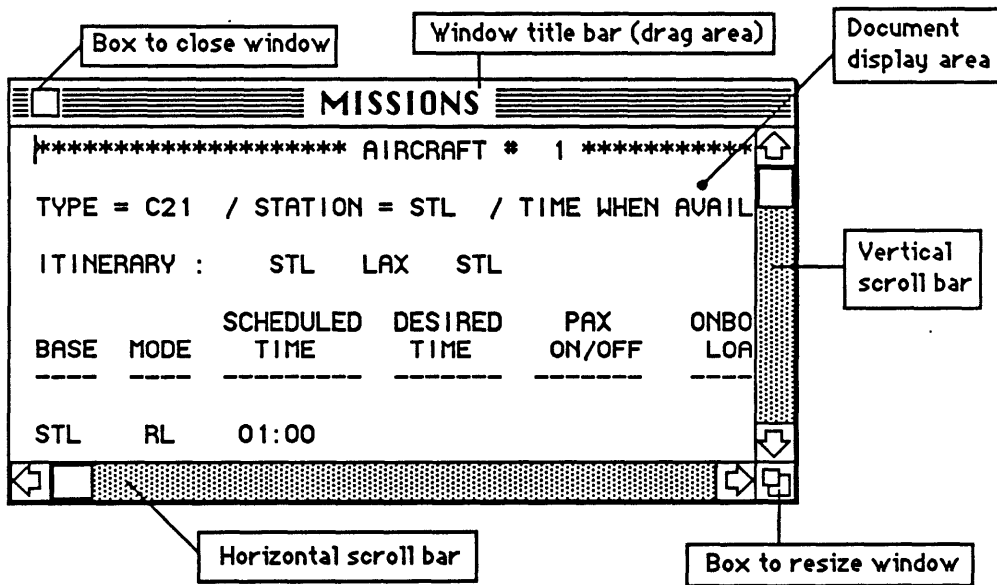


Figure 3.5: A typical Macintosh window

A control typically has :

- 2 arrows that when clicked over with the mouse make the system scroll the document view by a fixed amount.

- a square ("elevator") that can be dragged along the control element to position the view anywhere over the document.

The dragging is done as usual (point, click, hold, move and release). When the elevator is at one end of the control, the corresponding end of the document will be displayed in the window (see figure 3.6).

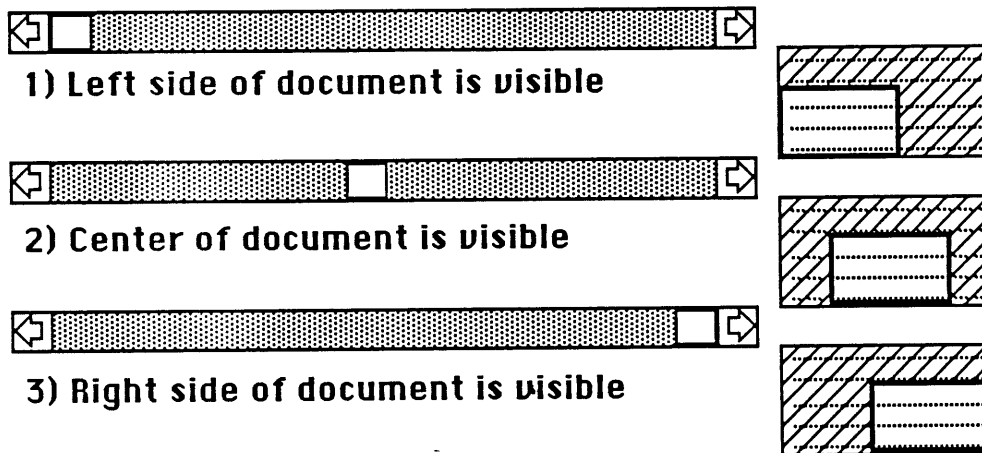


Figure 3.6: Relationship between the "elevator" position and the visible area of a document

When an application such as the scheduling system is started (by clicking twice on its icon), it takes over the system, creates its own desktop environment with the menu bar and opens appropriate windows on top of it. We will now look at how such an application keeps track of the user interaction through the mouse and the keyboard as well as how it monitors external events such as messages arriving from a network.

3.2.3.2 Application programming.

Two special techniques have to be used in order to put together applications such as A.I.S.E. let us review them successively.

1) Event driven system.

In order to exploit the desktop capabilities each Macintosh application must be programmed as an "event driven system". In such a system the computer is considered as a central node that constantly monitors events occurring "around" it and takes appropriate actions depending of the event detected. The types of events detectible by the system are:

- Mouse button up or down,
- Keyboard key up or down,
- New diskette inserted,
- Make an opened window active (bring it to the front, etc..),
- Update a window (redraw the visible part of its content),
- "Appletalk" external network event,
- Application defined events.

In order to implement such an interaction, the computer operating system accumulates into a queue the codes and characteristics of the successive events it detects. Each application, in turn, consists of a main loop that constantly:

- Reads the next event in the queue, and
- Depending on its type,
 - reads its characteristics from the appropriate memory locations (which was pressed, what type of message is coming from the network, etc...).
 - Executes the appropriate processing.

Figure 3.7 summarizes this organization.

This process is made possible because most events originate from a human operator who is very slow compared to the processing speed of new microprocessors. For example, when a user types some text into the system each key stroke is considered as an individual event that is processed (find the character corresponding to the key and display it at an appropriate position in the active window) through a full iteration of the loop.

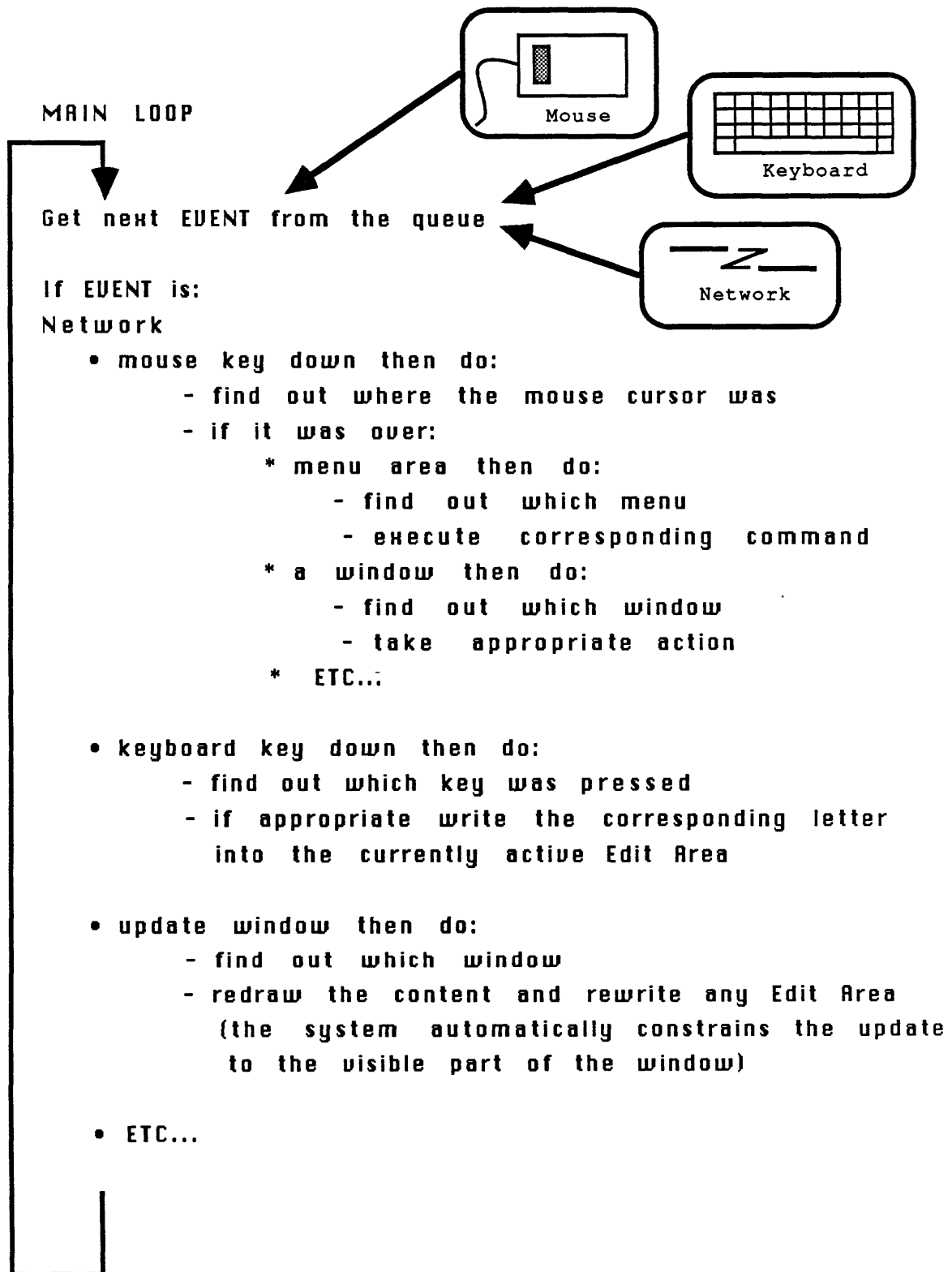


Figure 3.7: Typical event driven system

The program is fast enough to even loop a few times between each keystrokes and therefore detect and process other types of events that might occur while the user is typing (a message coming from the network, for instance).

The combination of sequential event handling and processor speed actually gives the illusion that many activities can be executed at the same time. This last point highlights the need for this type of programming. Since no single process takes over the machine, the user can switch easily from one environment to another by changing the active window,

- interrupt its current activity to resize or move a window, execute a command from the menu bar, etc... , and
- receive warnings about events occurring in the background of the main activity.

Moreover, none of these capabilities requires the system to terminate the current process that only needs to be temporarily interrupted.

2) Graphic objects.

It is possible to expand the desktop and mouse philosophy beyond simple operating system applications.

As we have seen, the principle behind the desktop concept is "object manipulation". Typically, the user selects a "graphic object" (window, folder icon, document icon, etc...) by clicking with the mouse button while the arrow is on top of the object image. It is then possible to:

- move the object on the screen, by keeping the mouse button down and dragging the image around, or
- act on the object, by selecting a command from the menu bar.

Derived from these concepts, the idea behind the experimental scheduling electronic drawing-board has been to consider flights (or more generally movements from one station to another) as individual objects that can be manipulated by a human scheduler with the methods described above.

When object oriented applications are designed, a certain number of general principles have to be considered. Let us review here some rules that should be kept in mind while programming such applications.

Rule 1: Each object on the graphic screen is uniquely related to a data record in the underlying data base.

Rule 2: Each time an object is acted upon, the corresponding data record must be updated before the object graphic representation.

Rule 3: If any of the updated information is subject to certain constraints, these constraints must be checked before the data base and the display are changed.

Rule 4: The system should distinguish between two types of constraints :

- those that are simple boundaries on the attributes of objects (the fields of the data base records); in this case, there should be three steps in the processing cycle: calculate the new values of the attributes, check them against the constraints, and either update the object location and attributes or cancel the process if there is any violation.
- those that relate attributes of different objects to each other; the number of steps in the cycle should be greater than three: calculate the new values of the attributes, check them against the constraints either cancel if there is any visible violation, or calculate new values of attributes of objects affected by the constraints, again check them for feasibility, etc..., if everything is feasible update each affected object graphic image.

These cycles are shown in the following figure.

Rule 5: Each object has two types of attributes.

- those that have a fixed value; their graphic representation will be fixed and belong to the background of the display. It may be selectable.
- those that have variable values; their representation will be selectable and movable over the background.

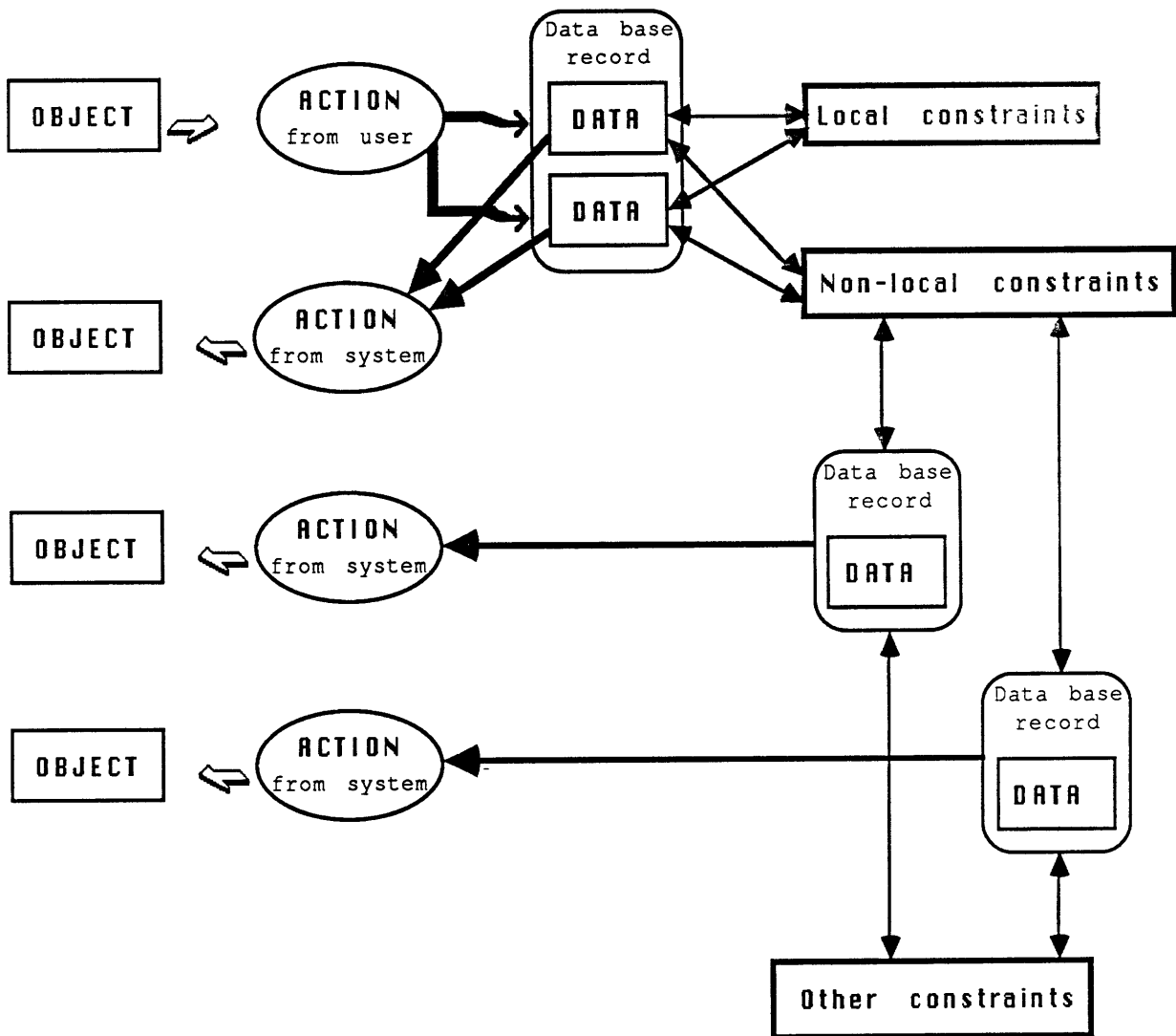


Figure 3.8: Graphic object manipulation cycle.

Rule 6: The graphic interface should be entirely separated from the rest of the application. It should act only as a transfer between the actions of the user on the screen (and with the mouse) and the data base records as well as procedures to be executed (figure 3.9).

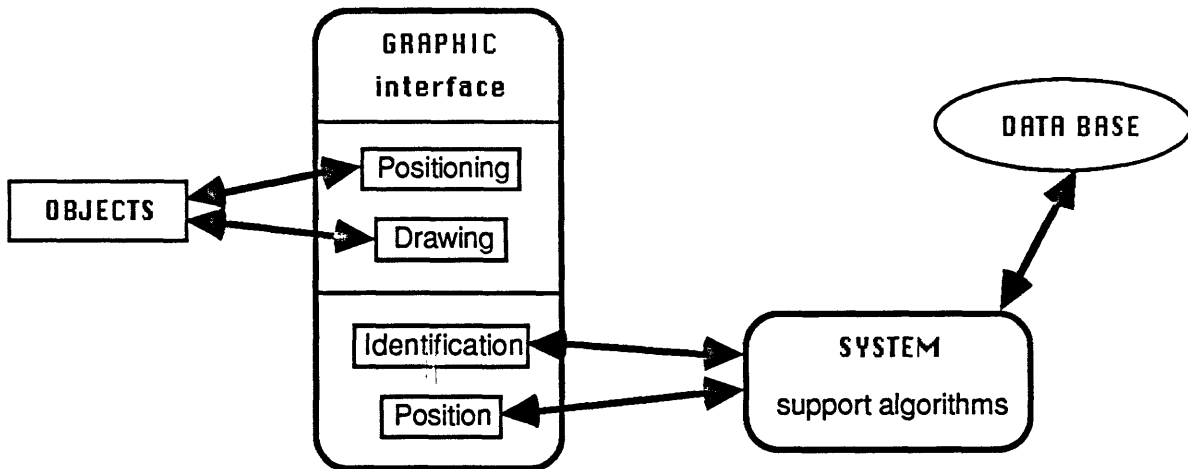


Figure 3.9: Graphic interface organization.

3.3 The graphic scheduling environment.

3.3.1 O.S.A. scheduling organization.

A description of O.S.A. activities was already given in section 1.3. We will look here at the personnel organization within the scheduling division and infer from it a possible setup of the support computer system. [Note: in the following description, we will call day 0 the day the missions are flown, day -1 the day prior to day 0, etc...]. Looking at a typical cycle of activity, we can distinguish between the following people:

I) The Request Handler.

From day -15 to day -3 or -2 requests are transmitted to the scheduling department in two steps:

- first, the request specification is electronically transmitted to an O.S.A. mainframe computer where it is added to a request data base.
- second, the request originator calls the Request Handler (RH.) to check that the request has been properly transmitted, confirm it and perhaps modify it or remove it.

The role of the R.H. is to update/modify requests already entered into the data base and make sure that they are all consistent and confirmed.

2) The Schedule Planner.

On day -2, the Schedule Planner (S.P.) uses all the requests accumulated into the data base to create a first draft of the schedule: the schedule plan.

3) The Schedule Controller #1 (S.C.1).

On day -1, the S.C.1 does flight management on the schedule plan. He adds to mission specifications the actual tail number of the airplane that is going to be used, the names of the pilots etc... Last minute requests (or modifications to requests) are usually handed to him. If they can be handled by the current schedule, he will modify the flight plane accordingly. If they cannot, S.C.1 will interact with S.P. to see how the schedule can be modified with minimum disruptions.

4) The Schedule Controller #2 (S.C.2)

On day 0, the S.C.2 follows the execution of each mission. If an airplane failure occurs, he will be in charge of all necessary re-scheduling and will have to contact all affected passengers.

Practically, two individuals occupy the positions of S.C.1 and S.C.2. They both rotate constantly: the S.C.1 on day -1 becomes S.C.2 for the same schedule on day 0. On day 1, he becomes again S.C.1 for the schedule of day 2, etc...

5) The Archiver.

After the missions have been flown, the Archiver will retrieve some operational data and store them into another data base. Stored information includes number of hours flown, fuel consumption, delays observed etc... At certain dates, the Archiver will produce summary reports about O.S.A. operations.

These roles are summarized in figure 3.10.

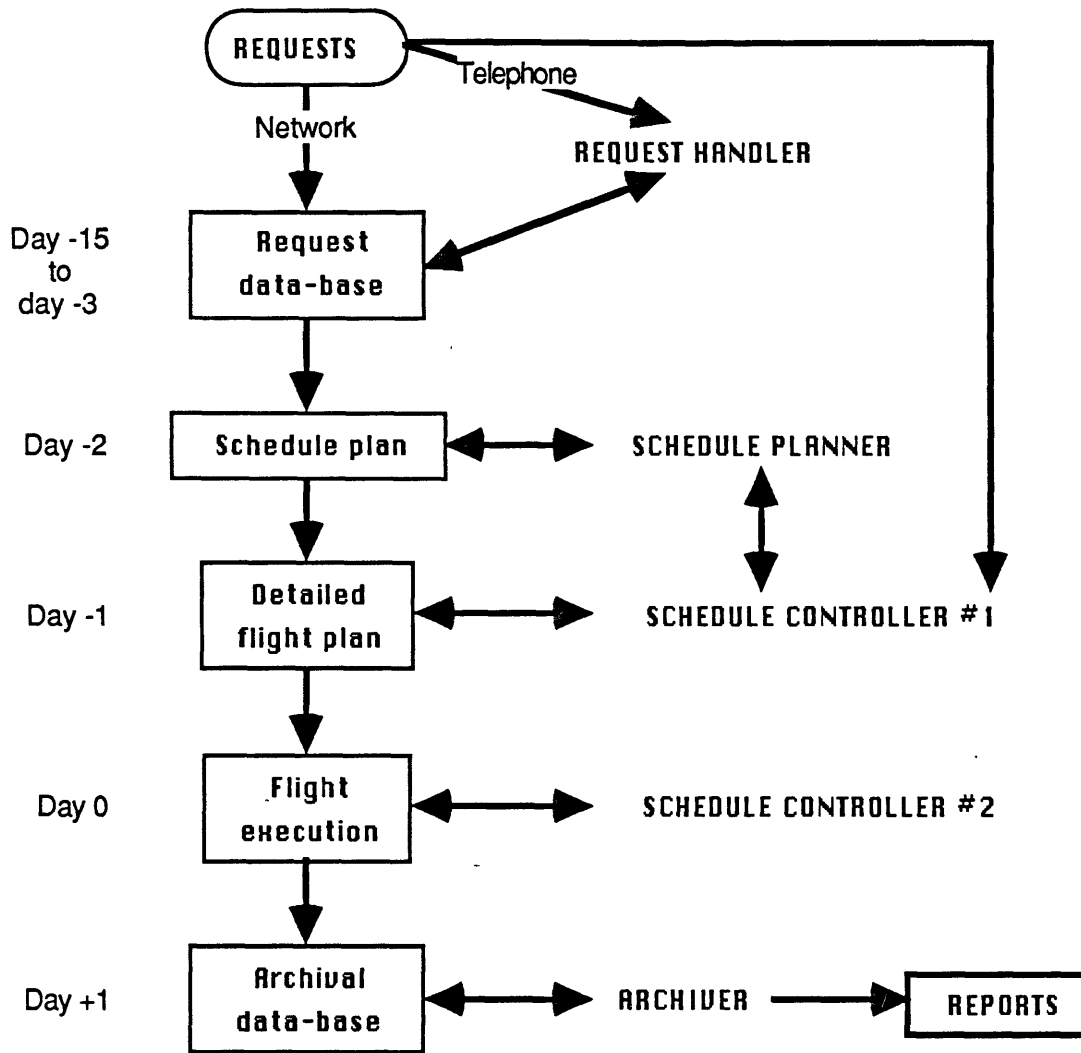


Figure 3.10: O.S.A. scheduling organization.

The next figure shows how A.I.S.E fits inside this organization. The following comments clarify the set-up.

- 1) All the Macintosh "stations" will be connected by a Local Area Network to each other (mail and message capability.) as well as to a main hard disk (shared data base capability).
- 2) The R.H. will be able to work either with the O.S.A mainframe computer or directly within the Macintosh network environment. In any case, request information available on the mainframe will be transmitted to the request file of the Macintosh system data base.

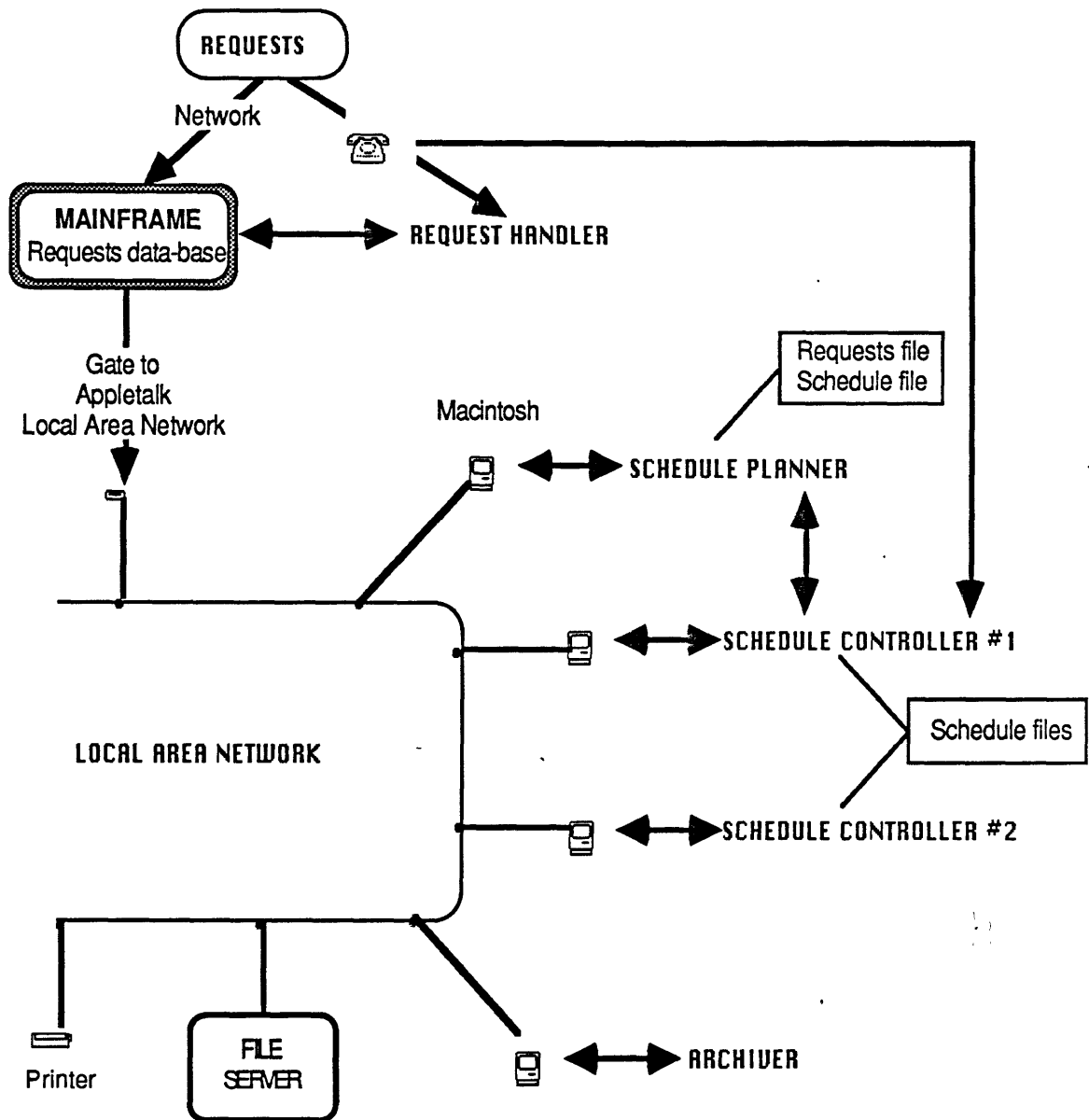


Figure 3.11: Physical set-up of the new scheduling organization.

- 3) The schedule planner will access most data from the main hard disk and will produce the schedule plan using both the insertion algorithm and the schedule drawing-board. The resulting plan will be saved back on the main electronic hard disk.
- 4) The schedule controllers will access the schedule plans with the electronic drawing board. The support algorithm will only be used on day 0 if an airplane becomes disabled. The actual

execution of missions (real landing times etc...) is recorded into an operational data base.

5) The archiver will retrieve appropriate information from the operational data base and update the archival data base from which summary reports will be produced. This function can certainly be automatized entirely.

3.3.2 The Schedule electronic drawing-board.

The drawing board appears as a typical desktop with its menu bar and windows (figure 3.12).

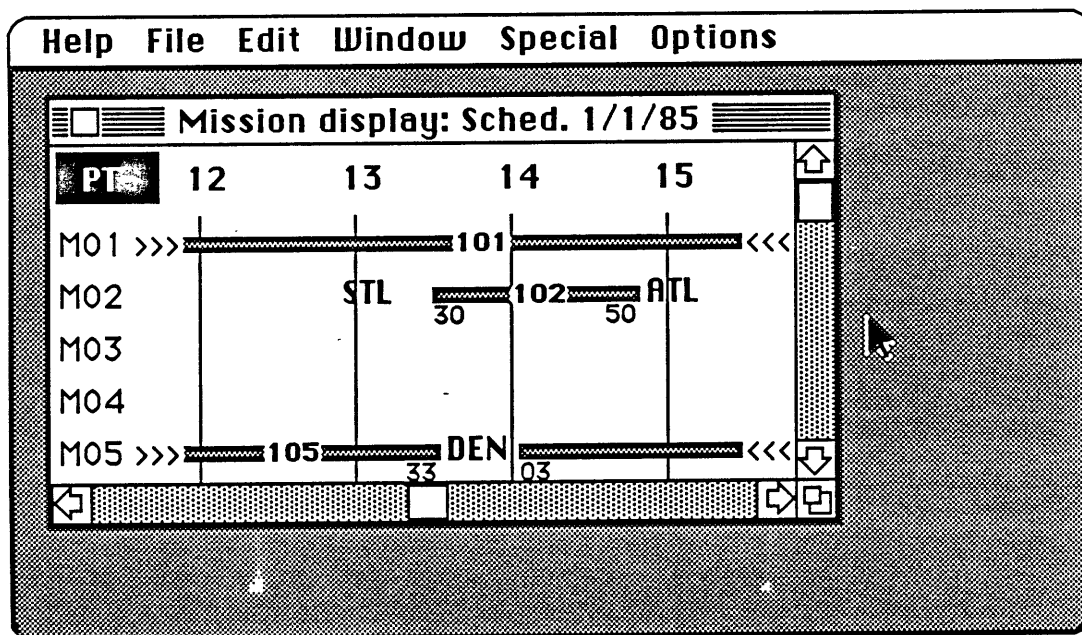


Figure 3.12: A.I.S.E. desktop display.

In this section we will look successively at the two types of windows that can be used and discuss the commands available from the menu bar.

3.3.2.1 The mission display window.

Inside this window appears part of a mission display [ref. 1] that is organized with time of day running horizontally and missions assigned vertically (see fig. 3.13). Let us review each element successively.

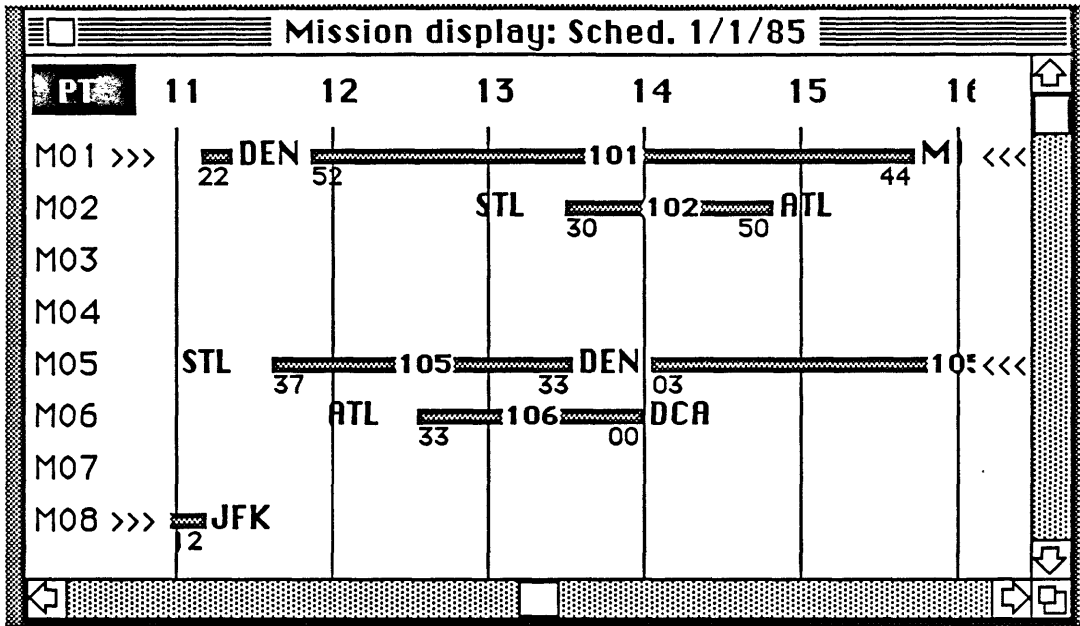


Figure 3.13: The mission display window.

1) Flights.

Each mission consists of a series of successive flights that appear one after each other throughout the day. As we discussed before, each flight is an object that can be moved around the display and inserted into any mission or moved within a mission. The flight graphic image is interpreted as follows:

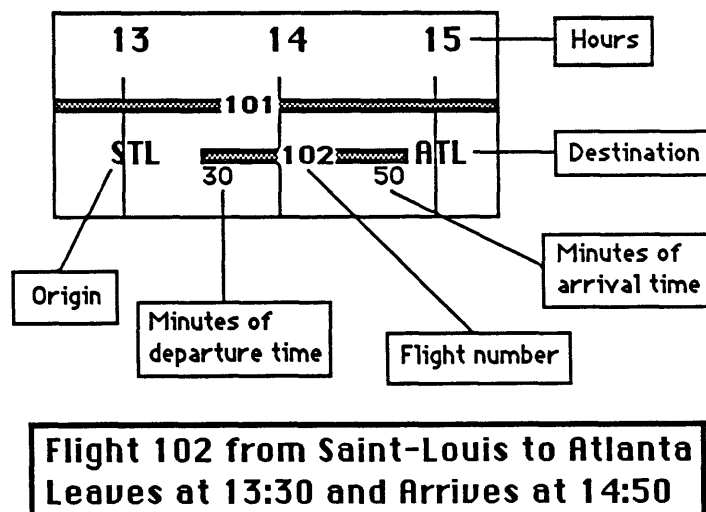


Figure 3.14: Image of the "flight" object.

Figure 3.14 shows how such a flight can be moved.

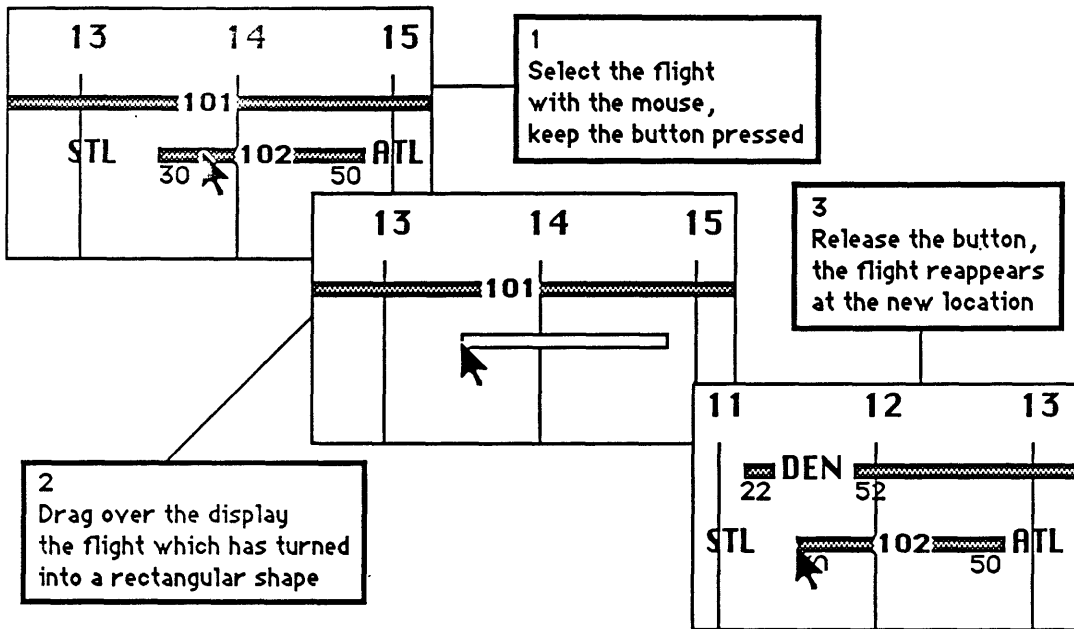


Figure 3.15: Moving a flight object over the display.

2) Display Area.

The area of the mission display visible in the window is divided into three sections.

- The left column lists the numbers of the visible missions, the time zone used to display time and a ">>>" sign if a mission has any more flights at the left of the visible area. Both the mission number and the time zone indicator will be selectable with the mouse. When the time zone is selected, the user will be able to change it. When the mission is selected, the user will be able to act upon it with commands from the menu bar.
- The central area displays the individual flights of each mission over a time grid that has a vertical bar for each hour.
- The left column contains a "<<<" sign if a mission has any more flights at the right of the visible area.

3) Window.

The window itself has the following standard items:

- A title bar that can be used to move the window around the desktop.
- A box on the left side of the title bar to close the window.
- A box on the lower right corner of the window to resize the window. If the window is resized horizontally, on the central part of the display is resized accordingly; the left and right column have a fixed width.
- A vertical scroll bar : The square is used for continuous scrolling, the arrows that scroll one mission at a time. If the mouse button is pressed while the cursor is on the gray area between the square and one of the arrows, the display is scrolled one full window at a time.
- The horizontal scroll-bar follows the same principles except that only the central will be scrolled and the arrows will allow to scroll one hour at a time.

4) Menu Commands.

The following commands will be available under each menu title.

- **Help menu.**

This menu will include a Help command that will display a window with explanations about the action currently undertaken by the user. At any point in time the user should be able to get help panel on any command available in the system.

Under Help, we will find a succession of "desk accessories" such as calculator, calendar, clock etc... that can be brought over the desktop at any time.

- **File menu.**

It will include the following commands:

- New mission display : a completely blank mission display window appears on the desktop.
- Open mission display : a list of schedule files appears first for selection, the mission display window is opened afterwards with the flights corresponding to the schedule selected.
- Close mission display : closes the window and asks the user if the changes made (if

any) are to be saved on the file.

- Print : the user can print a certain number of standard reports with mission information.

- **Edit menu.**

It will include :

- Undo last change : allows the user to cancel the effect of some modifications made to the display (usually by mistake).
- Cut : an object such as a flight or an entire mission or even the entire schedule can be removed from the display and saved into a temporary clipboard.
- Copy : same as cut except that the object affected is not removed from the display.
- Paste : bring back on the next available slot the object that has been put into the clipboard.
- Select all : the entire schedule can be selected in one command (in order to copy it, for example).

- **Window menu.**

Enables to select the type of the next window to be opened on the display :

- Mission display
- Station display (see 3.3.2.2)

- **Special menu.**

This menu will include most of the commands that do not fit under any of the other categories.

- Information : when an object is selected, this command will enable the user to access any data that is related to that object and not currently visible on the display. This information will appear inside a window that will be automatically opened over the mission display window; if appropriate, the user will be able to modify data at that point. Three types of information windows will be implemented:

* Schedule information :

<u>INFORMATION on Schedule</u>	
Name:	Sched. 1/1/85
Date created:	12/30/84
Last modified:	12/31/84
Active options:	
Crew duty day	<input type="checkbox"/> ON (16:00)
Minimum ground time	<input type="checkbox"/> OFF
Keep departure times unchanged	<input type="checkbox"/> ON

Figure 3.16: Schedule information window.

* Mission information (the editable items appear inside rectangles):

<u>INFORMATION on Mission</u>		Schedule:	Sched. 1/1/85
Number:	14		
Aircraft type:	C12		
tail number:	<input type="text" value="USMAC-1234"/>		
capacity:	6		
return constraint:	<input type="checkbox"/> ON		<input type="text" value="(23:30 in ATL)"/>
Crew	Captain:	<input type="text" value="Cpt. J. Smith"/>	
	Copilot:	<input type="text" value="Sgt. A. Jones"/>	
	Mechanics:	<input type="text" value="Lt. F. Charles"/>	
Starts:	06:15		
Ends:	16:55		
Data status:	<input type="text" value="SCHEDULED"/>		

Figure 3.17: Mission information window.

* Flight information:

<u>INFORMATION on Flight</u>		Schedule: Sched. 1/1/85	
Number: 104			
Origin:	<input type="text" value="JFK"/>	Leaves:	<input type="text" value="06:15"/>
Destination:	<input type="text" value="DCA"/>	Arrives:	<input type="text" value="07:55"/>
Flight time:	<input type="text" value="01:40"/>		
Aircraft capacity:	6	Loads on board:	
<u>Request Load EPT LDT Priority</u>			
<input type="text" value="53"/> <input type="text" value="3"/> <input type="text" value="06:00"/> <input type="text" value="10:00"/> <input type="text" value="2"/>			
<input type="text" value="68"/> <input type="text" value="3"/> <input type="text" value="06:15"/> <input type="text" value="11:30"/> <input type="text" value="3"/>			
Constraints	stay on ground:	<input type="text" value="ON"/>	
	non-stop:	<input type="text" value="ON"/>	
Data status:	<input type="text" value="SCHEDULED"/>		

Figure 3.18: Flight information window.

- Auto-create : this command will be used to create a new schedule using the insertion algorithm; It will be available only if the current mission display is empty. Before any calculation starts, the user will be prompted for the name of the requests list to be used as well as the name of the schedule created.
- Insert request : the user will be able to insert a single request into the current active schedule. He will be prompted for the characteristics of the request, and after the insertion the display will scrolled automatically to show the flight with which the request will be served. If needed, all displaced flights could also be highlighted.

If the request cannot be inserted the system will warn the user and give him a certain number of options depending on how many of the items discussed in chapter 2 have been implemented. The choices could be :

- * **OK** : the request is added to the list of rejected requests.
- * **Force after same priority requests** : (see 2.3.2.10) insert the request after all others of same priority and reschedule all lower priorities.
- * **Force before same priority requests** : (see 2.3.2.10) reschedule all requests of same priority with the request appearing at the top of the list; reschedule all lower priorities.

Again the system will scroll the display over the appropriate flight or announce the unfeasibility.

- **Cutoff time** : (see 2.3.2.11) The system will first ask the user to give the desired time. It will then scroll horizontally the display in order to have the cutoff time at the center, a gray line will show the separation and all affected flights will be surrounded by a frame. At that point the user will have a chance to select some of the flights that should not be changed : they will become highlighted inside their box (figure 3.19). When the selection is done the user will tell the system to go ahead and reschedule. Every eliminated requests will be put back into the request list.

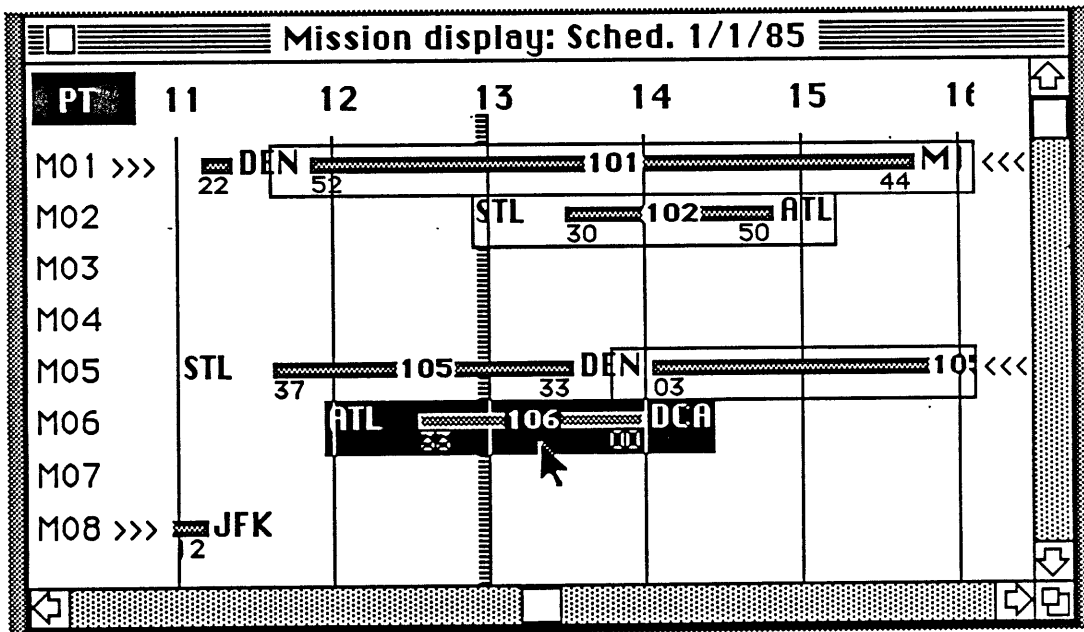


Figure 3.19: Mission display with cut-off time.

- Failure : this command will be used to start the process described in 2.3.2.12. The system will first ask for the number of the mission affected and the station at that the airplane is grounded. The support algorithm will then be used to propose some solutions that can be rejected by the user.
- Keep airplane on the ground (ON/OFF) (see 2.3.2.7): the user selects the flight after which the airplane should stay on the ground, and sets the constraint on. If the flight selected is already subject to the constraint, this command allows to remove it .
- Keep flight non-stop (ON/OFF) (see 2.3.2.8): this command works like the previous one.

- **Option menu.**

This menu will be available in order to select the options that should be active when the insertion algorithm (see 2.3.2.3) is used. Its items will be :

- Crew duty day : the system will give the user two choices :
 - * constraint on; enter length of maximum duty day .
 - * constraint off.
- Minimum ground time: (see 2.3.2.5) as above the choices will be :
 - * constraint on; enter minimum ground time.
 - * constraint off.
- Keep departure times unchanged (see 2.3.2.9):
 - * constraint on.
 - * constraint off.

This option will be used only when the algorithm inserts a single request into an "old" schedule.

3.3.2.2 The Station display window.

Nearly all the characteristics of the mission display described above, are exactly the same with the station display. This section will cover only the differences.

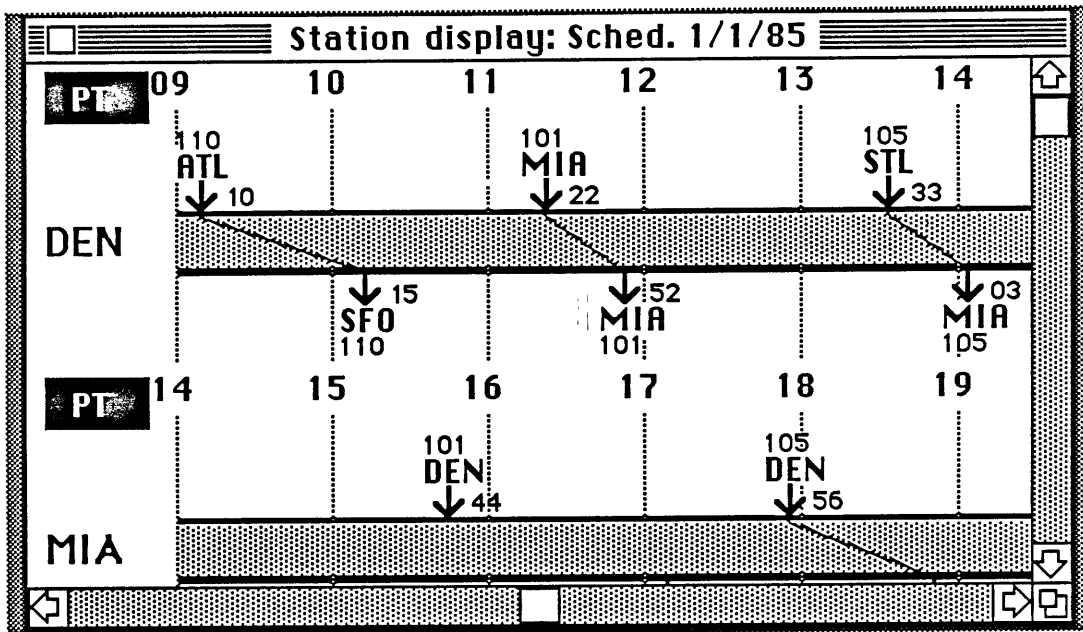


Figure 3.20: The mission display window.

1) Flights.

Flights are handled differently in the station display; they are represented graphically by two objects rather than one. This is due to the way the station information is displayed; flights arrive at the station over the "station line" and depart from it underneath.

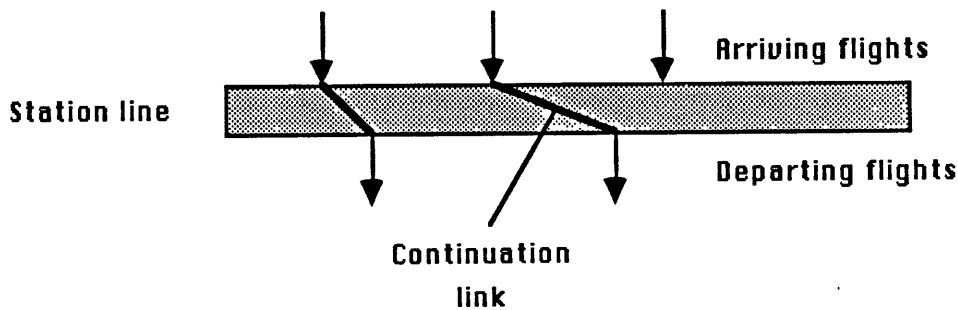


Figure 3.21: Organization of flights in the station display

The flight objects will therefore be :

- a flight arrival

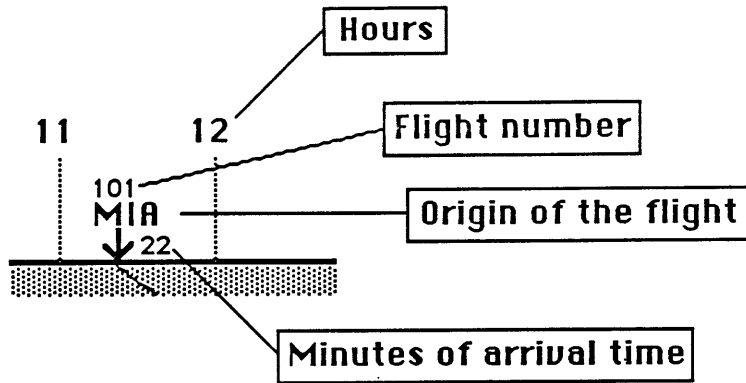


Figure 3.22: Arriving flight graphic object.

- a flight departure

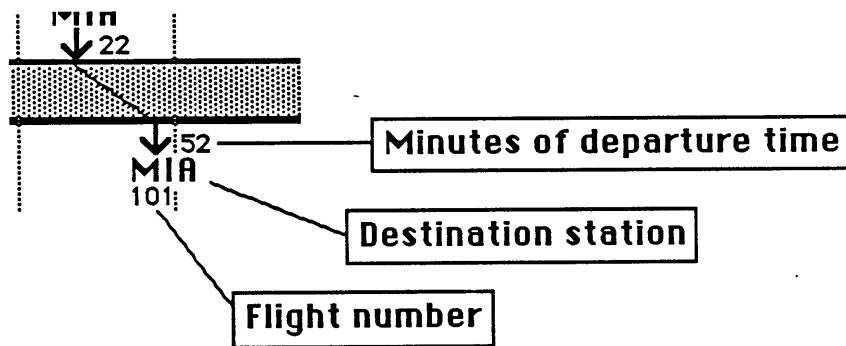


Figure 3.23: Departing flight graphic object.

Both objects will be attached to two different stations since it is unlikely that a flight will take-off and land at the same base. Moreover, a continuation link (see figure 3.21) will relate an arrival to a departure if the corresponding flights are sequentially served by the same airplane.

As before, flights can be moved in the usual fashion:

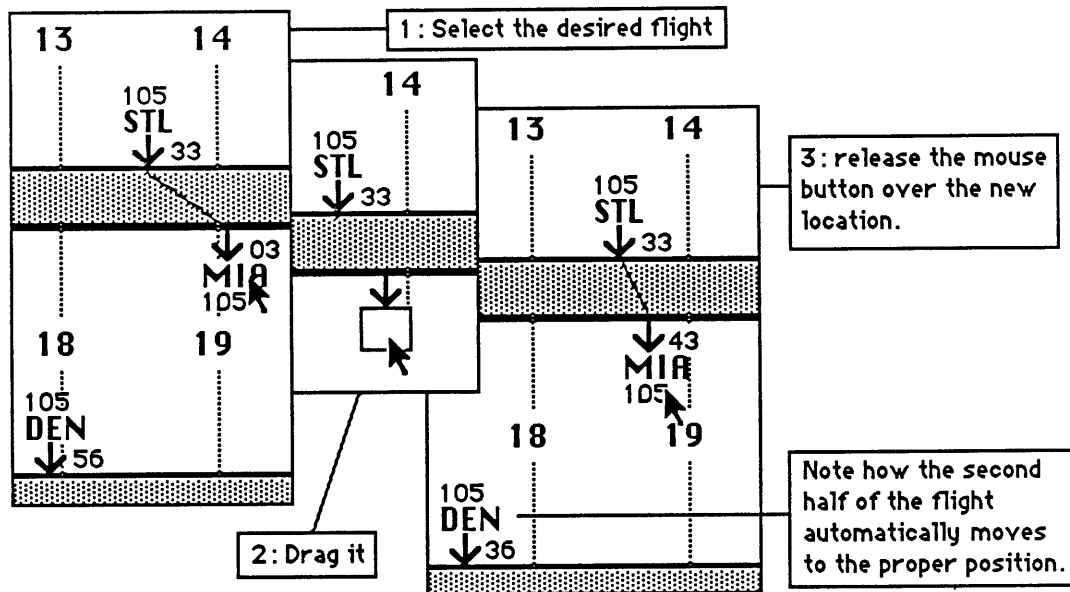


Figure 3.24: Moving a flight on the station display.

2) Display area.

It was decided to allow the display of only two stations at a time. For this reason, the window is divided into 3 sections.

- The left column that includes the station names and the time zone indicator. Both can be selected as before.
- The upper station area.
- The lower station area.

3) Window.

It is again a very standard type of window with the following exceptions:

- Horizontal scrolling does not affect the left hand-side area.
- If the window is large enough to display entirely both stations areas, the vertical scroll bar will disappear.
- It will also be possible to scroll horizontally one station area at a time. If two stations are shown with the same time zone and if the difference between the times on each area is equal to the flight-time between the stations the scheduler will see the departure of a flight

just over its arrival. This is quite convenient when a flight has to be moved since the effect of the change at the next station can be observed instantly as in figure 3.24.

4) Menu commands.

All the commands described above will be available when working with the station display.

The only differences will be :

- **Special menu**

When the station name at the left of the display is selected., a station information window can be displayed with the Information command. This window is shown in the next figure with all editable items displayed inside rectangles.

The constraints in effect at a station will all be selectable from this window rather than from the option menu. This set-up is preferred since it is unlikely to have a single constraint apply to every station of the network.

<u>INFORMATION on Station</u>		Schedule: Sched. 1/1/85
Name:	Kennedy airport	
Code:	<input type="text" value="JFK"/>	
Latitude:	40°38' N	Longitude: 73°47' W
Time zone:	Eastern time	
Phone number:	<input type="text" value="(516) 123 4567"/>	
Constraints	runway length:	<input type="text" value="OFF"/>
	curfew:	<input type="text" value="ON"/> <input type="text" value="23:00 to 05:30"/>
	max. gates:	<input type="text" value="ON"/> <input type="text" value="3"/>
	max. K-loaders:	<input type="text" value="OFF"/>
	fuel availability:	<input type="text" value="OFF"/>
Last update:	<input type="text" value="January 1 1985"/>	

Figure 3.25: Station information window.

3.3.2.3 Dialog windows.

As we discussed in 3.2.3.2, each time the user wants to change the characteristics of an object, a certain number of constraints have to be checked before the modification can be accepted. However, if a violation occurs the system will always have to indicate to the scheduler which constraint was not satisfied. This is done with "Dialog windows" which display an appropriate message together with an "acknowledgment" box. The user has to press the mouse button while the cursor is over that box in order to remove the window and resume normal execution; the object is left unchanged. The following figures show two examples of possible dialogs.

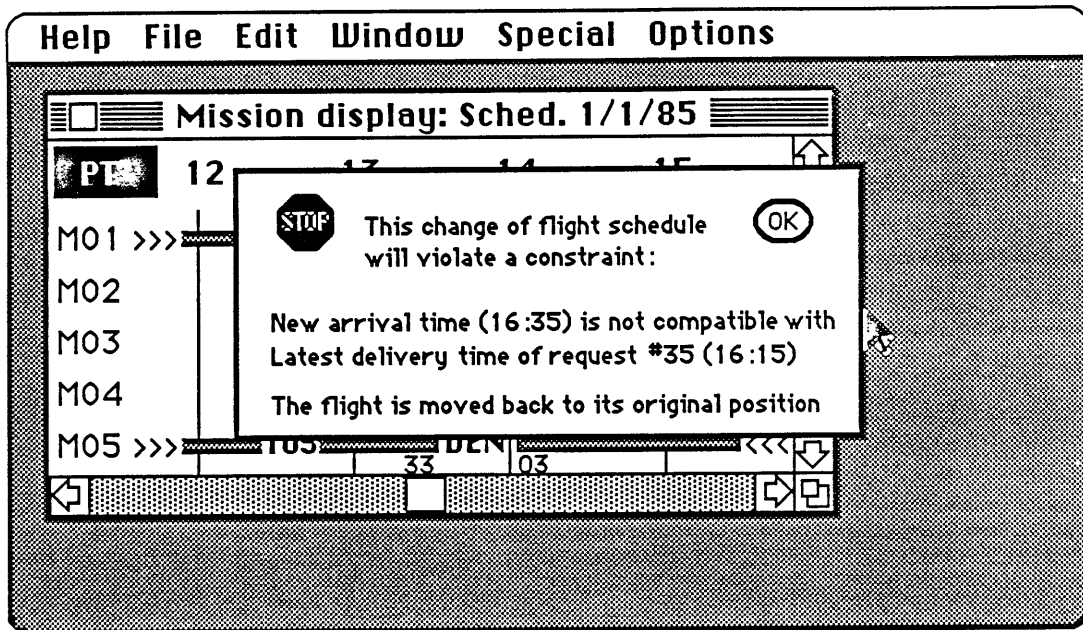


Figure 3.26: Dialog window in the mission display

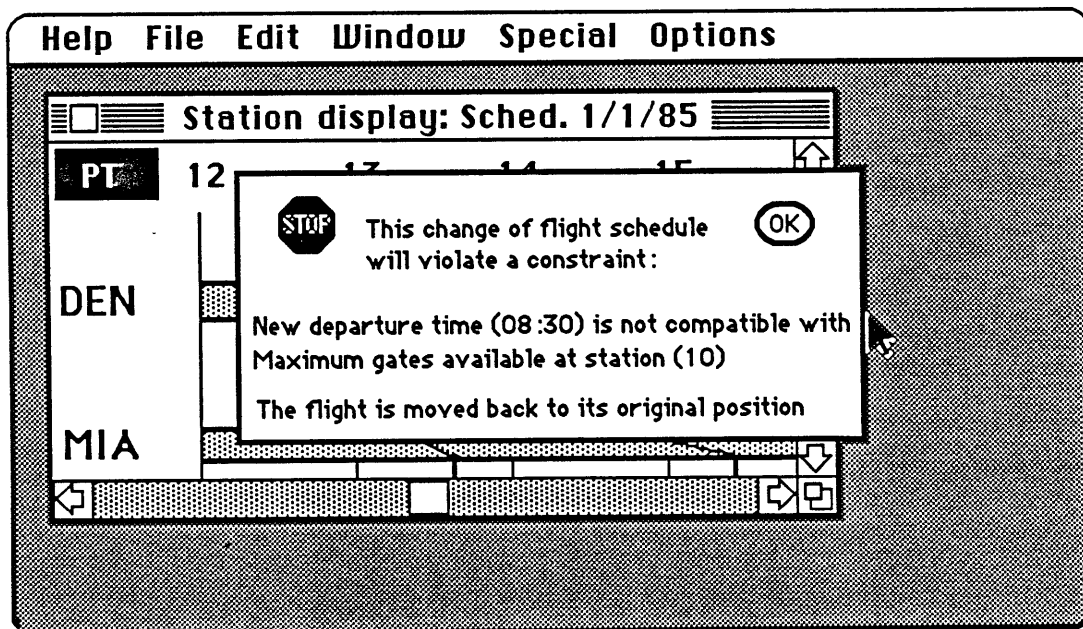


Figure 3.27: Dialog window in the station display

4 FUTURE DEVELOPMENTS AND APPLICATIONS.

4.1 Support algorithms.

Most of the thesis has described an insertion heuristic algorithm and has studied some of the extensions needed to complement the dial-a-ride version of it. Obviously, as more constraints from real life environments become known, the algorithm itself can be extended to incorporate new problem features. For example, discussions with O.S.A. have already indicated that it may be more appropriate to have "soft-bounds" on the earliest pick up or latest delivery times (rather than eliminate a request if a bound is violated -a hard bound-, increase the cost of the insertion according to a disutility function that increases as the constraint becomes more violated).

Unfortunately, many extensions will increase execution time beyond acceptable levels and, as we have already seen, it will be better at some point to simply let the human scheduler solve the problem. Given the complexity of a typical scheduling environment, it seems problem size will always be an issue, whether it is the number of calculations that will be required or the size of memory needed. For the time being, at least, all paths seem to lead us back to the human operator.

Most of the future research should therefore try to produce algorithms that can present to the user a better "initial guess" about the schedule. "Better" does not necessarily means "optimal" or "satisfying-as-many-constraints-as- possible" but able to accept further modifications without much deterioration. As we discussed in chapter 2, both the objective-function of our heuristic and its insertions selection process need to be studied further, but other optimization techniques should be investigated also. The Flight Transportation Laboratory at M.I.T. is currently testing

the Out-of-Kilter-Flow algorithm (O.K.F.) that was used in one of its vehicle routing systems [ref. 13, 14]. Given the characteristics of the algorithm, it could use a schedule produced with the insertion technique to generate a much better solution which could then be further re-arranged by the scheduler.

In the end, will we ever be able to create a full schedule automatically? In the current state of research, we may be able to answer positively. Our heuristics would be used to execute half the task, and a "rule-based" system could possibly do the second half (developed by Artificial Intelligence scientists, rule based systems typically consist of base facts known initially and a set of rules that allow the system to draw some conclusions through inferences made from the initial knowledge). In our case,

- the initial knowledge for the rule-based system would consist of
 - the schedule produced by the support heuristic.
 - the numerous side constraints that have not been (or could not be) taken into account by the algorithm.
- the rules would reflect the reasoning of the scheduler, how he (she) moves flights around in order to account for detailed scheduling issues, and
- the conclusions would simply be which flights should be moved and how.

Even though many journal articles have recently praised the rule-based system (also called expert system) technology [ref. 2, 5], the author's experience with such systems has been quite different: the field looks promising, but it must be approached with great caution. A further investigation of these systems for scheduling is recommended but, especially since the technology is new, it may lead to disappointing results.

4.2 Schedule drawing board.

This part of the system can definitely be improved substantially.

First, it is very unlikely that professional schedulers working with major airlines in a pressure intensive environment will accept the small screen of a Macintosh XL computer as well as its

memory and speed limitations. For this reason, a natural upgrade of this system is to implement it on a more powerful machine such as the Apollo or Sun graphics-oriented computers. They both have much larger screens that enable the display of more information simultaneously (with color capabilities). Execution is faster and memory is not limited since both computers run with virtual memory systems.

Second, as the experience of O.S.A. scheduling teams is accumulated, it will be possible to refine, update or modify the capabilities of the drawing board. Finally, more research should be done on the best ways to present scheduling information to the user. The mission and station displays are certainly not the only ways to interact with the scheduler. In fact, if we expect the human user to take over the preparation of the schedule at some point, the display of information will be a critical factor that will affect the comfort of the scheduler and the speed at which he will release his final schedule.

4.3 Other applications, Scheduling Activities of non-Air Transport operations.

In this analysis we must distinguish between the support heuristic and the drawing-board.

In its current version, the mission display can be used to visually organize any activity that involves sequential events. The station display is somewhat more specific and might not be useful under all circumstances. For example, if the sequential events consist of beginnings and ends of processing activities in a factory floor, there is no notion of "station"; rather, a display that could give a cross-section of activities undertaken at a given time of the day by all production/manufacturing stations would be more appropriate. Ultimately, a general purpose system could consist of a library of displays that could be tailored to a specific application by the user.

On the other hand, it is very unlikely that a support heuristic such as the insertion algorithm could ever be made general purpose and still produce satisfactory solutions. The principle of the

insertion itself is rather general but all the constraints added to it will definitely depend on the application. In any case, it is possible to review how various types of scheduling activities could use an insertion based support algorithm.

1) Applications that could use the current version of the heuristic.

Beside O.S.A, nearly all organizations that must satisfy a certain number of predefined requests for transportation could use a similar scheduling system.

- Dial-a-ride systems with advance reservation (the original purpose of the algorithm).
- Air-taxi, again with advance reservation.
- Corporate air transportation departments that use their own business airplanes.

Transportation of cargo by trucks could also be handled by the same program:

- Delivery of goods from warehouses to supermarkets.
- Distribution of products to retailers (in this case, a request would correspond to a shipment from a warehouse to a store, the earliest pick-up time would be the time at which the product becomes available in the warehouse and the latest delivery time would be set by the retailer himself).

2) General transportation services such as those provided by public systems.

Airlines, train or bus companies could also use this system but with some modifications due to basic differences in their type of activity:

- a) they do not try to satisfy "individual" requests that might change every day but rather a "general average" demand that is cyclical but evolves slowly.
- b) that demand will depend on the type of service provided (fare etc...).

In this case, requests should be turned into levels of demand spread along the day, time bounds should be soft and new techniques must be found to enable the demand to be served by more than two vehicles unless that demand corresponds to a small enough period of the day.

3) Activities generally requiring the scheduling of sequential events.

We already mentioned the operations of a factory floor, but it seems that the insertion heuristic

coupled with the graphic interface could be used in general for task scheduling.

Recently, for instance, some interest has been expressed on applying the system to the scheduling of Space Shuttle missions activities. Typically a crew of 8 members or less must perform as many experiments, exercises and other activities as possible during a limited time period (7 to 10 days); these activities have priorities and each crew member has its specialties, moreover some tasks require a certain number of astronauts for their execution [ref. 11]. Nowadays, this scheduling requires many man-months of work at N.A.S.A and an appropriately modified A.I.S.E. could dramatically reduce this effort.

In this environment, the requests would simply be tasks to be executed and the vehicles would be the crew members. However, the number of constraints to be handled is much greater than in our case and the constraints themselves are quite different from ours; this application would certainly require a complete reprogramming of the algorithm.

5 CONCLUSION.

5.1 On the insertion heuristic.

Because it is a heuristic, the insertion algorithm has been shown to be flexible enough to accommodate many modifications. Some of these modifications will affect the way requests are processed, others will only alter the internal data structure of the schedule. It can be expected, however, that as more of these extensions are incorporated in the algorithm, execution time will increase substantially. Considering that the schedule obtained is never the most optimal one, there may be a point where a professional scheduler will produce solutions that are as good as those proposed by the system, but that require much less computation time. Especially with the graphic techniques presented in the next section, it is important for system engineers to remember that a human scheduler should also be "used" to accomplish certain tasks .

The modifications discussed in this section do represent a minimum set required to produce a usable schedule; it may not be appropriate to expand the features of the algorithm beyond this set.

5.2 On the graphic interface.

At this time, it is too early to present a real life evaluation of the Schedule electronic drawing board since the first system will be implemented only during the summer of this year. However, reactions to the presentations made have been quite positive. In the airline industry where the approach to scheduling is shifting away from entirely automated systems [ref. 3, 4],

many domestic and foreign carriers have expressed considerable interest. At the user level, one of the most commonly heard comments has been that the ability to override the decision made by the system was a major strength.

In any case, it becomes more apparent that the man-machine interactive approach to scheduling may become the state of the art in the near future. The new graphic-based computer systems that are developed everywhere will certainly favor this trend and, indeed, make it possible.

5.3 On the future.

Even though some operation research results have been criticized earlier, it is important to acknowledge that without such results this scheduling tool would have never been possible. Operation research's major contribution has been the thorough investigation of each component of the scheduling process as well as its conceptualization. For this reason, not only should all new quantitative procedures which might emerge in the future be carefully reviewed as possible support algorithms in our interactive environment, but research should also be monitored since it might further improve our understanding of the scheduling process itself.

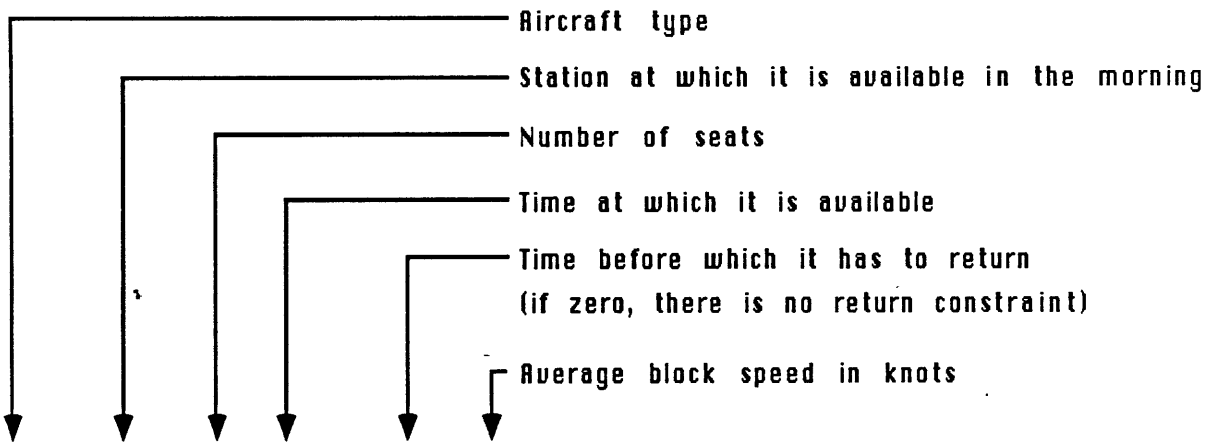
Because of their general purpose, the insertion algorithm and the electronic drawing-board have been shown to be adaptable to many different applications as long as they support each other within a unified set-up. We should not infer from this capability that a universal scheduling tool has been found, but the hybrid design of A.I.S.E is believed to be a first step in that direction.

6 APPENDIX.

6.1 Sample input and output from the insertion algorithm.

The following listings successively represent the three input data bases needed by the insertion algorithm followed by the output it produces. This data represents on a small scale the O.S.A. environment.

6.1.1 Aircraft data.



	Aircraft type	Station at which it is available in the morning	Number of seats	Time at which it is available	Time before which it has to return (if zero, there is no return constraint)	Average block speed in knots
C21	STL	5	01:00	24:00	420	
C21	DEN	5	05:00	21:00	420	
C21	SFO	5	06:00	22:00	420	
C21	LAX	5	05:00	24:00	420	
C21	DCA	5	06:00	23:00	420	
C21	MIA	5	06:00	23:00	420	
C12	BTV	5	04:00	23:00	420	
T39	LAX	6	05:00	23:00	420	
T39	STL	6	04:00	23:00	420	
T39	BED	6	07:00	23:00	420	
T39	STL	6	04:00	23:00	420	

6.1.2 Station data.

Station code	Latitude	Longitude
BTV (Burlington, VT)	44.28	73.09
BED (Bedford, MA)	42.28	71.17
JFK (New-York)	40.38	73.47
ATL (Atlanta)	33.39	84.26
MIA (Miami)	25.48	80.17
STL (Saint Louis)	38.45	90.22
DEN (Denver)	39.45	104.53
SFO (San Francisco)	37.37	122.23
LAX (Los Angeles)	33.56	118.24
DCA (Washington DC)	38.51	77.02

6.1.3 Request data.

Origin	Destination	Earliest pick-up time	Latest delivery time	Number of passengers	Split indicator (1=no, 2=yes)	Priority
BTV	BED	06:30	10:00	5	1	1
BED	JFK	10:00	15:50	2	1	1
DEN	MIA	10:00	21:00	7	2	1
SFO	STL	04:00	09:00	3	1	1
LAX	STL	07:00	12:00	7	2	1
SFO	JFK	12:30	23:00	5	1	1
STL	MIA	11:00	19:00	2	1	2
DCA	ATL	14:00	18:00	3	1	2
ATL	JFK	06:00	10:00	5	1	2
ATL	MIA	18:00	23:00	2	1	3
STL	ATL	13:00	16:00	3	1	3
DEN	MIA	06:00	12:00	2	1	3
STL	BED	04:00	08:30	4	1	3
DCA	ATL	06:30	09:50	3	1	3
MIA	DEN	07:00	20:00	2	1	3
ATL	DEN	01:00	19:00	5	1	3

6.1.4 Resulting mission assignment.

This final listing shows how the different requests can be handled by the fleet in a way that requires as little flying time as possible. A few comments are needed to understand this output:

- In a mission description, the column "MODE" contains the following codes:
 - RL: a ReLocation flight (aircraft is empty) is needed to go to the first station where a load is available.
 - Pn: Pick-up a load of priority n.
 - Dn: Deliver a load of priority n.
 - RT: The aircraft must ReTurn to its morning base.
- The column "SCHEDULED TIME" represents the earliest time at which an event could be executed. Since events have no duration in this version of the algorithm, landing, deliveries, pick-up's and take-off can all occur at the same instant at each station.
For RT, the time indicated is simply the latest return time to the station.
- The column "DESIRED TIME" contains alternatively:
 - the earliest time at which the load will be available for pick-up.
 - the latest time before which the load must be delivered.
- The mission listing is followed by lists of
 - the requests that were split in two parts and carried with two different aircraft.
 - the requests that were denied.

For example in the next listing, aircraft #1 will have the following mission:

- Take-off from Saint-Louis at 1 a.m. in order to relocate the empty airplane to Los-Angeles.
- Pick-up 5 priority 1 passengers at 7 a.m. and leave Los-Angeles for Saint-Louis immediately. Since the earliest pick-up time for this group of passengers was 7 a.m., the constraint is satisfied. The load on board of the airplane is now 5 passengers and there are no seats left.

- Arrive in Saint-Louis at 11:07 a.m. and deliver the 5 passengers. Since the latest delivery time for the group was noon, the constraint is satisfied again. The airplane had to return to Saint-Louis by midnight, it will therefore end its mission here.

***** AIRCRAFT # 1 *****
 TYPE = C21 / STATION = STL / TIME WHEN AVAILABLE = 01:00 / CAPACITY = 5 /
 SPEED (KNTS) = 420

ITINERARY : STL LAX STL

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
STL	RL	01:00				
LAX	P1	07:00	07:00	5	5	0
STL	D1	11:07	12:00	5	0	5
STL	RT	24:00				

***** AIRCRAFT # 2 *****
 TYPE = C21 / STATION = DEN / TIME WHEN AVAILABLE = 05:00 / CAPACITY = 5 /
 SPEED (KNTS) = 420

ITINERARY : DEN MIA DEN

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
DEN	P1	10:00	10:00	5	5	0
MIA	D1	14:22	21:00	5	0	5
MIA	P3	14:22	07:00	2	2	3
DEN	D3	18:44	20:00	2	0	5
DEN	RT	21:00				

***** AIRCRAFT # 3 *****
 TYPE = C21 / STATION = SFO / TIME WHEN AVAILABLE = 06:00 / CAPACITY = 5 /
 SPEED (KNTS) = 420

ITINERARY : SFO

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
SFO	RT	22:00				

***** AIRCRAFT # 4 *****
 TYPE = C21 / STATION = LAX / TIME WHEN AVAILABLE = 05:00 / CAPACITY = 5 /
 SPEED (KNTS) = 420

ITINERARY : LAX STL DEN MIA LAX

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
LAX	P1	07:00	07:00	2	2	3
STL	P2	11:07	11:00	2	4	1
STL	D1	11:07	12:00	2	2	3
DEN	P1	13:33	10:00	2	4	1
MIA	D2	17:55	19:00	2	2	3
MIA	D1	17:55	21:00	2	0	5
LAX	RT	24:00				

***** AIRCRAFT # 5 *****
 TYPE = C21 / STATION = DCA / TIME WHEN AVAILABLE = 06:00 / CAPACITY = 5 /
 SPEED (KNTS) = 420

ITINERARY : DCA ATL DCA ATL MIA DCA

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
DCA	P3	06:30	06:30	3	3	2
ATL	D3	08:27	09:50	3	0	5
DCA	P2	14:00	14:00	3	3	2
ATL	P3	18:00	18:00	2	5	0
ATL	D2	18:00	18:00	3	2	3
MIA	D3	20:04	23:00	2	0	5
DCA	RT	23:00				

***** AIRCRAFT # 6 *****
 TYPE = C21 / STATION = MIA / TIME WHEN AVAILABLE = 06:00 / CAPACITY = 5 /
 SPEED (KNTS) = 420

ITINERARY : MIA

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
MIA	RT	23:00				

***** AIRCRAFT # 7 *****
 TYPE = C12 / STATION = BTV / TIME WHEN AVAILABLE = 04:00 / CAPACITY = 5 /
 SPEED (KNTS) = 420

ITINERARY : BTV ATL JFK BTV

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
BTV	RL	04:00				
ATL	P2	06:49	06:00	5	5	0
JFK	D2	09:13	10:00	5	0	5
BTV	RT	23:00				

***** AIRCRAFT # 8 *****
 TYPE = T39 / STATION = LAX / TIME WHEN AVAILABLE = 05:00 / CAPACITY = 6 /
 SPEED (KNTS) = 420

ITINERARY : LAX DEN MIA LAX

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
LAX	RL	05:00				
DEN	P3	07:35	06:00	2	2	4
MIA	D3	11:57	12:00	2	0	6
LAX	RT	23:00				

***** AIRCRAFT # 9 *****
 TYPE = T39 / STATION = STL / TIME WHEN AVAILABLE = 04:00 / CAPACITY = 6 /
 SPEED (KNTS) = 420

ITINERARY : STL BED STL ATL STL

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
STL	P3	04:00	04:00	4	4	2
BED	D3	06:57	08:30	4	0	6
STL	P3	13:00	13:00	3	3	3
ATL	D3	14:50	16:00	3	0	6
STL	RT	23:00				

***** AIRCRAFT # 10 *****
 TYPE = T39 / STATION = BED / TIME WHEN AVAILABLE = 07:00 / CAPACITY = 6 /
 SPEED (KNTS) = 420

ITINERARY : BED BTV BED JFK BED

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
BED	RL	07:00				
BTV	P1	08:10	06:30	5	5	1
BED	D1	09:20	10:00	5	0	6
BED	P1	10:00	10:00	2	2	4
JFK	D1	11:12	15:50	2	0	6
BED	RT	23:00				

***** AIRCRAFT # 11 *****
 TYPE = T39 / STATION = STL / TIME WHEN AVAILABLE = 04:00 / CAPACITY = 6 /
 SPEED (KNTS) = 420

ITINERARY : STL ATL DEN SFO JFK STL

BASE	MODE	SCHEDULED TIME	DESIRED TIME	PAX ON/OFF	ONBOARD LOAD	REMAINING CAPACITY
STL	RL	04:00				
ATL	P3	05:50	01:00	5	5	1
DEN	D3	09:09	19:00	5	0	6
SFO	P1	12:30	12:30	5	5	1
JFK	D1	18:40	23:00	5	0	6
STL	RT	23:00				

THE FOLLOWING REQUEST(S) HAVE BEEN SPLIT :

ORIGIN	DESTINATION	EARLIEST	LATEST	PAX	PRIORITY	AIRCRAFT
LAX	STL	07:00	12:00	7	1	1 , 4
DEN	MIA	10:00	21:00	7	1	2 , 4

THE FOLLOWING REQUEST(S) CANNOT BE SERVED :

ORIGIN	DESTINATION	EARLIEST	LATEST	PAX	PRIORITY
SFO	STL	04:00	09:00	3	1

Calculation time in minutes: 0.054

6.2 Effect of different pool sizes in the algorithm.

CASE	Pool size	Pax-miles	Average LF	Travel time	Calculation time	Idle aircraft
Same as example	1	68432	40%	88.6	0.041	2
	2	67928	40%	91.1	0.043	3
	3	67928	37%	94.5	0.044	2
	4	67928	37%	94.5	0.044	2
	5	67928	37%	95.5	0.047	2
Twice as many requests	1	113939	47%	131.9	0.082	1
	2	109823	47%	121.6	0.087	1
	3	109823	47%	121.6	0.091	1
	4	109823	47%	121.6	0.094	1
	5	105833	45%	121.3	0.095	1
Four times as many requests	1	158032	49%	169.4	0.207	0
	2	145852	51%	155.9	0.213	0
	3	156975	52%	163.6	0.213	0
	4	156534	48%	166.4	0.233	0
	5	153118	45%	165.9	0.215	0
Same as above + twice as many aircraft	1	231252	51%	238.7	0.288	2
	2	230748	51%	241.2	0.294	3
	3	234738	51%	239.3	0.312	2
	4	230748	51%	241.2	0.321	3
	5	224392	47%	242.5	0.344	3

In this table headers must be read as follows:

- CASE: Content of the initial data-base
- Pool size: Number of slots in the algorithm pool (for final selection)
- Pax-miles: Number of passengers multiplied by the number of (nautical) miles they flew
- Average LF: Average system load-factor (in %)
- Travel time: Total travel time used to produce the schedule (in hours)
- Calculation time: Time needed to calculate the schedule (in minutes)
- Idle aircraft: Number of aircraft not used

BIBLIOGRAPHY.

1. Deckwitz T.A. : **"Interactive Dynamic Aircraft Scheduling"**. M.S. thesis and Flight Transportation Laboratory report R84-5 -MIT May 1984.
2. Duda O.R. and Gaschnig J.G. : **"Knowledge-Based Expert Systems Come of Age"**. Byte magazine, September 1981.
3. Etschmaier M.M. and Mathaisel D.F.X. : **"Aircraft Scheduling: The State of the Art"**. AGIFORS (Airline Group, International Federation of Operational Research Societies) XXIV, Strasbourg, France, September 1984.
4. Etschmaier M.M. and Mathaisel D.F.X. : **"Aircraft Scheduling: An Overview"**. Transportation Science, Vol 19 No. 2, May 1985.
5. Foley M.J. : **"Expert Systems Shells Boost A.I. Market"**. High Technology, Vol. 5, No. 3, March 1985.
6. Hung H.k., Chapman R.E., Hall W.G. and Neigt E. : **"A Heuristic Algorithm for Routing and Scheduling Dial-a-Ride Vehicles"**. Presentation at the ORSA/TIMS conference at San Diego, 1982.
7. Jaw J.J., Odoni A.R., Paaraftis H.N and Wilson N.H.M. : **"A Heuristic Algorithm for the Multi-Vehicle Advance-Request Dial-a-Ride Problem"**. Working paper MIT-UMTA-82-3, M.I.T. 1982 (Submitted to Management Science).

8. **Jaw J.J. : "Solving Large-Scale Dial-a-Ride Vehicle Routing and Scheduling Problems"**. Ph.D. dissertation and Flight Transportation Laboratory report R84-3, June 1984.
9. **Jaw J.J., de Lamotte H.J., Mathaisel D.F.X. and Simpson R.W. : "Automation of Dynamic Airlift Scheduling procedures"**. Presentation at the ORSA/TIMS conference, Boston April 30 1985.
10. **Krolak P., Felts W. and Nelson J. : "A Man-Machine Approach toward Solving the Travelling Salesman Problem"**. Comm. ACM, 14, 327-334, 1971.
11. **Kurtzman C. : "Expert Systems For Space Station Crew Activity Planning"**. Working paper, Space Systems Laboratory, M.I.T. May 1985.
12. **Lubow B. : "Aircraft Scheduling: An Interactive Graphics Approach"** M.S. Thesis, Department of Aeronautics & Astronautics, MIT January 1981.
13. **Simpson R.W. : "A Review of Scheduling and Routing Models for Airline Scheduling"**. AGIFORS VIII, 1968.
14. **Simpson R.W. : "Scheduling and Routing for Airline systems"**. Flight Transportation Laboratory report R68-3, MIT 1969.
15. **Simpson R.W. : "Analysis of Airlift Scheduling Function: Revised Workplan"**. Flight Transportation Laboratory, MIT, September 1983.
16. **Simpson R.W. and Mathaisel D.F.X. : "Automation of Airlift Scheduling for the Upgraded Command and Control System of Military Airlift Command"**. Flight Transportation Laboratory report R84-6 , MIT April 1984.
17. **Roy S, Chapleau L., Ferland J., Lapalme G. and Rousseau J.M. : "The Construction of Routes and Schedules for the Transportation of the Handicapped"**, Working paper, University of Montreal, 1983.