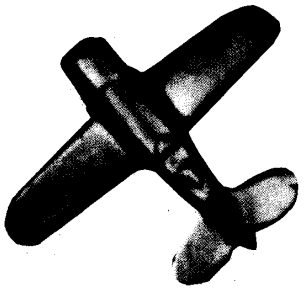


GATES

An Airline Gate Assignment and Tracking Expert System

Robert P. Brazile and Kathleen M. Swigger
North Texas State University

We will describe GATES, an expert system that assigns gates to arriving and departing flights at New York's John F. Kennedy International Airport (JFK). GATES uses flight information and knowledge about current constraints to produce possible gate assignment schedules. System operators can modify schedules by retracting rules, adjusting tolerances, and deleting information. We developed the system for a PC, thereby providing an efficient and flexible user environment. Our approach is extensible to various engineering and industrial problems where limited resources and weakly defined constraints exist and in which scheduling must occur.

Most airline companies generate new flight schedules every quarter, and sometimes as frequently as every month. They prepare these schedules using information about airport gate configuration to ensure that valid gate assignments can be made. The flight schedules are then transmitted to individual airports that use the information to produce daily flight schedules containing flight numbers, arrival and departure times, and aircraft identification numbers.

Once daily flight schedules are prepared, ground controllers assign specific flights to specific gates. At airports having few gates and many flights, gate assignment is a nontrivial problem. Gate assignment is equally difficult where many planes arrive and depart at the same time (the hub concept). The amount of activity occurring during these specific times overloads scarce resources—congested taxiways, for instance, or service personnel and equipment. Gate assignment then becomes a task of allocating flights to gates while not violating resource constraints.

GATES captures knowledge and procedures used by an experienced ground controller who solves gate allocation problems daily. Trans World Airlines (TWA) uses this system at JFK. In its current form, GATES assists ground controllers in preparing monthly and daily gate schedules.

0885-9000/88/0500-0033 $1.00©1988 IEEE

Related research

Bellman et al. have intensively investigated scheduling problems including job shop scheduling, project scheduling, and assembly line balancing.[1] Most often, researchers have used mathematical programming techniques to solve these problems. Mark Fox, who studied the scheduling problem more recently using constraint-directed reasoning, suggests that some scheduling problems can be solved using AI techniques.[2] His assertion has been tested in other areas, such as managing and automating process planning and scheduling functions.[3,4]

A separate but related research area concerns using AI techniques to solve planning problems. Build[5] and Strips[6] exemplify two early planning systems. Both generated plans enabling robots to perform certain tasks. Like scheduling, planning involves opportunistic action as opposed to always following a preprogrammed, fixed operational order.[7] As researchers have suggested,[8] planning and scheduling are closely linked: Planners generate a list containing several possible operational sequences. Selecting one sequence over another, and assigning operations to specific tasks, is commonly called *scheduling*. Together, they produce acceptable solutions to problems.

Similarly, GATES combines planning and scheduling to assign planes to specific gates. As in other scheduling tasks, identifying time, cost, and personnel constraints is extremely important. We have tried to identify constraint categories that could be useful for other scheduling problems.

Problem description

Gate assignment problems can be divided into two highly related problem areas. The first involves creating quarterly (or monthly) schedules. Once a month, operators receive lists of flights for their airport, from which they prepare daily gate assignments for the month. Time is not a critical element. The average ground controller requires 10 to 15 hours to prepare a gate schedule, depending on the time of year.

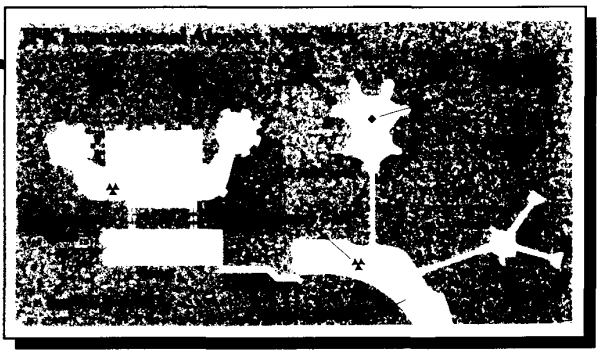
The second problem area occurs when schedules are interrupted by unanticipated events such as bad weather, mechanical failure, late arrivals, and so forth. In such situations, ground controllers have only minutes to rearrange existing gate assignments that will accommodate incoming planes. Time replaces optimality as the most critical constraint whenever daily schedules must be changed in real time. Experienced ground controllers become expert at adjusting gate assignments quickly (our GATES expert has over 14 years of scheduling experience).

One gate assignment strategy attempts to formulate gate scheduling as an optimization problem using an integer linear programming model. Some airports currently use this method to produce monthly schedules. Unfortunately, linear programming proved too slow for JFK. For example, for a total of 100 flights that must be assigned to 30 gates—if a 0-1 variable is assigned to each possibility—the number of problem states is 2 to the 3000th power. Most likely, no polynomial algorithm exists for solving the general constraint satisfaction problem.[9] Moreover, the random nature of constraints and ground controller preferences makes gate problem solutions nondeterministic. Prolog's descriptive, logic-based, and nondeterministic nature—plus its ability to backtrack—enables one to obtain a deductive database using facts and rules to represent gate-scheduling constraints. Consequently, we selected the Prolog language for GATES. We developed a small prototype using Prolog and, based on its success, designed and implemented a more extensive system.

The solution process

GATES is a constraint satisfaction expert system.[10] It has production rules guiding search, can tell users why it chose specific gate assignments, and produces an audit trail of rules it used in reaching conclusions. GATES does not use confidence factors. It does have a user interface enabling clients to change rules and data and to perform "what if" analyses.

The GATES problem solver consists of two logic levels: The first produces the standard daily schedule that is used as the "initial" plan; the second adjusts the daily gate schedule in real time as information about flight delays, weather changes, and equipment failures becomes available. As input, the standard gate scheduler uses flight schedules and known constraints. Initially, flights are not assigned to gates. At the completion of the process, all flights are assigned to gates and no conflicts exist. The second-level problem solver (that is, the reassignment process) uses the current gate schedule plus any new constraints or changes to the flight schedule as input. If changes to the flight schedule result in conflicts, then GATES adjusts the gate schedule to resolve them.



TWA operates out of Terminals A and B at New York's Kennedy Airport. An enclosed walkway connects the two terminals. All international flights arrive and depart from Terminal A.

First-level problem solution—the constraints

Five different constraint categories exist for gate assignment. We determined the relative importance of each, and list them below.

(1) Constraints without exceptions—Some gates have physical constraints prohibiting them from servicing certain aircraft types. For example, a gate may be too small to accommodate a 747. Obviously, such restrictions cannot be relaxed under any circumstances.

(2) Constraints with exceptions—These can be violated under some conditions. As a general rule, domestic flights arrive and depart at domestic gates and international flights arrive and depart at international gates. However, some service both international and domestic flights. In addition, the general rule can be relaxed for special cases such as allowing domestic flights to arrive at international gates if they "turn" to international departures. Finally, international flights receive special priority during gate assignment because their passengers must go through customs.

(3) Constraints with changing tolerances—The time required to park an incoming aircraft, service it, and ready it for departure varies for different aircraft types. To avoid conflicts between incoming and outgoing flights using the same gate, gate schedulers must consider the total time needed to perform these tasks as a constraint. However, there are "desired" times, "minimum" times, and "Well, if we have to, we'll make this assignment because everything will probably change anyway" times affecting gate assignment.

(4) Guidelines—A single taxiway feeds several gates. We should avoid multiple arrivals and departures for gates served by the same taxiway within the same time frame. Clearly, relaxing this guideline can create congestion problems.

(5) Convenience constraints—Most airplanes arrive and depart on the same day. Although the intervening time between arrival and departure may be several hours, every effort is made to keep a plane at its gate and to limit the number of times it must be moved from the gate and back again. Ground crews appreciate minimizing these moves; however, a plane can be moved if another plane needs that gate.

Although our final system will adapt to new knowledge and new constraints, the initial GATES version sought to implement sufficient constraints to produce an acceptable schedule. That is, its major requirement was to produce a schedule acceptable to our expert.

Production rules

To make its decisions, GATES uses two types of production rule, permissive rules and conflict rules. Permissive rules determine when it's appropriate to consider a particular gate for a particular flight, and permit the system to search the next level of rules to obtain an assignment. They also rule out gates that are unacceptable choices for a flight, thereby accelerating the selection process. An example of such a rule would be—"If the flight is not continuing, then it is acceptable to consider remote gates for this flight." Simply put, this means that if no departure time is listed for the flight, we can assume the plane will "layover" at the airport. GATES then assigns the plane to overnight parking. This exemplifies rules using convenience constraints to make assignments. A Prolog version of this rule might resemble the following:

```
gateok(G,F):-not(continue(F)),
   remote_gates(L),member(G,L).
```

The symbol ":- " can be read as "if" and the commas between atoms as "and." A literal English translation of the above rule would be—"Gate G is okay for flight F if F is not continuing, if L is the set of remote gates, and if G is a member of L."

Conflict rules determine when particular flights cannot be assigned to particular gates. An example of this type of rule would be—"Gate 2 cannot service 747-type aircraft." This exemplifies rules using constraints without exceptions; as such, it cannot be set to inactive by users whenever they do "what if" analyses. An example of this type of rule would be

```
conflict(2,F):-flight_plane(F,P),ptype(P,747).
```

An English translation of this rule would be—"Gate 2 has a conflict for flight F if P is the plane used by flight F and if P is a 747-type aircraft."

```
enter tolerance #
1:dpt - arv          =   8
2:arv - arv          =  45
3:dpt - dpt          = 110
4:arv - dpt          = 110
5:taxi way tolerance =   1
6:turn interval      = 600
7:continue interval  = 600
```

Figure 1. The change tolerance menu.

```
enter f,g,a,u,d,n,q, or h for help
?h
enter f to find a flight
enter g to find a gate
enter a to assign a flight to a gate
enter u to unassign a flight from a gate
enter d to say don't assign a flight to a gate
enter n to list flights not assigned
enter q to quit
enter f,g,a,u,d,n,q, or h for help
?
```

Figure 2. The assignment menu.

Tolerances

Tolerance levels are time intervals that the schedule maintains throughout deduction. A special conflict-type-rule subcategory uses tolerance levels to determine appropriate gate assignments. For example, one tolerance level is the time interval between departing flights and arriving flights using the same gate. Airlines estimate that it takes 15 minutes to perform this interchange—that is, 15 minutes is generally enough time for a departing flight to clear the gate before an arriving flight approaches the gate. Tolerances used in conflict production rules implement constraints with changing tolerances. The following exemplifies a conflict rule using a tolerance level:

```
conflict(G,F):-
        assign(G,F1),
        depart_time(F1,T1),
        arrive_time(F,T),
        T1<T,
        dptarv(I),
        timeadd(T1,I,X),
        X>T.
```

This rule states that a conflict exists on gate G for flight F if some other flight F1, already assigned to gate G, has a departure time of T1 that is within the departure/arrival time interval tolerance of flight F's arrival.

GATES sets (or defaults to) initial tolerance levels during its first attempt to produce a gate schedule. If this first attempt is unsuccessful, or if an alternative assignment is desired, then GATES will automatically adjust tolerance levels. Tolerance levels can be set by GATES or preset by individuals using the system. Users can set tolerance levels dynamically, while the system is running, through the user interface (see Figure 1). Users can also make other changes—they can assign flights to gates, unassign flights from gates, or specify that flights cannot be assigned to specific gates (see Figure 2). Once tolerances are adjusted, users can either restart the entire gate assignment process or resume assignment from where the system was interrupted.

Priority operation

GATES prioritizes flights and gates by first assigning those flights and gates that have the most constraints. For example, only a few gates are equipped to service wide-bodied aircraft. Therefore, wide-bodied planes are processed first. Nearly all gates can accommodate 727-type aircraft, whereas only a few gates can accommodate 747-type aircraft. Therefore, 747s are assigned before 727s. When considering the next gate assignment, GATES seeks unassigned flights that have special characteristics limiting their assignment.

Gate assignment is also prioritized according to gate characteristics and location. For example, some gates are close together and require special attention. In most cases, aircraft using these gates cannot safely taxi to the gate but must be towed. Since towing involves additional time and resources, GATES gives these types of gate assignment lower priority.

Gate assignment

Gate assignments are made when no permissive rule, conflict rule, or tolerance level is violated. If all flights have been assigned to gates after one pass, the process terminates. If some flights cannot be assigned, then GATES deactivates optional rules or reduces noncritical tolerance levels. GATES may unassign some flights and try again to assign all flights. Thus far at JFK, GATES has successfully assigned all flights to gates.

GATES also maintains (1) a list of permissive rules allowing assignments to be made, (2) conflict rules that prevent assignments from being made, and (3) tolerance levels in effect at any given moment. At the end of deduction, operators can ask why a specific assignment was made or not made. Figure 3 shows a partial listing of why some flights were not assigned to gate 2. Figure 4 shows why departing flight 805 was not assigned a gate. Whenever a gate schedule is displayed, active rules and tolerance levels are also displayed.

Operators can also assign or reassign flights to gates as well as change tolerance levels and active rules. A simple window/menu interface enables operators to adjust any value at any time during scheduling. Operators can try various "what if" scenarios and note the different schedules generated under different conditions.

```
Gate 2 not assigned 803d because
plane can't use gate
F=803, AD=d
Gate 2 not assigned 34a because
scheduled departure too close
F=34, AD=a
Gate 2 not assigned 11a because
scheduled arrival too close
F=11, AD=a
Gate 2 not assigned 814a because
continuation flight interval violated
F=814, AD=a
```

Figure 3. Why gate 2 was not assigned some flights.

```
Gate 26 not assigned 805d because
scheduled departure too close
G=26
Gate 27 not assigned 805d because
747 blocking gate
G=27
Gate 31 not assigned 805d because
plane can't use gate
G=31
```

Figure 4. Why flight 805d was not assigned some gates.

The GATES data representation

GATES organizes its data under two main predicates. One stores the flight schedule. The second produces actual gate assignments. We used three criteria when designing these data structures. First, they had to store and retrieve information efficiently. Second, they had to be flexible and easy to modify through a user interface. Third, they had to be in a format that could be generated from the airline's existing database. The final GATES version extracts current flight schedules and aircraft information stored on the airline's large database. It uses this information, along with updated flight information, to make ongoing assessments and changes. Based on these requirements and features available in Turbo-Prolog, we designed a pair of database predicates to store assigned and unassigned flights. A database predicate in Turbo-Prolog can be "asserted" into the database.

GATES builds the first predicate—"unsch"—from the initial flight schedule data. This predicate is defined as

```
unsch(plane,flight1,arrival.time,
  flight2,departure.time)
```

where plane is the identifier for the actual aircraft or aircraft type assigned to these flights, flight1 is the arriving flight number of a continuing aircraft, and flight2 is the flight number of the departing flight for this aircraft. Arrival time and departure time are the scheduled arrival and departure times for these flights. If an aircraft is scheduled to depart but not to arrive, or to arrive but not to depart (that is, to layover), then the flight number for the nonexistent portion is set to zero.

The data structure includes variables for flight1 and flight2 of the same aircraft because most aircraft continue to other cities (and are called continuation flights). Therefore, it is very important to assign both segments of continuation flights to the same gate. One data structure includes this information to avoid extensive search later in the program. Since only a few flights do not have a continuation portion, GATES handles these rare cases as exceptions.

The second database predicate—"adsch"—represents the arrival and departure schedules, plus their gate assignments, and is defined as follows:

```
adsch(gate1,gate2,plane,flight1,arrival.time,
  flight2,departure.time)
```

where gate1 is assigned to the arriving flight, and gate2 is assigned to the departing flight. Other parameters listed in this data structure resemble those

described for the "unsch" predicate. Again, either variable for the flight numbers can be set to zero.

GATES looks through "unsch" database predicates to determine which unscheduled flight to assign. If it can assign a flight, it retracts the "unsch" entry for that flight and asserts the flight under an "adsch" entry along with its gate assignment. If it can assign a gate to only one of the flight pair, it retracts the "unsch" entry, asserts an "adsch" entry for the successfully assigned flight, and asserts another "unsch" entry for the unassigned flight. Whenever an arrival time changes, GATES can either retract the "adsch" entry for that flight and assert an entry in "unsch" with the new time, or it can rebuild the entire "unsch" data structure. Since assignment takes about 30 seconds to run, several alternative solutions may be explored before selecting the final solution.

Reassignment—the second level

GATES receives information from the airline database regarding the current status of arriving and departing flights. When a flight deviates from its schedule by some threshold value, GATES knows that the current gate assignment may require modification. If modification must occur, GATES checks the assignment to see if the change creates any new conflicts. If not, the schedule remains unchanged. GATES reassigns the questionable flight if conflict exists, using default rules and tolerances to reassign the deviant flight. For example, GATES may deduce that another gate is available at the same time and that all other assignments will be unaffected by this change. If this modification is successful, GATES makes the gate assignment and asks the user to confirm it. If GATES cannot resolve the conflict, it must take further action. It can either retract all assignments and reassign the entire schedule, or it can resolve the conflict by changing assignments minimally.

If the user decides to reschedule using minimum changes, GATES identifies the specific conflicting gate assignments where changing only one secondary (previously unchanged) flight will resolve the conflict. If such a case exists, GATES tries to reassign the secondary flight to another gate. This process continues until no conflicts exist or until all cases of this type have been examined and no single flight can be reassigned. It may then be necessary to examine combinations of two, three, or more gate changes to resolve a single conflict. The rescheduling activity occurring during this phase depends on the amount of time the operator specifies. If GATES cannot resolve the conflict within the specified time, it asks the operator to determine the new gate assignment.

The GATES reassignment process is being implemented at JFK. Tests thus far indicate the system can assign flights in five to 10 seconds that were not assigned previously due to conflicts or undesired assignments.

Prolog has proven effective and flexible in describing constraints and rules controlling schedule generation. GATES evolved from about 100 lines of Prolog code in its initial prototype to over 1000 lines of code with about 100 different Prolog predicates in the current system. TWA, handling about 100 to 115 flights per day at its JFK terminals, uses GATES to assist ground controllers. GATES accesses TWA's database for the current day's schedule and creates gate assignments in about 30 seconds. Previously, that scheduling task took an operator between 10 and 15 hours to prepare manually—and then as much as an hour to modify each morning.

As the program has grown, we have added several software engineering tools to better organize the rules and the inferencing process. Two of the more helpful tools are a data dictionary for the Prolog predicates and a special "list" function that displays predicates in the order they are used.

Thus far, the system has been well received. It presently assists ground controllers in preparing TWA's quarterly gate schedules for JFK airport. The daily scheduler will soon be fully operational and ready for test and evaluation. Future work on this project will expand the user interface to handle additional graphics. We also want to add a feature enabling us to capture daily changes the operator makes to computer-generated schedules. If these changes can be analyzed and incorporated into GATES, then—like its human counterparts—GATES can learn from experience.

Acknowledgments

Our work was supported in part by a grant from Trans World Airlines, Kansas City, Missouri. This article's preparation was supported in part by the Office of Scientific Research.

GATES in action: A TWA 747 disembarks and boards passengers during a JFK transit.

References

1. R. Bellman et al., *Mathematical Aspects of Scheduling and Applications,* Pergamon Press, Elmsford, N.Y., 1982.

2. M.S. Fox, *Constraint-Directed Search: A Case Study of Job Shop Scheduling,* PhD dissertation, Carnegie Mellon University, Pittsburgh, Pa. 15213, Dec. 1983.

3. B.R. Fox and K.G. Kempf, "Complexity, Uncertainty, and Opportunistic Scheduling," *Proc. IEEE Conf. Artificial Intelligence Applications,* Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, Calif. 90720, 1985, pp. 487-492.

4. B.R. Fox and K.G. Kempf, "A Representation for Opportunistic Scheduling," *Proc. Third Int'l Symp. Robotic Research,* MIT Press, Cambridge, Mass., 1986.

5. S.E. Fahlman, "A Planning System for Robot Construction Tasks," *Artificial Intelligence,* Vol. 5, No. 1, 1972, pp. 251-288.

6. R.E. Fikes and N.J. Nilsson, "Strips: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence,* Vol. 2, Nos. 3/4, 1971, pp. 189-208.

7. E.D. Sacerdoti, *A Structure for Plans and Behavior,* Elsevier North-Holland, New York, N.Y., 1977.

8. R. Wilensky, *Planning and Understanding,* Addison-Wesley, Reading, Mass., 1983.

9. A.K. Mackworth and E.C. Freuder, "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems," *Artificial Intelligence,* Vol. 25, No. 1, 1985, pp. 65-73.

10. B.G. Buchanan and E.H. Shortliffe, eds., *Rule-Based Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project,* Addison-Wesley, Reading, Mass., 1984.

Robert P. Brazile is an assistant professor of computer science at North Texas State University. His research interests include the effective storage, retrieval, and use of data and knowledge. He has designed and implemented operations research systems and very large database systems. Before joining North Texas State, he worked for GTE, Honeywell Information Systems, and the Mitre Corporation.

Kathleen M. Swigger is an associate professor of computer science at North Texas State University, where she has taught since 1980. She spent the 1986-87 academic year developing intelligent tutoring systems for Air Force training applications at the Brooks AFB Human Research Lab in San Antonio, Texas. Her research interests include AI and human factors in computer systems. Specifically, she is interested in developing generic planning tools that can be used to create expert systems for intelligent instructional planners and for scheduling problems.

The authors can be reached at Box 13886, NT Station, Denton, TX 76203.