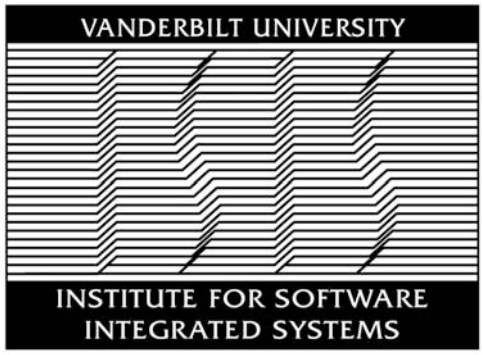

Institute for Software-Integrated Systems

Technical Report

TR#: **ISIS-02-303**

Title: **Computer-Aided Aircraft Maintenance Scheduling**

Authors: **Chris vanBuskirk, Benoit Dawant, Gabor Karsai, Jonathan Sprinkle, Gabor Szokoli, Karlkim Suwanmongkol and Russ Currer (IDEA Services)**

Abstract
This report presents the architecture and algorithms used in a system designed to assist maintenance chiefs in planning both aircraft to mission allocations and scheduled maintenance activities on aircraft. The system operates in two stages: the first stage assigns qualified planes to forecasted flight operations. This permits the projection of expected flight hours and equipment usage across the fleet. The second stage schedules, down to a quarter-hour resolution, both usage-based and calendar-based maintenance actions. The algorithm used to implement the first stage is a custom-built, multi-level, greedy search algorithm. The second stage has been implemented as a constraint satisfaction problem that is solved with a search engine built using the Oz language of the Mozart programming system[1]. Results obtained with squadrons of military attack jets (~17 jets/squadron) are presented.

1. INTRODUCTION

Scheduling maintenance tasks for a fleet of aircraft requires significant expertise and necessitates constant reassessment of short and long-term plans, re-prioritization, or task re-allocation in response to changes in fleet status, change in personnel and resource availability, or modifications to flight plans. Tasks the maintenance chief needs to schedule can be categorized into two broad categories: scheduled and unscheduled. Scheduled maintenance tasks can further be subcategorized as usage-based maintenance and calendar-based maintenance. Usage-based maintenance is performed after aircraft have flown a fixed number of hours while calendar-based inspections need to be performed at regular time intervals regardless whether the aircraft has flown. Unscheduled maintenance/repair must be performed in response to unexpected events/ breakdown. Though long-term plans for scheduled maintenance are beneficial, in practice, their scope and detail are limited by the time required to create them combined with the fact that they so frequently have to be modified or redone in response to unforeseen events.

MAPLANT (Maintenance Planning Tool) is a system currently under development that is designed to rapidly generate detailed schedules for aircraft maintenance over a 30+ days period. The system generates *maintenance schedules,* which define what specific maintenance actions need to be taken on which aircraft of the fleet. At present, the maintenance actions scheduled are (1) calendar-based *Daily Special Inspections* (DSIs), and (2) *Usage-based Special Inspections* (USIs) such as basic hourly inspections, phased-maintenance inspections, and engine pulls, all of which are defined in the maintenance manuals of the aircraft.

Currently, the input to the system consists of:

- Maintenance Plan: A requirement specification, which states what DSIs must be done on which aircraft in the squadron, and what are their deadlines.
- Aircraft Roster: The list of aircraft with their type, airframe identification number and usage hours, engine number and usage hours, status (“up” or “down”), etc.
- Personnel Roster: The list of maintainers, their qualification codes, and their availability over time.
- Inspection Manual Data: The specification of all the maintenance tasks in the system. It includes: the specification of the DSIs and USIs: what tasks they consist of, their duration, their resource requirements in terms of maintainer skills and special tools, and any temporal sequencing.
- Flight schedule: A condensed version of the operational flight schedule. It consists of the mission requirements: what type of aircraft is needed, how many, when, and how long, pit-turn information (what actions are needed when flying missions ‘back-to-back’), and event priority.

[1] http://www.mozart-oz.org/

- Commander's Guidance: Work shift specification (how many and how long), holidays, flight-hour distribution preferences for particular aircraft, strategies or *rules-of-thumb* that tend to increase efficiency, resource capacities to attempt to reserve for emergencies, etc.
- Support Equipment Data: A description of the support equipment available for use, in terms of types and quantity.

From this input, the scheduler generates:

- A detailed Maintenance Schedule: A precise schedule of specific maintenance actions scheduled to specific points in time. By default, actions are scheduled with a resolution of 15minutes.
- Aircraft Availability Report: A set of numbers projecting the amount of available aircraft, by type/capability, over time during the scheduling period. Availability is also calculated with a resolution of 15minutes.

2. METHOD

At the time of writing, the scheduler operates in two stages. First, aircraft are assigned to missions without attempting to generate a detailed schedule. This first stage permits the computation of flight hours/aircraft over time. When this is done, calendar-based and usage-based inspections can be scheduled.

2.1 Aircraft-to-mission assignment

The method for assigning aircraft to missions is based on a specialized "greedy" algorithm. The difference between this implementation and a pure greedy one is that there are several layers of parameters that control the decisions in a heuristic, rather than just one number and one state of calculation of assignment.

The parameters of the algorithm are based entirely on *guidance* given to the system. To simplify the solution space, certain blocks of time are marked where the aircraft is most definitely not flying due to calendar based maintenance. The guidance given to the system specifies which days are available for flight, and any preference for flight on those days. It also gives time region guidance, such as a cumulative number of hours preferred by a certain date. This could be used, for example, to fly a piece of equipment into some usage-based maintenance by a certain date, or conversely to *milk* a jet in order to postpone the deadline of some costly usage-based inspection.

The following parameters make up the guidance data given for each plane

- "Daily" parameters
    - Number of preferred flying hours
- "Cumulative" parameters: monthly goals (normalized to the length of the planning period) and optionally hours to be achieved by some intermediate, target date
    - Number of Minimum, Preferred, and Maximum flying hours

Using this guidance, the heuristic acts in the following way to satisfy as many aircraft as possible in the assignment. The two main strategies for achieving this are

a) *maintaining* a database of actual flight hour assignments for each aircraft, and checking this database for compliance with the parameters at certain intervals in the algorithm, and, after each assignment,
b) *checking* which plane is in the most in need of an assignment to satisfy the current state of the parameter (min, preferred, max).

By default, these checks occur before each assignment, but the algorithm may be configured (at compile time) so that the checks occur at less computationally demanding intervals. The algorithm proceeds in three stages:

1. **Each aircraft must fly its minimum number of cumulative hours.** Inability to satisfy even one plane's minimum requirement is an error, as the guidance given is indicative of required downtime of the aircraft, which is rigidly enforced when hourly-based maintenance is

required. Once an aircraft has been assigned its minimum number of hours, it is not assigned to any flights at this stage.

2. **Each aircraft wishes to fly exactly its preferred number of cumulative hours.** Next, the system attempts to fly each aircraft the number of hours that it requests. Once a jet has been assigned its preferred number of hours, the aircraft does not actively seek out more work for itself.
3. **No aircraft may fly in excess of its maximum cumulative hour limit.** No assignment is made if making that assignment means that an aircraft will exceed its maximum number of hours. This check is made before each assignment, regardless of the state of the heuristic. This hard limit means that the system may fail to assign aircraft to some of the proposed sorties. This is not considered a fatal error, however.

Details of the algorithm

The algorithm stages mirror the parameter types used to express the heuristics. At each stage, there is a driving factor that determines which aircraft, or which flight is to be considered next.

Stage 1: Satisfy *minimum* requirements, day driven
- **Find** the busiest day (the day with the most flight hours)
  - **Find** the aircraft in most need of an assignment to meet its minimum goal
    - **If** assigning this aircraft on this day is not possible (because, for example, the aircraft is already assigned during this general timeframe)
    - **then** choose the next neediest aircraft,
    - **else** assign this aircraft to this flight.
  - **If** no aircraft can be found for this time **then** choose the next busiest day.
- **When** all aircraft have met their minimum goals, **proceed** to Stage 2.
- **If** all aircraft do not meet the minimum goals, **then** the algorithm fails.

Stage 2: Satisfy *preferred* requirements, day driven
- **While** there are still unassigned flights **do**
  - **Find** the busiest day (the day with the most flight hours)
    - **Find** the aircraft in most need of an assignment to meet its preferred goal
      - **If** assigning this aircraft on this day is not possible (because the aircraft is already assigned at this time, or it would exceed its maximum hours)
      - **then** choose the next neediest aircraft,
      - **else** assign this aircraft to this flight.
    - **If** no aircraft can be found for this time **then** choose the next busiest day.
  - **When** all aircraft have met their preferred goals, or all of the flights have been assigned,
    - **proceed** to Stage 3.

Stage 3: *Ensure all* flights have aircraft assigned to them, flight driven
- **While** there are still unassigned flights **do**
  - **Choose** any flight without an aircraft assigned to it
    - **Find** the aircraft with the most hours to spare before it exceeds its maximum
    - **If** assigning this aircraft on this day is not possible (because the aircraft is already assigned at this time, or it would exceed its maximum hours)
    - **then** choose the next most appropriate aircraft,
    - **else** assign this aircraft to this flight.
  - **If** no aircraft can be found for this time **then** choose the next flight
  - **When** all of the flights have been assigned **then** the algorithm is finished.
  - **If** not all flights are assigned **then** the algorithm fails.

In the event of failure, the algorithm returns such answer as it could find, so that the user can use the incorrect result to target changes in guidance or to modify the required flights.

2.2 Maintenance Scheduling

Once aircraft have been assigned to missions, their expected number of flying hours can be computed and both calendar-based and usage-based maintenance can be scheduled. Calendar- and usage-based inspections must be performed within a time segment called the due window, where this window is larger than the time required to execute the inspection. The task of the scheduler is to schedule all maintenance actions within these time windows taking into account availability of resources, both in terms of manpower and tools as well as operational obligations (i.e., aircraft cannot be maintained while in the air). The scheduler also attempts to generate a schedule that conforms to a certain number of business rules. These include, for example, guidance indicating that, when possible, it is beneficial for particular types of maintenance activities to be performed at the same time on a single aircraft or that overlapping, complex maintenance tasks on several aircraft should be avoided. Guidance also specifies the length of the various shifts, holidays, weekends, and safety margins of resources to hold in reserve for unexpected events.

To solve this problem, we have chosen a constraint-based approach. According to the taxonomy put forth by Derrick and Pegman [2], constraint-based scheduling problems can be categorized as follows: pure scheduling problems, pure resource allocation problems, or mixed problems. Pure scheduling is concerned only with the positioning of activities in time. Pure resource allocation problems do not require deciding when the activities take place. One needs only guarantee that enough resources are available to perform the tasks. Mixed problems involve scheduling and allocating resources at the same time. Most real-world problems fall in the latter category which is also the one for which ready solutions do not exist. This is especially true for very large-scale problems such as the one we are dealing with here (we need to schedule in excess of 3000 tasks). Practical solutions involve the use of heuristics design to take advantage of the application's idiosyncrasies. Here, we have used the following approach.

The time window over which a schedule needs to be computed is divided into available time slots. Information used to define time slots available for maintenance tasks include: the planning window (e.g. 6 weeks), information about holidays, weekends, and shift durations. Resources available to perform the various maintenance tasks are also entered as parameters. These include available mechanics and their qualifications, available tools, and available facilities. Note that the architecture supports time-dependent resource capacity. This permits taking into account real variations in resource availability due to, for instance, a short-term focus on training for a particular specialty or perhaps to an imbalanced distribution of the organization's workforce across shifts.

Properties of each task used by the scheduler include:

1) Resource requirements: the number of mechanics required to perform the task and their required qualifications, as well as tools and facilities
2) Due window: a time interval during which the task needs to be performed (the due window contains two parameters: the Ready Time (the time at which the task can start) and the Deadline (the time by which the task must be completed).
3) Duration: the typical time required to execute the task
4) Not-on: this property is used to instruct the system that a task should not be performed during certain time intervals (this is used, for instance, to encode the fact that maintenance tasks cannot be performed on aircraft while they are flying).
5) Task precedence: this property captures information about required task sequencing (i.e., task A needs to be completed before task B can start)
6) Task conflict: this property states that task A cannot be performed while task B is running (due to, for example, the side-effects of a constrained work area such as a crawl space that is only big enough for a single maintainer)

7) Task overlap: this property specifies that the larger duration task must encompass the smaller task.
8) Coincidence: task A must start or end at the same time as task B.

The problem is then transformed into a finite domain constraint problem, and a solver implemented in the Oz programming language [1] is used to find a solution. The goal is to find the starting time for each and every task while satisfying constraints imposed by the task properties and resource availability. To do so, task properties are first transformed into finite-domain constraints as described below. The constraints are inserted into a *constraint store,* and then the problem is solved using a general-purpose search engine. Our description below summarizes the constraint expressions. The "Starts" structure contains the finite integer domain variables representing start-times of individual tasks.

1) Due window: The task start time should be at or after the task ready time, and it should be before the deadline minus the task duration. Formally,

T.readyTime =<: Starts.T =<: T.deadline - T.duration

2) Not-on: For each "not-on" time interval specified for this task, impose that the task cannot start between the beginning of the time interval minus the length of the task and the end of the time interval. Formally,

```
For each interval I in list T.notOn:
   For each number N between (I.beginning - T.duration) and
                             (I.beginning + I.length)
      Starts.T \=: N
```

3) Precedence: For each task in the precedence list, constrain its ending time to be before the starting time of the current task. Formally,

```
For each task P in list T.precedes:
    Starts.P + P.duration =<: Starts.T
```

4) Conflict: For each task in the conflict list, constrain its ending time to be earlier than the starting time of the current task or its starting time to be after the ending time of the current task.

```
For each task C in list T.conflicts:
    Starts.C + C.duration =<: Starts.T
  or
    Starts.T + T.duration =<: Starts.C
```

5) Overlap: for each task in the overlap list, constrain its starting time to be larger than the starting time of the current task and its ending time to be smaller than the ending time of the current task

```
For each task R in list T.runsWhile:
    Starts.R >=: Starts.T
    Starts.R + R.duration =<: Starts.T + T.duration
```

6) Coincidence: for each task in the coincidence list constrain its starting time to be equal to the starting time of the current task:

```
For each task C in list T.coincides:
    Starts.C = Starts.T
```

To enforce the resource capacity constraints, we use a tool from the Oz built-in library: the "**Schedule.cumulative**" constraint propagator that enforces the resource limitations. Time variant resource availability is modeled by applying a constraint, which reduces the available capacity of the resource at specific times by a fixed quantity through artificial tasks we call "resource blanker"-s.

These constraints are only inserted into the constraint store after individually checking for contradiction with the existing store. Contradictory constraints are lifted, and reported to the user with a brief explanation on how to resolve the contradiction to achieve a perfect solution where all constraints are met. An imperfect solution is calculated to help the user verify the necessity of the waived constraint. This pseudo "soft-constraint"-based, advisory behavior was found most valuable to our users.

Once all constraints are posted to the constraint store and all constraint propagations relax to a stable state, the search engine is initiated. The engine chooses a basic constraint C, explores the search tree by adding C and **not**(C) to the constraint store, and propagates all effects. This process is called, in Oz-parlance, "distributing over C", and the choice algorithm is called the *distributor*. Note that Oz separates the concepts of *distribution* and *search control*: distribution determines the shape of the search tree, while the search strategy prescribes the order in which it is traversed. Obviously, the distributors, through the choice of the constraint C, can influence the performance of the search. An example of a distribution strategy is the standard *FirstFail* heuristic. This distributor explores that branch first, which is most likely to fail. In general, this means choosing the most constrained variable X. The measure of a variable's degree of constraint is inversely proportional to the number of options left open to that variable (i.e. the size of its feasible domain). In the event of a tie, the number of constraints a variable participates in may be used to arbitrate. The distributing constraint C is in the form of X =: V where V is an element from the domain of X.

A standard depth-first search algorithm is used with recomputation. Oz uses a copying approach to search, as opposed to trailing in Prolog and its derivatives. Constraint propagation always happens in a *computational space,* which represents a particular state of the search. The spaces do not support backtracking, but computational spaces can be copied, even embedded in each other. Recomputation simply means that the number of copies saved during search is decreased, only certain spaces have to be saved, and intermediate states can be reconstructed from the nearest copy using a journal of search choices made since.

3. RESULTS

These tools have been applied to the scheduled maintenance planning problem in AV-8B Harrier II attack jet squadrons. Currently, MAPLANT focuses on providing decision support at the squadron level where an organizational unit normally consists of approximately 17 jets, 20+ pilots and a couple of hundred maintainers. Table 1 lists typical experimental results of the maintenance scheduling algorithms described in 2.2 for a few representative datasets[4].

Unit Type	Planning Horizon	# A/C	# Maintrs	# Sorties	# Maint Actions	Time	Man Yrs Scheduled	Dataset
squadron	3 mos	17	201	1236	3750	19 m 58 s	6.439	2001-11-06T08:31:00
squadron	3 mos	17	201	1236	3758	19 m 42 s	6.446	2001-11-06T08:31:00
squadron	6 weeks	16	202	677	2130	4 m 5 s	4.113	2002-02-06T06:31:00
squadron	6 weeks	17	202	678	2089	4 m 16 s	3.855	2002-02-08T14:31:00
squadron	4 weeks	17	202	272	1155	0 m 58 s	1.909	2001-11-29T05:31:00
squadron	4 weeks	17	202	411	1158	1 m 24 s	2.391	2001-12-07T08:31:00
detachment	6 weeks	6	66	448	**0**	1 m 12 s	**0**	2002-05-21T16:15:00
detachment	6 weeks	6	**67**	448	1085	1 m	2.033	2002-05-

[4] All performance figures were measured on a 500Mhz Celeron-based laptop.

						1 s		21T16:15:00
detachment	6 weeks	6	66	443	1049	0 m 56 s	2.021	2002-05-22T08:13:00

Note that the first set of numbers based on the 2002-05-21 dataset demonstrates a situation where MAPLANT identified that it was impossible to perform the maintenance mission with the available resources. The system did, however, pinpoint what the bottleneck resource was and the period in time that was causing the problem. The results presented in the following row are the consequence of asking MAPLANT *what-if* we had one more maintainer of this particular type on the day shift.

4. DISCUSSION AND CONCLUSIONS

Though scheduled maintenance is only a portion of the work required to maintain aircraft, long-term planning is an important, and often neglected, activity for efficient operations. The tools described here facilitate the management of preventative maintenance activities, especially usage-based inspections since their due dates can be strategically manipulated through aircraft-to-mission allocations. MAPLANT provides views into the upcoming workload with a lead-time and a level of detail that is otherwise infeasible due to manpower constraints. In general, this leads to less erratic and more manageable situations for the maintenance department. Additionally, these tools greatly benefit flight operations as they allow rational, detailed decisions to be made about how best to satisfy operational goals (e.g. equitably increase the war fighting skills of all pilots using the standard curriculum of training defined by the Navy), given the goals of the maintenance department and the current health of the fleet. Given a proposed plan of flight operations, MAPLANT can quickly identify (a) how maintenance operations should be directed to accommodate ops and (b) which portions of the proposed flight schedule conflict with maintenance goals and/or represent unacceptably high risk of failure and (c) areas in time that represent relatively low risk of failure, which ops might want to take advantage of by adding additional missions or by shifting identified problem sorties in time.

Acknowledgement

The DARPA ANTS program (F30602-96-2-0227) has supported the activities described in this paper.

Reference:

1. www.mozart-oz.org
2. Derrick, T.C, and Pegman, M: Overview of constraint-based scheduling http://www.cs.unh.edu/ccc/archive/constraints/archive/app-sched.txt