# Aspects of Job Scheduling

K. Phillips
DSN Facility Operations Office

*A mathematical model for job scheduling in a specified context is presented. The model uses both linear programming and combinatorial methods. While designed with a view toward optimization of scheduling of facility and plant operations at the Deep Space Communications Complex (DSCC) at Goldstone, the context is sufficiently general to be widely applicable. The general scheduling problem including options for scheduling objectives is discussed and fundamental parameters identified. Mathematical algorithms for partitioning problems germane to scheduling are presented. A more detailed description of algorithms and of operational aspects of the model is planned for a later report.*

## I. Introduction

The efficiency and productivity of a service or production facility can be affected by the way in which the jobs performed by the facility are scheduled. As a part of the move to increase operational efficiency in the DSN, it was decided to study the effect of scheduling on the facility and plant function at the Deep Space Communications Complex at Goldstone and to partially automate the scheduling of maintenance. This report is a preliminary description of some of these efforts.

In general terms the object of mathematical scheduling is to identify measures of performance on the jobs done by a facility and to schedule the jobs to maximize the performance. Measures of performance are stated in such terms as maximal efficiency, minimal work inventory (backlog), and minimal lateness. The performance is measured by a function. The mathematical model of the problem then takes the form of maximizing or minimizing this performance function subject to the constraints of the problem. The constraints take many forms, of which a few are: one man cannot do two jobs at once; some jobs have higher priorities than others; the jobs are not all available to be worked on at all times; the jobs have different skill levels. The constraints in most real problems define a large and combinatorially complex set. This is the case for the model presented here. The model presented minimizes work inventory and flow time and maximizes gross efficiency. With additional analysis it can also yield schedules for maximal average efficiency, minimal maximal tardiness, or minimal average tardiness. Precise definitions appear in Section III.

Most of the results and discussion are presented in a general context, after an initial brief description of the immediate problem at Goldstone in Section II. The ideas have wide applicability. In Section III the general scheduling problem is discussed. A model for "flow-time scheduling" is presented in Section IV. Section V treats partitioning algorithms important for scheduling. In Section VI the Goldstone problem is again discussed.

## II. The Maintenance Operation at Goldstone

Facility maintenance is accomplished by seven shops: electrical, carpenter, etc. Each of these shops performs jobs in three categories, as follows:

(1) *Preventive maintenance*, consisting of periodic servicing and checking of equipment.

(2) *Corrective maintenance*, consisting of minor repairs and modifications.

(3) *Special jobs*, consisting of original construction or installation, and generally more complex than (2).

The information for scheduling function (1) is known well in advance, but functions (2) and (3) occur rather randomly and have varying priorities. Function (1), and possibly some portions of functions (2) and (3), have top priority in that the tasks must be done during the month called for by specifications. The users generally compete for services in categories (2) and (3). In this context possible scheduling objectives are:

(1) Keep all users as happy as possible by minimizing either the average or the maximum time the users must wait for completion of jobs.

(2) Operate the shop efficiently by minimizing job backlog.

(3) An intelligent balance between (1) and (2).

As explained in Section III, (1) and (2) are not generally compatible.

In Sections III through V we discuss aspects of general scheduling with only occasional mention of Goldstone. In Section VI we return to the context described here to give a more complete discussion of the options at Goldstone. An overall description of the proposed automation of the Deep Space Network Facility Operations appears in Ref. 7.

## III. Scheduling Problems

There is a large literature on scheduling and assignment problems, Refs. 1, 2, 5, and 6 form a sample and contain between them a large current list of references. Many difficult mathematical and computational problems arise in considering optimal scheduling. For this report a particular setting is chosen—general enough for several applications—to discuss some of these problems.

### A. A Context

Suppose that a service or manufacturing facility has $n$ jobs to do in a time interval $[0, q]$ and that there are $m$ processors (women, men, or machines) to do the jobs. Label the jobs $\{J_i\}_{i=1}^{n}$. Assume that the processors all have the same capability and that a certain fixed number $z_i$ of them work simultaneously on job $J_i$ until it is finished, but that this number can vary with the job. Associated with the job $J_i$ is its *processing time* $p_i$, the number of man hours required to complete the job. The actual time the job $J_i$ is being worked on is thus $y_i = p_i/z_i$. If $h$ is the total number of man hours available in the interval $[0, q]$, then we assume that

$$\sum_{i=1}^{n} p_i \leq h,$$

so that the jobs can in fact be done. It is assumed that $p_i$ and $z_i$ are schedule-independent; scheduling does not affect the time it takes to do a job or the number of processors needed. Each job $J_i$ also comes with a *due date* $d_i$ and is ready to be worked on at a *ready time* $r_i$ in $[0, q]$. Thus $p$, $z$, $d$, $r$, and $y$ are vectors or $n$-tuples; e.g., $d = (d_i)_{i=1}^{n}$. Other vectors associated with the jobs are defined below.

### B. Utility

Utilization of the facility is measured by the $n$-tuple

$$u = (u_i)_{i=1}^{n}$$

where

$$u_i = p_i/h \qquad \text{(utilization on } J_i\text{)}$$

Also let

$$U = \frac{1}{h}\sum_{i=1}^{n} p_i \qquad \text{(total utilization)}$$

$$Au = \frac{1}{n}U \qquad \text{(average utilization)}$$

These factors simply measure what portion of the capability of the facility is being used. They are schedule-independent.

### C. Schedule-Dependent Variables

Listed below are several schedule-dependent variables associated with the jobs $J_i$:

starting times: $s_i$

completion times: $c_i$

flow times: $f_i = c_i - r_i$
(time $J_i$ is in the shop)

late times: $\ell_i = c_i - d_i$

tardy times: $t_i = \max(0, \ell_i)$

waiting times: $w_i = s_i - r_i$

These definitions are illustrated on the time axis in Fig. 1.

For example, suppose that there are three jobs, $J_1$, $J_2$, $J_3$, only one processor ($m = 1$), and that $q = 5$. Suppose that the schedule-independent vectors are

$r = (0, 1, 1)$ ready times (i.e., $r_1 = 0$, $r_2 = 1$, $r_3 = 1$)

$p = (1, 1, 3)$ processing times

$d = (1, 5, 5)$ due dates

The vectors $z$ and $y$ are necessarily given by

$z = (1, 1, 1)$ only one processor

$y = p = (1, 1, 3)$ (work times)

If the jobs are done in order $J_1$, $J_2$, $J_3$ with no lag between the jobs, then

$$s = (0, 1, 2)$$

$$c = (1, 1 + 1, 2 + 3) = (1, 2, 5)$$

$$f = c - r = (1, 1, 4)$$

$$\ell = c - d = (0, -3, 0)$$

$$t = \max(0, \ell) = (0, 0, 0)$$

$$w = s - r = (0, 0, 1)$$

Notice in particular that flow time $f_i$ is the time that the job $J_i$ is in the shop. It is generally larger than the actual work time $y_i$ and in fact will be the same only if the ready time $r_i$ is equal to the completion time of the preceding job in the schedule. Such is the case for $J_1$ and $J_2$ in this schedule, but not for $J_3$. The total flow time for this schedule is $f_1 + f_2 + f_3 = 6$, while the total processing time is $p_1 + p_2 + p_3 = 5$.

### D. Efficiency

We define the efficiency vector $e = (e_i)_{i=1}^n$ of the schedule by

$$e_i = \frac{y_i}{f_i} = \frac{p_i}{z_i f_i} \tag{1}$$

Thus $e_i$ is the ratio of work time to flow time. The average efficiency is

$$Ae = \frac{1}{n} \sum_{i=1}^{n} e_i \tag{2}$$

Another measure of efficiency is *gross efficiency*, defined by

$$G = \left( \sum_{i=1}^{n} y_i \right) \Big/ \left( \sum_{i=1}^{n} f_i \right) \tag{3}$$

In general, $Ae \neq G$ (of course).

### E. Scheduling Objectives

If the jobs have no priorities (e.g., no bigger profit or penalty is associated with some jobs than with others) then objectives available for scheduling are:

(1) Minimize the average of $f$, $\ell$, $w$, or $t$.

(2) Minimize the maximum of $f$, $\ell$, $w$, or $t$.

(3) Maximize the average efficiency $Ae$.

(4) Maximize the minimum of $e$.

(5) Maximize gross efficiency $G$.

In this report we emphasize minimizing average flow time $Af$, discussed in Subsection III-F. Before beginning, note that the problem is combinatorially much too large to solve on a computer by direct computation of all possible schedules, even for fairly small $n$. For example if $m = 1$ and $r_i = 0$ for all $i$ then there are $n!$ schedules; recall that

$10! = 3{,}628{,}800$

$20! \cong 2{,}432{,}900{,}000{,}000{,}000{,}000$

$50! \cong 3 \times 10^{64}$

If the jobs have priorities, weights can be given to the coordinates of the vectors, and weighted averages and maximums can then be considered. Weighting factors are not considered in this report. There are in fact priorities for the jobs at Goldstone. In the model described in Section VI, these priorities are handled by categorizing then optimizing within categories. Weighted vectors will be developed as another alternative.

### F. Flow Time

Minimizing total flow time

$$\sum f \left( = \sum_{i=1}^{n} f_i \right)$$

or average flow time

$$Af = \frac{1}{n} \sum_{i=1}^{n} f_i$$

also achieves each of points (1)–(3) below. The letter $A$ is used for average, so that for a vector $v$, $Av$ denotes the average of the components of $v$.

(1) Maximizes gross efficiency $G$.

(2) Minimizes average waiting time $Aw$ and average lateness $A\ell$.

(3) Minimizes average backlog (or work inventory).

Assertion (1) is apparent from the definition of gross efficiency. To see (2) observe that

$$\sum_{i=1}^{n} w_i = \sum_{i=1}^{n} f_i - \sum_{i=1}^{n} y_i \tag{4}$$

$$\sum_{i=1}^{n} \ell_i = \sum_{i=1}^{n} f_i + \sum_{i=1}^{n} (r_i - d_i) \tag{5}$$

and that $y_i$, $r_i$, $d_i$ are all schedule-independent. Assertion (3) is not quite so trivial, although it is intuitively clear that the faster the work moves through the shop the smaller the outstanding workload. Letting $N(t)$ denote the work inventory at time $t$ (the number of jobs ready to be processed or in processing), the critical formula is

$$\int_0^q N(t)\, dt = \sum_{i=1}^{n} f_i \tag{6}$$

A proof of Eq. (6) is given in Appendix A. The formula proves assertion (3), for the left side of Eq. (6) is $q$ times the average backlog.

Another way to state Eq. (6) is

$$AN = \frac{1}{q} \sum_{i=1}^{n} f_i = \frac{n}{q} Af \tag{7}$$

The last expression is (arrival rate) · (average flow time). Equality (7) appears in Ref. 2, but with the unneeded hypothesis that the original $c_i$ are ordered.

### G. Minimizing Flow Time

The problem of minimizing total flow time $\Sigma f$ in all but the simplest cases appears to be unsolved in that no generally effective algorithms exist. A discussion appears in Ref. 2, where much of the exposition is devoted to the problem.

The simplest case is that in which all jobs have the same starting time ($r_i = 0$) and there is one processor ($m = 1$). Then a schedule is simply a permutation of the $n$ jobs. Suppose that the jobs are done in the order of the indices. The flow time of $J_1$ is simply its processing time $p_1$. Job $J_2$ is started on the completion of $J_1$, so its starting is $s_2 = p_1$ and its flow time is

$$f_2 = c_2 - r_2 = s_2 + p_2 - 0 = p_1 + p_2$$

Job $J_3$ is started at time $p_1 + p_2$ and so

$$f_3 = p_1 + p_2 + p_3$$

The flow time of the job $J_i$ (i.e., time required to complete $J_i$ from the time it was first available) is

$$f_i = \sum_{j=1}^{i} p_j \tag{8}$$

and total flow time is given by

$$\sum_{i=1}^{n} f_i = \sum_{i=1}^{n} \sum_{j=1}^{i} p_j = \sum_{i=1}^{n} (n - i + 1) p_i \tag{9}$$

For the fixed set of numbers $\{p_i : 1 \leq i \leq n\}$ the number on the right in Eq. (9) is minimal for the permutation in which the $p_i$ are ordered by increasing magnitude (See Appendix B). Hence the optimal flow time schedule in this case is that in which the jobs are done according to increasing processing times. This is called an LPT schedule, for "least processing time" first. In more complicated scheduling problems, the "LPT principle" can be applied to portions of the schedule after other requirements have been met.

In Section IV a mathematical model for achieving minimal flow time for the problem stated in Subsection III-A is presented.

### H. Tardiness and Flow Time

It seems clear that tardiness is a more important measure of performance than lateness, as defined above. There is no advantage as far as due dates are concerned in having the jobs finished early, that is in having $\ell_i < 0$. Unfortunately it is not true that minimum flow time also gives minimum average tardiness. The equality corresponding to Eq. (5) is

$$\sum_{i=1}^{n} t_i = \sum_{i=1}^{n} \max(0, c_i - d_i) = \sum_{i \in I} f_i + \sum_{i \in I} (r_i - d_i) \tag{10}$$

where $I$ is the set of indices defined by $I = \{i : c_i - d_i > 0\}$. The set $I$ is dependent on the schedule so there is no reason to expect that the right side of Eq. (10) is minimal with

$$\sum_{i=1}^{n} f_i$$

To see explicitly that minimal $\Sigma t_i$ and minimal $\Sigma f_i$ may require different schedules, consider the simple case where $m = 1$, $n = 2$. Assume that the ready times $r_1 = r_2 = 0$. In schedule $J_1J_2$ the start time vector is $s = (0, p_1)$ and the completion vector is $c = (p_1, p_1 + p_2)$. It is always assumed that $d_i > r_i + y_i$, so in this case

$$t_1 = \max(0, p_1 - d_1) = 0$$

and

$$t_2 = \max(0, p_1 + p_2 - d_2) = p_1 + p_2 - d_2$$

the last equality holding if $d_2 \leq p_1 + p_2$. The same analysis applies to the schedule $J_2J_1$ with the roles of 1 and 2 exchanged. Total flow time and total tardiness are thus given in the table below.

| Schedule | $\Sigma f$ | $\Sigma t$ |
|---|---|---|
| $J_1J_2$ | $2p_1 + p_2$ | $p_1 + p_2 - d_2$ |
| $J_2J_1$ | $2p_2 + p_1$ | $p_1 + p_2 - d_1$ |

If $p_1 < p_2$ and $d_1 > d_2$ then the first schedule minimizes total flow time but the second schedule minimizes total tardiness.

In the context of this report (the problem of Subsection III-A) the major tradeoff is between due date satisfaction in terms of either average of minimal maximal tardiness and minimal average flow time.

### I. Remarks on Flow Time and Outstanding Work

In discussing backlog in Subsection III-F, we called backlog the number of jobs outstanding. Another measure of remaining workload is processing time remaining, say $P(t)$ at time $t$. If it is assumed that all processors work continuously and no new work arrives, then it is clear that $P(t)$ is independent of schedule and in fact is linearly decreasing with slope $-1$. However, if we assume gaps in the work interval, then the graph of $P(t)$ will be a decreasing function with intervals of constancy or of smaller slopes.

To illustrate, suppose that $n = 5$ and $m = 1$ and that the gaps are constant. Then the area under the graph of $P$ is minimal if the *longest* jobs are done first (see Fig. 2). Hence under these assumptions minimizing average $P(t)$ is antithetical to minimizing average $N(t)$. In some cases minimizing $AP$ might be more realistic; "get the big jobs out of the way first." Such an optimization procedure could be further developed.

## IV. Model for Flow Time Scheduling

### A. Formulation

The context and notation are as in Subsection III-A; the goal is to achieve minimum flow time. That is, the goal is to find starting times for the jobs that make the corresponding flow time minimal. Clearly there are constraints on the starting times. For example, if there are two processors and ten jobs then an obvious constraint is that not all ten jobs can be started at $t = 0$. The constraints will yield certain allowable $n$-tuples $s = (s_i)_{i=1}^{n}$ as starting times for the jobs $J_i$. In geometric language, there is a certain allowable subset $A$ of $n$-dimensional real space in which the vector $s$ of starting times can lie. Among all these possible starting times in $A$ the object is to choose one which makes the total flow time for the jobs minimal. The set $A$ is a rather unwieldy collection of corners of an $n$-dimensional right parallelepiped. It is first described below for the general case and then its geometric properties are developed by considering simpler cases.

To begin, observe that total flow time can be expressed

$$\sum_{i=1}^{n} f_i = \sum_{i=1}^{n} (s_i + y_i - r_i) = \sum_{i=1}^{n} s_i + \sum_{i=1}^{n} (y_i - r_i) \tag{11}$$

Since $y_i$ and $r_i$ are schedule-independent, the problem of minimizing total flow time is the same as the problem of minimizing

$$\sum s \left( = \sum_{i=1}^{n} s_i \right)$$

The analysis that follows emphasizes $s$.

The ready-time and finish-time constraints are simple enough and are expressed by

$$r_i \leq s_i, \qquad s_i + y_i \leq q \tag{12}$$

Thus the allowable set $A$ must lie in the set $C$ of $n$-dimensional space $R^n$ defined by

$$C = \{s : r_i \leq s_i \leq q - y_i\} \tag{13}$$

The set $C$ is an $n$-dimensional right parallelepiped. The processor constraint that at most $m$ processors can be in use at any time is a little trickier. It may be phrased in terms of subsets of $R^n$ defined by $n$-tuples $\delta = (\delta_i)_{i=1}^{n}$ in which each $\delta_i$ is 0 or 1. Let $T$ denote the set of all such $\delta$ and define a subset of $T$ by

$$\Delta = \left\{ \delta \in T : \sum_{i=1}^{n} \delta_i z_i > m \right\} \tag{14}$$

For $\delta \in \Delta$, the set of jobs $J_i$ for which $\delta_i = 1$ cannot occur simultaneously; i.e., the number of processors required for this combination of jobs $J_i$ exceeds the total number of processors available. The condition that the jobs *do* occur simultaneously is that there is some time $t$ at which all jobs $J_i$ for $i$ in the set $I(\delta) = \{i : \delta_i = 1\}$ are being processed; i.e., $t$ is in $(s_i, s_i + y_i)$ for each $i$ in $I(\delta)$. In set notation, the subset of $R^n$ defined by

$$E(\delta) = \left\{ s \in R^n : \bigcap_{\delta_j = 1} (s_j, s_j + y_j) \neq \phi \right\} \tag{15}$$

is thus excluded as possible starting times, for each $\delta$ in $\Delta$. Hence the set

$$E = \cup_{\delta \in \Delta} E(\delta) \tag{16}$$

is excluded. The starting times must therefore be in the complement $E'$ in $R^n$ of $E$. The allowable set $A$ is thus given by

$$A = E' \cap C \tag{17}$$

where $C$ is defined by Eq. (13). Hence the flow time problem may be formulated as follows.

*Flow time problem:* find the minimum value of $\Sigma s_i$ for $s \in A$.

The set $A$ is discussed in more detail in Subsections IV-D and IV-E, after simpler cases have been described in Subsections IV-B and IV-C.

The set $A$ can also be written

$$A = \bigcap_{\delta \in \Delta} A(\delta) \tag{18}$$

where

$$A(\delta) = E(\delta)' \cap C \tag{19}$$

In many cases the intersection defining $A$ can actually be taken over a smaller set than $\Delta$ : for, if $A(\delta) \subset A(\sigma)$ then $A(\sigma)$ need not be included in the intersection.

### B. Illustrative Example

Suppose there are two jobs for scheduling and one processor, i.e., $n = 2$, $m = 1$. Suppose that $r_1 = 0$, $r_2 > 0$. The set $C$ is

$$C = \{s : s_1 \leq q - p_1;\ r_2 \leq s_2 \leq q - p_2\} \tag{20}$$

a rectangle. The only $\delta$ in $\Delta$ is $\delta = (1, 1)$; i.e., both jobs cannot be processed at once. The excluded set $E = E(\delta)$ is

$$E = \{s = (s_1, s_2) : (s_1, s_1 + p_1) \cap (s_2, s_2 + p_2) \neq \phi\}$$

from which it follows that

$$E' = \{s : s_1 + p_1 \leq s_2 \text{ or } s_2 + p_2 \leq s_1\} \tag{21}$$

Hence the allowable set $A$ is given by

$$\begin{aligned} A &= (C \cap \{s : s_1 + p_1 \leq s_2\}) \cup (C \cap \{s : s_2 + p_2 \leq s_1\}) \\ &= [C \cap A_1] \cup [C \cap A_2] \end{aligned} \tag{22}$$

Simply stated, the requirement that the starting time vector $s = (s_1, s_2)$ be in $E'$ given in Eq. (21) means that either $J_2$ must start after $J_1$ is finished, or $J_1$ must start after $J_2$ is finished. In Fig. 3, $C$ has boundary indicated by short cross lines and $C \cap A_1$ and $C \cap A_2$ are indicated by shading. The graph of $\Sigma s$ is a plane inclined at 45 deg to

the $s_1 - s_2$ plane and lying above the first quadrant. The minimum of $\Sigma s$ for $s \in A$ occurs at one of the nodes (or extreme points) of the set A, namely the one for which $s_1 + s_2$ is minimal. If lines of slope $-1$ are drawn through the nodes, then the line that lies closest to the origin gives the node for minimal $\Sigma s$ because $s_1 + s_2$ is constant on these lines. In Fig. 3 these lines are sketched with dots and labeled.

In the case sketched, minimal $\Sigma s$ occurs at $N_1 = (0, r_2)$ with value $\Sigma s = r_2$. The flow time value is

$$f(0, r_2) = r_2 + (p_1 - r_1) + (p_2 - r_2) = p_1 + p_2$$

hardly unexpected for this simple case.

The set A has two components in this case. Clearly one need not check all nodes, but only one node in each component, the one closest to the origin; $N_1$ and $N_3$ in Fig. 3.

If $r_2 < p_1$, then $N_1$ does not appear and the choice is between $N_2$ and $N_3$. In this case $N_2 = (0, p_1)$. What is required for $N_3$ to give the minimum? Since

$$\Sigma s(N_2) = p_1 \text{ and } \Sigma s(N_3) = 2r_2 + p_2$$

the inequality

$$2r_2 + p_2 < p_1 \tag{23}$$

is required. Descriptively speaking, if Eq. (23) holds, it pays to wait and do the short job first even if the long job is available and the short one is not, provided that the long job is sufficiently long. A case in which this occurs is: $r_1 = 0$, $r_2 = 5$, $p_1 = 17$, $p_2 = 2$. See Fig. 4.

### C. The One Processor Case

If $m = 1$, then $z_i = 1$ for all $i$. The set $\Delta$ is all $n$-tuples of zeros and ones that contain at least two ones. Suppose $\delta$ contains *exactly* two ones, say $\delta_i = \delta_j = 1$ and $\delta_k = 0$ if $k \neq i, k \neq j$. Then

$$E(\delta) = \{s : (s_i, s_i + p_i) \cap (s_j, s_j + p_j) \neq \phi\}$$

and

$$E'(\delta) = \{s : s_i + p_i \leq s_j \text{ or } s_j + p_j \leq s_i\} \tag{24}$$

The set $E'(\delta)$ is like $E'$ in Eq. (21). In fact the projection of $E'(\delta)$ in the $s_i - s_j$ plane is precisely as analyzed in Subsection IV-B ($n = 2$). Every set $E(\sigma)$ for $\sigma \in \Delta$ contains some $E(\delta)$ in which $\delta$ has exactly two ones. It follows that the set A can be expressed

$$A = C \cap \left[ \bigcap_{i \neq j} (\{s : s_i + p_i \leq s_j\} \cup \{s : s_j + p_j \leq s_i\}) \right] \tag{25}$$

where the second intersection is to be taken over all pairs $(i, j)$ with $i \neq j$. The projection of A in each $s_i - s_j$ plane has precisely the geometry described in Subsection IV-B. The set A is thus the union of certain corners of an $n$-dimensional right parallelepiped. The restriction on each pair of coordinates "chops off" a corner at 45 deg.

Another description of A follows. Let $\Omega$ be the set of $\delta$s with exactly two ones. One can think of each set $E(\delta)$ for $\delta \in \Omega$ as the infinite cylinder over the infinite rectangle in the $s_i - s_j$ plane defined by

$$R_{ij} = \{s : s_i < s_j + p_j \text{ and } s_j < s_i + p_i\} \tag{26}$$

The set $R_{ij}$ is inclined 45 deg to the $s_i$ and $s_j$ axes. The excluded set

$$E = \bigcup_{\delta \in \Omega} E(\delta)$$

is the union of these cylinders and the allowable set A is the intersection of C with the complement of E.

### D. General Case; $z_i$ and $m$ Arbitrary

Each set $E(\delta)$ for $\delta \in \Delta$ can be analyzed by considering pairs of coordinates. For a given $\delta$ and each pair $(i, j)$ for which $\delta_i = \delta_j = 1$, let

$$R_{ij} = \{s : (s_i, s_i + y_i) \cap (s_j, s_j + y_j) \neq \phi\}$$

just as described above. Then

$$E(\delta) = \cap_P R_{ij} \quad (P = \{(i, j) : \delta_i = \delta_j = 1\}) \tag{27}$$

To see that Eq. (27) holds suppose first that $s \in E(\delta)$. Then clearly $(s_i, s_i + y_i) \cap (s_j, s_j + y_j) \neq \phi$ for each pair $(i, j) \in P$. Hence $s \in \cap_P R_{ij}$ and $E(\delta) \subseteq \cap_P R_{ij}$ holds. To prove the reverse inclusion, suppose $s \in \cap_P R_{ij}$. Let $s_k$ be the largest coordinate of $s$. Then $s_k \in [s_i, s_i + y_i]$ for all $i$ having $\delta_i = 1$ because $s \in R_{ki}$ for all $i$. Hence

$$s_k \in \cap_{\delta_i = 1} [s_i, s_i + y_i]$$

and so $s \in E(\delta)$.

Each $\delta \in \Delta$ thus gives rise to an excluded set which is an intersection of rectangular cylinders above the coordinate planes. The excluded set within the set $C$ is then the union over $\delta \in \Delta$ of all these $E(\delta)$. This is the "heart" of the parallelepiped $C$ and the allowable set $A$ is pieces of the corners. The set $T$ of all $\delta$ can be ordered by "$\delta \leq \sigma$ if $\delta_i = 1 \Rightarrow \sigma_i = 1$." The intersection defining $A$ can be taken over the set $\Delta_1$ of least elements in $\Delta$.

### E. Summary and Algorithm Outline

A procedural outline for finding minimal $\Sigma s$, and hence minimal flow time $\Sigma f(s)$ follows.

(1) Determine a minimal set $\Delta_1$ of $\delta$s for which

$$\sum_{i=1}^{n} \delta_i z_i > m$$

The set $\Delta$ over which the intersection is taken can be replaced by $\Delta_1$. This is a combinatorial partitioning problem and will be discussed in more detail in Section V.

(2) Determine the set $C$. This is a simple matter; $C$ is defined by $2n$ inequalities, hence by $2n$ numbers.

(3) Identify the nodes of the components of the set $A$ and determine those which minimize $\Sigma s$. Use a modification of the simplex method to determine feasible nodes and minimum. While $A$ is not convex its complement in $C$ is the set $E = C \cap [\cup_{\delta \in \Delta} E(\delta)]$, which is a union of convex sets. However $E$ does not have all the extreme points of the complement of $A$, as a glance at Fig. 2 clearly demonstrates; $N_1$ is an extreme point of $A$ but not of $E$.

## V. Remarks on Partitioning

As indicated in Section IV, partitioning algorithms are important for scheduling. Given positive integers $\{z_i : 1 \leq i \leq n\}$ and a number $m$, the problem is to find all subsets $J$ of $\{1, 2, \cdots, n\}$ for which

$$\sum_{J} z_i = m$$

The problem is equivalent to finding all $n$-tuples $\delta$ of zeros and ones for which

$$\sum_{i=1}^{n} \delta_i z_i = m$$

Bounds for the number of such partitions can be obtained using combinatorial methods appearing in Refs. 3 and 4. To the author's knowledge there is no known probabilistically optimal algorithm for obtaining all partitions. The state of the art appears to be contained in the papers Refs. 5 and 6, where statistical comparisons are made for various algorithms. In two steps, Subsections V-A and V-B, we describe the best of Ref. 5.

### A. Obtaining All Sums

We use the notation $\{0, 1\}^n$ for the set of $n$-tuples of zeros and ones. In this algorithm the sum

$$S(\delta) = \sum_{i=1}^{n} \delta_i z_i$$

is calculated for every $\delta \in \{0, 1\}^n$ and the $\delta$ are coded. For $0 \leq r \leq n$ define sets of ordered pairs $A_r$ recursively by

$$A_0 = \{(0, 0)\}$$

$$A_r = A_{r-1} \cup \{A_{r-1} + (z_r, 2^{r-1})\}$$

All sums using $z_1, \cdots, z_r$ appear as first coordinates in $A_r$. Since the correspondence

$$\delta \leftrightarrow \sum_{i=1}^{r} \delta_i 2^{i-1}$$

is $1 - 1$ between $\{0, 1\}^r$ and $\{0, 1, 2, \cdots, 2^{r-1}\}$ the second coordinate of an element of $A_r$ contains a unique description of the $\delta$ used to obtain the first coordinate. The set $A_n$ is thus

$$A_n = \{(S(\delta), \delta) : \delta \in \{0, 1\}^n\}$$

where the second coordinate is "coded" as above.

If only certain sums are wanted, then in the recursive step from $A_{r-1}$ to $A_r$ all sums that cannot possibly yield the desired sums are eliminated. This elimination is optimized by ordering the $z_i$. The sum is then decoded to give $\delta$ for the desired sums.

### B. An Improvement

For the problem of obtaining all partitions of $m$ for a given $m$ an improvement in computing time is achieved by first splitting the $z_i$ into two sets, applying Subsection V-A procedures to each of these, then combining the resulting sums. The analysis appears in Ref. (5). Combinatorial complexities prevent additional improvement by further splitting.

## VI. Models for Maintenance Scheduling at Goldstone

Some of the jobs in categories (2) and (3) of Section II have high priority; essentially they must be done when they arise. A subset of these can be anticipated; that is, their ready times $r_i$ are known, while the others occur at random. Using scheduling interval $[0, q]$ equal to a month, in the notation of Section III the jobs fall into categories $C_1$, $C_2$, $C_3$, and $C_4$ defined by

$C_1$ : preventive maintenance; $r_i = 0, d_i = q$

$C_2$ : high priority; $r_i$ known but variable, $d_i = r_i + y_i$; "must" be done at $t = r_i$

$C_3$ : variable $r_i$ and $d_i$; no other priorities

$C_4$ : random; unknown prior to scheduling, but must be done as they arise.

There is an additional problem among the $C_1$ jobs in that the individual jobs are so numerous that they cannot reasonably be handled directly. They are first categorized by building and site and then consolidated by a partitioning algorithm like that in Subsection V-A.

A first step in any scheduling routine leaves time for the expected $C_4$ jobs. A weekly modification routine can then be applied to adjust for the $C_4$ jobs that occurred. Given this context, three options for scheduling are listed below.

Option 1:
1. Schedule the $C_2$ jobs.
2. Apply the flow time model described in Sections IV and V to all remaining jobs.
3. Make weekly modifications.

Option 2:
1. Schedule the $C_2$ jobs.
2. Apply the flow time model to the $C_1$ jobs and as time allows to the $C_3$ jobs for the first three weeks only.
3. At beginning of fourth week apply the flowtime model to all jobs remaining as result of interruption by $C_4$ jobs.

Option 3:
1. Schedule the $C_2$ jobs.
2. Apply a "Due-Date" algorithm instead of a "Flow-Time" algorithm to the entire month.

Computer programs for these options are being written. The options will be tried experimentally on data available for scheduling and compared using the ideas presented in Section III.

## References


1. Ashour, S., *Sequencing Theory*, Springer, 1972.
2. Conway, Maxwell, and Miller, *Theory of Scheduling*, Addison-Wesley, 1967.
3. Dilworth, R. P., "A Decomposition Theorem for Partially Ordered Sets," *Ann. Math.* Vol. 2, No. 51, pp. 161–166, 1950.
4. Hall, Marshall, Jr., *Combinatorial Theory*, Blaisdell, 1967.
5. Horowitz, Ellis and Sahni, Sartaj, "Computing Partitions With Applications to the Knapsack Problem," *Jour. of the Assoc. for Computing Machinery*, Vol. 21, No. 2, pp. 277–292, April 1974.
6. Horowitz, Ellis and Sahni, Sartaj, "Algorithms for Scheduling Independent Tasks," *Jour. of the Assoc. for Computing Machinery*, Vol. 23, No. 1, January 1976.
7. Maiocco, F. R., and Hume, J. P., "Computerizing Goldstone Facility Maintenance Data for Management Decisions," in *The Deep Space Network Progress Report 42-32*. Jet Propulsion Laboratory, Pasadena, California, April 15, 1976.

# Appendix A

## Proof of Equation (6)

**Lemma.** With definitions as in Section III, the equality

$$\int_0^q N(t)\,dt = \sum_{i=1}^{n} f_i \qquad \text{(A-1)}$$

holds.

***Proof.*** Assume that the jobs are numbered so that

$$0 = r_1 \le r_2 \le \cdots \le r_n < q$$

Let $r_{n+1} = q$. Let $\{c'_i : 1 \le i \le n\}$ be a relabeling of $\{c_i\}_{i=1}^{n}$ which puts them in order; thus $0 < c'_1 \le c'_2 \le \cdots \le c'_n = q$.

The number $N(t)$ is the number of jobs with $r_i \le t$ and $c_i > t$. The equality $N(t) = R(t) - S(t)$ holds where

$$R(t) = \text{number of } \{r_i : r_i \le t\}$$

$$S(t) = \text{number of } \{c_i : c_i \le t\}$$

The integrals of $R$ and $S$ are given by

$$\int_0^q R(t)\,dt = \sum_{i=1}^{n} \int_{r_i}^{r_{i+1}} R(t)\,dt$$

$$= \sum_{i=1}^{n} i(r_{i+1} - r_i)$$

$$= nq - \sum_{i=2}^{n} r_i$$

and

$$\int_0^q S(t)\,dt = \sum_{i=2}^{n} \int_{c'_{i-1}}^{c'_i} S(t)\,dt$$

$$= \sum_{i=2}^{n} (i-1)(c'_i - c'_{i-1})$$

$$= (n-1)q - \sum_{i=1}^{n-1} c'_i$$

(recall that $r_1 = 0$). Hence

$$\int_0^q N(t)\,dt = nq - (n-1)q + \sum_{i=1}^{n-1} c'_i - \sum_{i=2}^{n} r_i$$

$$= \sum_{i=1}^{n} c'_i - \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} f_i$$

which proves Eq. (A-1).

# Appendix B

## A Minimal Sum

**Lemma.** Let $\{p_i\}_{i=1}^{n}$ be positive numbers. For a permutation $\nu$ of $\{1, \cdots, n\}$, let

$$S(\nu) = \sum_{i=1}^{n} (n - i + 1)\, p_{\nu(i)} \tag{B-1}$$

Then $S(\nu)$ is minimal for all permutations if

$$p_{\nu(1)} \leq p_{\nu(2)} \leq \cdots \leq p_{\nu(n)} \tag{B-2}$$

***Proof.*** Clearly $S(\nu)$ is minimal if and only if

$$E(\nu) = \sum_{i=1}^{n} i p_{\nu(i)}$$

is maximal. If $\mu$ is such that $E(\mu)$ is maximal but Eq. (B-2) does not hold then

$$p_{\mu(i)} > p_{\mu(j)} \qquad \text{and} \qquad i < j \tag{B-3}$$

holds for some $i$ and $j$. However the inequality

$$i p_{\mu(i)} + j p_{\mu(j)} < i p_{\mu(j)} + j p_{\mu(i)}$$

holds by Eq. (B-3) and shows that $E(\nu) > E(\mu)$ if $\nu$ is obtained from $\mu$ by interchanging $i$ and $j$. Thus $E(\mu)$ cannot be maximal if Eq. (B-2) does not hold for $\mu$.

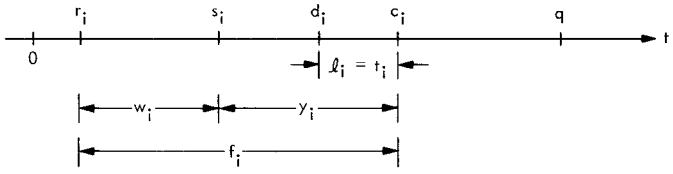
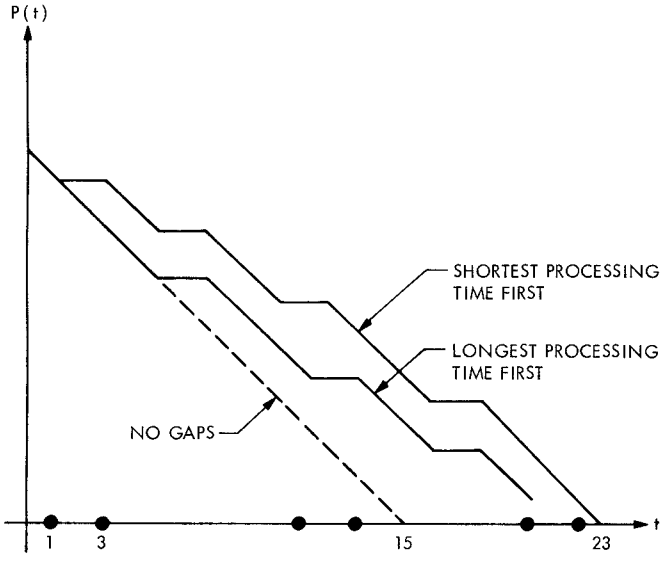



Fig. 1. Illustration of scheduling parameters

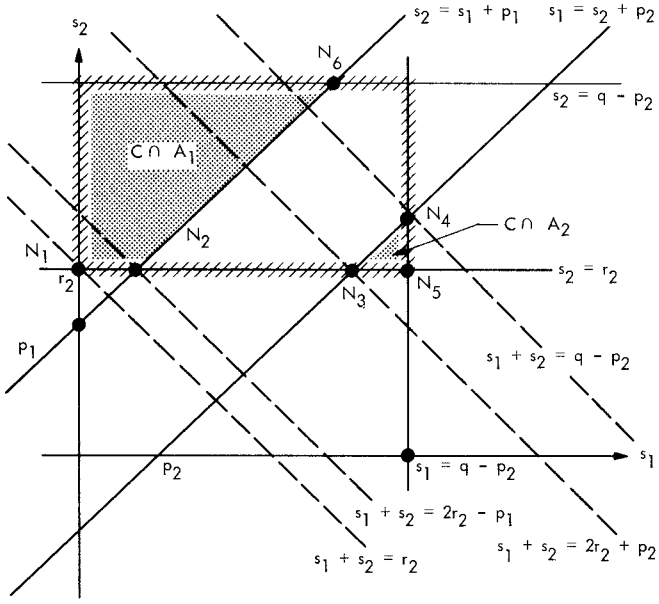



GAP = 2

$p_1 = 1,\ p_2 = 2,\ p_3 = 3,\ p_4 = 4,\ p_5 = 5$

P(t) = PROCESSING TIME REMAINING

Fig. 2. Outstanding processing time with gaps



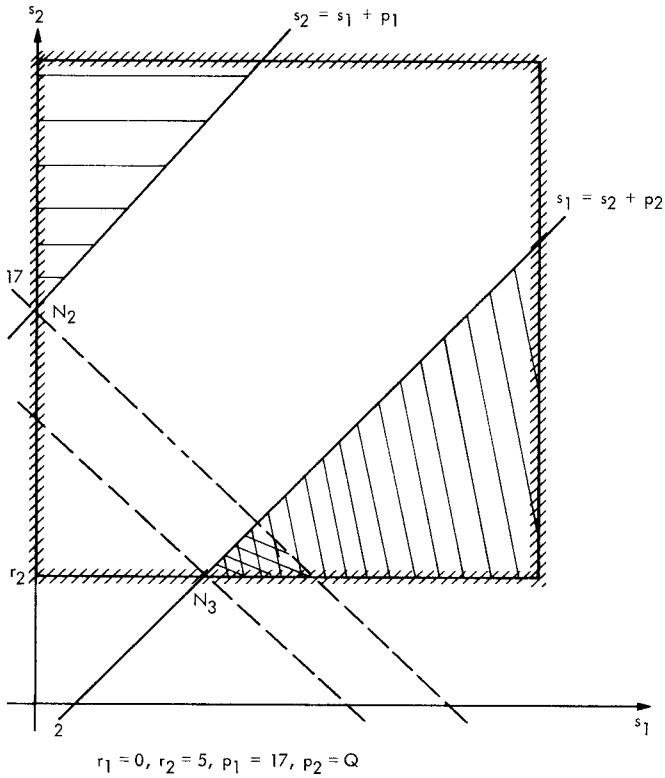

NODES OF A: $N_1 = (0, r_2)$

$N_2 = (r_2 - p_1, r_2)$

$N_3 = (r_2 + p_2, r_2)$

$N_4 = (q - p_2, q - 2p_2)$

Fig. 3. Allowable set of starting times




$r_1 = 0,\ r_2 = 5,\ p_1 = 17,\ p_2 = Q$

Fig. 4. Allowable starting times ($p_1 >> p_2$)