Selling television time: an optimisation problem

By A. R. Brown*

This paper presents the continuing effort by the author and his colleagues to solve a problem inherent in the selling activities of their Company. The problem is optimisation of spare capacity: in this case of saleable advertising time, but similar problems often arise in stockholding operations where the stock can be reduced by part of a whole item.

The first solution, now superseded, is presented at length because of its interesting recursive programming technique.

(Received December 1968)

The problem

Industry background

Commercial television in the United Kingdom finances itself from the transmission of advertising material between programmes. The commercial television companies transmitted over 507,500 individual advertisements in 1967, which will give some idea of the size of the business.

Thames Television transmits about 110 individual 'commercials' to the London area each weekday. Generally, time for each advertisement is reserved three to four months before transmission. This clearly represents quite a reservation problem administratively. The problem becomes more involved when it is realised that there is a large amount of 'chopping and changing' of future bookings by the advertising agencies.

The mammoth reservation problem has obvious similarities with the airlines' seat-reservation activities. The unit-cost of the item being sold is high in both cases. In this kind of environment, even a small increase in efficiency in the reservation system can significantly affect revenue. Maximum efficiency can only be achieved if the reservation files can be kept up-to-date and immediately accessible. The answer is clearly a real-time computer system.

System background

In August 1966 a UNIVAC 1050 real-time system came into operation for Rediffusion Television. This remained the case until 30 July 1968 when Thames Television took over the London weekday ITV contract from Rediffusion. During its first year of operation for Rediffusion, the system greatly increased the selling efficiency of the Company culminating in a substantial addition to the annual revenue figure.

The selling operation and its use of the computer has been described briefly elsewhere (Tatham, 1967). The basic system consists of a number of on-line teleprinters located in the sales offices; these feed into the computer system, in real-time mode, all details of bookings and changes. The terminals are obviously also given facilities for interrogation of the computer-held files. New bookings are, if space is still available, slotted into a matrix of future commercial breaks held within the system. The number of transactions (i.e. bookings, cancellations, enquiries, etc.) entered through the teleprinters each day averages around 1200.

Need for optimisation

The process of slotting new advertisements into the previously set-up pattern of future commercial breaks is straightforward—an exercise in efficient data storage and retrieval. There are, however, a number of constraints imposed on the process (described later) which are fairly complex.

The most troublesome complications arise over optimisation of the usage of available space. Small amounts of time may be left unsold in several advertising breaks, which are usually of a predefined duration. Consequently, if this could be collected together, it may be possible to create space for other commercials. This is a very common problem (which has parallels in other industries) and its solution, in the television environment, is the concern of this paper.

Some definitions

Adjacent advertising breaks in an evening are grouped together into price *segments*. Advertisements are generally priced according to the expected audience, and, for example, advertisements transmitted early or late in the evening are much cheaper than during 'peak viewing' hours. The difference in price between late-afternoon segment (say 5 p.m.) and 'peak' segment (usually 6.30–10.30 p.m.) is roughly a fivefold increase, as is the size of the audience.

The number of breaks in a price-segment varies from less than five for most of the early or late evening to about 12 breaks in 'peak' segment. The duration of breaks is usually a multiple of 15 seconds and varies between one minute and $3\frac{1}{2}$ minutes. There is a maximum allowed limit of advertising which is seven minutes in any one hour.

Incidentally, it should be made clear that, except in the rare case of 'live' or videotaped commercials, the provision of advertising material in the form of film is the responsibility of the advertiser. All he buys from

* *Thames Television Ltd., Television House, Kingsway, London WC2: now with Jeffreys & Hill Ltd., 4 Half Moon St., London W1*

the television company are the facilities to show his film to their audience, plus associated services.

The usual term for a length of advertising time is a *spot*. Besides the obvious details of who reserved them and for when, spots have only a few characteristics that can vary and are of interest during the reservation process. These factors are:

(*a*) *Duration:* which may be 7 seconds or any multiple of 15 seconds. The most common durations are 7, 15 and 30 seconds.
(*b*) *Product Type:* a manufacturer of, say, dog food would not be pleased if his product and other dog foods were advertised in the same break. Such a situation is called *product clash* and its avoidance is a rigid requirement of the reservation system.
(*c*) *Spot Mobility:* a spot is normally booked into a specific segment on a specified day and it is then left up to the television company in which actual break within that segment the spot is transmitted. This is termed a *broad spot*. However, for a surcharge, the advertiser may specify precisely in which break he wishes his spot to appear. This is known as a *fixed spot*.

Precise definition of problems

The program which performs the booking process within the computer system may come across several specific problems. If it could solve these problems automatically, a great saving in time would result. This paper discusses several automatic solutions.

Problem 1—Booking fixed spot

In handling a fixed-spot booking the program is only interested in the break requested. Assuming time is available for the spot in the segment as a whole, it may find several conditions in the break examined:

(*a*) Time available and no 'clash' with products already booked for that break, in which case the time can be reserved as requested.
(*b*) Time available but the spot clashes with a similar product *fixed* into that break. In this case, nothing can be done. The spot already booked fixed cannot be moved, so a new booking is impossible.
(*c*) Time available but the spot clashes with a *broad* spot already booked. The broad spot could possibly be moved to another break.
(*d*) Time is not available in the break but there are sufficient broad spots in the break such that, if some or all of them were moved to other breaks, time could be created.
(*e*) Time is not available in the break and there are not sufficient broad spots so that, even if they were all moved to other breaks, there would still not be enough time to book the new spot.

Condition (*a*) is clearly not problematical. States (*b*) and (*e*) are impossible to handle. The cases in which some useful effect might be felt from a process of moving broad spots are (*c*) and (*d*).

Problem 2—Booking broad spot

When attempting to book a broad spot the program examines all breaks in the segment concerned. Assuming there is available time in the segment to accept the spot, several conditions may be found:

(*a*) There is a break with sufficient time available for the spot and no clashing spot already booked. In this case the spot can be reserved as requested.
(*b*) The break or breaks in which there is time for the spot already have a fixed spot which clashes.
(*c*) The break or breaks with time available have a clashing broad spot already booked.
(*d*) Combination of (*b*) and (*c*) over several breaks.
(*e*) 'Time spread': small amounts of time (not sufficient for the new spot) are available in several breaks. In total they offer enough time for the new spot; thus the available time needs optimising into one break.

There is no problem in case (*a*) but all the other conditions could benefit from a re-distribution of spots over the breaks in the segment.

Manual solution

The Terminal Operator discovers that one of the problems defined above is present from the computer's response to her booking request. In the absence of any automatic problem-solving facilities, the sequence of events would then be:

1. Obtain through the teleprinter a printout of all spots (and their break by break disposition) in the segment.
2. By examining the printout, decide if a reorganisation of spots to fit in the new one is possible.
3. If so, perform a series of transactions through the terminal to change the position of spots as required.
4. Again attempt the booking, which should now be successful.

The above sequence of events can be highly time-consuming. A peak-viewing segment can easily contain 50 or 60 spots spread over 12 or so breaks; an off-peak segment may only have 15 spots in, say, 4 breaks. Typical times for the procedure above might be:

	PEAK, MINS.	OFF-PEAK, MINS.
1. Discover problem	1	1
2. Take segment printout	6	2
3. Decide what to be done, say	5	1
4. Perform reorganisation	10	5
5. Retry booking	1	1
Total time spent on booking one spot	23	10

This kind of problem is extremely common and can easily arise twenty times a day. Clearly the 'manual' solution uses a very large amount of terminal time and the introduction of an automatic solution when the problem is discovered (i.e. in the booking program) provides an enormous benefit.

Our first automatic solution

Technique

The first automatic re-scheduling algorithm is a complex piece of logic added to the spot booking transactions. Its basic technique has been named recursive broad spot dispersal. At the heart of the logic is a routine called Broad Spot Dispersal (described in detail below), the purpose of which is to move as many broad spots as possible from a given break. If it is unsuccessful in moving a broad spot into another break it can *call itself* to disperse the broad spots in that other break.

The logic (which will be called *auto-rescheduling* in the following) was inserted into the spot booking transaction at the point where one of the problem conditions described above has been found to exist. If the auto-rescheduling algorithm succeeds in improving the situation enough to book the spot, this is done and the user is almost unaware that the segment has been rescheduled for him. If it is unsuccessful, an appropriate message is sent to the terminal.

In order that the 'reshuffling' attempts of auto-rescheduling should be as fast as possible, rather than working on the actual drum-held spot records themselves, the algorithm first of all extracts all relevant details and stores them in memory as a 'spot map'. All work, up to the point just after deciding if the attempt has been successful, is done on this memory-held 'spot map'. If the operation is a success, the drum records are then set to match the disposition of spots in the 'map'. It should be pointed out that, during the rescheduling process, the relevant main-file records are locked out from all other terminals.

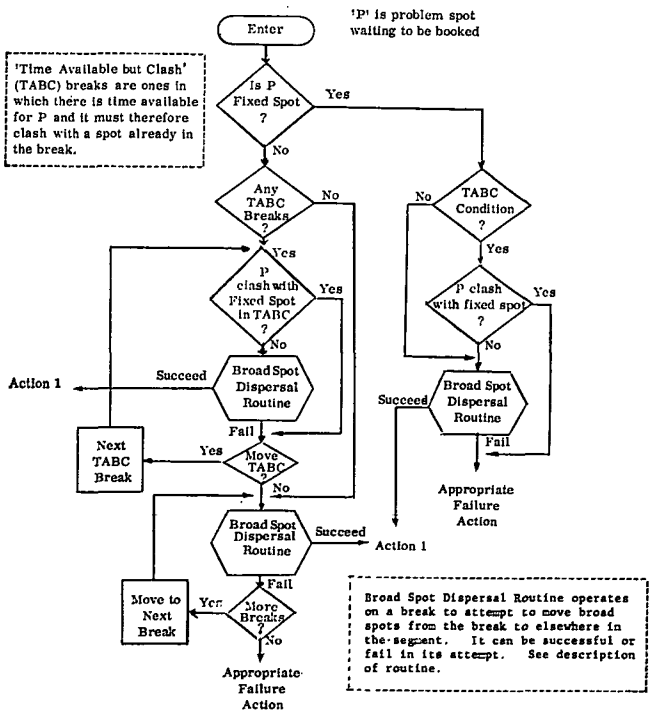


Fig. 1. Flowchart of central auto-rescheduling logic

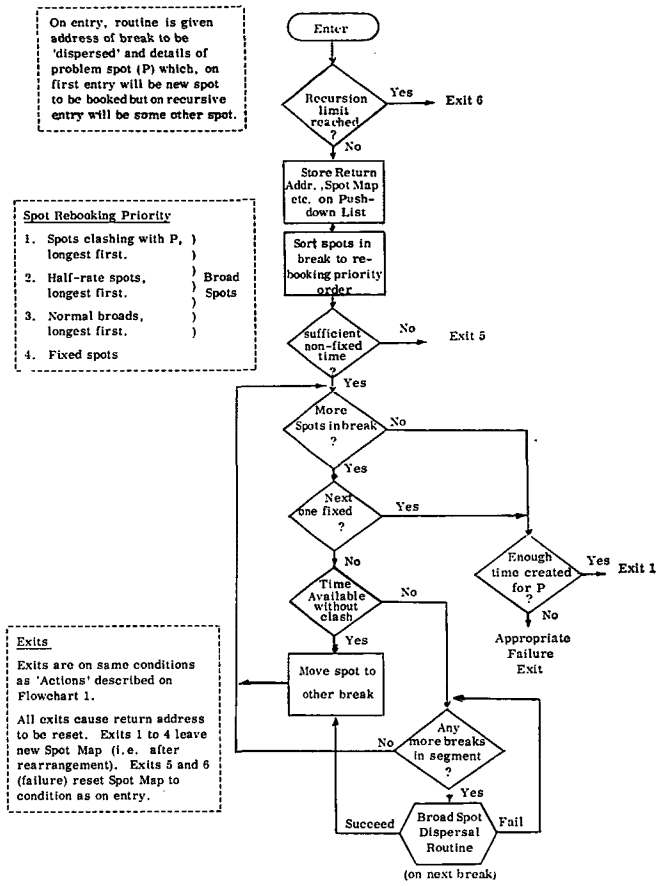


Fig. 2. Flowchart of broad spot dispersal routine

Main logic

The flowchart of **Fig. 1** outlines the central part of the algorithm. This central part is preceded by an initial detail-extraction phase (to make up the spot map in memory) and followed, in the case of Actions 1 to 4 (success or partial-success), by logic to reorganise the drum-held spot records to match the 'spot map'.

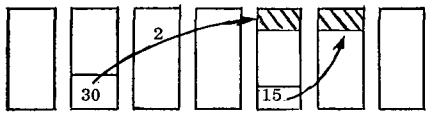
The process when we are solving a fixed spot problem is fairly straightforward. The dispersal routine is simply called to attempt to move broad spots from the break concerned.

For the broad spot case, the attack is on two levels. First of all the most likely breaks in the segment are examined: these are the ones that have time available for our problem spot but have a clashing spot already booked. If no successful conclusion is reached after trying to move these clashing spots, a more general attempt is made to reorganise the segment by examining each break in turn. As soon as a suitable reorganisation is found, the attempt terminates. If one or more partially-successful states have been found, the most suitable is chosen. If the attempt is a complete failure, the algorithm exits accordingly. See the terminal conditions on the flowchart of **Fig. 1.**

Broad spot dispersal algorithm

The flowchart shown in **Fig. 2** describes this subroutine. The essential technique is to attempt to rebook, according to a predefined priority, all broad spots in the break concerned. The power of the algorithm stems from its recursive nature (i.e. it can call itself). For each broad spot in turn, it first of all tries a simple rebooking procedure by searching for a suitable block of available time elsewhere. If this is not found, it tries to create it by calling itself to operate on another break in the segment. For example (see **Fig. 3**), the routine could be successful in creating time in, say, break 2 of the segment illustrated by moving a spot from break 5 to break 6 so that it could move a spot from 2 to 5. The maximum allowable number of breaks that may be being dispersed at the same time (i.e. levels of recursion) is one less than the number of breaks in the segment. Thus it can be seen that the example just described is fairly simple; considerably more complex situations involving multiple consequential moves are possible.

Object: to create 30 seconds available time in Break 2
(shading is available time)



		Break	1	2	3	4	5	6	7
Time Available (seconds)	(	Before	0	0	0	0	15	15	0
	(	After	0	30	0	0	0	0	0

Order of Movement

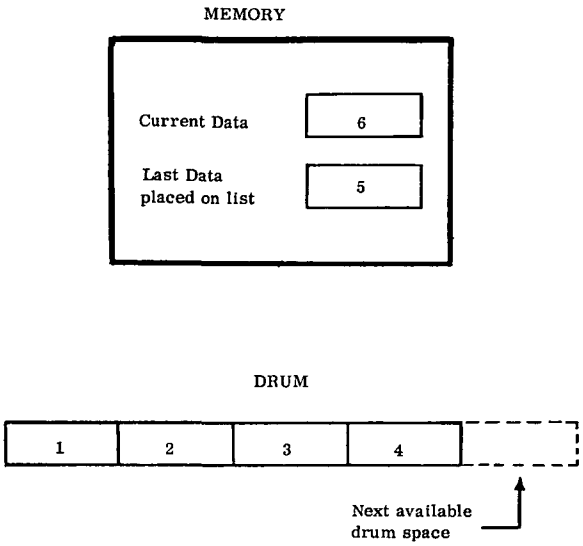
1. 15 sec. spot from 5 to 6
   (giving 30 secs. available in 5)
2. 30 sec. spot from 2 to 5

Fig. 3. A typical recursion-level 1 move

The programming techniques involved in a recursive routine are not necessarily complicated. All information in respect of the 'current state' has to be stored on entry, as it has to be reset on exit. This status data is stored on a *push-down list*: a sequence of items where the last on the list will be the first to be removed. The data stored in this way by Broad Spot Dispersal is:

(*a*) Routine Return Address.
(*b*) State of 'spot map' on entry.
(*c*) Location of break/spot being processed.

The last set of data stored is kept in memory and earlier stored data on backing drum, purely because of size problems: ideally all stored data should be in memory.



Numbers refer to (and represent) sets of "current data" (return addresses, etc.) stored on successive recursive calls of routine.

Fig. 4. State of push-down list on a 5-level recursion

Fig. 4 represents the situation of the push-down list during a 5-level recursion. A further recursion would cause status 5 data to be written to the next space on drum and status 6 copied to the 'last' (i.e. top of list) position. A failure of recursion 5 would cause status 5 to be reset into the 'current' position and status 4 data copied (from drum) to the 'last' position.

A wholly- or partially-successful entry of the routine causes the spot map in the current position to be left where it is (the situation has been improved) and the only information reset is return address. An unsuccessful entry causes all information to be reset as on entry.

Implementation difficulties

The main problematical area in the writing of a recursive routine is testing. In the early stages of 'debugging', recursion has to be limited or deciphering what actually happened when an error occurs quickly becomes impossible. The program described above was only proved beyond two levels of recursion by its 'statistical' success: providing it did not actually stop, its success was measured as the number of cases it solved expressed as a percentage of the number of cases it was given that did have tractable solutions. This figure was around 70%.

A difficulty in the early stages of testing was finding various addresses, indicators and other factors which unexpectedly contributed to the 'current state of things' and should have been stored on recursion. This was obviously soon rectified but caused some extremely involved debugging at first.

In action

When the automatic rescheduling logic was added into the various spot booking transactions available, a significant difference was noted by the users. About

5 or 6 optimisation problems a day were being solved by the machine automatically. Each time this was done a marked time saving was effected.

However, the *processor time* taken to attempt the problems soon showed itself to be a major drawback. With an average transaction processing time (without auto-rescheduling) of around 2 seconds, a delay in system response is very rarely noticeable. The time taken for the above algorithm to function to a certain extent depends on the reorganisation problem to be solved, but is mainly affected by the number of breaks in the segment (and thus the number of levels of recursion possible). As the number of breaks in a segment increased, the auto-rescheduling time increased significantly.

For example, in a 4-break off-peak segment, the process might take 5 or 6 seconds—a very acceptable delay to the user. Working on a 10-break peak segment it can easily take *20 minutes* of central processor time, which is obviously unacceptable.

In practice, a limit of one minute's processor time was put on transactions and any transaction, including one using auto-rescheduling, taking longer was rejected when the time-limit was up. This limit caused the success rate on peak segment problems to be seriously cut: somewhere around 10% of tractable cases were solved within the limit, whereas the success rate on off-peak segments was still over 70%.

One particular class of problem was found not to be solvable by this method of auto-rescheduling. This is the kind of situation shown in **Fig. 5** that can only be solved by 'swapping' two spots. Any process that involves moving one spot then another (as does the algorithm described above) cannot accomplish the swap which implies the 'simultaneous' moving of two spots. The swap situation is moderately common and undoubtedly accounts for a proportion of the 30% of problems not solved by the algorithm.

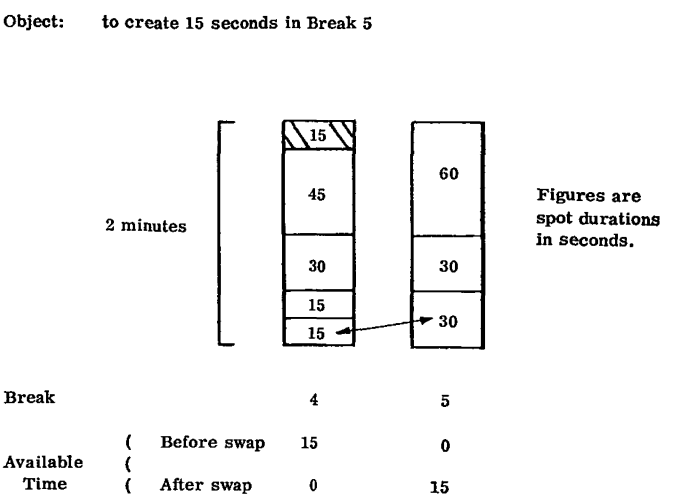


Fig. 5. The 'swap' situation

Conclusions

As a first automatic solution to the problem, this logic was a distinct success. It has saved a considerable amount of terminal time during its life. It was never conceived as a permanent and final solution: rather as a first experiment to gain experience and try out the techniques involved.

The optimisation problem is most common (and takes longest to solve 'manually') on peak segments and auto-rescheduling, because of its time-consuming nature, was of limited use in just that situation.

The other principal drawback of the program is its lack of flexibility in terms of the type of spot it expects to find and how it treats them. It has been rendered useless, at least without extensive alteration, by the introduction of a new class of spot called *fixed periods*. For a moderate surcharge an advertiser may specify during what period (i.e. range of breaks) he wants his spot transmitted. On the 'mobility' scale these spots are therefore between fixed spots (must stay in same break) and broad spots (may be transmitted in any break in segment) and clearly need special treatment in any organisation algorithm.

Special purpose automatic solution

Purpose

From time to time the Company has on hand large numbers of advertisement bookings of a fairly similar nature. For example, every six months a further six months' advertising time is thrown open to the advertisers and, usually within a few days, bookings for hundreds of spots have been received. This obviously puts a heavy and rather humdrum load on the real-time terminals, which could be occupied for days on end simply entering details of spots. The amount of terminal time required to feed in each spot, assuming the spot does not fall into some multiple pattern, is around 30 seconds. As the booking files are empty (at the beginning anyway) and few complications ensue, this operation is rather wasteful of real-time capabilities.

In this situation it is relatively easy to transfer the appropriate information to punched cards and insert the bookings by means of a batch program. Such a program has been written and has proved remarkably effective. The average time for each spot is about 1 second.

As the reservations file begins to fill up, the first optimisation problem that arises is the fixed spot block: an incoming fixed spot cannot be accepted because the break concerned is full of other spots, many of which are probably broad and could be moved elsewhere. This is Problem 1 as described in **The Problem.** Problem 2, the more general broad spot condition, does not usually occur until the files are much fuller.

A simple 'automatic moving' section was added to the booking logic of the batch program. Its specification is: to come into operation only when a fixed spot is being blocked by broad spots in the break concerned. It moves as many broads as necessary to other breaks in the segment, providing this is possible.

Technique

The programming logic and techniques employed in this special-purpose solution to one of the optimisation problems are very straightforward. The routine simply takes each broad spot in the break in turn and tries to find space for it elsewhere in the segment. However,

the logic has proved extremely effective in its rather special environment.

Latest automatic solution

Purpose

It was decided recently to attempt to find a radically different method of automatic solution of the time-optimisation problems for inclusion in the real-time spot booking transactions, in place of the obsolete first method. The deliberate intent was that this new method should solve more tractable cases than the old logic and yet be more flexible. At the same time, the programming techniques involved were to be straightforward as it was required to write and test the logic fairly quickly.

Heuristic approach

Rather than sitting down and theorising, it was decided to try a highly practical, empirical method of finding a solution to the problem, examining in depth the human mental processes involved when someone performs a segment reorganisation.

A model of an actual, fairly complex rescheduling problem was set up, using a table marked with break details and small pieces of card to represent spots. From this model it was possible to see the problem as a whole and to attempt solutions with ease.

Several individuals were invited to solve the problem and to try to describe their mental processes as they did so. It soon became apparent that at the root of all the manual solutions were a few basic techniques:

(*a*) 'Swapping': pairs of spots were found that, if swapped, would improve the situation in some way. It was essential to be able to *see* the segment as a whole to do this: allowing someone to see only half the segment at one time very effectively prevented them from using this method.

(*b*) Removing and holding in suspense certain 'troublesome' spots while the rest of the segment was reorganised using straight transfer or swapping methods. This would be done bearing in mind the 'suspended' spots and space would be left to reslot them back into the segment later.

It soon became apparent that most reorganisation difficulties were caused by long spots (say longer than 45 seconds), which were difficult to 'fit in', and the spots of common product-types.

It is difficult, if not impossible, to program efficiently a machine to attempt the 'swapping' method of reorganisation described above; it implies being able to see the whole problem-picture and the thought processes involved are subtle and of a high order. The idea of removing spots and holding them in suspense pending reslotting was considered worthy of further investigation and soon suggested a most interesting thought: should it not be possible, if the problem has a solution at all, to remove *all* the spots from the segment and then rebook them in the most efficient manner? Discovering the 'rules' to cause the most efficient rebooking was the only difficulty.

Taking this idea as a starting point, the author and several colleagues sat down round the segment model and had a protracted 'brainstorming' session to attempt to find these 'efficient booking rules'. Having formulated a set of rules, they were tried out on the model (and on others), the criterion for the best set being success in every case. The main difficulty came in reconciling the relative importance of different kinds of problem spot: fixed periods, common product-types, long spots, short 7 second spots. However, a set of reslotting rules eventually emerged that solved every problem encountered and it was decided that this procedure, essentially fast in computer terms (although tedious manually), should be adopted as the basis for a new automatic method.

Method

The technique which emerged from the simulation exercises has been named *priority reslotting*. Throughout the reslotting, as all breaks are in fact multiples of 30 seconds, an 'odd' 7 or 15 seconds is never left in a break unless absolutely necessary. The sequence of events is as follows:

1. All spots are extracted, details of the spot desired to be booked added to them and the whole lot sorted into a rebooking priority order. Certain 'classes' of spots are distinguished: fixed spots, fixed period spots, long broad spots (45 seconds or longer), normal broad spots. It is recognised that it is important to book spots of a frequent product-type early: if, say, there are 8 sweet advertisements in an 8-break segment, you cannot afford to let one of the breaks become full without including a sweet advertisement. Within each spot class, the spots are therefore sorted in order of product-type frequency.

2. Fixed spots are booked into their specified breaks—there is no room for selection. Then:

3. Fixed period spots are slotted. All breaks within the specified period for a spot are examined in relation to the spot and listed under six headings:

(*a*) spot will fit with no complications;
(*b*) spot will fit but will exactly fill break;
(*c*) spot will fit but leave an odd 15 seconds of time (i.e. 45 seconds, etc.);
(*d*) spot will fit but leave an odd 8 seconds of time;
(*e*) spot clashes with a spot already in the break;
(*f*) no time for spot in break.

If all the breaks are in categories (*e*) and (*f*), then it is impossible to book the spot and the reorganisation attempt has failed. If any breaks are in category (*a*), the one with most available time is chosen. If there are no breaks in (*a*), any in (*b*) are examined to check the advisability of filling a break at this stage. Category (*c*) is examined if no suitable break in (*a*) or (*b*), and so on.

Then:

4. Long broad spots (45 seconds or longer) are rebooked using exactly the same procedure as for fixed periods but considering the whole segment as the period.

5. Each break in turn, starting with the first one, is then filled up using each time the next most suitable spot from the priority-ordered remaining broad spots.

Any 'odd' available times are made into multiples of 30 seconds as soon as possible. If any time has to be left in a break (e.g. 8 seconds left in a break and there are no 7 second spots available) a factor known as 'Leeway Time' is examined. This is calculated at the beginning of the reslotting process and is the amount of time you can 'afford' to leave unoccupied. If the Leeway Time is nil, then all breaks must be filled.

6a. If the attempt at reorganisation fails to reslot all spots into the segment, an appropriate response will be sent to the terminal.

6b. If Priority Reslotting has been successful, the actual drum-held spot records are set to match the reorganised disposition worked out in memory. The spot is then booked.

Current state

At the time of writing, the reslotting algorithm has been proved highly workable by simulation. Plans are being made for its implementation.

Acknowledgement

The author would like to thank Thames Television Limited for permission to present this paper.

Reference

TATHAM, L. (1967). Computers in Television, *Data and Control Systems*, May 1967.