Junction optimisation technique

By M. J. Savage*

This paper describes new optimisation procedures for specialised types of scheduling problem within the broad framework of transportation network theory. Rigorous proofs of the techniques are not presented, but their validity has been verified empirically by application to railway junctions. Possible consequences in automatic train control are discussed.

(Received April 1969)

The basic scheduling problem

In scheduling problems a set of N events $E_1, E_2, \ldots, E_N$ have to be arranged in an optimum sequence subject to certain constraints. In the type of problem considered here, it is assumed that the N events comprise n groups of events, where group i contains m_i similar events $(i = 1, 2, \ldots, n)$. We shall describe an event in group i as an event 'of type i'.

We have

$$1 \leqslant n \leqslant N \tag{1}$$

$$1 \leqslant m_i \leqslant N \quad (i = 1, 2, \ldots, n) \tag{2}$$

$$\sum_{i=1}^{n} m_i = N \tag{3}$$

The constraints on the schedule of events are provided by an $(n \times n)$ 'Event Matrix'

$$A = (a_{ij}) \tag{4}$$

where $a_{ij} \geqslant 0$ is the minimum time interval which must elapse between an ordered pair of events of types i and j *whether or not there are intermediate events.* (5)

The last requirement proves to be crucial in railway applications of the kind considered below, and marks the main point of divergence from standard transportation network theory. In its absence, the present problem reduces to a variant of the Travelling Salesman Problem (Reference 1) by regarding events as towns and equating time with distance.

We are tacitly assuming in (4) and (5) above that real events have zero duration. In an application when this is not the case, the theory can still be applied by defining theoretical events as, say, the start of corresponding real events, provided that a constraint of the form (5) holds for the starting times. The basic scheduling problem can now be stated in the following form:

'Find a schedule for the events

$$E_1, E_2, \ldots, E_N$$

which minimises the time interval, θ, between the first and last events, where the relative timings of all pairs of events are subject to the constraint (5) imposed by the Event Matrix.'

A 'schedule' in the above sense is defined as an N-vector

$$T = [\tau(1), \tau(2), \ldots, \tau(N)]$$

where $\tau(I)$ is the time of occurrence of event E_I $(I = 1, 2, \ldots, N)$; together with a permutation ρ of the integers 1 through N which defines the sequence of events as follows:

$$\rho(I) = J \Rightarrow E_J \text{ is the } I\text{th event to occur.}$$

We now have

$$\theta = \tau(\rho(N)) - \tau(\rho(1)) \tag{6}$$

It should be noted that the schedule T does not define ρ uniquely since we can have

$$\tau(I) = \tau(J) \qquad I \neq J$$

because some elements of A may be zero. On the other hand ρ clearly does not define T uniquely, since, for example, the time at which the final event occurs (namely $\tau(\rho(N))$ can be increased by an arbitrary amount without conflicting with (5). However, for any given ρ, it is possible to determine a schedule T which minimises θ, as described immediately below.

The JOT algorithm

Suppose that event E_I is of type $\mu(I)$ $(I = 1, 2, \ldots, N)$ consistent with the grouping specified above at equations (1) through (3). Define σ as the product of the mappings ρ and μ, so that

$$\sigma(I) = \mu(\rho(I)) \quad (I = 1, 2, \ldots, N) \tag{7}$$

Thus $\sigma(I)$ is the type of the event in position I in the sequence. For a given ρ (and hence a given σ) a minimising schedule T can be determined as follows:

$$\text{Put} \qquad \tau(\rho(1)) = 0 \tag{8}$$

$$\text{and} \qquad \tau(\rho(I)) = \min_K \left(\tau(\rho(K)) + a_{\sigma(K), \sigma(I)}\right) \tag{9}$$

$$(I = 2, 3, \ldots, N)$$

where the minimisation is over the range

$$\max(L, M) \leqslant K \leqslant I - 1$$

where L is the largest integer, J, which satisfies the conditions

$$\left.\begin{array}{r} \sigma(J) = \sigma(I) \\ \text{and} \quad 1 \leqslant J \leqslant I - 1 \end{array}\right\} \tag{10}$$

or $L = 1$ if no such integers J exist;

and where M is the largest integer R which satisfies the conditions:

* *The Railway Technical Centre, Wilmorton, Derby*

$$\left.\begin{array}{ll} & \tau(\rho(R)) + \max_{i,j}(a_{ij}) \leqslant \tau(\rho(I-1)) \\ \text{and} & 1 \leqslant R \leqslant I-2 \end{array}\right\} \quad (11)$$

or $M = 1$ if no such integers R exist.

The integers L and M, which impose limits on the range of K in equation (9), are introduced to limit unnecessary computation. When testing for pairs of events whose relative timings conflict with (5), L prevents duplication of tests, and M avoids the testing of pairs of events whose separation exceeds the largest element of A.

The problem has thus been reduced to the determination of an optimising permutation ρ. However it will be sufficient to determine the mapping σ, since although this does not define ρ uniquely, the nature of the constraints imposed by A are clearly such that any one of the

$$\prod_{i=1}^{n} (m_i!)$$

permutations ρ corresponding to a given σ (obtained by permuting events of each type amongst themselves) will be optimum; furthermore, given σ, the derivation of all corresponding permutations ρ, is trivial (using equation (7)).

Before describing the determination of σ, it is worth noting that for a 'typical' problem, with, say,

$$\left.\begin{array}{l} N = 28 \\ n = 7 \\ m_i = i\,(i = 1, 2, \ldots, 7) \end{array}\right\} \quad (12)$$

the number of distinct mappings σ is about $2{\cdot}4 \times 10^{18}$. Hence present digital computers are far too slow to guarantee absolute optimality by trial and error methods. The JOT algorithm produces a near-optimum, overcoming the difficulty of size by splitting the problem into manageable portions, using a technique which exploits the nature of the Event Matrix A. In railway junction applications, as described below, the diagonal elements of A are in general greater than off-diagonal elements. Furthermore, the diagonal elements are always nonzero, whereas many off-diagonal elements are zero, so optimum or near-optimum schedules will include few instances where consecutive events are of the same type.

We now return to the determination of σ, and without loss of generality, suppose that

$$m_1 \geqslant m_2 \geqslant m_3 \geqslant \ldots \geqslant m_n$$

(In practice computationally this means renumbering event types using a simple sort subroutine.)

Consider the N- vector:

$$S = [\sigma(1), \sigma(2), \ldots \sigma(N)]$$

We now split S into m_1 subvectors. The m_1 subvectors comprise

Δm_1 subvectors of size * 1
Δm_2 subvectors of size 2
. . . .
. . . .
. . . .
and Δm_n subvectors of size n

where

$$\left.\begin{array}{l} \Delta m_i = m_i - m_{i+1}\,(i = 1, 2, \ldots, n-1) \\ \Delta m_n = m_n \end{array}\right\} \quad (13)$$

so that

$$\sum_{i=1}^{n} \Delta m_i = m_1$$

A subvector of size* i will contain as its elements the integers 1 through i in some sequence, and we regard it as a permutation of event-types 1 through i.

Qualitatively; the aim of this synthesis of σ is to mix event-types thoroughly, yet to maintain as far as possible a repeated pattern of event-types to ease the subsequent task of optimising individual permutations. The sequence of the m_1 permutations which constitute S can be defined by a 'permutation vector' of size m_1:

$$S_1 = [\sigma_1(1), \ldots \sigma_1(m_1)]$$

where $\sigma_1(j)$ is the size of the jth permutation in the sequence. The problem has now been reduced to the determination of S_1, which is itself a sequencing problem of a similar type, but with 'sizes of permutations' replacing 'event-types'.

The determination of S_1 can thus be tackled by the above method, and itself reduced to the determination of a 'permutation of permutations' vector S_2 of size

$$\max_{1 \leqslant i \leqslant n} (\Delta m_i)$$

The above process is repeated, by continued differencing and sorting, until after at most n steps we reach a scalar

$$S_k = [\sigma_k(1)]$$

We now introduce the notation

P^0 meaning 'event-type'
P meaning 'permutation' (of event-types)
P^2 meaning 'permutation of permutations' (of event-types)
etc.

Then S will be constructed to consist of one P^k of size $\sigma_k(1) \leqslant n$, the sequence of P^{k-1}s which constitute the P^k being determined by the natural sequence arising out of the sorting process at stage $k - 1$. Hence the *sequence* of elements in S_{k-1} is determined (the *constitution* of these elements was already known from the differencing and sorting process) and by continued back substitution in this way we obtain the sequence of elements in S_{k-2}, S_{k-3} . . . and eventually obtain the permutation vector S_1. A numerical example is shown schematically in **Fig. 1**. (In this diagram the above notation is extended straight-forwardly. Thus 13 P^0 1 should be read as '13 type-1-events', and 5 P 4 as '5 size-4-permutations of event-types' and so on.)

It remains to determine S (and hence σ) from S_1.

The Event Matrix A is used to optimise individual permutations (whose sizes are the elements of S_1) in the sequence specified by S_1. A permutation of given size, is optimised once only, then used throughout S wherever it occurs.

* The term 'size' is used in the above context throughout this paper, in preference to the term 'order' which has the alternative meaning 'sequence'.

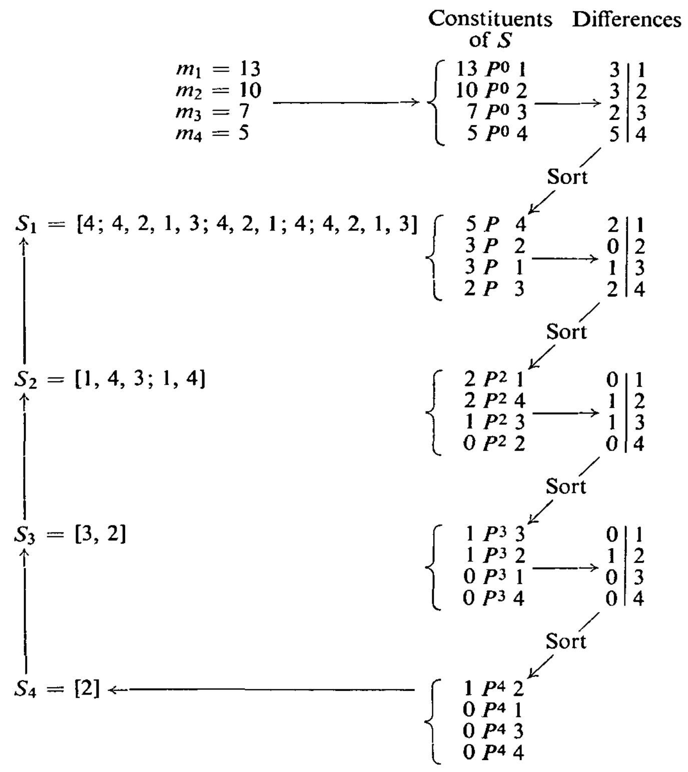

Fig. 1. $N = 35, n = 4$

The optimisation of an individual permutation makes use of the timing technique of equations (8) through (11) applied to appropriate subsequences of event-types in S. These subsequences include all event-types in previously-optimised permutations adjacent to the permutation being optimised.

All $i!$ permutations of size i are considered, and that for which the corresponding subsequence contributes the least time to θ, is selected as the optimum.

The ratio of the computer time required for this process to that needed for trial of all possible vectors S is approximately α/β where

$$\alpha = \sum_{i=1}^{n} i!$$

and

$$\beta = \frac{N!}{\prod_{i=1}^{n} (m_i!)}$$

In the example of equations (12) we have

$$\alpha = 5913$$
$$\beta \doteqdot 2{\cdot}4 \times 10^{18}$$
$$\alpha/\beta \doteqdot 2{\cdot}4 \times 10^{-15}$$

However, for problems with large n, computer-time limitations may still dictate sub-optimisation of the larger permutations. For a permutation of size i, this can be achieved by dividing the event-types into two groups, the first containing event-types 1 through i' and the second event types $(i' + 1)$ through i, where

$$i' = \left[\frac{i}{2}\right]$$

The event-types in each group are then permuted separately, but all ordered pairs of permutations are considered, so that the number of trials is reduced from $i!$ to $2.i'!(i - i')!$

Application of JOT to railway junctions

Fig. 2 shows a simple railway junction with 4 routes. For the purposes of this illustration trains are categorised according to their route, although in practice further categories could be introduced to cater for different classes of train.

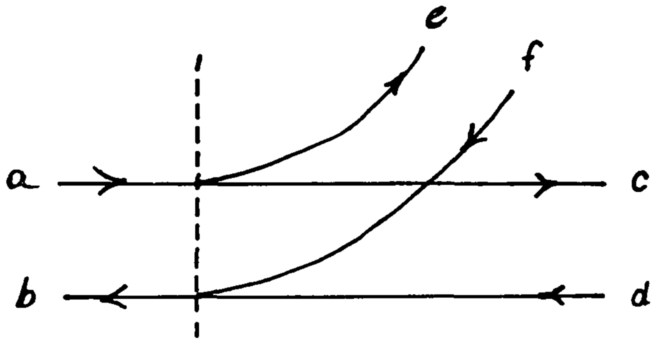

Fig. 2

	1	2	3	4
1	3	0	2	0
2	0	3	1·5	2
3	2	1·5	2·5	0
4	0	2	0	2·5

Fig. 3. **Event Matrix (times in minutes);** $n = 4$

Trains of type 1 run from a to e
Trains of type 2 run from f to b
Trains of type 3 run from a to c
Trains of type 4 run from d to b

We define an 'event' as the passage of the rear of a train past a given reference line (shown dotted in **Fig. 2**). The events $E_1, E_2 \ldots E_N$ represent a set of N trains—a given quantity of traffic which has to use the junction. The Event Matrix A represents the operating and signalling rules applicable at the junction, and is shown in **Fig. 3.** The diagonal elements are headways between trains taking the same route. The validity of this type of representation of railway junction operations is demonstrated in Reference 2. The aim of the optimisation is to maximise traffic throughput at the junction.

The JOT algorithm has been programmed for the B.R. Research Department's computer, and applied to railway junctions both real and ficticious. For small theoretical cases where the absolute optimum was determinable manually by trial and error the algorithm always gave a traffic throughput within 10% of the optimum. On a busy real junction on the Southern Region of B.R. it produced, in 2½ minutes computer time, a throughput for the evening rush hour which was 3% better than that attained by the existing working

TRAIN TYPE	NUMBER OF THIS TYPE
1	2
2	10
3	12
4	16
5	4
6	3
7	7
8	7
9	0
10	4
11	3
12	0
13	10
14	9
15	11
TOTAL TRAINS	98

Fig. 4. JOT phase one. Junction B. 1700–1800 hrs.

timetable which had been produced manually by experience over the years.

Part of the computer output for this case is shown in **Figs. 4, 5** and **6**.

Future developments

The JOT algorithm is a satisfactory tool for planning the sequence of operations at busy railway junctions. It is sufficiently fast for potential incorporation in software for real-time control of trains by computer in congested areas, but, in the form described above, is not suitable for on-line control because it fails to cater for the additional constraints imposed by a real-time situation.

However, research has been in progress for some time to extend the basic model to deal with all known constraints, and in particular those associated with the presence of a large main line terminus such as passenger requirements, and platform capacity. This has involved the writing of a large software package combining JOT scheduling techniques with new algorithms for optimum platform selection, and accepting as data the expected arrival times of a set of trains which provides an implied constraint on the sequence of movements. Meanwhile the associated computer hardware requirements are being investigated for an experimental pilot scheme based on the above software.

PERMS	TRAIN TYPES	TIME
1	4	0
1	4	7
8	3	7
	15	7
	8	9
	2	13
	4	14
	13	14
	7	20
	14	21
1	4	21
8	3	26
	15	29
	8	29
	2	33
	4	33
	13	35
	7	40
	14	42
6	4	42
	3	46
	14	50
	2	54
	13	56
	15	62
13	4	62
	3	62
	8	64
	2	68
	13	69
	1	71
	15	75
	7	75
	5	77
	10	81
	11	87
	14	88
	6	88
1	4	88
8	3	94
	15	96
	8	96
	2	100
	4	100
	13	102
	7	107
	14	109
6	4	109
	3	113
	14	117
	2	121
	13	123
	15	129
13	4	129
	3	129
	8	131
	2	135
	13	136
	1	138
	15	142
	7	142
	5	144
	10	148
	11	154
	14	155
	6	155
2	3	161
	4	161
3	4	168
	15	168
	3	168
10	8	170
	3	174
	4	175
	13	175
	2	180
	15	181
	10	187
	14	194
	7	194
	5	194
12	11	194
	8	198
	4	201
	7	201
	15	202
	3	207
	13	208
	5	208
	2	214
	10	214
	6	220
	14	221
5	4	221
	3	226
	13	227
	2	232
	15	233

Fig. 6. Optimised schedule

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	6	3	3	6	6	4	4	2	2	0	2	2	0	2	0
2	3	6	6	0	6	6	7	2	2	0	2	2	0	2	0
3	3	6	6	0	0	7	7	2	2	0	2	2	0	2	0
4	6	0	0	7	7	0	0	0	0	0	0	0	0	0	0
5	6	6	0	7	7	7	0	0	0	0	0	0	0	0	0
6	3	6	6	0	7	7	7	0	0	0	0	0	0	0	0
7	3	6	6	0	0	7	7	0	0	0	0	0	0	0	0
8	4	4	4	0	0	0	0	8	8	8	4	5	5	0	0
9	4	4	4	0	0	0	0	8	8	8	5	5	5	6	6
10	0	0	0	0	0	0	0	8	8	8	6	6	6	7	6
11	4	4	4	0	0	0	0	4	4	6	8	8	8	0	0
12	4	4	4	0	0	0	0	4	6	6	8	8	8	6	6
13	0	0	0	0	0	0	0	4	6	6	8	8	8	7	6
14	4	4	4	0	0	0	0	0	6	6	0	6	6	8	8
15	0	0	0	0	0	0	0	0	6	6	0	6	6	8	8

Fig. 5. Event Matrix Times in $\frac{1}{4}$ mins.

This initial system will be called TRAC—'Train Regulation Advisory Control', and if successful, may well form the basis for automatic train control systems in conurbations. It is hoped to publish further details of this work in the future.

Acknowledgements

I am indebted to Mr. P. J. Coates, formerly Assistant Director of Electrical Research, British Railways Board, for introducing me to the application of matrices to railway junctions and for his considerable encouragement throughout this research; and also to Mr. S. F. Smith, the Director of Research for permission to publish this paper.

References

1. Berge, Claude, and Ghouila-Houri, A. (1965). *Programming, Games and Transportation Networks*, Methuen (see Chapter 10).
2. Potthoff, Gerhart (1961). 'Verkehrsströmungslehre', Vol. 1, Transpress.

Book Reviews

Continued from page 267

Easy proofs are included and the difficult proofs quite properly are given only by reference. The 'why' as well as the 'how' is fully documented, with comments on where the theory is far from complete. A notable feature is the treatment of relative as well as absolute minimax error. The author moreover nicely reveals both the parallel and the divergent features of polynomial and rational approximations, and puts continued fractions and the Padé table in proper perspective. Above all he reveals a degree of 'numerical sense' which these days is almost unfashionable!

Finally, one observes with awe and amazement the extreme care with which the expert obtains his approximation and bounds the error, and compares this, more in sorrow than in anger, with the attitude of the average computer-user, who often ruins the beautifully accurate approximation in the first few error-laden arithmetic operations of his own program.

L. Fox (Oxford)

Computer Approximations, by J. F. Hart, E. W. Cheney, C. L. Lawson, H. J. Maehly, C. K. Mesztenyi, J. R. Rice, H. C. Thacher Jr., C. Witzgall, 1968; xii, 344 pages. (John Wiley & Sons, Inc. £8 4s. 0d.)

This book is a sequel to and extension of the work *Approximations for Digital Computers* by Cecil Hastings, Jr., which appeared in 1955. This book had a noticeable impact amongst computer users when it appeared—it was more a work of art than mathematics, yet valuable and attractive. The present work is an extension, with a substantial introduction on mathematical methods of derivation, and giving a great number of numerical approximations for a variety of functions, yet one sighs for a Hastings to present it to the reader—the version now reviewed is more comprehensive but seems relatively dull and uninspiring in its presentation and layout.

To indicate the scope of the text, it may be useful to list chapter headings, and length in pages:

1. The Design of a Function Subroutine	9
2. General Methods of Computing Functions	32
3. Least Maximum Approximations	16
4. The Choice and Application of Approximations	24
5. Description and Use of the Tables	7
6. Function Notes	100
7. Tables of Coefficients	152

Chapter 2 includes techniques for acceleration of convergence and error-elimination as well as familiar computation techniques.

Chapter 6 contains, interspersed in the text, and not easy to find rapidly, lists of approximations to various functions given in Chapter 7.

There are also Appendices giving conversion algorithms (programs in ALGOL), a Bibliography of approximations, and a list of decimal and octal constants.

On p. 87 the authors say 'Because of the cumbersomeness of some expressions and as a convenience in computer work, each function has been assigned a code name'. Because of this, and the condensed codification of individual approximations, the list of approximations is unnecessarily cumbersome to use. The authors seem to have overlooked the fact that a book is produced for the convenience of the average user, not for that of the authors. A few more pages spent on an intelligible and attractive presentation and layout of results would have produced a less forbidding table, much more readily usable.

It would also have helped if the illustrative examples on p. 87 had not referred to Chapter 6 where Chapter 7 is meant, and if the range $[\frac{1}{2}, 1]$ had not appeared as $\left[\frac{1}{\sqrt{4}}, 1\right]$ on p. 95.

It must be said however, that the work contains a great number of useful polynomial and rational approximations, of various precisions up to about 25 figures, for $\sqrt{x}$, $\sqrt[3]{x}$; 2^x, 10^x, e^x; $\log_{10} x$, $\log_e x$; $\sin \alpha\pi x$, $\cos \alpha\pi x$, $\tan \alpha\pi x$; $\arcsin x$, $\arctan x$; $\Gamma(x)$, $\Phi(x)$; erfc x; $J_0(x)$, $J_1(x)/x$, $\bar{Y}_0(x)$, $\bar{Y}_1(x)/x$, $P_0(x)$, $P_1(x)$, $Q_0(x)/x$, $Q_1(x)/x$; $E(1-x)$, $K(1-x)$.

A useful book, not quite obvious in use.

J. C. P. Miller (Cambridge)