ANALYSIS OF FUTURE EVENT SET ALGORITHMS FOR DISCRETE EVENT SIMULATION

Technical Report #CS80002-R

by

William M. McCormack
Department of Computer Science
Virginia Polytechnic and State University
Blacksburg, Virginia 24061

and

Robert G. Sargent
Department of Industrial Engineering
and Operations Research
Syracuse University

April 1980

This report is identical to working paper 80-005, Department of Industrial Engineering and Operations Research, Syracuse University.

ABSTRACT

This work reports on new analytical and empirical results on the performance of algorithms for handling the future event set in discrete event simulation. These results provide a clear insight to the factors affecting algorithm performance; evaluate the the "hold" model, often used to study future event set algorithms; and determine the best algorithm(s) to use.

ANALYSIS OF FUTURE EVENT SET ALGORITHMS FOR DISCRETE EVENT SIMULATION

1. Introduction

In discrete event simulation, the time flow mechanism (TFM) causes the events in the simulation to occur at the proper time and in the proper sequence [8,10]. Implementation of a Variable Time Increment TFM, the most common, involves maintaining a set of records, one record for each scheduled future event. The usual operations on this set are adding a record when an event is scheduled and removing the record with minimum time of event occurrence when the clock is advanced to the time of the next event.

The data structure used to maintain this set can be crucial to the execution time of a simulation and recently many articles have reported major improvements in execution time by implementing an algorithm other than the commonly used linear list [5,7,12, 17,19,24, 37,38,39,42]. Generally these papers have either shown the improvement for a specific simulation by using a particular algorithm instead of a linear list or else algorithms have been compared using a model of the simulation TFM.

This article presents both analytical and empirical results concerning the behavior of the TFM future event set

for discrete event simulation. The results provide insight into the algorithms' performance, permit evaluation of the (hold) model of the TFM often used, and determine the best algorithm(s) to use.

The remainder of this paper is divided into five sections. In the next section, the hold model and two distributions associated with its use are described. In sections 3 and 4, analytical results are derived for the linear list and then these results applied to other algorithms. Section 5 reports the results of experiments conducted and explains them. The last section contains the conclusions and recommendations.

2. The Hold Model

2.1 Model Definition

The two basic operations performed on the future event set are insertion of new events and deletion of the next event. Most of the research [5,7,9,12,13,17,19,39,40] performed to date has used a model which combines these two operations based on the Simula hold operation [6] and will be referred to as the hold model. A hold operation determines and removes the event record with minimum time value from the future event set; increases the time value of that record by T, where T is a random variate distributed according to some distribution F(t); and inserts that record back into the future event set [39].

The hold model is essentially the following:

1. [INITIALIZATION] Insert N event records into the future event set structure with the time value of each generated from the distribution F(t).

2. [TRANSIENT] Execute N1 hold operations to permit the model to reach steady state. In analytical work, it is assumed that N1 is infinity. In practice, N1 is usually some small multiple of the size of the future event set [12,17,39].

3. [STEADY STATE] Execute N2 hold operations and measure the performance of the future event set structure to do this. The measure may be the total time to execute the N2 operations or counts of key steps done in the insertion or deletion routines, for example the total number of comparisons needed to do inserts.

This model has two parameters: N, the number of records in the future event set; and F, the distribution used to determine how long an inserted record will remain in the future event set. F will be referred to as the scheduling distribution.

This model may be viewed as a special case of the classical machine repairman model with N machines and a single repairman [18]. The scheduling distribution, F, corresponds to the distribution describing the time between failures of a machine, while the time to repair a machine is zero.

2.2 Future Event Set Distribution

A second distribution, which is important when studying the future event set, is the future event set distribution. Basically, this is a measure of where the time values of those records in the future event set are situated in time between the current time and infinity. This distribution can be obtained using results in renewal theory and in

steady state is independent of the current time value and only a function of the distribution F [2]. The future event set distribution will be denoted by G,

$$G(x) = P(\text{remaining life} \leq x)$$

$$G(x) = (1/u) \int_0^x [1 - F(w)]\, dw \qquad x \geq 0$$
$$= 0 \qquad \text{otherwise}$$

where u is E[W], the mean time an event is in the future event set.

If all of the events are independent and identically distributed (as in the hold model), G(x) is the fraction of event records which will occur in the next "x" time units. For example, if F is the uniform distribution over one time unit, (0,1), then

$$G(x) = \begin{cases} 0 & x \leq 0 \\ 2*x - x^2 & 0 \leq x \leq 1 \\ 1 & x \geq 1 \end{cases}$$

Considering F and G, while new events are equally likely inserted anywhere between 0 and 1, most (75%) of the event records in the future event set will occur in less than one-half time unit (G(0.5) = 0.75).

2.3 Limitations of the Hold Model

In using the hold model, care must be taken to remember it is just a model and differs significantly from real

simulations in many ways. These differences include that the size of the future event set does not change, all the event records are independent of each other, and the same scheduling distribution is used for all events. The effect of these assumptions has never been tested. In at least one case [7], much of the overhead of the algorithm was eliminated by taking advantage of the fact that the number in the future event set did not change.

3. Linear List

3.1 Linear List and the Hold Model

Knowledge of the event set distribution permits calculation of the amount of work required to insert new records when the future event set is implemented as a doubly-linked linear list [22]. For example, for the uniform distribution above, a new event would likely be inserted after most of the other event records in the list. Our analysis of the linear list will be the most extensive as it is the most commonly used, is simple to implement and study, uses minimal storage, and its analysis will aid in understanding the behavior of the other data structures.

Let %F (%B) denote the percentage of the linear list passed when an event is inserted from the front (back) of a linear list. The following result was first obtained by Vaucher [40] and later by Englebrecht-Wiggans and Maxwell [9]:

$$\%F = \int_0^\infty G(x)\ dF(x) \quad * \quad 100\%$$

$$\%B = \int_0^\infty [1 - G(x)]\ dF(x) \quad * \quad 100\%$$

For the example given earlier, %F is 66.7% and %B is 33.3% so insertions are more efficient from the back for this scheduling distribution. For the hyperexponential distribution [30], the front is more efficient while for the

exponential the direction makes no difference due to the memoryless property of that distribution.

An important question is what are the characteristics of those distributions for which starting the insertion scan from, for example, the front of the list, in order to apply these results to actual simulations. The following theorem solves this problem for two large families of distributions.

As defined in Barlow and Proschan [3], a distribution F is New Better(Worse) Than Used in Expectation [NBUE(NWUE)] if

(a) F has finite (finite or infinite) mean;

(b) $\int_t^\infty [1 - F(x)]\, dx \leq (\geq)\ u * \bar{F}(t)$ for $t \geq 0$

where $\bar{F}(x) = 1 - F(x)$ and u is E[X].

This classification of distributions covers most of those distributions for which random variate generators are provided in most simulation languages. For example, NBUE includes the uniform distribution, Normal, and Erlang; while the hyperexponential, a mixture of exponentials, and certain Gamma distributions are all NWUE. Some distributions fall outside this classification, e.g. Beta(p,1), where $0<p<1$. Note also the categories are not disjoint, as the

exponential is a member of both.

This leads us to the following theorem:

THEOREM 1.

For the hold model, if the scheduling distribution, F, is NBUE (NWUE), then %F $\geq$ ($\leq$) 50% so the scan should start from the back (front) of the linked linear list.

PROOF.

As a first step, the definition of NBUE (NWUE) will be restated in terms of the scheduling and future event set distributions.

$$\int_t^\infty \bar{F}(x)\,dx = \int_0^\infty \bar{F}(x)\,dx - \int_0^t \bar{F}(x)\,dx$$

$$= u - \int_0^t \bar{F}(x)\,dx$$

$$= u * [1 - (1/u)\int_0^t \bar{F}(x)\,dx\,]$$

$$= u * [1 - G(t)]$$

$$= u * \bar{G}(t).$$

Thus, F is NBUE (NWUE) if and only if (for $t \geq 0$):

$$\int_t^\infty \bar{F}(x)\,dx \leq (\geq)\ u * \bar{F}(t) \qquad \text{or}$$

$$\bar{G}(t) \leq (\geq) \bar{F}(t) \qquad \text{or}$$

$$G(t) \geq (\leq) F(t).$$

The proof of the theorem is now straightforward:

$$\%F = \int_0^\infty G(x)\, dF(x) * 100\%.$$

Since F is NBUE(NWUE) and using the alternate definition just developed:

$$\%F \geq (\leq) \int_0^\infty F(x)\, dF(x) * 100\%$$

$$\geq (\leq) \int_0^1 u\, du * 100\%$$

$$\geq (\leq) \quad 0.50 * 100\% = 50\%.$$

This theorem permits concentration on characteristics of these two families to understand future event set behavior. If F is NWUE, then the probability is high that values generated are small but there is also a small probability that the values are extremely large. It can be shown that if F is NBUE (NWUE) then the coefficient of variation (standard deviation / mean) is greater (less) than or equal to one [3]. The coefficient of variation is not

sufficient to insure %F < (>) %B but it is often a good indicator. In addition, the coefficient of variation will play an important role in the remainder of this article.

3.2 Linear List and the Interaction Hold Model

Theorem 1 determines the most efficient end of a linear list from which to insert events for many distributions and provides some understanding of the distributions; however, it is of little application to the real world. As stated earlier, the hold model differs in many ways from actual simulations; one major simplification being that all event records have the same scheduling distribution. In real simulations, however, it is usually the case that the events have not all been scheduled by the same distribution. For example, in even a simple model like a single server queueing system, there are the arrival and service distributions. Thus, a model more representative of simulation would have event records scheduled by different distributions.

In the machine repairman model interpretation of the hold model given earlier, instead of all N machines having failure distribution F, n_1 would have failure distribution F_1, n_2 failure distribution F_2, etc. Since only one linear list is used, when scanning to insert the failure time of one machine it is most likely some event records scanned will represent machines with other failure distributions.

In this sense, the distributions are interacting in the event set; hence, this extension will be referred to as the Interaction Hold Model.

The interaction hold model is defined as having N independent event records of which n_i ($n_i > 0$ and $\sum_{i=1}^{I} n_i = N$) event records use scheduling distribution $F_i(x)$. $\%F_i$ will denote the percentage of a list passed in which only distribution F_i is used (e.g. if F_i were exponential, then $\%F_i = 50\%$) and $\%F_{int}$ will denote the average percentage of the list passed for this model.

That brings us to the following theorem:

THEOREM 2.

In steady state, for the interaction hold model with I distributions and $\%F_i$, as defined above, the average fraction of the future event set that will be scanned when starting from the front of a linear list, $\%F_{int}$, is bounded:

$$\%F_{int} \leq \%F_{max} = \max\ (\%F_1, \%F_2, \ \ldots\ , \%F_I).$$

The proof is quite long and is given in Appendix A.

The theorem simply provides a worst case upper bound but says nothing about a lower bound; that is, $\%F_{int}$ need not be greater than the minimum of $\%F_1$, $\%F_2$, ... , $\%F_I$. The bound, although worst case, does show that $\%F_{int}$ is less than or equal to 50% if all of the scheduling distributions are NWUE, are exponential, or for each $\%F_i \leq 50\%$.

The theorem does not, however, provide any information about $\%F_{int}$ if any or all of the scheduling distributions have $\%F_i$ greater than 50%; therefore, %F was evaluated for a number of cases (see Table 1). It can be seen from the table that $\%F_{int}$ is often much less that the bound provided by Theorem 2 and, in fact, can be less than 50% even if for each scheduling distribution $\%F_i$ is larger than 50%. Similar behavior is exhibited when more than two distributions are used.

The reason for the above is that $\%F_{int}$ is more heavily influenced by the mean of each distribution and the number of records of each type than the distribution itself. For example, in cases 3 and 9, there are many event records from a scheduling distribution with a large mean but each record with a small mean is inserted one hundred times more often. Thus, most insertions will go near the front of the list while most of the records will be nearer the back of the list. Note that this is similar to the use of an NWUE distribution for the simple hold model. The other situation, exemplified in cases 2 and 8, has most records from a distribution with a small mean; thus, the other records play very little role and $\%F_{int}$ is close to $\%F_2$.

Thus, Theorem 2 and Table 1 indicate when the event list of a simulation contains event records scheduled by greater than one distribution, that the insertion scan should start from the front of a linear list. This is in

Table 1. Effect of Interaction of Some Specific Scheduling Distributions on Linear List Insertion Scanning

	Scheduling distrib. 1	Scheduling distrib. 2	n_2	$\%F_1$	$\%F_2$	$\%F_{int}$
1.	exponential mean(u)=10	exponential mean(u)=1	10	50%	50%	32.4%
2.	expon,u=100	expon,u=1	100	50%	50%	45.5%
3.	expon,u=1	expon,u=100	100	50%	50%	9.5%
4.	expon,u=10	uniform(0,1)	10	50%	67%	33.5%
5.	expon,u=100	uniform(0,1)	100	50%	67%	45.5%
6.	expon,u=1	uniform(0,100)	100	50%	67%	10.9%
7.	uniform(0,10)	uniform(0,1)	10	67%	67%	40.8%
8.	uniform(0,100)	uniform(0,1)	100	67%	67%	60.5%
9.	uniform(0,1)	uniform(0,100)	100	67%	67%	12.1%
10.	constant 10	constant 1	10	100%	100%	56.9%
11.	constant 50	constant 1	10	100%	100%	49.4%
12.	constant 100	constant 1	100	100%	100%	90.9%
13.	constant 1	constant 10	50	100%	100%	49.2%
14.	constant 1	constant 100	100	100%	100%	17.4%

"n_i" is the number of nodes using distribution "i". $n_1 = 10$.

direct contrast to what the major simulation languages currently do. Furthermore, it suggests that NWUE distributions (for which a scan from the front is more efficient) be given much greater consideration when using the simple hold model in order to more realistically model actual simulations.

Results similar to those based on Theorem 2 may be obtained [27] for a scheduling distribution which is a mixture [3] of other distributions. These results are also important in their relation to previous research into preservation of distribution properties by mixtures [3].

4. Other Data Structures

4.1 Multiple Lists

SIMSCRIPT II.5 maintains a separate list for each event and process in the simulation [20,35] which decides the number of lists there will be and to which list an event is inserted. The results of the previous section can be used to determine how insertions should be done for each list in a simulation in SIMSCRIPT. For those lists associated with events, it is likely that the same scheduling distribution is used for all insertions to that list so the results of section 3.1 are applicable. For a list associated with a process, an interaction of distributions is probable so the results of section 3.2 are significant.

4.2 Single Pointer Method

Laughlin [24], Pritsker [34], and Davey and Vaucher [7] have studied the effect of adding a pointer to a "middle" record of a linear list to improve the efficiency of the insertion operation. The most extensive analysis was done by Davey and Vaucher who concluded the pointer should be to the record with 50% of the future event set to each side of it in a linear list and that the scan should start from the front (back) of the list if the insertion would be before (after) the median record.

This same choice was arrived at in a thorough analysis using a different approach and the Interaction Hold Model [27]. Theorem 1 provides an insight to the choice of the scanning directions. If the scheduling distribution is NWUE (NBUE), then most insertions should go near the front (back) of a linear list; thus, using a single pointer and always scanning from the ends of the list should perform well in any case. This method is quite easy to implement and would require essentially no more storage than the linear list so would probably be preferable to the latter in most cases.

4.3 Multiple Pointer Methods

A number of algorithms have been proposed which keep the future event set records in a linked linear list but, in addition, maintain a set(s) of pointer records. These pointers are used to logically divide the linked linear list into a number of sublists with the purpose of reducing the time to insert a new record.

The first such multiple pointer algorithm, called the indexed-list algorithm by Vaucher and Duval [39] spaces the pointer records equal amounts of time (DELTA) apart. Wyman independently proposed a similar algorithm [42]. Analysis of this algorithm [7] to determine the optimum value of DELTA has assumed the number of pointers is infinite so that an overflow sublist is not required. However, the number of pointers, NPTRS, must be finite and, in fact, reasonably

small if the storage requirements of this method are not to be excessive. Because of this, the probability of an insertion into the overflow sublist is 1 - F(NPTRS*DELTA) and the fraction of the future event set in the overflow sublist at any time is, at least, 1 - G(NPTRS*DELTA).

These values are most significant if F is NWUE or if there is greater than one scheduling distribution present since, in both cases, a high probability exists of event records being scheduled far into the future. Using the optimum value of DELTAT derived in [7], for a hyperexponential distribution with coefficient of variation near five, even if there is one pointer record for each event record, over 38% of the future event set will be in the overflow list [27]. In addition, if DELTAT is too large there is also the possibility of many event records falling into only a small number of sublists and thus eliminating the advantage of the indexed-list over a linear list.

Two different algorithms have been proposed as alternatives to the indexed-list due to the potential problems associated with its use. Franta and Maly [12,13] added a second set of pointers, accessed via the indexed-list pointers, which were dynamically created in order to keep the sublists of the future event set linear list to a fairly small size. This was called the Two-Level Structure. To do an insertion, after calculating the index in the pointer array, the sublist of secondary pointers is

scanned to find the pointer to the proper sublist of the future event set. The event is then inserted by scanning this sublist from the back.

Henriksen [17] uses only one set of pointer records kept in an array, as does the indexed-list, but instead of pointing at records spaced equal amounts of time apart, the algorithm essentially attempts to keep an equal number of records between pointers. The pointer record from which to start the scan to do an insertion is determined by doing a binary search of the pointer records. This permits the pointers to be dynamically adjusted by the algorithm as the need arises to keep the sublists short.

4.4 Non-Linear Structures

Although the results of section 2 can not be directly applied to nonlinear structures, they provide insight and understanding to their behavior. A special kind of binary tree, a p-tree (priority-tree), was recently described and analyzed [19]. The authors point out it is sensitive to the scheduling distribution but that insertions would be extremely efficient if the times of newly inserted records were generally larger than those already in the tree (that is, exhibit a NBUE behavior according to Theorem 1). The results of section 2 indicated NWUE distributions may be more representative of actual simulations and for these cases the performance of the p-tree is quite poor.

Another nonlinear structure proposed as suitable for this problem is a heap [15]. A heap remains balanced, unlike a p-tree, and in the worst cases can do an insertion or deletion in $O(\log_2 N)$. A problem with the heap is that records with the same key (time) value are not automatically processed first-in-first-out (i.e. a heap is not stable [23]). This can be overcome easily, although with some decrease in performance, by recording in each event record the number of records inserted prior to it and using this to break ties when necessary.

Generally insertions to a heap are done from the bottom of the heap and it has been shown the average time, when insertions are random, to do an insertion is bounded by a constant [32]. The next event record (to be removed) is at the root of the heap and after removing it, the heap is restored. It is more efficient; however, not to restore the heap since, if the next operation is an insertion, processing the insertion will restore the heap (this will be referred to as the modified heap algorithm). An insertion following a deletion would be from the top of the heap; thus, the insertion time would be $O(\log_2 N)$, if the times are random or likely to be greater than most in the heap (NBUE). If, however, the scheduling distribution is NWUE, one would expect the insertion/restoration of the heap time to be much less. Simultaneous events can be handled as described above.

5. Comparison of Algorithms

5.1 Description of Testing

In order to determine the best algorithm(s) for use in discrete event simulation, the execution time efficiency of twelve algorithms were experimentally compared. The algorithms, with mnemonics for each and an estimate of the overhead to maintain the future event set, are listed in Table 2 and were discussed in sections 2 and 3. Descriptions of them and their implementations may be found in [27].

Most of the algorithms have about the same amount of storage overhead, approximately two units per event record for pointers or time values. HPT needs one extra unit per record in order to break ties while HNR utilizes three short arrays to permit a binary search of the future event set. The storage overhead for multiple lists will be small as there will seldom be a very large number of lists. For VAU and FRA, in addition to that for linking records, there is storage overhead proportional to the number of dummy records each algorithm uses. A dummy record need not contain any information related to the simulation so can be quite short unless all records must be of equal length (c.f. GASP-IV [33]) in which case these two algorithms, especially FRA, could require a large amount of additional storage.

Table 2. Algorithms Tested

Algorithm	mnemonic	overhead	reference
Linked Linear List		2N+d+2	
Scan from Front	LLF		[22]
Scan from Back	LLB		[22]
Multiple Lists		2N+2m(d+3)	
Scan all from Front	MLF		[20]
Scan all from Back	MLB		[20]
Median Single Pointer		2N+2(d+2)	
Scan both halves from Front	MFF		[7]
Scan 1st half from Front, 2nd half from Back	MFB		[7]
Indexed-List (Vaucher's)	VAU	2N+n1(d+3)	[39]
Two-Level Structure (Franta's)	FRA	2N+n1(d+3) +n2(d+2)	[12]
Binary Search Indexed-List (Henriksen's)	HNR	2.75N	[17]
Heap Structures			
Unmodified, not FIFO ties	HEP	2N	[15]
Modified, not FIFO ties	HPM	2N	[27]
Modified, FIFO for ties	HPT	3N	[27]

N - size of future event set
m - number of lists (MLF and MLB)
d - size of each dummy record (used to mark end of a list or special records in VAU and FRA)
n1 - size of array used in VAU and FRA (n1 = 30 suggested)
n2 - number of dummy records created by FRA (dynamically) (optimum calculated as square root of N [12])

Three classes of models were used in the experiments. The hold model, used by others in their testing, was tested with six different scheduling distributions and the size of the future event set taking on seven values ranging from 10 to 1000. A large number of tests were conducted using simple simulation models based on simulations in textbooks and articles. Seven different systems were used including simulations of two-echelon inventory, job shop, computer, population, reliability, and machine repair systems. The final class included models of real world systems and included the models with the smallest and largest future event sets. Details of all models may be found in [27].

In addition to comparing the performance of the algorithms, the tests permit evaluation of some of the conclusions reached earlier. For the simulation models, the coefficient of variation and %F were measured permitting, for example, a comparison between them and observing whether the coefficient of variation is greater than one for simulation models. By using the hold model, it is possible to determine if that model can realistically be used to compare algorithm performance and, if so, which scheduling distribution(s) are most critical.

Execution time was used as the measure of complexity as it reflects, in addition to the number of key comparisons, the overhead associated with each of the algorithms. The algorithms were all written in FORTRAN, the general purpose

language most commonly used for simulation [21], with execution time the prime consideration. The algorithms were compliled using the IBM FORTRAN level H compiler, optimization level 2, and the experiments performed on an IBM 370/155 with no other jobs in the system.

For each simulation model tested, a trace of all insertions and deletions to the future event set was made and stored. The traces were used to test the algorithms reducing the cost of the tests, making it easier to to measure only the time to do future event set operations, and insuring the algorithms were tested under identical conditions.

Essentially no variability was observed between three observations of execution time made in steady state indicating the deleted transient period (although short) was sufficiently long and that the behavior of the future event set was not variable. Complete details of the methodology may be found in [27].

5.2 Results of Experiments

5.2.1 Results for Simple Simulation Models

Table 3 shows for each simple simulation model the number of different cases run of that model and the minimum, average, and maximum of the average size (N) of the future event set, average coefficient of variation (cv), and %F observed for cases of that model.

Table 3. Simple Simulation Models

Machine Repairman (MRP)
reference: [18] 7 cases
N: (30.1,163.1,745.4)
cv: (0.53,1.52,2.83)
%F: (21.2%,37.5%,71.4%)

Time-Shared Computer (TSC)
reference: [1] 4 cases
N: (21.7,79.6,243.7)
cv: (1.98,3.53,4.06)
%F: (5.7%,10.2%,20.2%)

Terminal System (TER)
reference: [25] 1 case
N: 12.9; cv: 3.90; %F: 8.1%

Job Shop (JOB)
reference: [36] 4 cases
N: (35.5,39.2,43.1)
cv: (1.24,1.49,1.74)
%F: (32.6%,37.0%,40.9%)

Inventory Model (INV)
reference: [33] 2 cases
N: (23.5,49.3,75.0)
cv: (2.33,2.57,2.81)
%F: (46.1%,47.0,47.8%)

System Reliability (REL)
reference: [14] 2 cases
N: (49.0,99.0,149.0)
cv: (1.23,1.24,1.24)
%F: (51.1%,51.1%,51.1%)

Contagious Disease (DIS)
reference: [41] 3 cases
N: (220.3,275.8,374.8)
cv: (0.51,0.54,0.60)
%F: (64.7%,68.4%,69.7%)

In Table 4 the average relative execution time (r.e.t.) is given for each algorithm and in Table 5 a summary of these results. The relative execution time is the algorithm's average execution time, to perform 1000 insertion and deletions, for all cases of a model relative to the minimum execution time for all of the algorithms for that model. The minimum average time is provided for each model in Table 4. The r.e.t. was used as it is easier to interpret and study the results. Results for each case of each model are in [27].

The first observation to make concerns %F and the coefficient of variation. The coefficient of variation and %F were usually greater than one and less than 50% respectively, despite the wide variety of models tested and even for models where the coefficient of variation of each scheduling distribution was much less than one. Again, the reason for this is the variation in the means of each scheduling distribution and these findings confirm the conclusions made in section 2 regarding the interaction of distributions. The expected relationship of coefficient of variation and %F was also observed (correlation of -0.80, high coefficient of variation suggests low %F) for these simulation cases.

The performance of most of the algorithms was also fairly consistent despite the differences in models, future event set size, and coefficient of variation. The

Table 4. Relative Execution Times (r.e.t.'s) for Each Model

ALGO-RITHM / model:	MRP	TSC	TER	JOB	INV	REL	DIS
LLF	2.59	1.02	1.00	1.19	1.53	2.69	7.38
LLB	4.72	3.99	1.71	1.53	1.54	2.33	3.06
MLF	2.83	1.51	1.93	1.73	1.78	2.00	5.29
MLB	2.67	1.36	1.95	2.01	1.31	1.84	2.68
MFF	1.98	1.13	1.24	1.16	1.36	2.32	4.64
MFB	1.60	1.07	1.24	1.06	1.32	2.10	2.20
VAU	1.24	1.38	1.69	1.11	1.03	1.11	1.00
FRA	1.42	1.60	2.24	1.50	1.49	1.53	1.25
HNR	1.00	1.12	1.45	1.06	1.00	1.08	1.03
HEP	1.34	1.65	2.02	1.25	1.18	1.23	1.08
HPM	1.07	1.00	1.33	1.00	1.06	1.00	1.06
HPT	1.22	1.12	1.52	1.16	1.25	1.15	1.26
min. avg. time:	77.3	61.3	42.0	70.5	72.0	71.0	92.3

Table 5. Summary of Relative Execution Times (r.e.t.)

ALGO-RITHM	Minimum	Maximum	Average*
LLF	1.00	7.38	2.49
LLB	1.53	4.72	2.70
MLF	1.51	5.29	2.44
MLB	1.31	2.68	1.98
MFF	1.13	4.64	1.98
MFB	1.06	2.20	1.51
VAU	1.00	1.69	1.22
FRA	1.25	2.24	1.58
HNR	1.00	1.45	1.10
HEP	1.08	2.02	1.39
HPM	1.00	1.33	1.08
HPT	1.12	1.52	1.24

* equal weight for each model

algorithms can be divided into three groups: those that generally performed poorly, LLF, LLB, MLF, MLB, and MFF; those of fair performance, HEP, FRA, and MFB; and those that performed well, HPM, HNR, VAU, and HPT.

Of those that performed poorly, only LLF and MFF performed well in any cases; in particular, for those cases with small future event sets. LLF also was generally more efficient that LLB as would be expected since %F was usually less than %B. MLB and MLF performed the worst which is not surprising since the choice of the number of lists and assignment to a list is not done with with execution efficiency a consideration.

Of those in the second group, of fair performance, the performance of MFB and FRA are the most significant. MFB performed quite well when the size of the future event set was small to intermediate in number (under 50) and would be relatively easy to implement. It can be seen that MFF was not quicker, serving as confirmation of the conclusions drawn in section 4 regarding their relative performance.

The performance of FRA was never very good; however, its execution time did not increase tremendously when the size of the future event set grew. The explanation is that because of the large overhead, FRA does not appear suitable for use with models of under 50, and perhaps 100, records in the the future event set.

These results differ completely from previously published results using FRA [12,13] for which there are two possible explanations. One is that previous test were done using PASCAL which has record structures which are ideal for implementing this algorithm. If this is the case, it only serves to emphasize the need to test these algorithms under conditions (languages) most likely to be encountered in actual simulations.

The second reason could be that an error exists in the implementation of this structure as reported in the technical report [11] on which the tests [12] of this algorithm were based. The effect of this error is to lose some inserted records; thus reducing the size of the future event set. This error does not appear in a later implementation written in Simula [14].

The final group can be divided in half, those that performed very well, HPM and HNR, and those that performed well, HPT and VAU. The outstanding performance of HPM and HNR was evident in many ways. For example, not only were their average execution times the lowest but in only a few cases did either rank worse than third and, in these cases, the size of the future event set was small, so it is not critical. HPT and VAU performed nearly as well but generally their times were slightly higher than those of HNR and HPM.

There are potential problems associated with these algorithms which could affect their use. HNR is a rather complicated algorithm to understand and write, especially the manner in which the binary search is done. In addition, the relative performance has been found to be very sensitive to the amount of code optimization done by the compiler [27,28]. The major problem with VAU is how critical the choice of DELTA is to its performance. The value is a function of the average future event set size and average time used in scheduling new events. This will be discussed further shortly. Finally, HPM is easy to implement, even in FORTRAN, but does not handle simultaneous events first-in-first-out and to do so deteriorates performance about 15% as seen in HPT.

5.2.2 Results for Real World Simulation Models

The results for the cases run for real world simulation models are summarized in Table 6. The three models are very different; one has a very high coefficient of variation, while the other two have very large and small future event sets. For the production shop model, those algorithms with very little overhead performed very well since the event set was so small. For the time sharing system, those algorithms which do insertions from the front of a linear list did very well, especially LLF even though the number of event records approached 100. Finally, for the large job shop, the

Table 6. Real World Simulation Model Results

Algorithm	Large Job Shop reference: [31] 1 case N: 3986.7 cv: 2.91 r.e.t.	Production Shop reference: [4] 1 case N: 10.36 cv: 0.59 r.e.t.	Michigan Time Sharing System reference: [29] 2 cases N: (48.6,65.1,81.5) cv: (7.92,8.79,9.66) r.e.t.
LLF	****	1.00	1.00
LLB	****	1.40	4.80
MLF	****	****	2.12
MLB	****	****	2.02
MFF	****	1.11	1.25
MFB	28.37	1.11	1.26
VAU	5.39	1.82	2.04
FRA	1.66	2.18	2.24
HNR	1.00	1.27	1.56
HEP	****	1.76	2.66
HPM	1.09	1.18	1.22
HPT	1.34	1.33	1.42
min. avg. time:	30.9	15.0	14.8

**** : Model not run for this algorithm

extremely large future event set (it reached 6862 records) was handled well by only HNR, HPM, and HPT.

These results demonstrate vividly the robustness in performance of HNR and HPM, and HPT. Despite the great differences in these models, these algorithms performed very well for all three in agreement with their performance for the simple simulation models.

5.2.3 Results for the Hold Model

Table 7 contains the summary data for the experiments performed using the hold model. Details of the data are contained in [27]. Again the same general breakdown of categories is evident: HNR, HPM, and HPT perform well; FRA performs poorly unless the future event set size is very large; MFB is very good when the future event set size is about 50 or less; and VAU performs well unless the coefficient of variation is high or the future event set is large. MLB and MLF were not tested as there is no basis for assigning events to different lists.

Care must be taken in using this data since it suggests that LLB is more efficient than LLF, in contrast to earlier conclusions. The reason is that the distributions used here; which with one exception, the hyperexponential, are those used by others in their testing; generally have coefficient of variation less than one and no interaction of distributions. The results for the hold model for the two

Table 7. Hold Model r.e.t. Averages

ALGO-RITHM	GRAND MEAN	DISTRIBUTIONS EXPON	U(1±1)	U(1±.1)	BIMOD	HYPER	DISCR
LLF	7.73	6.44	8.04	13.05	3.79	5.20	10.89
LLB	5.13	5.60	3.92	1.52	8.65	6.73	4.44
MFF	5.37	4.53	5.17	8.05	4.00	4.13	7.00
MFB	2.87	3.38	2.69	1.41	3.33	3.64	2.78
VAU	1.38	1.28	1.17	1.32	2.05	1.43	1.09
FRA	1.58	1.49	1.44	1.65	1.69	1.53	1.78
HNR	1.00	1.00	1.00	1.00	1.04	1.01	1.00
HEP	1.51	1.46	1.32	1.34	1.83	1.61	1.60
HPM	1.06	1.01	1.00	1.21	1.00	1.00	1.26
HPT	1.21	1.12	1.12	1.35	1.10	1.11	1.61
min. avg. time:	23.6	24.9	25.7	22.2	22.6	24.5	20.6

ALGO-RITHM	Future Event Set Sizes 10	25	50	100	200	400	1000
LLF	1.07	1.31	1.83	2.93	5.10	10.04	24.31
LLB	1.00	1.12	1.47	2.13	3.43	6.84	15.28
MFF	1.12	1.21	1.56	2.24	3.73	6.76	16.17
MFB	1.02	1.00	1.08	1.37	1.97	3.36	8.13
VAU	1.44	1.26	1.18	1.19	1.21	1.43	1.92
FRA	2.00	1.76	1.70	1.58	1.48	1.46	1.48
HNR	1.13	1.04	1.00	1.00	1.00	1.00	1.00
HEP	1.56	1.47	1.48	1.54	1.56	1.55	1.56
HPM	1.09	1.05	1.05	1.09	1.08	1.10	1.12
HPT	1.25	1.16	1.20	1.23	1.22	1.27	1.29
min. avg. time:	16.3	19.4	21.3	22.9	25.3	27.3	29.8

EXPON - exponential; cv = 1; %F = 50%
U(1±1) - Uniform; cv = 0.577; %F = 67%
U(1±.1) - Uniform; cv = 0.0577; %F = 97%
BIMOD - 90% U(0,S),10% U(100S,101S),S=.095;cv=2.856;%F=12%
HYPER - Hyperexponential; cv = 4.951; %F = 27%
DISCR - Equally likely 0,1,2; cv = .707;%F not applicable

distributions with coefficient of variation greater than one are in agreement to those found earlier and thus suggest that tests done using the hold model should place greater emphasis on such distributions.

Additional tests were performed for the hold model for VAU and FRA to determine the dependence of their performance on the parameter DELTAT used to space pointer records. The values of DELTAT used were five and ten times smaller and larger than the recommended value since the performance of both algorithms was reported to be fairly insensitive to changes in DELTAT in this range [12,39]. It was found the performance generally deteriorated when DELTAT was not the recommended value, especially when the number of events was greater than 50 and more so if DELTAT was too small.

5.3 Transient Effect

All previous data reported in this article has been for the performance of the data structures under steady state conditions. Timings were also collected [27] while the models were in the transient state in order to determine whether the performance of any of the algorithms was seriously affected by the transient conditions.

The difference in performance is usually small between the transient and steady state results. Typically, the difference was large only in the hold model, for the two

distributions with coefficient of variation greater than one, and those simulation cases, with a large number of events and high coefficient of variation.

The explanation for this is quite simple. Initially, all records are in the future event set with a value of 0.0. The first deletion/insertion operation for each record consists of removing a value of 0.0 and inserting the record with a new value greater than 0.0. For LLF this means these insertions must pass all of the 0.0 records while LLB need not. Thus, the execution time for the transient period for LLB is less than that for LLF but the reverse is true of the execution time in steady state. Similar reasons hold for the other data structures for these two distributions. For small future event sets, the effect is not as pronounced as much of the time considered transient is essentially steady state.

Despite the general closeness of the results for transient and steady state, the need to remove the transient remains when analyzing future event set behavior for a simulation in steady state. This is because, as was found here for hold and simulation models, for situations in which the coefficient of variation is greater than one the results for the transient period can be quite different from the steady state test results.

For terminating simulations [26], like the contagious disease model [41], where there are no transient or steady

state periods, the closeness of the transient and steady state results would indicate that the performance of the algorithms would not be significantly different. The same could be said for simulations in which only the transient phase is of interest, unless the transient is very short. In that case, those algorithms that perform well when the coefficient of variation of the scheduling distribution is mall would be preferable, as demonstrated by the performance of LLB for the transient cases here.

6. Conclusions

There has been extensive research recently on the subject of algorithms for handling the future event set in discrete event simulations. Most of this research, however, has confined itself to evaluating algorithms using a simple model, used in one of the first studies [39], without questioning whether this model or methodology was appropriate. One of the key differences in the research described here is the analysis done using this model and extensions to it prior to any testing of the algorithms. Analysis of the algorithms and of the scheduling distributions provided a better understanding of the subject and thus the results of the experiments are meaningful to real simulations.

Analytical study of the problem led to greater understanding of the effect the scheduling distribution has on insertion time. One key result, Theorem 1, established a relation between distributions being of class NBUE or NWUE to the percentage of a linear list scanned to do an insertion from the front (%F). There was also established analytically a relation between %F and the coefficient of variation (cv) of a scheduling distribution (seen empirically in section 5). That is, if a scheduling distribution is NBUE(NWUE) then not only is %F $\geq$ ($\leq$) 50% but also cv $\leq$ ($\geq$) 1.

Theorem 2 showed that when more than one scheduling distribution is present (which is typical of actual simulations), the effect is to increase the coefficient of variation and lower %F. The implication is that in real simulations, the coefficient of variation of the interaction of the scheduling distributions will often be greater than one. Thus, tests of algorithms using the hold model should use scheduling distributions with that characteristic, whereas most tests have had only one such distribution [5,12,13,17,39]. Knowledge of the effect of interaction of scheduling distributions was used to decide the scanning directions for the median pointer methods and indicated that problems exist with the indexed list algorithm due to the high coefficient of variation. It also resulted in a modification to the use of a heap for handling the future event set which improved its performance significantly.

The methodology of section 5 reflected the goal of obtaining useful results. One step was to use simulation models with which to conduct the majority of the tests rather than the simple hold model. In addition, data on the models themselves were gathered to compare to the conclusions drawn in section 3. There was strong agreement between the analytical results of section 3 and the empirical ones of section 5 regarding the models. Typically, the models had a coefficient of variation greater than one and %F less than 50%.

Finally, the relative performance of the algorithms is significant. These results differed from previous studies largely due to the methodology used in conducting the tests. Four algorithms performed well (HNR, HPM, HPT, and VAU), three were of fair performance (HEP, FRA, MFB), and five, generally, performed poorly (LLF, LLB, MLB, MLF, MFF). Storage considerations should not be a major issue since most of the algorithms, including those that performed best, required very little storage beside that necessary to link the event records. Of those classified as only fair performers, MFB is very good provided the future event set is not extremely large and is essentially no more difficult to implement than the traditional linked list.

REFERENCES

1. Adiri, I. and Avi-Itzhak, B. "A Time-Sharing Queue With a Finite Number of Customers," Journal of the ACM 16, 2 p. 315-323.

2. Barlow, R.E. and Proschan, F. Mathematical Theory of Reliability, New York: John Wiley and Sons, 1969.

3. Barlow, R.E. and Proschan, F. Statistical Theory of Reliability and Life Testing: Probability Models, New York: Holt, Rinehart and Winston, Inc., 1975.

4. Biles, W.E. "Computer Simulation of Material Handling Configurations in Production Systems," presented at the 1978 Spring National ORSA/TIMS Meeting, May 1978.

5. Comfort, J.C. "A Taxonomy and Analysis of Event Set Management Algorithms for Discrete Event Simulation," Proceedings of the 12th Annual Simulation Symposium, March 1979, p. 115-146.

6. Dahl, O., Myhrhaug, B., and Nygaard, K. Common Base Language, Publ. No. 5-22, Norwegian Computing Center, October 1970.

7. Davey, D. and Vaucher, J. "An Analysis of the Steady State Behaviour of Simulation Sequencing Set Algorithms," University of Montreal, December 1976.

8. Emshoff, J.R. and Sisson, R.L. Design and Use of Computer Simulation Models, New York: The MacMillan Co., 1970.

9. Englebrecht-Wiggans, R. and Maxwell, W.L. "Analysis of the Time Indexed List Procedure for Synchronization of Discrete Event Simulations," Management Science, Volume 24, September 1978, p. 1417-1427.

10. Fishman, G.S. Concepts and Methods in Discrete Event Digital Simulation, New York: John Wiley and Sons, 1973.

11. Franta, W.R. and Maly, K. "An Event Scanning Algorithm of Nearly Constant Complexity," Technical Report 75-18, Computer, Information, and Control Sciences Department, University of Minnesota, November 1975.

12. Franta, W.R. and Maly, K. "An Efficient Data Structure for the Simulation Event Set," Communications of the ACM 20, 8 p. 596-602.

13. Franta, W.R. and Maly, K. "A Comparison of HEAPS and the TL Structure for the Simulation Event Set," Communications of the ACM 21, 10 p. 873-875.

14. Franta, W.R. The Process View of Simulation, New York: North-Holland, 1977.

15. Gonnet, G.H. "Heaps Applied to Event Driven Mechanisms," Communications of the ACM 19, 7 p. 417-418.

16. Hardy, G.H., Littlewood, J.E., and Polya, G. Inequalities, 2nd edition, Cambridge: University Press, 1964.

17. Henriksen, J.O. "An Improved Events List Algorithm," Proceedings of the Winter Simulation Conference, December 1977, p. 554-557.

18. Hillier, F.S. and Lieberman, G.J. Operations Research, 2nd edition, San Francisco: Holden-Day, Inc., 1974.

19. Jonassen, A. and Dahl, O.J. "Analysis of an Algorithm for Priority Queue Administration," BIT 15, 4 p. 409-422.

20. Kiviat, P.J., Villanueva, R., and Markowitz, H.M. Simscript II.5 Programming Language, edited by E.C. Russell, Arlington, Virginia: CACI, Inc., 1973.

21. Kleine, H. "A Second Survey of User's Views of Discrete Simulation Languages," Simulation, 17, 2 p. 89-94.

22. Knuth, D.E. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Reading, Massachusetts: Addison-Wesley, 1969.

23. Knuth, D.E. The Art of Computer Programming, Vol. 3: Sorting and Searching, Reading, Massachusetts: Addison-Wesley, 1969.

24. Laughlin, G.W. "Reducing the Run Ratio of a Multiprocessor Software System Simulator," Proceedings of the 8th Annual Simulation Symposium, 1975, p. 115-134.

25. Lavenberg, S.S., Moeller, T.L., and Welch, P.D. "Control Variates Applied to the Simulation of Queueing Models of Computer Systems," in Computer Performance edited by K.M. Chandy and Martin Reiser, North-Holland, 1977, p. 459-467.

26. Law, A.M. "Statistical Analysis of the Output Data from Terminating Simulations," Technical Report No. 78-4, Department of Industrial Engineering, University of Wisconsin, Madison, 1978.

27. McCormack, W.M. Analysis of Future Event Set Algorithms for Discrete Event Simulation, Ph.D. Dissertation, Syracuse University, 1979.

28. McCormack, W.M. and Sargent, R.G. "Comparison of Future Event Set Algorithms for Simulations of Closed Queueing Systems," in Current Issues in Simulation, edited by Adam, N.R. and Dogramaci, A., New York: Academic Press, 1979.

29. Moore, C.G. "Network Models for Large-Scale Time-Sharing Systems," Technical Report No. 71-1, Department of Industrial Engineering, University of Michigan, April, 1971.

30. Morse, P.M. Queues, Inventory and Maintenance, New York: John Wiley and Sons, Inc., 1958.

31. Parmelee, W. Personal communication, August 1978.

32. Porter, T. and Simon, I. "Random Insertion Into a Priority Queue Structure," IEEE Transactions on Software Engineering, SE-1 3, p. 292-298.

33. Pritsker, A.A.B. The GASP IV Simulation Language, New York: John Wiley and Sons, 1974.

34. Pritsker, A.A.B. "Ongoing Developments in GASP," Proceedings of the Winter Simulation Conference, December 1976, p. 81-83.

35. Russell, E.C. Simulating with Processes and Resources in Simscript II.5 (User's Manual), Arlington, Virginia: CACI, 1976.

36. Schriber, T.J. Simulation Using GPSS, New York: John Wiley and Sons, 1974.

37. Taneri, D. "The Use of Subcalendars in Event Driven Simulations," Proceedings of the Summer Simulation Conference, July 1976, p. 63-66.

38. Ulrich, E.G. "Event Manipulation for Discrete Simulations Requiring Large Numbers of Events," Communications of the ACM 21, 9 p. 777-785.

39. Vaucher, J.G. and Duval, P. "A Comparison of Simulation Event List Algorithms," Communications of the ACM 18, 4 p. 223-230.

40. Vaucher, J.G. "On the Distribution of Event Times for the Notices in a Simulation Event List," INFOR 15,2 p. 171-182.

41. Wyman, F.P. Simulation Modeling: A Guide to Using Simscript, New York: John Wiley and Sons, 1970.

42. Wyman, F.P. "Improved Event Scanning Mechanisms for Discrete Event Simulation," Communications of the ACM 18,6 p. 350-353.

Appendix A

Proof of Theorem 2

PROOF.

The proof is quite long and thus has been divided into a number of lemmas which will be proven first. All of the lemmas have the same assumptions as Theorem 2.

LEMMA 2.1.

Suppose an insertion is picked at random, then the probability that the record being inserted is of type i is given by:

$$P(\text{inserted record of type } i) = (n_i/u_i) \;/\; \left(\sum_{k=1}^{I} [n_k/u_k]\right)$$

where u_i is the mean scheduling time for event records of type i.

PROOF.

Consider insertions over a period of time of length T. On the average, each record of type i will be inserted into the event set structure T / u_i times. There are n_i such records, so (T * n_i) / u_i records of type i will be inserted over T and the total number of records inserted over T will be $\sum_{k=1}^{I}[(T * n_k) / u_k]$. Thus,

$$P(\text{insert is type } i) = [(T * n_i) / u_i] \;/\; \sum_{k=1}^{I} [(T * n_k) / u_k].$$

LEMMA 2.2.

$$\int_0^\infty G_j(x) * dF_i(x) = (1/u_j) * \int_0^\infty \bar{F}_i(x) * \bar{F}_j(x)\ dx$$

where G_j is the event set distribution for scheduling distribution F_j. Furthermore, if i=j, an alternative form for $\%F_i$ is the result:

$$\%F_i = \int_0^\infty G_i(x) * dF_i(x) * 100\%$$

$$\%F_i = (100\%/u_i) * \int_0^\infty \{[\bar{F}_i(x)]^2\}\ dx.$$

PROOF.

By the definition of G(x) in section 2.2,

$$\int_0^\infty G_j(x) * dF_i(x) = \int_0^\infty [(1/u_j) * \int_0^x \bar{F}_j(t)\ dt]\ dF_i(x)$$

$$= (-1/u_j) \int_0^\infty \{ \int_0^x [\bar{F}_j(t)\,dt][-dF_i(x)] \}.$$

Using integration by parts,

let $r = \int_0^x \bar{F}_j(t)\ dt$ and $ds = -dF_i(x)$, so

$dr = \bar{F}_j(x)\ dx$ and $s = \bar{F}_i(x)$.

Thus,

$$\int_0^\infty G_j(x)\ dF_i(x) = (-1/u_j) * \{ [\bar{F}_i(x) * \int_0^x \bar{F}_j(t)\,dt] \Big|_0^\infty - \int_0^\infty [\bar{F}_i(x)\,\bar{F}_j(x)\,dx] \}$$

At x = infinity, $\bar{F}_i(x) = 0$ and at $x = 0$, $\int_0^x \bar{F}_j(t)\,dt = 0$, thus the first term is zero leaving:

$$\int_0^\infty G_j(x)\ dF_i(x) = (1/u_j) * \int_0^\infty \bar{F}_i(x) * \bar{F}_j(x)\ dx.$$

LEMMA 2.3.

For all $1 \leq i,j \leq I$,

$$\int_0^\infty G_j(x)\ dF_i(x) * 100\% \leq [\mathrm{sqrt}(u_i/u_j)] * \%F_{max}.$$

PROOF.

The proof uses the Schwartz Inequality [16]: For any two functions r and s,

$$[\int r(x)*s(x)dx]^2 < [\int \{r(x)\}^2\ dx] * [\int \{s(x)\}^2\ dx]$$

unless $Ar \doteq Bs$ (equivalent) where A and B are constants not both equal to zero. $g \doteq h$ means $g(x) = h(x)$ except possibly in a set of measure zero. If $Ar \doteq Bs$, then equality occurs.

Since both sides are positive, the lemma is proved if it can be shown:

$$Z^2 \leq (u_i/u_j)*(\%F_{max})^2.$$

where $Z = \int_0^\infty G_j(x)\ dF_i(x) * 100\%$.

By Lemma 2.2,

$$Z^2 = [(100\%/u_j)^2] * [\int_0^\infty \bar{F}_i(x)\bar{F}_j(x)dx]^2.$$

Using the Schwartz Inequality,

$$z^2 \leq [(100\%/u_j)^2]) * [\int_0^\infty \{\bar{F}_i(x)\}^2\, dx] * [\int_0^\infty \{F_j(x)\}^2\, dx].$$

Rearranging terms and then using the alternative form for %F derived in Lemma 2.2,

$$z^2 \leq (u_i/u_j) * \{(100\%/u_i) \int_0^\infty [\bar{F}_i(x)]^2\, dx\} * \{(100\%/u_j) \int_0^\infty [\bar{F}_j(x)]^2\, dx\}$$

$$z^2 \leq (u_i/u_j) * (\%F_i) * (\%F_j)$$

Thus, using the definition of $\%F_{max}$ the lemma is proved,

$$[100\% * \int_0^\infty G_j(x)\, dF_i(x)]^2 \leq (u_i/u_j) * (\%F_{max})^2.$$

PROOF of THEOREM 2.

When inserting a record of type i, the average number of records in the future event list that will be passed in the scan will be:

$$a + \sum_{j \neq i} b_j$$

where a is the average number of records of type i passed and b_j is the average number of records of type j passed.

The expression for a comes directly from Theorem 1. $\%F_i$ is the average percentage of records scanned of type i when inserting a record of type i. There are $(n_i - 1)$ records of type i when inserting a record of type i, so

$$a = (n_i - 1)*\%F_i = (n_i - 1)*\int_0^\infty G_i(x)\ dF_i(x).$$

The expression for b_j is only slightly more complex and follows from the derivation of %F in Theorem 1:

$$b_j = \int_0^\infty n_j*(\%\text{ type j records with remaining life} \leq x)*$$

$$[P(\text{type i is scheduled for x units from now})]$$

$$= \int_0^\infty [n_j * G_j(x)]\ dF_i(x) * 100\%.$$

Thus, the fraction of the total list scanned when inserting a record of type i is

$$(a + \sum_{j \neq i} b_j) / (N - 1)$$

since N-1 is the size of the future event set when doing an insertion.

The expression for $\%F_{int}$ is thus a weighted average:

$$\%F_{int} = \sum_{i=1}^{I} P(\text{insert type i record})$$

$$* (\%\text{ scanned to insert a type i})$$

$$= \sum_i [(n_i/u_i)/(c*[N-1])]$$

$$* \{[(n_i-1)*\int G_i dF_i] + [\sum_{j \neq i} n_j * \int G_j dF_i]\} * 100\%$$

where the abbreviation $\int G_j dF_i$ represents $\int_0^\infty G_j(x) dF_i(x)$ and $c = \sum_{K=1}^{I} (n_k/u_k)$.

By Lemma 2.3,

$$\%F_{int} \leq \{1/c*[N-1]\}$$

$$* \sum_i (n_i/u_i)*\{(n_i-1)([\mathrm{sqrt}(u_i/u_i)]*\%F_{max})$$

$$+ \sum_{j\neq i} n_j([\mathrm{sqrt}(u_i/u_j)]*\%F_{max})\}$$

$$=\{[\%F_{max}]/[c(N-1)]\}$$

$$* \{\sum_i (n_i/u_i)([\sum_j n_j*\mathrm{sqrt}(u_i/u_j)]-1)\}.$$

Thus, Theorem 2 is true if

$$\sum_i (n_i/u_i)*([\sum_j n_j\ \mathrm{sqrt}(u_i/u_j)] - 1) \leq c * (N-1),$$

or equivalently, if

$$\sum_i (n_i/u_i)(\sum_j [n_j\ \mathrm{sqrt}(u_i/u_j)]) \leq \sum_i (n_i/u_i) + c(N-1) = cN$$

This is true if and only if

$$[\sum_i n_i/\mathrm{sqrt}(u_i)][\sum_j n_j/\mathrm{sqrt}(u_j)]=[\sum_i n_i/\mathrm{sqrt}(u_i)]^2 \leq cN.$$

$$\{\sum_i n_i*n_i/u_i\} + \{2\sum_{i<j}\sum [n_i*n_j/\mathrm{sqrt}(u_i*u_j)]\}$$

$$\leq \{\sum_i n_i*n_i/u_i\} + \{\sum_{i<j}\sum n_i*n_j*[(1/u_i)+(1/u_j)]\}.$$

This last inequality will now be shown to be true.

The geometric mean is less than the arithmetic mean [16],

$$\mathrm{sqrt}(u_1*u_2) \leq (u_1 + u_2) / 2$$

thus,

$$2/\mathrm{sqrt}(u_i*u_j) \leq (1/u_i) + (1/u_j)$$

since n_i and n_j are both greater than 0,

$$(2*n_i*n_j)/\mathrm{sqrt}(u_i*u_j) \leq (n_i*n_j)*[(1/u_i) + (1/u_j)]$$

Finally, by summing both sides of this last inequality for all i and j with j>i, the proof of Theorem 2 is completed.