RC 8001 (#34717) 12/10/79
Computer Science 10 pages

A Simple and Robust Data Structure for the Simulation Event Set

Charles E. Mear

Hewlett-Packard
Fort Collins, Colorado 80525

Charles H. Sauer

IBM Thomas J. Watson Research Center
Yorktown Heights, N.Y. 10598

Typed by Kathleen A. Brown (cs.2447)
Formatted Using the Yorktown Formatting Language
Printed on the Experimental Printer

ABSTRACT: Most simulation systems use a relatively inefficient linear list structure for the event set. A number of structures have been proposed for the event set, but the best of these, Franta and Maly's TL structure, is fairly complicated, especially when modified for general purpose use. We propose a simple structure, the ML (Multi-Level) structure, which has efficiency comparable to the TL structure in general and better than the TL structure when the event set is small.

Key Words and Phrases: simulation, event lists, data structures
CR Categories: 3.34, 4.22, 5.5, 8.1

1. Introduction

Discrete event simulation is a frequently used tool for evaluating general dynamic systems, including computer systems [FISH73,FISH78,FRAN77a,SAUE80]. In discrete event simulation, transitions from one distinct state to another within the modeled system are represented by the occurrence of events. Each event is characterized by the state change and the time of occurrence. The simulation program must maintain a set of events and schedule the events within the system so that state changes occur in the correct time sequence. After all state changes have been made at the time of a particular event, simulated time is advanced to the time of the next event, where requisite state changes are made again; hence simulated time advances in discrete leaps corresponding to the occurrence of events. At each state change new events may be added to the event set.

The principal operations on the event set are the removal of the event with earliest time and the insertion of a new event. (Another common operation is cancelling of a pending event. The actions involved in cancelling an event are essentially the same as actions for event removal and insertion.) When the event set is large, as is often the case, a large proportion of the execution time of the simulation program is due to these operations.

The usual data structure for the event set in simulation languages such as GPSS, SIMSCRIPT and SIMULA is a doubly linked linear list (or several such lists, in SIMSCRIPT) [FISH73]. Note that languages such as GPSS and SIMULA which use a "process view" of simulation are based on an event set mechanism, hidden from the user [FISH73]. Searching such a list to insert or remove an event may be expensive computationally. In recent years alternative data structures have been proposed to reduce insertion and removal times for the event set. The most promising of these is the TL (Two Level) structure of Franta and Maly [FRAN77b, FRAN78]. Insertion and removal times for the TL structure are small and nearly independent of the size of the event set. However, the TL structure *as originally proposed* makes strong assumptions about the characteristics of the event set. (E.g., it is assumed in FRAN77b and FRAN78 that the size of the event set is fixed. We show that this assumption is not necessary for the TL structure to be effective.) The TL structure is also dependent on availability of parameters prior to the simulation run. These parameters are used to define the details of the structure. Given the necessary parameters, the TL structure dramatically outperforms all previous structures except when the event set is small. The linear list is the most efficient structure when the event set is sufficiently small, say less than thirty events.

This work proposes a new data structure, the ML (Multi-Level) structure (similar to B-trees [KNUT68]) which is simpler than the TL structure, comparably robust and more efficient for small event sets. The worst case complexity is shown to be O(log N), where N is the (current) size of the event set. As with the TL structure, empirical results show the average complexity to be much better than the worst case.

We next describe the ML structure and characterize its complexity. Subsequently, we (1) describe empirical results for the test cases of FRAN77b, (2) describe our modifications to the TL algorithm for dynamic estimation of parameters and restructuring, and (3) describe

empirical results for simple queueing systems. Our discussions of the TL algorithm assume the reader is familiar with FRAN77b.

2. The ML (Multi-Level) Structure

The motivation of the ML structure was the desire to capture the efficiency of a binary search without the full cost of maintaining a balanced binary tree. Several linear lists of *keys* are created. Each list of keys uniquely partitions the event list and is referred to as a level of the structure. For a fast insertion process the keys would partition the list as follows. The top level key would point to the middle of the event list, dividing it into halves; the second level of keys would divide the list into fourths, and third into eighths, etc. Each key contains a pointer to the corresponding key on the next lower level. To find the proper insertion point for a new event, a binary search could be performed on the event list by scanning from the top, down through the levels of keys.

Preserving these exact divisions of the event list however, would be expensive since each insertion or removal could cause several keys to be modified. The ML structure reduces the amount of restructuring by partitioning the list into larger, more irregular, sublists.

Level one keys are created to divide the event list into sublists of roughly NLIM events each. The idea is to maintain an equal number of events in each sublist. Whenever the number of events in a sublist exceeds the limit NLIM, an event is either moved to an adjacent sublist or another sublist, and level one key, is created. The level one keys in turn are partitioned by the level two keys. Additional levels are added to the structure until the number of keys on the top level is small. The ML structure is pictured in figure 1.

Each key record contains pointers to predecessor and successor keys on the same level, a variable for event time, a pointer to the head of its sublist (a key on the next lower level), a count of the number of keys in its sublist, and a flag indicating whether the key itself is the head of a sublist. For ease of programming events are just keys with an event descriptor included. The keys at level zero of the structure are the events in the event set.

An array of pointers indicates the first key in each level of keys. The variable LEVEL indicates the highest level in the structure which contains a key.

Insertion of an event is straightforward. The top level of keys is scanned linearly until the sublist which contains the event is found. The linear scan continues within this sublist of lower level keys. When level zero is reached, the event may be inserted in its proper location in the event list. Should it prove expedient, the ML structure can be easily modified to be searched backwards.

Dynamic restructuring of the keys in a sublist exceeds NLIM. Insertion of a key on level i will require a key on level $i+1$ to be updated or a new level $i+1$ key to be created. Another level of keys is added to the structure when the number of keys on the top level exceeds NLIM. The removal of an event may cause one or more level of keys to be updated.

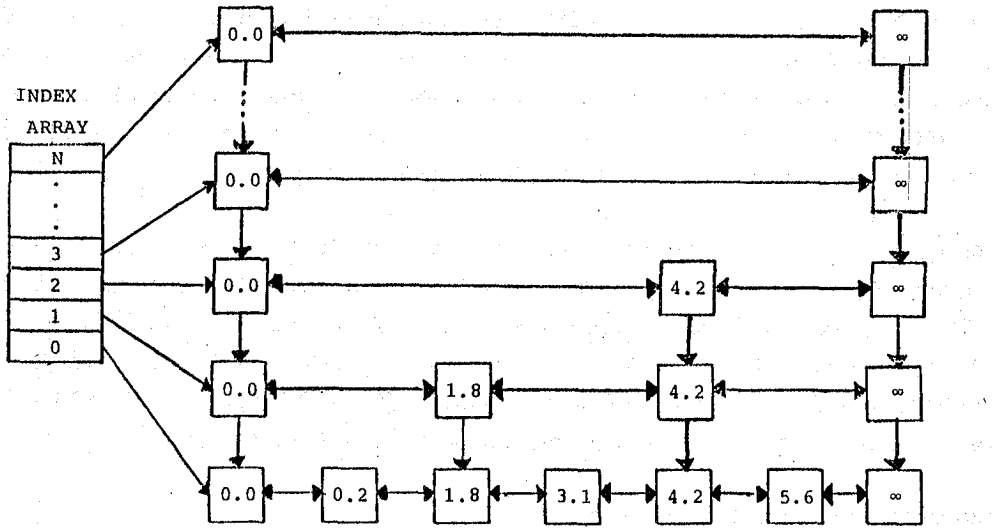

Figure 1. ML Structure

The ML structure admits a similarity to B-trees [KNUT75]. The ML structure is more flexible in that keys may migrate from node to node on each level and additional pointers are present to maintain the data structure. A PASCAL implementation of the ML structure is given in MEAR79.

3. ML Structure Complexity

Let us examine how many keys must be scanned to insert an event. A full sublist is split by dividing it into two sublists of NLIM/2 elements each. Suppose there are N events in the list and L levels in the structure. The number of keys on level L, L-1, L-2, ..., 0 is at least 1, (NLIM/2), $(NLIM/2)^2$., ..., $(NLIM/2)^L$. Therefore

$$N > (NLIM/2)^L \quad \text{and} \quad L < \log_{(NLIM/2)}(N).$$

The number of keys scanned on each level is at most NLIM; hence the number of keys scanned to insert an event is at most NLIM • $LOG_{(NLIM/2)}(N)$.

We expect the average complexity to be much better for two reasons:

1. A full sublist is split only if the right adjacent sublist is full or pointed to by a higher level key. Otherwise an event is moved from the full sublist to the right adjacent sublist. Most sublists therefore will average more than NLIM/2 elements.

2. In many cases the expected number of scans on each level will be closer to NLIM/2 than NLIM.

When an event is removed from the list, we may have to remove as many as L keys from the structure. Since there are at least NLIM/2 elements in each sublist, the average number of level 1 keys removed per event removal is at most 2/NLIM. The average number of level 2 keys removed is at most $(2/NLIM)^2$, etc. Therefore the average number of keys removed from the structure per event removal is at most $(2/NLIM) + (2/NLIM)^2 + ... + (2/NLIM)^L$ which is less than $1/(NLIM/2 - 1)$, assuming NLIM is greater than 2.

When an event is inserted we may have to split as many as L lists in the structure, but the average number of splittings per insertion is much less. Each key above level zero represents a splitting that occurred when the structure was being built. There are at most $N/(NLIM/2)$, $N/(NLIM/2)^2$, ..., $N/(NLIM/2)^L$ keys on levels 1, 2, ..., L, respectively. Therefore the number of keys above level zero is less than $N((2/NLIM)+(2/NLIM)^2+...+(2/NLIM)^L)$, and the average number of splittings per insertion is less than $1/(NLIM/2 -1)$.

4. Empirical Results - Fixed Event Set Size

It is difficult to design a series of tests which would subject the event list algorithm to the conditions existing in a general simulation system. An accepted test procedure is to measure the average time to perform the HOLD operation on the event set [VAUC75]. The HOLD operation is a combination of insertion and removal. The first event is removed from the list and rescheduled t time units in the future; t is a random variable described by the probability distribution function F. To initialize the procedure, N events are placed in the event list at time zero. The number of events in the list remains constant throughout the test.

By appropriate choices for N and F various simulation problems can be represented. The following distributions for F were chosen, as in FRAN77b:

1. Negative exponential with mean 1.
2. Uniform distribution on the internal [0, 2].
3. Uniform distribution on the interval [0.9, 1.1].
4. Ninety percent probability - uniform on the internal [0, S]. Ten percent probability - uniform on the internal [100×S to 101×S] where S is chosen to give the distribution mean 1.
5. Constant value 1.
6. The values 0, 1, and 2 with equal probabilities.

Tests were performed for N = 5, 10, 20, 50, 100, 300, 500. The ML and TL algorithms were programmed in PASCAL and the tests run on a Control Data 6600.

The time required to generate random numbers and call the insert and remove procedures was subtracted from the running times so that the times from each distribution could be directly compared.

The only internal parameter which affects the performance of the ML structure is NLIM. Tests were made for a number of values of NLIM and all combinations of F and N. The best

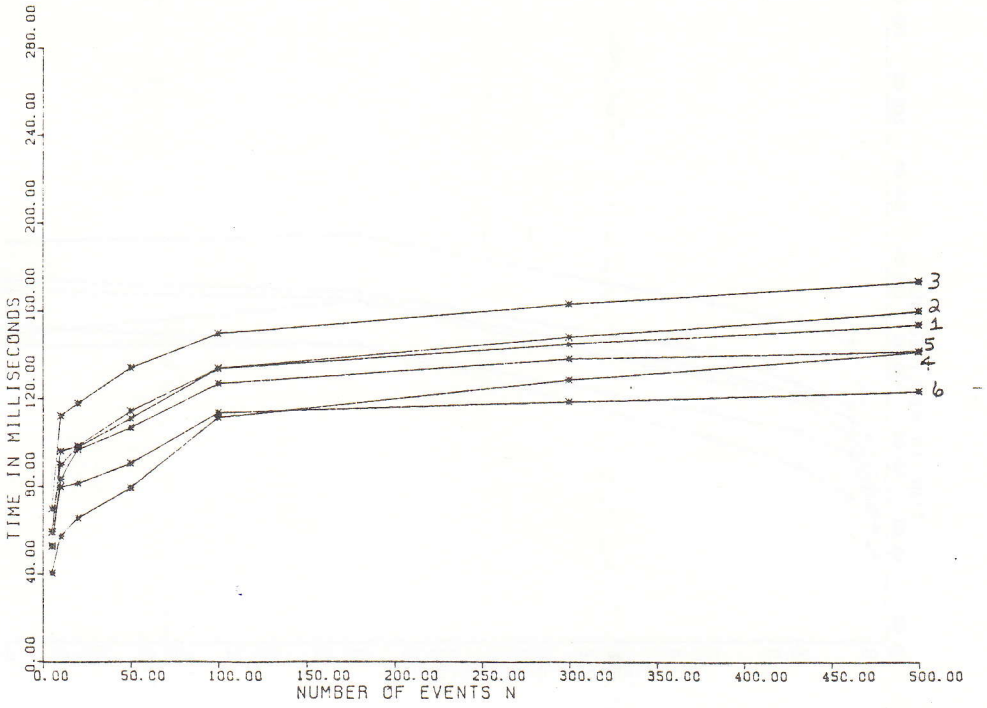

Figure 2. ML Structure HOLD Times - NLIM=10

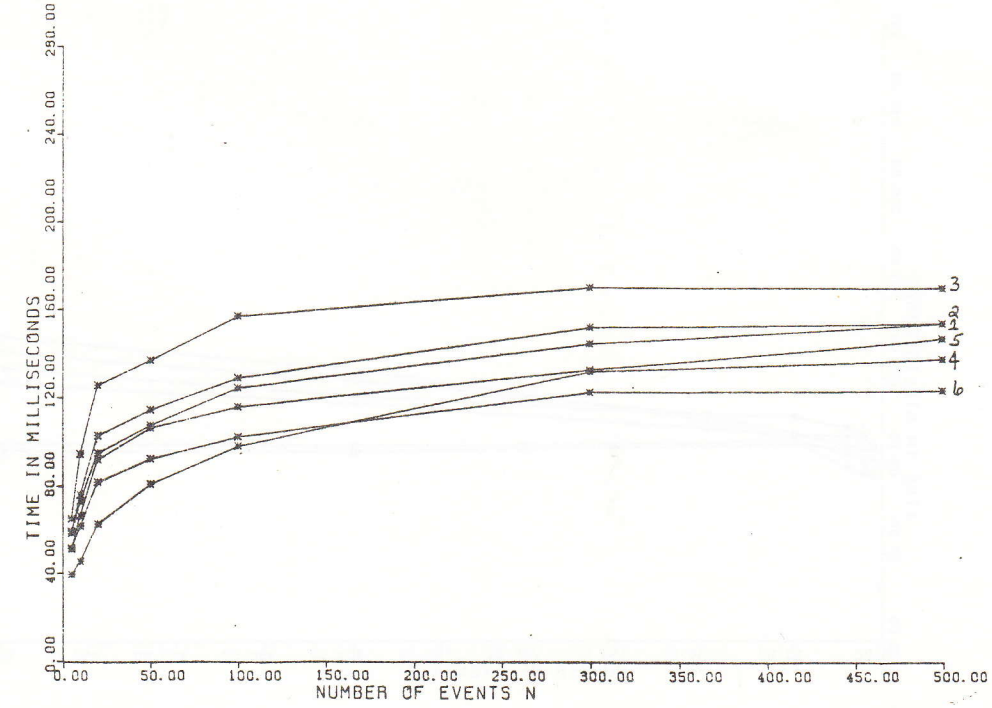

Figure 3. ML Structure HOLD Times - NLIM=14

performance, for nearly all combinations of F and N, occurred with NLIM = 14. Graphs of HOLD times vs N are presented for NLIM = 10, 14, and 18 in figures 2, 3, and 4, respectively. The performance of the ML structure deteriorated gradually for NLIM less than ten or

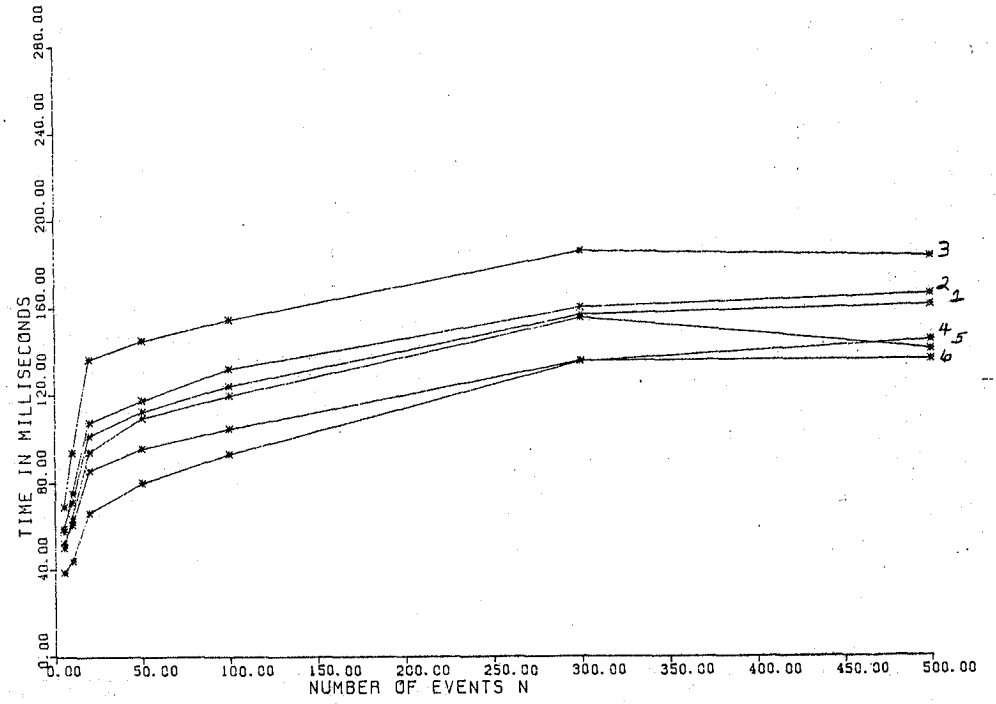

Figure 4. ML Structure HOLD Times - NLIM=18

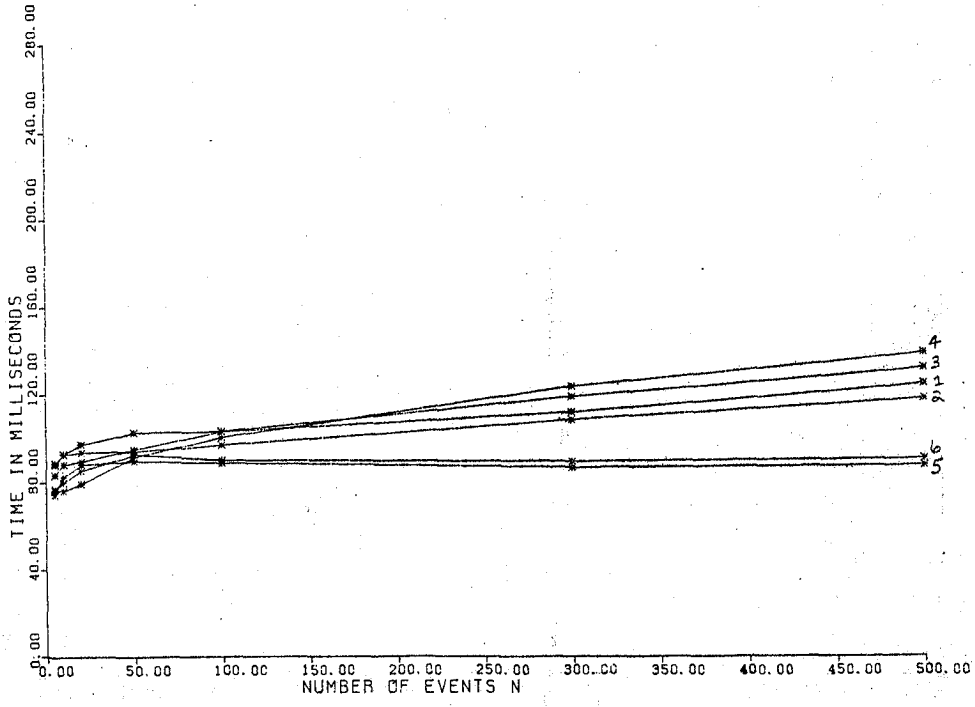

Figure 5. TL Structure HOLD Times
DT=(8/(N−1)+.07)T, DU=30, NLIM=max(8,$N^{1/2}$)

greater than eighteen. The optimum value of NLIM does not seem to be dependent upon the

number of events in the list nor the distribution of HOLD times. Experiments showed no advantage gained by varying NLIM from level to level.

Figure 5 is a graph of HOLD time versus N for the TL structure. The results are in agreement with those given in FRAN77b. The figures show the ML structure performs better when the number of events is small, and its performance is comparable to that of the TL structure for large N.

5. Dynamic Reorganization of the TL Structure

So that the TL structure could be included in a general simulation system, an algorithm to dynamically estimate its parameters and reorganize its structure was developed. The best algorithm is not apparent and valid comparisons of different algorithms is difficult. Three problems must be resolved.

1. How are the estimates for the TL parameters to be made?
2. When should the structure be reorganized?
3. How should the reorganization proceed?

As far as possible, we used the same methods for parameter estimation as in FRAN77b. The two levels of the TL algorithm consist of (1) an indexed list with DU entries in the index and a width of DT time units between indices and (2) sublists partitioned by keys with a maximum of NLIM events per sublist. The estimate chosen for DT is the one suggested in [VAUC75]. The mean HOLD time T, and mean event set size N are estimated and the formula $(8/(N-1)+0.07)T$ is used to estimate DT. An arbitrary decision was made to estimate DT after every fifty events scheduled. The frequency of estimation can be made a function of the stability of the estimates. The averages for T and N were computed using the exponential smoothing functions

$$T_{i+1} = T_i\alpha + T_{50}(1.0-\alpha)$$

and

$$N_{i+1} = N_i\alpha + N_{50}(1.0-\alpha)$$

where the subscript 50 indicates the value estimated over the past 50 events. For long simulations this will discount the values from the early history of the simulation. α was arbitrarily chosen at 0.96. The value of DU was computed by $DU=\max(14,2N^{1/2})$. Overestimating DU helps ensure a small overflow list and does not greatly increase the restructuring time.

We reorganize the TL structure when the value for DT differs by more than a factor of ten from its estimate. Keys are repositioned with a single pass through the list. All old dummy events are removed from the list. Dummy keys are changed to secondary keys and new dummy keys are inserted for the new value of DT. Keys are deleted from the structure if the events in their sublists can be absorbed by adjacent sublists.

6. Empirical Results - Dynamic Event Set Size

The HOLD procedure does not test how the event set algorithms will perform when the size of the list is dynamic. The following test was developed to evaluate the algorithms for this case.

For a Jacksonian network of queues, the probability that server i is active is independent of the other servers [JACK63]. If there are M queues, each with utilization ρ, the probability that N servers are active has the binomial distribution

$$\binom{M}{N} \rho^N (1-\rho)^{M-N}.$$

A straightforward simulation of such a network would have events for arrivals from sources and completions at active servers. Thus the number of completion events in the event set would have a binomial distribution. (With one source there will always be exactly one arrival event in the event set.)

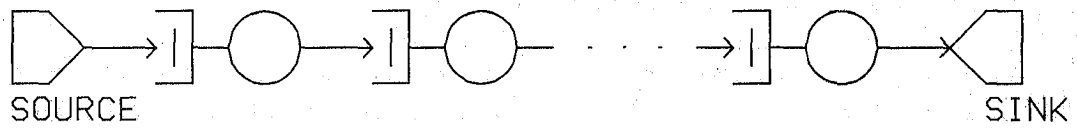

Figure 6. Tandem Network

A open tandem network of queues (figure 6) was simulated with the event set algorithm managing the arrival and completion events. The HOLD time for the tandem network is the average time to process a completion or arrival event. Because of the simplicity of the tandem network, event scheduling makes the greatest contribution to the HOLD time. The tests were performed with ρ=0.5 and M = 10, 20, 40, 100, 200, 600, 1000. (The mean and variance of the binomial distribution are ρM and $\rho(1\text{-}\rho)M$, respectively. Thus the mean numbers of completion events in the set are 5, 10, 20, 50, 100, 300 and 500, respectively, and the respective variances are 2.5, 5, 10, 25, 50, 150 and 250.) To initialize the network, a completion event was scheduled for every other queue.

The ML and TL (with code for dynamic reorganization) algorithms were programmed in PASCAL and the tests run on a Control Data 6600. (Source code for all test programs is found in MEAR79.)

The results for the tandem network are presented in figure 7 in the form of a graph of HOLD time versus M. (For the ML structure NLIM=14.) Again the ML algorithm outperforms the TL algorithm when the event set is small, and is competitive when the set is large. The slope of the graph indicates the log N performance of the ML structure.

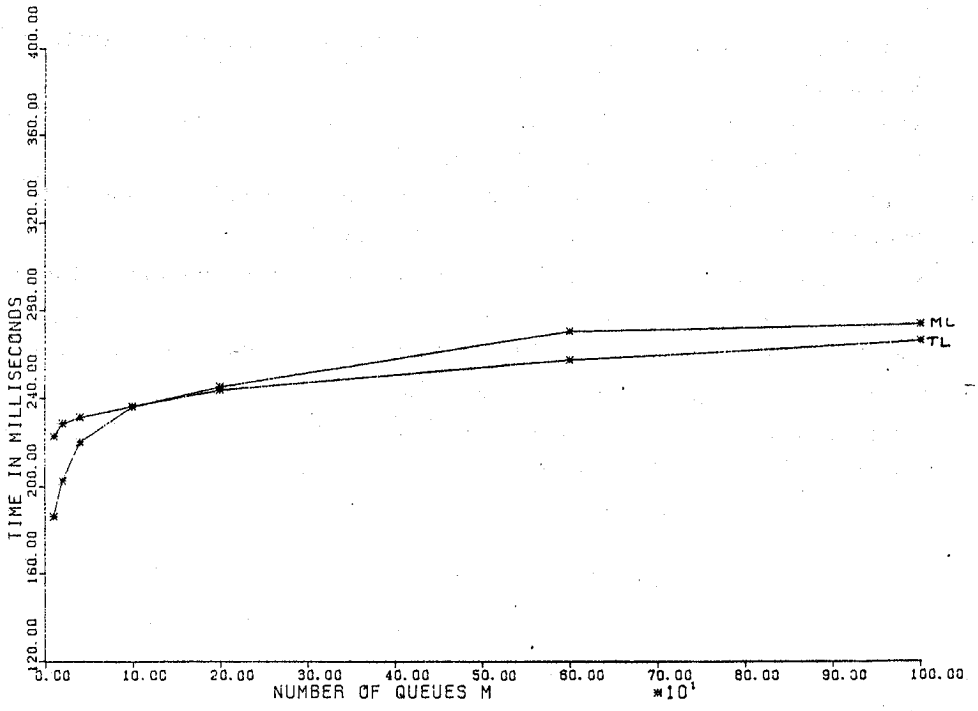

Figure 7. Tandem Network - ρ=0.5
ML Structure vs. TL Structure with Dynamic Reorganization

7. Summary

We have given a simple and robust data structure for the simulation event set. Empirical results show this structure to have at least comparable, and sometimes better, efficiency than the more complicated TL structure. The ML structure seems to be the best candidate for replacing linear structures in general simulation systems and languages.

Acknowledgement

This work was conducted while the authors were with the Department of Computer Sciences at the University of Texas at Austin.

References

FISH73 Fishman, G.S., *Concepts and Methods in Discrete Event Digital Simulation*, John Wiley and Sons, New York, 1973.

FISH78 Fishman, G.S., *Principles of Discrete Event Simulation*, Wiley, New York, 1978.

FRAN77a Franta, W.R., *The Process View of Simulation*, Elsevier North-Holland, 1977.

FRAN77b Franta, W.R. and Maly, K., "An Efficient Data Structure for the Simulation Event Set", *CACM 20*, 8, August 1977, pp. 596-602.

FRAN78 Franta, W.R. and Maly, K., "A Comparison of Heaps and the TL structure for the Simulation Event Set", *CACM*, 10, October 1978, pp. 873-875.

JACK63 Jackson, J.R., "Jobshop-like Queueing Systems", *Management Science*, 10, 1963, pp. 131-142.

JONA75 Jonassen, A. and Dahl, Ole-Johan, "Analysis of an Algorithm for Priority Queue Administration", *BIT*, 15, 1975, pp. 409-422.

KNUT68 Knuth, D.E., *The Art of Computer Programming: Vol. 3 Sorting and Searching*, Addison-Wesley, Reading Massachusetts, 1969.

MEAR79 Mear, C.E., "A Robust Data Structure for the Simulation Event Set", M.A. Thesis, University of Texas at Austin, August 1979.

SAUE80 Sauer, C.H. and Chandy, K.M., *Computer System Performance Modeling: A Primer*, to appear, Prentice Hall.

VAUC75 Vaucher, J.G., and Duval, P., "A Comparison of Simulation Event List Algorithms", *CACM*, 4, April 1975, pp. 223-230.

VAUC77 Vaucher, J.G., "On the Distribution of Event Times for the Notices in a Simulation Event Set", *INFOR*, 2, June 1977, pp. 171-182.