

AD-A136 814

OPTIMAL SEARCH FOR THE WAKE OF A MOVING TARGET WHEN
SEARCHER MOTION IS CONSTRAINED(U) NAVAL POSTGRADUATE
SCHOOL MONTEREY CA D 8 GUTHE SEP 83

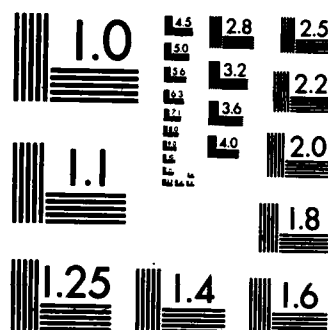
1/1

UNCLASSIFIED

F/G 20/4

NL

END



2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A136814



DTIC
JAN 13 1984
H

THESIS

OPTIMAL SEARCH FOR THE WAKE OF A MOVING
TARGET WHEN SEARCHER MOTION IS CONSTRAINED

by

Douglas Burden Guthe, Jr.

September 1983

Thesis Advisor:

J. N. Eagle

Approved for public release; distribution unlimited

DTIC FILE COPY

84 01 13 11 11

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A136814	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Optimal Search for the Wake of a Moving Target when Searcher Motion is Constrained		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1983
7. AUTHOR(s) Douglas B. Guthe, Jr.		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE September 1983
		13. NUMBER OF PAGES 64
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		Accession For NTIS GRA&I ☒ DTIC TAB ☐ Unannounced ☐ Justification ☐ By _____ Distribution/ _____ Availability Codes _____ Announced/or _____ Dist _____ A-1
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Optimal Search Constrained Searcher Discrete time and space Wake detector Moving Target		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
A method for determining the optimal or near-optimal search path for the wake of a moving target when the searcher's motion is constrained is presented. The problem uses a Markov motion model in discrete time and space for the target and assumes that the searcher is constrained to move only from the currently occupied cell j to a specified set of "neighbor cells" $I(j)$. First, a discussion of the complexity of the problem is		

#20 - Abstract - (Continued)

presented. Next, an extension of T.J. Stewart's constrained searcher algorithm is given. Stewart's algorithm uses S.S. Brown's unconstrained searcher algorithm to calculate bounds on the probability of non-detection. An extension of Brown's algorithm to allow the use of a wake detector is also given. Several alternatives to both algorithms are offered and compared. Finally, some further extensions to the algorithms are suggested.

Approved for public release; distribution unlimited.

Optimal Search for the Wake of a Moving Target
when Searcher Motion is Constrained

by

Douglas B. Guthe, Jr.
Lieutenant, United States Navy
B.S., United States Naval Academy, 1976

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN OPERATIONS RESEARCH
and
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1977

Author:

Douglas B. Guthe, Jr.

Approved by:

J.F.J.

Thesis Advisor

Professor J. J. Kelly

Second Reader

Dr. W. L. Rouse

Chairman, Department of Operations Research

Chris R. Kodur

Chairman, Department of Computer Science

K. T. Marshall

Dean of Information and Policy Sciences

ABSTRACT

A method for determining the optimal or near-optimal search path for the wake of a moving target when the searcher's motion is constrained is presented. The problem uses a Markov action model in discrete time and space for the target and assumes that the searcher is constrained to move only from the currently occupied cell j to a specified set of "neighbor cells", $I(j)$. First, a discussion of the complexity of the problem is presented. Next, an extension of T.J. Stewart's constrained searcher algorithm is given. Stewart's algorithm uses S.S. Brown's unconstrained searcher algorithm to calculate bounds on the probability of non-detection. An extension of Brown's algorithm to allow the use of a wake detector is also given. Several alternatives to both algorithms are offered and compared. Finally, some further extensions to the algorithms are suggested.

TABLE OF CONTENTS

I.	INTRODUCTION	9
A.	THE PROBLEM	9
II.	THE ALGORITHMS USED	11
A.	STEWART'S ALGORITHM	11
1.	Definition of Symbols	11
2.	Logic	12
3.	Alternatives	13
B.	BROWN'S ALGORITHM	15
1.	Definition of Symbols	17
2.	Logic	18
3.	Alternatives	19
III.	WAKE DETECTION	22
A.	BACKGROUND	22
B.	DEFINITION OF SYMBOLS	22
C.	LOGIC	23
IV.	RESULTS	27
A.	DESCRIPTION OF THE COMPUTER PROGRAM	27
B.	OPTIMALITY OF THE ALGORITHM OUTPUT	33
C.	RUN TIME COMPARISON OF THE ALGORITHMS	34
V.	EXTENSIONS	40
A.	Pd AS A FUNCTION OF WAKE AGE	40
B.	COUNTER-DETECTION	40
C.	APPROXIMATIONS TO THE OPTIMAL SOLUTION	42
VI.	CONCLUSIONS	43
A.	PROBLEM COMPLEXITY	43
B.	CHOICE OF AN ALGORITHM	43

C. WAKE SEARCH	44
D. ACKNOWLEDGEMENTS	45
APPENDIX A: DISCUSSION OF THE COMPLEXITY OF THE SEARCH PROBLEM	46
APPENDIX B: GRAPHICAL COMPARISON OF ALGORITHM RUNTIMES	50
LIST OF REFERENCES	63
INITIAL DISTRIBUTION LIST	64

LIST OF TABLES

I.	Non-Optimal Solutions	34
II.	Number of solutions for 25 Cell Grid, $T_1=2$	36
III.	Algorithm Runtimes for 25 Cell Grid, $T_1=2$	37
IV.	Number of Solutions for $T_{max}=10$, All Cases	38
V.	Algorithm Runtime for $T_{max}=10$, All Cases	39

LIST OF FIGURES

2.1	Example Search Problem	17
2.2	Sample Calculation of $P_{CS}(1)$	18
4.1	Example Deterministically Generated Problem	28
4.2	Example Randomly Generated Problem	28
4.3	Problem Generation Subroutines	29
4.4	Program Solution Subroutines	31
4.5	Example Program Output	32
5.1	Search Problem Modified for Counter-Detection	41
B.1	4 Cell Grid, $T_1=0$, and $T_{max}=1,10$	51
B.2	4 Cell Grid, $T_1=1$, and $T_{max}=1,10$	52
B.3	4 Cell Grid, $T_1=2$, and $T_{max}=1,10$	53
B.4	9 Cell Grid, $T_1=0$, and $T_{max}=1,10$	54
B.5	9 Cell Grid, $T_1=1$, and $T_{max}=1,10$	55
B.6	9 Cell Grid, $T_1=2$, and $T_{max}=1,10$	56
B.7	16 Cell Grid, $T_1=0$, and $T_{max}=1,10$	57
B.8	16 Cell Grid, $T_1=1$, and $T_{max}=1,10$	58
B.9	16 Cell Grid, $T_1=2$, and $T_{max}=1,10$	59
B.10	25 Cell Grid, $T_1=0$, and $T_{max}=1,10$	60
B.11	25 Cell Grid, $T_1=1$, and $T_{max}=1,10$	61
B.12	25 Cell Grid, $T_1=2$, and $T_{max}=1,10$	62

I. INTRODUCTION

A. THE PROBLEM

The problem considered here is the search for the wake of a moving target in discrete time and space. The target is assumed to move through a finite number of cells according to a known Markov process. The goal is to maximize the probability of detection by some specified time, t . The searcher is assumed to be partially constrained in his motion through the search area, being able to move, in one time period, from one cell to a limited number of cells in close proximity. The searcher is also assumed to have an 'n time period wake detector.' The detector is capable of detecting wakes left in a cell by the target up to n time periods earlier.

The approach taken to solve the problem was to extend the method of T.J. Stewart, [Ref. 1] and S.S. Brown, [Ref. 2]. Stewart addressed searcher employment for three cases: where the searcher's effort was infinitely divisible among all cells, only partially divisible, or totally indivisible. The three cases correspond to the situation where: searcher effort could be divided among all the possible cells during each time period, searcher effort could cover some of the cells during each time period, and searcher effort could only cover one cell during each time period. Stewart presented algorithms for finding optimal or near-optimal solutions for the first and third cases and suggested possible methods for solving the second case. This thesis generalizes the third case to allow for wake detection; i.e., the target can be detected at each time period in each of several cells through which the target has previously passed in the last n time periods.

A major difficulty of the algorithms of Stewart and Brown is that they are not guaranteed to produce the optimal solution for reasons which will be discussed later. This failure to always find optimal solutions for the indivisible effort case suggests that this problem might be fundamentally intractable.

One of the recent advances in computer science is the idea of algorithmic complexity and NP-Completeness. NP-Completeness refers to the inherent intractability or difficulty of a problem. Generally, NP-Completeness describes a set of problems which have been shown to be solvable by an efficient algorithm which is run on a non-deterministic Turing Machine. An efficient algorithm is one which will solve a problem in an amount of time proportional to a polynomial function of the length of the input needed to describe the problem. Well-known NP-Complete problems include traveling salesman problems, the knapsack problem, and integer programming problems. In Appendix 1 a more complete discussion is presented of the complexity of the search problem forming the basis for this thesis. It is suspected (but has not been proved) that this search problem is no easier than NP-Complete problems and may be NP-Hard. If so, then the investigation of heuristic or approximate solution algorithms is warranted. For further information on NP-Completeness refer to [Ref. 3].

II. THE ALGORITHMS USED

As mentioned earlier, the algorithms used to solve the search problem were: 1) R.J. Stewart's Constrained Search algorithm [Ref. 1], and 2) Brown's Unconstrained Search algorithm, [Ref. 2]. The Constrained Search Algorithm assumes the searcher is constrained in his motion during the search, i.e., the searcher can go from his current cell to some subset of all the other cells in the grid. The unconstrained searcher algorithm puts no constraints on searcher motion but does assume that the first t search cells are fixed and tries to optimize the search by choosing the cells to search from $t+1$ to T . Even in the unconstrained problem, search is limited to one cell per time period.

A. STEWART'S ALGORITHM

1. Definition of Symbols

J = set of all cells in the problem

$j \in J$ = a particular cell

$I(j)$ = matrix of all cells in J that the searcher can reach from cell j (different for each $j \in J$)

j_i = the cell to searched at time i

$\{j_0, j_1, \dots, j_t\}$ = a given search plan

$K(t, j_t)$ = matrix of all cells in J searchable at time $t+1$ from cell j_t that have not yet been considered in candidate search plans

p = probability of non-detection for the current best solution (search plan)

$S_t(i)$ = the $n \times n$ search matrix for time period t where cell i is the cell to be searched

1_j = a column vector of 1's

2. Logic

The following algorithm was taken from [Ref. 1].

1. Set $t=0$, $\hat{p}=1$; Select a cell to be searched, j_0 ; set $K(0, j_0) = I(j_0)$.
2. Solve the substitute problem (discussed below) for search over periods $t+1, t+2, \dots, T$, with the searcher at j_t prior to the search, and with searcher location in the first period (i.e. $t+1$) restricted to $K(t, j_t)$. Obtain thereby (also discussed later) a lower bound $\bar{p}$ on the optimal probability of non-detection when searcher paths are restricted to those passing through $\{j_0, j_1, \dots, j_t\}$.
3. If $\bar{p} < \hat{p}$, goto step (6). Otherwise, it is now proved that all continuations of the current path are non-optimal, i.e. the arc from $(t-1, j_{t-1})$ to (t, j_t) is fathomed. goto step (4).
4. If $t=0$, the algorithm terminates (all paths from j_0 are fathomed); the current best solution is optimal. If $t>0$, goto step (5).
5. Delete the current j_t from $K(t-1, j_{t-1})$ and set $t=t-1$. If $K(t, j_t)$ is now empty, the arc from $(t-1, j_{t-1})$ to (t, j_t) is implicitly also fathomed. Thus return to step (4). Otherwise, return to step (2).
6. Select the element of $K(t, j_t)$ appearing in the solution to the substitute problem and call this j_{t+1} ; set $t=t+1$. If $t < T$, set $K(t, j_t) = I(j_t)$ and return to step (2). If $t=T$, evaluate the probability of non-detection when the searcher path is $\{j_0, j_1, \dots, j_T\}$. If this probability is less than $\hat{p}$, then replace $\hat{p}$ by

this value and install $\{j_0, j_1, \dots, j_T\}$ as the current best solution. Return to step (5).

Essentially the algorithm solves the partially constrained problem (unconstrained after time $t+1$) at each level to obtain a lower bound on the probability of non-detection for any fully constrained searchers that have the same initial search $\{j_0, \dots, j_{t+1}\}$. The algorithm then traverses each branch whose lower bound is less than the current best solution probability. When $t=T$, the search is now fully constrained; and, if its probability of non-detection $\bar{p}$ is less than $\hat{p}$, then it is a better search plan and is saved. When no branches with probabilities less than the current best plan are left, the problem is completed and the current best plan is optimal.

Optimality is guaranteed if the lower bounds obtained from the application of Brown's algorithm are indeed lower bounds. Unfortunately, due to the nature of the problem, (an integer programming problem) the results of Brown's algorithm are not necessarily lower bounds. A search plan may exist whose "lower bound", from Brown's algorithm is higher than the current best solution yet whose actual probability of non-detection for a fully constrained solution is lower than the current best solution. Thus the branch containing the optimal solution may be pruned. Therefore, the solution produced by Stewart's algorithm using Brown's algorithm as a bound will not necessarily be optimal. Nevertheless, the solution might be close enough to the optimal for practical problems of interest.

3. Alternatives

Stewart's algorithm is essentially a depth-first search through a tree of possible solutions. It uses Brown's algorithm to bound the branches and remove those branches which do not contain better solutions than the current one.

The depth-first search strategy generates bounds at each level of the tree and picks the best bound at that level to continue the bounding procedure. In this way it reaches a fully constrained search in T steps as it progresses down through the tree. After reaching the first constrained solution the algorithm progresses back up the tree checking other branches on the way up. The algorithm stops when it has progressed all the way up to the top of the tree and has no bounds which are better than the current best solution.

Another approach is to use a best-first strategy. In this strategy, the best of all the bounds currently calculated is chosen as the branch to further investigate. After the new bounds are calculated for the one step investigation, the best bound is again chosen to investigate. As the best bound is chosen it is checked to see if it meets the motion constraints. If the solution does meet the constraints then the algorithm stops with the best solution. Using the best-first strategy presented by D.R. Smith in [Ref. 4], Stewart's algorithm was revised and is presented below:

1. Set $t=0$, $\hat{p}=1$; select j_0 to be the initial search cell, set $K(0, j_0) = I(j_0)$.
2. Solve the substitute problems (discussed earlier) for search over periods $t+1, t+2, \dots, T$, with searcher at j_t prior to the search, and with searcher location in the first period (i.e. $t+1$) restricted to $K(t, j_t)$. Store the lower bound on the optimal probability of non-detection for each of the elements of $K(t, j_t)$ in a priority queue based on the lower bound. (i.e. store the search $\{j_0, \dots, j_{t+1}\}$, the length of the constrained part of the search, $t_c=t+1$, and the lower bound for each element of $K(t, j_t)$).
3. Select from the priority queue the search with the smallest lower bound and set the current search to that chosen search. Set $t=t_c$, set $K(t, j_t) = I(j_t)$.

4. Check the current search to see if it meets the movement constraints (i.e. check for feasibility). If feasible, then stop, the current search is the best search. Otherwise, goto step (2).

Essentially what the best-first search of the tree does is generate bounds for each branch below the root. Then it picks the branch with the best bound to continue the investigation of the tree. After bounds are generated for each branch they are saved on a priority queue from which the selection of the best bound is made. A priority queue (in the implementation, a priority heap) is used to minimize the time and storage requirements necessary to store the bounds and find the best bound. Then the algorithm picks the branch with the best bound to continue the investigation of the tree. Once the best bound is found, it is checked to see if it conforms to the movement constraints. If it does conform, the algorithm stops, the optimal solution has been found. Otherwise, the branches emanating from the branch under investigation are bounded and the algorithm repeats.

D.R. Smith, in [Ref. 4], argued that the best-first search of most classes of random trees has a strictly smaller expected time and space complexity. Therefore, it was decided to compare the best-first and depth-first tree search algorithms for both use of computer time and optimality of results. The best-first search, however, is also plagued by the possible non-optimality of Brown's algorithm.

B. BROWN'S ALGORITHM

To generate a lower bound on the constrained search given that the search during time periods $1, \dots, t$ is fixed, an algorithm presented by S.S. Brown in [Ref. 2] was used.

The target position distribution was generated using an initial probability density of target location P_0 and a

Markovian motion model. P_0 , the target initial distribution, is a vector with dimension equal to the number of cells in the spatial grid. Markovian motion is described by a transition matrix T that contains the probability of moving to cell j given that the target was in cell i . After one time period, the new target location density is $P_0 \cdot T$. After two time periods it becomes $P_0 \cdot T^2$, and after n moves, it becomes $P_0 \cdot T^n$. To implement search in the model, a search matrix $S_t(i)$ for time period t was used in the form of a diagonal matrix with the i th diagonal element given by the probability of a missed detection, and the other diagonal elements equal to unity. Therefore the vector $P_0 \cdot S_1(1)$ contains the probabilities that the target is in each of the cells and remains undetected after a one time-unit search in cell 1. We refer to this vector as "the defective target location mass at time period 1". Figure 2.1 illustrates an example search problem. Figure 2.2 illustrates a sample calculation of $P_0 \cdot S_1(1)$. A four time-unit search in cells 1, 2, 4, and 3 would yield a defective target location mass, $P_t(i)$ given by :

$$P_t(i) = P_0 \cdot S_1(1) \cdot T \cdot S_2(2) \cdot T \cdot S_3(4) \cdot T \cdot S_4(3) \quad (2.1)$$

The final probability of non-detection p_{nd} for the search is given by:

$$p_{nd} = P_t(i) \cdot 1] = 0.612 \quad (2.2)$$

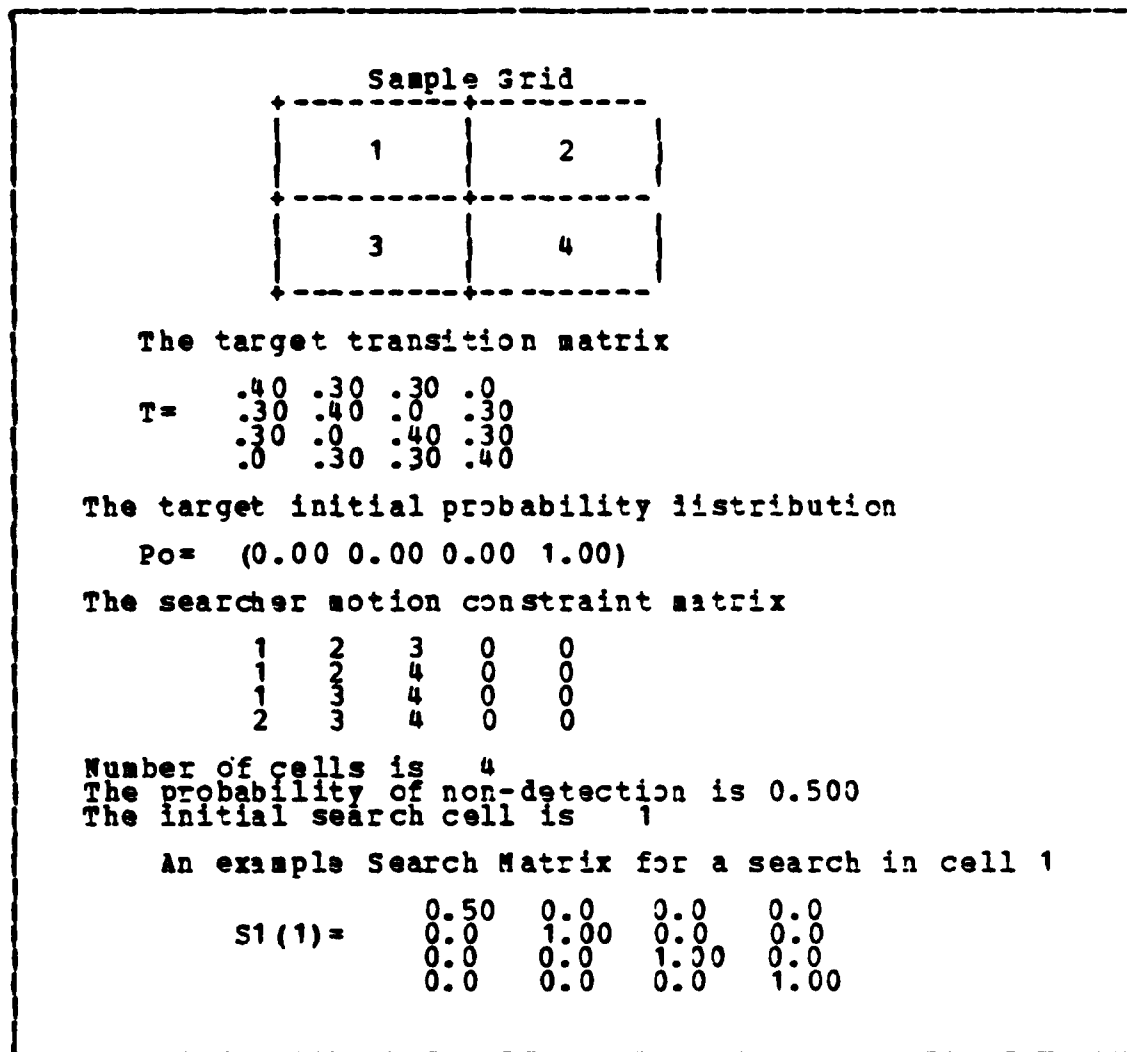


Figure 2.1 Example Search Problem.

1. Definition of Symbols

S_j = The search plan chosen after j iterations of Brown's algorithm

$S_j(t)$ = The cell to be searched in time period t in the search plan S_j

$P_{nd}(S_j)$ = The final probability of non-detection associated with the search plan S_j

$$\begin{aligned}
 P_0 \cdot S_1(1) &= (.00 \ .00 \ .00 \ 1.00) \cdot \begin{pmatrix} .5 & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \end{pmatrix} \\
 &= (.00 \ .00 \ .00 \ 1.00)
 \end{aligned}$$

Figure 2.2 Sample Calculation of $P_0 \cdot S_1(1)$.

2. Logic

1. Given an initial guess S_0 at a search plan, and given a solution tolerance $\epsilon > 0$, set $t=1$, $j=1$, and $S_1=S_0$.
2. Choose $S_j(t)$ to minimize $pnd(S_j)$, ($S_j(i)$ is fixed in this minimization for $i \neq t$.) The minimization is done by calculating $Pnd(S_j)$ for all possible choices of $S_j(t)$ and choosing the $S_j(t)$ which has the smallest $Pnd(S_j)$.
3. If $t=T$, goto step 4. Otherwise, increment t and goto step 2.
4. If $Pnd(S_{j-1}) - Pnd(S_j) < \epsilon$, stop. $Pnd(S_j)$ is the desired bound. Otherwise, increment j , set $t=1$, set $S_j=S_{j-1}$, and goto step 2.

Brown's algorithm is an iterative improvement algorithm, where at each time step t the cell which minimizes Pnd is chosen, given that the rest of the search before and after t is unchanged. When the Pnd for S_j is within ϵ of the Pnd for S_{j-1} , the algorithm is terminated and search S_j is used.

Brown's algorithm in the totally divisible case, where at each time period t the solution to the stationary search problem is found holding the searches for time periods before and after t fixed, will converge to the optimal solution as proved in [Ref. 2]. However, because

the implementation here only allows search in one cell during each time period the solution space becomes discontinuous and thus non-convex. Therefore, the algorithm is not guaranteed to converge to an optimal solution.

3. Alternatives

Brown's algorithm can be considered to find the local optimal solution given a starting solution, S_0 . Therefore, the total solution space is partitioned by Brown's algorithm into those starting solutions which result in the optimal solution (the global optimum) and those starting solutions which result in non-optimal solutions (local but not global optimums.) Based on this partitioning idea several alternatives are available.

The first is to choose a good starting solution. If the optimal solution were used as the starting solution, Brown's algorithm would always arrive at the optimal solution. However, the optimal solution is not known, otherwise Brown's algorithm would be unnecessary. Two approaches were used to guess a good starting solution: 1) the myopic solution, $S_0(t)$ is chosen to give the most improvement to $Pnd(S_0)$; and 2) a random solution, a random number generator is used to generate random starting solutions.

As mentioned in [Ref. 2], if a zero solution is used to start Brown's algorithm then the myopic solution will result after the first iteration. The myopic solution seems like a reasonable choice to start the algorithm. On the other hand, there might be some negative correlation between the optimal solution and the myopic solution, i.e. the myopic solution may lie in another partition which does not lead to the optimal solution. Therefore the random starting solution was also considered as a means to get around any possible negative correlation. Since the random solution has a finite probability of choosing the optimal solution it may have a better chance of starting in the right partition.

Another approach used was to restart Brown's algorithm any time a change was made to the current solution. The revised algorithm is presented below:

1. Given an initial guess S_0 at a search plan set $t=1$, $j=1$, and $S_1=S_0$.
2. Choose $S_1(t)$ to minimize $Pnd(S_j)$, ($S_1(i)$ is fixed in this minimization for $i \neq t$.) The minimization is done by calculating $Pnd(S_1)$ for all possible choices of $S_1(t)$ and choosing the $S_1(t)$ which has the smallest $Pnd(S_1)$.
3. If $S_1(t) \neq S_0(t)$ then set $S_0(t) = S_1(t)$ and goto step 1.
4. If $t=T$, goto step 5. Otherwise, increment t and goto step 2.
5. Stop. $Pnd(S_1)$ is the desired bound.

This variation of Brown's algorithm takes an initial solution and starts at the first time period checking for local optimality for that time period. If the choice of cell to be searched at the current time period is locally optimal for that time period then the algorithm goes on to the next time period. Otherwise, the algorithm inserts the new search cell and starts back at the first time period. When the algorithm goes all the way through the solution without changes it stops. This approach may repartition the solution space increasing the size of the partition which leads to the optimal solution.

Another approach which was not used is to run the algorithm several times with different starting solutions S_0 . Then the lower bound can be taken as the minimum of the locally optimal solutions returned by the algorithm. Also, the lower bound can be estimated by

$$est = (P95 \cdot P5 - P32.5^2) / (P95 + P5 - 2 \cdot P32.5). \quad (2.3)$$

P5, P95 and P32.5 are the 5TH, 95TH and 32.5TH percentile values. This estimate is based on the idea that, as the number of independent solutions produced by a given algorithm for a given problem increase, the solution values converge to a Weibull distribution where the optimal lower bound is the location parameter. This approach was not used because it required running Brown's algorithm two or more times to calculate each bound and it was felt that the additional computer time would not improve the bounding process significantly.

A final approach which also was not investigated, is to locally optimize the choice of cells over 2, 3 or more time periods in the Brown algorithm. This again might increase the size of the partition which leads to the optimal solution.

III. WAKE DETECTION

A. BACKGROUND

As mentioned earlier, this thesis is an attempt to extend existing optimal search theory techniques to the problem of wake detection. Stewart's and Brown's algorithms, as presented in the last chapter, indicate one method for solving the constrained searcher problem. The next question is: "How does one extend the algorithms to handle the possibility of wake detection?"

Stewart's algorithm does no actual calculation of the probability of non-detection, it uses Brown's algorithm for the calculation of bounds. Therefore Brown's algorithm is the one that needs to be altered to allow for use of a wake detector. To extend the algorithm we proceed as follows:

B. DEFINITION OF SYMBOLS

P_0 = initial target probability distribution (i.e., the probability that the target is in any cell i at the start of the problem.)

$S_t(i)$ = an $n \times n$ matrix which reduces the defective probability mass of the target by $1 - P(d)$ in cell i searched at time t . The defective probability mass of the other cells remains unchanged.

T = a $n \times n$ Markov transition matrix which contains the probability that a target in cell i at time period t will transition to cell j at time period $t+1$.

1_j = a $n \times 1$ column vector of 1's.

C. LOGIC

What needs to be extended is the model under which the non-detection probability is calculated. As shown earlier, this calculation for a non-wake detector in a 4 time unit search is

$$P_{nd} = P_o \cdot S_1 \cdot T \cdot S_2 \cdot T \cdot S_3 \cdot T \cdot S_4 \cdot 1]. \quad (3.1)$$

It is now proposed to allow the searcher to carry an 'n-time unit wake detector.' This detector has the capability of detecting wakes in the cell being searched which were made by the target up to n time periods earlier. To extend the model to handle 'n-time unit wake detectors' two assumptions are made.

First, it is assumed that each wake search for each time period is independent of all other searches. For example, when a searcher searches cell 2 at time period 3 with a 2 time unit wake detector he is completing three separate and independent searches. The first is of cell 2 looking for the wake made by the target during time period 3, the second is of cell 2 looking for the wake made during time period 2, and the third is of cell 2 looking for the wake made during time period 1. Note the searcher has also completed a search during time periods 2 looking for wakes made during time periods 1 and 2, and a search during time period 1 looking for wakes made during time period 1. Therefore, in a 3 time period search with a two time period detector the searcher has made a total of 6 independent searches in 3 cells, the cells he chose to search at time periods 1, 2, and 3.

Also note that the target is assumed not to have left a wake prior to the start of the problem for ease of computer

calculation and formulation of the problem. This does not restrict the result.

The second assumption is that we are trying to calculate the probability of non-detection after the specified length of search is completed. We are not trying to evaluate the probability of non-detection at intermediate points in the search.

Based on those two assumptions, the wake detection concept can be considered a simultaneous search of the cell the searcher is in now, the cell the searcher will be in next time period, the cell the searcher will be in two time periods from now, etc., out to the maximum number of time periods the wake is detectable or to the number of time periods remaining in the problem, whichever is smaller. The wake search is modeled as if an equivalent search is conducted simultaneously with the real-time search, i.e. searching for a wake in cell j which is t time periods old is equivalent to searching in cell j , t time periods previous, simultaneously with the other searches being conducted t time periods previous and waiting t time periods to discover the results of the search.

As shown previously, the probability of non-detection for the search without wake detection can be calculated from the product of a series of matrices as shown in equation 3.1. For a search in the same cells but with a one time-unit wake detector, the probability of non-detection is

$$P_{nd} = P_0 \cdot S_1 \cdot S_2 \cdot T \cdot S_2 \cdot S_3 \cdot T \cdot S_3 \cdot S_4 \cdot T \cdot S_4 \cdot 1]. \quad (3.2)$$

The vector P_0 , as defined earlier, is the probability that the target is in cell i at the start of the problem. The vector $P_0 \cdot S_1$ is the probability the target was in cell i at time period 1 and has not been detected by the wake

search conducted during time period 1 looking for a wake left in time period 1. The vector $Po \cdot S1 \cdot S2$ is the probability the target was in cell i at time period 1 and has not been detected by wake searches during time periods 1 and 2 looking for wakes created in time period 1. The vector $Po \cdot S1 \cdot S2 \cdot T$ is then the probability that the target is in cell i at time period 2 and has not been detected by wake searches looking for wakes created during time period 1.

Following the same line of reasoning, the vector $Po \cdot S1 \cdot S2 \cdot T \cdot S2 \cdot S3 \cdot T \cdot S3 \cdot S4$ is the probability that the target is in cell i at time period 3 and has not been detected by wake searches looking for wakes created during time period 3 and prior. Of note is the idea that the searcher in effect conducts simultaneous searches of 2 cells during a given time period but does not receive the results of the one time unit wake search until the next time period.

The wake detection problem is analagous to the problem where a homeowner thinks there are mice in his house and he wants to confirm his suspicions. Each room of the house has two tape recorders: one which is available for replay of what noises were recorded in the previous hour and the other which is recording the current hour. The owner can only go to adjacent rooms which have doors to the current room. He spends one hour in the room of his choice listening for mice. Once in a room he can listen for mice and at the same time listen to the tape recording of the previous hour. The probability of detecting the mice on the tape recorder or by listening in a room is the same. The owner now must decide how to search the house to maximize his chances of finding the mice.

The tape recordings of the noise in each room are essentially independent and simultaneous searches of each room in the house. The owner can only determine the results of the search by tape recorder if he enters the room and listens to

the tape recorder. If the owner doesn't listen to the tape recorder during the hour the second recorder is taping the room, the first recorder is rewound and starts taping thus destroying the results of that particular search.

The key assumptions, as stated earlier, are that the searches are independent and that the goal is to minimize the probability of non-detection over a given search length. Therefore the matrix multiplication in equation 3.2 is valid. For a two time unit detector, the probability of non-detection is

$$P_{nd} = P_o \cdot S_1 \cdot S_2 \cdot S_3 \cdot T \cdot S_2 \cdot S_3 \cdot S_4 \cdot T \cdot S_3 \cdot S_4 \cdot T \cdot S_4 \cdot 1]. \quad (3.3)$$

The matrix multiplication can be extended for a T1 capacity wake detector where T1 is the time late a searcher can enter a cell and still detect the target's wake. For the T1 capacity detector equation 3.2 becomes

$$P_{nd} = P_o \cdot \overline{ST_1} \cdot T \cdot \overline{ST_2} \cdot T \cdot \overline{ST_3} \cdot T \cdot \overline{ST_4} \cdot \dots \cdot \overline{ST_{max}} \cdot 1]. \quad (3.4)$$

where

$$\overline{ST_i} = \prod_{j=i}^{\min(T_{max}, i+T_1)} St(j) \quad (3.5)$$

IV. RESULTS

A. DESCRIPTION OF THE COMPUTER PROGRAM

The algorithms presented in the preceding chapters were implemented on the U.S. Naval Postgraduate School's IBM 3033 computer system in Fortran H (extended). The program was constructed of a main driver program which called various subroutines for input, calculation and output.

The operation of the program can be broken down into two main functions: 1) generation of the problem including the initial target distribution, the transition matrix and the searcher movement constraint matrix; and 2) solution of the generated problem.

The generation of the problem used two methods. In one method the problem generated was a deterministic one where the target's initial position was at the far corner away from the searcher who started in cell 1. The target transition matrix was generated assuming that the target had a fixed probability of staying in the cell it currently occupied and the remaining probability of movement was divided evenly among the cells which were adjacent. Diagonal movement was not allowed. Figure 4.1 shows a example initial target distribution and target transition matrix for the deterministic case. The searcher movement constraint matrix was generated by allowing the searcher to move only from the current cell to an adjacent cell.

In the second method, the problem was randomly generated. The target was randomly distributed between all the cells at the start. The target transition matrix was generated assuming a fixed probability of remaining in the current cell and then the remaining probability of movement

$$P_0 = (0.00 \ 0.00 \ 0.30 \ 1.00)$$

$$T = \begin{matrix} & .40 & .30 & .30 & .30 \\ & .30 & .40 & .00 & .30 \\ & .30 & .00 & .40 & .30 \\ & .00 & .40 & .30 & .40 \end{matrix}$$

Figure 4.1 Example Deterministically Generated Problem.

was randomly distributed among the adjacent cells. The searcher movement constraint matrix was determined as

$$P_0 = (0.23 \ 0.34 \ 0.19 \ 0.24)$$

$$T = \begin{matrix} & .40 & .35 & .25 & .00 \\ & .17 & .40 & .00 & .43 \\ & .43 & .00 & .40 & .17 \\ & .00 & .25 & .35 & .40 \end{matrix}$$

Figure 4.2 Example Randomly Generated Problem.

before. Figure 4.2 illustrates a typical random problem. Figure 4.3 shows the program structure and subroutines used to generate the deterministic and random problems. The program requires 6 inputs as defined below:

1. Size of the grid i.e. 3x3, 4x4, etc.
2. Probability of detection of the target if the searcher and target are in the same cell. Also used as the probability of detection of the target using the wake detector.
3. Fixed probability of the target staying in the cell it currently occupies. (i.e. T_{ii})

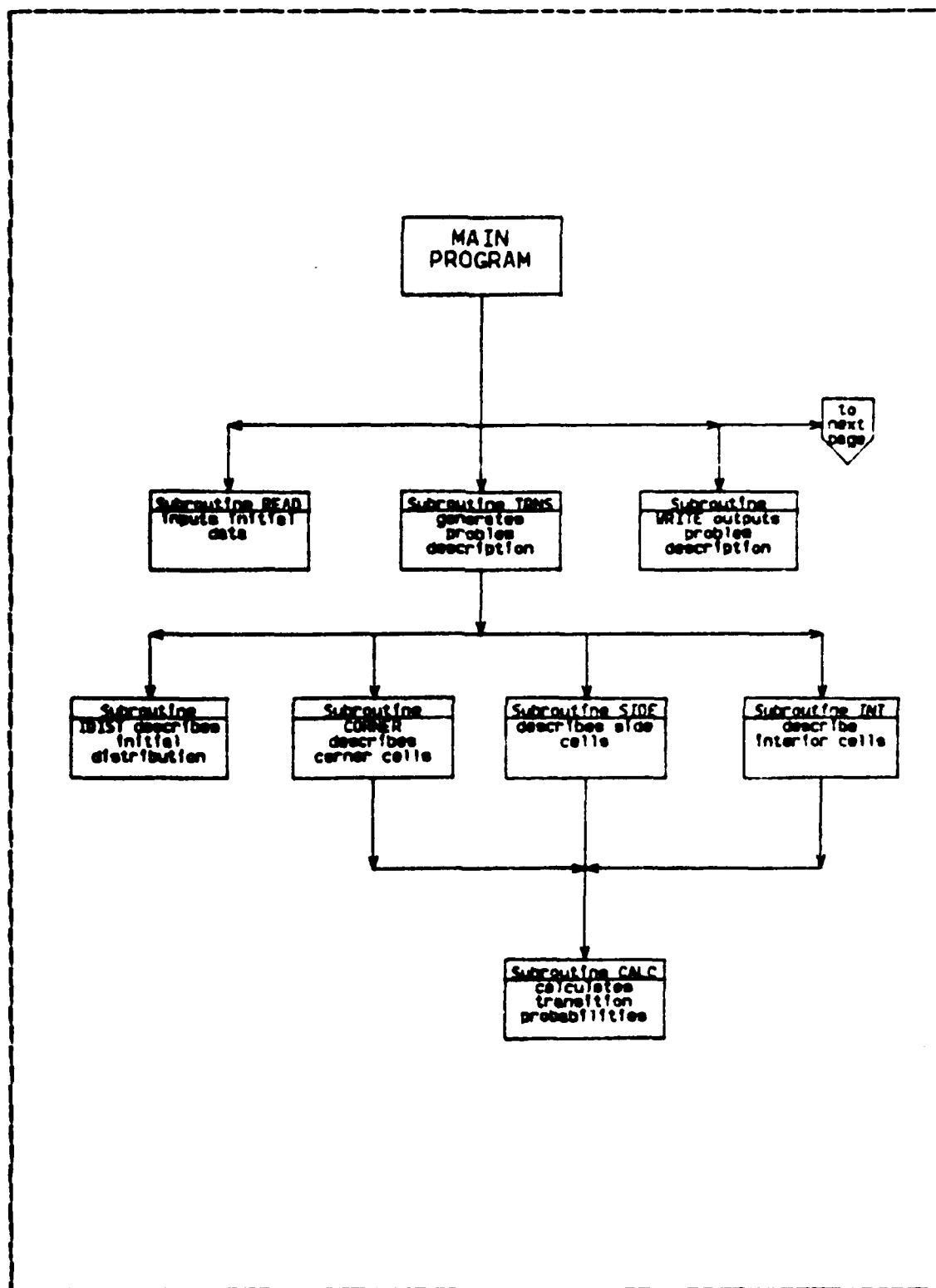


Figure 4.3 Problem Generation Subroutines.

4. Whether the generated problem is to be deterministic or random.
5. Epsilon, used as a stopping criteria for Brown's algorithm.
6. The capability of the wake detector, (i.e. how many time units late will the detector detect a wake.)

Once the problem was generated it was then necessary to find the optimal solution using the various algorithms mentioned earlier. Seven algorithms were used to find the solution:

1. Depth-First search using the myopic solution to start Brown's algorithm.
2. Depth-First search using a randomly chosen solution to start Brown's algorithm.
3. Depth-First search using a random solution to start Brown's algorithm and using the modification which restarted Brown's algorithm every time the solution was changed.
4. Best-First search using the myopic solution to start Brown's algorithm.
5. Best-First search using a randomly chosen solution to start Brown's algorithm.
6. Best-First search using a random solution to start Brown's algorithm and using the modification which restarted Brown's algorithm every time the solution was changed.
7. Total enumeration of all possible search solutions to determine the one with the lowest probability of non-detection.

The only algorithm guaranteed to produce the optimal solution was the total enumeration algorithm which generated all possible feasible solutions. Figure 4.4 shows the program structure and subroutines necessary to implement the algorithms and problem solution part of the program.

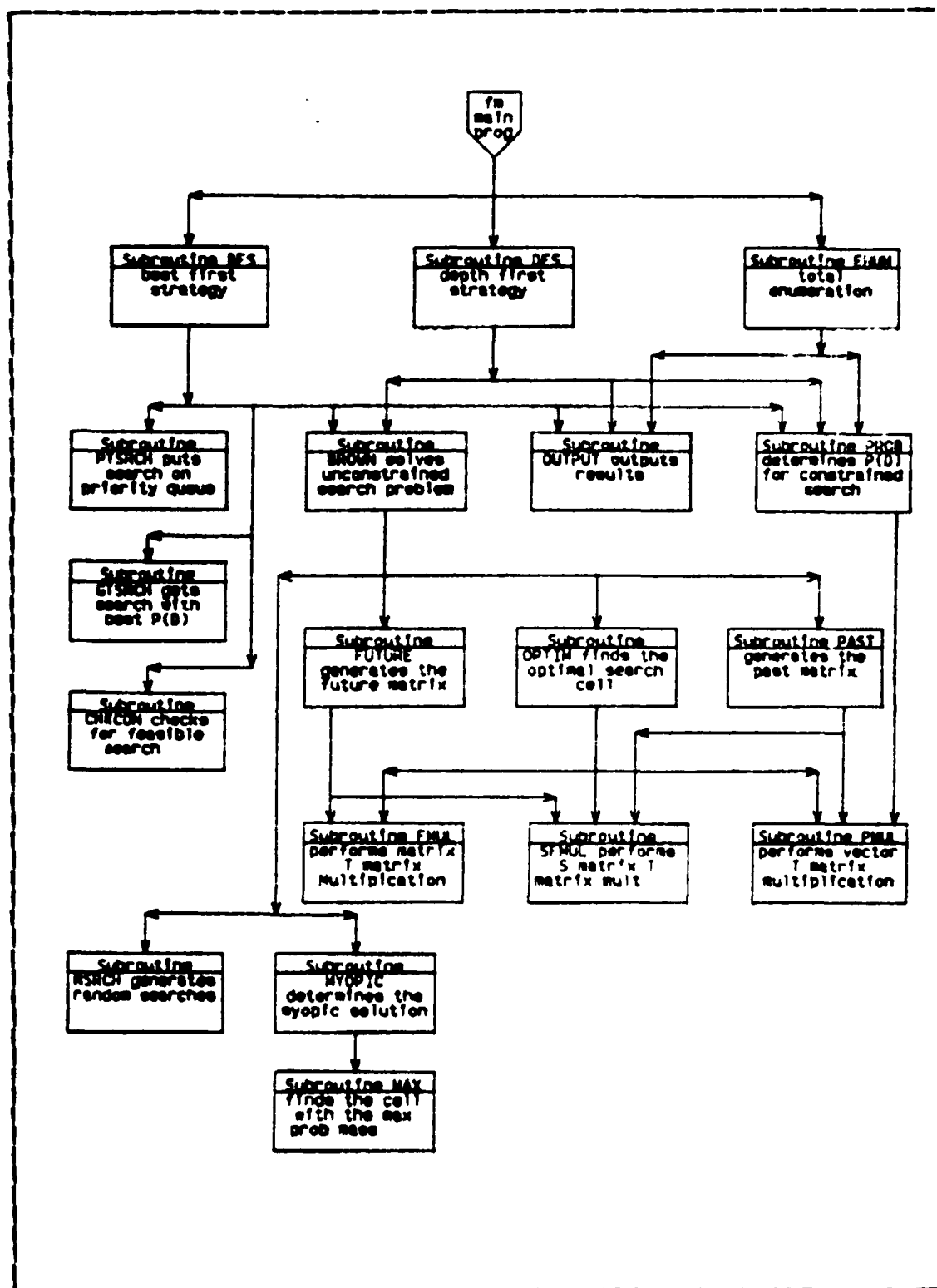


Figure 4.4 Program Solution Subroutines.

The output generated by the program included:

1. cpu seconds used by each algorithm
2. number of unconstrained solutions checked
3. number of constrained solutions checked
4. probability of detection of the target after search
5. search plan used

Figure 4.5 presents an example program output.

THE TARGET TRANSITION MATRIX

```
.40 .30 .30 .0
.30 .40 .0 .30
.30 .0 .40 .30
.0 .30 .30 .40
```

THE TARGET INITIAL PROBABILITY DISTRIBUTION

```
0.0 0.0 0.0 1.00
```

THE SEARCHER MOTION CONSTRAINT MATRIX

```
1 2 3 0 0
1 2 4 0 0
1 3 4 0 0
2 3 4 0 0
```

NUMBER OF CELLS IS 4

THE PROBABILITY OF NON-DETECTION IS 0.500

THE INITIAL SEARCH CELL IS 0

THE ALGORITHMS USED ARE:

- 1 - DEPTH FIRST STRATEGY, MYOPIC INITIAL SEARCH
- 2 - DEPTH FIRST STRATEGY, RANDOM INITIAL SEARCH
- 3 - DEPTH FIRST STRATEGY, RESTART AFTER EACH CHANGE
- 4 - BEST FIRST STRATEGY, MYOPIC INITIAL SEARCH
- 5 - BEST FIRST STRATEGY, RANDOM INITIAL SEARCH
- 6 - BEST FIRST STRATEGY, RESTART AFTER EACH CHANGE
- 7 - TOTAL ENUMERATION

SEARCH PLANS FOR TMAX= 4 AND TL= 0

ALG	TIME	#UNC	#CON	P(D)	SEARCH PLAN
1	0.02	6	3	0.3900	1 2 4 4
2	0.01	6	3	0.3900	1 2 4 4
3	0.02	6	3	0.3900	1 2 4 4
4	0.02	3	0	0.3900	1 2 4 4
5	0.02	3	0	0.3900	1 2 4 4
6	0.01	3	0	0.3900	1 2 4 4
7	0.01	0	27	0.3900	1 2 4 4

Figure 4.5 Example Program Output.

Using the program described above, grid sizes of 4, 9, 16, and 25 cells were investigated. All seven algorithms were tested on deterministic problems with $T_{max}=2, \dots, 10$ and $T_1=0, 1, 2$. While the problem sizes considered are not large enough for real life problems it was felt that they were large enough to demonstrate the algorithms. The results can then be extrapolated to determine expected running times for real life problems. The goal was to answer two questions:

1. How close to the optimal are the solutions from the branch-and-bound algorithms?
2. How do the running times of the algorithms compare; to each other, and to the total enumeration algorithm?

B. OPTIMALITY OF THE ALGORITHM OUTPUT

As mentioned earlier, the problems under consideration ranged from a 4 cell grid with $T_{max}=2$ and $T_1=0$ to a 25 cell grid with $T_{max}=10$ and $T_1=2$. The total number of possible constrained solutions ranged from 3 to 275905. Table I shows the number of non-optimal solutions and the maximum percent difference between the optimal and non-optimal solutions for each of the algorithms. Since the problems were symmetric there were several optimal solutions, so the branch-and-bound solution could be an equivalent although different solution from the total enumeration solution. This may have increased the probability of the branch-and-bound algorithms finding the optimal solution. Further investigation into larger random problems may yield more non-optimal solutions and larger percent differences, however, fully 96% of Stewart's results, as reported in [Ref. 1], were optimal with the non-optimal results being within .41% of the optimal solution. Therefore, the results of this run tend to confirm Stewart's results concerning the optimality of the branch-and-bound algorithm solutions.

TABLE I
Non-Optimal Solutions

	1	2	Algorithms		5	6
			3	4		
Number of Non-optimal Solutions	2	0	0	2	1	0
Maximum percent Error	.04%	0%	0%	.04%	.01%	0%

C. RUN TIME COMPARISON OF THE ALGORITHMS

Since all the algorithms seem to be able to find the optimal or near-optimal solution without too much difficulty, it is now important to determine which algorithm performs the fastest. While the total enumeration algorithm was used to provide a benchmark for the optimality of the solutions it can also serve as a benchmark for the run-time of the algorithms.

The running time of the total enumeration algorithm is a function of the number of possible feasible solutions. The number of feasible solutions is based on how many places the searcher can go from the cell he currently occupies. From each cell the searcher can transition to between 3 and 5 cells. If the searcher is in a corner cell he can stay where he is or transition to either of the two cells which border the cell he is in. If the searcher is in a side cell he can stay where he is or transition to any of the three border cells. If the searcher is in an interior cell he can again stay where he is or transition to any of the four border cells. Therefore the total number of feasible solutions is bounded below by the exponential function $3^{T_{\max}}$, where $T_{\max}$

is the length of the search. The number of feasible solutions is bounded above by the exponential function $5^{T_{\max}}$. Since the total enumeration algorithm looks only the feasible solutions and calculates Pnd for each one the algorithm is therefore $O(e^{T_{\max}})$ or of exponential complexity.

The branch-and-bound algorithms don't lend themselves to such easy analysis. The worst case analysis would be where the bounds are so weak that all the feasible solutions would have to be checked. The average case analysis is much more difficult. Smith in [Ref. 4], argued that the best-first strategy had a smaller average case complexity than the depth-first strategy. Since it is unclear whether the trees generated by the search problem fit into the class of random trees covered by Smith's argument, further complexity analysis will not be attempted.

Appendix B contains graphs of the run-time for all seven algorithms for all the cases. From the graphs it is clear that the best-first strategy did outperform the depth-first strategy especially as the problem (as measured by the number of possible feasible solutions) increased in difficulty. This is supported by analyzing the number of solutions checked by each algorithm. Table II shows the number of solutions checked by each algorithm for the 25 cell grid with $T_1=2$.

From table II it appears that until time period 6 the algorithms looked at all the possible solutions. Since the total enumeration algorithm didn't look at any intermediate solutions it was faster. After time period 6 the branch-and-bound algorithms were able to use the pruning feature to a great extent in removing unpromising branches. Table III shows the runtimes needed to used in generating the solutions in table II. Based on the number of solutions calculated by each algorithm it appears that the best-first strategy is more efficient. Since the best-first strategy

TABLE II
Number of solutions for 25 Cell Grid, Tl=2

PARTIALLY CONSTRAINED SOLUTIONS
(Brown's Algorithm Output)

Tmax	1	2	3	Algorithm 4	5	6	7
2	0	0	0	0	0	0	0
3	3	3	3	3	3	3	0
4	14	14	14	14	14	14	0
5	57	57	57	57	57	57	0
6	234	234	234	234	234	234	0
7	983	983	983	983	983	983	0
8	2116	2109	2402	1123	1123	1123	0
9	3988	4215	2309	1573	1593	1598	0
10	2787	4066	4532	1375	1359	1383	0
Totals	10182	11591	10534	5362	5366	5395	0

CONSTRAINED SOLUTIONS

Tmax	1	2	3	Algorithm 4	5	6	7
2	3	3	3	3	3	3	3
3	11	11	11	11	11	11	11
4	43	43	43	43	43	43	43
5	177	177	177	177	177	177	177
6	749	749	749	749	749	749	749
7	916	563	386	100	100	100	3235
8	1273	987	1439	0	0	0	14139
9	601	1552	723	160	160	160	62309
10	246	251	765	0	0	0	275905
Totals	4019	4336	4296	1243	1243	1243	356571

algorithms had almost exactly the same number of solution calculations whereas the depth-first strategy algorithms were more divergent it could be hypothesized that the best-first strategy is more tolerant of the non-optimal bounds produced by Brown's algorithm.

Table IV shows how the number of solutions varied for all the cases where Tmax=10 (i.e. where the maximum number of possible feasible solutions was noted.)

TABLE III
Algorithm Runtimes for 25 Cell Grid, T1=2

Tmax	1	2	3	Algorithm 4	5	6	7
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.2	0.2	0.1	0.2	0.2	0.1	0.0
4	1.2	1.2	0.9	1.2	1.2	0.8	0.1
5	5.8	6.6	4.5	5.7	6.4	4.6	0.3
6	26.5	27.8	21.5	26.2	27.3	22.2	1.8
7	112.3	118.3	92.6	106.0	113.5	87.8	9.3
8	331.4	342.6	344.4	240.9	253.3	278.9	48.2
9	821.4	868.2	794.5	531.8	547.5	743.6	248.1
10	1253.4	1500.3	2252.4	843.0	845.0	1494.1	1249.1
Totals	2552.2	2865.2	3510.9	1755.0	1794.4	2632.1	1556.9
runtime in cpu seconds							

Table V tabulates the running time required to generate the solutions for the cases where Tmax=10. From the tables it can be seen that the best-first strategy dominated the depth-first strategy, mainly because it had to look at fewer solutions. Also it can be seen that the restart algorithm was not competitive when Tmax=10 for any of the cases. This reflects the fact that it must restart after every change in the current solution and as Tmax increases the number of changes also increases. Therefore the restart algorithm is forced to restart significantly more frequently and thus takes longer to arrive at the local bound. From table IV it is seen that the extra time taken to calculate the bound did not reduce the number of solutions which were investigated.

It is apparent that the best-first strategy, using either the myopic or random starting solution, performed very well when compared to any of the other algorithms. As expected, the runtime for the total enumeration algorithm

TABLE IV
Number of Solutions for T_{max}=10, All Cases

PARTIALLY CONSTRAINED SOLUTIONS (Brown's Algorithm Output)								
Grid Size	Tl	1	2	3	Algorithms 4	5	6	7
4	0	978	1092	1011	912	894	885	0
	1	24	24	36	27	27	24	0
	2	39	24	51	51	33	33	0
9	0	628	547	279	286	223	275	0
	1	247	218	213	121	81	80	0
	2	270	163	306	186	151	141	0
16	0	1413	1002	1029	740	718	723	0
	1	910	419	419	332	409	400	0
	2	347	891	380	320	339	373	0
25	0	4843	3481	4013	1343	1343	1343	0
	1	3371	3798	1266	1295	1283	1313	0
	2	2787	4065	4532	1375	1359	1383	0
Totals		15857	15665	13535	6988	6860	6973	0
TOTALLY CONSTRAINED SOLUTIONS								
Grid Size	Tl	1	2	3	Algorithms 4	5	6	7
4	0	198	390	375	189	195	195	19683
	1	3	3	6	3	3	3	19683
	2	3	3	6	3	3	3	19683
9	0	31	27	10	0	0	0	128715
	1	14	32	23	0	0	0	128715
	2	63	5	53	45	54	81	128715
16	0	272	183	210	0	0	0	230175
	1	121	19	19	0	0	0	230175
	2	63	130	90	0	0	0	230175
25	0	1365	479	801	0	0	0	275905
	1	1046	1240	10	0	0	0	275905
	2	246	251	765	0	0	0	275905
Totals		3425	2762	2368	240	255	282	1963434

increased rapidly, and it appears, exponentially. The other algorithms, all the depth-first algorithms and the

TABLE V
Algorithm Runtime for Tmax=10, All Cases

Grid Size	Tl	Algorithms						
		1	2	3	4	5	6	7
4	0	4.3	4.4	3.7	4.1	3.9	3.6	10.4
	1	0.4	0.4	0.7	0.4	0.4	0.6	10.7
	2	0.7	0.4	1.1	0.9	0.6	0.8	10.9
9	0	17.9	16.1	18.8	10.4	8.0	17.6	150.8
	1	14.7	10.6	19.7	8.8	6.9	15.0	152.1
	2	14.3	8.2	25.3	10.9	9.0	17.1	153.5
16	0	120.8	92.1	191.8	113.1	101.0	212.7	672.9
	1	174.0	119.4	221.5	119.2	114.9	220.4	671.9
	2	163.4	193.8	318.3	156.4	125.4	250.2	674.8
25	0	661.0	679.2	1028.6	561.9	547.8	1094.6	1249.3
	1	1124.4	1092.0	1277.3	709.8	698.9	1262.0	1244.5
	2	1253.4	1500.3	2252.4	843.0	845.0	1494.1	1249.1
Totals		3549.3	3734.9	5359.2	2538.9	2461.8	4588.7	6250.9

runtime in cpu seconds

best-first restart algorithm, tended to be somewhere between the two extremes. In the final case, where Tmax=10 and Tl=2, the total enumeration algorithm was still competitive. It is anticipated that as Tmax is increased beyond 10 the total enumeration algorithm will surpass all the other algorithms in runtime required.

Based on this example, it appears that the best-first strategy with either the myopic or random starting solution is preferred. Further examples might indicate a preference between the myopic and random starting solutions.

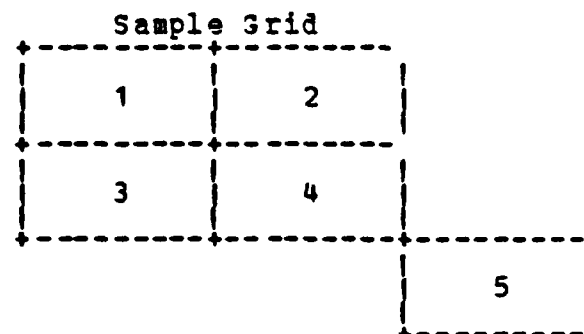
V. EXTENSIONS

A. PD AS A FUNCTION OF WAKE AGE

The program as currently written, assumes that the probability of detection of the wake is constant with the age of the wake. In reality, one would expect the probability of detection to decrease as the wake age increased. This would be relatively simple to implement. What would be needed would be either a separate probability of detection as the wake aged or a functional relationship between wake age and probability of detection. The program would then have to be modified to use the appropriate probability for the various search matrices, $St(i)$.

B. COUNTER-DETECTION

Another aspect of interest is the idea of counter-detection of the searcher by the target. It could be speculated that if the target detected the searcher when they were both in the same cell at the same time, the search would be blown and therefore any target probability mass which detected the searcher could be removed from the problem. One way of accomplishing this would be to have an added cell to the grid. The searcher would be unable to search this cell and the target once in the added cell would never transition out. Then two search matrices would have to be used. The modified search matrix St' would include a probability of counter-detection and a transition of that target probability mass which counter-detected into the extra cell. The search matrix St for the wake search would remain unchanged. Figure 5.1 illustrates the problem input, initial probability distribution, target transition matrix, and example search matrices.



The target transition matrix

$$T = \begin{bmatrix} .40 & .30 & .30 & .0 & .0 \\ .30 & .40 & .0 & .30 & .0 \\ .30 & .0 & .40 & .30 & .0 \\ .0 & .30 & .30 & .40 & .0 \\ .0 & .0 & .0 & .0 & 1.0 \end{bmatrix}$$

The target initial probability distribution

$$P_0 = \begin{bmatrix} 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \end{bmatrix}$$

The searcher motion constraint matrix

$$\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 1 & 2 & 4 & 0 & 0 \\ 1 & 3 & 4 & 0 & 0 \\ 2 & 3 & 4 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The probability of non-detection is 0.500

The probability of counter-detection is 0.200

Example Search Matrices for a search in cell 1

Original Wake Search Matrix

$$S_1(1) = \begin{bmatrix} 0.50 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.00 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.00 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.00 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.00 \end{bmatrix}$$

Modified Real-Time Search Matrix

$$S_1'(1) = \begin{bmatrix} 0.50 & 0.0 & 0.0 & 0.0 & 0.20 \\ 0.0 & 1.00 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.00 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.00 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.00 \end{bmatrix}$$

Pnd for a 4 time unit Search with a 2 time unit detector

$$P_{nd} = P_0 \cdot S_1' \cdot S_2 \cdot S_3 \cdot T \cdot S_2' \cdot S_3 \cdot S_4 \cdot T \cdot S_3' \cdot S_4 \cdot T \cdot S_4' \cdot 1]$$

Figure 5.1 Search Problem Modified for Counter-Detection.

C. APPROXIMATIONS TO THE OPTIMAL SOLUTION

For real life problems on the order of 25×25 grids and larger with $T_{\max} \geq 50$ and $T_1 \geq 5$, the computer time to reach a close approximation of the optimal solution may be excessive. Therefore, Stewart in [Ref. 1] suggested using the first feasible solution that arose from the depth first search. The first solution could be viewed as a form of the myopic search. The optimality characteristics of such a solution have not been subjected to worst case or average case analysis.

VI. CONCLUSIONS

A. PROBLEM COMPLEXITY

Mention has been made of the apparently fundamental intractability of the discrete time and space moving target constrained searcher problem. Earlier it was conjectured that the problem is at least NP-Complete and possible NP-Hard. Appendix A discussed the complexity of the problem and provides some justification for the conjecture.

If the conjecture is true then the implications are clear. Conventional attempts at trying to find 'efficient' algorithms are doomed to failure, particularly if the problem is NP-Hard. Thus heuristic algorithms are the most fruitful avenue for finding optimal or near-optimal solutions.

B. CHOICE OF AN ALGORITHM

The myopic starting solution for Brown's algorithm combined with the best-first strategy produces the best results. This combination was consistently better than the total enumeration algorithm and, as the problem became more difficult, performed better than the other five variations.

The solutions to the constrained searcher problem (Stewart's algorithm output) were compared for both optimality and runtime. However, the solutions to the unconstrained searcher (Brown's algorithm output) were not compared except indirectly as the output affected Stewart's algorithm. There may be significant differences in the bounds provided by the different variations of Brown's algorithm. It did appear that the restart approach was significantly slower as T_{max} increased. This had a significant

effect on the performance of Stewart's algorithm. As T_{max} increased the restart approach became increasingly slower than the myopic or random approaches and the difference could not be explained by a change in the number of solutions which had to be calculated. Thus even if the optimality of the bounds produced by the restart approach was improved over the other approaches, it was overshadowed by the increased time required. It is conjectured that the same effect would occur if the estimation technique were used.

There are several alternative ways to approach the solution of the constrained searcher problem. As shown by Stewart in [Ref. 1], the idea of network flows could be applied. Also, J.N. Eagle in [Ref. 6] proved that a dynamic programming approach was guaranteed to provide an optimal solution for the non-wake search constrained searcher problem. Either of the above approaches may be extendable to the wake search case.

C. WAKE SEARCH

The extension of the constrained searcher problem to deal with wakes rested on two assumptions: 1) the independence of the searches and 2) the goal of minimizing the probability of non-detection for a fixed time period search. The first assumption may not in fact be valid under all conditions. To say that two searches have independent probabilities of detecting a target when the searches are constrained to use the same search path might be incorrect.

For example, suppose that at time period i the target moved from North to South in cell j . Also suppose that in time period $i+2$ the target moved from East to West in the same cell j . Now the searcher enters the cell at time period $i+4$ with a 5 time unit wake detector. If the searcher uses a search plan that searches from East to West in an

exhaustive fashion he has several chances and thus a higher probability of detecting the time period i wake than the time $i+2$ wake. Conversely if the searcher uses an exhaustive search path from North to South he will have several chances to detect the wake from time period $i+2$. Therefore it may be an oversimplification to assume that the wake searches are all independent of each other. The assumption did make the calculations possible and therefore even if the assumptions stretches the truth it will still yield answers of interest.

D. ACKNOWLEDGEMENTS

The basic idea underlying this thesis, that of investigating applications of optimal search theory techniques to wake search, was suggested to me by Dr. Steven MacGruder at the Applied Physics Laboratory of Johns Hopkins University. He was interested in studying the exploitability of wakes in general and had been using heuristic techniques to determine reasonable search strategies. After a 6 week experience tour working under Dr. MacGruder I returned to the Naval Postgraduate School where I worked closely with Professor Eagle in completing this thesis.

APPENDIX A

DISCUSSION OF THE COMPLEXITY OF THE SEARCH PROBLEM

In this appendix, I hope to lay the foundation for the determination of the complexity of the wake search problem. The problem will be approached in two ways:

1. Show a restricted version of the constrained wake search problem is an element of NP.
2. Show that the unconstrained wake search problem is functionally similar to the Knapsack Problem which has been shown to be NP-Complete.

As defined in [Ref. 3], a problem is in NP if a tentative solution can be written down and checked in time polynomial in the size of the problem input. The wake search problem can be formulated as follows:

INPUT: A finite set $J = \{j_1, j_2, \dots, j_n\}$ of cells which can be searched, a probability $P_0 = \{p_1, p_2, \dots, p_n\}$, $0 \leq p_i \leq 1$ for all $i=1, N$ that the target will start the problem in each cell, a probability T_{ij} , $0 \leq T_{ij} \leq 1$ for all $i=1, N$, $j=1, N$ that the target will move from cell i to cell j , a bound $B \geq 0$, and the sets $I(j_i)$ for each j_i , a subset of J containing the cells reachable from cell j_i in the next time period.

QUESTION: Is there a "search" of the cells in J of length T having a total probability of non-detection no more than B such that each j_{t+1} is an element of $I(j_t)$? The probability of non-detection is

$$P_{nd} = P_0 \cdot \overline{ST_1} \cdot T \cdot \overline{ST_2} \cdot T \cdot \overline{ST_3} \cdot T \cdot \dots \cdot \overline{ST_{max}} \cdot 1 \quad (A.1)$$

Given the above problem definition, what non-deterministic algorithm will solve it in polynomial time? A

non-deterministic algorithm is one which contains two stages; a guessing stage which simply guesses an arbitrary search $j_1, j_2, \dots, j_T$ and a polynomial time "checker" which answers the question, "does the arbitrary search have a non-detection probability less than B ?" It will suffice to show that the "checker" will stop in polynomial time and answer the question, yes or no.

The "checker" merely calculates the probability of non-detection using equation A.1 and compares the result with B . The calculation entails $(T-1) + (T \cdot T_1) + 1$ matrix multiplications each of which requires $N \times N$ multiplications. Therefore, the calculation will require $O(T \cdot T_1 \cdot N^2)$ time to complete. The input to the problem has length $O(N^2 + \log(T) + \log(T_1))$ (N elements of P_0 , N^2 elements of T_{ij} , at most N^2 elements of $I(j_i)$, N elements of the guessed search and the values of T and T_1 .) If T and T_1 are restricted to be less than or equal to some specified polynomial function of N , $P(N)$, then the input becomes $O(P(N) \cdot N^2)$ and the calculation time becomes $O(P(N)^2 \cdot N^2)$, both of which are polynomial in N . Therefore the "checker" will answer the question in polynomial time based on the length of the input. Based on the above, the wake search problem is in NP. If the restriction on the size of T and T_1 are removed the problem may not be in NP, i.e. the problem may be NP-Hard.

Once it has been shown that a given problem is in NP, it is of interest to see if the problem belongs to the class of problems called NP-Complete. This class of problems is known to be the set of the hardest problems in NP. It is conjectured that the wake search problem as described above is no easier than the NP-Complete problems, however, a proof is not available. Following is a partial justification for the conjecture.

Proof that a problem is NP-Complete can be done by showing equivalence between the given problem and a problem

already shown to be NP-Complete. As shown above, a restricted version of the wake search problem is in NP. Now if it could be shown that the wake search problem is no easier than a problem which is NP-Complete the proof would be done. What will be done is to show that a relaxed version of the wake search problem namely the wake search problem where the searcher motion is unconstrained is functionally similar to the knapsack problem which is NP-Complete.

The knapsack problem can be stated as follows:

GIVEN: Finite set U , for each $u \in U$ a size $s(u) \geq 0$ and a value $v(u) \geq 0$, and positive integers B and K .

QUESTION: Is there a subset U' of U such that $\sum_{u \in U'} s(u) \leq B$ and such that $\sum_{u \in U'} v(u) \geq K$?

The unconstrained searcher problem can be stated as follows:

GIVEN: a finite set $J \times T_{\max}$, a finite set T , a finite set P_0 , for each $j \in J$ a size $s(j) = 1$ and a value $v(j) = P_0$, and a positive integer $T_{\max}$ and a rational number $K \leq 1$.

QUESTION: Is there a search $J = \{j_1, \dots, j_{T_{\max}}\}$ a subset of $J \times T_{\max}$, such that each element j_i of J is an element of the disjoint subset J_i where the subsets J_i form a partition of $J \times T_{\max}$ and such that $\sum_{i=1}^{T_{\max}} s(j_i) = T_{\max}$ and such that equation A.1 $\leq K$?

While there appears to be significant differences between the wake search problem and the knapsack problem, there is enough similarity that the conjecture that the wake search problem is at least as hard as the knapsack problem is reasonably justified. Since the wake search problem as compared to the knapsack problem is the unconstrained version it also appears that the constrained wake search is also at least as hard.

It has been shown that a restricted version of the constrained wake search problem is in NP. It can be hypothesized that if the restriction (that T and T_1 be polynomial

functions of N) were removed the problem would not be in NP. It has also been shown that the unconstrained wake search problem is functionally similar to a problem which has been previously determined to be NP-Complete. Therefore, it seems reasonable to conjecture that the constrained wake search problem is at least NP-Complete and possibly NP-Hard. The consequences of this conjecture are that the problem is fundamentally intractable and there do not exist 'efficient' algorithms for solving the problem. Thus any heuristic algorithms such as those presented in this thesis are appropriate methods for attempting to find optimal or near-optimal solutions.

APPENDIX B
GRAPHICAL COMPARISON OF ALGORITHM RUNTIMES

This appendix contains graphs of the runtimes of the algorithms for the following cases:

1. 4 cell grid, $T_l=0$, and $T_{max}=1,10$
2. 4 cell grid, $T_l=1$, and $T_{max}=1,10$
3. 4 cell grid, $T_l=2$, and $T_{max}=1,10$
4. 9 cell grid, $T_l=0$, and $T_{max}=1,10$
5. 9 cell grid, $T_l=1$, and $T_{max}=1,10$
6. 9 cell grid, $T_l=2$, and $T_{max}=1,10$
7. 16 cell grid, $T_l=0$, and $T_{max}=1,10$
8. 16 cell grid, $T_l=1$, and $T_{max}=1,10$
9. 16 cell grid, $T_l=2$, and $T_{max}=1,10$
10. 25 cell grid, $T_l=0$, and $T_{max}=1,10$
11. 25 cell grid, $T_l=1$, and $T_{max}=1,10$
12. 25 cell grid, $T_l=2$, and $T_{max}=1,10$

Each graph contains the runtimes for each of the seven algorithms, with the following symbols representing each algorithm:

- - Depth-First Strategy, myopic starting solution
- * - Depth-First Strategy, random starting solution
- + - Depth-First Strategy, restart algorithm
- x - Best-First Strategy, myopic starting solution
- ∇ - Best-First Strategy, random starting solution
- Δ - Best-First Strategy, restart algorithm
- o - Total Enumeration

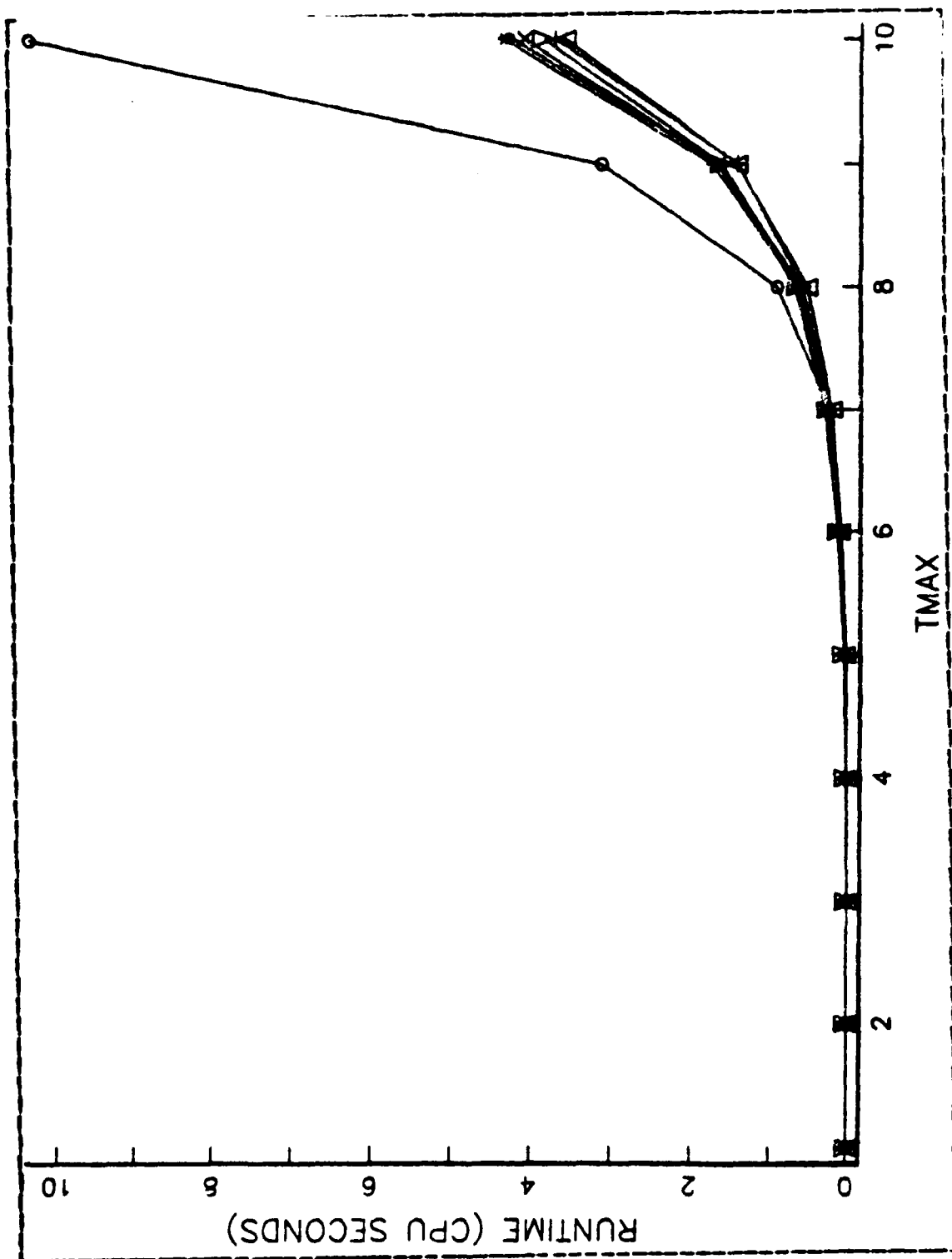


Figure B.1 4 Cell Grid, $T_1=0$, and $T_{max}=1, 10$.

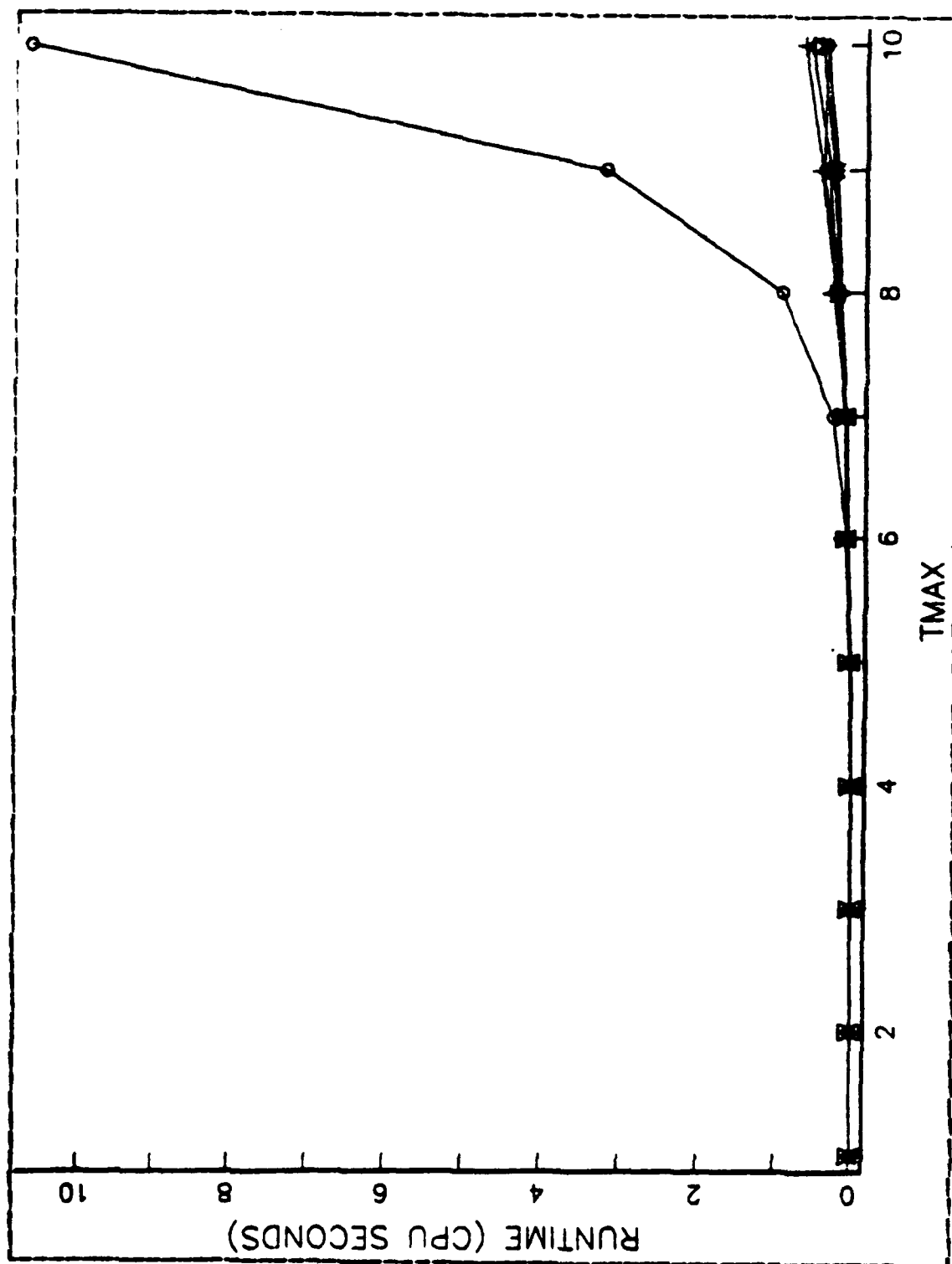


Figure B.2 4 Cell Grid, $T_1=1$, and $T_{max}=1, 10$.

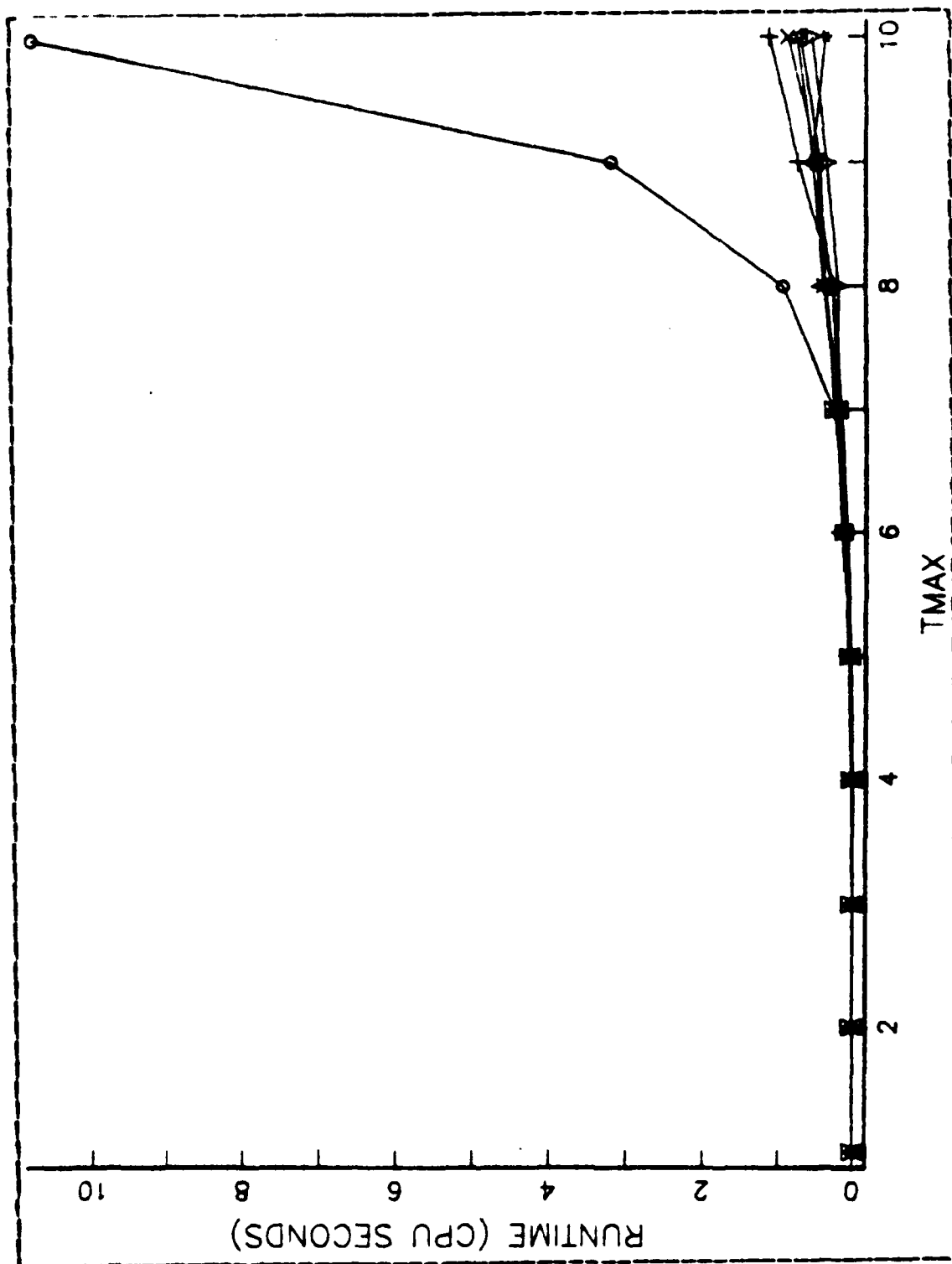


Figure B.3 4 Cell Grid, $T_l=2$, and $T_{max}=1, 10$.

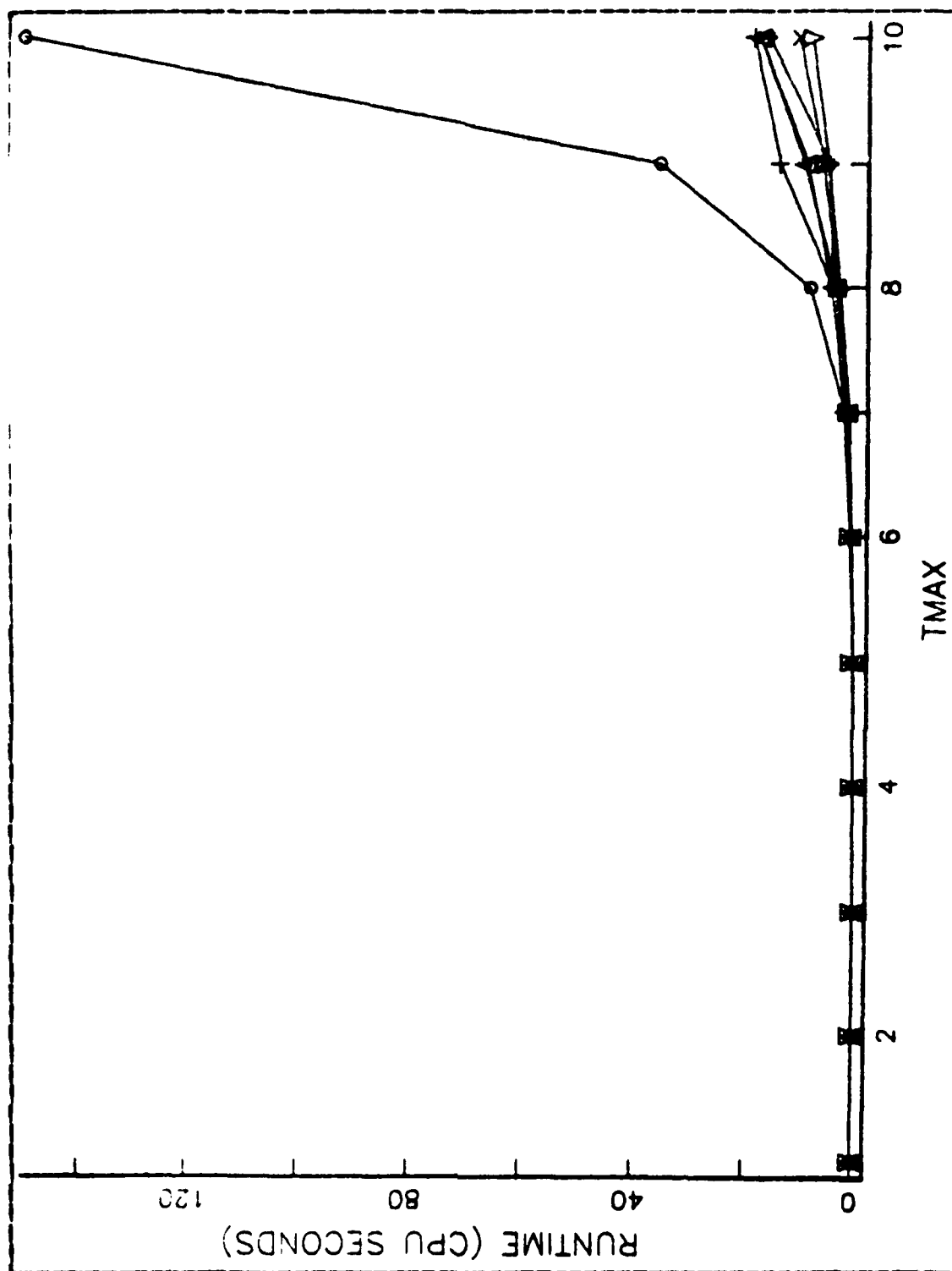


Figure B.4 9 Cell Grid, $T_1=0$, and $T_{max}=1,10$.

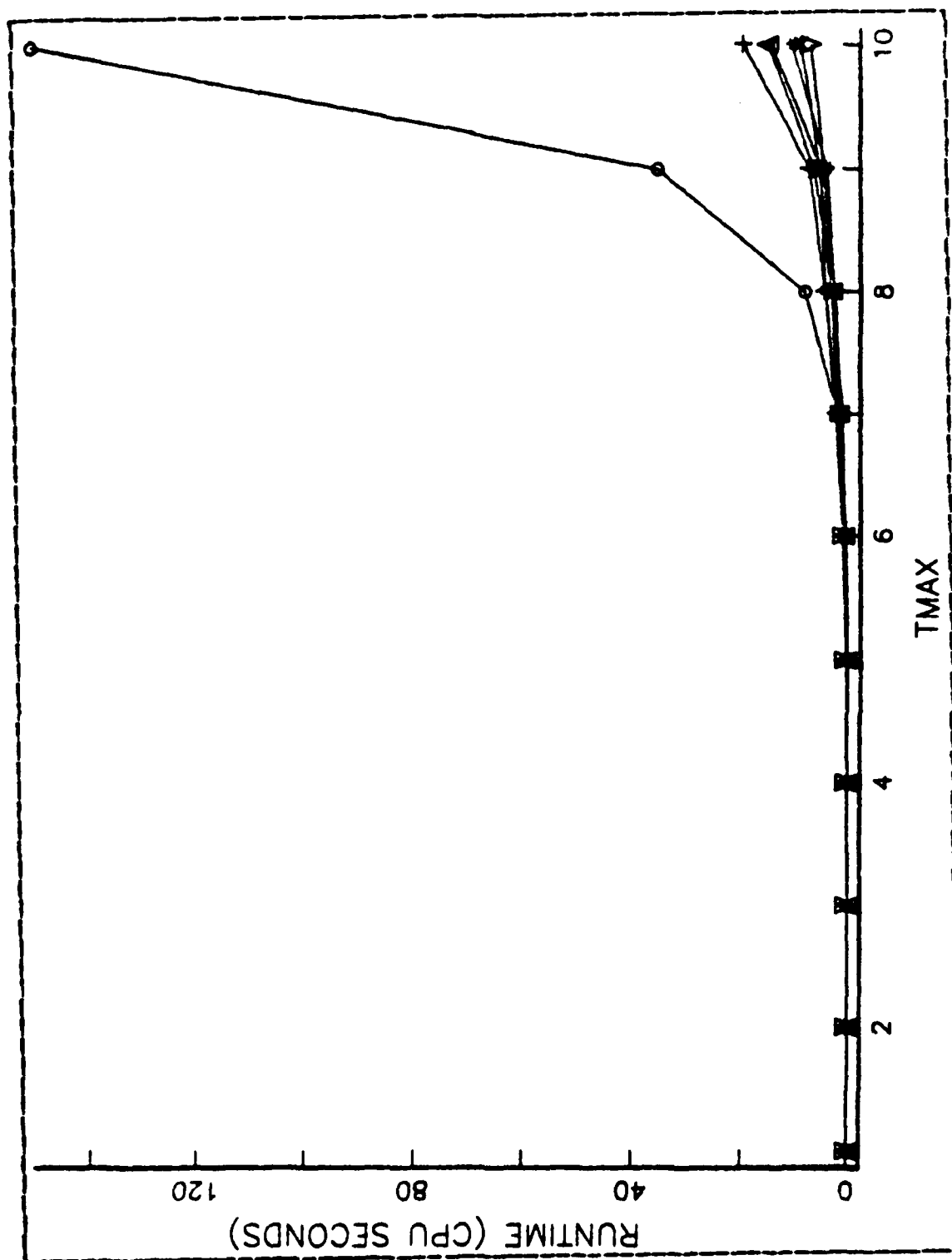


Figure B.5 9 Cell Grid, $T_1=1$, and $T_{max}=1, 10$.

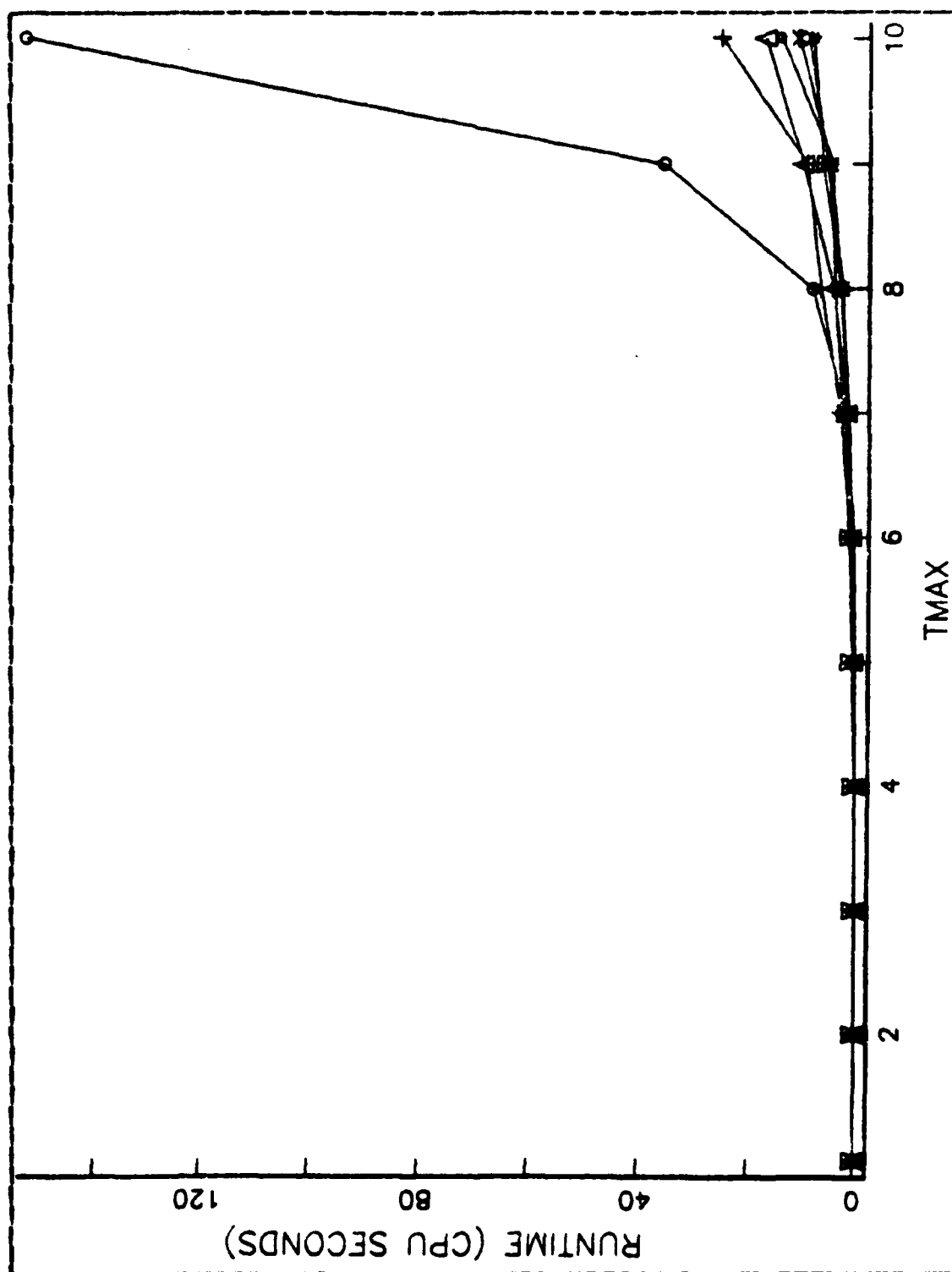


Figure B.6 9 Cell Grid, $T_1=2$, and $T_{max}=1, 10$.

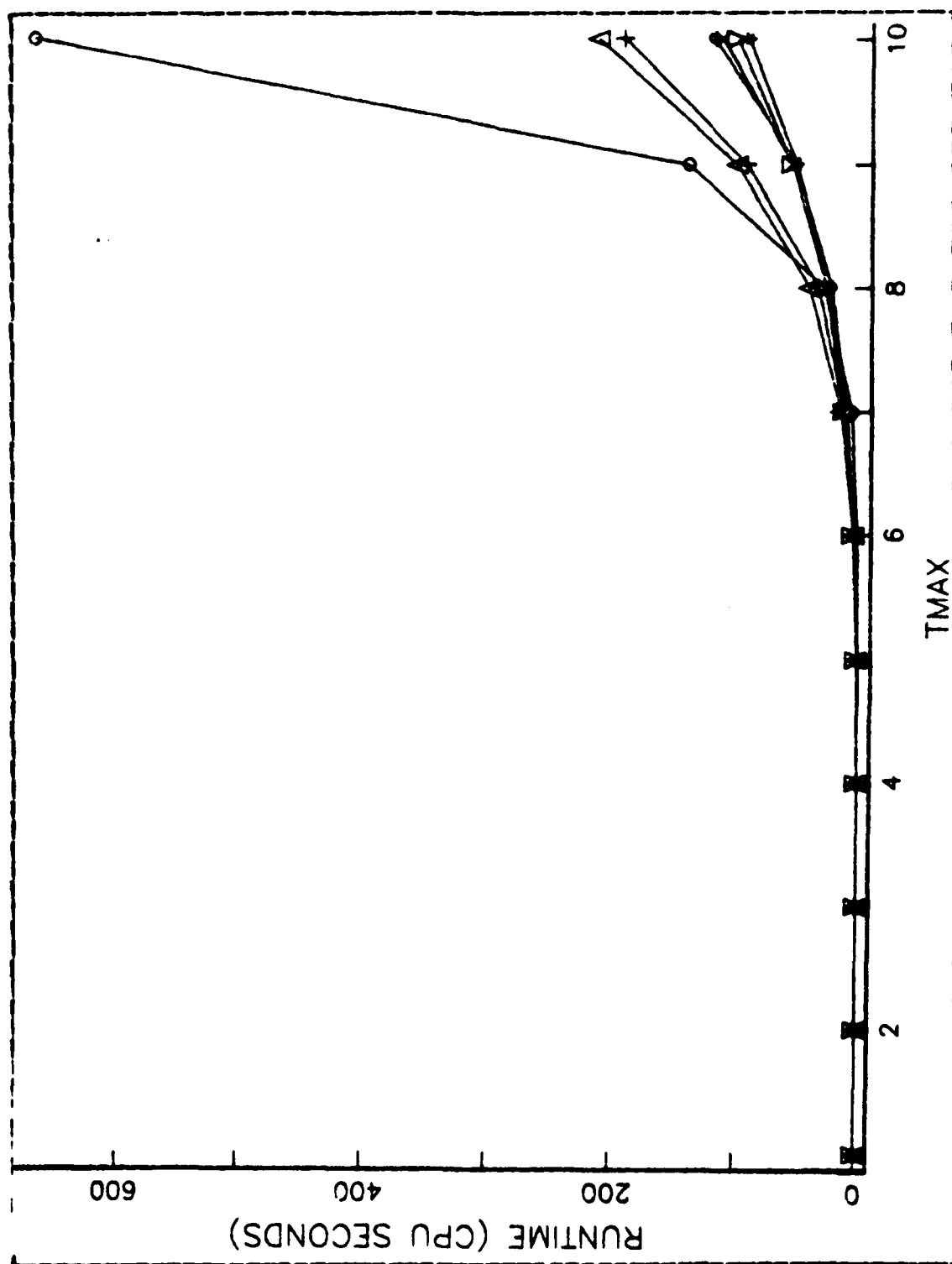


Figure 8.7 16 Cell Grid, $T_1=0$, and $T_{max}=1, 10$.

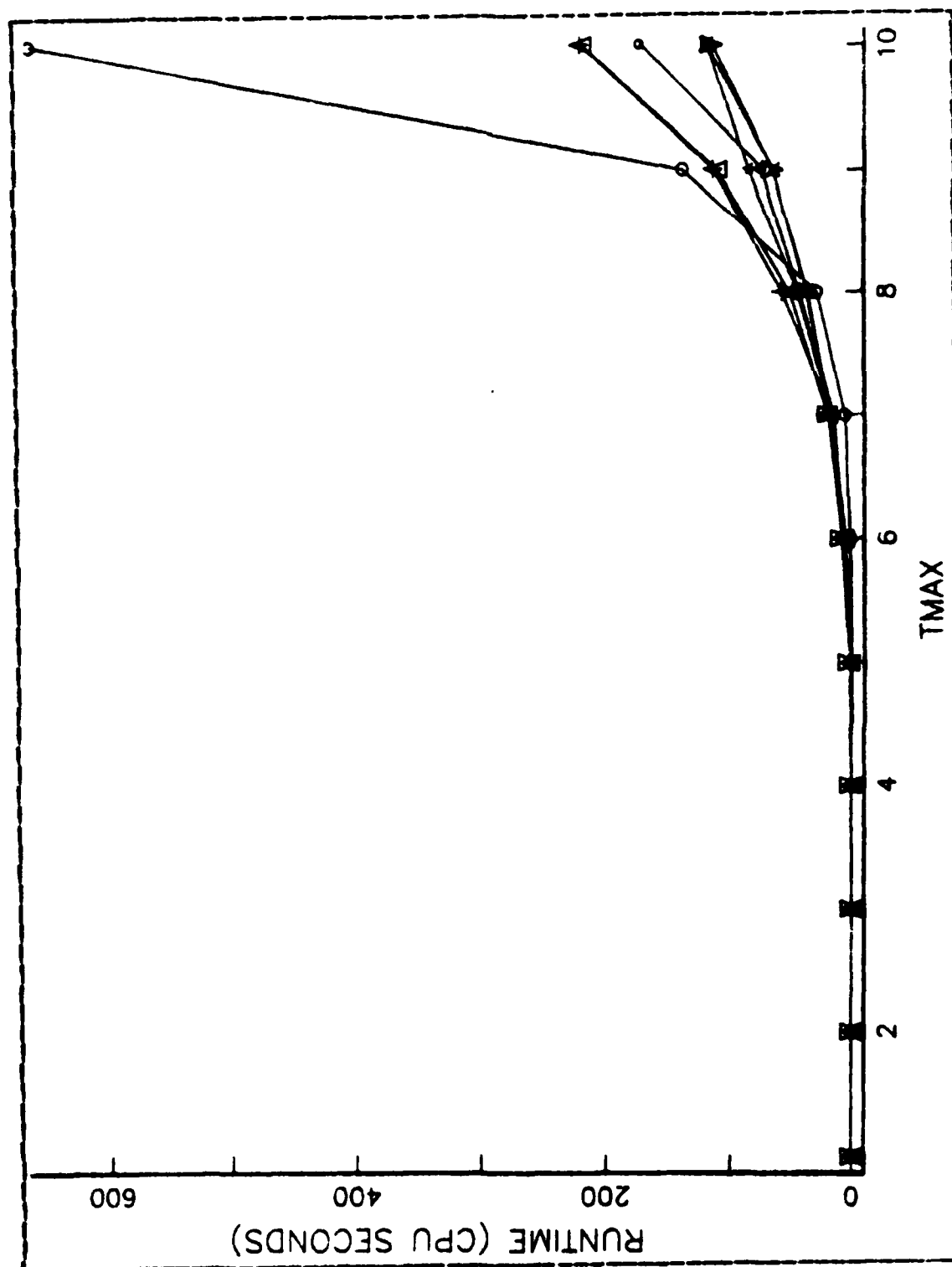


Figure B.8 16 Cell Grid, $T_1=1$, and $T_{max}=1, 10$.

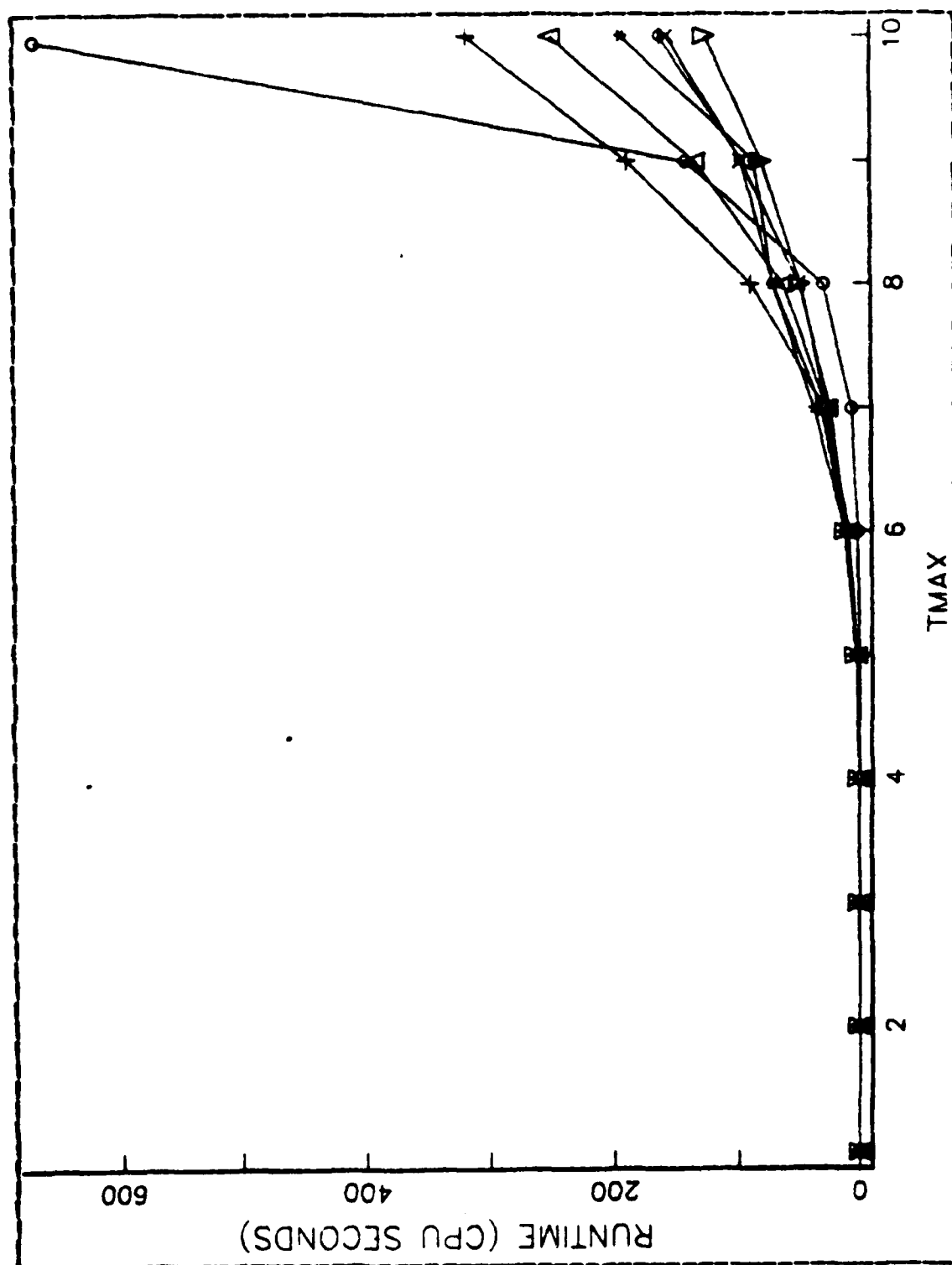


Figure B.9 16 Cell Grid, $T_1=2$, and $T_{max}=1,10$.

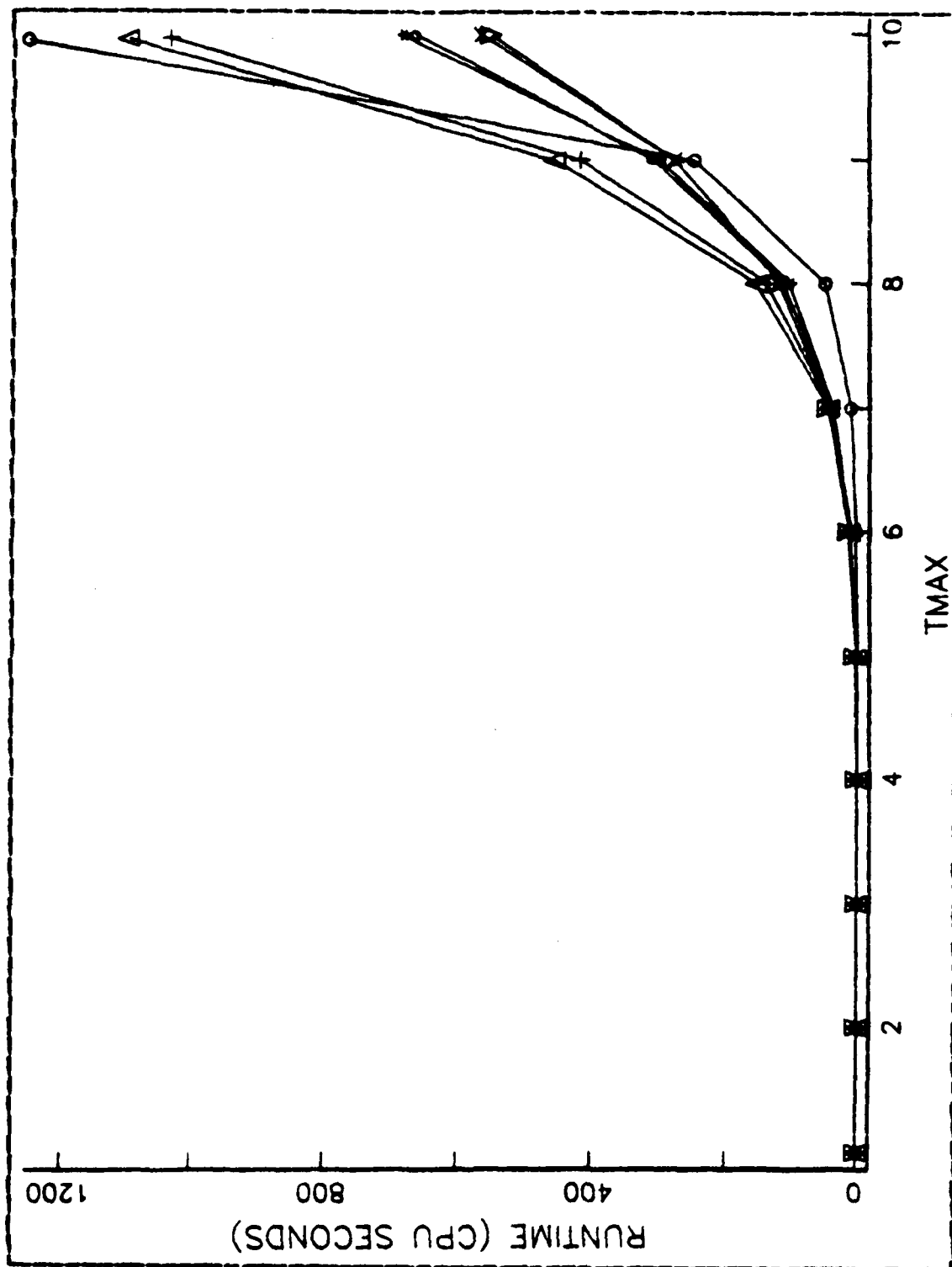


Figure 8.10 25 Cell Grid, $T_1=0$, and $T_{max}=1, 10$.

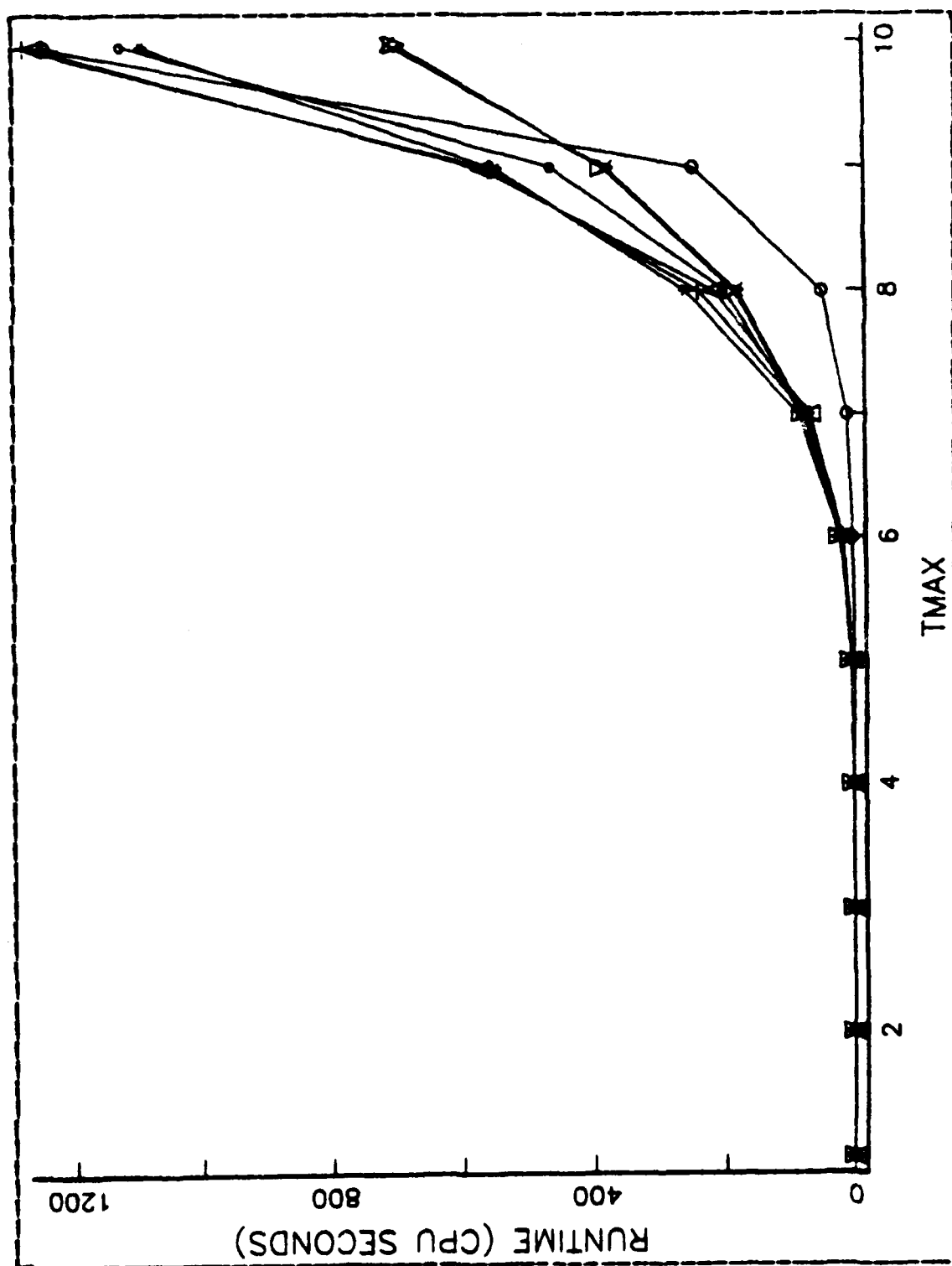


Figure B.11 25 Cell Grid, $P_1=1$, and $T_{max}=1, 10$.

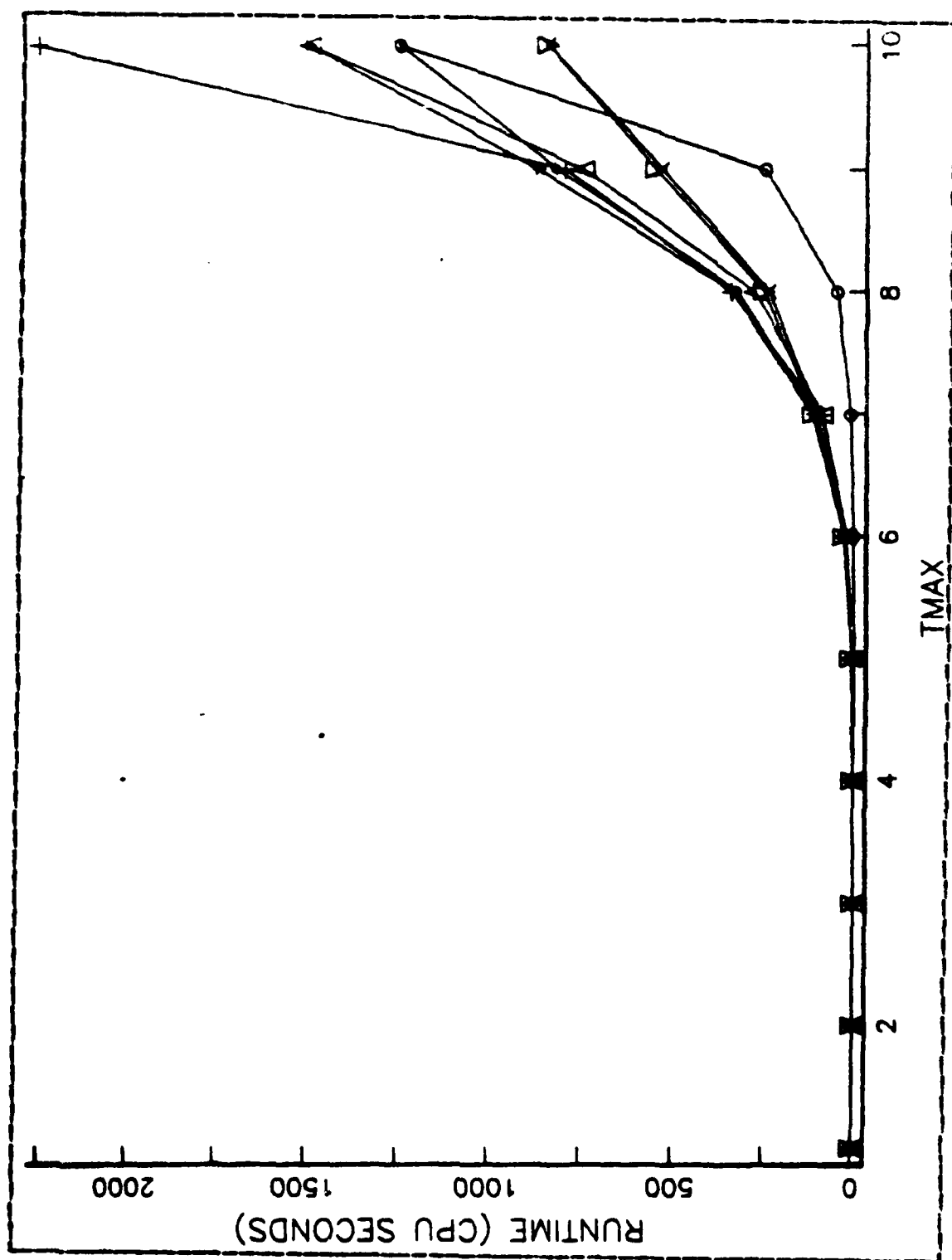


Figure B.12 25 Cell Grid, $T_1=2$, and $T_{max}=1, 10$.

LIST OF REFERENCES

1. T. J. Stewart, "Search for a Moving Target When Searcher Motion is Constrained," Computers and Operations Research, Vol. 6, p. 129-140, 1979.
2. Scott S. Brown, "Optimal Search for a Moving Target in Discrete Space and Time," Operations Research, Vol. 28, p. 1275-1286, 1980.
3. Michael R. Garey and David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, San Francisco, 1979.
4. Douglas R. Smith, Random Trees and the Analysis of Branch and Bound Procedures, Working Paper, Naval Postgraduate School, 1981.
5. Bruce Golden and Edward Wasil Jr., A Curve Fitting Experiment in Estimating Optimal Solution Values to Traveling Salesman Problems, Working Paper, University of Maryland #81-011, April 1981.
6. Naval Postgraduate School Report No. NPS55-83-014, The Optimal Search for a Moving Target When the Search Path is Constrained, by J.N. Eagle, August 1982, revised May 1983.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3. Prof. James N. Eagle, Code 55Er Naval Postgraduate School Monterey, California 93943	1
4. Prof. Jordon H. Bradley, Code 52Bz Naval Postgraduate School Monterey, California 93943	1
5. Dr. Steven F. Macgruder The Johns Hopkins University Applied Physics Laboratory Laurel, Maryland 20707	1
6. COMMANDER Submarine Development Squadron Twelve U.S. Naval Submarine Base Groton, Connecticut 06340	1
7. Lt. Douglas B. Guthe, Jr., USN 46 Laurel Drive Oakdale, Connecticut 06370	1

END

FILMED

2-84

DTIC