Using Multiple Searchers in Constrained-Path, Moving-Target Search Problems

Dell, Robert F.; Eagle, James N.; Martins, Gustavo Henrique Alves; Santos, Almir Garnier

Using Multiple Searchers in Constrained-Path, Moving-Target Search Problems

Robert F. Dell and James N. Eagle
Operations Research Department, Naval Postgraduate School, Monterey, California 93943

Gustavo Henrique Alves Martins and Almir Garnier Santos
Brazilian Navy, Centro de Analises de Sistemas Navais (CASNAV), Centro Postal da Marinha, Centro Rio de Janeiro, R.J. 20091, Brazil

The search theory open literature has paid little, if any, attention to the multiple-searcher, moving-target search problem. We develop an optimal branch-and-bound procedure and six heuristics for solving constrained-path problems with multiple searchers. Our optimal procedure outperforms existing approaches when used with only a single searcher. For more than one searcher, the time needed to guarantee an optimal solution is prohibitive. Our heuristics represent a wide variety of approaches: One solves partial problems optimally, two use paths based on maximizing the expected number of detections, two are genetic algorithm implementations, and one is local search with random restarts. A heuristic based on the expected number of detections obtains solutions within 2% of the best known for each one-, two-, and three-searcher test problem considered. For one- and two-searcher problems, the same heuristic's solution time is less than that of other heuristics. For three-searcher problems, a genetic algorithm implementation obtains the best-known solution in as little as 20% of other heuristic solution times. © 1996 John Wiley & Sons, Inc.

The constrained-path, moving-target search problem [6, 15, 16] has the following characteristics:

- A single searcher and target move among a finite set of cells in discrete time.
- The searcher and target occupy only one cell each time period.
- Each time period, the searcher moves from its current cell to one of a specified set of accessible cells.
- The target moves among cells according to a specified stochastic process.
- If the target occupies the searched cell, the random search formula determines the probability of detection—otherwise the detection probability is zero.
- The target's probability distribution is Bayesian updated for nondetection each time period.

The objective of the search is to find a feasible search path that maximizes the probability of detecting the target in T time periods.

The main contributions of this article center around extending the constrained-path, moving-target search problem to consider multiple searchers explicitly. As this article demonstrates, exact procedures developed to effectively solve single-searcher versions of this problem can be directly extended to multiple searchers, but require excessive computation. We develop an optimal branch-and-bound procedure and six heuristics for moving-target constrained-path problems with multiple searchers.

The specific organization of this article is as follows. Section 1 introduces notation and provides a formulation. Section 2 presents a literature survey on related problems. Section 3 details each heuristic and our optimal algorithm. Section 4 provides detailed computational comparisons of the heuristics applied to a set of test problems using one, two, and three searchers. Finally, Section 5 presents conclusions.

1. NOTATION AND FORMULATION

Indices

i, i', k	cells (e.g., $i = 1, 2, \ldots, N$),
j	searcher,
t	time step (e.g., $t = 1, 2, \ldots, T$),
ω	target or search path [where $\omega(t)$ is the cell occupied at time t].

Data

α_{ij}	detection rate for searcher j in cell i. If the target and searchers j and j' are in cell i, the probability of detection is $1 - \exp(-\alpha_{ij} - \alpha_{ij'})$,
Ω	set of all feasible target paths,
C_i	set of cells that can be reached from cell i in one time step,
p_ω	probability of target following path ω,
s_{ij}	equals 1 if searcher j starts in cell i and zero otherwise.

Binary Variables

$x_{i,i',j,t}$ equals 1 if searcher j moves from cell i at time $t-1$ to cell i' at time t, and 0 otherwise.

Formulation

$$\text{Maximize } PD = \left(1 - \sum_{\omega\in\Omega} p_\omega \exp\left(-\sum_t \sum_j \alpha_{\omega(t),j} \sum_{i|\omega(t)\in C_i} x_{i,\omega(t),j,t}\right)\right),$$

subject to the constraints

$$\sum_{i'\in C_i} x_{i,i',j,1} = s_{ij}, \qquad \forall\, i, j, \tag{1}$$

$$\sum_{i'|i\in C_{i'}} x_{i',i,j,t-1} = \sum_{k\in C_i} x_{i,k,j,t}, \qquad \forall\, i, t > 1, j. \tag{2}$$

The formulation maximizes the probability of detection (PD) within the set of feasible target paths Ω. Constraint set (1) restricts each searcher's initial search effort ($t = 1$) to be in a cell accessible from the starting position. Constraint set (2) ensures continuity of searcher movement between time periods.

The formulation is a concave nonlinear program with network constraints. The searchers move through a $N \times T$ acyclic network and receive a reward at node (i, t) of $p(i, t)U(i, t)$, where

- $p(i, t) = (1 - \exp(-\sum_j \sum_{i'|i \in C_{i'}} \alpha_{i,j} x_{i',i,j,t}))$, the probability of detection if the target is in cell i
- $U(i, t)$ (undetected mass in cell i at time t) is the probability that the target is in cell i at time t and has not been detected by previous search at times $1, 2, \ldots, t - 1$.

Thus, $p(i, t)U(i, t)$ is the probability of initial detection occurring at time period t in cell i and $\sum_{i,t} p(i, t)U(i, t)$ is the expected number of initial detections or PD.

This problem is difficult, because $p(i, t)U(i, t)$ is a function of the search path used to reach cell i at time t. Trummel and Weisinger [21] show that the path-constrained search problem for a stationary target is NP-complete. An example highlights the problem's complexity. A single searcher using five time steps to search a nine-cell problem, with each cell having four cells accessible in one time step, has approximately 1,000 feasible paths to choose from. The same problem with 10 time steps has about 10^6 feasible paths. This problem with three searchers has about 10^{18} feasible paths. The path-constrained search problem for a stationary target is a special case of the path-constrained search problem for a moving target with multiple searchers. Consequently, the main thrust of this article is the development and testing of relatively fast and robust heuristics. This article also presents an optimal branch-and-bound procedure that outperforms other optimal procedures reported in the literature for single-searcher problems.

2. BACKGROUND AND MOTIVATION

Search problems with a stationary target are well studied [17]. The case of a lone searcher looking for a single moving target has also been widely studied and can be divided into two major classes: two-sided search and one-sided search.

Two-sided search problems consider situations where the target is aware that a search effort is being carried out against him and attempts to avoid detection or capture. Game theory is the natural tool here (see [5, 20]). One-sided search problems assume that either the target is not aware of the search or the target needs to accomplish its own task and it is not willing to evade the searcher. Through this reasoning the idea of a Bayesian probability distribution and update of the target's position is straightforward. The one-sided search problems can be further divided into continuous search effort or discrete search path problems. Both groups in more recent work have dealt with the target motion being modeled as a discrete-time Markov chain.

Continuous search effort problems are generally easier to solve to optimality than are discrete search path problems, because the integrality constrains on the search effort are relaxed. In the continuous problem, the total search effort in cell i can be arbitrarily divided among the cells in C_i for search in the next time period. This formulation is the limit of the multiple-searcher problem considered here as the number of searchers becomes large while the total search effort available in each time period is kept fixed. This formulation can be appropriate to model a single searcher with sufficient speed to allow search in several accessible cells during one time period. The continuous problem provides the bound for the branch-and-bound procedure of Eagle and Yee [6]. It is noted that Brown [2] made important progress in solving the continuous search effort problem for a moving target by developing an algorithm that solves a sequence of stationary target problems. Washburn [22] gave a counterpart algorithm for the discrete search effort case. Stone [18] generalized Brown's results. An alternative approach, not examined here, is to model the problem as a partially observable Markov decision process. This was suggested by Smallwood and Sondik [14] and extended by Eagle [4] and Thomas and Eagle [19].

Another interesting model for optimal path problems is the continuous-time and -space case, where the constraints on the searcher's motion are given by a set of differential equations that the searcher's path has to obey. Ohsumi [11] is a good example of such a model.

Benkoski, Dew, Grunert, and Kastner [1] investigate the application of genetic algorithms to search problems. They solve problems of placing stationary active sonobuoys and of determining a path for an active sonar sensor.

The 1991 survey of Weisinger, Monticino, and Benkoski lists 125 references on one-sided search problems and 61 references on search games, but no references for multiple searchers or team effort under the same modeling assumptions.

3. ALGORITHMS FOR MOVING-TARGET CONSTRAINED-PATH PROBLEMS

We develop seven algorithms (six heuristics and one exact procedure) to determine a path that maximizes the probability of detecting a randomly moving target using multiple searchers. This section describes each algorithm using pseudocode. For simplicity, the pseudocode is written for a single searcher. The pseudocode is also appropriate for multiple searchers if the underlying network has an expanded state space to account for extra searchers. An example demonstrates the conceptual state space for multiple searchers. Suppose two searchers are initially stationed at cell 1 and $C_1 = \{1, 2, 4\}$. At the next time step ($t = 1$), each searcher has a separate choice for the next move resulting in the possible combined states of $\{(1, 1), (1, 2), (1, 4), (2, 1), (2, 2), (2, 4), (4, 1), (4, 2), \text{or } (4, 4)\}$, where the first (second) entry is the location of Searcher 1 (Searcher 2).

3.1. Optimal Branch-and-Bound Procedure (BB)

Two optimal branch-and-bound procedures exist in the literature for single-searcher moving-target constrained-path problems. Stewart [15, 16] relaxes the searcher's path constraints to obtain bounds. Eagle and Yee [6] maintain the path constraints, but relax the binary condition of the searcher's position. Our approach bounds *PD* above by calculating the expected number of detections (*ED*), and thus maintains both the searchers' paths and the binary constraints. That is, we replace the objective function *PD* with

$$\text{maximize } ED = \sum_{\omega\in\Omega} p_\omega\left(\sum_t \sum_j \alpha_{\omega(t)j} \sum_{i|\omega(t)\in C_i} x_{i,\omega(t),j,t}\right).$$

This objective function is linear in the decision variables, so maximizing *ED* is equivalent to finding the longest path through an acyclic network. The maximum ED provides an upper bound for the maximum PD, because for search path ω,

$$\max_{\omega} PD(\omega) = PD(\omega^*) \leq ED(\omega^*) \leq \max_{\omega} ED(\omega).$$

Our implementation for finding the longest path is adapted from the directed acyclic graph longest path algorithm of Cormen, Leiserson, and Rivest [3], which is $O(V+E)$, where V is the number of vertices and E is the number of edges.

The pseudocode below uses the notation $PDP(t)$ for the PD up to time t and UB for the upper bound associated with the path maximizing ED.

```
1  Initialize $\omega_{\text{best}}$ and $PD_{\text{best}}$ using a heuristic
2  Find path $\omega_{\text{new}}$ maximizing $ED$ using a longest-path algorithm
3  If $UB_{\text{new}} > PD_{\text{best}}$
4     $t \leftarrow 0$
5     For each cell $i \in C_j$ where $j = \omega_{\text{new}}(t)$
6        $\omega_{\text{new}}(t+1) \leftarrow i$
7        BRANCH($\omega_{\text{new}}, t+1$)
8  Return $\omega_{\text{best}}$ and $PD_{\text{best}}$
```

```
Procedure BRANCH($\omega_{\text{new}}, t$)
 1  If $t < T$
 2     Update probability mass of target up to time $t$ assuming $\omega_{\text{new}}$
 3     $PDP(t) \leftarrow$ probability of detection up to time $t$ using $\omega_{\text{new}}$
 4     Find $\omega_{ED}$, where $\omega_{ED}(t') \leftarrow \omega_{\text{new}}(t')$ for $t' \leq t$ and for $t < t' \leq T$,
       $\omega_{ED}(t')$ is the path starting at node $\omega_{\text{new}}(t)$ that maximizes $ED$
 5     $UB_{\text{new}} \leftarrow PDP(t) + ED$ on path $\omega_{ED}$ from $t+1$ to $T$
 6     If $UB_{\text{new}} > PD_{\text{best}}$
 7        For each $i \in C_j$ where $j = \omega_{\text{new}}(t)$
 8           $\omega_{\text{new}}(t+1) \leftarrow i$
 9           BRANCH($\omega_{\text{new}}, t+1$)
10  Else
11     $PDP(t) \leftarrow$ probability of detection up to time $t$ assuming $\omega_{\text{new}}$
12     If $PDP(t) > PD_{\text{best}}$
13        $PD_{\text{best}} \leftarrow PDP(t)$, $\omega_{\text{best}} \leftarrow \omega_{\text{new}}$
```

Using Step 5 of the main algorithm and recursive calls to Step 7 of the BRANCH procedure, the pseudocode investigates all cells at each time step where the bound (Step 3 of the main algorithm and Step 6 of BRANCH) indicates the potential to be on the path with maximum PD. Martins [10] contains further details.

3.2. Local Search (LS)

This article includes local search (see [12]) as a benchmark of how a simple heuristic performs on our test problems. The pseudocode below explains our implementation of local search with random restarts.

```
1   Repeat
2     Randomly create a feasible searcher path, $\omega_{\text{old}}$
3     Compute $PD_{\text{old}}$ of $\omega_{\text{old}}$
4     $PD_{\text{best}} \leftarrow PD_{\text{old}}$, $\omega_{\text{best}} \leftarrow \omega_{\text{old}}$
5     For $t = 1$ to $(T - 2)$ Do
6       $\omega_{\text{new}} \leftarrow \omega_{\text{old}}$
7       For each cell $i \in C_j$ where $j = \omega_{\text{new}}(t)$
8         $\omega_{\text{new}}(t + 1) \leftarrow i$
9         For each cell $i' \in C_i$
10          $\omega_{\text{new}}(t + 2) \leftarrow i'$
11          If $\omega_{\text{new}}$ feasible compute $PD_{\text{new}}$
12            If $PD_{\text{new}} \geq PD_{\text{old}}$
13              $PD_{\text{old}} \leftarrow PD_{\text{new}}$, $\omega_{\text{old}} \leftarrow \omega_{\text{new}}$
14          next $i'$
15      next $i$
16    If $PD_{\text{old}} \geq PD_{\text{best}}$ go to step 4
17  Until exceed number of restarts or time limit
18  Return path that yielded the highest $PD$
```

3.3. Expected Detection Heuristic 1 (H1)

This heuristic obtains its motivation from the relative ease of calculating the path maximizing *ED*. It myopically picks a path one move at a time. The first move is set as the first move on the path with maximum *ED*. Conditioning on nondetection for the first move (any subsequent path must use this first move), the probability mass of the target is Bayesian updated. The second move is the second move on the path with maximum *ED* for the updated probability mass. This process repeats for all subsequent moves.

The pseudocode below employs the shorthand Path(t) to store the cells occupied by the searcher at time t on the path chosen by the heuristic:

```
1  Path(0) $\leftarrow$ initial cell of searcher
2  For $t = 1$ to $T$ Do
3    Let Path($t - 1$) be the searcher's cell
4    Find path $\omega$ maximizing $ED$ for $t, \ldots, T$
5    Path($t$) = $\omega(t)$
6    Bayesian update the probability mass of the target
     up to time $t$ assuming nondetection
7  Compute $PD$ when searcher follows Path
8  Return Path and $PD$.
```

3.4. Expected Detection Heuristic 2 (H2)

This heuristic expands on H1 by basing the next node added to the searcher's path on more than the path with the maximum *ED*. Specifically, H2 generates a path for every possible single next move, extends the path to T by maximizing *ED*, and picks the path with the largest *PD*. The pseudocode below fully explains H2 using $PD(\omega_i)$ for the probability of detection associated with path ω_i:

```
1   Path(0) ← initial cell of searcher
2   For $t = 1$ to $T$ Do
3     Let Path($t - 1$) be the searcher's cell
4     For all cells $i \in C_{\text{Path}(t-1)}$ Do
5       Find path $\omega_i$ maximizing $ED$ for $t + 1, \ldots, T$, where $\omega_i(t) = i$
6       Compute $PD(\omega_i)$
7     Path($t$) = $k$ such that $PD(\omega_k) = \max_i PD(\omega_i)$
8     Bayesian update the probability mass of the target
      up to time $t$ assuming nondetection on $\omega_k(t)$
9   Compute $PD$ when searcher follows Path
10  Return Path and $PD$.
```

3.5. Genetic Algorithm (GA)

Genetic algorithms (see [7] and [8]) are self-improving algorithms that work by means analogous to natural selection, or survival of the fittest. A crude implementation of a genetic algorithm to our problem provides an introduction to basic operators and characteristics of such algorithms. Each step of this crude implementation is then refined into the form used for the computational work reported in this article. For a complete description of the GA implementation see Santos [13].

```
1   Randomly create a population of feasible paths ($\Omega_{\text{old}}$)
2   $\Omega_{\text{new}} \leftarrow \emptyset$
3   For generation 1 to maximum number of generations
4     While insufficient number of paths in $\Omega_{\text{new}}$
5       Select two paths ($\omega_1, \omega_2$) from $\Omega_{\text{old}}$
6       Apply Cross-Over on $\omega_1$ and $\omega_2$ to form new path
7       Apply Mutation operator on the new path
8       Calculate $PD$ for the new path and add to $\Omega_{\text{new}}$
9     $\Omega_{\text{old}} \leftarrow \Omega_{\text{new}}$, $\Omega_{\text{new}} \leftarrow \emptyset \cup \{$3 best paths from $\Omega_{\text{old}}\}$
10  Return path that yielded the highest $PD$
```

A characteristic of genetic algorithms is the need to set run-time parameters, such as the population size, the probability of crossover, the probability of mutation, and the number of generations. We automate this process in our genetic algorithm implementation to provide values that are empirically robust across a variety of problems. This, of course, may limit the efficiency of the algorithm for particular cases but allows the algorithm's results to be more easily generalized to new problem instances.

We allow the population size to vary between generations by increasing (decreasing) the number of paths when the population tends toward poor (good) paths. We impose a maximum of 200, 400, and 600 paths for the one-, two-, and three-searcher problems, respectively. The size of the initial population is calculated as a function of the number of searchers, size of the network, and number of time steps available to search.

Careful creation of the initial population ensures that every path in that population is distinct. As a general rule, the more diversity that exists within a population, the greater the ability of genetic algorithms to evolve in improving directions. To improve the best path in the initial population, the LS heuristic (Steps 5 to 15 only) is used on the path.

Step 5 of the GA pseudocode selects paths from the old generation to create the new generation. We randomly pick individuals with probability equal to the square root of the path's *PD* divided by the sum of the square roots of each path's *PD* in its population. The square-root function allows enhanced discrimination between very good and very bad paths but reduces discrimination between paths with high *PD*. Within any new generation the three best paths from the previous generation are left unchanged guaranteeing the best path encountered so far survives. The genetic algorithm literature (see Koza [9]) refers to this process as elitism.

The crossover operation (step 6) probabilistically creates a random mix of the two parent paths (perhaps leaving a parent intact). It is a random mix, because the time where the crossing takes place is uniform and randomly chosen between times that produce a feasible child. Specifically, given a feasible crossing time t' and two parents ω_1 and ω_2, the child is ω, where

$$\omega(t) = \begin{cases} \omega_1(t), & \text{for } t \le t', \\ \omega_2(t), & \text{for } t > t'. \end{cases}$$

As in life, every newborn child has the opportunity to evolve by acquiring characteristics that are not inherited from his or her parents. The mutation operator (Step 7) serves this role by probabilistically changing path cells. We randomly pick a cell from the path (each cell is equally likely) and then randomly pick a change to the chosen cell from all feasible alternatives (each feasible alternative is equally likely).

As generations progress, decreasing the probability of a crossover operation and increasing the probability of a mutation operation empirically improves our genetic algorithm's performance. An initial high crossover probability provides for a diverse population. Increasing the probability of mutation helps avoid the tendency of the best individual converging to a local optimal. The initial probability of crossover (mutation) is 0.5 (0.2). The probabilities are adjusted each generation by a continuous change, a change based on convergence, and a change based on the potential of the previous population (based on an approximation of the skewness coefficient). The upper and lower limits of crossover (mutation) are 0.8 (0.8) and 0.4 (0.1), respectively.

Every three generations, a further attempt to diversify and improve the population of paths is made by applying the LS heuristic to one of the four best paths in the population. Additionally, when the top 10% of the paths are not distinct they are replaced probabilistically (according to a diversity parameter) with a new random path. The diversity parameter in this implementation has an initial value of 0.5 and an upper (lower) bound of 0.7 (0.2). Its value is adjusted based on approximations of the skewness coefficient.

Another step taken to diversify the population, aimed at the poorest group of paths, is to randomly generate a new path to replace any path that shows *PD* less than 20% of the best path. The 20% value is adjusted based on an approximations of the skewness coefficient within 10% and 30% between generations.

When the best solution has not changed for 11 generations, the first one-third of the population is decimated and replaced by random paths. Decimation takes place only once in our implementation. A number of stopping conditions terminate the GA heuristic when exceeded: maximum number of generations (100 times the number of searchers); maximum run time based on the number of searchers, number of cells in the network, and time available to search; and when the best solution has not changed for 20 generations.

3.6. Hybrid Genetic Algorithm (HGA)

The HGA algorithm is the GA that includes in the starting population the three heuristic solutions produced by H1, H2, and the path that maximizes ED.

3.7. Moving Horizon (MH)

The MH heuristic solves a T-time-period problem by optimally solving (using BB) a series of H-time-period subproblems where H (the horizon length) is less than T. After picking a H that allows the subproblems to optimally solve using a reasonable amount of computer time, the heuristic is straightforward, as shown in the pseudocode. In our implementation, H never exceeds 11 for the single-searcher and 6 for the two-searcher test problems.

The MH pseudocode uses $\text{Path}_{IS}(t)$ and $\text{Path}_O(t)$ for cells contained on the path of the initial solution and partial optimal solution, respectively:

```
1   Compute horizon length ($H$)
2   Compute initial solution (IS) using H1
3   For $k = 1$ to $T - H$
4      Solve the subproblem for $t = k$ to $k + H$ optimally
5      If $PD = 0$ then
6         $\text{Path}(k) = \text{Path}_{IS}(k)$
7      else
8         $\text{Path}(k) = \text{Path}_O(k)$
9      Update the Probability mass for the target
10  Compute $PD$ when searchers follow Path
11  Return Path and $PD$.
```

4. COMPUTATIONAL EXPERIENCE

We implement our algorithms using Pascal. Comparisons between the algorithm of Eagle and Yee [6] coded in FORTRAN and our optimal procedure are obtained using an AMDAHL 5990-700A mainframe. We use an IBM-compatible 486/33 personal computer for all other testing.

The test problems investigated in this article are the same 9, 25, and 49 cell problems presented in Eagle and Yee [6]. The initial positions for target and searcher(s) and the searcher(s) probability of detection are also as in Eagle and Yee [6]. When multiple searchers are present, their initial position is the same.

Table 1 provides a computational comparison of total enumeration, the branch-and-bound algorithm of Eagle and Yee, and our BB algorithm where the initial heuristic solution is taken as the best provided by H1 or H2. For these problems, 10 time steps are available to search, as in the original computational work of Eagle and Yee. As shown in Table 1, our BB procedure requires less computation time. The increased number of node evaluations required for the 9-cell problems is apparently caused by the BB's bound not being as tight as that found in the Eagle-Yee procedure. As the probability of detection increases, the expected number of detections becomes a less efficient bound. This is encountered, for example, when the target probability mass is spread over most of the cells or the initial distance between searcher and target is small. This explains why the new procedure branches more in the 9-cell problem than in the 49-cell problem. This also affects the efficiency of the Eagle-Yee procedure.

Table 1. The CPU seconds required to solve the test problems reported by Eagle and Yee [6] using total enumeration, the procedure of [6], and our optimal BB approach. BB requires less time to solve these test problems.

	Total Enumeration		Eagle and Yee		BB	
Cells	No. of node evaluations	CPU time (seconds)	No. of node evaluations	CPU time (seconds)	No. of node evaluations	CPU time (seconds)
9	667,030	44	6,208	18	9,660	4
25	1,582,194	304	3,686	31	3,398	4
49	1,708,266	666	1,945	8	2,083	3

Table 2. Test problem variations.

Searchers	Cells	Time steps	Detection probability
1	9	4	0.33212
2	25	12	0.63212
3	49	20	0.93212

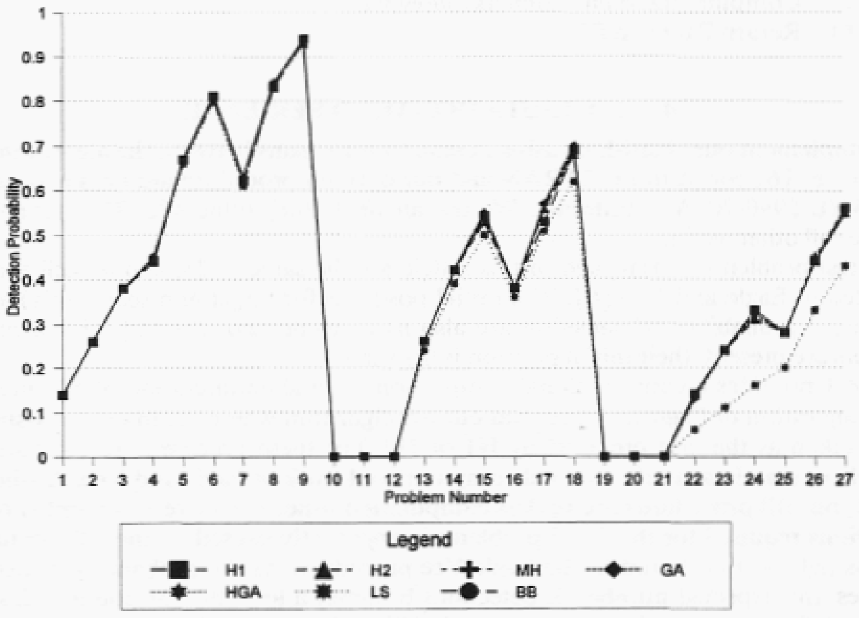

Figure 1. The probability of detection determined by each algorithm on the 27 test problems using a single searcher. With the exception of LS, all heuristics obtain solutions close to the highest probability of detection. Table 3 shows the percentage away from the best known solution obtained by each heuristic.

For all subsequent testing, four basic characteristics (number of searchers, number of cells, number of time steps, and each searcher's probability of detection) are varied to values shown in Table 2. Possible permutations of the number of cells, time steps, and probability of detection produces 27 test problems for each algorithm to solve with each number of searchers. We refer to the test problems using numbers 1–27. Problems 1–9 correspond to 9 cells, 10–18 to 25 cells, and 19–27 to 49 cells. Within the same number of cells, the first three problems ordered in increasing detection probability have $T = 4$, the second three have $T = 12$, and the last three have $T = 20$. For example, problem 1 has 9 cells, $T = 4$, and detection probability of 0.33212; problem 4 has 9 cells, $T = 12$, and detection probability of 0.33212; and problem 27 has 49 cells, $T = 20$, and detection probability of 0.93212.

4.1. One-Searcher Results

Figure 1 shows the probability of detection achieved by the six heuristics and the optimal branch-and-bound procedure for each test problem using a single searcher. This figure highlights the superior performance of all the heuristics when time is not considered. The only exception is, LS, which is included as an indication of how well a simple heuristic

Table 3. Percentages away from the best known solution obtained by each heuristic for the single-searcher problems. The asterisks indicate an optimal solution obtained by BB.

Problem	H1	H2	MH	GA	HGA	LS
1*	0	0	0	0	0	0
2*	0	0	0	0	0	0
3*	0	0	0	0	0	0
4*	1	0	0	0	0	2
5*	1	1	0	1	0	2
6*	0	2	0	1	0	2
7	0	1	0	1	1	3
8	1	1	0	1	1	2
9	2	1	0	1	0	1
10*	0	0	0	0	0	0
11*	0	0	0	0	0	0
12*	0	0	0	0	0	0
13*	0	1	0	1	0	9
14*	0	3	0	0	0	9
15*	1	4	0	0	1	8
16	1	1	0	1	1	7
17	1	3	0	1	1	12
18	1	3	0	1	1	11
19*	0	0	0	0	0	0
20*	0	0	0	0	0	0
21*	0	0	0	0	0	0
22*	0	0	0	1	0	55
23*	0	2	0	2	0	53
24*	0	7	0	2	0	51
25	1	1	1	0	1	30
26	1	2	0	1	1	27
27	2	3	0	2	2	25

performs. Table 3 shows the percentage away from the best known solution achieved by each heuristic.

Run times differ significantly, as shown in Figure 2, where BB has a limit of 60 minutes. LS run times are limited to 15 minutes or until the number of restarts exceeds the population size of GA.

Figure 3 presents run times from 0 to 10 minutes to better illustrate the differences among the various heuristics.

From Figure 3 and Table 2 it is clear that H1 outperforms the other heuristics in run time and always obtains a solution within 2% of the best known. The MH heuristic uses a maximum of 11 time steps for the horizon, which provides superior performance, as indicated in Table 2, but with increased run time, as indicated in Figure 3.

Not shown are the paths each algorithm provides, because they show no discernible pattern for the test problems considered. For some problems, the paths recommended by each algorithm are similar and for others they vary greatly. All test problems appear to have many solutions close to the optimal: A partial enumeration of paths for a nine-cell 12-time-step problem found over 1,000 different paths within 5% of optimal.

4.2. Two-Searcher Results

The same set of 27 problems is solved for the two-searcher case. The complexity of this instance grows exponentially with the number of searchers. However, it gives more insight

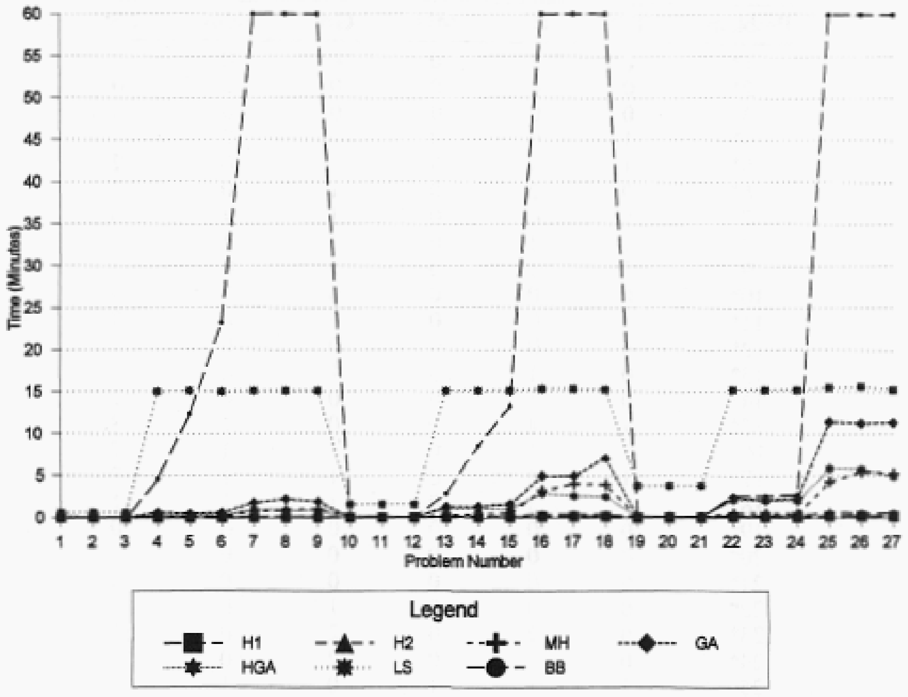

Figure 2. Run time for single-searcher problems. H1, H2, MH, GA, and HGA require less than 10 minutes for each test problem. The BB has a limit of 60 minutes. LS has a limit of 15 minutes.

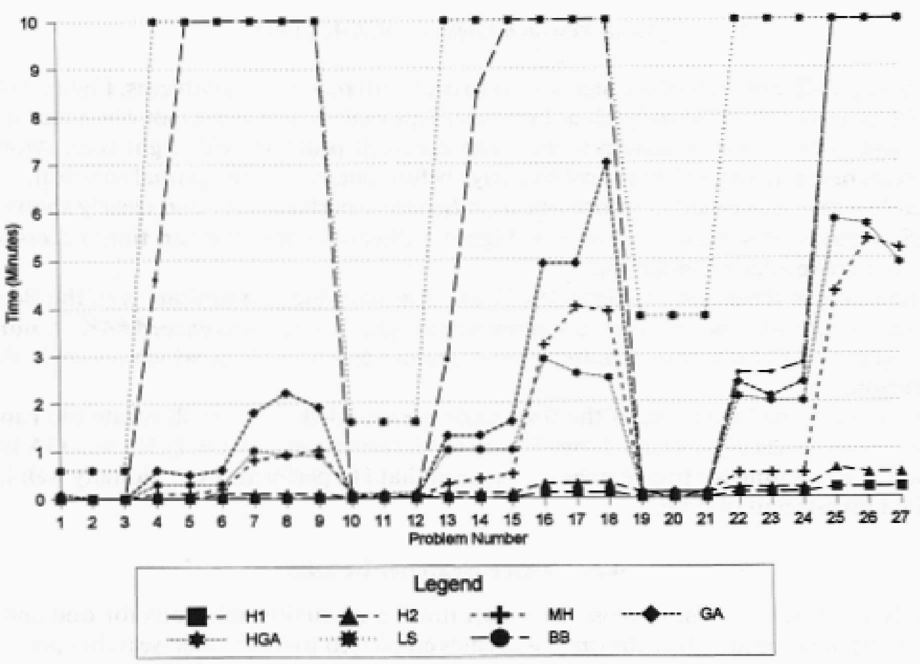

Figure 3. Single-searcher run times from 0 to 10 minutes. Both H1 and H2 require less than 1 min for each test problem.

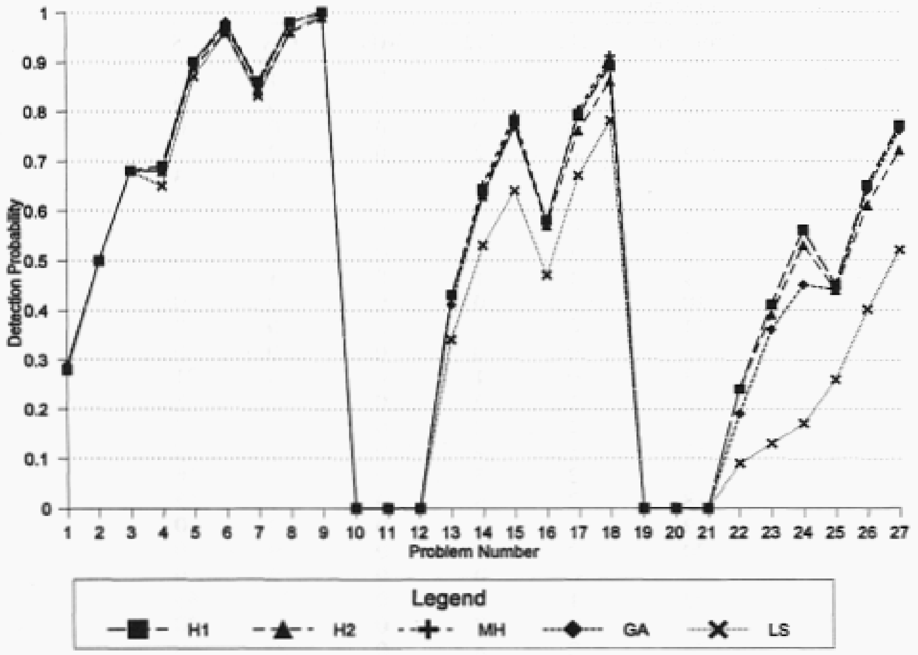

Figure 4. The probability of detection determined by each heuristic on the 27 test problems using two searchers. All heuristics with the exception of LS obtain solutions close to the highest probability of detection. Table 4 shows the percentage away from the best known solution obtained by each heuristic.

on the capability of each of the algorithms to deal with real-world problems. Figure 4 shows results achieved by each individual heuristic. Optimal solutions are not obtained, due to excessive computational requirements. The nine-cell problem with eight time steps and two searchers had to run for almost five days before obtaining the optimal solution.

Table 4 provides detailed information on heuristic performance and clearly shows MH provides superior solutions. However, Figure 5 illustrates that the run time necessary to obtain these results is significant.

The solution times for problems 19–27 are distorted due to a limitation of the Borland Pascal 7.0 compiler, which does not allow single data structures to exceed 64K. A number of programming changes are conducted to overcome this limitation, which results in slower execution.

Once again the lower end of the time scale is expanded to better illustrate the running times of the fastest algorithms. From Figure 6 it is transparent that only H1 and GA keep a reasonable run time for two searchers. It is seen that H1 performs exceptionally well for its limited investment of time.

4.3. Three-Searcher Results

Only H1 and GA exhibit reasonable run times and quality solutions for one and two searchers; they are therefore the only heuristics employed to solve three-searcher problems.

Table 4. Percentages away from the best known solution obtained by each heuristic for the two-searcher problems.

Problem	H1	H2	MH	GA	LS
1	0	0	0	1	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	2	0	2	6
5	1	2	0	1	3
6	1	3	0	1	3
7	0	2	0	1	4
8	1	2	0	1	2
9	1	1	0	1	1
10	0	0	0	0	0
11	0	0	0	0	0
12	0	0	0	0	0
13	0	1	0	4	21
14	1	3	0	4	18
15	2	3	0	3	20
16	1	4	0	1	20
17	2	5	0	2	16
18	2	6	0	1	15
19	0	0	0	0	0
20	0	0	0	0	0
21	0	0	0	0	0
22	1	0	0	32	65
23	0	7	0	13	70
24	0	6	0	20	69
25	0	2	0	2	42
26	0	7	0	3	40
27	0	7	0	2	34

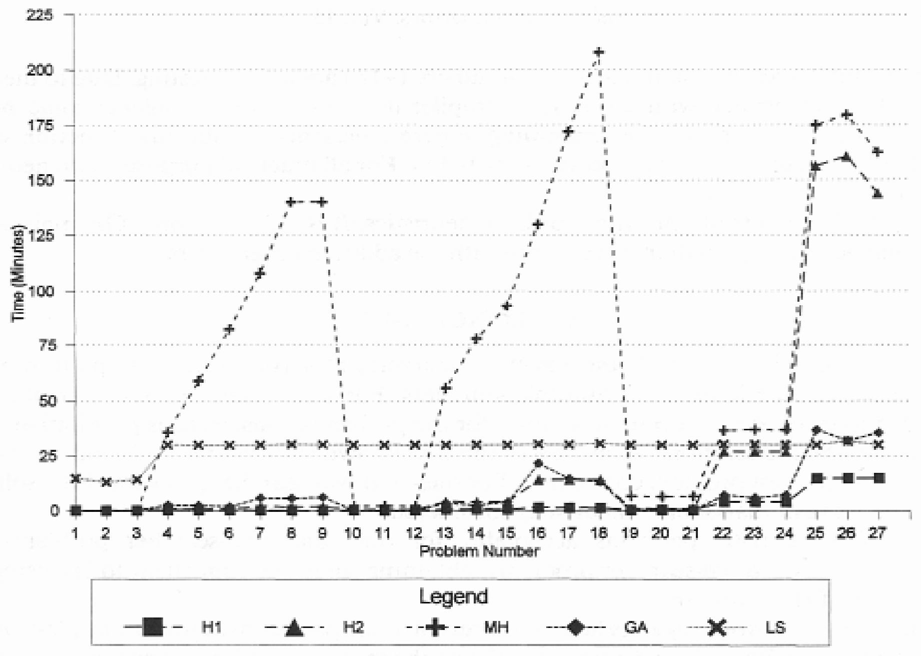

Figure 5. Computational effort to solve the 27 test problems with two searchers. H1 requires less time than all other heuristics. GA maintains reasonable run times for problems 25, 26, and 27. LS has a run-time limit of 30 minutes.

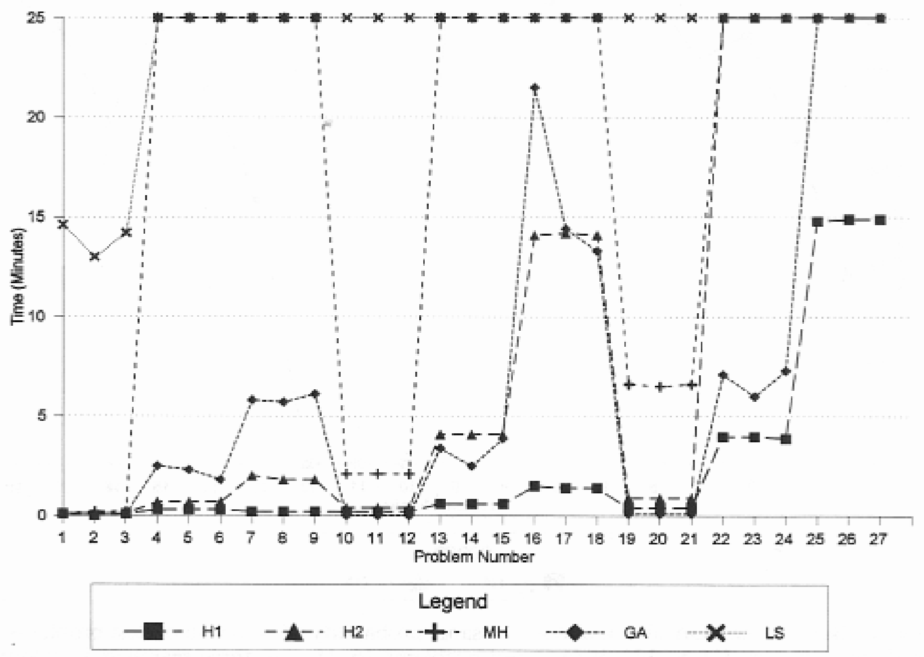

Figure 6. Two searcher run times from 0 to 25 minutes highlights H1's comparatively low computational requirement.

The 9- and 25-cell problems (problem numbers 1–18) are used in testing. Due to memory limitations associated with the Pascal compiler used, the 49-cell problems could not be solved without extensive reprogramming. Figure 7 presents the objective function values that GA (H1) obtains for problems 1–18 (1–15). For all practical purposes both heuristics obtain the same results.

Figure 8 presents the run time for both heuristics. It is clear that only GA maintains a reasonable rate of growth in its run time with the addition of searchers.

5. CONCLUSIONS

This article develops and tests effective heuristics to solve constrained-path, moving-target search problems using multiple searchers. For more than one searcher, the time needed to guarantee an optimal solution for the problems considered is prohibitive. Heuristic H1 obtains solutions within 2% of the best known solution for each one-, two-, and three-searcher test problem considered. For one- and two-searcher problems H1's solution time is less than that of other heuristics considered.

The GA heuristic performs acceptably for one- and two-searcher problems and effectively solves three-searcher problems, obtaining solutions equivalent to H1 using less than 20% of H1's run time.

Our empirical work suggests heuristics can solve the path-constrained multiple-searcher problem both effectively and efficiently. Given the myriad of estimated parameters needed

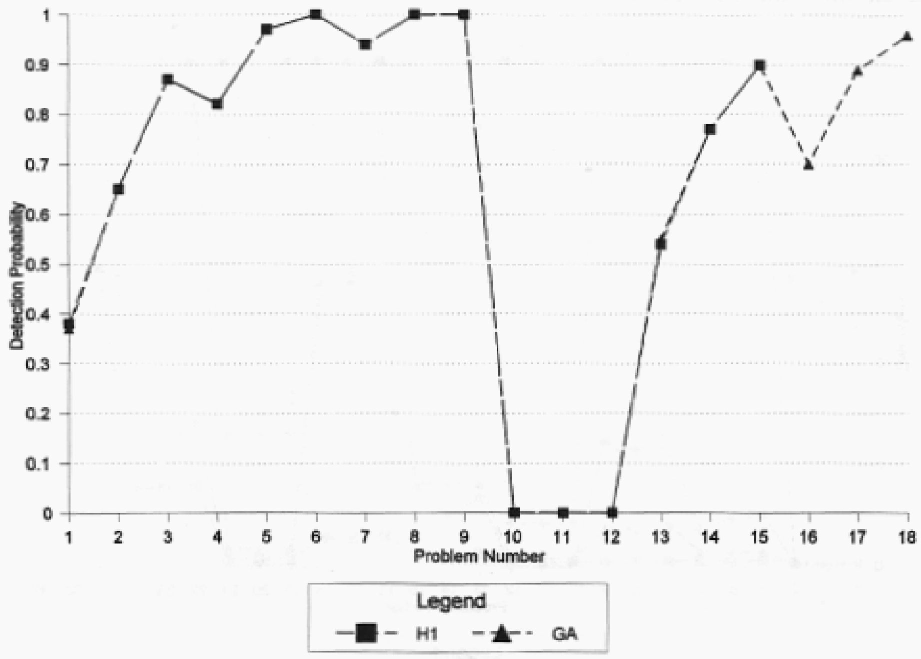

Figure 7. Both H1 and GA obtain almost the same probability of detection on test problems 1–15 using three searchers. H1 could not solve test problems 16–18 with three searchers in reasonable time.

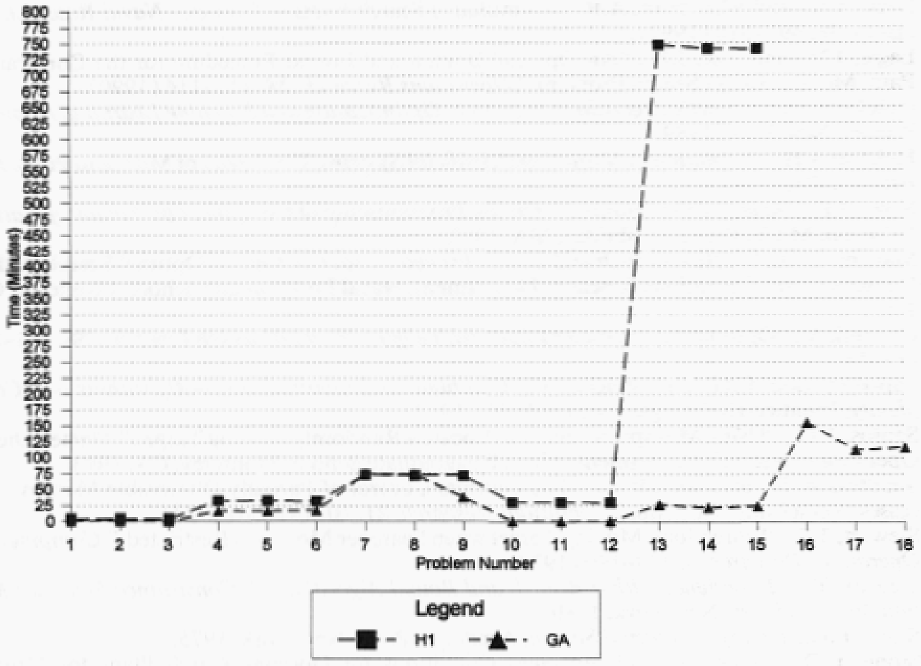

Figure 8. Run time for three-searcher problems and highlights of GA's lower computational requirement. Problems 16–18 are not solved by H1.

to model this problem, obtaining an optimal solution with respect to these estimates does not guarantee better true performance. Hence, a heuristic solution is recommended for many practical applications.

ACKNOWLEDGMENTS

The research contained in this article has benefited from discussions with Professor Alan R. Washburn (Naval Postgraduate School). The authors thank the Brazilian Navy and the Naval Postgraduate School for providing Brazilian Naval officers post-baccalaureate education opportunities in the United States.

REFERENCES

[1] Benkoski, S.J., Dew, E., Grunert, M.E., and Kastner, J.C., "Applications of Genetic Algorithms to Development of Optimal ASW Search Tactics in a Tactical Oceanography Environment (Interim Report)," Daniel H. Wagner Associates Report to Naval Research Laboratory Detachment, 1993.

[2] Brown, S.S., "Optimal Search for a Moving Target in Discrete Time and Space," *Operations Research,* **28,** 1275–1289 (1980).

[3] Cormen, T.H., Leiserson, C.E., and Rivest, R.L., *Introduction to Algorithms,* McGraw-Hill, New York, 1991.

[4] Eagle, J.N., "The Optimal Search for a Moving Target when the Search Path is Constrained," *Operations Research,* **32,** 1107–1115 (1984).

[5] Eagle, J.N., and Washburn, A.R., "Cumulative Search-Evasion Games," *Naval Research Logistics*, **38**, 495–510 (1991).

[6] Eagle, J.N., and Yee, J.R., "An Optimal Branch and Bound Procedure for the Constrained Path, Moving Target Search Problem," *Operations Research*, **38**, 110–114 (1990).

[7] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, New York, 1989.

[8] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[9] Koza, J.R., *Genetic Programming—On the Programming of Computers by Means of Natural Selection*, M.I.T. Press, Cambridge, MA, 1992.

[10] Martins, G.H., "A New Branch and Bound Procedure for Computing Optimal Search Paths," Master's thesis, Operations Research Department, Naval Postgraduate School, Monterey, CA, 1991.

[11] Ohsumi, A., "Optimal Search for a Markovian Target," *Naval Research Logistics*, **38**, 531–554 (1991).

[12] Papadimitriou, C.H., and Steiglitz, K., *Combinatorial Optimization—Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[13] Santos, A.G., "Using Multiple Searcher to Locate a Randomly Moving Target," Masters thesis, Operations Research Department, Naval Postgraduate School, Monterey, CA, 1993.

[14] Smallwood, R., and Sondik, E., "The Optimal Control of Partially Observable Markov Processes over a Finite Horizon," *Operations Research*, **21**, 1071–1088 (1973).

[15] Stewart, T.J., "Search for a Moving Target when Searcher Motion is Restricted," *Computers & Operations Research*, **6**, 129–140 (1979).

[16] Stewart, T.J., *Experience with a Branch and Bound Algorithm for Constrained Searcher Motion*, Plenum Press, New York, 1980.

[17] Stone, L.D., *Theory of Optimal Search*, Academic Press, New York, 1975.

[18] Stone, L.D., "Necessary and Sufficient Conditions for Optimal Search Plans for Moving Targets," *Mathematics of Operations Research*, **4**, 431–440 (1979).

[19] Thomas, L.C., and Eagle, J.N., "Criteria and Approximate Methods for Path-Constrained Moving Target Search Problems," *Naval Research Logistics*, **42**, 27–38 (1995).

[20] Thomas, L.C., and Washburn, A.R., "Dynamic Search Games," *Operations Research*, **39**, 415–422 (1991).

[21] Trummel, K.E., and Weisinger, J.R., "The Complexity of the Optimal Search Path Problem," *Operations Research*, **34**, 324–327 (1986).

[22] Washburn, A.R., "On Search for a Moving Target," *Naval Research Logistics Quarterly*, **27**, 315–322 (1980).

[23] Weisinger, J.R., Monticino, M.G., and Benkoski, S.J., "A Survey of the Search Theory Literature," *Naval Research Logistics*, **38**, 469–494 (1991).

Manuscript received August 1994
Revised manuscript received February 1995
Accepted October 16, 1995