Optimal Search for a Moving Target with Two-Way Patch Constraints

Jeff Johnson, Director of Business Solutions, Supported Intelligence LLC

Supported Intelligence Technical Paper Series, 2013

This technical paper presents an application of a novel recursive method, using the Rapid Recursive® Toolbox, to quickly determine an optimal path in the search for a moving target.

Potential areas of application for this model are numerous and include civil aviation; remote sensing, particularly with unmanned aerial vehicles; and the military, especially targeting weapons systems and intelligence.

I. Abstract

This technical paper presents an application of a novel recursive method, using the Rapid Recursive® Toolbox, to quickly determine an optimal path in the search for a moving target.

Potential areas of application for this model are numerous and include:

- civil aviation;
- remote sensing, particularly with unmanned aerial vehicles; and
- the military, especially targeting weapons systems and intelligence.

The classic search problem is organized as follows: In each period a searcher must select one cell from the search grid in which to look for the target. If the target is apprehended before the end of the game, the searcher receives a reward. If, however, the target successfully evades capture, the target wins and no reward is given to the searcher. Throughout the game, the target moves around the search grid in an effort to evade capture.

The problem is complicated by logical constraints on the actions of the searcher and the movement of the target (referred to as "two-way path constraints"). Use of the novel recursive method allows the quick calculation of an optimal path and values associated with each stage based on the parameters of the game.

To demonstrate this new technology we apply it to a problem posed in a 1984 paper from the US Naval Postgraduate School. The results of that paper are approximately replicated below, through a process completed in only a few seconds on a modern laptop in contrast with the roughly 19 minutes on IBM's top-of-the-line mainframe needed 30 years ago.

II. Introduction

Searching for a moving target is never easy. Decisions based on numerous assumptions and detailed calculations must be made quickly and accurately to ensure the best chance of apprehending the target. Recognizing the ultimate structure of these situations and framing them as sequential decision problems allows for the formulation of smaller subproblems that are much easier to solve. Novel recursive solution methods can then be easily applied, adapted, and updated.

In addition to rapid calculation, the recursive solution method[1] tracks the actual decision making process more closely than other available methods. It does so by partitioning the game into separate periods, at the beginning of which decisions are made. Each decision calculated under this method takes into account the state of the world and the information available to the searcher at the time of the decision rather than calculating some initially optimal path and assuming the searcher will blindly follow it. Where other methods provide information used as an intermediate step in determining the optimal decision the recursive method delivers actionable results based on changing information.

Eagle provides an extended discussion of a classic example on this topic. He poses a problem where the searcher must find a moving target in ten periods or less, otherwise the target escapes. Calculations in this model are complicated by the presence of path constraints for both the searcher and the target. Given the size and complexity of the model, Eagle required over 19 CPU minutes on a top-of-the-line IBM mainframe to obtain a solution using his own custom-written FORTRAN H code. Solving a generalized form of the model on the same equipment required 152 CPU minutes (over two and a half hours!), and this only after the code was running successfully all the way through.

This paper provides a formulation of search games as sequential decision processes and demonstrates the recursive solution method using a specific example. In particular, this paper aims to demonstrate the value of formulating these problems in the manner described, as well as the benefits of using the Rapid Recursive® Toolbox when it comes time to solve these models. Whereas Eagle's code only solved the model, the Rapid Recursive® Toolbox composes, error-checks, solves, and displays results for the model, all while earning the name

1. Presented by Eagle (1984), among others.

"Rapid." Rather than 20 minutes with a mainframe that filled its own room, these models can now be solved in a matter of seconds on a modern laptop.

OUTLINE

The rest of this technical paper is organized as follows: Section III presents the general model and introduces the specifics of the example. Section IV details the set-up of the problem for use with the recursive solution method and finally Section V discusses the results of using said method to solve the specific example.

ACKNOWLEDGEMENTS

I must express my sincere gratitude to the people who made this paper possible. Their contributions to this work were invaluable, making this paper better and the writing process smoother. They are:

- Tyler Theile, who conducted a thorough review of the contents, formatting, and grammar;
- David Quach, who offered excellent suggestions on earlier drafts, as well as some much-needed technical assistance; and
- Patrick L. Anderson, whose guidance and support helped me at every stage of this process to keep going and to push for more.

III. Characterization of the Model

OUTLINE OF THE GAME

The model presented in this technical paper may rightly be thought of as a form of the childhood game of cops and robbers. In this case one cop (the searcher) chases one robber (the target), with the goal of apprehending the robber before the game ends. Should he be successful, the cop is rewarded. If he fails to catch the robber, however, the cop receives no reward and the robber runs free. The model assumes no error in identifying the target: if the searcher searches the cell containing the target at any point, the target will be found with certainty.

The formalized model takes place on a grid representing the entire search area, say the backyard in the framework above. Extending the model to a higher-stakes arena, the grid could cover a city block, an entire city, or even an area encompassing multiple cities. This model can easily be expanded to cover all possible target destinations.

STARTING POINTS

In many situations, the exact starting point of the target is unknown to the searcher. Often, however, some information exists in the form of the last known location of the target. Think of this as a tip received by a police officer.[2] This model assumes that the searcher is privy to such information, but has no additional knowledge of the target's location. Thus the game begins in the period immediately following the target's known location. Essentially the searcher knows the target's position in Period 0 and begins his search in Period 1, after the target has had sufficient time to move to another cell.

In addition to the known starting location for the target, the model also takes an exogenous parameter for the searcher's starting location. This establishes the first neighborhood (see Figure 1 on page 5) of cells accessible by the searcher. It is assumed that both the searcher's starting point and the target's last known location are on the search grid.

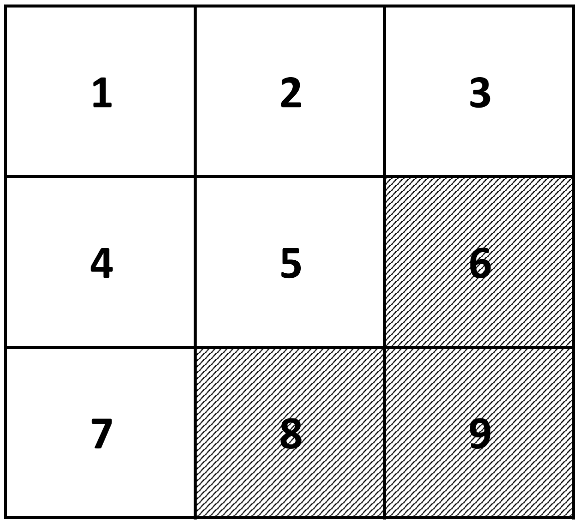
PATH CONSTRAINTS

With the search grid visualization come natural path constraints. Each cell on the grid is connected to the others either directly (the cells share a border), or indirectly via other cells. These connections lead to the construction of *cell neighborhoods*, or groups of adjacent cells.

2. Note that this model assumes the tip is 100% accurate. Easy changes allow the model to incorporate uncertainty, modeling the cases where the target may have been misidentified by the person providing the tip or where the location may be incorrect (the tip-giver may recall seeing the target, but not exactly where the sighting occurred).

Definition: A cell's neighborhood, denoted *C[i]* for Cell *i*, is the set containing cell i and all adjacent cells. Only cells that share a border (not just a vertex) with cell i are considered adjacent to cell i. As an example, *C[9]* from Figure 1 below is the set *{6, 8, 9}*.

Figure 1: Search Grid (3x3) with *C[9]* Shaded



Though the grid succeeds at segmenting the search space, it does not imply that the search space is discrete, rather that it is continuous. For this reason, the model restricts the searcher and the target to moving between the cells in the neighborhood of their current location only to prevent path discontinuities. In each period, the searcher or target may stay in the current cell or move at most one cell away from their current location. This movement complicates the searcher's strategy by forcing him to consider not only the target's likely location in the current period, but also in future periods in order to maximize the probability of finding the target before the end of the game.

One can further understand the need for path constraints as follows: once the searcher or target has entered the search grid they must move through it in a continuous path. Neither party has the option of skipping entire cells on their way through the grid. The costs of searching a given cell while passing through are negligible, such that it always makes sense for the searcher to conduct a search, even if he is simply passing through a cell on his way to one with higher probability of containing the target. Time periods are constructed such that neither the searcher nor the target may move more than one cell away from their current location in the time between periods.

TARGET MOTION

This model takes the perspective of the searcher, and solves for the optimal path from there. As a result, the target's movements through the search grid must be treated exogenously. In this case it is assumed that the target's motion follows a Markov process independent of the searcher's path decisions. These assumptions allow the target more freedom as opposed to defining a single path for the target to take. Essentially this method models the situation where the target makes observations at each point in the game and acts upon them in an effort to minimize his probability of being caught.

Another benefit of this specification is that the searcher can easily put himself in the shoes of the target and identify a likely description of the target's strategy. If the searcher determines at any point in the game that his model for the target's motion can be improved, the change is simple, quick, and relatively low-cost. This ability to make nearly real-time changes makes the model quite valuable in real-world applications.

The target faces the same path constraints as the searcher. Thus the target cannot move from his current location, *s*, to a cell outside of *C[s]* in a single period.

Eagle refers to the target's motion as a Partially Observable Markov Decision Process (POMDP), and this characterization is maintained here. The model clearly demonstrates the Markov property, "which intuitively means that all the information useful to predict the next time period is contained in the information available in the current time period."[3] The partially observable portion stems from the fact that this information is not known with certainty. In the case of a strict Markov Decision Process the location of the target would be fully observable at the end of each period. Here, however, the only period in which this holds is Period 0. In all subsequent periods, the target's location is described by a vector of probabilities. Thus the searcher has an idea as to where the target may be, but never complete information.

EXAMPLE MODEL SPECIFICS

For illustrative purposes, the rest of this technical paper examines a specific instance of the general search model outlined above. In this example the search grid is 3x3 (see Figure 1 on page 5), the searcher starts in Cell 1, and the target was last seen in Cell 9. The game terminates in one of two situations: when

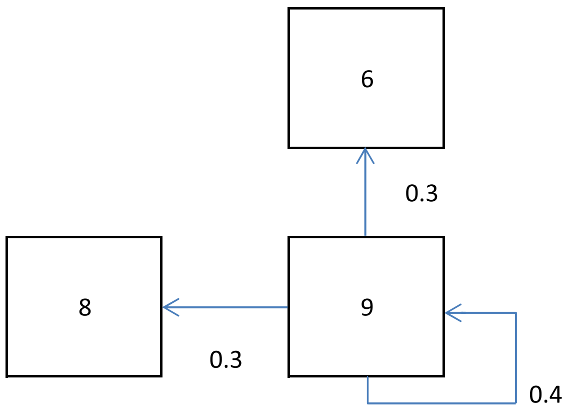
1. the target is found and attacked; or
2. the target successfully evades capture for nine consecutive periods.

3. Anderson, Patrick L, (2013). *The Economics of Business Valuation*. Stanford University Press. page 248.

Each allowable move,[4] other than searching the same cell in consecutive periods, costs the searcher \$1. Searching the current cell again costs the searcher \$10, which includes the extra expense of hiding the fact that the agent has decided to stay in the same location (which the target would otherwise freely observe). Upon finding and attacking the target, the searcher receives a reward of \$1,000.

Target motion through the search grid is governed by the Markov process presented in Eagle's numerical example. Under this assumption, the target remains in its current location with probability 0.4 and moves to an adjacent cell with equal probability (*0.6/m*, where *m* is the number of cells in the neighborhood of the current cell, excluding the current cell). Figure 2 on page 7 depicts the target's movement from Cell 9, the last known location of the target in this example. From here, one can see that the target moves to Cell 8 with probability 0.3, Cell 6 with probability 0.3, and stays in Cell 9 with probability 0.4. The value functional approach outlined below requires the construction of a transition probability matrix, which here contains the probabilities of the target moving from its current location to any of the available cells in the next period. For the full transition probability matrix used and a more complete discussion of its use in this example see Appendix II on page 17.

Figure 2: Sample Target Movement from Cell 9



4. Remember "allowable moves" consist of searching any of the cells in *C[s]* where *s* is the current location of the searcher. Infinite costs are used to prohibit the search of *C[s]*.

IV. Value Functional Formulation

OVERVIEW

In order to solve optimization problems of this type using the Rapid Recursive® Toolbox, the models must first be specified in terms of a value functional equation.[5] Doing so allows the decomposition of the larger problem into many two-stage optimization subproblems. Solving each of these and stringing their solutions together leads to the identification of an optimal search path.

THE CONCEPT OF VALUE IN THIS MODEL

The first step in using the value functional approach to solve a problem like the one presented here is to define a concept of value for the model. In this case, the natural construct of value stems from the benefit to the searcher of apprehending the target and the related search costs. In each period the searcher chooses one cell to search. Doing so, he incurs a cost of either \$1 or \$10 (see above). Once he finds the target, however, he is rewarded with \$1,000. Thus the value for each specific period is the sum of the immediate reward (or cost) from being in that period and choosing optimal action and the discounted expected value from all future periods. Whereas earlier methods solve the classic search problem based on maximizing the probability of catching the target, this model realistically incorporates cost and calculates an optimal path by maximizing the value over the entire game. In cases where search costs and rewards are non-negligible, this approach will assist the searcher in the important decision of just how valuable the search is. Note that the probability of catching the target directly influences the value in a given period since capturing the target is the only way for the searcher to obtain a positive reward. Thus the discounted expected value of future periods takes into account the likelihood of apprehending the target before the end of the game. A mathematical characterization of this idea follows below.

STATE AND ACTION SPACES

Another part of using the value functional approach involves specifying discrete state and action spaces to characterize the model. The state space captures all information available to the searcher at a given time, and the action space includes all possible actions the searcher may choose between.

For this model the states contain both the number of the current period and the current location of the searcher (last cell searched). Two additional states represent the situations where the target has

5. See Anderson (2013) for extensive coverage of the value functional iteration approach to optimization and valuation problems.

been found (in any period) or the game has ended. In all there are 83 states (81 to cover all of the possible period-location pairs plus the two outlined above) in this model.

The action space for the searcher in any time period includes searching any of the cells in the neighborhood of the current location, as well as the option to attack, but only if the target has been found. Notice that the action space changes based on the period and the searcher's current location. Once the game has ended, the only action available is to attack (since this is the only action that does not involve continued searching).

THE VALUE FUNCTIONAL

At each stage of the game the searcher faces the task of maximizing both the reward in the current period and the discounted expected reward from future periods. Knowing this, a value functional can be created as follows:

Let *f(s,x)* represent the immediate reward to the searcher from being in state *s* and choosing action *x*. Then the value of a specific state can be represented by

Value Functional Equation **(EQ 1)**

$$V(s_t) = \max_{x \in \Gamma} \left\{ f(s,x) + \beta E\left[V(s_{t+1}) \right] \right\}$$

where Γ represents the set of all action choices available to the searcher in time *t*, β is the discount factor[6] in use by the searcher, and *E[•]* is the expectation operator. Recall that based on the formulation of the model above, Γ consists of searching any of the cells in *C[s]* or "attacking" the target if the location is known.

An optimal search path maximizes $V(s_t)$ for all periods *t*. The Rapid Recursive® Toolbox allows for quick evaluation of this recursive model, which in turn provides guidance to the searcher in the early periods where the probability of finding the target in any of the neighboring cells is zero. Without the ability to look ahead into the game, all available actions in the first period appear the same. They may not be, however, as future action sets are determined by past actions and thus the choice made in the first period carries great importance by setting up future actions with higher expected rewards.

6. In general, the discount factor β can be calculated using the formula $\beta = (1+g)/(1+d)$ where *g* is the growth rate and *d* is the discount rate. This model assumes a growth rate $g=0$. See Anderson (2013, Chap. 16) for more information.

V. Results

OVERVIEW

A summary of the results from running the above model through the Rapid Recursive® Toolbox is presented below.[7] In the process of solving the problem, the software calculated the optimal action and corresponding value for each of the 83 states. Included here are a summary of the results by period when the identified optimal search path is followed and a visualization of this optimal path and the evolution of the target's location.These results are shown in full in Appendix III on page 18. For a full description of the inputs and outputs for running this model with the Rapid Recursive® Toolbox see Appendix IV on page 19.

The results below indicate, for each time period, the action that maximizes the value functional from the Equation (EQ 1) for that time period. There is an inherent trade-off[8] in this equation between the immediate reward (the probability of finding the target now) and discounted future rewards (the probability of finding the target in any of the remaining periods, given a certain action in the current period). Thus the optimal move simultaneously maximizes the searcher's chances of finding the target in the current period while also putting the searcher in the best position to find the target in a later period should the current-period search fail to locate the target.

A NOTE ON EAGLE'S MODEL

When James Eagle published his paper on this topic in 1984, his solution method consisted of running custom FORTRAN H code on an IBM 3033. IBM offers the following description of the 3033 on their website:

> Back when Jimmy Carter was the newly inaugurated President of the United States, the industry publication *Datamation* termed it "the big bombshell" of IBM's spring product announcements. *THINK* magazine later simply dubbed it—"The Big One."
>
> The two publications were referring to the IBM 3033, the company's new top-of-the-line processor. When it was rolled out on March 25, 1977, the 3033 eclipsed the internal operating speeds of the company's previous flagship—the System/370 Model 168-3—by 1.6 to 1.8 times.[9]

7. All code pertaining to this specific example was created by the author.
8. See Anderson's (2013, Chap. 15) discussion of "Euler tension" for more information.

Even with the best computer available, Eagle's method "required a total of 19.3 CPU minutes"—minutes, not seconds!—to find the optimal search paths. Keep in mind that this was not due to inefficient methods on Eagle's part: his paper outlines an elegant and concise methodology for streamlining the calculations required by this model. In contrast, using the Rapid Recursive® Toolbox today requires only fractions of a second to find an optimal path.

A COMPARISON TO EAGLE'S RESULTS

Eagle presented a complex formulation of this classic search problem, along with a new method for solving it. While his methods are not directly matched here, a comparison is presented in Figure 3 below, to demonstrate the close similarities between the solution methods.

The results from the Rapid Recursive® Toolbox rely on the same starting points for the searcher and target as Eagle's model. Furthermore, the Markov process underlying the target's motion is the same in both models. Eagle's model, however, updates the probability of finding the target in a given cell in the next period after every search. Thus, in Eagle's model, the probability of finding the target in Cell 6 in Period 3 is the probability that the target is in that cell during that period *given that* the target was not in the cells searched in the first two periods. The model developed in this paper does not make these updates.

Figure 3: Comparison of Results: Rapid Recursive® (2013) and FORTRAN H (1984)

Period	Rapid Recursive® Policy	FORTRAN H Policy
1	2	2 or 4
2	3	3, 5, or 7
3	6	6 or 8
4	9	9
5	6	6 or 8
6	5	5
7	6	6 or 8
8	5	5
9	5	6 or 8
10	5	5

The observant reader will quickly notice that whenever there is an unambiguously optimal policy this model agrees with Eagle's. There is one case, however, in Period 9, where the optimal decision identified by this model does not align with any of those laid out by Eagle. While it is true that the search options identified by Eagle (Cells 6 and 8) do have identical probabilities of containing the target in

9. "IBM's 3033," <www-03.ibm.com/ibm/history/exhibits/3033/3033_intro.html>, accessed April 4, 2013.

Period 9, Cell 5 actually has a slightly higher chance based on the underlying probabilities in this model and is thus reported as the optimal cell to search.

OUTPUT FIGURES AND DISCUSSION

In Figure 5 on page 14, one can clearly see the tendency of the target toward Cell 5. Notice that the optimal path reaches Cell 5 in Period 8 and stays there for Period 9 even though doing so is ten times more costly than searching any other neighboring cell. This stems from the fact that the by Period 9 the likelihood of finding the target in Cell 5 is so much higher than that of finding him in any other cell that it is worth paying a penalty of at least $9 to search that cell.

The summary table in Figure 4 on page 13 presents the results of the model for each period when the identified optimal path is followed. Consistent with the specifications of the model from Section II, the table is constructed under the assumption that the searcher's starting location is Cell 1. One can quickly confirm this by looking at the pane for Period 1 in Figure 5 on page 14 and noticing that the optimal choice with no constraints would be to search Cell 9.

Comparing the relative values across periods in Figure 4 on page 13 provides an interesting perspective on this problem. The values can be interpreted as an indicator of the probability of apprehending the target before the game ends given that the game is in the indicated period (see "The Concept of Value in this Model" on page 8 for a review of this idea). As the game progresses, the probability of the target being in any one specific cell diminishes rapidly, until the final period when the cell with the highest probability of containing the target, Cell 5, only yields a roughly 17% chance for the searcher to win the game. The values peak in Period 3 and begin a steady decline in Period 4. This represents the benefits of smart decisions in the early periods, which place the searcher in a strong position by Period 3. The value of $441 indicates a nearly 50% chance of the searcher apprehending the target between periods three and nine.

As a note, there are a few periods where multiple actions have the same value (both the immediate reward and the discounted future rewards are the same for multiple policy options). The software used to solve this recursive model breaks these "ties" by choosing the first such action. Assume, for example, that in a certain period searching Cell 2 has the same value as searching Cell 4. In such a case, this solution method would choose to search Cell 2. Thus the search path represented here is not the unique optimal solution, but one of many possible optimal solutions. No one of these tie-breakers is any better than the others, but some method must be established in cases where only one optimal path is reported.

Figure 4: Summary of Outputs by Period, Rapid Recursive® 2013

Period	Value	Policy
1	$351	2
2	$422	5
3	$441	6
4	$401	9
5	$376	6
6	$336	5
7	$287	6
8	$277	5
9	$129	5
Found	$1,000	Attack
Game Over	$0	Attack

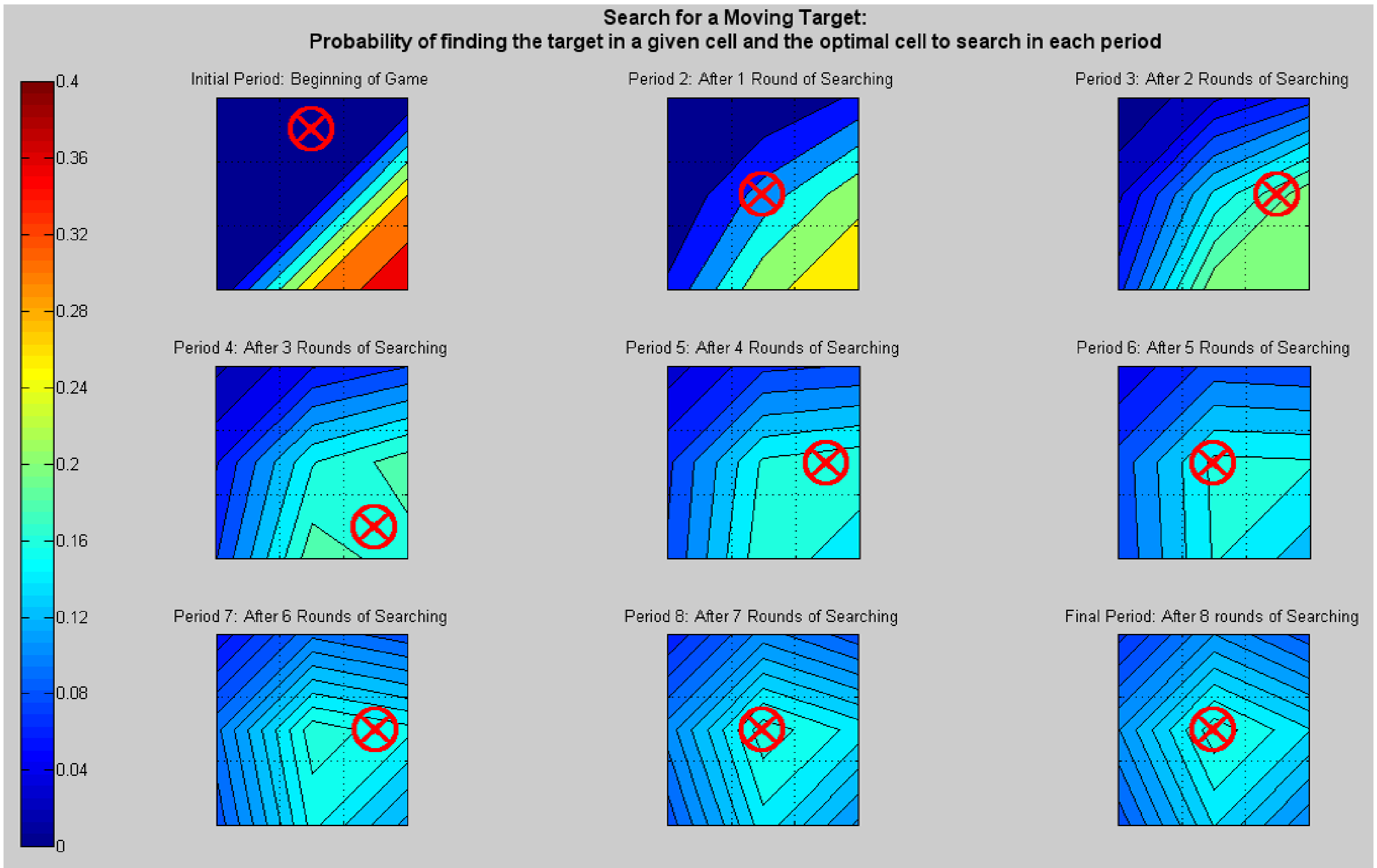
Figure 5 on page 14 shows the output obtained by running this model through the Rapid Recursive® Toolbox. The grids show the probability of the target being located in each cell for all nine periods of the game. Note that darker colors indicate lower probabilities. The red symbol on each plot indicates the optimal search decision for that period, assuming the target's location is still unknown. Cell numbers have been omitted, but are understood to be consistent with those in Figure 1 on page 5.

At first glance, one quickly notices that the probability distribution of the target's location seems to even out as the game goes on. This is indeed true, and can be attributed to the fact that the searcher's information about the target's starting location essentially becomes stale as the game goes on. The deeper into the game we go, the less of an idea the searcher has about where the target may be. Another explanation for this is that in Period 1 the target only has access to three cells: 6, 8, and 9. By Period 9, however, the target could potentially be anywhere on the grid, meaning the probability for his location is distributed over all nine cells, an area three times larger than that of Period 1.

Figure 5 on page 14 also clearly displays the tendency of the target toward Cell 5. This can be confirmed numerically by calculating the limiting distribution[10] of the Markov chain for the target's motion. Doing so shows that the target will most frequently occupy Cell 5 (though this only accounts for his location less than 20% of the time). Thus, as the game goes on and the searcher has less and less of an idea where the target may be, the best policy is to repeatedly search Cell 5, since the model of the target's motion predicts he will eventually show up there with the highest likelihood.

10. See Sigman (2009) for a comprehensive introduction to the concept of limiting distributions for Markov chains.

Figure 5: Optimal Search Path and Evolution of Target Location



The plots above show the game in each of its nine time periods. The contour map for each period represents the probability of finding the target in the given cell during that period while the red symbol indicates the optimal cell to search, as determined by the Rapid Recursive® Toolbox. Note that the likelihood of finding the target diminishes quickly as the game progresses. This is due to the fact that as time goes by the target has the potential to occupy a larger set of cells. In the initial period, the target is limited to the three cells adjacent to Cell 9. By Period 9, however, it is possible for the target to be in any cell on the grid.

Created by Rapid Recursive® Toolbox

VI. Conclusion

This technical paper presented the idea of using Partially Observable Markov Decision Processes to model target motion in the search for a moving target. In addition, a specific example demonstrated the use of the Rapid Recursive® Toolbox to solve such models in only a few seconds—much faster than the almost 20 minutes on a mainframe computer required just 30 years ago. Building on the model presented by Eagle (1984), this paper included search costs and rewards to create a more realistic picture of maximizing value in terms of expected benefits rather than simply the probability of catching the target.

Though this technical paper presented only one specific example, the model above can be easily generalized and applied in many fields. Extensions can easily be made to incorporate more periods into the game, or to obscure the information of the target's starting location (perhaps by delaying the start of the game). The practical applications of this model reach into many fields, including law enforcement, counterterrorism, animal control, and more. Potential searchers include humans, land- or water-bound robots, and manned or unmanned aerial vehicles.

Appendix I: References

SOFTWARE

Matlab® is a product of The Mathworks; http://www.math-works.com.

The Rapid Recursive® Toolbox is a product of Supported Intelligence LLC; http://www.supportedintelligence.com.

SCHOLARLY PUBLICATIONS

Anderson, Patrick L. (2013) *The Economics of Business Valuation.* Stanford University Press.

Eagle, James N. (1984) "The Optimal Search for a Moving Target When the Search Path is Constrained." *Operations Research,* Vol. 32 No. 5 (Sep. - Oct., 1984), pp 1107-1115.

"IBM's 3033," <www-03.ibm.com/ibm/history/exhibits/3033/3033_intro.html>, accessed April 4, 2013.

Ishida, Toru and Richard E. Korf. (1991). "Moving Target Search." IJCAI-91, pp. 204-210.

Melax, Stan. (1993). "New Approaches to Moving Target Search." AAAI Technical Report FS-93-02, pp. 30-38.

Sigman, Karl. (2009). Lecture Notes[PDF]. <www.columbia.edu/~ks20/stochastic-I/stochastic-I-MCII.pdf>, accessed April 5, 2013.

Stone, Lawrence D. and Joseph B. Kadane. (1981) "Optimal Whereabouts Search for a Moving Target." *Operations Research,* Vol. 29, No. 6 (Nov. - Dec., 1981), pp. 1154-1166.

Thomas, Lyn C. and Alan R. Washburn. (1991) "Dynamic Search Games." *Operations Research*, Vol 39, No. 3 (May - Jun., 1991), pp. 415-422.

Appendix II: Target Location Transition Probability Matrix

Target Transition Matrix

		Next Period Cell								
		1	2	3	4	5	6	7	8	9
Current Cell	1	0.4	0.3		0.3					
	2	0.2	0.4	0.2		0.2				
	3		0.3	0.4			0.3			
	4	0.2			0.4	0.2	0	0.2		
	5		0.15		0.15	0.4	0.15		0.15	
	6			0.2	0	0.2	0.4			0.2
	7				0.3			0.4	0.3	
	8					0.2		0.2	0.4	0.2
	9						0.3		0.3	0.4

Note: Omitted values are equal to 0.

The above matrix depicts the Markov process governing the target's motion in each period. Given the current cell *s*, the target stays in the same place with probability 0.4 and moves to any of the other cells in *C[s]* with equal probability (notice the exact value of this probability changes with the size of *C[s]*).

In order to solve the example using the Rapid Recursive® Toolbox a different transition probability matrix was created, representing the probability of moving from a current state to a certain next period state, given a certain action choice. Instead of the 9x9 matrix above, this new transition matrix had dimensions 83x83x10 to account for all of the possible states and actions. In a given current state (row), the game moves to the "Found" state with probability equal to the probability of the target being located in the searched cell in the given period.[11] From the same current state, the game moves to the next period and the cell searched with a probability equal to one minus the probability of moving to the "Found" state. From the ninth (terminal) period, the game moves to the "Game Over" state if the target is not found.

11. This probability can be found in the sth row of the vector **n**, where $\mathbf{n} = \mathbf{A}^t\mathbf{i}$ with **A** being the target transition matrix above, t the current period, and i a vector capturing the target's last known location. Here s represents the cell to be searched.

Appendix III: Full Results Table

NOTE

When reading the results table, keep in mind that the “State” column represents the period-location pair for the searcher. “1-1” indicates that the game is in its first period and the searcher is currently in cell 1. Also recall that some states will never occur due to the path constraints outlined above.

Full Table of Outputs

State	Value	Policy
1-1	350.84	Search 2
1-2	397.84	Search 3
1-3	538.78	Search 6
1-4	397.84	Search 5
1-5	538.78	Search 6
1-6	576.22	Search 9
1-7	538.78	Search 8
1-8	576.22	Search 9
1-9	567.22	Search 9
2-1	340.59	Search 2
2-2	422.20	Search 5
2-3	478.61	Search 6
2-4	422.20	Search 5
2-5	478.61	Search 6
2-6	496.77	Search 9
2-7	478.61	Search 8
2-8	496.77	Search 9
2-9	487.77	Search 9
3-1	346.15	Search 2
3-2	409.91	Search 5
3-3	440.73	Search 6
3-4	409.91	Search 5
3-5	440.73	Search 6
3-6	441.49	Search 9
3-7	440.73	Search 8
3-8	441.49	Search 9
3-9	440.73	Search 6
4-1	337.21	Search 2
4-2	394.79	Search 5
4-3	407.82	Search 6
4-4	394.79	Search 5
4-5	407.82	Search 6
4-6	400.61	Search 9
4-7	407.82	Search 8
4-8	400.61	Search 9
4-9	407.82	Search 6
5-1	320.64	Search 2
5-2	369.53	Search 5
5-3	376.13	Search 6
5-4	369.53	Search 5
5-5	376.13	Search 6
5-6	369.53	Search 5
5-7	376.13	Search 8
5-8	369.53	Search 5
5-9	376.13	Search 6
6-1	291.99	Search 2
6-2	336.27	Search 5
6-3	336.33	Search 6
6-4	336.27	Search 5
6-5	336.33	Search 6
6-6	336.27	Search 5
6-7	336.33	Search 8
6-8	336.27	Search 5
6-9	336.33	Search 6
7-1	250.83	Search 2
7-2	289.37	Search 5
7-3	286.82	Search 6
7-4	289.37	Search 5
7-5	286.82	Search 6
7-6	289.37	Search 5
7-7	286.82	Search 8
7-8	289.37	Search 5
7-9	286.82	Search 6
8-1	188.67	Search 2
8-2	227.19	Search 5
8-3	218.64	Search 6
8-4	227.19	Search 5
8-5	218.64	Search 8
8-6	227.19	Search 5
8-7	218.64	Search 8
8-8	227.19	Search 5
8-9	218.64	Search 8
9-1	90.19	Search 2
9-2	137.82	Search 5
9-3	116.14	Search 6
9-4	137.82	Search 5
9-5	128.82	Search 5
9-6	137.82	Search 5
9-7	116.14	Search 8
9-8	137.82	Search 5
9-9	116.14	Search 6
Found	1000.00	Attack
Game Over	0.00	Attack

Appendix IV:Input and Output Data for Recursive Model

Data structures in Matlab® provide a convenient means for storing variables of differing types when all of the variables are related to the same model. Each variable is stored in a field of the structure. Examples of fields from the standard input structure used with the Rapid Recursive® Toolbox include: S, the number of states, stored as a number; statelabels,[12] a list of the labels for each state, stored as a cell; and the R and P matrices, stored as matrices. After defining each of these, they are assigned to the appropriate field of the input structure for safekeeping. When it comes time to run the model using the Rapid Recursive® Toolbox, the only necessary input argument is the input structure, rather than all of the individual variables. In addition to convenience, storing the variables in this way streamlines the error-checking process conducted by the toolbox.

Key fields from the input structure for the model in this paper are shown below.

Figure 1: Description of Input Structure

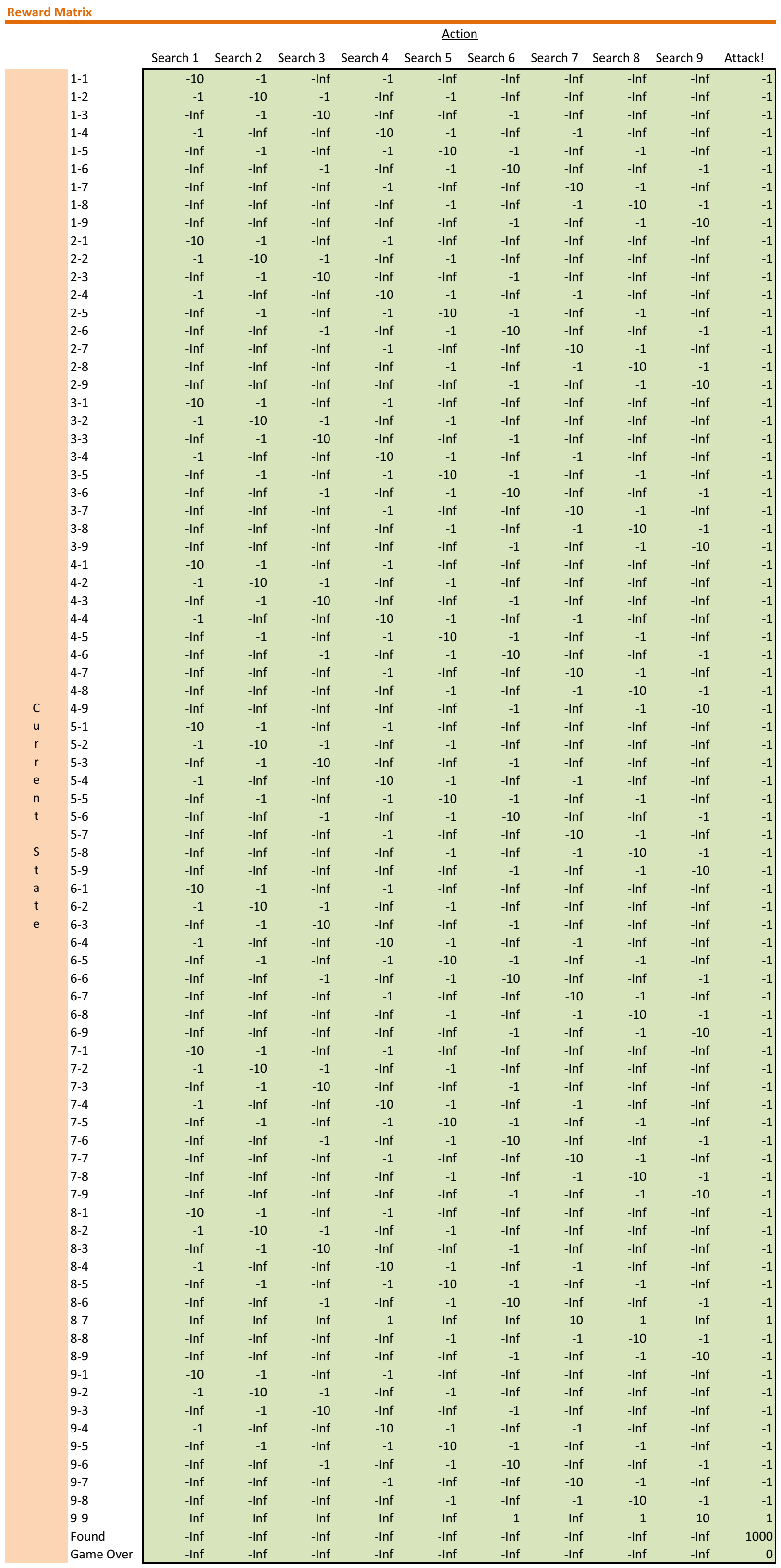
Field	Value or Type
desc	'Optimal Strategy in the Search for a Moving Target'
S	83
statelabels	1-1, 1-2, . . . , 9-8, 9-9, Found, Game Over
A	10
actionlabels	Search 1, Search 2, . . . , Search 9, Attack!
formP	'Search for a moving target'
formR	'Search for a moving target'
beta	0.8333
d	0.2
g	0
verbose	FALSE
T	9
note1	'P matrix calculated within RR toolbox'
note2	'R matrix calculated within RR toolbox'

Figure 2: Description of Out Structure

Field	Value or Type
desc	'Optimal Strategy in the Search for a Moving Target'
Input	Input Structure
iterations	11
calculationtime	0.0066
algorithm	'Value function iteration'

12.Note that the first 81 state labels are of the form: period - last cell searched. Thus 2-1 indicates that the game is in the second period and the searcher decided to search Cell 1 in the first period.

Figure 3: Rewards Matrix: Input.R



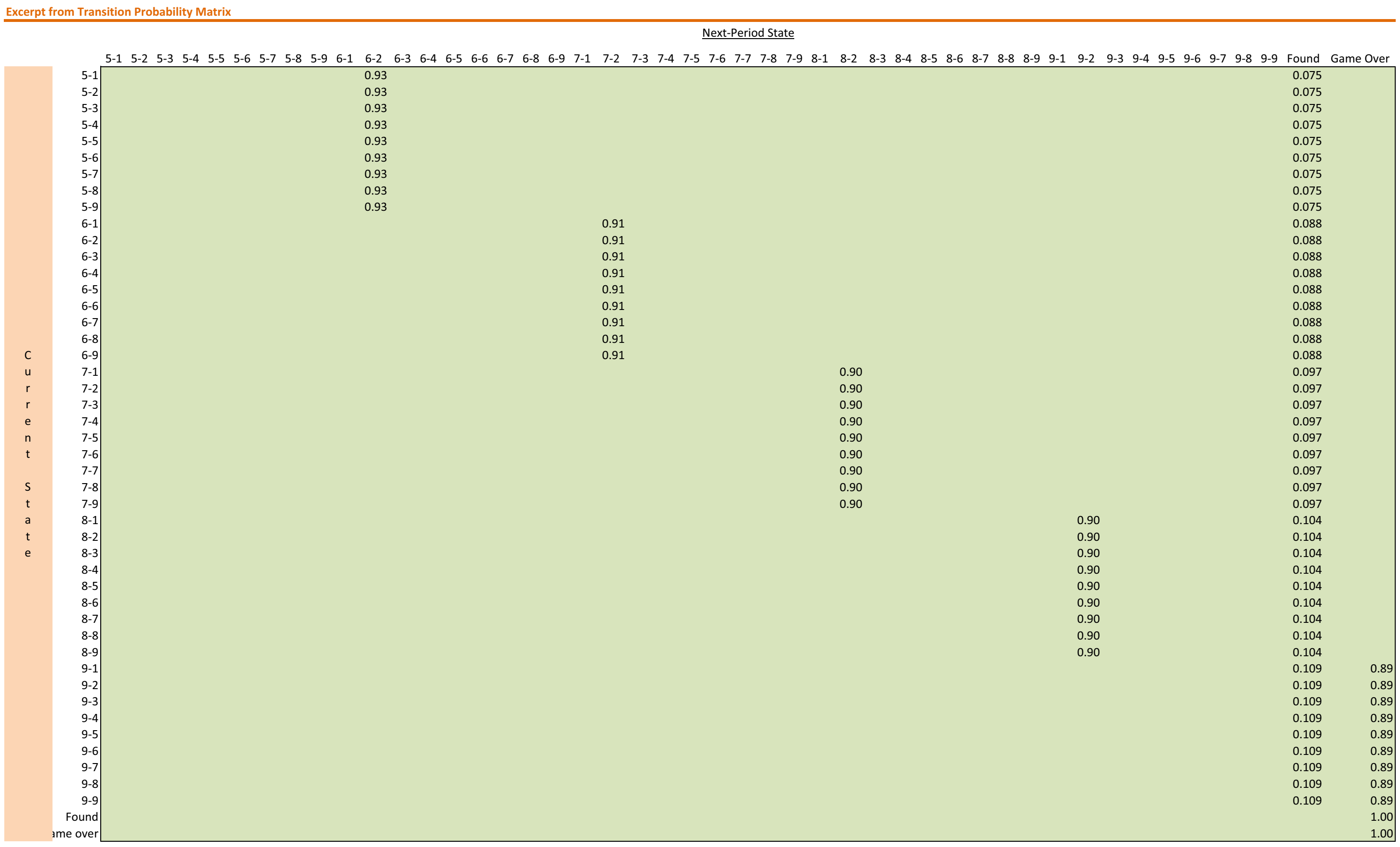
Reward Matrix

	Action									
Current State	Search 1	Search 2	Search 3	Search 4	Search 5	Search 6	Search 7	Search 8	Search 9	Attack!
1-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
1-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
1-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
1-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
1-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
1-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
1-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
1-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
1-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
2-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
2-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
2-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
2-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
2-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
2-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
2-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
2-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
2-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
3-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
3-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
3-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
3-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
3-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
3-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
3-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
3-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
3-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
4-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
4-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
4-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
4-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
4-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
4-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
4-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
4-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
4-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
5-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
5-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
5-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
5-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
5-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
5-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
5-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
5-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
5-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
6-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
6-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
6-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
6-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
6-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
6-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
6-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
6-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
6-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
7-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
7-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
7-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
7-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
7-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
7-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
7-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
7-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
7-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
8-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
8-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
8-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
8-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
8-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
8-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
8-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
8-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
8-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
9-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-Inf	-1
9-2	-1	-10	-1	-Inf	-1	-Inf	-Inf	-Inf	-Inf	-1
9-3	-Inf	-1	-10	-Inf	-Inf	-1	-Inf	-Inf	-Inf	-1
9-4	-1	-Inf	-Inf	-10	-1	-Inf	-1	-Inf	-Inf	-1
9-5	-Inf	-1	-Inf	-1	-10	-1	-Inf	-1	-Inf	-1
9-6	-Inf	-Inf	-1	-Inf	-1	-10	-Inf	-Inf	-1	-1
9-7	-Inf	-Inf	-Inf	-1	-Inf	-Inf	-10	-1	-Inf	-1
9-8	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1	-1
9-9	-Inf	-Inf	-Inf	-Inf	-Inf	-1	-Inf	-1	-10	-1
Found	-Inf	-Inf	-Inf	-Inf	-Inf	-Inf	-Inf	-Inf	-Inf	1000
Game Over	-Inf	-Inf	-Inf	-Inf	-Inf	-Inf	-Inf	-Inf	-Inf	0

Note the use of "-Inf" to prohibit certain actions, as described in "Path constraints" on page 4.

Figure 4 below presents a small excerpt from the transition pro ability matrix. Given the specifics of this model, the full transition probability matrix has size 83x83x10, which is much too large to show here. As such, Figure 4 below shows only a portion of this matrix, corresponding to the action “Search 2,” to give the reader a taste of what the transition probability matrix looks like. For readability, zero-valued cells are left blank.

Figure 4: An Excerpt from the Transition Probability Matrix: Input.P



Excerpt from Transition Probability Matrix

Next-Period State

Current State	5-1	5-2	5-3	5-4	5-5	5-6	5-7	5-8	5-9	6-1	6-2	6-3	6-4	6-5	6-6	6-7	6-8	6-9	7-1	7-2	7-3	7-4	7-5	7-6	7-7	7-8	7-9	8-1	8-2	8-3	8-4	8-5	8-6	8-7	8-8	8-9	9-1	9-2	9-3	9-4	9-5	9-6	9-7	9-8	9-9	Found	Game Over
5-1											0.93																																			0.075	
5-2											0.93																																			0.075	
5-3											0.93																																			0.075	
5-4											0.93																																			0.075	
5-5											0.93																																			0.075	
5-6											0.93																																			0.075	
5-7											0.93																																			0.075	
5-8											0.93																																			0.075	
5-9											0.93																																			0.075	
6-1																				0.91																										0.088	
6-2																				0.91																										0.088	
6-3																				0.91																										0.088	
6-4																				0.91																										0.088	
6-5																				0.91																										0.088	
6-6																				0.91																										0.088	
6-7																				0.91																										0.088	
6-8																				0.91																										0.088	
6-9																				0.91																										0.088	
7-1																													0.90																	0.097	
7-2																													0.90																	0.097	
7-3																													0.90																	0.097	
7-4																													0.90																	0.097	
7-5																													0.90																	0.097	
7-6																													0.90																	0.097	
7-7																													0.90																	0.097	
7-8																													0.90																	0.097	
7-9																													0.90																	0.097	
8-1																																						0.90								0.104	
8-2																																						0.90								0.104	
8-3																																						0.90								0.104	
8-4																																						0.90								0.104	
8-5																																						0.90								0.104	
8-6																																						0.90								0.104	
8-7																																						0.90								0.104	
8-8																																						0.90								0.104	
8-9																																						0.90								0.104	
9-1																																														0.109	0.89
9-2																																														0.109	0.89
9-3																																														0.109	0.89
9-4																																														0.109	0.89
9-5																																														0.109	0.89
9-6																																														0.109	0.89
9-7																																														0.109	0.89
9-8																																														0.109	0.89
9-9																																														0.109	0.89
Found																																															1.00
ame over																																															1.00

A NOTE ON READING THE P MATRIX

When reading the above matrix, keep in mind that each row represents a current state and each column a future state. Notice that this matrix is upper triangular, which indicates that the game never moves back to a previous state. Furthermore, since the entries in this matrix represent probabilities, they must sum to one for each current state (row).