2001

Dynamic Programming Solution for a Class of Pursuit Evasion Problems: The Herding Problem

Pushkin Kachroo
University of Nevada, Las Vegas, pushkin@unlv.edu

S. A. Shedied
Virginia Polytechnic Institute and State University

J. S. Bay
Virginia Tech

H. Vanlandingham
Virginia Tech

Dynamic Programming Solution for a Class of Pursuit Evasion Problems: The Herding Problem

Pushkin Kachroo, *Senior Member, IEEE*, Samy A. Shedied, *Student Member, IEEE*, John S. Bay, *Senior Member, IEEE*, and Hugh Vanlandingham, *Life Senior Member, IEEE*

***Abstract*—A herding dog and sheep problem is studied where the agent "dog" is considered the control action for moving the agent "sheep" to a fixed location using the dynamics of their interaction. The problem is solved for the deterministic case using dynamic programming. Proofs are provided for the correctness of the algorithms. The algorithm is analyzed for its complexity. A software package developed for experimentation is described.**

***Index Terms*—Dog–sheep, dynamic programming, herding, value function.**

NOMENCLATURE

$x_d(k)$	x coordinate of the dog position at time instance k.
$y_d(k)$	y coordinate of the dog position at time instance k.
$x_s(k)$	x coordinate of the sheep position at time instance k.
$y_s(k)$	y coordinate of the sheep position at time instance k.
$\mathbf{x}(k)$	state vector given by $\mathbf{x(k)=[x_s(k)y_s(k)x_d(k)y_d(k)]}$ at time instance k.

I. INTRODUCTION

THE problem of herding has not been well studied in the past as is demonstrated by lack of relevant literature on the topic. However, related topic of pursuit–evasion problems has been studied to some extent [1]–[3]. Many of the pursuit–evasion problems are set in the differential games theory where the modeling of the system is done using differential or difference equations. A typical termination state for these games is capture, which is very different from the termination state for the herding problem where the sheep needs to enter the pen. Some problems have been studied in stochastic framework and in discrete time [4].

In this paper we present a dynamic programming based solution to the dog sheep herding problem, where the sheep has been modeled as a passive entity, responding only to dog position. We present the dynamics of the problem then give two different algorithms to the optimal solution. The solutions are proven to be correct and then simulations are performed to illustrate some example runs.

Manuscript received June 30, 2000; revised February 16, 2001. This work was sponsored by the Office of Naval Research under Project N00014-98-1-0779. This paper was recommended by Associate Editor R. Popp.

P. Kachroo, S. A. Shedied, and H. Vanlandingham are with the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA (e-mail: pushkin@vt.edu; sshedied@vt.edu; hughv@vt.edu).

J. S. Bay is with the Information Technology Office, Defense Advanced Research Projects Agency (DARPA/ITO), Arlington, VA 22203 USA (e-mail: john.bay@ieee.org).

Publisher Item Identifier S 1094-6977(01)03528-3.

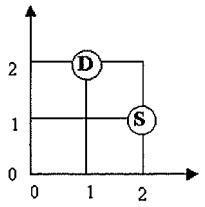

Fig. 1. The 3 × 3 dog–sheep problem grid.

II. 3 × 3 GRID DOG-SHEEP PROBLEM

We consider the dog–sheep problem in a 3 × 3 grid as shown in Fig. 1. The dog can occupy one of the nine positions and so can the sheep. Therefore, there are 81 states in the system. The aim of the dog is to make the sheep go to the pen that is the (0, 0) state for the sheep.

For the 3 × 3 dog sheep problem, we have $\forall k\, x_d(k) \in [0,1,2]$, $y_d(k) \in \{0,1,2\}$, $x_s(k) \in \{0,1,2\}$, and $y_s(k) \in \{0,1,2\}$. However, the dog and the sheep can not have the same location on the grid as their initial positions. It can be proven that if they have different initial positions, then based on the allowable actions of both (as described later), they can never end up on the same location. There is a cost of one unit for each step (horizontal or vertical or diagonal) of a dog as well as of a sheep. The aim of the dog is to move the sheep to the pen, i.e., to the (0, 0) coordinate, with the least cost. Fig. 1 shows the 3 × 3 grid for the dog–sheep problem.

Definition 1: Equilibrium State of the Sheep: The sheep is in an equilibrium state when given a time instant T the following condition is satisfied:

$$\begin{aligned}&\forall k \geq T,\\ &\text{if } x_d(k)=x_d(T) \text{ and } y_d(k)=y_d(T)\\ &\text{then } x_s(k)=x_s(T) \text{ and } y_s(k)=y_s(T).\end{aligned}$$

Definition 2: Final Equilibrium State of the Sheep: The sheep is in the final equilibrium state when given a time instant T the following condition is satisfied:

$$\begin{aligned}&\forall k \geq T,\\ &\text{if } x_d(k)=x_d(T) \text{ and } y_d(k)=y_d(T)\\ &\text{then } x_s(k)=0 \text{ and } y_s(k)=0.\end{aligned}$$

Definition 3: Positive Successor Function: Positive successor function is a function given by $PS(\cdot): X=\{0,1,2\} \rightarrow Y=\{1,2\}$; PS(0) =1, PS(1) = 2, and PS(2) = 2

Definition 4: Negative Successor Function: Negative successor function is a function given by $NS(\cdot): X=\{0,1,2\} \rightarrow$

$Z=\{0,1\}$; NS(0) = 0, NS(1) = 0, and NS(2) = 1. The following rules generate the dynamics of the sheep and dog movements.

1) $\forall k\ x_d(k)\in\{0,1,2\}$, $y_d(k)\in\{0,1,2\}$, $x_s(k)\in\{0,1,2\}$, and $y_s(k)\in\{0,1,2\}$.
2) The dog can only move when sheep is in an equilibrium state.
3) The dog can only move one step in one time instant. That step can be in horizontal, vertical, or diagonal direction. The sheep can also only move one step in horizontal, vertical, or diagonal direction.
4) The sheep moves based on the following rules.
   a) *Far Condition:* **If** $x_s(k) < NS(x_d(k))$ or $x_s(k) > PS(x_d(k))$ or $y_s(k) < NS(y_d(k))$ or $y_s(k) < NS(y_d(k))$ **then** $x_s(k+1)=x_s(k)$ and $y_s(k+1)=y_s(k)$.
   b) *Left Top Corner Dog Right Condition:* **If** $x_s(k)=0$ and $x_d(k)=PS(x_s(k))$ and $y_s(k)=y_d(k)=2$ **then** $x_sk+1)=x_s(k)$ and $y_s(k+1)=NS(y_s(k))$.
   c) *Left Top Corner Dog Down Condition:* **If** $x_s(k)=x_d(k)=0$ and $y_s(k)=2$ and $y_d(k)=NS(y_s(k))$ **then** $x_s(k+1)=PS(x_s(k))$ and $y_s(k+1)=y_s(k)$.
   d) *Right Top Corner Dog Left Condition:* **If** $x_s(k)=2$ and $x_d(k)=NS(x_s(k))$ and $y_s(k)=y_d(k)=2$ **then** $x_s(k+1)=x_s(k)$ and $y_s(k+1)=NS(y_s(k))$.
   e) *Right Top Corner Dog Down Condition:* **If** $x_s(k)=x_d(k)=2$ and $y_s(k)=2$ and $y_d(k)=NS(y_s(k))$ **then** $x_s(k+1)=NS(x_s(k))$ and $y_s(k+1)=y_s(k)$.
   f) *Left Bottom Corner Dog Right Condition:* **If** $x_s(k)=0$ and $x_d(k)=PS(x_s(k))$ and $y_s(k)=y_d(k)=0$ **then** $x_s(k+1)=x_s(k)$ and $y_s(k+1)=PS(y_s(k))$.
   g) *Left Bottom Corner Dog Up Condition:* **If** $x_s(k)=x_d(k)=0$ and $y_s(k)=0$ and $y_d(k)=PS(y_s(k))$ **then** $x_s(k+1)=PS(x_s(k))$ and $y_s(k+1)=y_s(k)$.
   h) *Right Bottom Corner Dog Left Condition:* **If** $x_s(k)=2$ and $x_d(k)=NS(x_s(k))$ and $y_s(k)=y_d(k)=0$ **then** $x_s(k+1)=x_s(k)$ and $y_s(k+1)=PS(y_s(k))$.
   i) *Right Bottom Corner Dog Up Condition:* **If** $x_s(k)=x_d(k)=2$ and $y_s(k)=0$ and $y_d(k)=PS(y_s(k))$ **then** $x_s(k+1)=NS(x_s(k))$ and $y_s(k+1)=y_s(k)$.
   j) *Other Conditions:*

If (a) to (i) are not satisfied and $x_s(k)=NS(x_d(k))$, **then** $x_s(k+1)=NS(x_s(k))$.

If(a) to (i) are not satisfied and $x_s(k)=PS(x_d(k))$, **then** $x_s(k+1)=PS(x_s(k))$.

If (a) to (i) are not satisfied and $y_s(k)=NS(y_d(k))$, **then** $y_s(k+1)=NS(y_s(k))$.

If (a) to (i) are not satisfied and $y_s(k)=PS(y_d(k))$, **then** $y_s(k+1)=PS(y_s(k))$.

Theorem 1: There are six final equilibrium states of the 3×3 dog–sheep problem.

Proof: Fig. 2 shows the six final equilibrium states. The figure implies that the dog can be in any of the six positions to obtain a final equilibrium state. We can prove that the state when the dog is in (1, 1) position is a final equilibrium state because

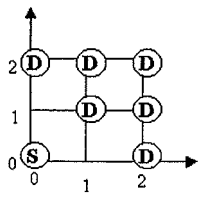

Fig. 2. Six final equilibrium states.

of the 4(j) rule of the dynamics. For the other ones we can prove the same by using the 4(a) rule.

For each given equilibrium-state, the dog is free to choose its next move based on a finite set of possible actions. This finite set is a function of the state $\mathbf{x}$. If the state is a nonequilibrium state then the dog is not allowed to move in that time instant. The sheep will move from the nonequilibrium-state to another state that could be (in general) another nonequilibrium-state or an equilibrium-state.

Theorem 2: If for any positive integer k, $x_s(k)\neq x_d(k)$, and $y_s(k)\neq y_d(k)$ then for any $t\geq k$, the following two statements cannot be simultaneously false: $x_s(t)\neq x_d(t)$ and $y_s(t)\neq y_d(t)$.

Proof: This can be easily proven by noting that in order for both statements to be false at time t, the system would have to be in equilibrium condition, and then the dog would have to move to acquire the same coordinates as those of the sheep. However, due to the constraint on the motion of the dog, i.e., that the dog can only move when the system is in a nonequilibrium state, the theorem is proven.

Let $U(x_s,y_s)$ be the discrete set of actions available to the controller (dog) when the system is in the state $\mathbf{x}$. The controller (dog) defines a *policy* $\mu:\mathbf{x}\rightarrow U$ that is a function from the state to actions. This defines a feedback control policy. We also define a *value function* $V_\mu(\mathbf{x})$, which is the sum of all future instantaneous costs given that the initial state of the system is $\mathbf{x}$ and the system follows the policy μ. We define *instantaneous cost* as

$$c_{\mathbf{x}}(u)=\min(|x_d(k)-x_d(k-1)|+|y_d(k)-y_d(k-1)| +|x_s(k)-x_s(k-1)|+|y_s(k)-y_s(k-1)|,1).$$

Notice that given any state $\mathbf{x}(k)$, we can find out the next state if the control action u is known. The value function $V_\mu(\mathbf{x})$ is given by

$$V_\mu(i)=\sum_{k=0}^{\infty} c_{\mathbf{x}(k)}(\mu(\mathbf{x}(k))),\quad \text{where } \mathbf{x}(0)=i.$$

1) Problem Statement: Find the *optimal policy* that minimizes the value function

$$V^*(i)=\min_{\mu} V_\mu(i).$$

This gives us the *optimal value function*. In general, optimal value function is unique but an optimal policy might not be.

III. Properties of the Digraph Associated with the Dog–Sheep Problem

We can represent the dog–sheep problem described above as a digraph $G=(V,E)$ that consists of a finite set V of vertices or

nodes representing the states of the system, and a finite set E of edges. V consists of all the possible values of the state $\mathbf{x}$. The cardinality of V denoted by $N(V)$ is 81. There exists an edge e from a state-value (node) v to w if for some k, $v=x(k)$ and $w=x(k+1)$, following the dynamics generated by the rules in Section II. The digraph is a directed network or a weighted-digraph since we associate a cost with each edge using the instantaneous cost formula from Section II. The digraph is also simple, since there are no loops or multiple edges. The adjacency matrix of the digraph is an $N(V)\times N(V)$ matrix whose diagonal elements are all zeros. The reader is referred to [5] and [6] for background reference.

Theorem 3: The instantaneous cost associated with each edge in the digraph of the dog–sheep problem is 1.

Proof: The proof of the theorem comes directly from calculating the cost for the motion of the dog and the sheep, according to their dynamics as given in Section II.

a) *Far Condition:* In this case, and by Definition 1, the sheep is at equilibrium state, so, only the dog is allowed to move:

$$x_s(k+1)=x_s(k)\ \&\ y_s(k+1)=y_s(k)$$

since only the dog is allowed to move, and for only one step. Then, the minimum distance the dog can move, will result from moving it one step in either the x direction or the y direction:

$$x_d(k+1)-x_d(k)=1 \quad \text{or} \quad y_d(k+1)-y_d(k)=1$$

Substitute with the following in the cost equation:

$$\begin{aligned} c_x(\mu)&=\min(|x_d(k+1)-x_d(k)|+|y_d(k+1)-y_d(k)| \\ &\quad +|x_s(k+1)-x_s(k)|+|y_d(k+1)-y_d(k)|,1) \\ c_x(\mu)&=\min(1+0+0+0,1)=1 \quad \text{or} \\ c_x(\mu)&=\min(0+1+0+0,1)=1. \end{aligned}$$

b) *Left Top Corner, Dog Right:* In this case, the sheep is not in an equilibrium state and, therefore, only the sheep is moving while not the dog:

$$\begin{aligned} &x_s(k)=0 \quad \text{and} \quad y_s(k)=2; \quad x_s(k+1)=0 \quad \text{and} \\ &y_s(k+1)=NS(y_s(k))=NS(2)=1 \\ &x_d(k)=PS(x_s(k))=PS(0)=1 \quad \text{and} \quad y_d(k)=2 \\ &x_d(k+1)=x_d(k)=1 \quad \text{and} \quad y_d(k+1)=y_d(k)=2 \\ &\therefore c_x(\mu)=\min(0+0+0+1,1)=1. \end{aligned}$$

c) *Left Top Corner, Dog Down Condition:*

$$\begin{aligned} &x_s(k)=0 \quad \text{and} \quad y_s(k)=2; \quad x_d(k)=0 \quad \text{and} \\ &y_d(k)=NS(2)=1 \quad x_s(k+1)=PS(x_s(k))=1 \quad \text{and} \\ &y_s(k+1)=y_s(k)=2 \quad x_d(k+1)=x_d(k)=0 \quad \text{and} \\ &y_d(k+1)=y_d(k)=1 \\ &\therefore c_x(\mu)=\min(0+0+1+0,1)=1. \end{aligned}$$

d) *Right Top Corner, Dog Left Condition:*

$$\begin{aligned} &x_s(k)=2 \quad \text{and} \quad y_s(k)=2; \\ &x_d(k)=NS(x_s(k))=1 \quad \text{and} \quad y_d(k)=2 \\ &x_s(k+1)=x_s(k)=2 \quad \text{and} \quad y_s(k+1)=NS(y_s(k))=1 \\ &x_d(k+1)=x_d(k)=1 \quad \text{and} \quad y_d(k+1)=y_d(k)=2 \\ &\therefore c_x(\mu)=\min(0+0+1+0,1)=1. \end{aligned}$$

e) *Right Top Corner, Dog Down Condition:*

$$\begin{aligned} &x_s(k)=2 \quad \text{and} \quad y_s(k)=2; \\ &x_d(k)=x_s(k)=2 \quad \text{and} \quad y_d(k)=NS(y_s(k))=1 \\ &x_s(k+1)=NS(x_s(k))=1 \quad \text{and} \quad y_s(k+1)=y_s(k)=1 \\ &x_d(k+1)=x_d(k)=2 \quad \text{and} \quad y_d(k+1)=y_d(k)=1 \\ &\therefore c_x(\mu)=\min(0+0+1+0,1)=1. \end{aligned}$$

f) *Left Bottom Corner, Dog Right Condition:*

$$\begin{aligned} &x_s(k)=0 \quad \text{and} \quad y_s(k)=0; \\ &x_d(k)=PS(x_s(k))=1 \quad \text{and} \quad y_d(k)=0 \\ &x_s(k+1)=x_s(k)=0 \quad \text{and} \quad y_s(k+1)=PS(y_s(k))=1 \\ &x_d(k+1)=x_d(k)=1 \quad \text{and} \quad y_d(k+1)=y_d(k)=0 \\ &\therefore c_x(\mu)=\min(0+0+0+1,1)=1. \end{aligned}$$

g) *Left Bottom Corner, Dog Up Condition:*

$$\begin{aligned} &x_s(k)=0 \quad \text{and} \quad y_s(k)=0; \\ &x_d(k)=x_s(k)=0 \quad \text{and} \quad y_d(k)=PS(y_s(k))=1 \\ &x_s(k+1)=PS(x_s(k))=1 \quad \text{and} \quad y_s(k+1)=y_s(k)=0 \\ &x_d(k+1)=x_d(k)=0 \quad \text{and} \quad y_d(k+1)=y_d(k)=1 \\ &\therefore c_x(\mu)=\min(0+0+1+0,1)=1. \end{aligned}$$

h) *Right Bottom Corner, Dog Left Condition:*

$$\begin{aligned} &x_s(k)=2 \quad \text{and} \quad y_s(k)=0; \\ &x_d(k)=NS(x_s(k))=1 \quad \text{and} \quad y_d(k)=0 \\ &x_s(k+1)=x_s(k)=2 \quad \text{and} \quad y_s(k+1)=PS(y_s(k))=1 \\ &x_d(k+1)=x_d(k)=1 \quad \text{and} \quad y_d(k+1)=y_d(k)=0 \\ &\therefore c_x(\mu)=\min(0+0+0+1,1)=1. \end{aligned}$$

i) *Right Bottom Corner, Dog Up Condition:*

$$\begin{aligned} &x_s(k)=2 \quad \text{and} \quad y_s(k)=0; \\ &x_d(k)=x_s(k)=2 \quad \text{and} \quad y_d(k)=PS(y_s(k))=1 \\ &x_s(k+1)=NS(x_s(k))=1 \quad \text{and} \quad y_s(k+1)=y_s(k)=0 \\ &x_d(k+1)=x_d(k)=2 \quad \text{and} \quad y_d(k+1)=y_d(k)=1 \\ &\therefore c_x(\mu)=\min(0+0+1+0,1)=1. \end{aligned}$$

1) Other Conditions:: For all the cases mentioned in rule 4(j), the sheep takes only one step at a time away from the dog which is not allowed to move since the sheep is not in an equilibrium state. Therefore, in all cases we have

$$\begin{aligned} &x_s(k+1)-x_s(k)=1 \quad \text{or} \quad y_s(k+1)-y_s(k)=0 \\ &\therefore c_x(\mu)=\min(0+0+1+0,1)=1 \\ &\text{or } x_s(k+1)-x_s(k)=0 \quad \text{or} \quad y_s(k+1)-y_s(k)=1 \\ &\therefore c_x(\mu)=\min(0+0+0+1,1)=1 \\ &\text{or } x_s(k+1)-x_s(k)=1 \quad \text{or} \quad y_s(k+1)-y_s(k)=1 \\ &\therefore c_x(\mu)=\min(0+0+1+1,1)=1. \end{aligned}$$

Theorem 4: The digraph of the dog–sheep problem is not strongly connected, but weakly connected.

Proof: It can be shown that starting from any allowable (all states except the ones with coincident positions for dog and sheep) state, one of the final equilibrium states can be reached. All the final equilibrium states have paths connecting them together. To see this, consider a final equilibrium state, and then move the dog back (to increase the distance between the dog and the sheep). This action will not result in any sheep movement. Then we can move the dog in positions that have a distance of more than one from the sheep (at the pen). Then the dog can be moved to a different position corresponding to another final equilibrium position. This shows that starting from any initial allowable state, there is a path to all the final equilibrium states. This proves that the underlying graph of the digraph is connected. To show that it is not strongly connected, consider any final equilibrium state. From these states, there is no dog action that can take the sheep from the boundary of the two-dimensional (2-D) space into the interior.

Some additional properties of the dog–sheep digraph are given as follows.

1) The number of nodes that are adjacent from a node representing an equilibrium state depends on the location of the dog position in the grid. There are the following three possibilities on the number of adjacent states.
   a) There are eight states adjacent from the equilibrium state node if the dog position is in the interior. Only seven out of the eight are allowed since dog and sheep are not allowed to have the same location.
   b) There are five states adjacent from the equilibrium state node if the dog position is in the side but not in a corner.
   c) There are three states adjacent from the equilibrium state node if the dog position is in the corner.
2) The number of nodes that are adjacent from a node representing a nonequilibrium state is one. The state adjacent from the nonequilibrium node can be another nonequilibrium node or an equilibrium node. An example is shown in Fig. 3.

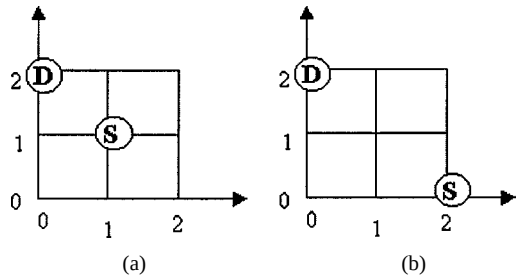

Fig. 3. Adjacent states for nonequilibrium initial state (a) before and (b) after.

IV. Dynamic Programming Solution to the 3 × 3 Grid Dog–Sheep Problem

The dynamic programming solution is based on Bellman's equation, which for our problem would look like the following:

$$V^*(\mathbf{x}(k)) = \min_{u \in \mu(\mathbf{x})} \{c_i(u) + V^*(\mathbf{x}(k+1))\}.$$

This equation indicates how the feedback controller can make decisions once the value function is available. This equation can also be used to find the value function using the boundary conditions from the problem.

We provide the solution to the problem using two different algorithms. Both algorithms are based on dynamic programming. The first algorithm uses Dijkstra's algorithm for each final equilibrium state and then uses minimization over all final equilibrium states to obtain the value function. The second algorithm directly uses dynamic programming to sequentially obtain the value function. The two algorithms are described after we define some terminology and some algorithms that will be used by the two main algorithms.

The following terminology is adapted from [7]. For the digraph $G = (V, E)$ weight function maps edges to weights as $w : E \to 1$. If a node v is adjacent from node u, we show that as $u \to v$. If there exists a path between a node u and node v possibly through other nodes, it is shown as $u \xrightarrow{p} v$. Weight of a path $p = (v_0, v_1, \cdots, v_k)$ is the sum of all the included edge weights, given as

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

If a path from u to v exists, then the shortest path weight from u to v is given by $\delta(u, v) = \min_p(w(p) : u \xrightarrow{p} v)$, else it is $\delta(u, v) = \infty$. Any path p from node u to node v in $G(V, E)$ with weight $w(p) = \delta(u, v)$ is the shortest path from u to v.

Algorithm 1: INITIALIZE (G, s): Given a source node $s \in V[G]$

$$\forall v \in V[G], \text{do } \{d[v] := \infty; d[s] := 0\}.$$

Here, := is the assignment operator. This algorithm initializes the estimates of the distances of the nodes to be infinity if the nodes are not the source node, and to zero if the node is the source.

Algorithm 2: RELAX(u, v): if $d[v] > d[u] + w(u, v)$ then $d[v] := d[u] + w(u, v)$. This algorithm takes all the immediate neighbors of node u and recalculates their estimated distances from the source node.

Algorithm 3: DIJKSTRA(G, s): Note: $S \subset V$, $S = \{v : d[v] = \delta(s, v)\}$.

```
INITIALIZE (G, s)
S := ∅
Q := V[G]
while Q ≠
  do u := EXTRACT-MIN(Q)
     S := S∪ {u}
     for each vertex v ∈ Adj[u]
        do RELAX(u, v)
```

In this algorithm, $u \in V - S$ is the vertex with the shortest path estimate in Q that contains all vertices in $V - S$ sorted by their d values. Hence EXTRACT-MIN(Q) extracts that node.

A. Algorithm Based on Dijkstra's Shortest Path Solution

Assume that the number of the final equilibrium states is m and the overall number of states is n. This technique is based on calculating the shortest path between any state $v_i \in V-S$ $\forall i=1,2,\cdots,n-m$, and all the final equilibrium states $s_j \in S$ $\forall j=1,2,\cdots,m$, then taking from the m calculated distances between v_i and all the final equilibrium states, the one with minimum weight. In other words, the Dijkstra algorithm is repeated m times, one for each final equilibrium state and the path with least weight is assigned to that state v_i. We will call S the set of final equilibrium states. The solution is obtained by using

$$d[v_j]=\min.\{d[v_j,s_i]\forall i=1,2,\cdots m\}.$$

Like Dijkstra's algorithm, this algorithm produces a set of vertices X whose final shortest path distances from the source set S is determined. That is, for all $v_I \in X$, we have $d[vj]=\delta(S,v_j)$. The algorithm repeatedly selects the vertex $v_i \in V-S$ with minimum shortest path estimate, inserts v_j into X and relaxes all the edges leaving v_j. At the end, we have a queue Q that contains all the vertices in $V-S$ with their corresponding distance value from the source set S.

Algorithm 4:

do $d[s_i] := 0\ \forall i=1,2,\cdots,m$;
for $i=1:m$
  do DIJKSTRA
for $i=1:n-m$ {
  $d_{min} := min.d[v_I,s_k]\ \forall k=1,2,\cdots m$
  d[v_i, s_k] := d_{min}}.

B. Direct Dynamic Programming Solution

Dijkstra algorithm solves the single source shortest paths problem on a weighted directed graph $G=(V,E)$ for nonnegative weights case. A modified version of this algorithm can be used to solve our problem after introducing the definition of distance of a vertex "state" from a set of vertices "states."

Definition 5: The distance of a vertex v_i from a set of vertices S is defined as

$$d[S,v_i]=\min.\{d[v_i,s_j]\forall v_i \in V-S, s_j \in S\}.$$

Like Dijkstra's algorithm, this algorithm produces a set of vertices X whose final shortest path distances from the source set S is determined. That is, for all $v_i \in X$, we have $d[v_j,S]=\delta(S,v_j)$. The algorithm repeatedly selects the vertex $v_I \in V-S$ with minimum shortest path estimate, inserts v_j into X and relaxes all the edges leaving v_j. At the end, we have a queue Q that contains all the vertices in $V-S$ with their corresponding distance value from the source set S.

Algorithm 5:

Initialize source set d[S] := 0;
Initialize the weights of the $V-S := \infty$.
do Q := $V-S$
while Q $\neq \emptyset$
  do u := EXTRACT-MIN(Q)
  S := S $\cup$ $\{u\}$
  for each vertex $v \in$ Adj[u]
    do RELAX (u,v).

V. Complexity Analysis

The following is the complexity analysis for both algorithms developed above.

A. Complexity Analysis for the Algorithm Based on Dijkstra's Shortest Path Solution

The following shows the complexity analysis for the algorithm (Algorithm 4):

do $d[s_i] := 0\ \forall i=1,2,\cdots,m$;
This pseudo-code has complexity of the order O(m)
for i = 1:m
  do DIJKSTRA

DIJKSTRA algorithm has complexity of the order O(n^2) and since this loop is executed m times, we get an order O($m \cdot n^2$) of complexity for this code.

for i = 1: n-m {
  $d_{min} := min \cdot d[v_i,s_k]\ \forall k=1,2,\cdots m$
  d[v_i, s_k] := d_{min}}
This loop has complexity of the order O($(n-m)\cdot m$).

Adding the complexity terms, we obtain the overall complexity as

$$\mathrm{O}(m)+\mathrm{O}(m\cdot n^2)+\mathrm{O}((n-m)\cdot m).$$

If we take m to be some fraction of n, so that $m=k\cdot n$, then the overall complexity of this algorithm becomes of the order O(n^3).

B. Complexity Analysis for Direct Dynamic Programming Solution

The following shows the complexity analysis for the algorithm (Algorithm 5):

Initialize source set d[S] := 0;
This code has complexity of order O(m).
Initialize the weights of the $V-S := \infty$.
This line has complexity of order O($n-m$).
do Q := $V-S$
while Q $\neq \emptyset$
  do u := EXTRACT-MIN(Q)
    S := S $\cup$ $\{u\}$
    for each vertex $v \in$ Adj[u]
      do RELAX(u,v)

These lines of code have complexity of the order O($(n-m)^2$), exploiting the fact that DIJKSTRA has complexity O(n^2).

Adding the complexity terms, we obtain the overall complexity as

$$\mathrm{O}(m)+\mathrm{O}(n-m)+\mathrm{O}((n-m)^2)$$

If we take m to be some fraction of n, so that $m=k\cdot n$, then the overall complexity of this algorithm becomes of the order

2	6	4
5	9	7
1	8	3

Fig. 4. Grid repartition.

O(n^2). Based on this analysis, this algorithm is superior to Algorithm 4.

VI. Simulation Software

We have developed a Multiple Document Interface (MDI) windows application using Visual Basic for performing experiments. This program allows us to run many simulations at the same time, in different modes. The three different modes are 1) automatic, 2) user-assisted, and 3) manual.

In the automatic mode, the simulation runs by itself once started. We just observe the behavior of the dog and the sheep. The simulation stops when the sheep has reached its final position. In the user-assisted mode, the user needs to click "*Next Button*" to make the sheep or the dog move one step. This option allows more time to the user to, for example, think about the problem between consecutive moves. Finally in the manual mode, the sheep still moves automatically according to the dynamics of the system whereas the user is controlling the dog movements using drag and drop. Presently, we can make the dog move in any direction one step at a time.

The software uses a "*Rules* table" where the dynamics of the sheep are expressed [see Fig. 5]. The default values are the ones used with the default dynamics of the system. This also implies that this table is useful when the distance between the dog and the sheep is equal to one on the x-axis or y-axis. If the sheep is "far" from the dog, it simply doesn't move. We can change the values in this table by hand and click "*Update Rules*" to actually make the simulation use the new dynamics defined. This "Rules table" can be used for any grid size. Indeed if we modify the sheep dynamics, the simulation will not show the optimal path unless we also change the V-matrix accordingly. The sheep dynamics description below uses the position numbers shown in Fig. 4. Based on these position numbers we can represent a matrix table 9×9 where the columns represent the dog positions and the rows represent the sheep positions (or vice-versa) and the values stored in the cells of this 9×9 table are the values of the dynamic program. This matrix is the above-mentioned V-matrix. The V-matrix is calculated by starting at the final states whose values are zero and then adding the other nodes following the algorithms developed in this paper.

We first identify in what position the sheep is in out of one of the corners, one of the edges, or in the center. Then we try to match "*Position of sheep—Position of dog*" with one row of the first two columns of the table. The columns 3 and 4 of the same row give the corresponding displacements of the sheep. For example, if the sheep is in the bottom-right corner (position 3) and the dog is in 8 (7th row of the table), then the next move of the sheep will be 0 on the x-axis and 1 on the y-axis.

The software produces a printable text history of the sheep and dog moves as shown in Fig. 6.

	xSheep-xDog	ySheep-yDog	ΔxSheep	ΔySheep
Bottom-Left Corner	-1	0	0	1
#1	0	-1	1	0
	-1	-1	0	0
Top-Left Corner	-1	0	0	-1
#2	0	1	1	0
	-1	1	0	0
Bottom-right Corner	1	0	0	1
#3	0	-1	-1	0
	1	-1	0	0
Top-Left Corner	1	0	0	-1
#4	0	1	-1	0
	1	1	0	0
Edge #5	0	1	0	1
	-1	1	0	1
	-1	0	0	0
	-1	-1	0	-1
	0	-1	0	-1
Edge #6	1	0	1	0
(5 rows)	...	...	...	...
Edge #7	0	1	0	1
(5 rows)	...	...	...	...
Edge #8	1	0	1	0
(5 rows)	...	...	...	...
Center	1	1	1	1
#9	1	0	1	0
(8 rows)	...	...	...	...

Fig. 5. Description of the sheep dynamics for the software.

```
Automatic Mode running ...

1) Sheep (2, 2) --> (2, 2)
2) Dog (1, 1) --> (1, 2)
3) Sheep (2, 2) --> (2, 1)
4) Dog (1, 2) --> (1, 2)
5) Sheep (2, 1) --> (2, 0)
6) Dog (1, 2) --> (2, 1)
7) Sheep (2, 0) --> (1, 0)
8) Dog (2, 1) --> (2, 1)
9) Sheep (1, 0) --> (0, 0)

Sheep has reached the home (0,0)
Simulation Ends ...
```

Fig. 6. Text description of a simulation run.

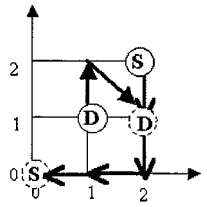

Fig. 7. Representation of a simulation run.

This simulation run can be graphically represented as shown in Fig. 7.

VII. Conclusion

In this paper we studied a class of pursuit evasion problems that is different than the traditional problems in that the aim of the pursuer is to force the evader into a pen. We gave two algorithms for optimal solution and showed one of them to be superior to the other in terms of complexity. We also presented a software package that has been developed to experiment with the problem.

References

[1] Y. Yavin and M. Pachter, *Pursuit-Evasion Differential Games*. New York: Pergamon, 1987.

[2] M. Isaacs and M. Rufus, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. New York: Dover, 1965.
[3] M. Basar, M. Tamer, M. Olsder, and G. Jan, *Dynamic Noncooperative Game Theory*, 2nd ed. New York: Academic, 1982.
[4] P. Bernhard, A. L. Colomb, and G. P. Papavassilopoloulos, "Rabbit and hunter game: Two discrete stochastic formulations," *Comput. Math. Applicat.*, vol. 13, no. 1–3, pp. 205–225, 1987.
[5] V. K. Balakrishnan, *Introductory Discrete Mathematics*. New York: Dover, 1991.
[6] A. Kaufmann, *Graphs, Dynamic Programming, and Finite Games*. New York: Academic, 1967.
[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1998.

Pushkin Kachroo (S'91–M'95–SM'00) was born in 1967. He received the B.Tech. degree from the Indian Institute of Technology, Bombay, India, in 1998, the M.S. degree from Rice University, Houston, TX, in 1990, and the Ph.D. degree from the University of California, Berkeley in 1993. He received the Professional Engineer license from the State of Ohio in 1995.

He is currently an Assistant Professor in the Bradley Department of Electrical and Computer Engineering at Virginia Tech. He was a Research Engineer in the Robotics R&D Laboratory of the Lincoln Electric Company from 1992 to 1994, after which he was a Research Scientist at the Center for Transportation Research at Virginia Tech for about three years. He has written two books, *Feedback Control Theory for Dynamic Traffic Assignment* (Berlin, Germany: Springer-Verlag, 1999) and *Incident Management in Intelligent Transportation Systems* (Norwell, MA: Artech House) three edited volumes and more than 50 research papers. His research interests include theory and applications of nonlinear and hybrid control systems. Applications of interest are in many diverse fields such as distributed processors, distributed robotics, communication networks, and transportation systems.

He has been the Chairman of ITS and Mobile Robotics sessions of the SPIE conference multiple times.

Samy A. Shedied (S'01) was born in Menofya, Egypt, in 1968. He received the B.Sc. degree in electrical engineering in 1990 and the M.Sc. degree in speech enhancement in 1993 from the MTC, Cairo, Egypt. Currently, he is pursuing the Ph.D. degree in electrical engineering at the Bradley Department of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg.

John S. Bay (S'85–M'88–SM'97) received the B.S.E.E. degree in electrical engineering from Virginia Polytechnic Institute and State University, Blacksburg, in 1984 and the M.S. and Ph.D. degrees in electrical engineering from The Ohio State University, Columbus, in 1985 and 1988, respectively.

From 1989 to 1999, he was with the Bradley Department of Electrical and Computer Engineering at Virginia Tech, where he taught control systems, robotics, and mechatronics, and was Director of the Robotics and Machine Intelligence Laboratories. In 2000, he moved to Raytheon Company, Falls Church, VA as an Engineering Fellow for the Raytheon Command, Control, Communications, and Information Systems (C3I) Strategic Systems Business Unit. There, he worked in military robotics and command and control (C2) systems, including human interfaces and decision support. More recently, in April 2001, he became a program manager for the Defense Advanced Research Projects Agency, Information Technology Office (DARPA/ITO), Arlington, VA.

He is author or co-author of over 50 technical publications in control systems, robotics, artificial intelligence, engineering education, and biomedical engineering.

Dr. Bay is a former IEEE Computer Society Distinguished Visitor, Associate Editor of IEEE CONTROL SYSTEMS MAGAZINE, and winner of the HKN C. Holmes MacDonald Award.

Hugh Vanlandingham (M'66–SM'80–LSM'01) received the Ph.D. degree from Cornell University, Ithaca, NY, in 1967.

Prior to his Ph.D. work, he was a Member of the Technical Staff at Bell Telephone Labs, Whippany, NJ (from 1957 to 1962). Since 1966, he has been with the Electrical and Computer Engineering Department at Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg.

His interests are in applications of soft computing, particularly to industrial applications.

He is active in the IEEE Systems, Man, and Cybernetics Society.