

Analysis of the Effect of Disinformation on Decision Model for Effective Detection and Usage of Disinformation in Adversarial Environment

Norman K. Ma and Michael S. Miller

Department of Computer Science
Texas A&M University
College Station , Texas

This work is being supported in part by the U. S. Air Force Office of Scientific Research under MURI grant #F49620-00-1-0326 and the Department of Computer Science, Texas A&M University

Overview

- Description of State Space
- Example
- OODA
- Situation Awareness
- Disinformation
- Discussion

Problem and Proposed Solution

Adversarial command and control problems can be summarized by interacting with the environment using repetition of observe, orient, decide, and act (ooda) steps.

Strategic analysis and decision making require a high-level, big-picture view. State space representation can provide an all encompassing representation which is conducive to proactive actions. Planning and information operations can also be described using state space representation.

Computing machinery problem solving techniques are conducive to state space representation.

State Space Representation

is a modeling technique that describe the situation at an instance of time. The set of possible situations are called States. The model also list possible actions from each of the states. Each action will transition the problem from one situation to another situation.

Three missionaries and three cannibals come to a river and find a boat that holds two. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten.

How shall they cross?*

* 2005/05/22, <http://www-formal.stanford.edu/jmc/elaboration/node2.html>

State Space Solution

Three missionaries and three cannibals come to a river and find a boat that holds two. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten.

How shall they cross?*

Let a state (mcb) be given by the number of missionaries, cannibals and boats on the initial bank of the river. The initial situation is represented by 331 and the goal situation by 000 .

A solution:

$331 \rightarrow 310 \rightarrow 321 \rightarrow 300 \rightarrow 311 \rightarrow 110 \rightarrow 221 \rightarrow 020 \rightarrow$
 $031 \rightarrow 010 \rightarrow 021 \rightarrow 000$

* 2005/05/22, <http://www-formal.stanford.edu/jmc/elaboration/node2.html>

Computing Machineries Generate Solution

```
(defun make-state (ml cl b mr cr) (list ml cl b mr cr)) (defun number-missionary-left (state) (nth 0 state)) (defun number-cannibal-left (state) (nth 1 state)) (defun position-boat (state) (nth 2 state)) (defun number-missionary-right (state) (nth 3 state)) (defun number-cannibal-right (state) (nth 4 state)) (defun missionary-takes-self (state) (cond ((and (= (position-boat state) 1) (> (number-missionary-left state) 0)) (safe (make-state (decrement (number-missionary-left state)) (number-cannibal-left state) (position-boat state) (number-missionary-right state) (number-cannibal-right state)))) ((and (= (position-boat state) 0) (> (number-missionary-right state) 0)) (safe (make-state (increment (number-missionary-right state)) (number-cannibal-right state) (position-boat state) (number-missionary-left state) (number-cannibal-left state)))) (t nil))) (defun missionary-takes-missionary (state) (cond ((and (= (position-boat state) 1) (> (number-missionary-left state) 2)) (safe (make-state (decrement2 (number-missionary-left state)) (number-cannibal-left state) (position-boat state)) (increment2 (number-missionary-right state)) (number-cannibal-right state)))) ((and (= (position-boat state) 0) (> (number-missionary-right state) 2)) (safe (make-state (increment2 (number-missionary-left state)) (number-cannibal-left state) (position-boat state) (decrement2 (number-missionary-right state)) (number-cannibal-right state)))) (t nil))) (defun cannibal-takes-self (state) (cond ((and (= (position-boat state) 1) (> (number-cannibal-left state) 0)) (safe (make-state (number-missionary-left state) (decrement (number-cannibal-left state)) (position-boat state) (number-missionary-right state) (increment (number-cannibal-right state)))) ((and (= (position-boat state) 0) (> (number-cannibal-right state) 0)) (safe (make-state (number-missionary-left state) (increment (number-cannibal-left state)) (position-boat state) (number-missionary-right state) (decrement (number-cannibal-right state)))) (t nil))) (defun cannibal-takes-cannibal (state) (cond ((and (= (position-boat state) 1) (> (number-cannibal-left state) 1)) (safe (make-state (number-missionary-left state) (decrement2 (number-cannibal-left state)) (position-boat state) (number-missionary-right state) (increment2 (number-cannibal-right state)))) ((and (= (position-boat state) 0) (> (number-cannibal-right state) 1)) (safe (make-state (number-missionary-left state) (increment2 (number-cannibal-left state)) (position-boat state) (number-missionary-right state) (decrement2 (number-cannibal-right state)))) (t nil))) (defun missionary-cannibal-together (state) (cond ((and (= (position-boat state) 1) (> (number-cannibal-left state) 0) (> (number-missionary-left state) 0)) (safe (make-state (decrement (number-missionary-left state)) (decrement (number-cannibal-left state)) (position-boat state) (increment (number-missionary-right state)) (increment (number-cannibal-right state)))) ((and (= (position-boat state) 0) (> (number-cannibal-right state) 0) (> (number-missionary-right state) 0)) (safe (make-state (increment (number-missionary-left state)) (increment (number-cannibal-left state)) (position-boat state) (decrement (number-missionary-right state)) (decrement (number-cannibal-right state)))) (t nil))) (defun opposite-boat (side) (cond ((= side 0) 1) ((= side 1) 0))) (defun safe (state) (cond ((and (> (number-cannibal-left state) (number-missionary-left state)) (not (= (number-missionary-left state) 0))) nil) ((and (> (number-cannibal-right state) (number-missionary-right state)) (not (= (number-missionary-right state) 0))) nil) (t state))) (defun path (state goal been-list) (cond ((null state) nil) ((equal state goal) (reverse (cons state been-list))) ((not (member state been-list :test #'equal)) (or (path (missionary-takes-self state) goal (cons state been-list)) (path (cannibal-takes-self state) goal (cons state been-list)) (path (missionary-takes-missionary state) goal (cons state been-list)) (path (cannibal-takes-cannibal state) goal (cons state been-list)) (path (missionary-cannibal-together state) goal (cons state been-list)))) (t nil))) (defun mission (state goal) (path state goal nil)) (defun decrement (number) (- number 1)) (defun increment (number) (+ number 1)) (defun decrement2 (number) (- number 2)) (defun increment2 (number) (+ number 2))
```

Let a state (mcb) be given by the numbers of missionaries, cannibals and boats on the initial bank of the river. The initial situation is represented by 331 and the goal situation by 000

331 -> 310 -> 321 -> 300 -> 311 -> 110 -> 221 -> 020 -> 031 -> 010 -> 021 -> 000

* <http://www.comp.nus.edu.sg/~shantanu/mission.txt>

State Space Description

$$G=(S,A)$$

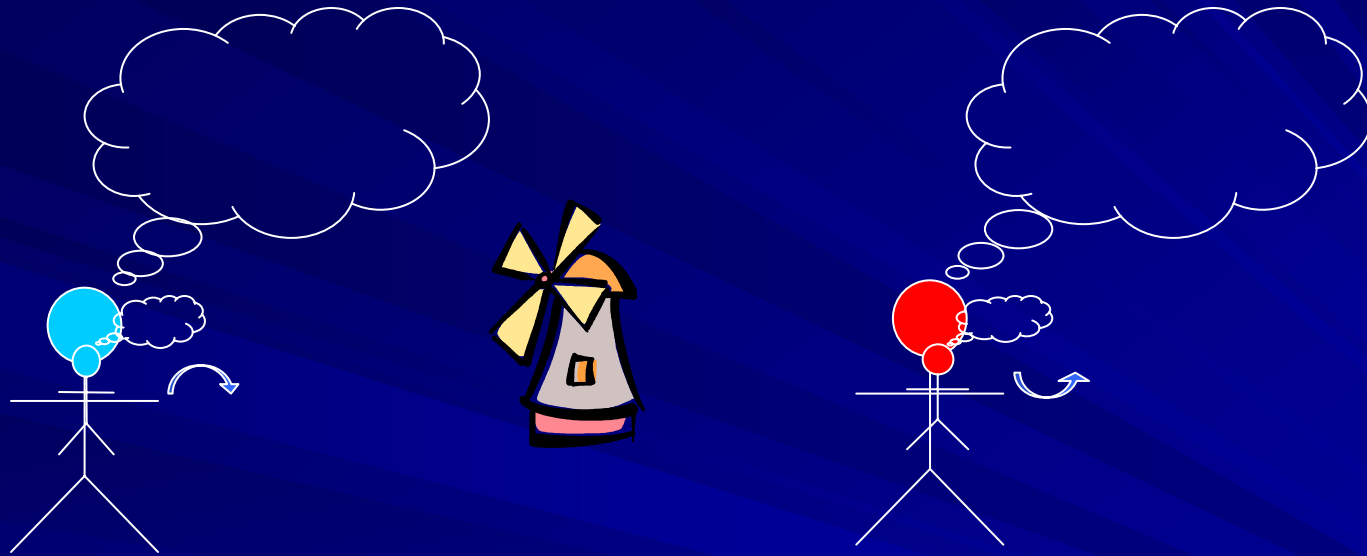
Where:

S is the set of states: $\{s_1, s_2, \dots, s_n\}$

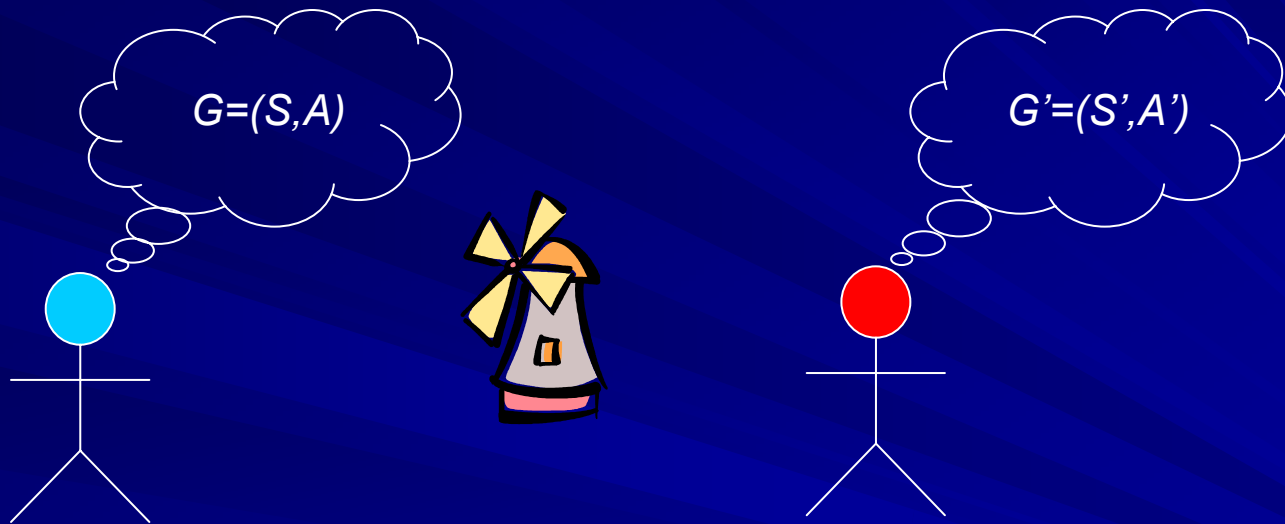
A is the set of actions: (a_i, s_j, s_k, pr, t)
In state s_j , action a_i will result in s_k with probability pr within time t .

s is a state: (p_a, p_b, \dots, p_m)
 p is a parameter of a state

Modeling of Physical, Information, and Cognitive domains



Modeling of Physical, Information, and Cognitive domains using State Space Representation



Anchor OODA concept with state space model

```
1 // Red team algorithm, disinformation distributor
2 // Input: Environmental observation
3 // Output: Recommended action
4 // Use state space representation
5 Loop
6   If goal state reached or continuation condition not met
7     Exit
8   Observe:
9     Identify the present state using situation assessment
10    If present state equals to a goal state then
11      Housekeeping
12    Repeat Loop
13  Orient:
14    If opponent changed plan
15      new_plan ← Identify opponent's plan
16      opp_plan ← old_plan + new_plan
17    If own plan not viable
18      own_plan ← re-plan(own_plan, opp_plan)
19  Decide:
20  Action: action ← Select next action(own_plan, opp_plan)
21
22    Recommend action(action)
23    wait(quiescence period)
24    Housekeeping
25 Repeat Loop
```

Situation Assessment Algorithm (identify the present state)

```
1 // Situation Assessment algorithm in state space
2 state_previous <- state_now
3 candidate_states <- neighbor(1, state_previous)
4   and state_now
5 for all candidate_states
6   if state parameters equal new parameters
7     state_now <- state
8     return state_now
9 // The new state is drastically different than
10 // previous state.
11 candidate_states <- states_of(own plan)
12 for all candidate_states
13   Match state to parameters
14   return state_now
15 else
16   add neighbor(1, state) to candidate_states
17 candidate_states <- states_of(opponent's plan)
```

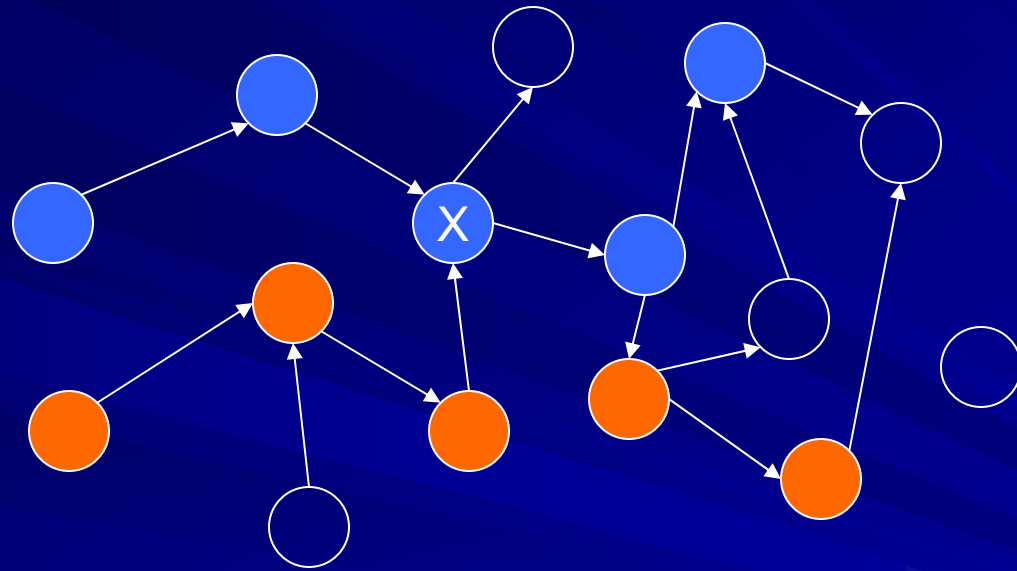
Situation Assessment Algorithm

```
18 for all candidate_states
19     Match state to parameters
20     return state_now
21     else
22         add neighbor(1, state) to candidate_states
23 Loop
24     if goal state reached
25         Exit
26     Use heuristics to guess a state
27     candidate_states <- neighbor(1, state) and state
28     for all candidate_states
29         match state to parameters
30     return state_now
31 repeat loop
```

State space representation

- Axiom 1, the final state of an adversarial interaction is recognizable by both teams and contains no emittable actions by either of the teams.
- Axiom 2: Information Operations is a sub-process of the Decision-making process.
- Axiom 3: Decision-making processes are represented through physical, informational, and cognitive objects.
- Axiom 4: Information operations (IO) representation contains physical, informational, and cognitive objects.
- Axiom 5: Informational and cognitive objects may represent physical objects.
- Axiom 6: Physical objects can be represented by a set of parameters.
- Axiom 7: Information Operation objects can be represented by a set of parameters.
- Axiom 8: In adversarial interaction with a dis-informative opponent, the perceived plan will change.
- Axiom 9: Disinformation can be detected and will cause opponent model to be rebuilt.

Visualization



Disinformation detection and generation

```
1 // Completed the observe process and knows the present state
2 if state_now is not equal next_state(p_opp)
3     p_opp <- assess_opponent_plan(plan history)
4     s_opp_goal <- find_opponent_goal(plan history)
5 if state_now is not equal next_state(p_own)
6     p_own <- re-plan()
7 if want to disinform
8     p_extern <- generate_external_plan(p_own)
9 else
10    p_extern <- p_own
```

Summary

- State space representation is usable for reasoning about actions
- Stochastic Models is another possible representation
- Established computing algorithms, machineries, and theoretical work available for state space representation (SSR)
- Situation assessment, planning, ooda process is representable using SSR
- Disinformation operations can be described using SSR

Discussion