Game theory - lab 2

David Milec

Czech Technical University in Prague

milecdav@fel.cvut.cz

December 13, 2021

Overview

Patrolling Security Game

- Two-player multi-stage game. Imperfect information, infinite horizon and simultaneous moves.
- Defender moves in a graph by $move(i)$ action.
- Attacker can *wait* or attack a target by $enter(t)$.
- $enter(t)$ blocks attacker for $\tau(t)$ moves.
- If defender visits t in that time he captures him and game ends by a capture. If not, the game ends by successful attack. The game can also end with no attack.

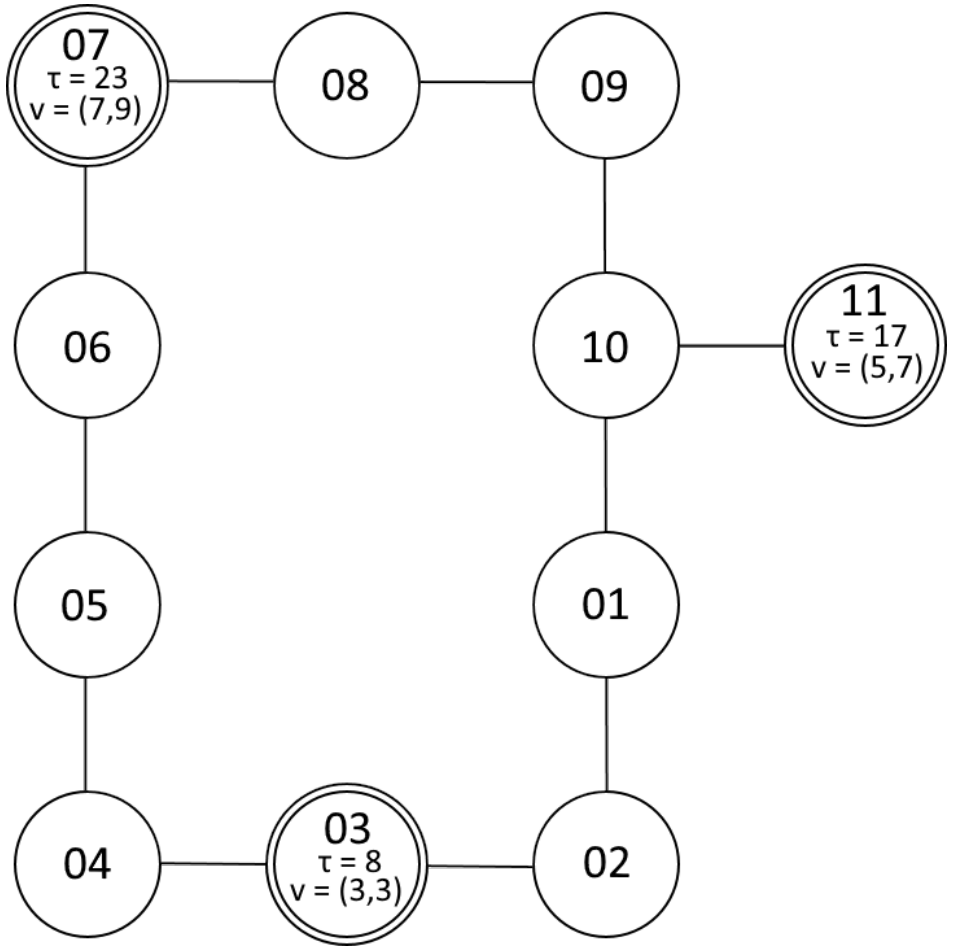
Patrolling Security Game

Possible outcomes:

- **Capture**: attacker receives penalty $-\epsilon$ and defender keeps all utilities for targets $\rightarrow \quad \sum_{i \in T} V_d(i)$
- **No attack**: attacker gets 0 and defender keeps all utilities for targets $\rightarrow \quad \sum_{i \in T} V_d(i)$
- **Successful attack at** t: attacker gets $V_a(t)$ and defender loses utility at $t \quad \rightarrow \quad \sum_{i \in T \setminus t} V_d(i)$

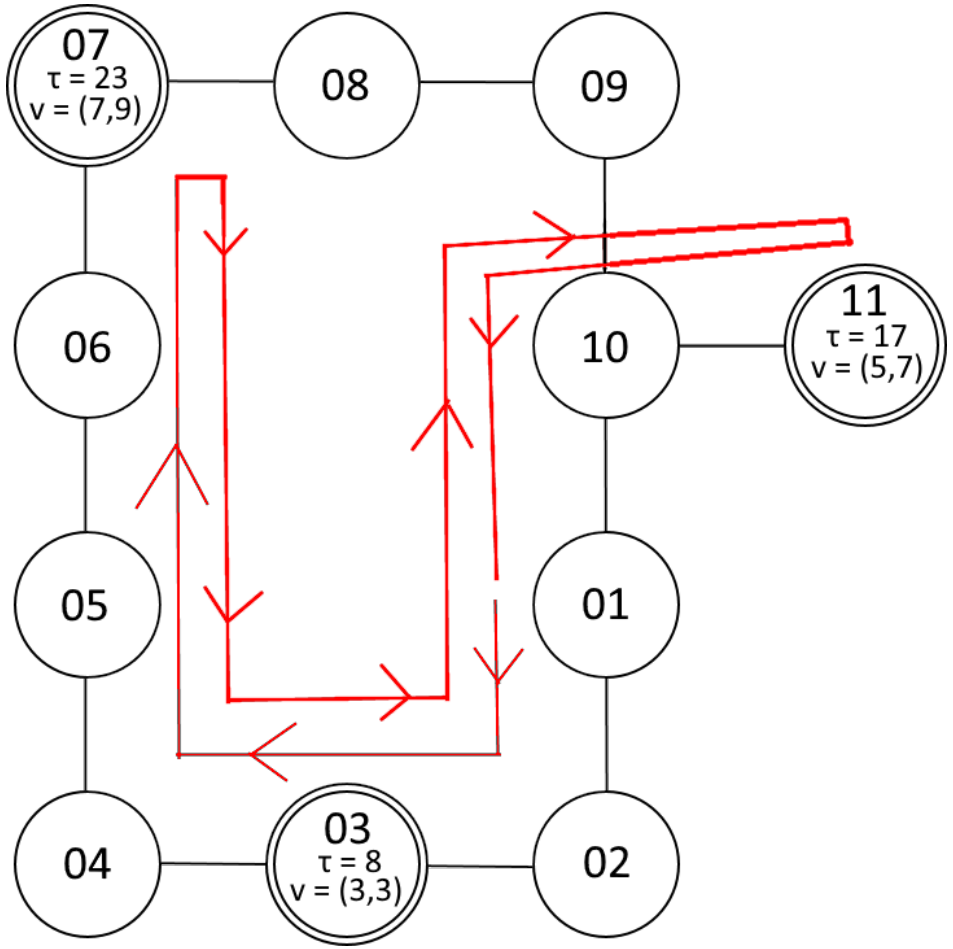
Patrolling Strategy

- Defender's patrolling strategy can be deterministic or probabilistic.
- Deterministic patrolling strategy is described by fixed sequence of nodes that the defender repeatedly visits.
- What is the optimal patrolling sequence for the example if the defender starts at 01?



Patrolling Strategy

- One optimal sequence is (01, 02, 03, 04, 05, 06, 07, 06, 05, 04, 03, 02, 01, 10, 11, 10, 01)
- We can see that the defender periodically visits every target in less than $\tau(t)$ steps.
- How to encode the correctness to constraints?



Confirming Deterministic Strategy

Let σ be the sequence. $O_i(j)$ gives index to the sequence when target i was visited for the jth time and o_i is number of visits of target i in the sequence.

$$\sigma(1) = \sigma(s) \tag{1}$$
$$o_i \geq 1 \quad \forall i \in T \tag{2}$$
$$adjacent(\sigma(j-1), \sigma(j)) = 1 \quad \forall j \in \{2, 3, \cdots, s\} \tag{3}$$
$$O_i(k+1) - O_i(k) - 1 \leq \tau(i) \quad \forall i \in T, \forall k \in \{1, 2, \cdots, o_i - 1\} \tag{4}$$
$$O_i(1) + s - O_i(o_i) - 2 \leq \tau(i) \quad \forall i \in T \tag{5}$$
$$\tag{6}$$

Check if it works for our sequence (01, 02, 03, 04, 05, 06, 07, 06, 05, 04, 03, 02, 01, 10, 11, 10, 01). (targets are 03, 07, 11 with $\tau(03) = 8$, $\tau(07) = 23$, $\tau(11) = 17$)

Checking the Sequence

Equation one is easy to check and we start and end by 01. o_i of all the targets is at least 1 and we know the vertices are adjacent.

i	$O_i(1)$	$O_i(2)$	$O_i(3)$	o_i
01	1	13	17	3
02	2	12		2
03	3	11		2
04	4	10		2
05	5	9		2
06	6	8		2
07	7			1
08				0
09				0
10	14	16		2
11	15			1

Equation 4:
Target 03:
11 - 3 - 1 = 7 $\leq$ 8 ✓

Equation 5:
Target 03:
3 + 17 - 11 - 2 = 7 $\leq$ 8 ✓
Target 07:
7 + 17 - 7 - 2 = 15 $\leq$ 23 ✓
Target 11:
15 + 17 - 15 - 2 = 15 $\leq$ 17 ✓

Computing the Strategy

- To construct the solution from scratch, we would need to solve the equations for all possible sequence lengths and o_i, which is infeasible even for small problems.
- We need to do search with backtracking guided by heuristic. For every node in T or until we find a solution create a one size sequence from it, set $j = 2$ and execute following function RECURSIVECALL(σ, j):

if $\sigma(i) = \sigma(j-1)$ and Equation 2 holds **then**
  if Equation 5 holds **then**
    return σ
  else
    return FAILURE
  end if
else
  F_j = FWD(σ,j)
  for all the i in F_j **do**
    $\sigma(j) = i$
    σ' = RECURSIVECALL(σ, j+1)
    if σ' not FAILURE **then**
      return σ'
    end if
  end for
  return FAILURE
end if

Computing the Strategy

- First *if* makes sure Equations 1, 2 hold and next *if* checks Eq. 5 and returns solution, otherwise we backtrack.
- We get to the *else* of the first *if* when the sequence is not a cycle or there are some unvisited targets. We call FWD which returns possible next nodes (making sure Equations 3 and 4 hold). Then we try the assignment of the variable and continue with the recursion.

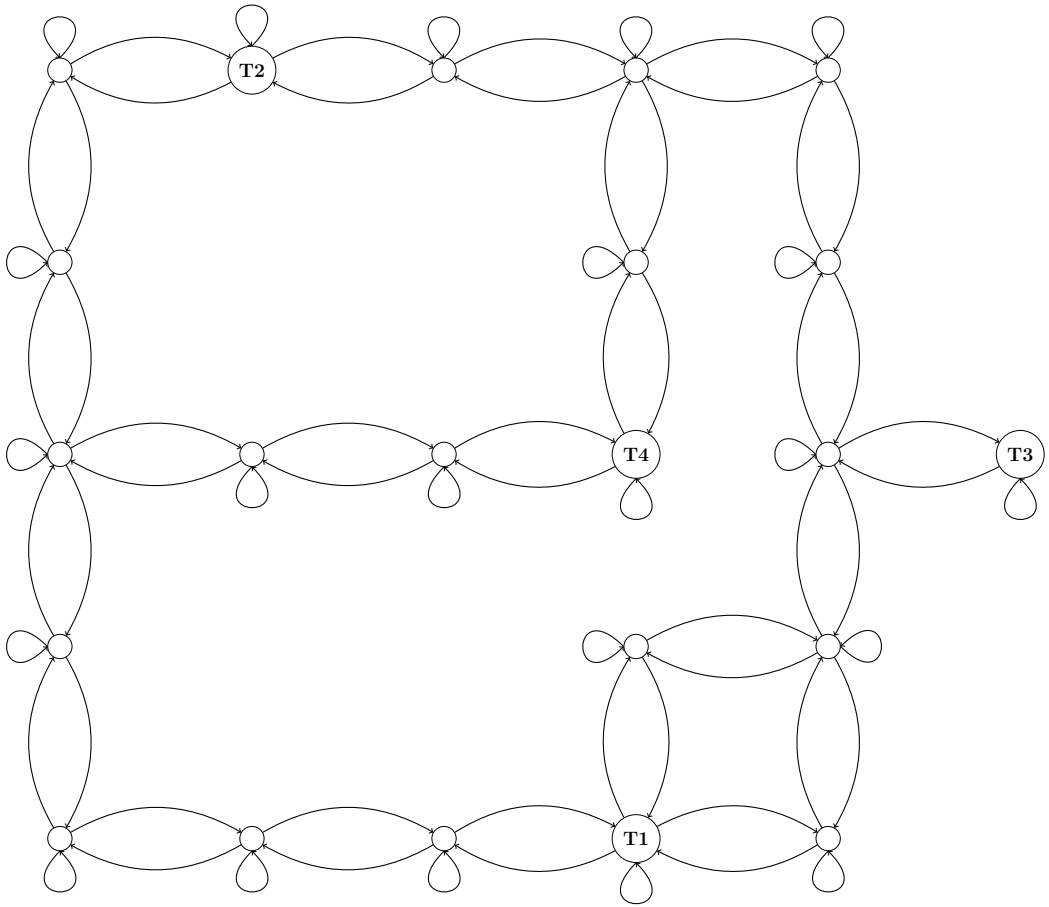
```
if σ(i) = σ(j − 1) and Equation 2 holds then
    if Equation 5 holds then
        return σ
    else
        return FAILURE
    end if
else
    F_j = FWD(σ,j)
    for all the i in F_j do
        σ(j) = i
        σ' = RECURSIVECALL(σ, j+1)
        if σ' not FAILURE then
            return σ'
        end if
    end for
    return FAILURE
end if
```

[1]

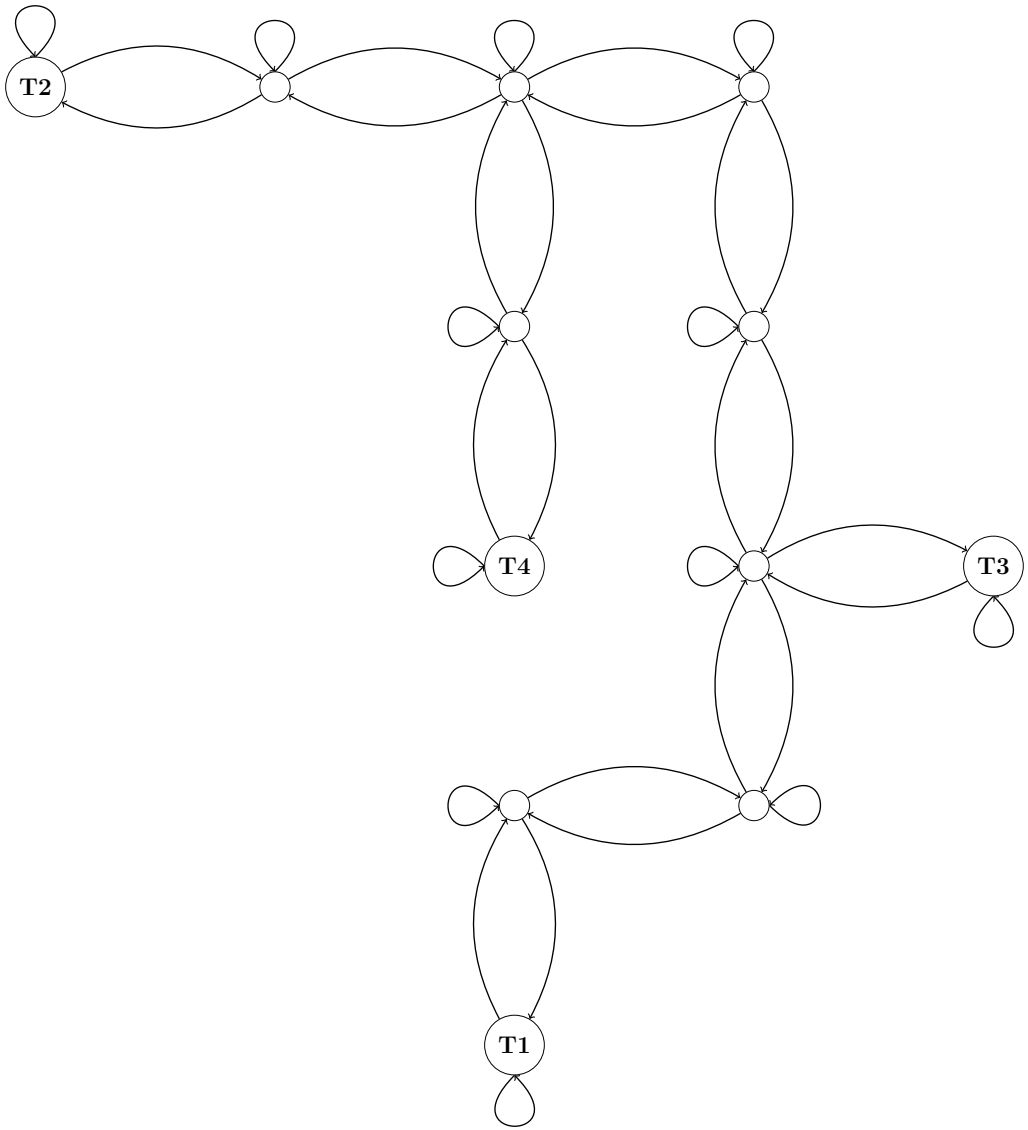
[1] Details https://core.ac.uk/download/pdf/55221143.pdf

Reducing the size of the patrolling graph

- We can reduce the computation problem by first thinking about what we need for optimal strategy and if we can remove some nodes.
- Generally, which nodes can we remove?
- Given following patrolling problem, remove unnecessary nodes and edges from the graph.

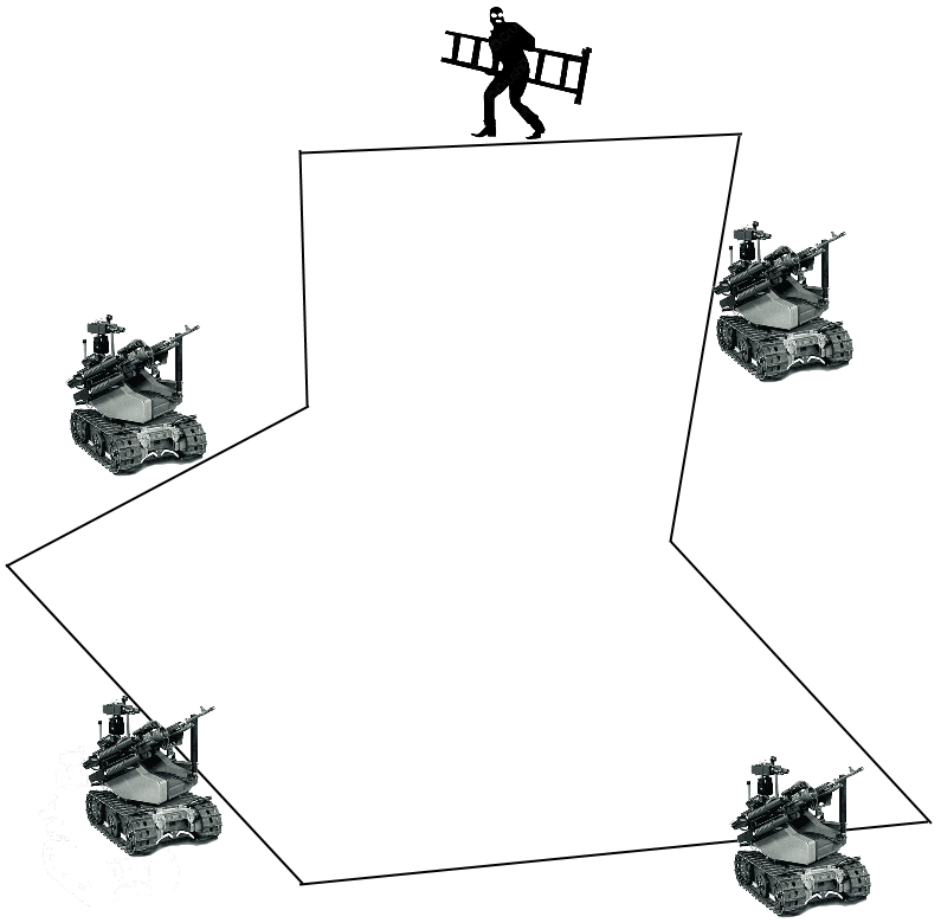


Example of reduced graph



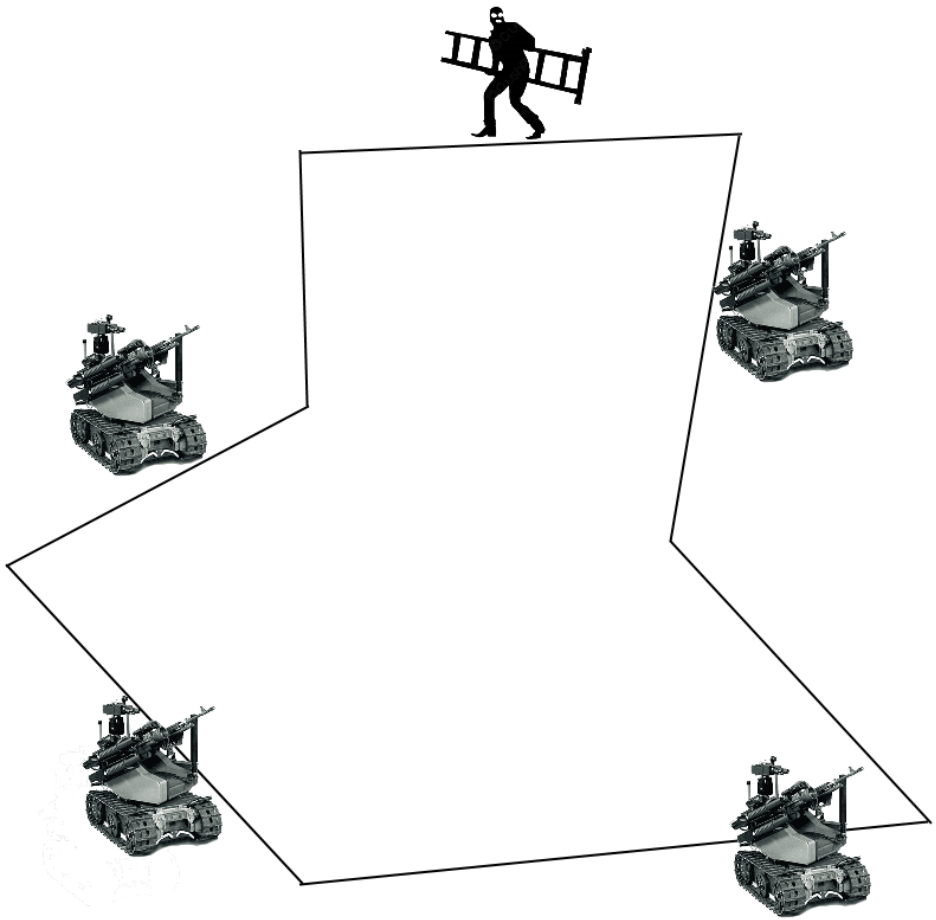
Patrolling around Polygon Description

- Polygon with some fence and we have n robots to guard it
- Robots go around the fence and attacker picks a spot on the fence and attacks, going through the fence takes him some time t



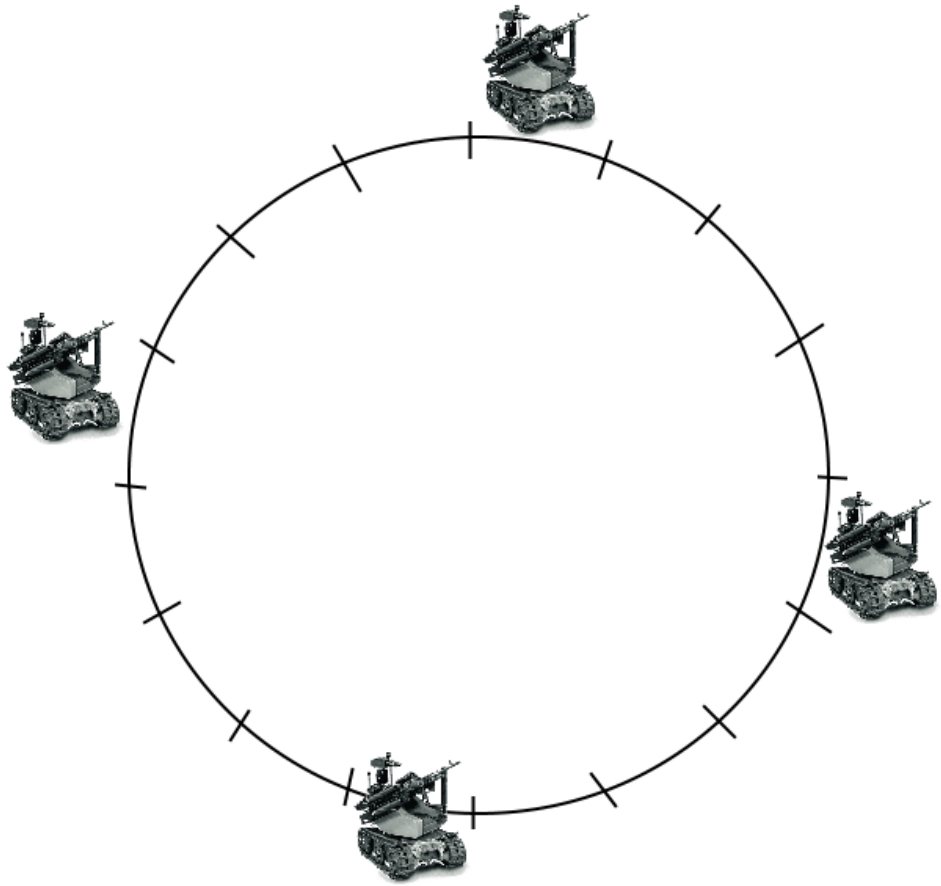
Patrolling around Polygon Observations

- We want to maximize probability we catch the adversary if he attacks on the worst possible point for us. (All the points on the fence are the same so the adversary has no incentive to go for specific point)
- The probability is maximized if the robots are distributed uniformly along the fence, they all face the same direction and move at the same speed.



Patrolling around Polygon Approach

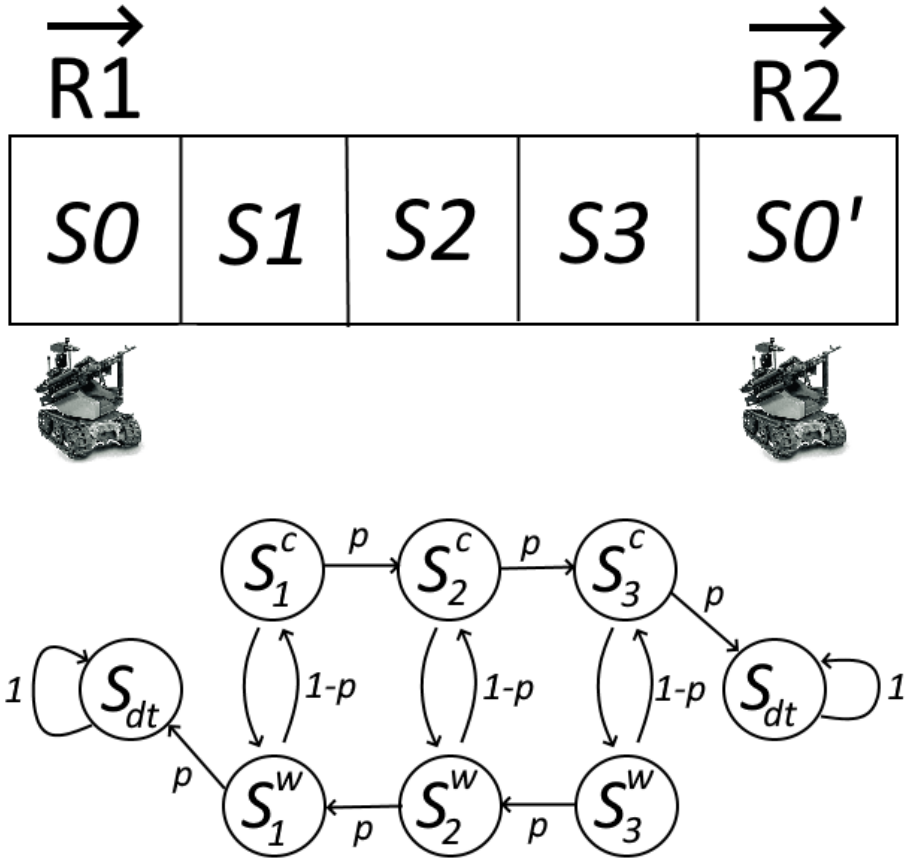
- We want the robots to go as fast as they can to be the most effective. Instead of maintaining the same speed, we measure how long the robot takes to go around and split the polygon to uniform segments based on time spent in each segment.
- We place the robots uniformly based on the segments.



Patrolling around Polygon Cases

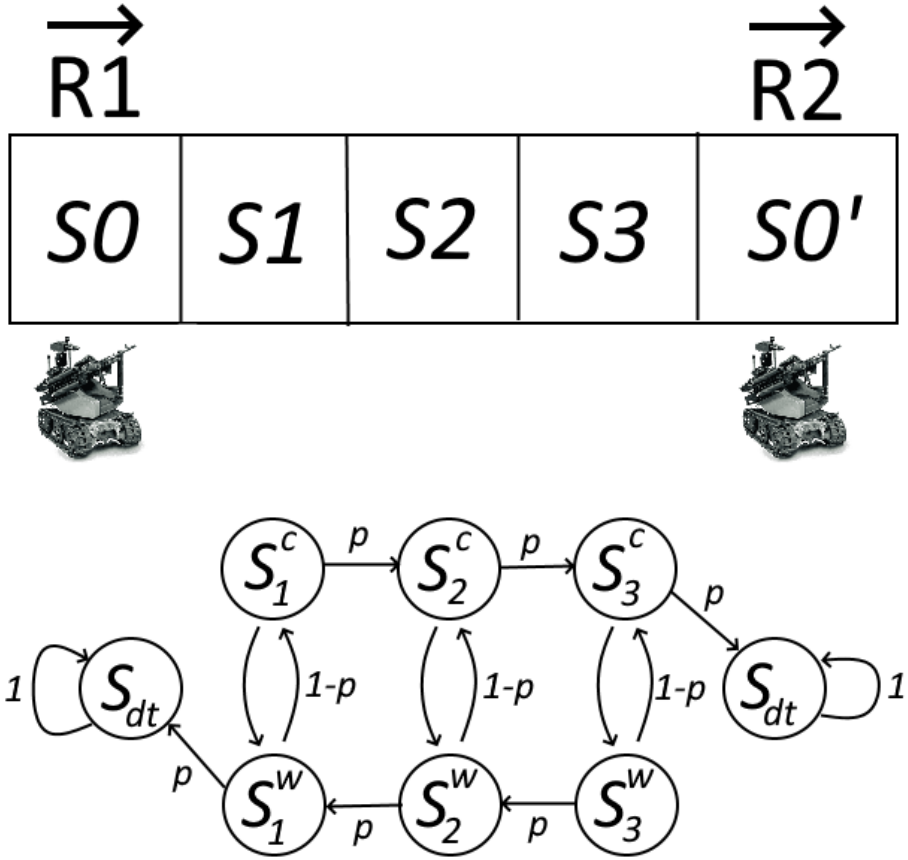
We look at segments between two robots. Based on the time t required to breach the fence we can have three cases:

- We can always catch every intruder if we simply circle around.
- We can never catch smart intruder.
- We have non-zero probability to catch smart intruder when we continue with probability p and turn around with probability $1 - p$. Turning takes time τ.



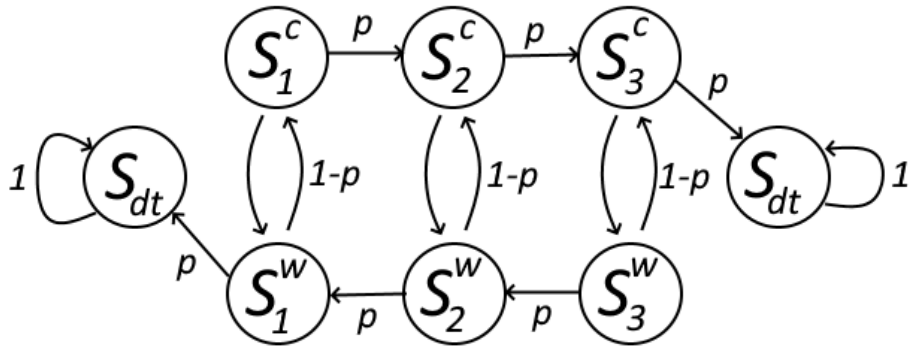
Patrolling around Polygon Converting to Graphical Model

- We want to compute probabilities that robots reach each segment in at most t steps.
- Reaching S_i is equivalent to starting in S_i^w and reaching S_{dt}.
- We want the probability of reaching S_{dt} from each S_i^w.



Patrolling around Polygon Computing p

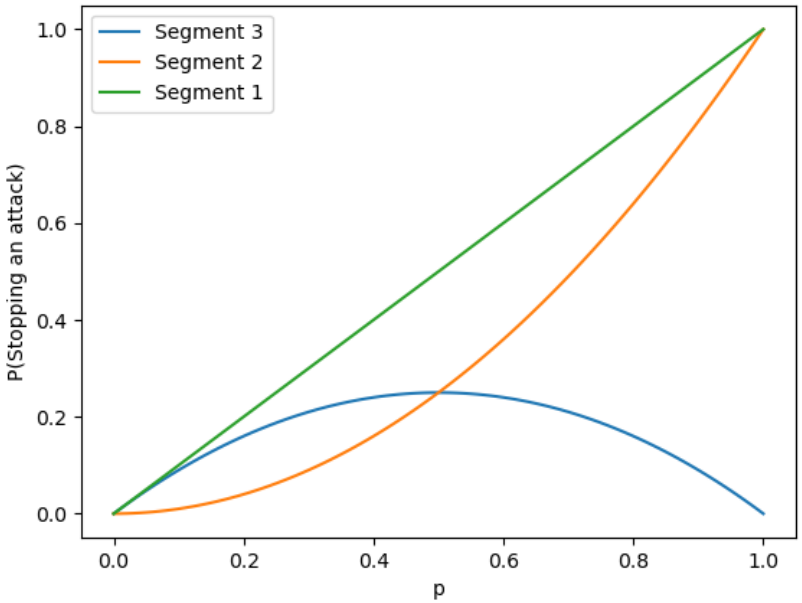
- Now we construct a transition matrix M from the graphical model.
- We do M^t to get transition probabilities after t steps.
- And we check the probabilities of transitions from each S_i^w to S_{dt}.

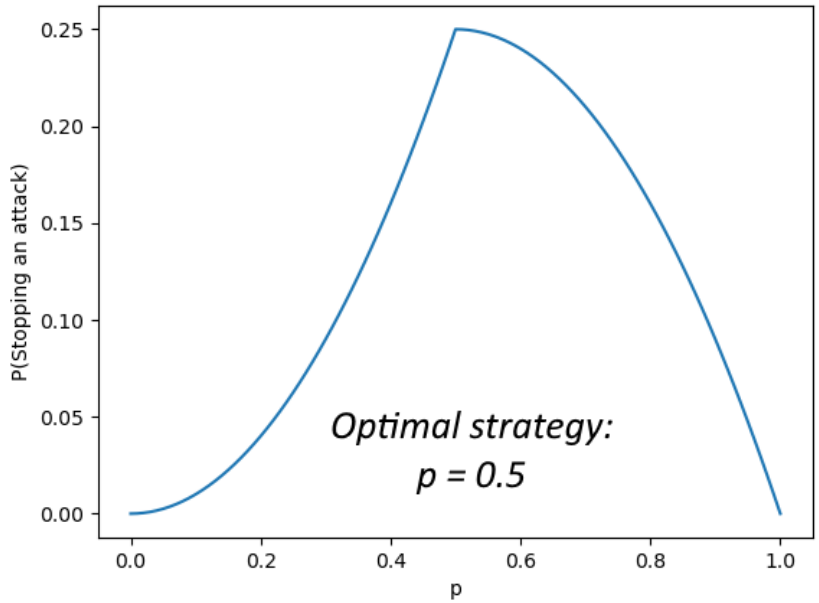


$$
\begin{array}{c}
 & \begin{array}{ccccccc} S_1^c & S_1^w & S_2^c & S_2^w & S_3^c & S_3^w & S_{dt} \end{array} \\
\begin{array}{c} S_1^c \\ S_1^w \\ S_2^c \\ S_2^w \\ S_3^c \\ S_3^w \\ S_{dt} \end{array} &
\begin{bmatrix}
0 & 1-p & p & 0 & 0 & 0 & 0 \\
1-p & 0 & 0 & 0 & 0 & 0 & p \\
0 & 0 & 0 & 1-p & p & 0 & 0 \\
0 & p & 1-p & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1-p & p \\
0 & 0 & 0 & p & 1-p & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\end{array}
$$

Patrolling around Polygon Computing p

Final step is to compute the probabilities of stopping an attack for selected values of p. Since we are defending against rational opponent we minimize over the segments for each p. And select p that maximizes the minimum.





The End