Pursuit-Evasion Strategies by Model Checking

Hongyang Qu

University of Sheffield

1 December 2015

Outline

- Introduction to pursuit-evasion problem
- Compute clearing strategies by model checking
- Find optimal execution of cleaning strategies
- Conclusions

Outline

- Introduction to pursuit-evasion problem
- Compute clearing strategies by model checking
- Find optimal execution of cleaning strategies
- Conclusions

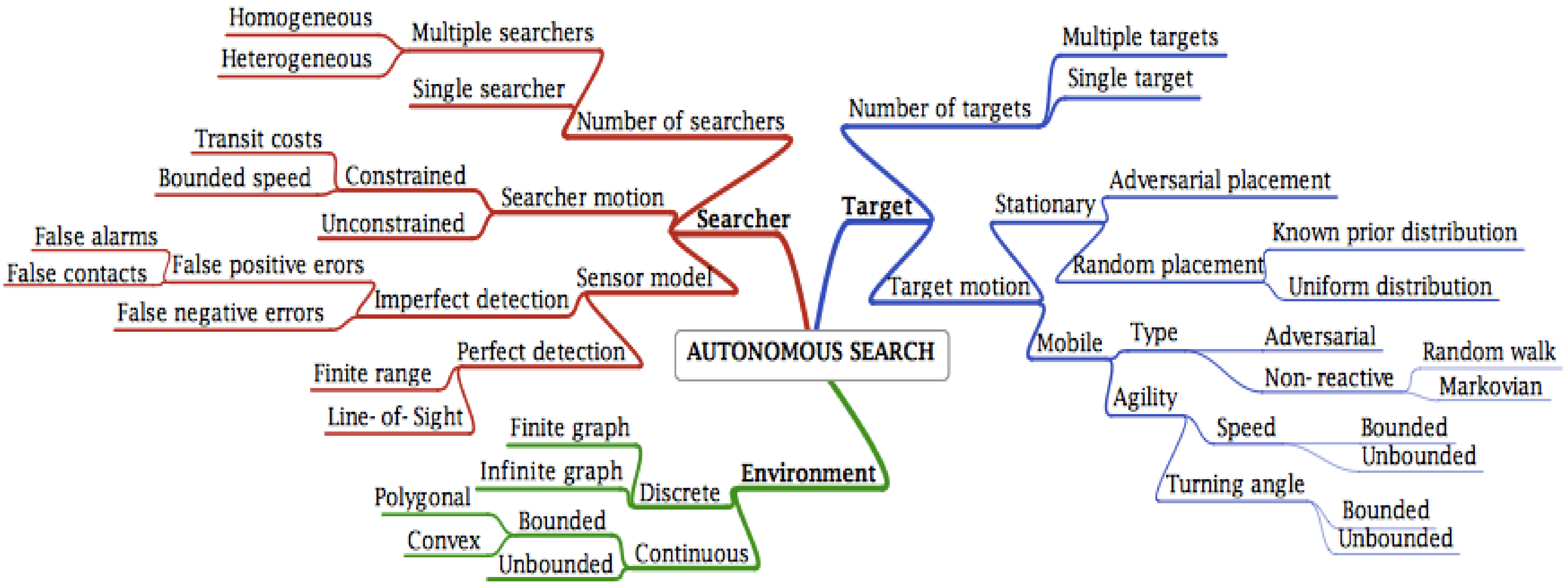
What is Pursuit-Evasion problem?

- It studies how to search for a smart, fast and evading target in an area
- Not only interesting to military, police or border patrol!
- The problem of closing a museum for the night with many rooms and few cameras – human guards need a P/E strategy - also by robots !
- Finding confused elderly people who wander off
- Capturing fleeing animals,
- Locating lost team members of first response teams or survivors in disaster scenarios,
- Finding people in cave systems, etc.

Assumptions for “good” theory

- Planar problems: sensor ranges, velocities of robots and evader, shapes of the environment, visibility conditions
- Buildings: layout known, connectivity known
- Natural environments: map is known
- Worst case assumptions about evaders: no knowledge about their numbers, no limits to their speed !
- Strategies for search in unknown terrains is largely unexplored area of research, i.e. under SLAM

A P/E Problems Map (Chung (2011))



Our Pursuit-Evasion problem

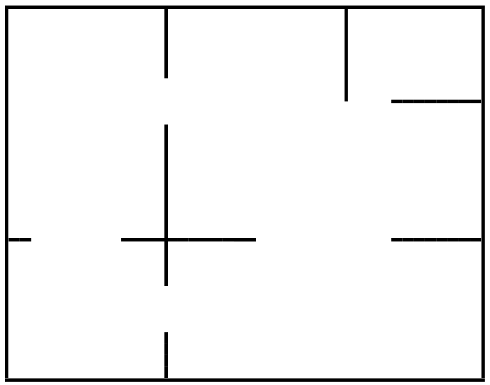
- Search for an omniscient and smart target that moves at unbounded speed (conservative assumption)
- Formal concept of “contamination” is used
- Searchers can execute actions of clearing and blocking
- A graph based model is used to abstract the environmental model into a graph of locations (vertices) and passages (edges).

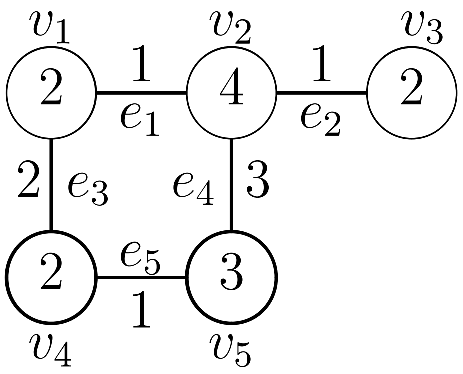
Our Objectives

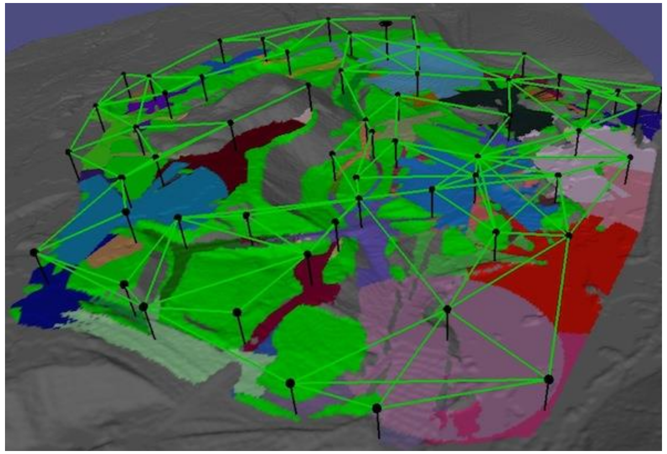
- Search time and cost optimization for autonomous robot teams in the graph clear (GC) model
- Solution: Application of model checking and LP to solve and optimize robotic search algorithms
- Modelling of different pursuit-evasion problems
- Automated generation of new search strategies from a temporal logic formula in MCMAS + application of an LP solver

Abstraction of the environment into a graph

In a building:







on a natural terrain

Graph states

- Vertices are either clear (R) or contaminated (C)
- Edges are clear (R), contaminated (C) or blocked (B)
- The state-space of surveillance graphs is

$$\nu \in \mathcal{V}(G) = \{\mathcal{R}, \mathcal{C}\}^n \times \{\mathcal{R}, \mathcal{C}, \mathcal{B}\}^m$$

where ν is a state (n=n.o.vertices, m=n.o.edges)

Clearing actions and costs

- A searcher can sweep a vertex (location)
- A searcher can block an edge (a passage)
- For n vertices and m edges the searcher action can be represented by

$$a = \{a_1, \ldots, a_{n+m}\} \in \{0,1\}^{n+m} = \mathcal{A}(G)$$

- The cost of an action is defined by

$$c(a) = \sum_{i=1}^{n} a_i w(v_i) + \sum_{j=1}^{m} a_{n+j} w(e_j)$$

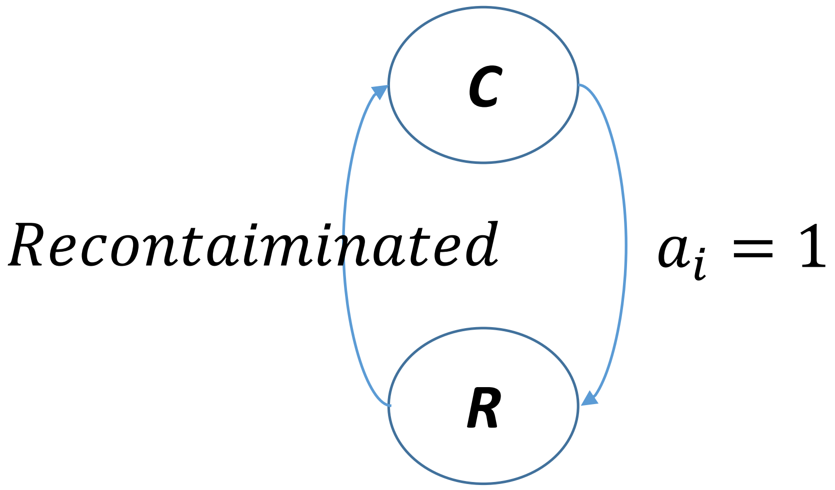
Actions rules for “intruders”

- If there is a non-blocked contamination path to a vertex from a contaminated edge or vertex then that vertex becomes automatically contaminated

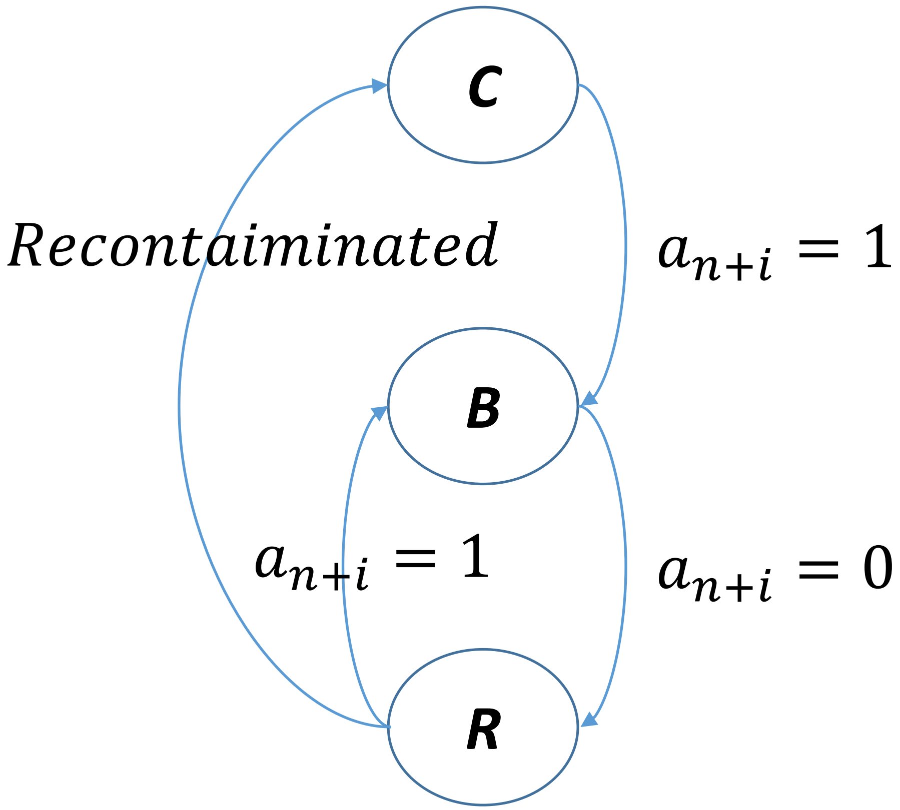
- If there is a non-blocked contamination path to an edge from a contaminated edge or vertex then that edge becomes contaminated

State changes of the surveillance graph

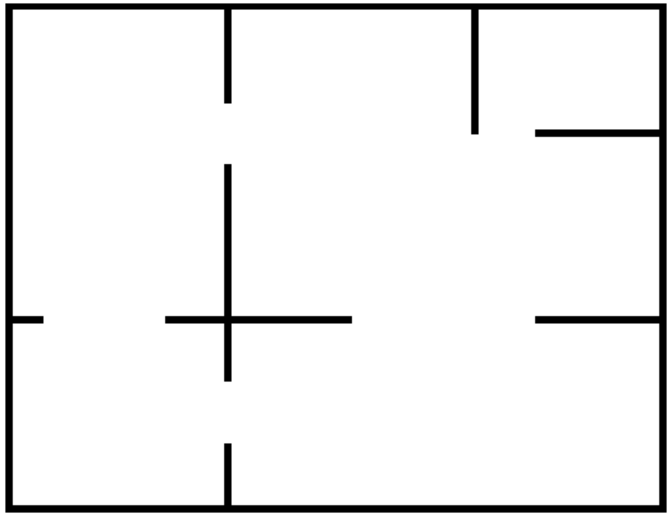
- Vertex i

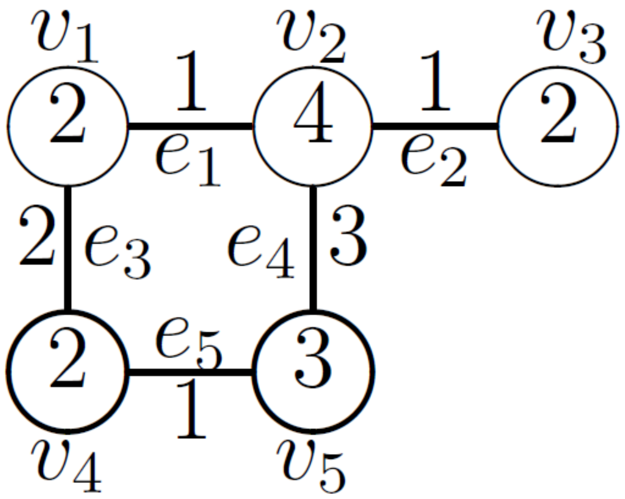


- Nodes j



Examples of cleaning strategies





$\nu(G)$	a	$c(a)$
$\mathcal{CCCCC}\ \mathcal{CCCCC}$	10000 10100	5
$\mathcal{RCCCC}\ \mathcal{BCBCC}$	00010 10101	6
$\mathcal{RCCRC}\ \mathcal{BCBCB}$	01100 11011	12
$\mathcal{RRRRC}\ \mathcal{BBRBB}$	00001 00011	7
$\mathcal{RRRRR}\ \mathcal{RRRBB}$	00000 00000	0
$\mathcal{RRRRR}\ \mathcal{RRRRR}$		

Outline

- Introduction to pursuit-evasion problem
- Compute clearing strategies by model checking
- Find optimal execution of cleaning strategies
- Conclusions

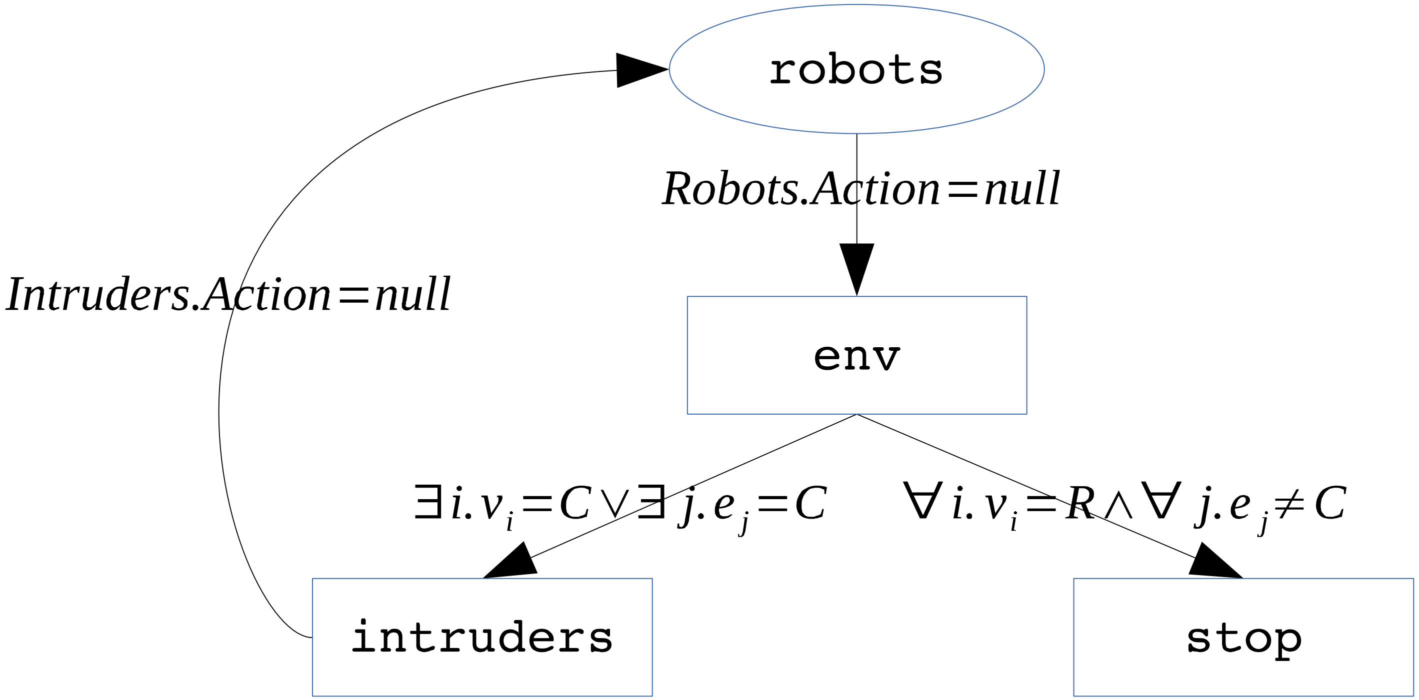
Objectives

- Application of model checking to robotic search algorithms
- Modelling of different pursuit-evasion problems
- Rigorous comparison between problem formulations
- Automated generation of new search strategies from a temporal logic formula

Modelling state transitions in SGs

- Three agents will be used to model the graph and its transitions: **Environment**, **Robots**, **Intruders**.
- **Environment** agent :
- Variables $v_i \in \{R, C\}$, $e_i \in \{R, C, B\}$, $nv_i \in \{1,0\}$ for sweeping, $ne_i \in \{1,0\}$ for blocking action
- Variable **turn** $\in \{robots, intruders, env, stop\}$ is used to schedule the turn of the agents and the environment itself for changes of variables .

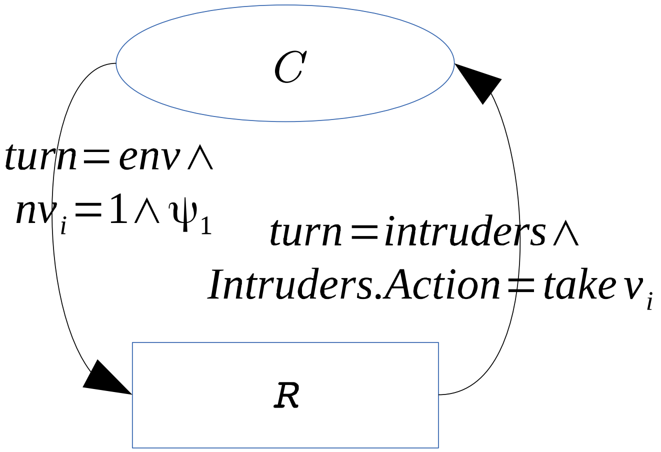
Environment actions and protocols for the variable **turn**

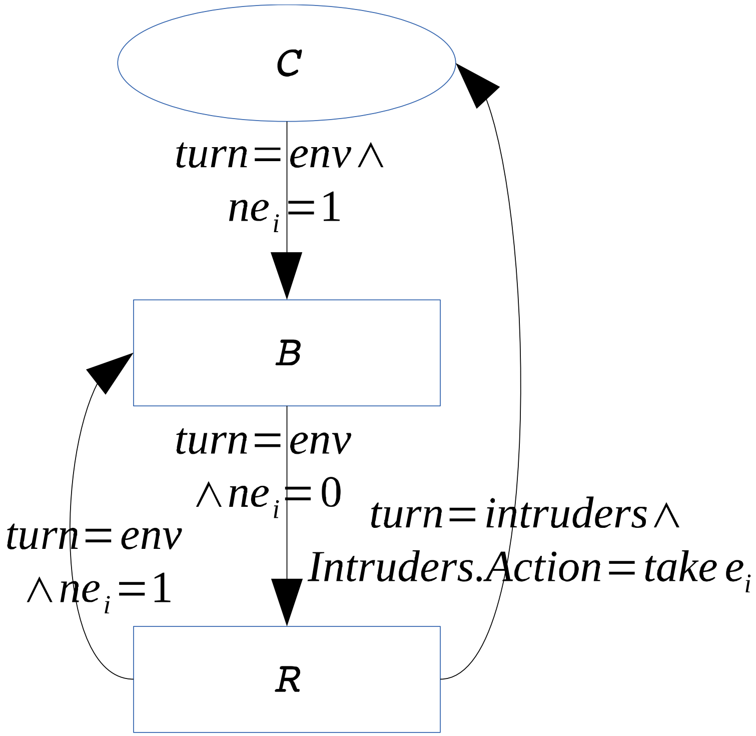


Evolution of v_i and e_i in **Environment**

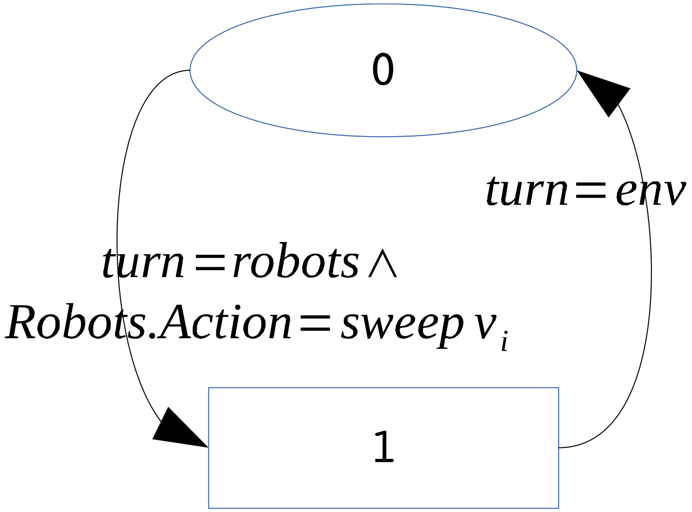
$$\psi_1 \equiv \bigwedge_{j=1}^{k} \overline{\mathtt{ne}}_j = 1$$

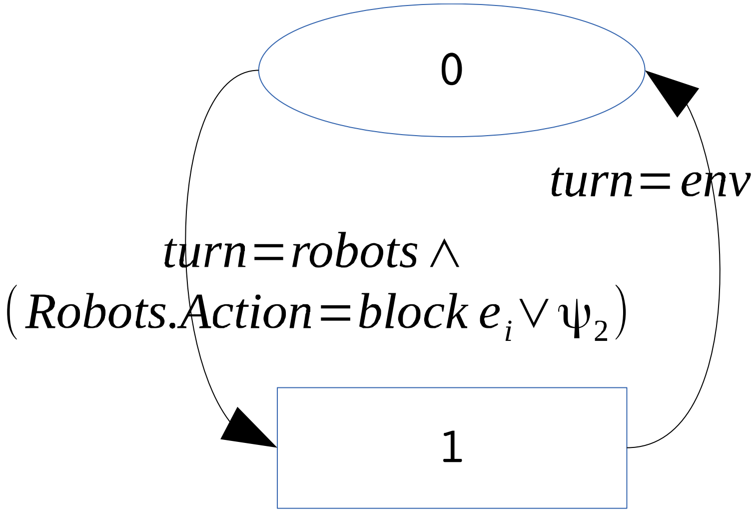
(Adjacent edges to v_i are e_j .)





Evolution of nv_i and ne_i in **Environment**





The **Robots** agent (1)

- Variables: $n = 0, \ldots, d$ (number of agents)
- Actions: **sweep** v_i, **block** e_i, **null**
- Protocol: initially

$$
\begin{gathered}
Environment.\mathtt{turn} = \mathtt{robots} \wedge \\
\mathtt{n} = d \wedge \sum_{i=1}^{n} Environment.\mathtt{v}_i = \mathcal{C},
\end{gathered}
$$

and all actions are enabled. Later **sweep** v_i is enabled if v_i is contaminated and an adjacent vertex is clear:

$$
\begin{gathered}
Environment.\mathtt{turn} = \mathtt{robots} \wedge \\
Environment.\mathtt{v}_i = \mathcal{C} \wedge \bigvee_{j=1}^{k} Environment.\bar{\mathtt{v}}_j = \mathcal{R},
\end{gathered}
$$

The **Robots** agent (2)

- **Block** e_j and **null** are enabled if

$$\begin{aligned} &Environment.\texttt{turn} = \texttt{robots} \wedge k \leq \texttt{n} < d \wedge \\ &Environment.\texttt{ne}_j = 0 \wedge \\ &((Environment.\texttt{v}_p = \mathcal{R} \wedge Environment.\texttt{v}_q = \mathcal{C}) \vee \\ &(Environment.\texttt{v}_p = \mathcal{C} \wedge Environment.\texttt{v}_q = \mathcal{R})), \end{aligned}$$

where v_p and v_q are the end vertices of e_j and k is the number of robots needed to block e_j.

- In all other cases only **null** is enabled.

Handling the number of robots

- For each sweep action sweep v_i , the value n is defined as
$$n' = n - k$$
where k is the number of robots needed to sweep v_i.
- For each block action block e_j , the value n is defined as
$$n' = n - t$$
where t is the number of robots needed to block e_j.
- When no vertices are to be swept or edges blocked, i.e. when action is $null$, then ne is reset to its initial value.

The **Intruders** agent

- Only one variable: ***recontamination*** (Boolean)
- Actions: **take** v_i , **take** e_j (to recontaminate)
- **take** v_i is enabled if

$$Environment.\mathtt{turn} = \mathtt{intruders} \wedge$$

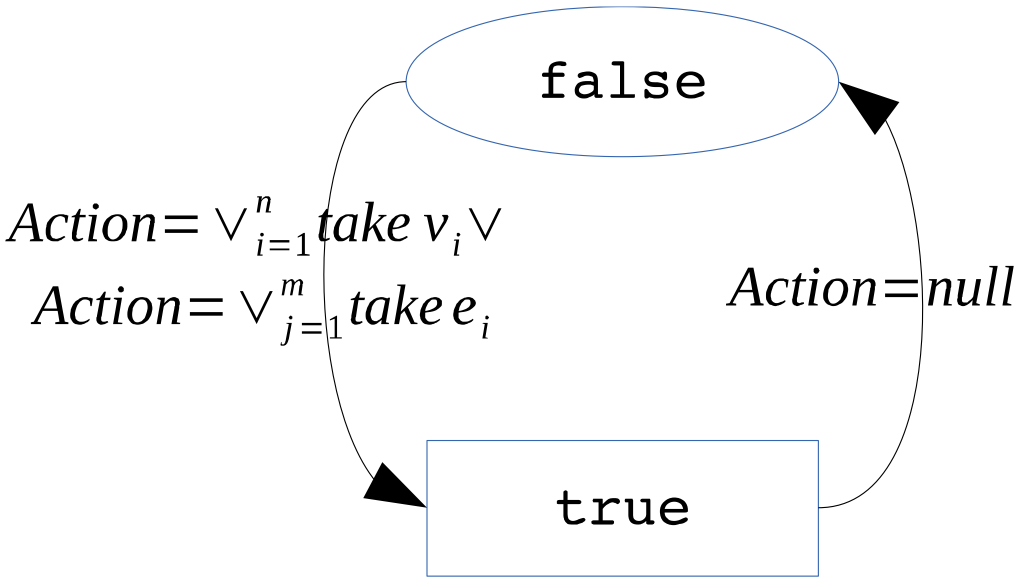
$$Environment.\mathtt{v}_i = \mathcal{R} \wedge \bigvee_{j=1}^{k} Environment.\bar{\mathtt{e}}_j = \mathcal{C},$$

- **take** e_j is enabled if

$$Environment.\mathtt{turn} = \mathtt{intruders} \wedge Environment.\mathtt{e}_j = \mathcal{R} \wedge$$
$$(Environment.\bar{\mathtt{v}}_1 = \mathcal{C} \vee Environment.\bar{\mathtt{v}}_2 = \mathcal{C})$$

Evolution of **Intruders** agent

Evolution of variable ***recontamination*** **:**



Specifications for CTL queries

- **Recontamindated** holds whenever the ***recontamination*** variable of the **Intruders** holds.
- **Graph-cleared** becomes true when all vertices and edges are free of contamination, i.e.

$$\bigwedge_{i=1}^{n} \mathtt{v}_i = \mathcal{R} \wedge \bigwedge_{j=1}^{m} (\mathsf{e}_j = \mathcal{R} \vee \mathsf{e}_j = \mathcal{B}).$$

The CTL query to be used

- MCMAS is run to check whether the formula

$$E(\neg recontaminated\ U\ \ graph\_cleared).$$

  can be satisfied.

- If the answer is yes then MCMAS provides sample paths, each of which can be used as our graph clear algorithm (strategy).

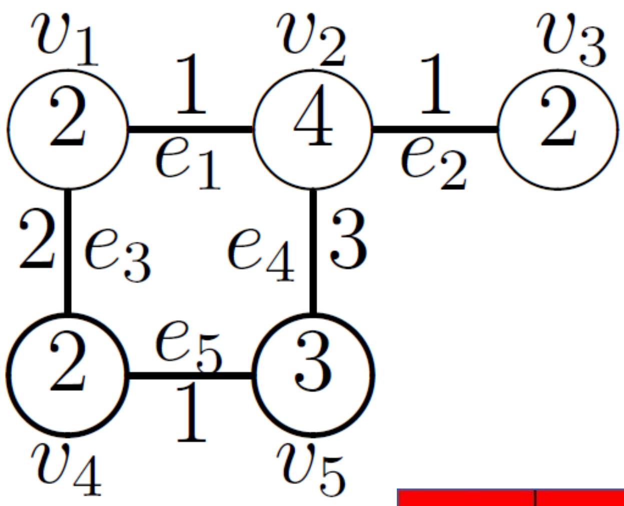
The main Theorem

If

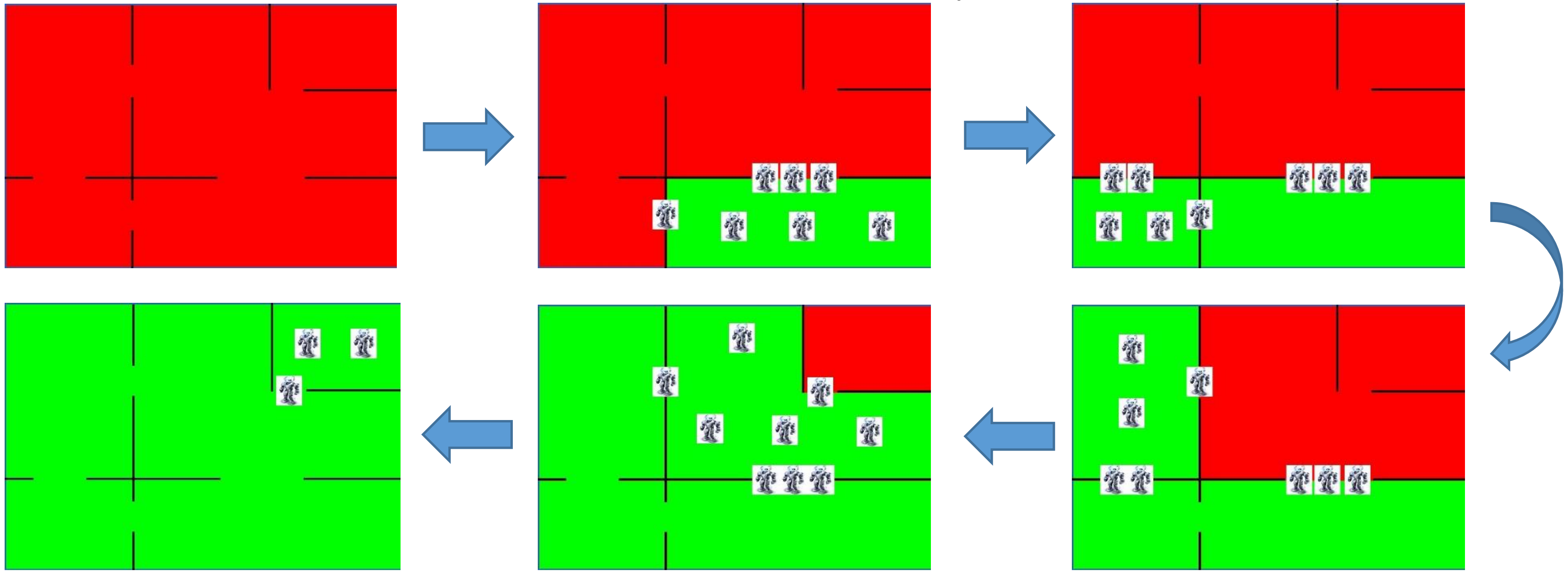
$$E(\neg recontaminated \; U \; graph\_cleared)$$

is satisfied by the SG/CG graph model, then every path satisfying it is an algorithms for the robots to clear the graph and no recontamination can occur during the clearing process.

Example: Graph-Clear strategy



$\nu(G)$	a	$c(a)$
$\mathcal{CCCCC}$ $\mathcal{CCCCC}$	00001 00011	7
$\mathcal{CCCCR}$ $\mathcal{CCCBB}$	00010 00111	8
$\mathcal{CCCRR}$ $\mathcal{CCBBB}$	10000 10110	8
$\mathcal{RCCRR}$ $\mathcal{BCBBR}$	01000 11010	9
$\mathcal{RRCRR}$ $\mathcal{BBRBR}$	00100 01000	3
$\mathcal{RRRRR}$ $\mathcal{RBRRR}$		

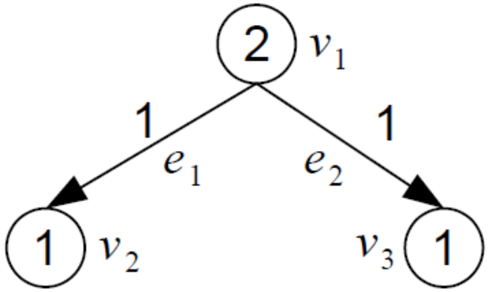


Outline

- Introduction to pursuit-evasion problem
- Compute clearing strategies by model checking
- Find optimal execution of cleaning strategies
- Conclusions

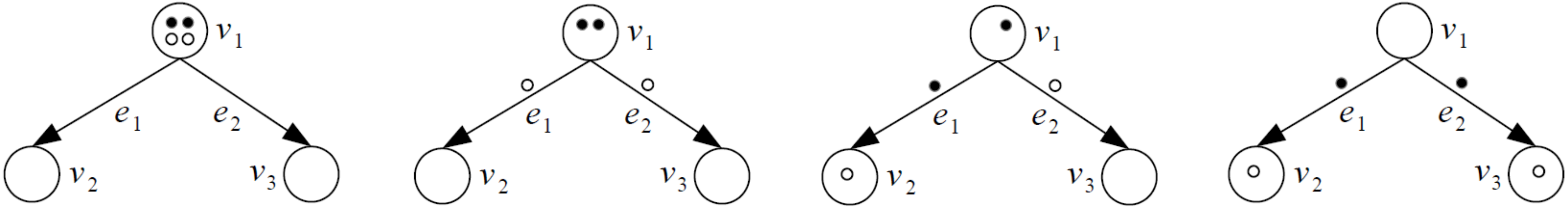
Time-optimal search strategies

- The model-checker-based strategy-search can result in solutions of varying time periods in terms of occupancy steps



$\nu(G)$	a	$c(a)$
$\mathcal{CCC}\ \mathcal{CC}$	100 11	4
$\mathcal{RCC}\ \mathcal{BB}$	010 11	3
$\mathcal{RRC}\ \mathcal{BB}$	001 01	2
$\mathcal{RRR}\ \mathcal{RR}$		

$\nu(G)$	a	$c(a)$
$\mathcal{CCC}\ \mathcal{CC}$	100 11	4
$\mathcal{RCC}\ \mathcal{BB}$	011 11	4
$\mathcal{RRR}\ \mathcal{RR}$		



Optimizing the clearance time under resource constraints: assumptions

- Vertex sweeping and edge clearing costs remain in terms of number of robots.
- Robot travel-distances along edges are specified.
- Robot transition from edge to vertex is assumed to need same time for all robots everywhere.
- All robots are assumed to travel with same speed, the travel time of robots is proportionate to distance

LP system for optimal strategies (1)

- Assumptions
  - Let $l = n + m$ be the number of possible locations, and k searchers.
  - The graph can be cleared in n steps.
  - Initially searchers are placed into a vertex or an edge.
- General constraints
  - $l \times (n+1)$ binary LP variables $X_1, \ldots, X_{l \cdot (n+1)}$ for locations of each robot
  - The initial location of each robot

    $$X_{j \cdot l+1} + \cdots + X_{(j+1) \cdot l} = 1$$
  - The location of each robot at the i-th step

    $$X_{i \cdot k \cdot l + j \cdot l + 1} + \cdots + X_{i \cdot k \cdot l + (j+1) \cdot l} = 1$$
  - $\Delta_1 = (n+1) \cdot k \cdot l$

LP system for optimal strategies (2)

- General constraints
  - For each robot moving from location p to q,

    $$2 \cdot X_{f(p,q)} - X_{(i-1)\cdot k\cdot l+j\cdot l+p} - X_{i\cdot k\cdot l+j\cdot l+q} \leq 0$$

    where

    $$f(p,q) = \Delta_1 + (i-1)\cdot k\cdot l^2 + j\cdot l^2 + (p-1)\cdot l + q$$
  - The following constraint guarantee that only one of l^2 variables is 1

    $$\sum_{1\leq p,q\leq l} X_{f(p,q)} = 1$$
  - $\Delta_2 = n \cdot k \cdot l^2$

LP system for optimal strategies (3)

- General constraints
  - Let D_i be the maximum distance that a robot can move at the i-th step

    $\Lambda_{0\leq j<k}\{(\sum_{1\leq p,q\leq l} d_{p,q} \cdot X_{f(p,q)}) - D_i \leq 0\}$
  - $\Delta_3 = n$
  - Object function

    $\sum_{1\leq i\leq n} D_i$

LP system for optimal strategies (4)

- Constraints for Graph-Clear strategies
  - Let c_{e_r} be the cost of blocking edge e_r
  - $Y_{i \cdot m+r}$ represents e_r being blocked at the i-th step
    $c_{e_r} \cdot Y_{i \cdot m+r} - \sum_{0 \leq j<k} X_{i \cdot k \cdot l+j \cdot l+n+r} \leq 0$
  - The following equation guarantees that $Y_{i \cdot m+r}$ is 1 iff the number of robots in the edge is sufficient
    $\sum_{j=0}^{k-1} X_{i \cdot k \cdot l+j \cdot l+n+r} + \left(c_{e_r} - 1 - k\right) \cdot Y_{i \cdot m+r} \leq c_{e_r} - 1$
  - $\Delta_4 = n \cdot m$

LP system for optimal strategies (5)

- Constraints for Graph-Clear strategies
  - Let c_{v_r} be the cost of sweeping vertex v_r
  - $Z_{i \cdot n+r}$ represents v_r being swept at the i-th step

    $$c_{v_r} \cdot Z_{i \cdot n+r} - \sum_{0 \leq j < k} X_{i \cdot k \cdot l+j \cdot l+r} \leq 0$$
  - The following equation guarantees that $Z_{i \cdot n+r}$ is 1 iff the number of robots in the vertex is sufficient

    $$\sum_{j=0}^{k-1} X_{i \cdot k \cdot l+j \cdot l+r} + \left(c_{v_r} - 1 - k\right) \cdot Z_{i \cdot n+r} \leq c_{v_r} - 1$$
  - Each adjacent edge e_s has to be blocked during sweeping

    $$Z_{i \cdot n+r} - Y_{i \cdot m+s} \leq 0$$
  - $\Delta_5 = n^2$

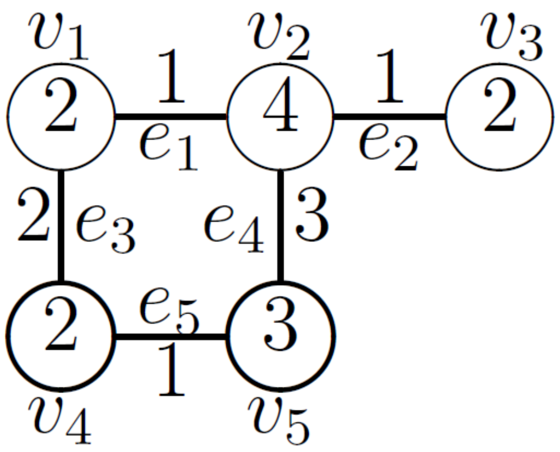
LP system for optimal strategies (6)

- Constraints for Graph-Clear strategies
  - A Graph-Clear strategy clears one vertex at each step
    $\sum_{r=1}^{n} Z_{i \cdot n+r} \geq 1$
  - When a strategy finishes, all vertices have to cleared
    $\sum_{i=1}^{n} Z_{i \cdot n+r} \geq 1$
  - Contiguous search requirement
    $Z_{i \cdot n+r} - \sum_{j=1}^{i-1} \sum_{p \in V_r} Z_{j \cdot n+p} \leq 0$
    $Y_{i \cdot m+r} - \sum_{j=1}^{i} \sum_{p \in E_r} Z_{j \cdot n+p} \leq 0$
    $Y_{(i-1) \cdot m+r} - \sum_{j=1}^{i} Z_{j \cdot n+p} - Y_{i \cdot m+r} \leq 0$
  - $\Delta = \Delta_1 + \Delta_2 + \Delta_3 + \Delta_4 + \Delta_5$

LP system for optimal strategies (7)

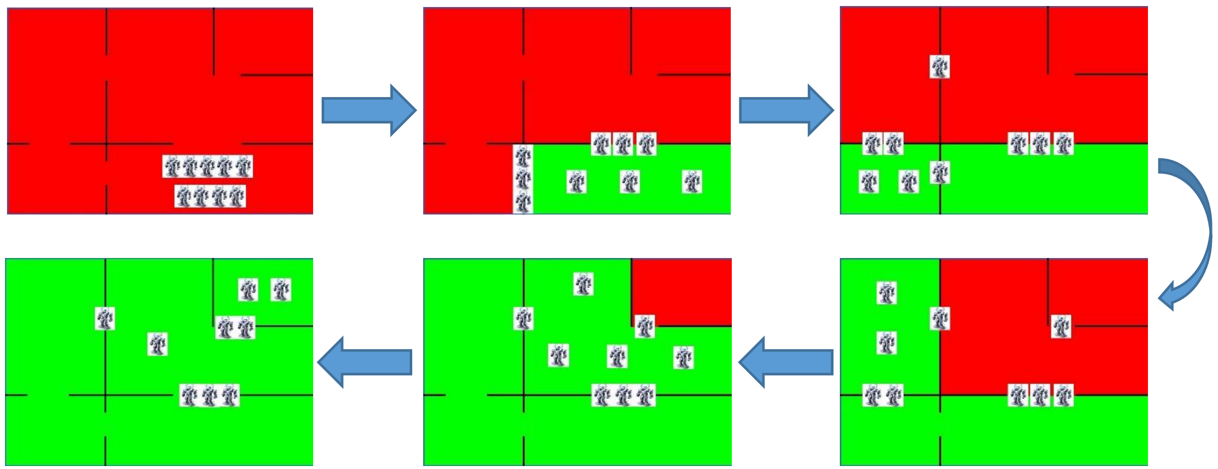
- Constraints for executing a predefined strategy
  - $\sum_{j=1}^{n} X_{i \cdot n \cdot l + j \cdot l + n + r} \geq c_{e_r}$
  - $\sum_{j=1}^{n} X_{i \cdot n \cdot l + j \cdot l + r} \geq c_{v_r}$
  - $\Delta = \Delta_1 + \Delta_2 + \Delta_3$

Example: execution of Graph-Clear strategy

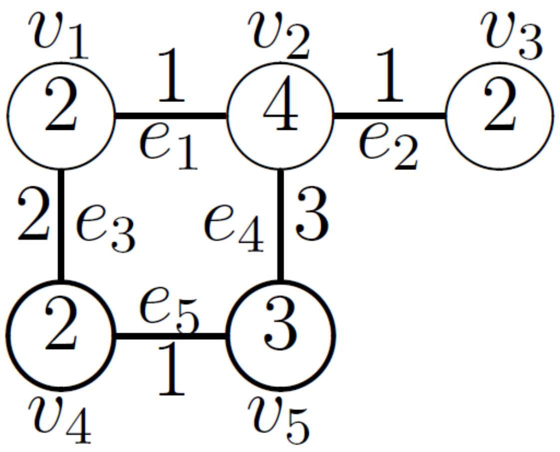


Step	Robot 1	2	3	4	5	6	7	8	9	Time
0	v_5	v_5	v_5	v_5	v_5	v_5	v_5	v_5	v_5	
1	v_5	e_5	v_5	e_5	e_4	e_4	e_4	v_5	e_5	1
2	v_4	e_3	e_4	e_3	e_4	e_1	e_4	e_5	v_4	2
3	e_3	v_1	e_4	v_1	e_4	e_1	e_4	v_5	e_3	1
4	v_2	v_2	e_4	e_1	e_4	e_2	v_2	e_4	v_2	3
5	e_2	v_2	e_4	e_1	e_4	v_3	e_2	e_4	v_3	2

$\nu(G)$	a	$c(a)$
$\mathcal{CCCCC\ CCCCC}$	00001 00011	7
$\mathcal{CCCCR\ CCCBB}$	00010 00111	8
$\mathcal{CCCRR\ CCBBB}$	10000 10110	8
$\mathcal{RCCRR\ BCBBR}$	01000 11010	9
$\mathcal{RRCRR\ BBRBR}$	00100 01000	3
$\mathcal{RRRRR\ RBRRR}$		

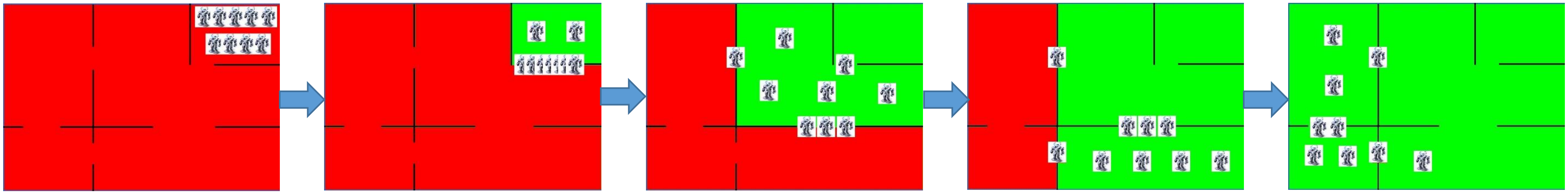


Example: an optimal Graph-Clear strategy



Step	Robot 1	2	3	4	5	6	7	8	9	Time
0	v_3	v_3	v_3	v_3	v_3	v_3	v_3	v_3	v_3	
1	e_2	v_3	e_2	v_3	e_2	e_2	e_2	e_2	e_2	1
2	e_4	e_2	e_1	v_2	v_2	e_4	e_4	v_2	v_2	2
3	e_5	e_4	e_4	v_5	e_4	v_5	v_5	e_1	v_5	2
4	v_4	v_4	v_1	e_1	v_1	v_5	e_5	e_3	e_3	3

$\nu(G)$	a	$c(a)$
$\mathcal{CCCCC}$ $\mathcal{CCCCC}$	00100 01000	3
$\mathcal{CCRCC}$ $\mathcal{CBCCC}$	01000 11010	9
$\mathcal{CRRCC}$ $\mathcal{BBCBC}$	00001 10011	8
$\mathcal{CRRCR}$ $\mathcal{BRCBB}$	10010 10101	8
$\mathcal{RRRRR}$ $\mathcal{BRBRB}$		



Outline

- Introduction to pursuit-evasion problem
- Compute clearing strategies by model checking
- Find optimal execution of cleaning strategies
- Conclusions

Conclusions

- Methodology was developed to use model checking methods to find pursuit-evasion solutions for robots and use Linear Programming to derive execution strategies for time optimization.
- Model checking methods can be implemented onboard robots to enhance their collective problem solving ability.
- Coordination of real-time execution robustness is a future problem yet.

Reference

- **Hongyang Qu, Andreas Kolling, Sandor M Veres.** *Formulating Robot Pursuit-Evasion Strategies by Model Checking*. 19th World Congress of the International Federation of Automatic Control (IFAC'14), pages 3048-3055, 2014
- **Hongyang Qu, Andreas Kolling, Sandor M Veres.** *Computing Time-Optimal Clearing Strategies for Pursuit-Evasion Problems with Linear Programming.* Towards Autonomous Robotic Systems - 16th Annual Conference (TAROS'15), page 216-228, 2015