Reinforcement Learning-Based Dynamic Adaptation Planning Method for Architecture-based Self-Managed Software

Dongsun Kim and Sooyong Park
Sogang University
South Korea

SEAMS 2009, May 18-19, Vancouver, Canada

Outline

- Introduction
- Planning Approaches in Architecture-based Self-Management
  - Offline Vs. Online
- Q-learning-based Self-Management
- Case Study
- Conclusions

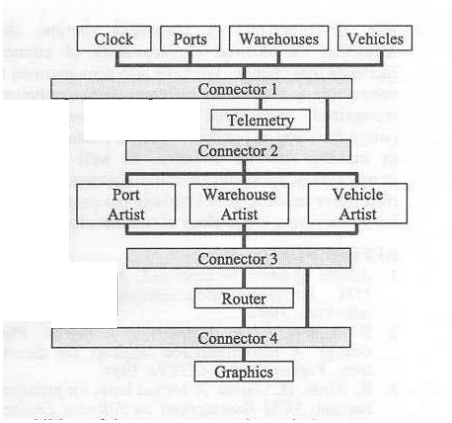
Introduction

Previous Work

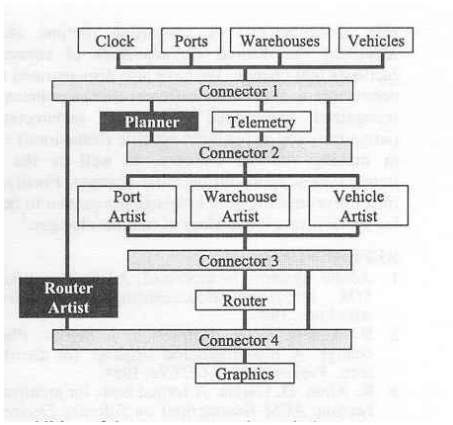
Fully manual reconfiguration

(in **Architecture-Based Runtime Software Evolution** by Oreizy et al. 1998)

Architectural configuration at T_1



Architectural configuration at T_2



System administrator
can reconfigure
the architecture using a console,

→ Difficult to react
rapid and dynamic changes

```
> add component
ClassName: c2.planner.RouterArtist
Name? RouterArtist
> weld
Top entity: Connector1
Bottom entity: RouterArtist
> weld
Top entity: RouterArtist
Bottom entity: Connector4
> start
Entity: RouterArtist
```

Or hard-coded reconfiguration plans

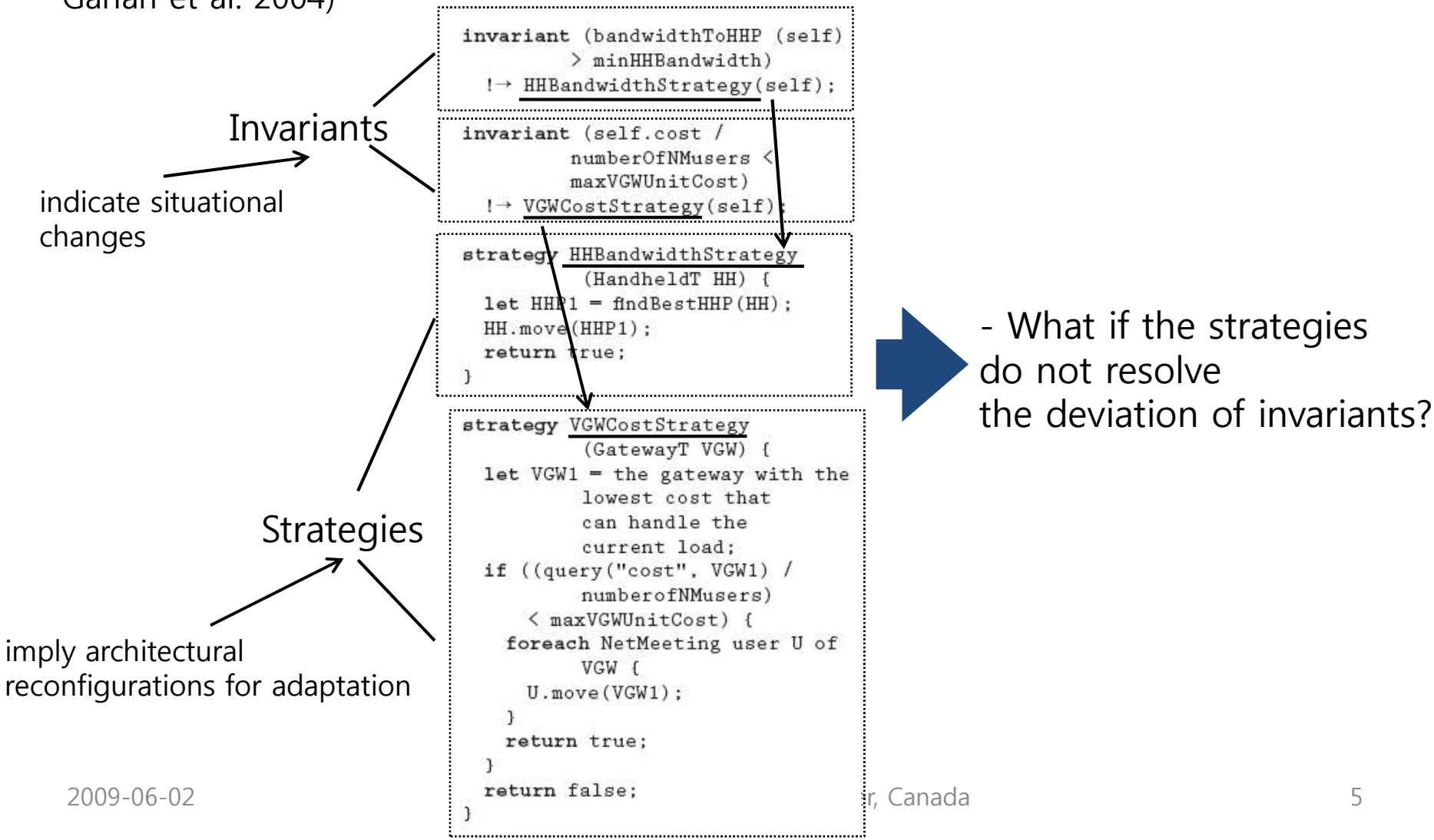
```
try {
    if (model.architectureName().equals("CargoSystem")) {
        Connector above = model.connectorBelow("Ports");
        Connector below = model.connectorAbove("PortArtist");
        model.addComponent("Planner", "planner");
        model.weld(above, "planner");
        model.weld("planner", below);
        model.startEntity("planner");
        return true;
    } else return false;
} catch (ArchitectureModificationException e) {
    return false;
}
```

→ What if the plan does not meets
environmental and situational changes?

Previous Work

Invariants and strategies

(in **Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure**, Garlan et al. 2004)

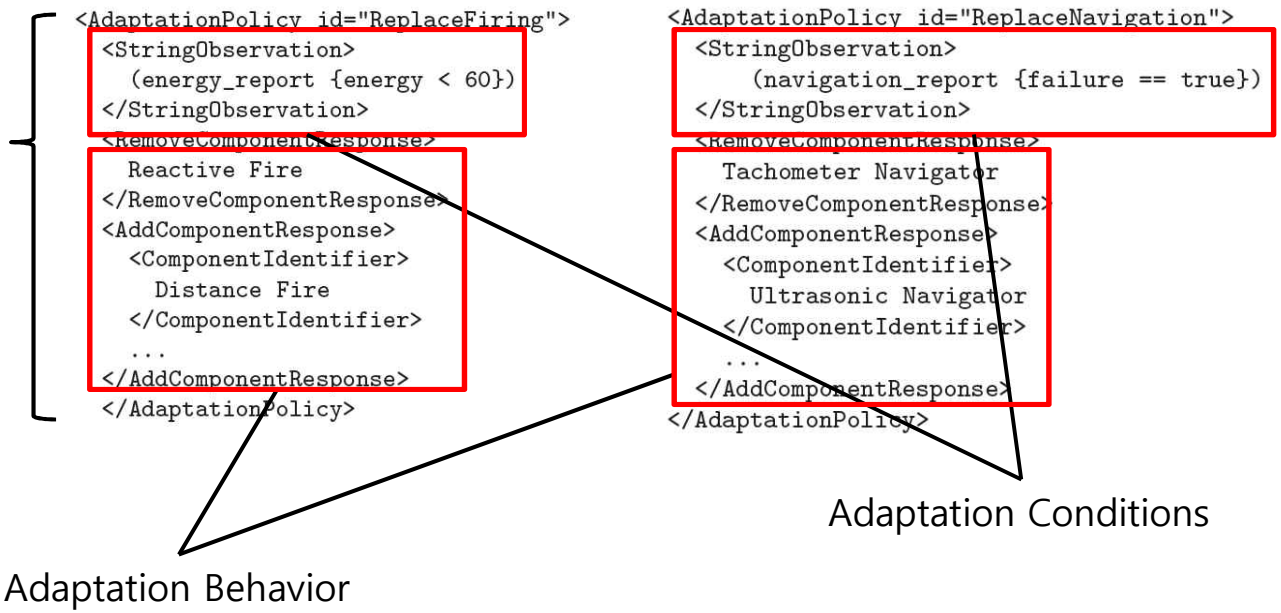


Previous Work

Policy-Based reconfiguration

(in **Policy-Based Self-Adaptive Architectures: A Feasibility Study in the Robotics Domain**, Georgas and Taylor, 2008)

Adaptation policies



- What if policies deviate from what developers (or system administrators) anticipated?

Problems

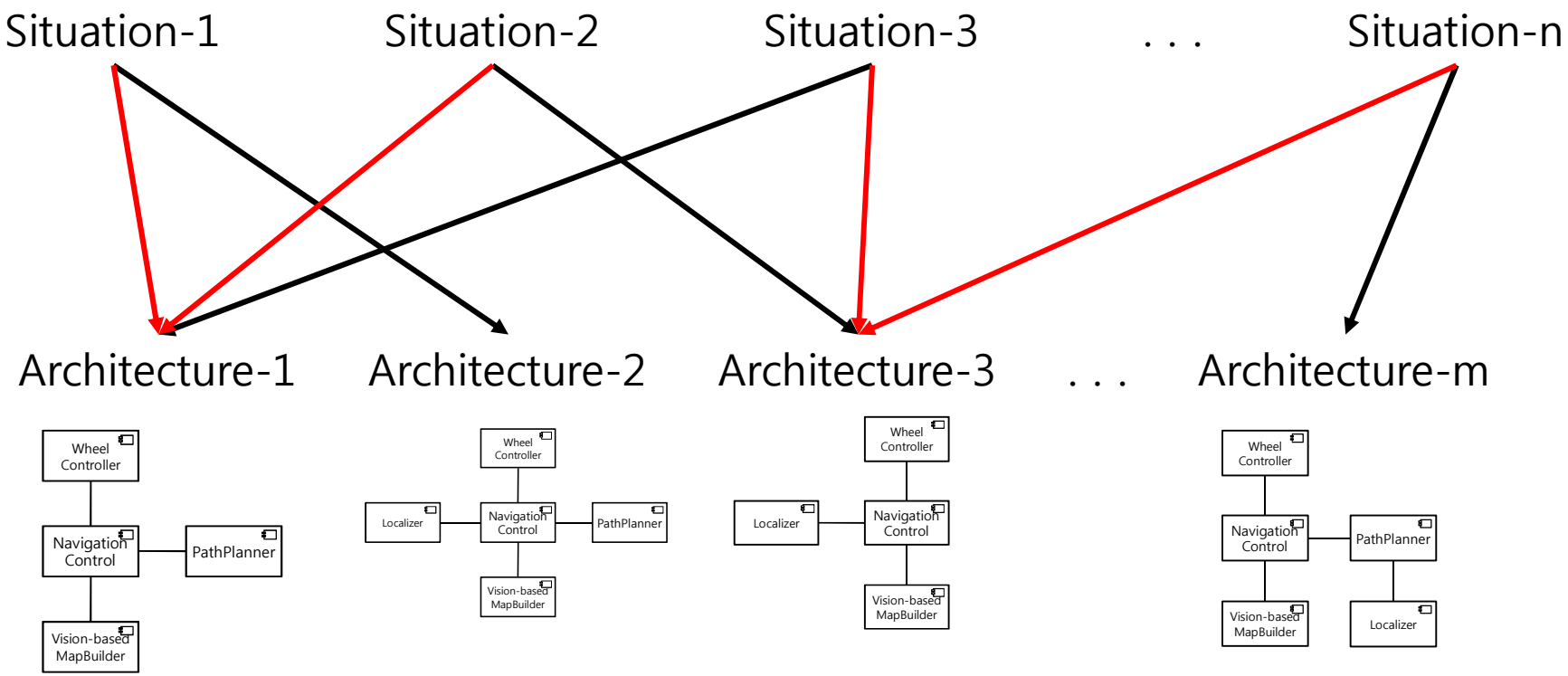
- Existing approaches
  - **Statically** (offline) determine adaptation policies.
  - Administrators can **manually** change policies when the policies are not appropriate to environmental changes.
  - Administrators must know the relationship between the situations and reconfiguration strategies. → **Human-in-the-Loop!**
  - These are appropriate for applications in which **adaptive actions** and their **consequences** are readily predictable.

Requirements on Dynamic Adaptation Planning

- Low chances of human-intervention
  - Some systems cannot be updated by administrators because of physical distances or low connectivity.

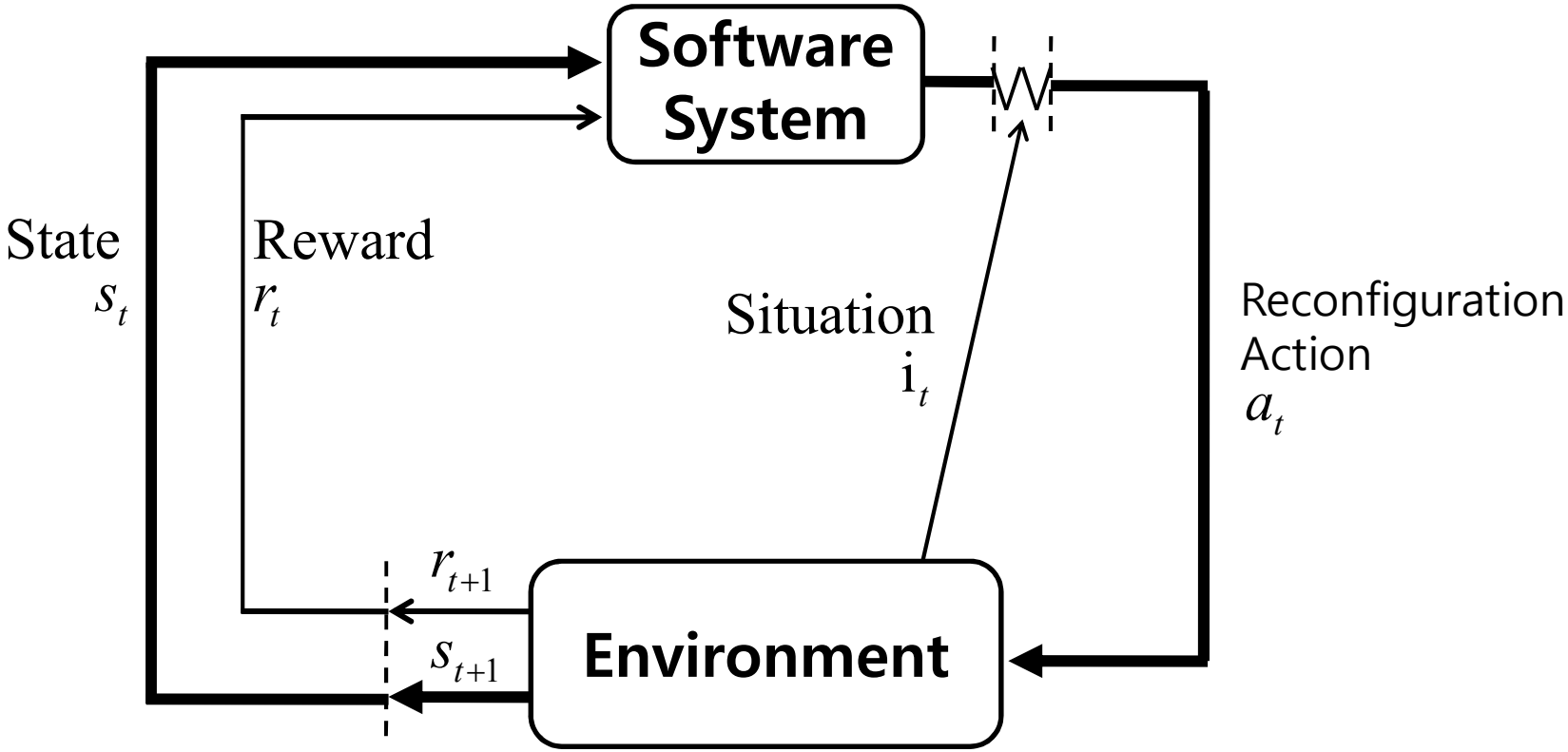
- High level of autonomy
  - Even if administrators can access the system during execution, in some cases, they may not readily make an adaptation plan for the changed situation.

Goals


Situation-1
Situation-2
Situation-3
. . .
Situation-n
Architecture-1
Architecture-2
Architecture-3
. . .
Architecture-m
Wheel Controller
Navigation Control
PathPlanner
Vision-based MapBuilder
Wheel Controller
Localizer
Navigation Control
PathPlanner
Vision-based MapBuilder
Wheel Controller
Localizer
Navigation Control
Vision-based MapBuilder
Wheel Controller
Navigation Control
PathPlanner
Vision-based MapBuilder
Localizer

Planning Approaches to Architecture-based Self-Management

S/W system and Environment Interaction



Formulation

Policy (P) (or a set of plans): indicates that which configuration should be selected when a situation occurs

Situations (S): environmental states that the system concerns

$$P : S \rightarrow C$$

Configurations (C): possible architectural configurations that the system can take

$$V(s_i, P(s_i)) = \max_{\forall c_j \in C} \{V(s_i, c_j)\}, \forall s_i \in S$$

Value Function (V): evaluates the utility of a selected configuration in a given situation

Offline Planning in Self-Management

- Predefined policies
  - Developers or administrators define policies of the system.
  - They generate policies based on already known information **prior to runtime**.
  - However, in general, the information is not complete and **it may become invalid** since the environment can be changed.
  - It is difficult to monitor all details of the environment

Online Planning in Self-Management

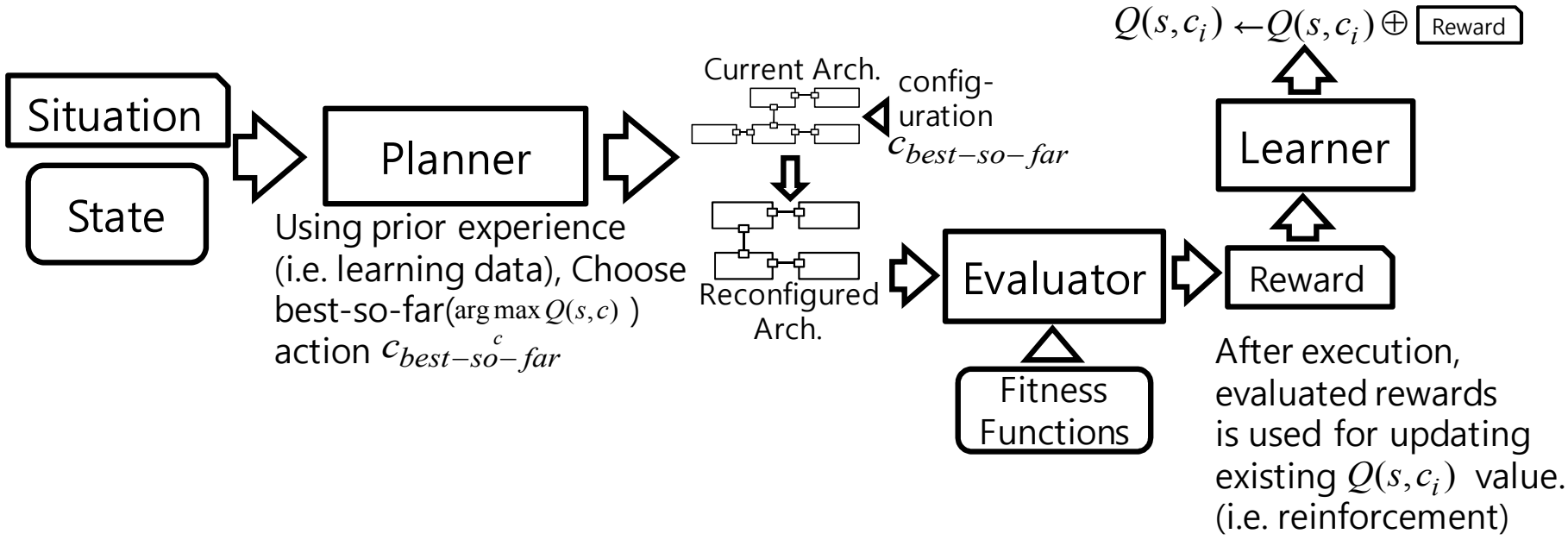
- Adaptive Policy Improvement
  - Allows incomplete information about the environment
    - Gradually **LEARNS** the environment during execution.
- Minimizing human-intervention
  - The system can **autonomously** adapt (update) policies to the environment.

Research Questions

- RQ1 : What kind of information should the system monitors?
- RQ2 : How can the system determine whether its policies are ineffective?
- RQ3 : How can the system change (update) the policies?

Q-learning-based Self-Management

On-line Evolution Process



Detecting Situations and States

- Situations are a set of events that the system is interested in (RQ1).
  - e.g., ‘signal-lost’, ‘hostile-detected’
- States are a set of status information that can be observable from the environment (RQ1).
  - e.g., distance={near,far}, battery={low,mid,full}
- Identifying states and situations
  - They can be identified from system execution scenarios.

Planning

- When a situation is detected,
  - The system should determine what it must do (i.e., **reconfiguration actions**)
  - State information may influence the decision.

- **Exploration** Vs. **Exploitation**
  - Usually the system use the current policy.
  - With some probability, it must try another action because the current policy may not the best one or the environment can be changed.

Execution

- Reconfiguration
  - According to the chosen policy by planning phase, the system change its configuration.

Evaluation

- Fitness (**Reward**)
  - After a series of execution with the reconfigure architecture, the system should evaluate the effectiveness of it (RQ2).
  - The fitness function can be identified from the goal of the system.
    → e.g., maximize survivability, minimize latency.
  - Systems can have several fitness functions and they can be merged.
    → e.g., weight sum: $r_t = f(t) = \sum_i w_i f_i(t)$

Learning

- Updating policies (RQ3)
  - REINFORCES the current knowledge on selecting reconfiguration actions for observed situations.

- Updating rule

Fitness (reward) observed

$$Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a)]$$

Situation detected

Reconfiguration action

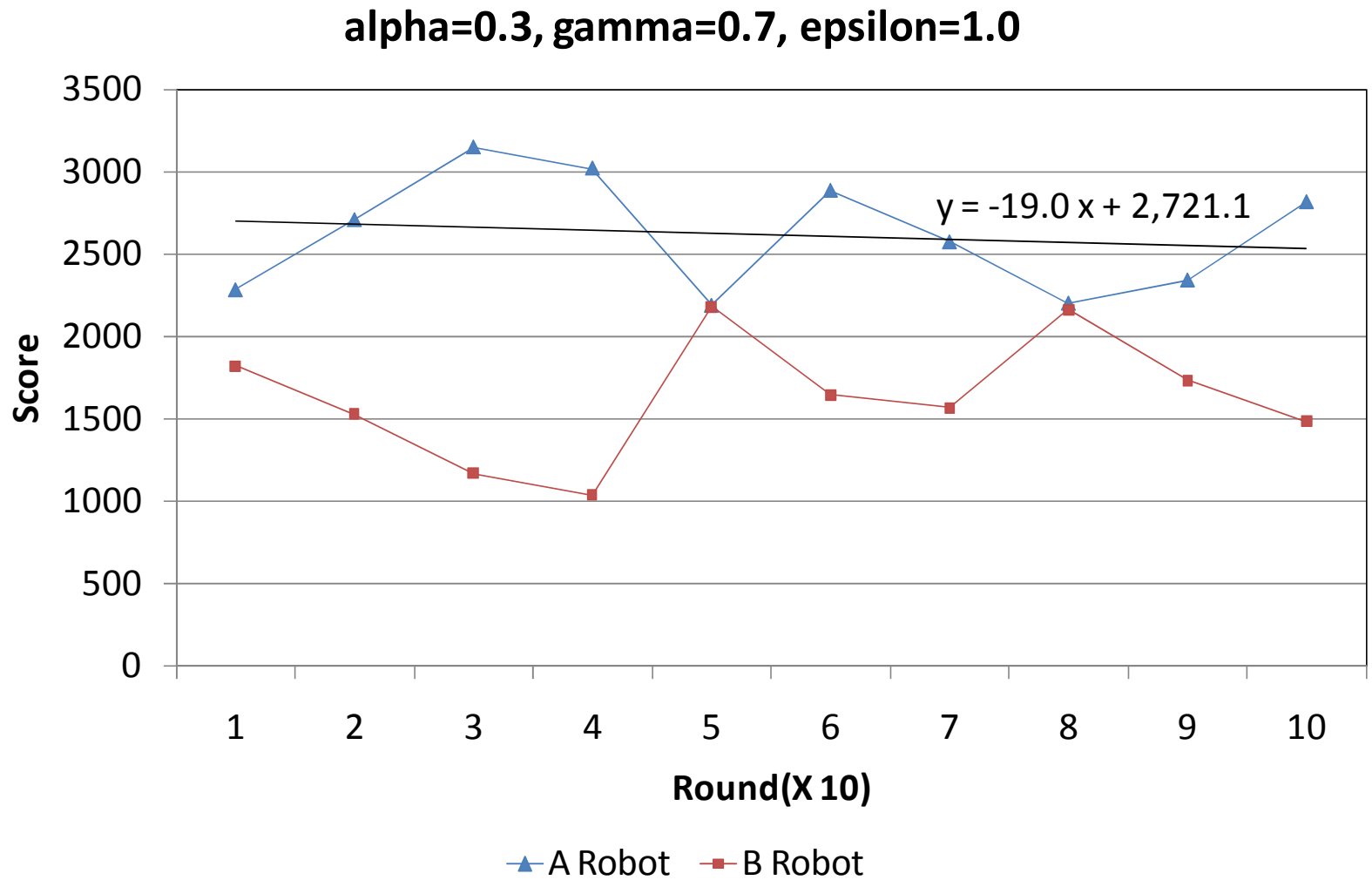
Case Study

Implementation

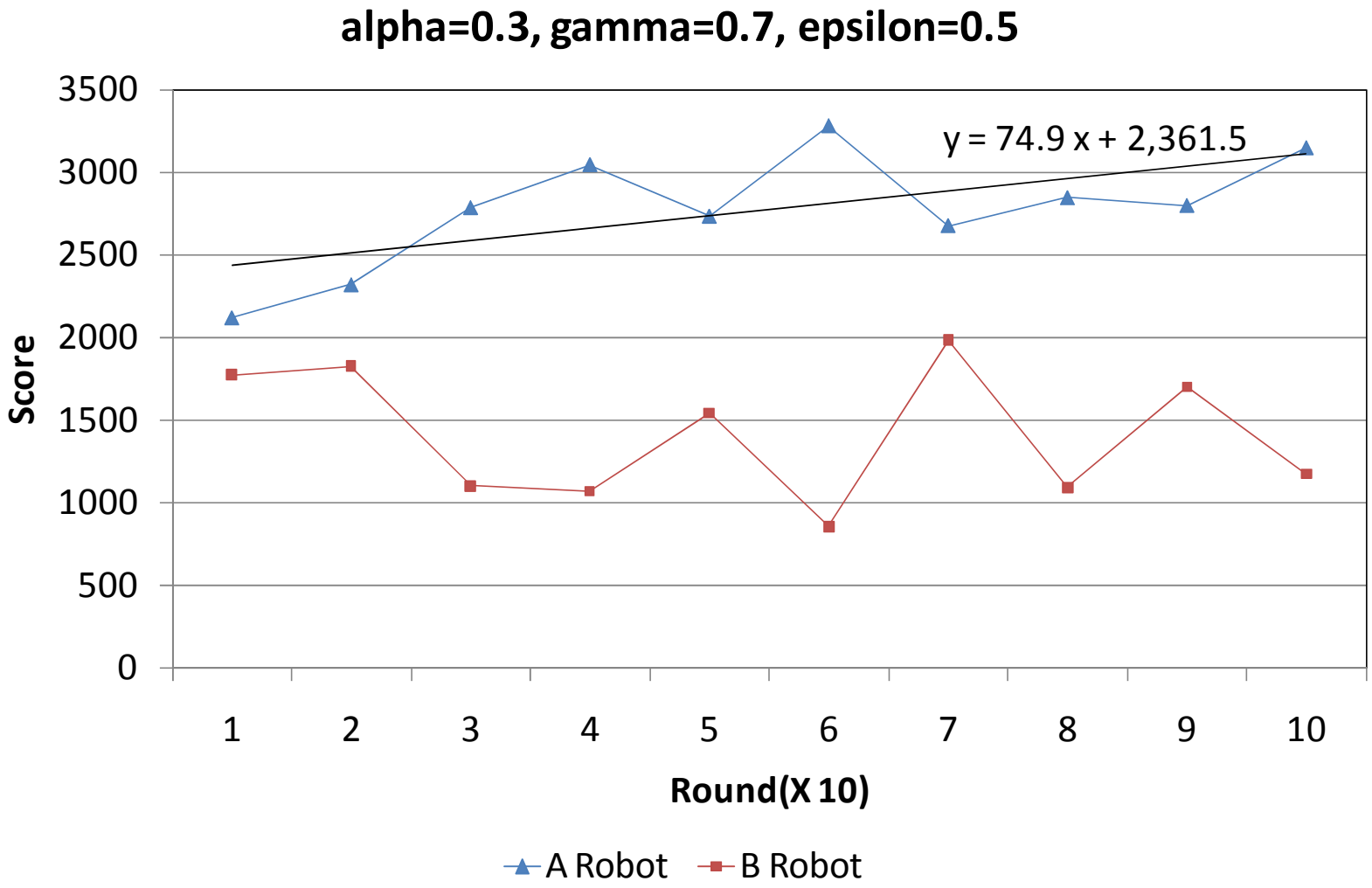
- Robots in Robocode
- 4 situations and 4 state variables
- 16 reconfiguration actions
- Fitness: score
- Environments: enemy robots

- Experiments outline
  - Static (offline) policy
  - Adaptive online policy
  - Online policy in changing environments
    - off (known environment) → off (new) → on

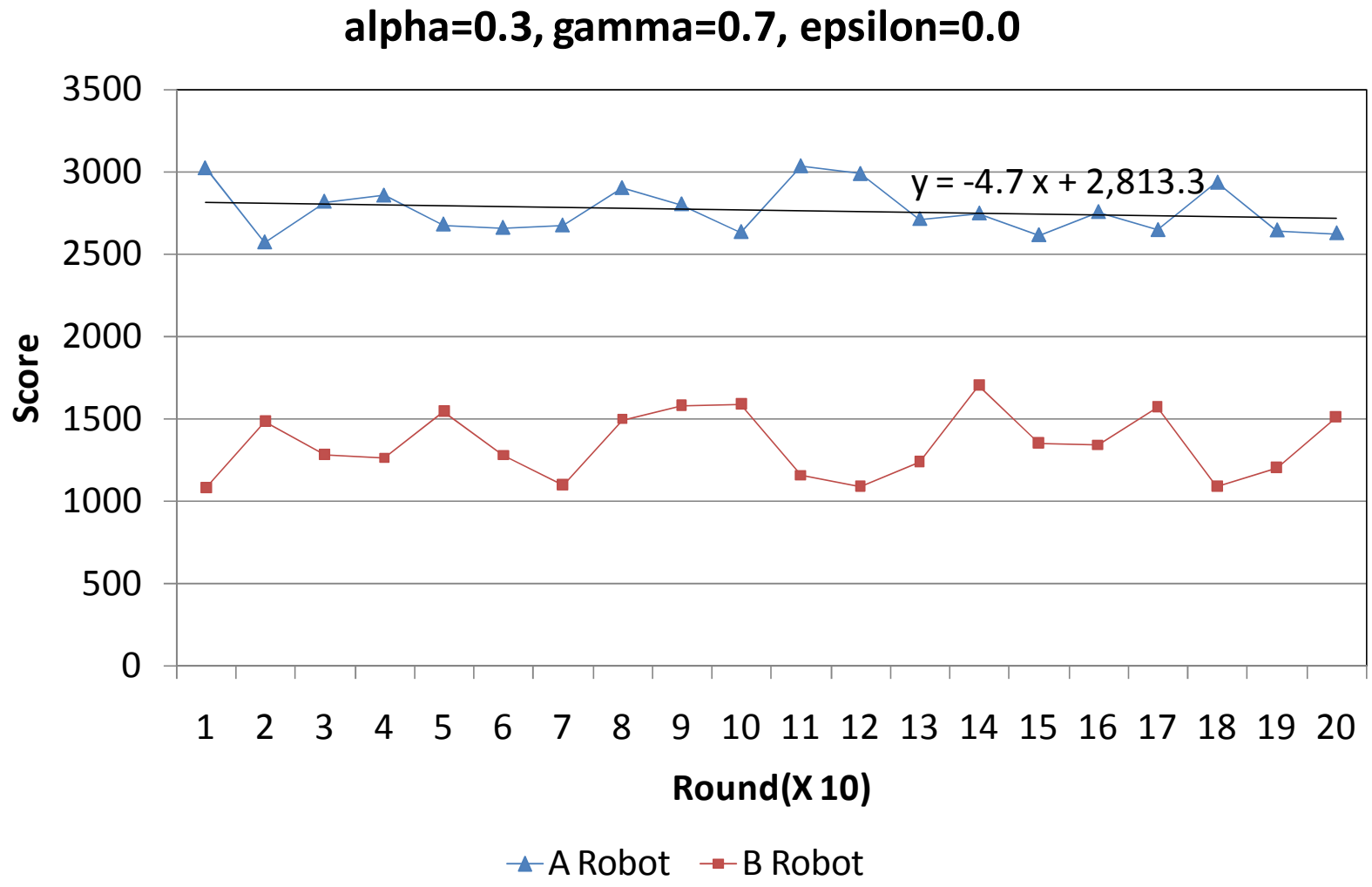
With Static (offline) Policy



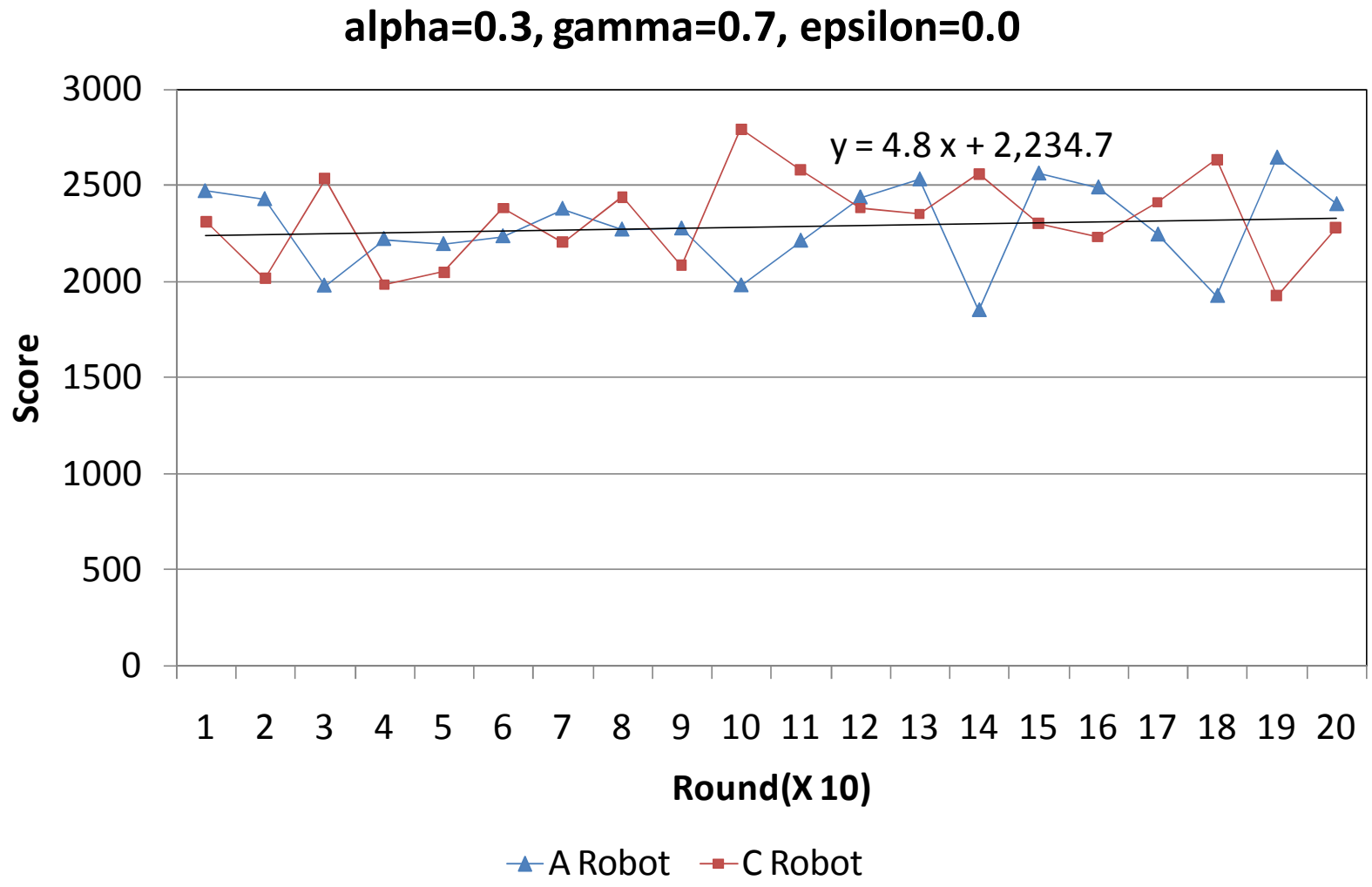
With Online Policy



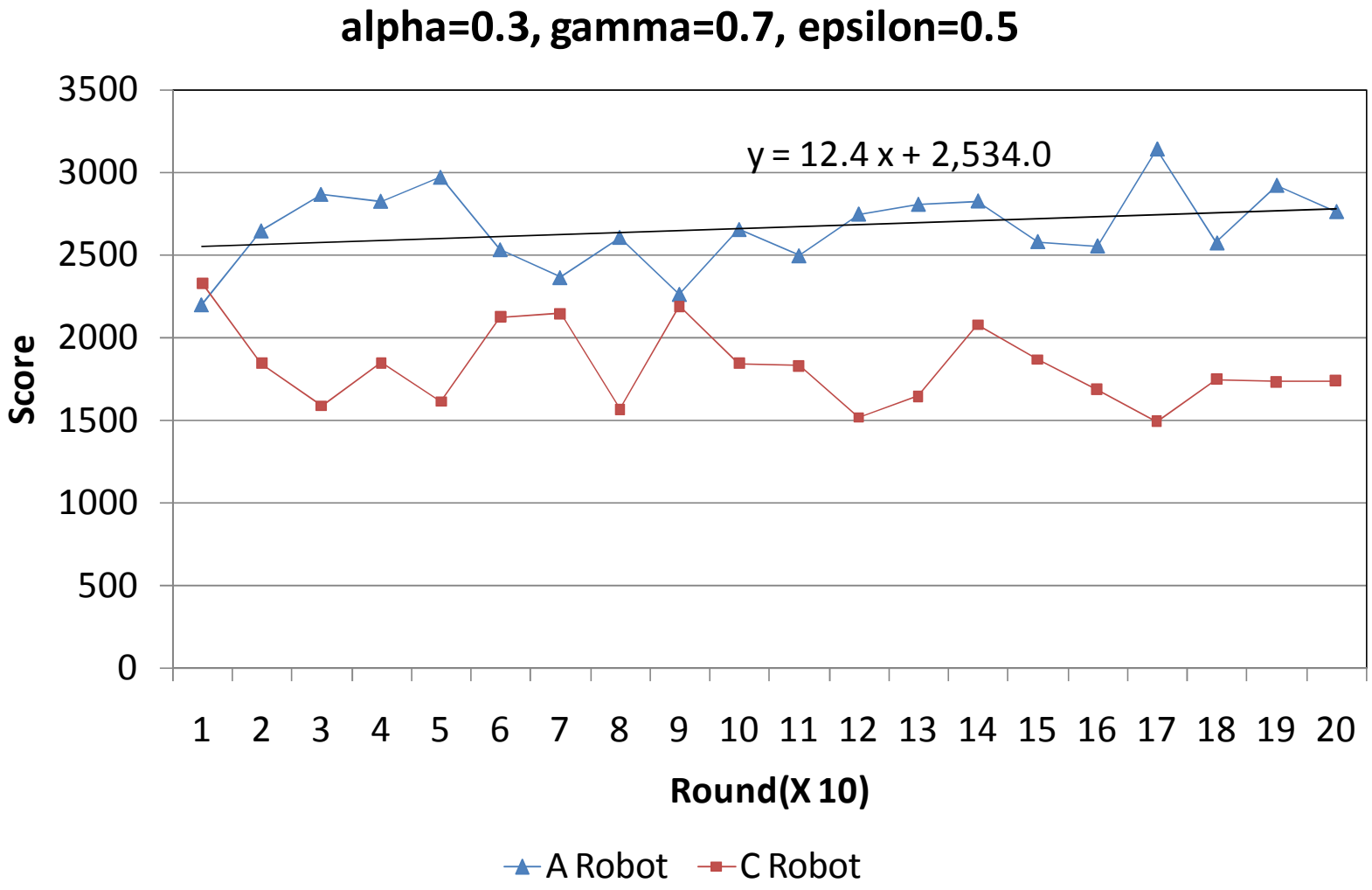
Revisited


alpha=0.3, gamma=0.7, epsilon=0.0
y = -4.7 x + 2,813.3
Score
3500
3000
2500
2000
1500
1000
500
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Round(X 10)
A Robot
B Robot

Enemy Changed (static policy)



Online Policy Again



Conclusions

- Making a decision in architecture-based adaptive software systems.
  - Offline Vs. Online

- A RL-based (Q-learning) Approach
  - Enables adaptive policy control.
  - When the environment has been changed, the system can adapt its policies about architectural configuration to the changed environment.

Further Study

- Better Learning
  - Q-learning might not be the best choice for adaptive software systems.
  - Learning techniques have a large number of parameters.
    → e.g., balancing 'exploration' and 'exploitation'
- Better Representations
  - Situations, configurations, and rewards
  - Better representation may lead to faster and more effective adaptation.
- Scalability
  - Avoiding state explosion.
- Need for Best Practice
  - Comprehensive and common examples

Q&A