

# Organizational Self-Design in Semi-Dynamic Environments

Sachin Kamboj and Keith S. Decker

Department of Computer and Information Sciences  
University of Delaware  
Newark, DE 19716

Multi-Agent Systems

# Outline

- 1 Introduction
  - Motivation
  - Basic Model
  - Problem Representation
- 2 Task and Resource Model
- 3 Approach
  - Agent roles and relationships
  - Organization Formation and Adaptation
  - Reasons for Organizational Change
  - Robustness
- 4 Evaluation and Preliminary Results
  - Comparison with the Contract Net Protocol
  - Evaluation of the three task allocation heuristics
  - Robustness evaluation

# Introduction

## Motivation

### What do we want to do?

- To organize agents and address the following issues:
  - The number of agents needed to solve the problem
  - Allocation of subtasks and resources to the agents
  - Coordination of inter-agent activities

# Introduction

## Motivation

### So what's the problem?

- There is no best way of organizing and all ways of organizing are not equally effective
  - The optimal organizational structure depends on the underlying problem being solved and the environmental conditions
- Environmental conditions or problem structure may change:
  - Precludes the use of a static (compile-time) organization
- All problem instances and environmental conditions are not unique:
  - Precludes the use of a new, bespoke organizational structure for every problem instance

# Introduction

## Motivation

### Our Answer:

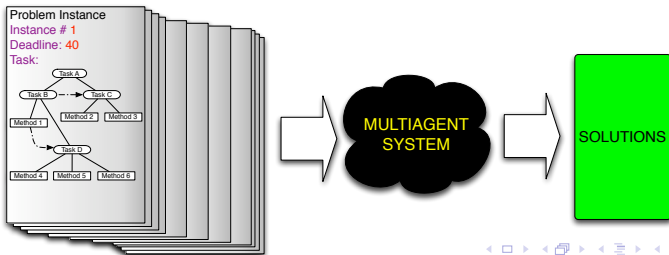
We propose a dynamic run-time approach to organization based on Organizational-Self Design (OSD)

- OSD means agents design their own organizational structures

# Introduction

## Basic Model

- Problem solving requests (task instances) arrive at the organization continuously at varying rates with varying deadlines
- The Agents in the Multiagent System use Organizational Self-Design to generate their own organizational structure and generate solutions to the problem



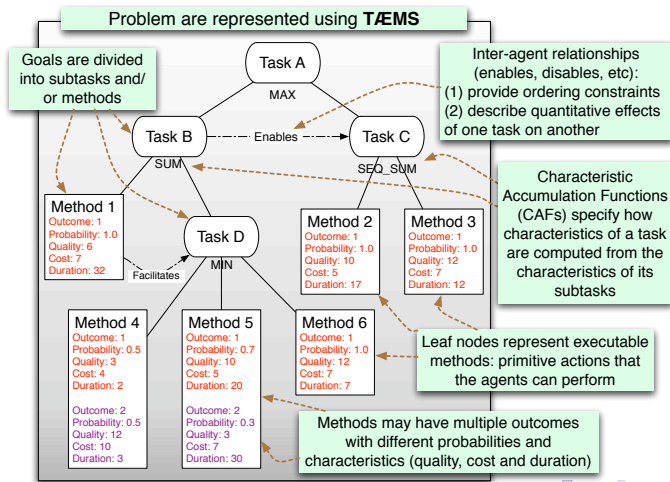
# Introduction

## Basic Approach

- Start off with an initial organization consisting of a single agent, solely responsible for all activities
- Each agent in the organization checks to see if:
  - It is overloaded:
    - It spawns off a new agent to handle part of its load
  - It is free (underloaded)
    - It combines with another agent to save resources

# Introduction

## Problem Representation



# Task and Resource Model

The primary input to the multi-agent system (MAS) is an ordered set of problem solving requests  $\langle P_1, P_2, P_3, \dots, P_n \rangle$ , where

- $P_i = \langle \mathbf{t}_i, \mathbf{a}_i, \mathbf{d}_i \rangle$
- $\mathbf{t}_i$  is the underlying TÆMS task structure
- $\mathbf{a}_i \in \mathbf{N}^+$  is the arrival time
  - the system has no prior knowledge about the task  $\mathbf{t}_i$  before this time
- $\mathbf{d}_i \in \mathbf{N}^+$  is the deadline of the  $i^{\text{th}}$  task instance
  - the task  $\mathbf{t}_i$  must be completed before this time in order to accrue quality

# Task and Resource Model

Every underlying task structure,  $\mathbf{t}_i$ , can be represented using the tuple  $\langle T, \tau, M, Q, E, R, \rho, C \rangle$ , where:

- $T$  is the set of tasks.
  - non-leaf nodes in a TÆMS task structure
  - used to denote goals that the agents must achieve
  - formally, each task  $T_j = (q_j, s_j)$ , where  $q_j \in Q$  and  $s_j \subset (T \cup M)$
- $M$  is the set executable methods, i.e.,  
 $M = \{m_1, m_2, \dots, m_n\}$ ,

# Task and Resource Model

- $Q$  is the set of quality/characteristic accumulation functions (CAFs)
  - determine how a task group accrues quality given the quality accrued by its subtasks/methods.
  - we use four CAFs: MIN, MAX, SUM and EXACTLY\_ONE.
- $E$  is the set of (non-local) effects.
- $R$  is the set of resources.
- $\rho$  is a mapping from an executable method and resource to the quantity of that resource needed (by an agent) to schedule/execute that method  
 $\rho(\text{method}, \text{resource}) : M \times R \rightarrow N.$
- $C$  is a mapping from a resource to the cost of that resource  
 $C(\text{resource}) : R \rightarrow N^+$

# Organizational Design

Organizational design is directly contingent on:

- 1 The task structure
- 2 The environmental conditions under which the problems need to be solved

# Agent roles and relationships

- Organizational structure is primarily composed of
  - Roles:
    - Parts played by the agents enacting the roles in the solution to the problem
    - Reflect long-term commitments made by the agents to a certain course of action (that includes task responsibility, authority, and mechanisms for coordination).
  - Relationships:
    - Coordination relationships that exist between the subparts of a problem

# Roles

## Definition

A role as a TÆEMS subtree rooted at a particular node

- The set  $(T \cup M)$  encompasses the space of all possible roles
- By definition, a role may consist of one or more other (sub-) roles

# Roles

Roles may be of two types:

## 1 Local

- Are the sole responsibility of a single agent
- Agent concerned is responsible for solving all the subproblems of the tree rooted at that node
- Formally,

$$TYPE(a, r) = Local \iff \forall r_i \in SUBTASKS(r) TYPE(a, r_i) = Local$$

## 2 Managed

- Must be coordinated between two or more agents
- Such roles will have two or more descendent local roles that are the responsibility of two or more separate agents
- Formally,

$$TYPE(a, r) = Managed \iff [\exists a_1 \exists r_1 (r_1 \in SUBTASKS(r)) \wedge (TYPE(a_1, r_1) = Managed)] \vee [\exists a_2 \exists a_3 \exists r_2 \exists r_3 (a_2 \neq a_3) \wedge (r_2 \neq r_3) \wedge (r_2 \in SUBTASKS(r)) \wedge (r_3 \in SUBTASKS(r)) \wedge (TYPE(a_2, r_2) = Local) \wedge (TYPE(a_3, r_3) = Local)]$$

# Organization Formation and Adaptation

- Two organizational primitives are used
  - ① Agent Spawning
    - Generation of a new agent
    - New agent handles a subset of the roles of the spawning agent
  - ② Agent Composition
    - Orthogonal to agent spawning
    - Involves the merging of two agents together
    - The combined agent is responsible for enacting all the roles of the agents being merged
- These primitives change:
  - The number of agents in the organization
  - The assignment of roles to the agents
- **OSD is a search in the space of all the role assignments for a suitable role assignment that minimizes or maximizes a performance measure**

# Organization Formation and Adaptation

- All organizational change is local
  - Each organizational change affects, at the most, **two** agents
  - Advantage: Allows the system to scale to 100s of agents
  - Disadvantage: Using more global information will allow better decisions to be made

# Organization Formation and Adaptation I

## Organizational Invariants

The organizational invariants always hold before and after every organizational change

- 1 For a local role, all the descendent nodes of that role will be local.

$$\begin{aligned} TYPE(a, r) = Local &\implies \\ \forall r_i \in SUBTASKS(r) &TYPE(a, r_i) = Local \end{aligned}$$

- 2 Similarly, for a managed (non-local) role, all the ascendent nodes of that role will be managed.

$$\begin{aligned} TYPE(a, r) = Managed &\implies \\ \forall r_i \in SUPERTASKS(r) &\exists a_i (a_i \in A) \wedge (TYPE(a_i, r_i) = Managed) \end{aligned}$$

# Organization Formation and Adaptation II

## Organizational Invariants

- 3 If two local roles that are enacted by two different agents share a common ancestor, that ancestor will be a managed role.

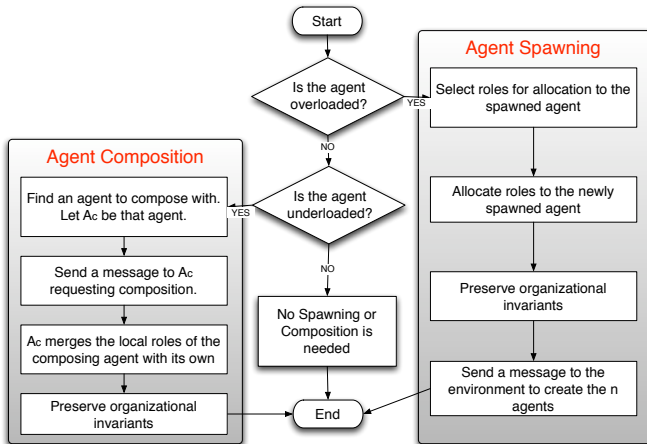
$$\begin{aligned} & (TYPE(a_1, r_1) = Local) \wedge (TYPE(a_2, r_2) = Local) \wedge \\ & (a_1 \neq a_2) \wedge (r_1 \neq r_2) \implies \\ & \quad \forall_{r_i \in (SUPERTASKS(r_1) \cap SUPERTASKS(r_2))} \exists a_i (a_i \in A) \wedge \\ & \quad (TYPE(a_i, r_i) = Managed) \end{aligned}$$

- 4 If all the direct descendants of a role are local and the sole responsibility of a single agent, that role will be a local role.

$$\begin{aligned} & \exists a \exists r \forall_{r_i \in SUBTASKS(r)} (a \in A) \wedge (r \in (T \cup M)) \wedge \\ & (TYPE(a, r_i) = Local) \implies \\ & (TYPE(a, r) = Local) \end{aligned}$$

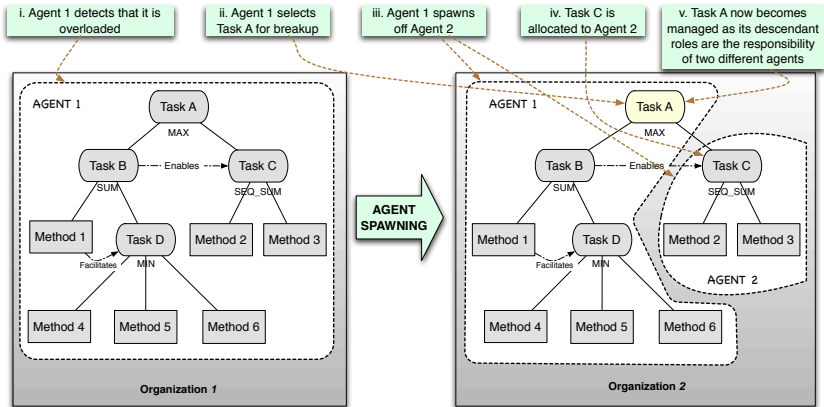
# Organization Formation and Adaptation

## Agent Composition



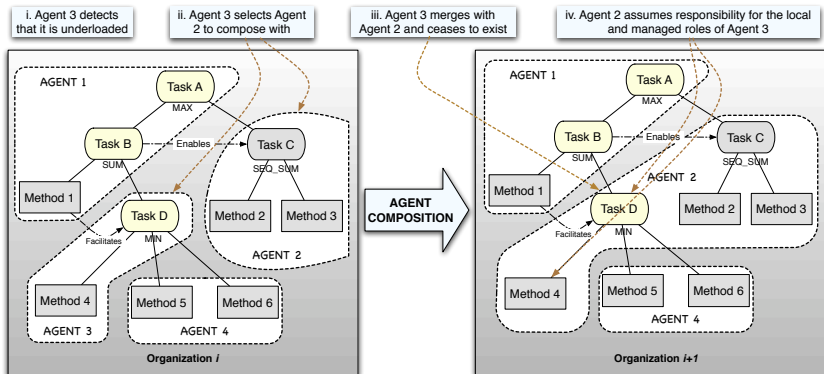
# Organization Formation and Adaptation

## Agent Spawning



# Organization Formation and Adaptation

## Agent Composition



# Organization Formation and Adaptation

## Agent Spawning

Algorithm (SPAWNAGENT(*SpawningAgent*) :  $A \rightarrow A$ )

- 1:  $LocalRoles \leftarrow \{r \subseteq (T \cup M) \mid TYPE(SpawningAgent, r) = Local\}$
- 2:  $NewAgent \leftarrow CREATENEWAGENT()$
- 3:  $NewAgentRoles \leftarrow FINDROLESFORSPAWNEDAGENT(LocalRoles)$
- 4: **for** *role* **in**  $NewAgentRoles$  **do**
- 5:    $TYPE(NewAgent, role) \leftarrow Local$
- 6:    $TYPE(SpawningAgent, role) \leftarrow Unassigned$
- 7: **end for**
- 8: PRESERVEORGANIZATIONALINVARIANTS()
- 9: **return**  $NewAgent$

# Organization Formation and Adaptation

## Agent Spawning

### Question

*Which of its local-roles should it assign to the newly spawned agent and which of its local roles should it keep to itself?*

- The `FINDROLESFORSPAWNEDAGENT()` function takes the set of local roles that are the responsibility of the spawning agent and returns a subset of those roles for allocation to the newly spawned agent.
- This subset is selected based on the results of a cost function (line 4)

# Organization Formation and Adaptation

## Agent Spawning

Algorithm (FINDROLESFORSPAWNEDAGENT  
(SpawningAgentRoles) :  $(T \cup M) \rightarrow (T \cup M)$ )

```
1:  $\mathcal{R} \leftarrow \text{SpawningAgentRoles}$ 
2:  $\text{selectedRoles} \leftarrow \text{nil}$ 
3: for  $\text{roleSet}$  in  $[\mathcal{P}(\mathcal{R}) - \{\phi, \mathcal{R}\}]$  do
4:   if  $\text{COST}(\mathcal{R}, \text{roleSet}) < \text{COST}(\mathcal{R}, \text{selectedRoles})$  then
5:      $\text{selectedRoles} \leftarrow \text{roleSet}$ 
6:   end if
7: end for
8: return  $\text{selectedRoles}$ 
```

# Organization Formation and Adaptation

## Agent Spawning

- Different cost functions will result in different organizational structures
- We evaluated three cost functions (based on three heuristics):
  - 1 Allocating top-most roles first
  - 2 Minimizing total resource cost
  - 3 Balancing execution time

# Organization Formation and Adaptation

## Agent Spawning

### Allocating top-most roles first

- This heuristic always breaks up at the top-most nodes first
  - If the nodes of a task structure were numbered, starting from the root, in a breadth-first fashion, then this heuristic would select the local-role of the spawning agent that had the lowest number and breakup that node
- We selected this heuristic because:
  - 1 it is the simplest to implement
  - 2 fastest to run
    - (The role allocation can be done in constant time)
  - 3 it makes sense from a human-organizational perspective
    - (Corresponds to dividing an organization along functional lines)

# Organization Formation and Adaptation

## Agent Spawning

### Minimizing total resource cost

- This heuristic attempts to minimize the total cost of the resources needed by the agents.
- If  $\mathcal{R}$  be the local roles of the spawning agent and  $R'$  be the subset of roles being evaluated for allocation to the newly spawned agent, then the cost function for this heuristic is given by:

$$\text{COST}(\mathcal{R}, R') \leftarrow \text{GETRESOURCECOST}(\mathcal{R} - R') + \text{GETRESOURCECOST}(R')$$

# Organization Formation and Adaptation

## Agent Spawning

### Balancing execution time

- This heuristic tries to ensure that each agent has an equal amount of work to do
- For each potential role allocation, this heuristic works by calculating the absolute value of the difference between the expected duration of its own roles after spawning and the expected duration of the roles of the newly spawned agent.
  - If this difference is close to zero, then the both the agents have roughly the same amount of work to do.
  - Formally,  
$$\text{COST}(\mathcal{R}, R') \leftarrow |\text{GETEXPECTEDDURATION}(\mathcal{R} - R') - \text{GETEXPECTEDDURATION}(R')|$$

# Reasons for Organizational Change

- Organizational change is expensive
  - requires clock cycles
  - allocation/deallocation of resources
  - various other overhead
- Hence, **stable organizations are desirable**
- Therefore, only change the organizational structure if
  - the task structure changes; or
  - the environmental conditions change
- Right now, we only handle the latter

## Reasons for Organizational Change

- Furthermore, change organizational structure only if the change is permanent
  - Impossible to know
- Make the probability of change be inversely proportional to the time since the last organizational change
  - If time since last change is relatively short, the agents are still adjusting to the changes in the environment
    - the probability of an agent initiating an organizational change should be high.
  - if the time since the last organizational change is relatively large, we have a well-suited stable organizational structure
    - the probability of an agent initiating an organizational change should be low.
- To allow this variation, we use **Simulated Annealing**.

# Reasons for Organizational Change

## Simulated Annealing

- The probability of keeping an existing organizational structure is calculated using the annealing formula,  
$$p = e^{-\frac{\Delta E}{kT}}, \text{ where}$$
  - $\Delta E$  is the "amount" of overload/underload,
  - $T$  is the time since the last organizational change; and
  - $k$  is a constant.
- If  $T$  is large,  $p$ , or the probability of keeping the existing organizational structure is large.
  - Problem: The organization may be too sluggish in its reaction to organizational change or may not respond at all.
  - Solution: Cap  $p$  at a certain threshold
- The probability of organizational change,  $q = 1 - p$
- The mechanism of computing  $\Delta E$  is different for agent spawning than for agent composition

# Reasons for Organizational Change

## Computation of $\Delta E$ for Agent Spawning

- Agent spawning only occurs when the agent doing the spawning is too overloaded and cannot complete all the tasks in its task queue by the given deadlines of the tasks.
- To compute if spawning is necessary,  $\Delta E = \frac{1}{\alpha * Slack}$ , where
  - $\alpha$  is a constant
  - $Slack$  is the difference between the total time available for completion of the outstanding tasks and the sum of the expected time required for completion of each task on the task queue
- If slack is negative, immediately spawn agent.

# Reasons for Organizational Change

## Computation of $\Delta E$ for Agent Composition

- Orthogonal to agent spawning
- Agent composition only occurs when the agents are underloaded
- To calculate if agent composition is necessary,  $\Delta E = \beta * Idle\_Time$ , where
  - $\beta$  is a constant
  - $Idle\_Time$  is the amount of time for which the agent was idle
- If the agent has been sitting idle for a long period of time,  $\Delta E$  is large, which implies that  $p$ , the probability of keeping the existing organizational structure, is low.

# Organization and Robustness

Two commonly used approaches to achieve robustness in multiagent systems are:

- 1 *Survivalist Approach* [Marin et. al.]
  - Involves replicating domain agents
  - Replicas can take over should the original agents fail
  - Advantage: fast recovery
- 2 *Citizen Approach* [Dellarocas et. al.]
  - Involves the use of special monitoring agents (called *Sentinel Agents*)
  - Sentinels detect agent failure and dynamically startup new agents in lieu of the failed ones
  - Advantage: simpler to implement
  - Advantage: requires little modification to the existing organizational structure and coordination protocol.

# Organization and Robustness

- Both of these approaches can be applied to OSD
- We chose the citizens approach
  - shown by [Dellarocas et. al.] to have better performance in the CNP.
  - We designed special monitoring agents
    - that send periodic “are you alive” messages to the domain agents
    - If domain agent fails, it won't respond to these polls
    - After timeout, monitoring agent creates a new agent and delegates the responsibilities of the failed agent to it.
  - Delegation of responsibilities to new agent is non-trivial
    - requires the deducting of state information from a record of the messages received by the failed agent

# Evaluation and Preliminary Results

We conducted 3 sets of experiments to evaluate our approach:

- 1 Compared with the Contract Net Protocol (CNP)
- 2 Evaluated the three task allocation heuristics
- 3 Tested the robustness of our approach

# Evaluation

## Comparison with the contract net protocol

### Experimental Setup

- Ran 40 experiments:
  - 20 experiments had a static environment
    - Arrival rate: 15 cycles (fixed)
    - Deadline window: 20 cycles (fixed)
  - 20 experiments had a dynamic environment
    - Arrival rate: varied from 15 cycles to 30 and back to 15 after every 20 tasks.
    - Deadline window: 20 cycles (fixed)
- All task structures were randomly generated
  - Maximum depth: 4
  - Maximum branching factor: 3
- Each experiment was repeated 5 times:
  - OSD approach was used the first time
  - CNP approach with 8, 10, 12, 14 agents were used in the subsequent 4 runs
- The runtime of all the experiments was 2500 cycles.

# Evaluation

## Comparison with the contract net protocol — static environments

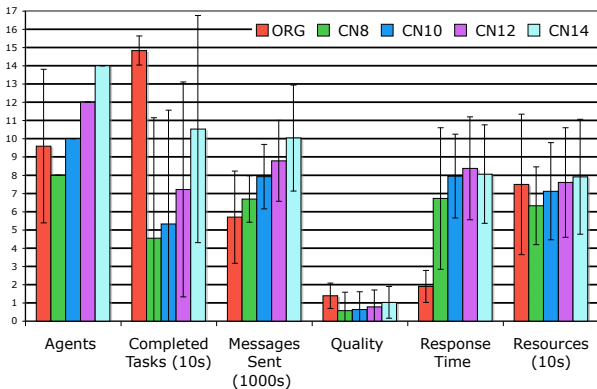


Figure: Average performance in static environments

# Evaluation

## Comparison with the contract net protocol — static environments

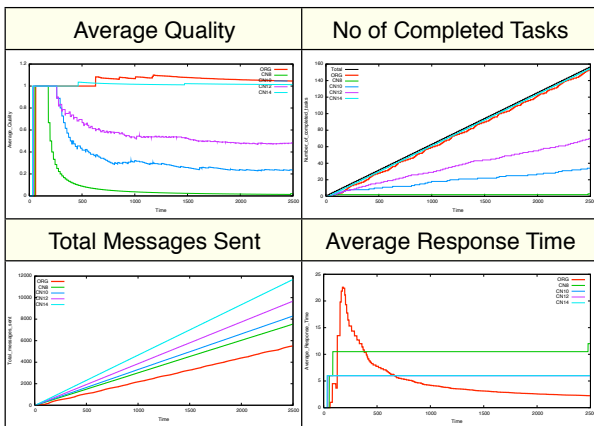


Figure: Performance of individual exp. in a static environment

# Evaluation

## Comparison with the contract net protocol — dynamic environments

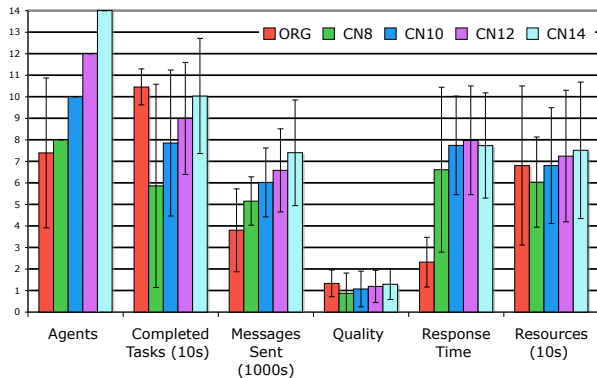


Figure: Average performance in dynamic environments

# Evaluation

Comparison with the contract net protocol — dynamic environments

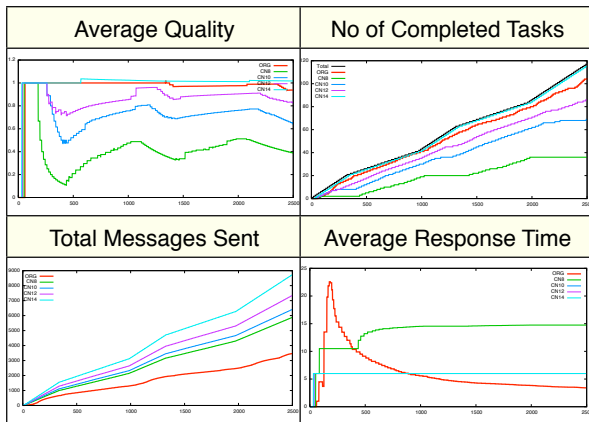


Figure: Performance of individual exp. in dynamic environment

# Evaluation

## Evaluation of the three task allocation heuristics

- Preliminary experiments demonstrated the lack of a clear winner amongst the three heuristics
  - No one allocation strategy dominates all the others
  - Different heuristics are better for different task structures and environmental conditions
    - Since each experiment starts with a different random task structure
    - we couldn't find one allocation strategy that always dominated the other

# Evaluation

## Evaluation of the three task allocation heuristics

To determine which heuristic performs the best, given

- a set of task structures
- environmental conditions and
- performance criteria

we performed a series of experiments, controlled using the following variables:

- Task structure depth: varied from 3 to 5.
- Branching factor: varied from 3 to 5.
- Probability of MIN CAF: varied from 0.0 to 1.0 in increments of 0.2.
- The arrival rate: from 10 to 40 cycles in increments of 10.
- The deadline slack: from 5 to 15 in increments of 5.

# Evaluation

## Evaluation of the three task allocation heuristics

### Experimental Setup

- Each experiment was repeated 20 times
  - A new task structure was generated each time
  - These 20 experiments formed an *experimental set*.
  - All the experiments in a set had the same values for the exogenous variables
- A static environment was used in each experiment
- Experiments in which:
  - the organization consisted of a single agent
  - the generated task structure was unsatisfiablewere removed from the set
- If more than 15 experiments removed from a set, that set was ignored
- The final evaluation was done on 673 experimental sets.

# Evaluation

## Evaluation of the three task allocation heuristics

### Tested Performance Criteria

- 1 The average number of agents used
- 2 The total number of organizational changes
- 3 The total messages sent by all the agents
- 4 The total resource cost of the organization
- 5 The number of tasks completed
- 6 The average quality accrued
- 7 The average response time of the organization
- 8 The average runtime of the tasks
- 9 The turnaround time

# Evaluation

## Evaluation of the three task allocation heuristics

- We ran the **Wilcoxon Matched-Pair Signed-Rank** tests on the experiments in each set.
- Null Hypothesis: *there is no difference between the pair of heuristics for the performance criteria under consideration*
  - Interested in the cases in which the null hypothesis can be rejected with 95% confidence ( $p < 0.05$ ).
- We noted the number of times that a heuristic performed the best or was in a group that performed statistically better than the rest.

# Evaluation

## Evaluation of the three task allocation heuristics

Criteria/Heuristic	BET	TF	MR	Rand
Number of Agents	<b>572</b>	567	100	139
No-Org-Changes	<b>641</b>	51	5	177
Total-Messages-Sent	<b>586</b>	499	13	11
Resource-Cost	346	<b>418</b>	337	66
Tasks-Completed	427	<b>560</b>	154	166
Average-Quality	367	<b>492</b>	298	339
Average-Response-Time	356	321	<b>370</b>	283
Average-Runtime	<b>543</b>	323	74	116
Average-Turnaround-Time	<b>560</b>	314	74	126

**Table:** The number of times that each heuristic performed the best or statistically equivalent to the best for each performance criteria

# Evaluation

## Evaluation of the three task allocation heuristics

Criteria/Heuristic	BET	TF	MR	Rand	BET+TF	BET+Rand	All
Number of Agents	94	88	3	7	<b>337</b>	2	85
No-Org-Changes	<b>480</b>	0	0	29	16	113	5
Total-Messages-Sent	170	85	0	2	<b>399</b>	1	5
Resource-Cost	26	100	<b>170</b>	42	167	0	15
Tasks-Completed	77	<b>197</b>	4	28	184	1	99
Average-Quality	38	147	26	104	76	0	<b>208</b>
Average-Response-Time	104	74	162	43	31	20	<b>169</b>
Average-Runtime	<b>322</b>	110	0	12	121	13	69
Average-Turnaround-Time	<b>318</b>	94	1	11	125	26	64

**Table:** Table showing the number of times that each individual heuristic performed the best and the number of times that a certain group of statistically equivalent heuristics performed the best. Only the more interesting heuristic groupings are shown. *All* shows the number of experimental sets in which there was no statistical difference between the three heuristics and a random allocation strategy

# Evaluation

## Evaluation of the three task allocation heuristics

### Interesting Observations

- For lowest number of agents in largest number of experimental sets
  - almost even split between *Balancing Execution Time* (BET) and *Topmost First* (TF) heuristics
  - Explanation:
    - BET usually results in organization with even number of agents (because BET always bifurcates the agents)
    - TF allows odd number of agents
- BET achieved the lowest number of organizational changes in the largest number of experimental sets.

# Evaluation

## Evaluation of the three task allocation heuristics

### More Interesting Observations

- Average Response Time:
  - *Minimizing Resources* (MR) heuristic performed the best
  - Explanation:
    - MR heuristic is extremely greedy
    - prefers to spawn off “small” agents that have a tiny resource footprint
    - the presence of a single small agent is sufficient to reduce the response time.
- MR heuristic is **not** the most effective heuristic when it comes to minimizing the resource-cost of the organization
  - It only outperforms a random task/resource allocation.
  - Explanation:
    - greedy nature of this heuristic
    - also only local information is used in this heuristic while (some) global information is needed to make good resource cost decisions

# Evaluation

## Robustness evaluation

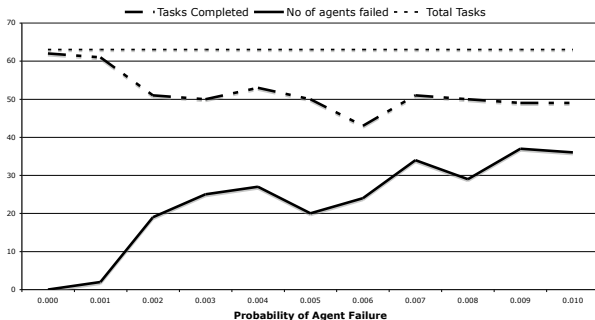


Figure: Robustness of the citizens approach

# Evaluation

## Robustness evaluation



Figure: Robustness of the citizens approach

As the probability of failure increases:

- The number of agents failing during a run also increases
- This results in a slight decrease in the number of tasks completed
  - When an agent fails, it loses whatever it was doing
  - The newly created agent must redo the work