Kansas State University
234 Nichols Hall
Manhattan, KS 66506-2302

Phone: (785) 532-6350
Fax: (785) 532-7353
http://macr.cis.ksu.edu/

Integrating Performance Factors into an Organization Model for Multiagent Systems

Christopher Zhong
czhong@k-state.edu

Scott A. DeLoach
sdeloach@k-state.edu

MACR-TR-2010-05

August 2010

1 Introduction

As computing systems become more advanced, more is being expected out of them. Two of these expectations are (1) to be able to adapt to more failures and (2) to have more integration with humans. Rising expectations for adaptive computing systems include the ability to automatically correct themselves in a fluid and dynamic environment (autonomic systems [10]). Multiagent concepts [3] are well-suited for developing adaptive systems. Russell and Norvig [20] define agents as able to perceive and act autonomously such that their actions are based on their own experiences rather than predefined knowledge. Multiagent systems exploit this behaviour to self-correct; if one agent should fail, another agent can take over.

One approach in multiagent research is to leverage organizational concepts such as agents, roles, and goals found in organizational models such as Organization Model for Adaptive Computational Systems (OMACS) [5], Organizations per Agents (OperA) [6], Organizational Model for Normative Institutions (OMNI) [7], and HarmonIA [34] to produce organization-based multiagent system. By leveraging these organizational models, a general approach to adaptivity can be achieved through task allocations. Task allocations can be handled in a general manner because these models capture the necessary information to reallocate a task should an agent fail. Various research teams have applied organizational concepts in robotics, particularly multirobot systems [2, 8, 9, 16, 26, 28, 31].

Another aspect that increases the complexity of computing systems is the inclusion of humans as part of the system. Traditionally, humans have been considered as users of a computing system; humans are not typically considered as a factor during a system's decision making process. As computing systems continue to grow, the environments in which these systems operate sometimes involve humans. By including humans as a factor in these systems' decision making process, such systems are able to increase their reliability; tasks that can no longer be completed by the system due to failures can be given to humans to complete. Interface requirements to allow such interactions is beyond the scope of this report.

One way of including humans as part of the system is to capture an abstract representation of humans in the system. Fortunately, organization-based models are well-suited to facilitate integration of humans because these models already provide a basic framework for representing humans; humans can be considered as agents. One of the open questions is the type of information about the humans that we should capture as agents. Our research focus is to capture performance information about humans to facilitate better task allocation. This leads us to the question of the types of performance information that are relevant to task allocation. In the book by Wickens *et al.* [37], they explain a large number of factors that are relevant to designing systems for human interaction. For instance, they explain the various factors that affect a human's ability to drive at night. The following are some of the factors that impact a human's ability to drive at night: the eyesight of the driver, the fatigue level of the driver,

the reaction time of the driver, the color of objects, the luminosity of objects, the current weather conditions, the ambient lighting, and the speed of the car. These factors are not exclusive to any particular task and they can be classified into three categories: human factors, task-specific factors, and environmental factors. In general, there are an enormous number of factors. When designing computing systems that include humans as part of the system, there is a significant increase to the amount of information to be handled and sometimes the complexity of these systems can spiral out of control. However, by leveraging model-driven engineering (MDE), we can develop adaptive mechanisms (commonly referred to as *models@run.time* [23]) that can manage the complexity of dealing with the enormous number of factors. One way to capture these factors is through Performance Moderator Functions (PMFs) [24], particularly human performance moderator functions (HPMFs).

Our previous work [39] was our first attempt at capturing PMFs. However, there are limitations to our initial attempt that are addressed in this report. The limitations are described in the next section (§ 1.1).

1.1 Limitations of Previous Work

This report expands on our earlier work [39] by addressing several limitations of the previous model (shown in Figure 1), which extends the OMACS [5] model. *Attributes* are description of agent properties. The *needs* relation specifies the attribute requirements for roles. The *has* function captures the attributes of agents and their associated values. The *affects* function defines the change to an agent's attribute when performing the role. The *requires* function specifies how good an agent's capability must be to perform the role.

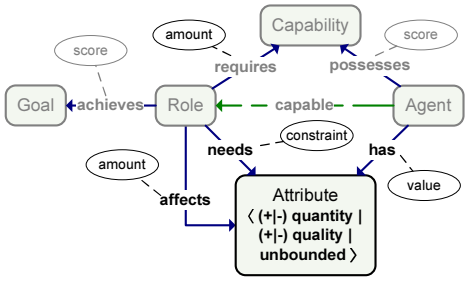
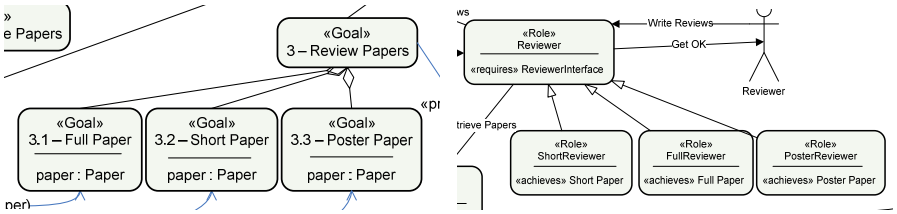


Figure 1. Previous Model

The first limitation addressed in this report is that the previous model does not provide a way to capture information about how performing the same role with different goals or goal parameters affects the attributes of agents. In our previous work, the design models (goal and role model) had to be modified to capture *workload* information for reviewing different types of papers. For example, the *review paper* goal was decomposed into three alternative goals: *full paper*, *short paper*, and *poster paper*. Additionally, the *reviewer* role was decomposed into three associated roles: *full reviewer*, *short reviewer*, and *poster reviewer*. The additional goals and roles are required because reviewing different types of paper increased the *workload* of the agent by different amounts (i.e. a

full paper increases the workload of the agent by 40%, a short paper increases the workload by 20%, and a poster paper increases the workload by 10%).



(a) Goal Model (b) Role Model

Figure 2. Previous CMS Models

The second limitation addressed in this report is the inflexibility of the *affects* function, which is defined as $affects : Role \times Attribute \rightarrow amount$. The *affects* function allows designers to specify changes that occur to an agent's attributes after performing a role. However, the *affects* function (1) does not take into account different goals or goal parameters that can be achieved by the same role (e.g. first limitation), (2) does not take into account different agents performing the same role (i.e. all agents are affected the same), and (3) does not have a built-in mechanism to change the amount at runtime. In the previous model, the three roles (*full reviewer*, *short reviewer*, and *poster reviewer*) increase the *workload* attribute of agents differently (40%, 20%, and 10% respectively). However, this change affects all agents the same, and cannot be changed at runtime except through an external mechanism.

The third limitation addressed in this report is that in our previous experiments a single *attribute*, *workload*, was used. Because of the single attribute, reorganization algorithms only had to select the agent with the lowest workload. However, with multiple attributes, a mechanism for evaluating multiple attributes in determining the most appropriate agent is required.

1.2 Overview

The rest of this paper is organized as follows: § 2 highlights the foundational work on which this work is based; § 3 describes about the Chazm Model (CzM); § 4 details the experimental setup in which we collect our data and explains the results of our experiments; § 5 provides an analysis of the time complexities of the algorithms; § 6 discusses research that are similar to the work in this report; § 7 highlights some of the limitations of CzM and ongoing work in addressing those limitations; and we conclude in § 8 by highlighting the contributions of CzM. Details about our previous work can be obtained from our earlier technical report [39].

2 Background

This section highlights three key background areas required to understand this report. § 2.1 highlights the organizational model that captures the ability of agents to carry out roles and facilitate reorganization. § 2.2 highlights the goal model that represents the goal of a system and captures the progress towards that goal. And finally, § 2.3 highlights the mathematical concepts that describe the performance and capabilities with regard to humans.

2.1 Organization Model for Adaptive Computational Systems (OMACS)

The Organization Model for Adaptive Computational Systems (OMACS) [5] is a model that captures the knowledge required to allow a team of autonomous agents to adapt to failures or changing goals. As shown in Figure 3, an OMACS *organization* consists of *goals*, *roles*, *agents*, and *capabilities*. *Goals* are high-level descriptions of what the system is supposed to accomplish [19]. *Roles* are high-level specifications on how to achieve specific goals. *Agents* are autonomous entities that can perceive and act within their environment [19]. *Capabilities* represent the notion of an agent's ability to perceive and act on its environment.

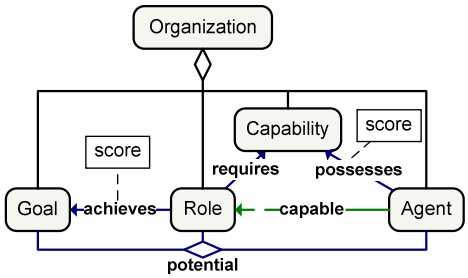


Figure 3. Organization Model

These entities are related to one another via a set of functions: *achieves*, *requires*, *possesses*, *capable*, and *potential*. The *achieves* function defines the effectiveness [0.0 .. 1.0] of a role in achieving a goal, where 0.0 means that the role is unable to achieve the goal. The *requires* function defines the capabilities that a role needs in order for agents to carry out the role's behavior. The *possesses* function defines the effectiveness [0.0 .. 1.0] of an agent's capabilities, where 0.0 means that the capability of the agent is broken or non-existent. The *capable* function specifies how well [0.0 .. 1.0] an agent can perform a role. In order to provide a score for the *capable* function, the *rcf* function is used. The *rcf* function computes the score by using the *requires* and *possesses* functions. The *rcf* function ensures that a given agent possesses the required capabilities of a given role and that the agent's capabilities are at a certain competency level. The *potential* function defines how well [0.0 .. 1.0] an agent can perform a role to achieve a goal, which is derived from the achieves and capable functions.

OMACS-based systems use the potential function to autonomously make assignments[1]. An assignment is a tuple consisting of a single goal, a single role, and a single agent. As the capabilities of agents change throughout the course of a system's execution, these changes are reflected by the possesses function, which then is also reflected by the capable function, and finally by the potential function. Should an agent reach a point where it is no longer capable of performing a role, the capable function would return a score of 0.0. This would, in turn, cause the potential function to return a score of 0.0, triggering the reorganization mechanism to replace the failed agent. In this manner, an OMACS-based system adapts to failures.

2.2 Goal Model for Dynamic Systems (GMoDS)

The set of goals captured by OMACS simply represents the current set of goals that the organization is actively pursuing. A sophisticated model is required to represent a more complex set of requirements such as optional goals, alternative goals, goal sequencing, and situational goals.

The Goal Model for Dynamic Systems (GMoDS) [15] captures a system's requirements as a single goal tree. The top-level or overall goal is decomposed into subgoals that follows the classic AND/OR goal decomposition [33]. If all subgoals must be achieved to achieve the parent goal, then the parent goal is an AND-goal. Conversely, a goal is an OR-goal if that goal is achieved when any of its subgoals are achieved. At the lowest level of the goal tree are the leaf goals, which are goals that are used by OMACS-based systems for making assignments.

In addition, GMoDS provides two extensions to the classic AND/OR goal tree: (1) a sequential ordering for achieving goals through the *precedes* relation, and (2) a *triggers* relation for specifying events that cause the creation of new goals or removal of existing goals. If goal A *precedes* goal B, then goal A must be completed first before goal B can be attempted. If goal A *triggers* goal B with event E, then while in the pursuit of goal A, goal B is created every time event E occurs.

GMoDS provides the needed ability to track progression in achieving a system's overall goal (the goal tree), the ability to dynamically adapt to variations in system parameters (the *triggers* relations), and the ability to systematically make incremental progress towards achieving the system's overall goal (the *precedes* relation).

2.3 Performance Moderator Function (PMF)

Human factors sciences is the study and understanding on the capabilities of humans. Human factors engineering is the application of the knowledge about human capabilities to design better systems. Wickens *et al.* [37] provide a number of design principles and methodologies on how using these knowledge can lead to better systems.

[1] Making assignments is synonymous with task allocation.

Performance Moderator Functions (PMFs) [24, 25] indicate the impact of internal and external factors on human performance. Examples of internal factors are the human's fatigue level, reaction time, and mental acuity. Examples of external factors are noise level, lighting, and task time. In addition, PMFs are able to capture impact of personality on performance such as emotion, cultural background, and biases. Furthermore, PMFs quantify performance differences between two humans such as intelligence, skill, and motivation. In other words, PMFs capture the relationship between performance moderators and the level of performance in the form of dose-response (or exposure-response). As the dose (or exposure) increases or decreases, PMFs indicate the change in the level of performance.

3 Chazm Model (CzM)

This section presents the Chazm Model (CzM) that bridges the gap between the human world and the robotic world. OMACS [5] is unable to explicitly capture performance factors such as *location* or *reaction time*. A number of additions and changes are required to allow OMACS to capture these performance factors. Figure 4 shows some of the additions and changes to OMACS.

There are three elements in Figure 4: rectangles, lines, and ellipses. Rectangles are entities. Lines between entities are relations; the arrows on the lines denote direction for reading purposes only. For instance, role X *requires* capability Y. The dashed lines are derived relations. Ellipses help to indicate which relations. Elements that are greyed out are elements that exists in OMACS. In CzM, there are four new entities, six new relations (two of which are functions), and changes to existing elements.

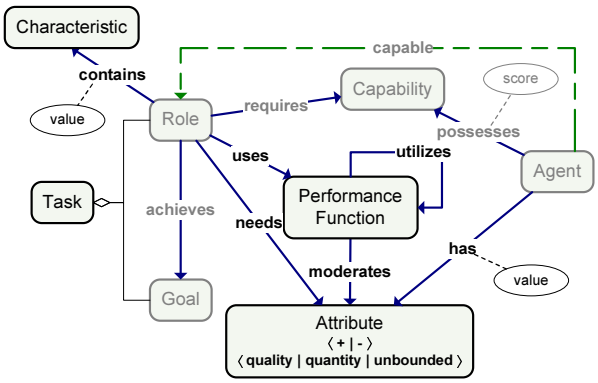


Figure 4. Chazm Model

The first new entity is called an *attribute*. An *attribute* describes a property of an agent. Currently, there are three types of attributes: quality-type, quantity-type, and unbounded-type. A quality-type attribute constrains the value to the $[0.0 \,..\, 1.0]$ range, a quantity-type attribute constrains the range to $[0.0 \,..\, \infty]$, and an unbounded-type

attribute does not constrain the values $[-\infty \; .. \; \infty]$. In addition, each type is either a positive-type or negative-type attribute, which indicates the type of scale used to measure the values in relation to one another. Some attributes can be represented as either a positive-type (the higher the value, the better) or a negative-type (the lower the value, the better). For example, consider the attributes *energy* and *fatigue*. These two attributes represent the same concept except that for energy, higher values are better, while for fatigue, lower values are better.

The purpose of the second new entity, *performance function*, is to capture the PMFs. Capturing PMFs as an entity allows user-defined PMFs to be used at runtime. For instance, two roles may have slightly different PMFs for computing the *fatigue* of agents after performing different roles because one role may require more strenuous activities than the other. PMFs are captured in CzM as functions of the form of Definition 1. The *Role*, *Agent*, and *Goal* inputs inform the PMF function which role the agent is performing to achieve the goal. The $Set\{Assignment\}$ is the relevant set of assignments for the PMF function, which can be all the assignments of the organization or a subset such as the assignments of a particular agent. Not all assignments affect the computation of PMFs; some assignments do not impact the computation of PMFs.

$$\text{pmf} : \text{Role} \times \text{Agent} \times \text{Goal} \times \text{Set}\{\text{Assignment}\} \rightarrow \text{value} \tag{1}$$

The *characteristic* entity describes a property of a role. A *characteristic* provides additional information that can be utilized by *performance functions*. For instance, the *exercise* role may contain information about the length of the exercise routine, which can be captured as the *exercise time* characteristic. The *exercise time* characteristic can be used by *performance functions* associated with the *exercise* role.

The last new entity is a *task*. A *task* is the composition of a role and a goal. The purpose of the *task* entity is purely for human understanding; computationally, a *task* does not provide any additional information other than what the associated role and goal already provide. For instance, a *delivery person* role and a *deliver package* goal comprises a *task*. In OMACS, an assignment is formally defined as $assignment : Agent \times Role \times Goal$. In CzM, we expand on the definition of an assignment by adding $assignment : Agent \times Task$.

The *has* function (Definition 2) takes in an agent and an *attribute* and returns a value consistent with the type of that attribute: quantity $[0.0 \; .. \; \infty]$, quality $[0.0 \; .. \; 1.0]$, or unbounded $[-\infty \; .. \; +\infty]$. Even though the *has* function specifies a relation between an agent's *attribute* and a single value, it is straightforward to model complex attributes such as compound attributes. A compound attribute such as *location* do not contain a value but is comprised of multiple single values. For example, the *location* attribute is typically comprised of three values: longitude, latitude, and altitude. The three values can be represented as three attributes: *longitude*, *latitude*, and *altitude*. A logical grouping of the three attributes (*longitude*, *latitude*, and *altitude*) into the *location* attribute would not provide any functional benefits. To ease the use of our model, design tools can provide logical groupings

for complex attributes such as *location*. These design tools would then translate these complex attributes for use in CzM.

$$\text{has} : \text{Agent} \times \text{Attribute} \rightarrow \text{value} \tag{2}$$

The *moderates* relation (Definition 3) specifies a relation between a *performance function* and an *attribute*. Because a *performance function* captures a PMF and a PMF computes the result for a particular *attribute*, the *moderates* relation is a many-to-one relation (i.e. a *performance function* moderates exactly one *attribute* but an *attribute* can be moderated by multiple *performance functions*). The *moderates* relation specifies the *attribute* to which the result of the PMF is applicable. For example, to capture a PMF that computes fatigue, the PMF is captured as a *performance function* that *moderates* the *fatigue* attribute.

$$\text{moderates} : \text{Performance Function} \times \text{Attribute} \tag{3}$$

The *needs* relation (Definition 4) specifies a relation between a role and an *attribute*. The purpose of the *needs* relation is to capture additional requirements for performing a role beyond just capabilities as currently used in OMACS. The *needs* and *requires* relations specify the complete requirements an agent must meet to perform a role. For example, to perform the role *run track*, the role requires the *run* capability and the *stamina* attribute because an athlete can run may lack the necessary stamina to succeed at the role.

$$\text{needs} : \text{Role} \times \text{Attribute} \tag{4}$$

The *uses* relation (Definition 5) specifies a relation between a role and a *performance function*. The purpose of the *uses* relation is to indicate which of the *attributes* associated with the role through the *needs* relation require the use of a PMF to compute the value. For instance, the *reaction time* attribute may not need a PMF because the value is obtained directly from the agent through the *has* function. But the *fatigue* attribute may need a PMF to compute the value because the result may depend on the roles. More importantly, the *uses* relation differentiates between attributes whose values are used and attributes whose values are changed as a result of performing roles. For correctness, there is one constraint on the *uses* relation. A role can only use a *performance function* if the *attribute* modified by the *performance function* is also the *attribute* needed by the role (Constraint 6).

$$\text{uses} : \text{Role} \times \text{Performance Function} \tag{5}$$

$$\forall r \in \text{Role}, f \in \text{Performance Function}, a \in \text{Attribute}$$
$$\mid (r, f) \in \text{uses} \wedge (f, a) \in \text{moderates} \Rightarrow (r, a) \in \text{needs} \tag{6}$$

The *utilizes* relation (Definition 7) specifies a relation between two *performance functions*. The reason for the *utilizes* relation is to indicate whether a *performance function* uses another *performance function* for computation. For example, to compute the overall workload, the overall workload PMF may require the auditory workload PMF, cognitive workload PMF, and visual workload PMF. There are two constraints on the *utilizes* relation: (1) the *utilizes* relation do not form a cycle (Constraint 8); (2) if a role uses a *performance function* (A), which utilizes another *performance function* (B), then the *attribute* moderated by *performance function* (B) is also needed by the role (Constraint 9). We denote the transitive closure of a relation with the $^+$ symbol.

$$\text{utilizes} : \text{Performance Function} \times \text{Performance Function} \tag{7}$$

$$\forall f \in \text{Performance Function} \mid (f, f) \notin \text{utilizes}^+ \tag{8}$$

$$\forall r \in \text{Role}, f, f' \in \text{Performance Function}, a \in \text{Attribute}$$
$$\mid (r, f) \in \text{uses} \wedge (f, f') \in \text{utilizes}^+ \wedge (f', a) \in \text{moderates}$$
$$\Rightarrow (r, a) \in \text{needs} \tag{9}$$

The *contains* function (Definition 10) takes in a role and a *characteristic* and returns a value. For instance, an exercise routine would take 30 minutes. This example could be modeled as the *exercise* role *contains* the *exercise time* characteristic with a value of 30. Then a *performance function* for computing *fatigue* for that role could use the *exercise time* characteristic for computing the new *fatigue* value of the agent after performing the *exercise* role.

$$\text{contains} : \text{Role} \times \text{Characteristic} \rightarrow \text{value} \tag{10}$$

In OMACS, to perform a role, an agent must have the required capabilities. In CzM, to perform a role, an agent must have the required capabilities and the necessary attributes. The `rcf` function defined in OMACS is defined as $rcf : Role \times Agent \rightarrow [0.0 \ .. \ 1.0]$ and the `rcf` function only evaluates the capabilities of an agent with respect

to the role. This definition is no longer sufficient due to the addition of attributes; thus, the `rcf` function is not part of CzM. Instead, we define a `goodness` function (Definition 11), that evaluates the capabilities required by agents and the attributes needed. Furthermore, the specific goal being pursued is also part of the input for the `goodness` function because the goal may contain parameters that affect how well agents may perform a particular role. The $Set\{Assignment\}$ is the relevant set of assignments for the `goodness` function, which can be all the assignments of the organization or a subset such as the assignments of a particular agent. Not all assignments affect the computation of PMFs; some assignments do not impact the computation of PMFs. The `goodness` function has one constraint (Constraint 12), where the return value must be 0.0 if the agent do not possesses a required capability, has a needed attribute, or the role does not achieve the goal.

$$\text{goodness} : \text{Role} \times \text{Agent} \times \text{Goal} \times \text{Set}\{\text{Assignment}\} \rightarrow [0.0 \ .. \ 1.0] \tag{11}$$

$$\text{goodness}(r, a, g, \phi) = \begin{cases} 0.0 & \text{if } \exists c \in (r, c) \in \text{requires} \mid (a, c) \notin \text{possesses} \\ & \quad \vee \exists n \in (r, n) \in \text{needs} \mid (a, n) \notin \text{has} \\ & \quad \vee (r, g) \notin \text{achieves} \\ [0.0 \ .. \ 1.0] & \end{cases} \tag{12}$$

Due to the removal of the `rcf` function from CzM, the *capable* function needs to be redefined. In OMACS, the *capable* function is defined as $capable : Agent \times Role \rightarrow [0.0 \ .. \ 1.0]$ with the constraint that return value is the same as the `rcf` function: $capable(a, r) = rcf(a, r)$. In CzM, the *capable* function is defined as $capable : Agent \times Role \rightarrow [true/false]$ and follows Constraint 13.

$$\text{capable}(a, r) = \begin{cases} \text{true} & \text{if } \forall c \in (r, c) \in \text{requires} \mid (a, c) \in \text{possesses} \\ & \quad \wedge \forall n \in (r, n) \in \text{needs} \mid (a, n) \in \text{has} \\ \text{false} & \end{cases} \tag{13}$$

In OMACS and our previous model, the *achieves* function was defined as $achieves : Role \times Goal \rightarrow [0.0 \ .. \ 1.0]$, which indicates how well the role achieves a specific goal. However, in CzM, *achieves* is defined as a relation between a role and a goal (Definition 14), because the `goodness` function takes in a goal as input, the score functionality is now part of the `goodness` function. This change provides greater flexibility than OMACS's inability to use goal information to compute how well agents can perform a role because CzM allows specific goals to contribute to the `goodness` score. For example, there are two agents capable of searching area A and it is preferable to select the

agent that is closer to area A.

$$\text{achieves} : \text{Role} \times \text{Goal} \tag{14}$$

In our previous model, *requires* was extended to a function defined as $requires : Role \times Capability \rightarrow amount$. However, we have discovered that this extension reduced the range of requirements that can be captured. For example, if a role requires capability A with a score of 0.8 and capability B with a score of 0.5, then only agents that possesses both capability with at least a score of 0.8 and 0.5 respectively can perform the role. But what if we also wanted the case where capability A is 0.7 and capability B is 0.7 to be acceptable? In CzM, the definition of *requires* is reverted back to what it was in OMACS, which is defined as $requires : Role \times Capability$.

4 Evaluation

We use the conference management system (CMS) [4, 38] as a basis for evaluating the usefulness of CzM versus OMACS. The CMS represents a conceptual model of the process that takes place leading up to a scientific conference, where authors submit their papers, reviewers are given papers to review, the PC chair makes decisions to accept papers, and accepted papers are sent to the printers for printing. Figure 5 shows the GMoDS model that captures the CMS process, where system goals are represented and further decomposed into subgoals. The top-level goal of the CMS is to *manage submissions*, which is decomposed into six conjunctive subgoals, which are also further decomposed into subgoals. At the bottom of the goal tree are the leaf goals, which are *collect papers*, *distribute papers*, *partition papers*, *review paper*, *collect reviews*, *make decision*, *inform declined*, *inform accepted*, *collect finals*, *master CD*, and *print proceedings*. These leaf goals are the only goals that can be pursued by agents to achieve the top-level goal. In our previous model, the *review paper* goal was decomposed into three subgoals (Figure 2a) to capture different workload amounts because of the inflexibility of the *affects* function (second limitation in § 1). In CzM, there is no need to decompose the *review paper* goal to capture the different workload amounts because the workload PMF captures the different values by looking at the parameters of the goal, one of which is the type of paper.

Figure 6 shows the role model where each leaf goal is mapped to a specific role. These roles are defined to achieve their associated leaf goal(s) and specify the capabilities required by agents in order to perform them. In our previous work, the *reviewer* role was decomposed into three roles so they could be mapped to the subgoals of the *review paper* goal, which was decomposed into three subgoals, to capture different workload amounts. However, this decomposition is no longer necessary because the *performance function* entity is able to capture the *workload* PMF, which captures the different workload amounts for different types of papers.

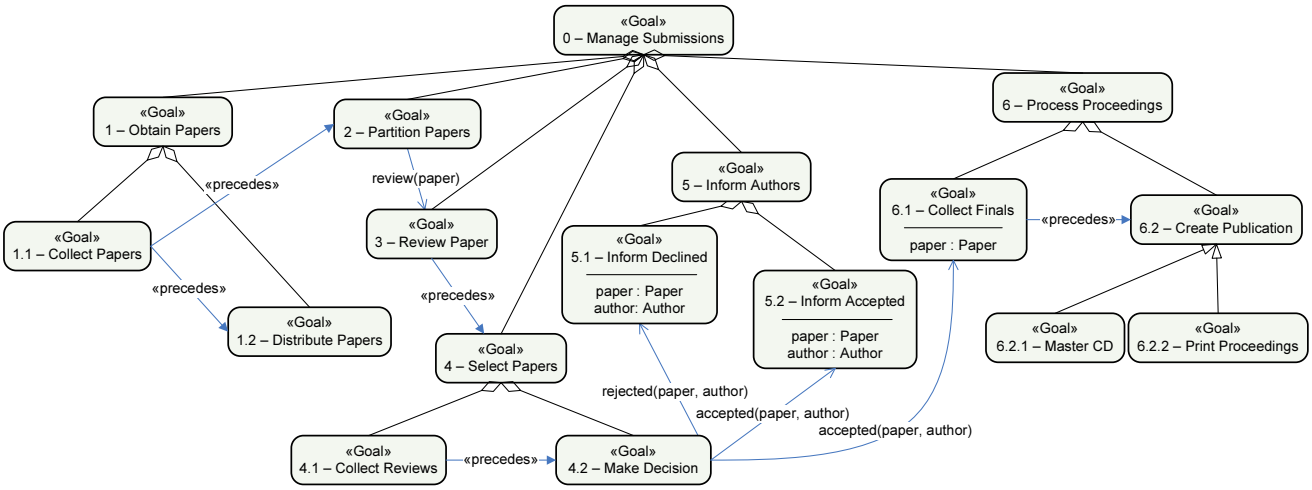


Figure 5. CMS Goal Model

The workload PMF, which is the sum of the workload values of each paper the agent is reviewing, is defined by two equations: Equation (15) and Equation (16). In Equation (15), $p.type$ refers to the type of paper. In Equation (16), $g.paper$ refers to the paper parameter of the goal g.

$$\text{workload}(p) = \begin{cases} 10 & \text{if } p.\text{type} = \text{poster} \\ 20 & \text{if } p.\text{type} = \text{short} \\ 40 & \text{if } p.\text{type} = \text{full} \end{cases} \tag{15}$$

$$\text{pmf}_{\text{workload}}(r, a, g, \phi) = \left(\sum_{(a', r', g') \in \phi} \text{workload}(g'.\text{paper}) \right) + \text{workload}(g.\text{paper}) \tag{16}$$

4.1 Experimental Setup

An experiment evaluates the usefulness of CzM by using different task allocation algorithms with a given number of reviewers, a given number of papers to review, a given number of papers to accept, and a given range for the quality of a submitted paper. The CMS experiments are set up such that for every experiment, the number of reviewers is fixed at 50, the number of papers accepted is fixed at 40, and the submitted paper quality range is from 45% to 55%. The range is kept small so as to increase the chance of a paper that is not in the top 40 to be accepted due to inaccurate reviews of that paper. Each submitted paper is given a quality that is randomly selected from the given range [45% .. 55%]. These submitted papers are ranked based on their quality; and ideally, only the top 40 papers are accepted. There are a total of 80 experiments. The first experiment starts at 40 papers to review, the second at 41 papers to review, the third at 42 papers to review, and so forth, up to the 80th experiment with

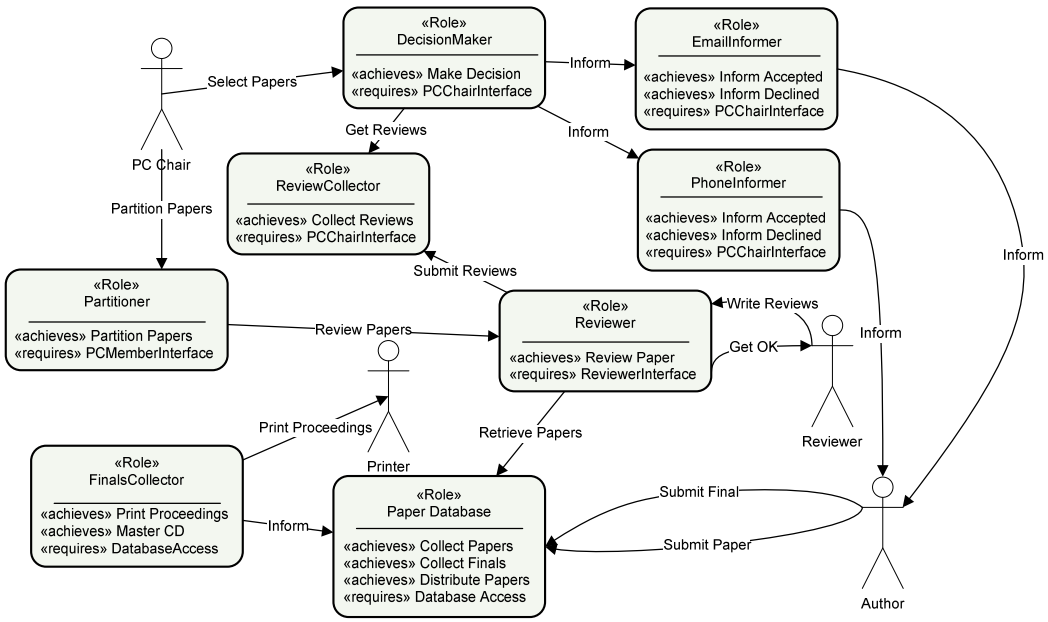


Figure 6. CMS Role Model

120 papers to review. Each submitted paper is reviewed by three reviewers. Once all reviews are in, the decision to accept or reject a paper is based on the three reviews. The purpose of an experiment is to evaluate how well a task allocation algorithm can assign these reviews to reviewers so that the set of accepted papers is as close as possible to the ideal set.

In each experiment, we compare the performance of four reorganization algorithms. Three of the algorithms are general reorganization algorithms, which means that the algorithms use only information directly available in the model, while the fourth algorithm it not entirely general. The implementation of the fourth algorithm utilizes information about the domain. Although the implementation of the fourth algorithm is not general, we believe that it is possible to generalize the implementation such that it is applicable to other domains. However, work on generalizing the implementation is in progress and is not available at this time. The three general reorganization algorithms are random, round robin, and attributes-greedy; the non-general algorithm is the attributes-enhanced. The random algorithm, which only uses information provided by the OMACS model, is the baseline algorithm. The round robin algorithm also uses only information provided by the OMACS model. The attributes-greedy and attributes-enhanced algorithms utilize information provided by CzM. These algorithms are discussed in detail in § 4.2. Although the goal model captures the entire CMS process, the focus of the experiments is on allocating the instances of the *review paper* goal.

There are three types of reviewers defined: tenured professors, assistant professors, and graduate students. All reviewers need three attributes: *incentive*, *stress*, and *workload*. These three attributes are of the same scale,

where a value of 0 means no incentive, no stress, and no workload respectively[2]. *Incentive* values are none, low, medium, and high, maximum. For computational purposes, the incentive values are mapped to 0, 30, 50, 70, and 100 respectively. *Stress* and *workload* are measured in terms of percentages and they do not have an upper-bound. These attributes determine the maximum number of papers a reviewer can review before becoming overloaded/overburdened. An overloaded reviewer will produce reviews that are less than 100% quality. As *incentive* increases, a reviewer is able to review more papers before becoming overburdened. As *stress* decreases, a reviewer is able to review more papers before becoming overburdened. Similarly, as *workload* decreases, a reviewer is able to review more papers before becoming overburdened. Table 1 shows the values of the attributes for the three types of reviewers.

	Incentive	Stress	Workload
Tenured Professors	low (30)	0%	0%
Assistant Professors	medium (50)	50%	0%
Graduate Students	low (30)	60%	0%

Table 1. Attribute Values of Agent Types

There are three types of papers defined: full paper, short paper, and poster paper. Reviewing a full paper would add 40% to a reviewer's workload; reviewing a short paper adds 20% to the workload; and reviewing a poster paper adds 10% to the workload. The quality (q_r) of a review produced by a reviewer is defined by Equation (17). For example, if a tenured professor has 6 short papers to review, the *workload* PMF will return a result of 120% workload, which results in the quality of all 6 reviews being $100 \div (120 + 30 - 0) \times 100 = 66.\overline{6}\%$.

$$\begin{aligned} &\text{workload} = \text{pmf}(\text{Role}, \text{Agent}, \text{Goal}) \\ &\text{total load} = \text{workload} + \text{stress} - \text{incentive} \\ &q_r = \begin{cases} 100 & \text{if total load} < 100, \\ \dfrac{100}{\text{total load}} \times 100 \end{cases} \end{aligned} \tag{17}$$

Incentive and stress do not change throughout the experiment. Workload is computed based on the number of papers given to a reviewer, with each paper contributing either 10%, 20%, or 40% to the reviewer's workload. The quality of a review (q_r) and the quality of the paper (q_p) determines the review score (s) as defined in Equation (18). As the review quality (q_r) approaches to 0, the range of possible review scores approaches $[0 \,..\, 100]$. For example, if $q_p = 60$ and $q_r = 80$, then $s = 60 + \text{random}([-10 \,..\, 10]) = [50 \,..\, 70]$.

[2]Determining what the incentive values means in terms of numbers is beyond the scope of this report.

$$s = \begin{cases} q_p & \text{if } q_r = 100, \\ q_p + \text{random}\left[-\dfrac{100 - q_r}{2} \;\; .. \;\; \dfrac{100 - q_r}{2}\right] \end{cases} \tag{18}$$

The distribution of the reviewers (n) remain the same for each experiment. There are approximately $n/3$ for each type of reviewers. The mathematical distribution for each type are as follows: tenured professors (r_{tp}) are $\lfloor \frac{n}{3} \rfloor$, assistant professors ($r_{ap}$) are $\lfloor \frac{n}{3} \rfloor$, and graduate students (r_{gs}) are $n - r_{tp} - r_{ap}$. Since there are 50 reviewers in our experiments, we have 16 tenured professors, 16 assistant professors, and 18 graduate students.

Because of the randomness in various aspects of the experiments such as the random paper qualities and the bounded-random error for review scores, each experiment is executed $10,000$ times to normalize the data.

4.2 Algorithms

The random algorithm randomly selects an agent capable of achieving a goal and assigns that goal to the agent. This process continues until all goals have been assigned. Because the random algorithm only cares about finding an agent that is capable of achieving a given goal, the random algorithm only uses the score of the `goodness` function to check that the agent is capable (i.e. the `goodness` function score is greater than 0.0). The `goodness` function is defined by Equation (19), which is the standard `rcf` function in OMACS, for all roles. For the purpose of this report, the results from the random algorithm act as the baseline for the other algorithms.

$$\sqrt[|\{c|(r,c)\in\text{requires}\}|]{\prod_{c\in\{c|(r,c)\in\text{requires}\}} \text{possesses}(a,c)} \tag{19}$$

The round robin algorithm assigns goals by evenly distributing the goals to capable agents. The `goodness` function for all roles is defined by Equation (19). Because not all agents are capable of achieving all the goals, the round robin algorithm keeps track of the number of goals assigned to each agent. For a given goal, the capable agent (`goodness` > 0.0) with the least number of goals currently assigned is assigned to that goal. This process continues until all goals have been assigned. Because the order of the goals impact the outcome (particularly the *review paper* goals, because a reviewer can only review a paper once), the goals are sorted to keep the *review paper* goals of the same paper together. Furthermore, the ordering of agents also affects the outcome; it could result in better or worse assignments. For example, if we have 3 reviewers (R_1 and R_2), and they can review a maximum of 1, 2 papers respectively before being overburdened, and there are 3 papers to review. If the order is R_1 and R_2, then R_1 will get 2 papers, which overburdens R_1. However, if the order is R_2 and R_1, then no agent is overburdened. Since the experiments are executed $10,000$ times, the ordering of agents is randomized to normalize the results.

The attributes-greedy algorithm uses the `goodness` function score to rank all agents for a given goal and assigns the agent with the highest score to that goal. Because CzM allows access to the workload, stress, and incentive values of an agent, the `goodness` function for the *reviewer* role is defined as computing the review quality (q_r). So the `goodness` function returns the value of q_r as defined by Equation (17). For the rest of the roles, the `goodness` function is defined by Equation (19). This process continues until all goal are assigned.

Similarly, for the attributes-enhanced algorithm, the `goodness` function for the *reviewer* role computes q_r as defined by Equation (17). In addition, the attributes-enhanced algorithm tracks the contributions (Δ) of an agent as shown in Equation (20)[3] and uses it to determine the best agent instead of just relying on the basic `goodness` score. The `goodness` function only captures the score of an agent performing the role in the current context (i.e. all the assignments of the agent). But the `goodness` function does not recomputes the scores of existing assignments, which may change if a new goal is assigned to the agent. For example, an agent is assigned one paper to review and the `goodness` function score is 1.0, which means that the agent will produce a review with 100% quality. If that agent were to be assigned a second paper to review and the `goodness` function score is 0.9, that means that the agent would produce two reviews with 90% quality. The attributes-enhanced algorithm does this "recomputation" of existing assignments by using Equation (20), which tracks each agent's overall contribution by comparing the current contribution (two 90% quality reviews) with the previous contribution (one 100% quality review).

$$\Delta_i = \texttt{goodness} * (\text{assignments} + 1) - \Delta_{i-1} \tag{20}$$

4.3 Experimental Results with Attributes

There are three types of data collected in the experiments: *score difference*, *set commonality*, and *review quality*. *Score difference* measures the sum of the accepted paper qualities versus the sum of the ideal paper qualities. *Set commonality* measures the percentage of ideal papers in the set of accepted papers. *Review quality* measures the average review quality of all reviews. In our experiments, we discovered two factors that impact the performance of algorithms: the number of assignments[4] for each agent and the quality of reviews produced by each agent. The quality of reviews factor can only be measured accurately by algorithms that have access to the workload, stress, and incentive attributes because these attributes affect the quality of a review as defined by Equation (17).

There is a relationship between the two factors; as the number of assignments increases, the quality of reviews tend to decrease. However, the importance of the two factors are not constant throughout the experiments. Because the number of reviewers are fixed at 50 for all experiments, the number of assignments is less important than quality of reviews when the number of submitted papers are low. However, as the number of submitted papers increases,

[3] As mentioned earlier, the equation is a result of analyzing the domain but we believe that the equation can be generalized.
[4] In these experiments, the number of assignments is the same as the number of papers to review.

the importance of the number of assignments also increases to a point where the number of assignments becomes more important than quality of reviews. Also, the importance of the two factors depends the measurement system. For example, the quality of reviews factor plays a more important part in the *review quality* measurement than the other two measurements. Based on the relationship between the two factors, our hypothesis is that the results are generally split into three parts: (1) the first part is where the quality of reviews factor is the dominant factor while the number of assignments factor is minor, (2) the second part is where the two factors are equally important, and (3) the third part is where the number of assignments factors is the dominant factor while the quality of reviews factor is minor.

The performance of the algorithms are linked to how the algorithms use the two factors. Although the random algorithm ignores both factors, indirectly and to a certain extent through random selection, the random algorithm utilizes the number of assignments factor. The round robin algorithm considers only the number of assignments factor and ignores the quality of reviews factor. The attributes-greedy algorithm considers only the quality of reviews factor and ignores the number of assignments factor. The attributes-enhanced algorithm considers both factors.

Figure 7 shows the results of the four algorithms as measured by the *score difference*. There are three points of interest in the graph. The first point of interest (around 52 submitted papers) is where the round robin algorithm begins to accept papers that are not in the top 40 papers because some reviewers are overburdened (i.e. producing reviews that are less than 100% quality) from being assigned too many papers to review. After this point, it is still possible to keep all reviewers from being overburdened. Both the attributes-greedy and attributes-enhanced algorithms that use CzM are able to produce assignments that keep the set of accepted papers the same as the top 40 papers. The second point of interest (around 72 submitted papers) is where the effect of overburdened reviewers becomes noticeable for the attributes-greedy and attributes-enhanced algorithms. After this point, the attributes-greedy and attributes-enhanced algorithms still produce better results than the random and round robin algorithms. At the third point of interest (around the 98 submitted papers), the performance of the attributes-greedy and round robin algorithms converge. We believe that this is the point where the number of assignments factor is just as important as the quality of reviews factor. The attributes-enhanced algorithm maintains a performance advantage over the other algorithm.

Figure 8 shows the results of the four algorithms as measured by *set commonality*. There are three points of interests in the graph. At the first point of interest (around 52 submitted papers), the round robin algorithm begins to fall off while the attributes-greedy and attributes-enhanced algorithms still produce assignments that keep all reviewers from being overburdened because there are 50 reviewers and about 52 papers to be reviewed three times. The second point of interest (around 70 submitted papers) is where the effect of overburdened reviewers

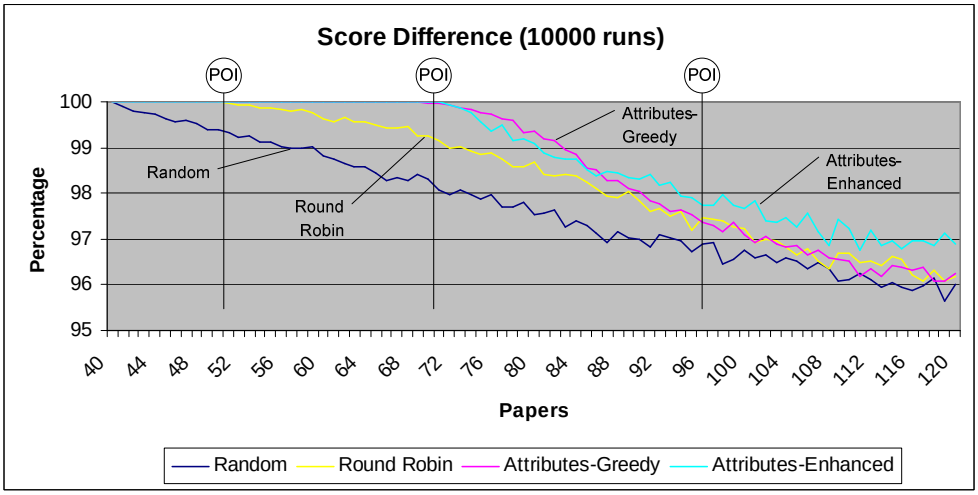


Figure 7. Score Difference Graph

becomes noticeable for the attributes-greedy and attributes-enhanced algorithms. Still, the attributes-greedy and attributes-enhanced algorithms maintain an advantage over the random and round robin algorithms. At the third point interest (around 90 submitted papers), the performance of the attributes-greedy algorithm converges to the performance of the round robin algorithm but the attributes-enhanced algorithm still maintains an advantage over the other algorithms. The performance advantage of the attributes-greedy algorithm over the round robin algorithm is gone because, around the third point of interest, the number of assignments factor is just as important as quality of reviews factor. However, the attributes-enhanced algorithm still maintains an advantage over the other algorithms because it uses both factors (number of assignments and quality of reviews) in the form of overall contributions (Equation (20)).

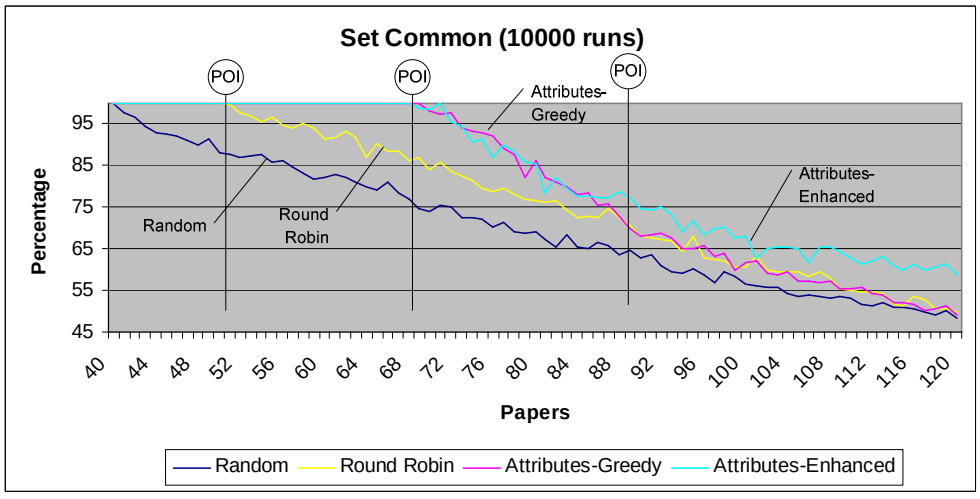


Figure 8. Set Commonality Graph

Figure 9 shows the results of the four algorithm as measured by *review quality*. Again, there are three points of interests in the graph. The first point of interest (around 52 submitted papers) is where the round robin algorithm begins to produce assignments that result in reviews that are less than 100% quality. The attributes-greedy and attributes-enhanced algorithms still produce assignments that result in 100% quality reviews. The second point of interest (around 72 submitted papers) is where the effect of overburdened reviewers becomes noticeable for the attributes-greedy and attributes-enhanced algorithms. Up to the third point of interest (around 88 submitted papers), the attributes-greedy and attributes-enhanced algorithms maintain a noticeable advantage over the random and round robin algorithms. However, after this point, the attributes-greedy begins to perform worse than the round robin algorithm, while the performance advantage of the attributes-enhanced algorithm over the round robin algorithm becomes smaller. Again, the attributes-enhanced algorithm maintains an advantage over the other algorithms because it uses both factors (number of assignments and quality of reviews) in the form of overall contributions (Equation (20)). Because the attributes-greedy algorithm only considers review quality and the number of assignments becomes more important than quality of reviews, the attributes-greedy algorithm begins to perform worse than the round robin algorithm.

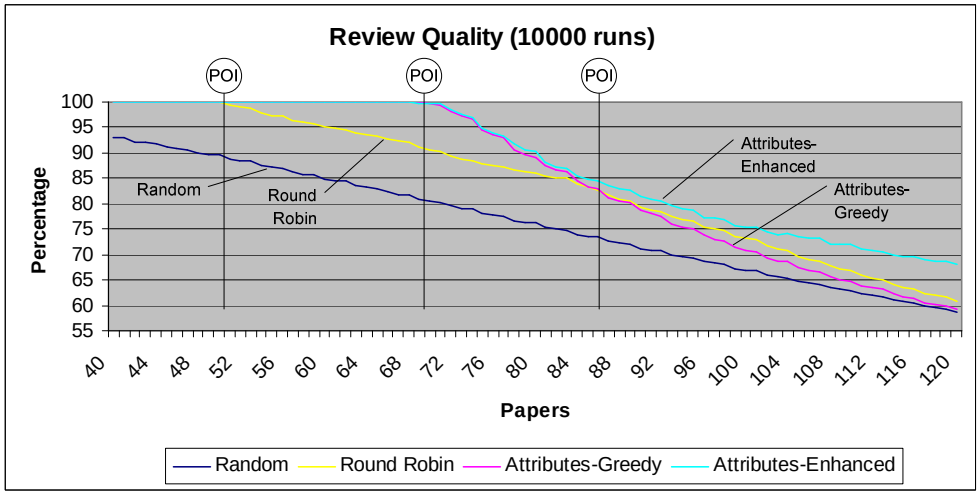


Figure 9. Review Quality Graph

Based on the three graphs, the attributes-enhanced algorithm is able to produce better assignments due to the use of the two factors: number of assignments and quality of reviews. On the other hand, the attributes-greedy algorithm, which uses only the quality of reviews factor, is only able to maintain an advantage over the round robin algorithm for the early portions of the graph. The performance of the attributes-greedy algorithm either converges to the performance of the round robin algorithm or performs worse than the round robin algorithm because the number of assignments factor, which is ignored by the attributes-greedy algorithm, becomes just as important or

more important than the quality of reviews factor. This leads us to conclude that just having the information available is not sufficient. The information needs to be used in the proper context, which leads us to the next section that explores the case where the scores of capabilities matter.

4.4 Experimental Results with Capabilities and Attributes

The setup of CMS experiments in § 4.1 assumes that every agent (in this case, the reviewers) are equally capable in their reviewing abilities. However, this is not generally the case. The setup of the experiments in this section uses different scores for the review capability to reflect the ability of the three reviewer types. Table 2 shows the values of the attributes and the reviewing capability of the three types of reviewers, where a value of 1.0 means 100%.

	Incentive	Stress	Workload	Review Ability
Tenured Professors	low (30)	0%	0%	1.0 (100%)
Assistant Professors	medium (50)	50%	0%	0.8 (80%)
Graduate Students	low (30)	60%	0%	0.6 (60%)

Table 2. Attribute and Capability Values of Agent Types

Since the capability scores are now different, a greedy algorithm would not return the same assignments as the round robin algorithm as the experimental setup in § 4.1. The `goodness` function for the *reviewer* role for the greedy and round robin algorithms is defined by Equation (19). The greedy algorithm makes assignment decisions based on Equation (21). If the numerator is always the same value, then the greedy algorithm is equivalent to the round robin algorithm when making assignments.

$$\frac{\text{goodness}(r, a, g)}{\text{number of assignments}} \tag{21}$$

In order to incorporate capability scores in the experiments, a minor change to Equation (17) is required. Equation (22) defines how review quality is computed that uses the score of an agent's reviewing capability. In Equation (17), max load is always 100. In Equation (22), max load is multiplied by the score of the agent's reviewing capability, which ranges [0.0 .. 1.0]. And thus, tenured professors have 100 max load, assistant professors have 80 max load, and graduate students have 60 max load.

$$\begin{aligned} &\text{max load} = 100 \times \text{reviewing capability} \\ &\text{total load} = \text{workload} + \text{stress} - \text{incentive} \\ &q_r = \begin{cases} 100 & \text{if total load} < \text{max load}, \\ \dfrac{\text{max load}}{\text{total load}} \times 100 & \end{cases} \end{aligned} \tag{22}$$

Similar to § 4.3, there are two factors to consider in the experiments: number of assignments and quality of reviews. The greedy algorithm considers the number of assignments factor and, in a limited degree, considers the quality of reviews by using just the capability score while ignoring the attributes. Likewise, the `goodness` function for the attributes-greedy and attributes-enhanced algorithms were changed to match Equation (22). All other aspects of the experiments remain the same as § 4.1.

Figure 10 shows the *score difference* graph. The greedy and round robin algorithms drop off immediately at the beginning of the graph although the greedy algorithm maintains an advantage over the round robin algorithm. The advantage of the greedy algorithm over the round robin algorithm is perhaps due to use of the two factors. Although the greedy algorithm considers both factors, the quality of reviews factor is incorrect as the `goodness` function for the greedy algorithm ignores attributes, which results in poorer performance when compared to the attributes-greedy and attributes-enhanced algorithms. The attributes-greedy and attributes-enhanced algorithms still produce assignments that result in 100% quality reviews. At the first point of interest (around 58 submitted papers), the attributes-greedy and attributes-enhanced algorithms are no longer able to keep some reviewers from being overburdened. However, the attributes-greedy and attributes-enhanced algorithms still maintain an advantage over the greedy and round robin algorithms. At the second point of interest (around 70 submitted papers), the attributes-greedy algorithm begins to perform worse than the greedy algorithm probably because the number of assignments becomes a more important factor than the quality of reviews factor. The attributes-enhanced algorithm still maintains a small advantage over the other algorithms because it considers both factors. The round robin algorithm barely maintains an advantage over the random algorithm because it ignores the score of an agent's reviewing capability, which matters in these experiments. At the third point of interest (around 106 submitted papers), the performance of all algorithms seem to converge probably because situation is bad enough that any algorithm would perform just as well.

Figure 11 shows the *set commonality* graph. Again, the round robin and greedy algorithms drop off immediately at the beginning of the graph but the greedy algorithm, which considers both factors, maintains an advantage over the round robin algorithm. At the first point of interest (around 56 submitted papers), the attributes-greedy and attributes-enhanced algorithms are not able to keep some reviewers from being overburdened but they still maintain an advantage over the other algorithms. At the second point of interest (around 66 submitted papers), the greedy algorithm almost catches up to the attributes-enhanced algorithm and the attributes-greedy algorithm begins to perform worse than the greedy algorithm. This change is probably due the number of assignments factor becoming the dominant factor. At the third point of interest (around 104 submitted papers), the performance of all algorithms seem to converge probably because the situation is severe enough that any algorithm would perform just as good. Although, the attributes-enhanced algorithm seem to be slightly better the other algorithms.

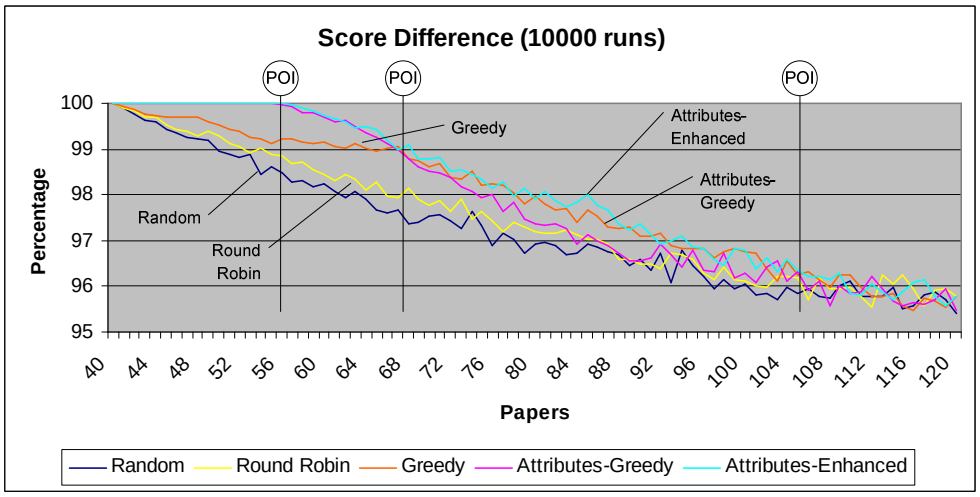


Figure 10. Score Difference Graph

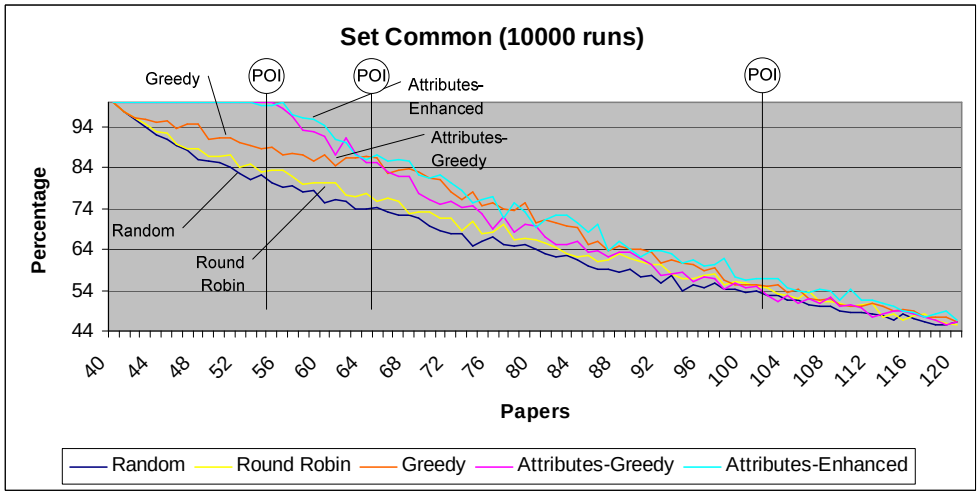


Figure 11. Set Commonality Graph

Figure 12 shows the *review quality* graph. Again, the round robin and greedy algorithms start out worse than the attributes-greedy and attributes-enhanced algorithms. However, the greedy algorithm, which considers both factors, maintains an advantage over the round robin algorithm. At the first point of interest (around 58 submitted papers), the attributes-greedy and attributes-enhanced algorithms are no longer able to keep some reviewers from being overburdened but they still maintain an advantage over the other algorithms. At the second point of interest (around 68 submitted papers), the greedy algorithm surpasses the attributes-greedy algorithm because the number of assignments factor becomes just as important as the quality of review factor. The attributes-enhanced algorithm still maintains a slight advantage over the greedy algorithm because it considers both factors properly. At the third point of interest (around 80 submitted papers), the number of assignments factor becomes the dominant factor.

This results in the attributes-greedy algorithm performing worse than the round robin algorithm. At the fourth point of interest (around 106 submitted papers), the attributes-greedy algorithm performs worse than the random algorithm. This is probably due to the overwhelming importance of the number of assignments factor over the quality of reviews factor. Also, the performance of the greedy algorithm is surpassed by the round robin algorithm probably because the greedy algorithm incorrectly considers the two factors. The attributes-enhanced algorithm maintains a slight advantage over the round robin algorithm because it considers the two factors properly.

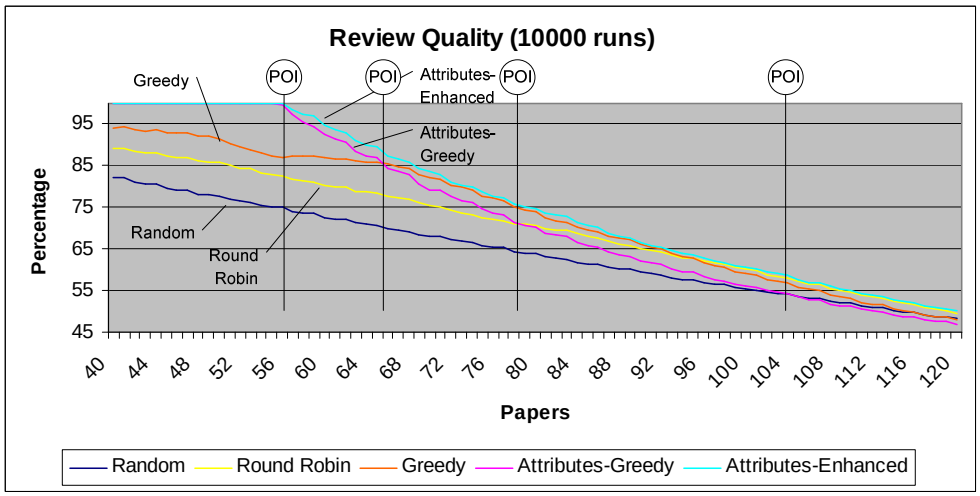


Figure 12. Review Quality Graph

With the introduction of attributes, algorithms that take advantage of this extra information are able to perform better. However, the caveat of this extra information is that it needs to be considered in the proper context as the attributes-greedy algorithm demonstrates through the three graphs. Also, just using the extra information but ignoring other existing information such as number of assignments can also lead to a decrease in performance as seen in the three graphs when the greedy algorithm surpasses the attributes-greedy algorithm.

5 Time Complexity

This section analyzes the time complexity of the four algorithms in § 4.2. None of the algorithms reshuffles current assigned goals, the algorithms only assign goals that have not been assigned.

Figure 13 shows the pseudo code for the random algorithm. The random algorithm computes a set of goals that has not been assigned yet (g') from the given set of goals (g) and it iterates through every goal from the set of unassigned goals (g'). From a goal (g_i), the set of roles (W_r) that can achieve g_i is obtained via the `achieves()` function. The set of agents (W_a) is copied to a' and an agent (a_i) is randomly picked from a'; the `random()` function randomly picks an element without replacement (i.e. a_i is removed from a'). An iteration occurs over

every role (r_i) to determine viable assignments (α), which are computed using the `goodness()` function. Once α is computed, an assignment (α_i) is randomly selected. If there are no viable assignments for a_i, another agent is randomly picked to repeat the process until either a viable assignment is found or all agents have been picked and there are no viable assignments for g_i.

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), β ← ∅
  for each g_i from g' do
    a' ← W_a, W_r ← achieves(g_i)
    repeat
      a_i ← random(a'), α ← ∅
      for each r_i from W_r do
        if goodness(r_i, a_i, g_i) > 0 then
          α ← α ∪ ⟨r_i, a_i, g_i⟩
        end if
      end for
      α_i ← random(α)
    until α_i ≠ ∅ or a' = ∅
    β ← β ∪ α_i
  end for
  return β
```

Figure 13. Pseudo Code – Random Algorithm

Proof. The time complexity of the random algorithm is $O(g \times a \times r \times c)$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, and c be the number of capabilities in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + g)$.

Duplicating the set of agents takes $\Theta(a)$ time because it iterates through each agent and since this duplication occurs inside the first loop, the time complexity so far is $O(g + (g \times a))$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop terminates when either a viable assignment is found or when a' is $\emptyset$. Randomly picking an agent through the `random()` function takes constant time. In the best case, the first randomly picked agent is capable ($\Theta(1)$). In the worst case, no agents are capable or the only capable agent is the last one to be picked

($\Theta(a)$). So, the time it takes for this loop is $O(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times (a + a)))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times (a + (a \times r))))$.

The `goodness()` function takes $O(c)$ because every capability required by the role has to be checked, which is c in the worst case as all the capabilities in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times (a + (a \times (r \times c)))))$.

Since the remaining pseudo code is constant time, the time complexity of the random algorithm is $O(g + g \times (a + a \times r \times c))$, or $O(g \times a \times r \times c)$. □

Figure 14 shows the pseudo code for the round robin algorithm. The round robin algorithm starts by computing the set of unassigned goals (g') from the given goals (W_g). In addition, it also sorts the unassigned goals so *review goals* for the same paper are grouped together. Next, the algorithm iterates through each goal (g_i) and obtains the roles (W_r) that can achieve g_i. It then iterates through each agent (a_i) to determine the best role for the a_i and g_i, which requires iterating through each role (r_i) and selecting the role with the best `goodness` score. If a_i is capable (i.e. $\beta \neq \emptyset$), the number of assignments (including those that the algorithm has already decided to assign) for a_i is obtained from the `assignments()` function. If the previously stored result (γ) is $\emptyset$ or the number of assignments is less than the one from γ, this agent (a_i) becomes the one that will be assigned to the goal (g_i). Otherwise, the algorithm moves to the next agent and repeats the process until all agents have been checked. The agent ($\gamma.a$) with least number of assignments is assigned to the goal (g_i). The algorithm moves to the next goal and repeats the process until all goals are processed.

Proof. The time complexity of the round robin algorithm is $O(g \times a \times r \times c)$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, and c be the number of capabilities in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time. Next, the unassigned goals are sorted, which takes $O(g \times \log g)$. The time complexity so far is $O(g + (g \times \log g))$.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + (g \times \log g) + g)$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

```
function reorganize(W_g, W_a)
  g' ← sort(unassigned(W_g)), λ ← ∅
  for each g_i from g' do
    W_r ← achieves(g_i), γ ← ∅
    for each a_i from W_a do
      β ← ∅
      for each r_i from W_r do
        α ← goodness(r_i, a_i, g_i)
        if α > 0 and (β = ∅ or α > β.α) then
          β ← ⟨r_i, a_i, g_i, α⟩
        end if
      end for
      if β ≠ ∅ then
        δ ← assignments(β.a)
        if γ = ∅ or δ < γ.δ then
          γ ← ⟨β.r, β.a, β.g, δ⟩
        end if
      end if
    end for
    λ ← λ ∪ ⟨γ.r, γ.a, γ.g⟩
  end for
  return λ
```

Figure 14. Pseudo Code – Round Robin Algorithm

The second loop iterates through every agent from W_a and thus, the time complexity for this loop is $\Theta(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times \log g) + (g \times a))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g+(g\times\log g)+(g\times(a\times r)))$.

The `goodness()` function takes $O(c)$ because every capability required by the role has to be checked, which is c in the worst case as all the capabilities in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times \log g) + (g \times (a \times (r \times c))))$.

Since the remaining pseudo code is constant time, the time complexity of the round robin algorithm is $O(g + g \times \log g + g \times a \times r \times c)$, or $O(g \times a \times r \times c)$. □

Figure 15 show the pseudo code for the greedy algorithm. The greedy algorithm starts by computing the set of unassigned goals (g') from the given goals (W_g). Next, the algorithm iterates through each goal (g_i) and obtains the roles (W_r) that can achieve the g_i. It then iterates through each agent (a_i) to determine the best role for the a_i and g_i, which requires iterating through each role (r_i) and selecting the role with the best `goodness` score. If a_i is capable (i.e. $\beta \neq \emptyset$), the δ is compared to the δ of the previously saved result (γ). If γ is $\emptyset$ or the δ is greater than the δ of γ, this agent (a_i) becomes the one that will be assigned to the goal (g_i). The algorithm moves to the next agent and repeats the process until all agents have been checked. The agent ($\gamma.a$) with the highest score is assigned

to the goal (g_i). The algorithm moves to the next goal and repeats the process until all goals are processed.

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), δ ← ∅
  for each g_i from g' do
    W_r ← achieves(g_i), γ ← ∅
    for each a_i from W_a do
      β ← ∅
      for each r_i from W_r do
        α ← goodness(r_i, a_i, g_i)
        if α > 0 and (β = ∅ or α > β.α) then
          β ← ⟨r_i, a_i, g_i, α⟩
        end if
      end for
      if β ≠ ∅ then
        δ ← β.α ÷ assignments(β.a)
        if γ = ∅ or δ > γ.δ then
          γ ← ⟨β.r, β.a, β.g, β.δ⟩
        end if
      end if
    end for
    δ ← δ ∪ ⟨γ.r, γ.a, γ.g⟩
  end for
  return δ
```

Figure 15. Pseudo Code – Greedy Algorithm

Proof. The time complexity of the greedy algorithm is $O(g \times a \times r \times c)$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, and c be the number of capabilities in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + g)$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop iterates through every agent from W_a and thus, the time complexity for this loop is $\Theta(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times a))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times (a \times r)))$.

The `goodness()` function takes $O(c)$ because every capability required by the role has to be checked, which is

c in the worst case as all the capabilities in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times (a \times (r \times c))))$.

Since the remaining pseudo code is constant time, the time complexity of the greedy algorithm is $O(g + g \times a \times r \times c)$, or $O(g \times a \times r \times c)$. □

Figure 16 shows the pseudo code for the attributes-greedy algorithm. The attributes-greedy algorithm starts by computing the set of unassigned goals (g') from the given goals (W_g). Next, the algorithm iterates through each goal (g_i) and obtains the roles (W_r) that can achieve the g_i. It then iterates through each agent (a_i) to determine the best role for the a_i and g_i, which requires iterating through each role (r_i) and selecting the role with the best `goodness` score. If a_i is capable (i.e. $\beta \neq \emptyset$), the score is compared to the score of the previously saved result (γ). If γ is $\emptyset$ or the score is greater than the score of γ, this agent (a_i) becomes the one that will be assigned to the goal (g_i). The algorithm moves to the next agent and repeats the process until all agents have been checked. The agent ($\gamma.a$) with the highest score is assigned to the goal (g_i). The algorithm moves to the next goal and repeats the process until all goals are processed.

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), δ ← ∅
  for each g_i from g' do
    W_r ← achieves(g_i), γ ← ∅
    for each a_i from W_a do
      β ← ∅
      for each r_i from W_r do
        α ← goodness(r_i, a_i, g_i)
        if α > 0 and (β = ∅ or α > β.α) then
          β ← ⟨r_i, a_i, g_i, α⟩
        end if
      end for
      if β ≠ ∅ and (γ = ∅ or β.α > γ.α) then
        γ ← ⟨β.r, β.a, β.g, β.α⟩
      end if
    end for
    δ ← δ ∪ ⟨γ.r, γ.a, γ.g⟩
  end for
  return δ
```

Figure 16. Pseudo Code – Attributes-Greedy Algorithm

Proof. The time complexity of the attributes-greedy algorithm is $O(g \times a \times r \times (c + n))$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, c be the number of capabilities in the organization, and n be the number of attributes in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + g)$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop iterates through every agent from W_a and thus, the time complexity for this loop is $\Theta(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times a))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times (a \times r)))$.

The `goodness()` function takes $O(c + n)$ because every capability and attribute required by the role has to be checked, which is $c + n$ in the worst case as all the capabilities and attributes in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is $O(g + (g \times (a \times (r \times (c + n)))))$.

Since the remaining pseudo code is constant time, the time complexity of the attributes-greedy algorithm is $O(g + g \times a \times r \times (c + n))$, or $O(g \times a \times r \times (c + n))$. □

Figure 17 shows the pseudo code for the attributes-enhanced algorithm. The attributes-enhanced algorithm starts by computing the set of unassigned goals (g') from the given goals (W_g). Next, the algorithm iterates through each goal (g_i) and obtains the roles (W_r) that can achieve g_i. It then iterates through each agent (a_i) to determine the best role for the a_i and g_i, which requires iterating through each role (r_i) and selecting the role with the best `goodness` score. If a_i is capable (i.e. $\beta \neq \emptyset$) and if the `goodness` score is 1.0, the contribution score is equal to the `goodness` score. However, if the `goodness` score is less than 1.0, the contribution score is computed as shown in Equation (20). Then the contribution score is compared to the contribution score of the previously saved result (γ). If γ is $\emptyset$ or the contribution score is greater than one from γ, this agent (a_i) becomes the one that will be assigned to the goal (g_i). The algorithm moves to the next agent and repeats the process until all agents have been checked. The agent ($\gamma.a$) with the highest contribution score is assigned to the goal (g_i). The algorithm moves to the next goal and repeats the process until all goals are processed.

Proof. The time complexity of the attributes-enhanced algorithm is $O(g \times a \times r \times (c + n))$.

Let g be the number of goals for the input W_g, a be the number of agents for the input W_a, r be the number of roles in the organization, c be the number of capabilities in the organization, and n be the number of attributes

```
function reorganize(W_g, W_a)
  g' ← unassigned(W_g), λ ← ∅
  for each g_i from g' do
    W_r ← achieves(g_i), γ ← ∅
    for each a_i from W_a do
      β ← ∅
      for each r_i from W_r do
        α ← goodness(r_i, a_i, g_i)
        if α > 0 and (β = ∅ or α > β.α) then
          β ← ⟨r_i, a_i, g_i, α⟩
        end if
      end for
      if β ≠ ∅ then
        δ ← β.α
        if δ < 1 then
          δ ← δ × (assignments(β.a) + 1) − previous(β.a)
        end if
        if δ > γ.α then
          γ ← ⟨β.r, β.a, β.g, δ⟩
        end if
      end if
    end for
    λ ← λ ∪ ⟨γ.r, γ.a, γ.g⟩
  end for
  return λ
```

Figure 17. Pseudo Code – Attributes-Enhanced Algorithm

in the organization.

Computing the set of unassigned goals requires checking each goal to determine if it is assigned. Thus, the `unassigned()` function takes $\Theta(g)$ time.

The first loop iterates through all unassigned goals. In the worst case, all goals from W_g are not assigned and the loop takes $O(g)$ time. The time complexity so far is $O(g + g)$.

The `achieves()` function, which returns a set of roles (W_r), takes constant time, although the cardinality of W_r varies. In the best case, every goal is achieved by 1 role; however, in the worst case, every goal can be achieved by r roles. The time complexity remains unchanged.

The second loop iterates through every agent from W_a and thus, the time complexity for this loop is $\Theta(a)$. Since the second loop occurs inside the first loop, the time complexity so far is $O(g + (g \times a))$.

The third loop iterates through every role from W_r, which has a worst case cardinality of r. Thus, the loop takes $O(r)$. Since the third loops occurs inside the second loop, the time complexity so far is $O(g + (g \times (a \times r)))$.

The `goodness()` function takes $O(c + n)$ because every capability and attribute required by the role has to be checked, which is $c + n$ in the worst case as all the capabilities and attributes in the organization could be required by a role. Since the `goodness()` function is called inside the third loop, the time complexity so far is

$O(g + (g \times (a \times (r \times (c + n)))))$.

Since the remaining pseudo code is constant time, the time complexity of the attributes-enhanced algorithm is $O(g + g \times a \times r \times (c + n))$, or $O(g \times a \times r \times (c + n))$[5]. □

The time complexity of the random, round robin, and greedy algorithms is $O(g \times a \times r \times c)$, while the time complexity of the attributes-greedy and attributes-enhanced algorithms is $O(g \times a \times r \times (c + n))$. Introducing attributes to CzM increases the time complexity of the `goodness` function by an expected amount. Although the time complexity of the attributes-enhanced algorithm is not affected by Equation (20), we expect that a general approach will increase the time complexity but we are unsure of how much.

6 Related Work

In multirobot systems, Parker [17] states that there are three approaches to task allocation: bioinspired, organizational, and knowledge-based.

In bioinspired approaches, observations on animal behaviors are applied to multi-robot systems. The robots are typically homogeneous and exists in large numbers (i.e., swarms). Individually, each robot possesses very limited capability. However, when they are grouped together in large numbers and interact as a *collective*, a group-level intelligent behaviour emerges. Because it is assumed that every robot has the ability to sense the relevant information in their environment (i.e., *stigmergy*), communication among robots is reduced significantly. Even in the situations when *stigmergy* is not available, robots only need to broadcast minimal information about their state or environment. No task allocation communication occurs. A task is allocated when a robot senses that a task needs to be performed and proceeds to perform it. Should a robot fail when performing a task, another robot simply replaces the failed robot. By following this basic behavior, a *collective* of these robots can achieve the overall system goal. Examples of bioinspired systems are [1, 11, 12, 13, 14, 18, 27, 29, 30].

Organizational approaches utilize organizational theory for task allocation in multi-robot systems. Robots in these systems are typically heterogeneous as they can possess varying capabilities. Within the organizational approaches are two approaches to task allocation: role-based and market-based. Role-based approaches employ the use of roles to divide up the work that needs to be done. A role can consists of one or more tasks that need to be completed. Robots then select, or are assigned, the roles that are best suited for them based on their capabilities. Examples of role-based approaches are [26, 28]. Market-based approaches use principles and theories of market economies to allow each robot to negotiate with other robots on which tasks they should perform. Robots communicate with each other on the cost or utility of tasks and the tasks are then assigned to the robot with the

[5]The implementation of Equation (20) is constant time. However, we believe that the same results can be achieved in a generalized way but we uncertain of the time complexity for the generalized way.

lowest cost or highest utility. Examples of market-based approaches are M+ [2], Sold! [9], RACHNA [36] and others [21, 22, 40].

Knowledge-based approaches share ontological or semantic information among the robots as the basis for task allocation. Because the robots in knowledge-based approaches are typically heterogeneous, knowledge-based approaches tend to focus on sharing information about the robots' capabilities with one another. Various techniques have been applied to sharing the information. COBOS [8] uses a *task suitability matrix* for task allocation. The *task suitability matrix* maintains the *suitability* of each robot for each task. The *suitability* of a robot is computed based on the *intrinsic* abilities of the robot to the task and the *extrinsic* factors of the task. In ALLIANCE [16], every robot contains a model of every other robot. This model contains information about the performance of the robots and tasks-related information. These models are populated through observation. Robots then use these models to determine which task(s) to perform. ASyMTRe [31] and ASyMTRe-D [32] captures the capabilities of robots as *schemas*. *Schemas* are building-block type capabilities with inputs and outputs that are semantic information types. By matching the inputs and outputs of *schemas*, a valid flow can be formed through various *schemas* in completing tasks. Vig and Adams [35, 36] provide a heuristic-based task allocation algorithm for coalition formation, where each robot contains a capability vector. These vectors are then used in computing an optimal coalition formation.

7 Future Work

The current setup of the CMS experiments focuses on one aspect of the CMS process: the reviewing of papers. In our experiments, only the reviewing process is affected by the agent's attributes; the stress, incentive, and workload attributes determines how many papers an agent can review before becoming overburdened and these attributes directly impact the quality of reviews produced by agents. What if the *make decision* goal, which captures the process of accepting papers, is also affected by the attributes of agents? Is the attributes-enhanced algorithm still the better algorithm (with respect to the other algorithms in this report) for making assignments with respect to the *make decision* goal? Or would a different algorithm be required? Our hypothesis is that the attributes-enhanced would still be the better algorithm and we are conducting additional experiments to show it.

Although the attributes-enhanced algorithm performs the best, it is not implemented in a general way as it contains Equation (20), which is domain-specific. Equation (20) can be generalized without modifying CzM; what we would do is to compute the change in overall score, which is the delta (Δ). The Δ can be computed by comparing the overall score of the current set of assignments for a particular agent and the overall score of the current set of assignments plus the new assignment. One way to compute the overall score is sum up the `goodness` score for each assignment. We hypothesize that a higher Δ is better, and so selecting the agent with the highest

Δ would be the best choice. However, we are currently investigating if using a sum to compute overall score is applicable across multiple domains or do some domains require a different way to compute overall score.

By capturing PMFs in CzM, we have an opportunity to predict the effect of performing various tasks. For instance, PMFs can be used to prevent situations in which the only agent capable of completing task A is assigned to task B, thus rendering the task A unachievable; one case that this situation can happen is because once the agent completes task B, that agent would be too tired to complete task A. This can create a ripple effect that eventually leads to the system failing to achieve the top-level goal as there are no longer agents left capable of completing task A. Using PMFs in a predictive manner is something that will be explored once CzM is in a stable state. However, prediction is more effective if we know the whole picture (in this case, all the goals that would be in the system). Unfortunately, this is generally not the case; goal models that react to events by creating new goals limit the ability to use PMFs to prevent system failures as the complete set of goals is not available ahead of time. One way to tackle this issue would be to use probability and set the probability threshold to allow some use of PMFs in prediction.

8 Conclusions

In conclusion, CzM extends OMACS to be able to capture information about the state of agents through the *attribute* entity and PMFs through the *performance function* entity. The relationships (*has*, *moderates*, *needs*, *uses*, *utilized*, and *contains*) provide the necessary structure so that task allocation algorithms can utilize this new information. The results shown in § 4.3 and § 4.4 indicate that general algorithms can benefit from using this information in the task allocation process. However, as shown in graphs in § 4.3 and § 4.4, having more information is not necessarily beneficial and can sometimes be detrimental as this information needs to be considered in the proper context. In addition, introducing *attributes* only increase the time complexity of the `goodness` function from $O(c)$ to $O(c + n)$. This is but the first step towards integrating humans as part of the system; there are other areas that needs to be explored. One area is the interface requirements to facilitate interactions between computing systems and humans. Another area is to explore the notion of teamwork; how would computing systems collaborate, cooperate, and coordinate with humans to achieve goals.

References

[1] T. Balch and R. C. Arkin. Communication in Reactive Multiagent Robotic Systems. *Autonomous Robots*, 1(1):27–52, March 1994.

[2] S. C. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1234–1239. IEEE, 1999.

[3] K. M. Carley and L. Gasser. Computational Organization Theory. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 299–330, Cambridge, MA, USA, 1999. MIT Press.

[4] S. A. DeLoach. Modeling Organizational Rules in the Multi-agent Systems Engineering Methodology. In R. Cohen and B. Spencer, editors, *Advances in Artificial Intelligence: 15th Conference of the Canadian Society for Computational Studies of Intelligence*, volume 2338 of *LNAI*, pages 1–15. Springer-Verlag, 2002.

[5] S. A. DeLoach, W. Oyenan, and E. T. Matson. A Capabilities-based Model for Adaptive Organizations. *Journal of Autonomous Agents and Multi-Agent Systems*, 16(1):13–56, February 2008.

[6] V. Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, Utrecht University, 2004.

[7] V. Dignum, J. Vázquez-Salceda, and F. Dignum. OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations. In *PROMAS*, pages 181–198, 2004.

[8] C.-H. Fua and S. S. Ge. COBOS: Cooperative Backoff Adaptive Scheme for Multirobot Task Allocation. In *IEEE Transactions on Robotics*, volume 21, issue 6, pages 1168–1178. IEEE, 2005.

[9] B. P. Gerkey and M. J. Matarić. Sold!: Auction Methods for Multirobot Coordination. In *IEEE Transactions on Robotics and Automation*, volume 18, issue 5, pages 758–768. IEEE, 2002.

[10] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.

[11] C. R. Kube and H. Zhang. Collective Robotics: From Social Insects to Robots. *Adaptive Behavior*, 2(2):189–218, September 1993.

[12] M. Kubo and Y. Kakazu. Learning Coordinated Motions in a Competition for Food between Ant Colonies. In *Proceedings of the Third International Conference on Simulation of Adpative Behavior: From Animals to Animats 3*, pages 487–492, Cambridge, MA, USA, 1994. MIT Press.

[13] M. J. Matarić. Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. In *Proceedings of the Third International Conference on Simulation of Adpative Behavior: From Animals to Animats 2*, pages 432–441, Cambridge, MA, USA, 1993. MIT Press.

[14] J. McLurkin and J. Smith. Distributed Algorithms for Dispersion in Indoor Environments Using a Swarm of Autonomous Mobile Robots. In R. Alami, R. Chatila, and H. Asama, editors, *7th International Symposium on Distributed Autonomous Robotic Systems*. Springer, 2004.

[15] M. Miller. A Goal Model for Dynamic Systems. Master's thesis, Kansas State University, April 2007.

[16] L. E. Parker. ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[17] L. E. Parker. Distributed Intelligence: Overview of the Field and its Application in Multi-Robot Systems. *Journal of Physical Agents*, 2(1):5–14, 2008.

[18] K. M. Passino. Biomimicry of Bacterial Foraging for Distributed Optimization and Control. *IEEE Control Systems Magazine*, 22(3):52–67, 2002.

[19] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[20] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[21] S. Sariel. *An Integrated Planning, Scheduling and Execution Framework for Multi-Robot Cooperation and Coordination*. PhD thesis, Istanbul Technical University, Institute of Science and Technology, Istanbul, Turkey, June 2007.

[22] S. Sariel, T. Balch, and N. Erdogan. Incremental Multi-Robot Task Selection for Resource Constrained and Interrelated Tasks. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2314–2319, 2007.

[23] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.

[24] B. G. Silverman. Performance Moderator Functions for Human Behavior Modeling in Military Simulations. `http://www.seas.upenn.edu/~barryg/PMFset.zip`, 2002.

[25] B. G. Silverman. Toward Realism in Human Performance Simulation. In J. W. Ness, D. R. Ritzer, and V. Tepe, editors, *The Science and Simulation of Human Performance*, volume 5, pages 469–498. Emerald Group Publishing, 2004.

[26] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith. First Results in the Coordination of Heterogeneous Robots for Large-Scale Assembly. In *Experimental Robotics VII*, volume 271 of *Lecture Notes in Control and Information Sciences*, pages 323–332. Springer Berlin / Heidelberg, 2001.

[27] D. J. Stilwell and J. S. Bay. Toward the Development of a Material Transport System using Swarms of Ant-like Robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 766–771, Atlanta, GA, USA, 1993.

[28] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.

[29] K. Sugihara and I. Suzuki. Distributed Algorithms for Formation of Geometric Patterns with Many Mobile Robots. *Journal of Robotic Systems*, 13(3):127–139, 1996.

[30] S.-J. Sun, D.-W. Lee, and K.-B. Sim. Artificial Immune-Based Swarm Behaviors of Distributed Autonomous Robotic Systems. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 4, pages 3993–3998, 2001.

[31] F. Tang and L. E. Parker. ASyMTRe: Automated Synthesis of Multi-Robot Task Solutions through Software Reconfiguration. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1501–1508. IEEE, April 2005.

[32] F. Tang and L. E. Parker. Distributed multi-robot coalitions through asymtre-d. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2606–2613. IEEE, August 2005.

[33] A. van Lamsweerde, E. Letier, and R. Darimont. Managing Conflicts in Goal-Driven Requirements Engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.

[34] J. Vázquez-Salceda and F. Dignum. Modelling Electronic Organizations. In V. Marik, J. Müller, and M. Pechoucek, editors, *Multi-Agent Systems and Applications III*, volume 2691 of *LNAI*, pages 584–593. Springer-Verlag, 2003.

[35] L. Vig and J. A. Adams. Multi-Robot Coalition Formation. In *IEEE Transactions on Robotics*, volume 22, issue 4, pages 637–649, August 2006.

[36] L. Vig and J. A. Adams. Coalition Formation: From Software Agents to Robots. *Journal of Intelligent and Robotic Systems*, 50(1):85–118, September 2007.

[37] C. D. Wickens, J. D. Lee, Y. Liu, and S. E. Gordon-Becker. *An Introduction to Human Factors Engineering*. Prentice Hall, 2nd edition, November 2003.

[38] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001.

[39] C. Zhong and S. A. DeLoach. Integrating Performance Factors into an Organization Model for Better Task Allocation in Multiagent Systems. MACR-TR 2010-02, Kansas State University, April 2010.

[40] R. Zlot and A. Stentz. Market-based Multirobot Coordination for Complex Tasks. *The International Journal of Robotics Research*, 25(1):73–101, 2006.