Multi-Agent Framework for Real-Time Processing of Large and Dynamic Search Spaces

John Korah
Dept. of Computer Science
University of Texas at El Paso
El Paso, TX-79968, USA
jkorah@utep.edu

Eunice E. Santos
Dept. of Computer Science
University of Texas at El Paso
El Paso, TX-79668, USA
eesantos@utep.edu

Eugene Santos Jr.
Thayer School of Engineering
Dartmouth College
Hanover, NH-03755, USA
esj@dartmouth.edu

ABSTRACT

Distributed systems based on cooperative multi-agents have been used in a wide range of application domains. However, the need for real-time processing in large and dynamic search spaces has led to new challenges. In addition to the constraints in time and computational resources, the agents have to operate under highly dynamic conditions in complex environments. Finding optimal solutions within time constraints may not be always possible. Anytime algorithms have shown great promise in providing approximate solutions. The quality of these intermediate/partial solutions depends on the amount of computational resources available for processing. The key insight that we describe in this paper is that anytime algorithms can be leveraged in a partial processing paradigm where the partial solutions can be used to quickly identify potential solutions and thereby efficiently utilize resources, even under dynamic conditions. The partial solutions can also be used for a coarse grained categorization of large search spaces that can support a mix of explorative and exploitative agent behaviors. We will describe how explicit modeling of the dynamism using a simple but unique search space model can help agents adapt to the changing information space. We describe a generic multi-agent framework that leverages our search space model while modeling various aspects of agent behavior such as candidate selection, agent interactions, etc. This framework can be used to design agents to work with partial processing in various application domains. We will develop suitable testbeds, simulation experiments, algorithms and performance metrics to validate the framework.

Categories and Subject Descriptors

I.6.5 [**Simulation And Modeling**]: Model Development—*Modeling methodologies*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

Keywords

Cooperative multi-agents, Anytime algorithms, Dynamism models, Resource allocation, Real-time processing, Performance analysis

1. INTRODUCTION

Cooperative multi-agent systems are a powerful paradigm to design distributed systems to solve large scale information search and processing problems. A set of agents are said to be cooperative when they are designed to maximize their group utility function[2]. Cooperative multi-agent systems have been used in various domains such as distributed optimization[5] and information retrieval[8]. However, the next generation of problems in many of these domains have new challenges that need to be addressed. One of the biggest challenges is brought on by the need to process dynamically changing information while providing real-time results. This places constraints on time, processing resources and communication costs that need to be addressed by multi-agent frameworks that deal with real world problems.

It is becoming more apparent that with such resource and time constraints, it may not be possible to provide optimal solutions in real-time applications. This has led to increased interest in anytime algorithms[12, 8] for its ability to provide approximate solutions. The quality of these solutions is proportional to the computational resources applied. Therefore, a candidate (for example, a document in a web search application) does not have to be fully processed for it to be evaluated and some form of judgment to be made on its viability or importance. By observing the quality of the partial solutions, the resources can be quickly allocated (or re-allocated) to a better candidate. This paradigm of processing information using anytime algorithms, called partial processing, can be leveraged to efficiently utilize computational resources. However, there is a need for a generic multi-agent framework that models the unique characteristics of the partial processing paradigm while being applicable to different domains.

The large size and dynamism of modern search spaces create a "complex environment" for the agents. Due to computational and time constraints, agents can have only a limited view of the search space and it is not possible to evaluate all options. Complex environments are also continuously changing due to the entry of new data or updates to existing data. Due to the partial-observability and dynamism, the outcome of agent actions can only be approximately modeled. The framework that we develop should support methodologies that help mitigate the effects of complex environment on resource allocation.

We propose a novel multi-agent framework for designing real-time cooperative agents to work with the partial processing paradigm and leverage its unique advantages. The key insight behind the framework is to leverage the strengths of partial processing and to group candidates based on their partial-values (i.e. approximation of their final value). Grouping candidates into regions in the search space can mitigate the effects of partial-observability as large and complex search space can now be compactly represented. Additionally, changes in the overall search space can be represented as a function of changes in these regions and shared between agents in the form of compact measures. Agents can utilize local (information of its local region) and global information to move around in the search space to better allocate resources. By the property of the partial-values generated by anytime algorithms, some groups (e.g. candidates that have large partial-values despite only a small portion being processed) show more promise for containing optimal or relevant solutions. Our search space model labels groups as explorative and exploitative, based on their potential to yield relevant results.Therefore, agents can follow a mix of strategies by moving between these regions. As a candidate is incrementally processed, it is moved to different regions based on its new partial-values. This generates a type of dynamism in the search space called internal dynamism. On the other hand, external dynamism is the traditional form caused by the entry of new candidates or update of existing ones. By capturing both these forms of dynamisms, the framework provides mechanisms for agents to react better to changes in the search space.

The main contribution of the paper is the generic multi-agent framework that leverages the unique aspects of partial processing and anytime algorithms while mitigating the effects of large and dynamic search spaces and facilitating real-time results. The framework is modular with each component dealing with different aspects such as agent interaction, decision making, etc. The components are designed to support plug-n-play so that different algorithms, metrics and analyses can be easily incorporated. In the following section, we provide a brief background on some relevant work. This is followed by a detailed description of the framework. We then provide details of the implementation that we use to validate the framework. The simulation setup is then described, followed by a discussion of the results.

2. BACKGROUND

In this section, we will focus on frameworks and methodologies that seek to mitigate key challenges in complex networks such as partial-observability and non-determinism. We will also describe some of the work in real-time multi-agent systems and discuss anytime algorithms. We begin by reviewing some of the related research in the field of multi-agent planning, that deals with decision making in complex environments. Markov Decision Process (MDP)[13] based frameworks are used to deal with non-deterministic environments by modeling contingencies. The possible outcomes of executing action a at state s are called contingencies and the MDP models use probability distributions for these transitions. This set of actions, outcomes and rewards is called a policy. The aim of the MDP models is to generate optimal policies. Partially Observable Markov Decision Process (POMDP)[4], a variation of the MDP models, works even with partial-observability. For real-time applications, POMDP based methods[6] have been developed that produce policies within time constraints. However MDP/ POMDP methods can be applied efficiently only to a domain with small and stationary state space. In a stationary state space the probability distribution of various parameters such as the transition function, observation function and state variables do not change over time. Another drawback is the assumption that all the states, actions and events can be anticipated in the beginning and explicitly represented. In many real world problems, the state space is initially unknown and the agent has to explore the environment along with the decision making process. Such agents do not perform long term planning and use local information and interactions with the environment in order to select the immediate action. Real-time heuristic algorithms used by agents in real-time path planning[3] are examples of this work.

A number of agent architectures have been proposed for real-time applications. ARTS[14] is based on the BDI (Beliefs Goals and Intention) cognitive agent architecture. The agents select predefined plans based on the goals they seek to achieve. These goals can alter depending on the changes in the environment. The innovation in ARTS is the inclusion of specific deadlines and priorities for the goals which are taken into account when scheduling the actions for the agent. SIMBA[1] is another real-time agent framework that can be used to incorporate both deliberative process and reactive process within a real-time system. Its novelty is in the use of a case based reasoning process to develop plans that is based on past experience. The frameworks, described above, focus on integrating agent planning with hard and soft real-time constraints. However, their main drawback is that they do not include mechanisms to deal with the partial-observability issues of a complex environment.

Anytime algorithms have been applied in multiple domains as they have the following properties[15]: 1. Interruptibility: The algorithm may be interrupted at any point and an approximation of the final result can be calculated, 2. Preemptability: The property of anytime algorithm by which it may be restarted with minimal overhead, 3. Result Quality: The quality of the result in anytime algorithms can be defined. The quality increases with the application of more computational resources, and 4. Predictability: It is possible to come up with a prediction scheme for the result quality, given the amount of resources applied. As part of the Information Foraging, Gathering and Matching(I-FGM)[7, 10, 9, 8] framework, anytime algorithms were developed for text and image documents, that used the partial processing paradigm to generate partial-values for the document similarity. The partial-values were used by the multi-agents to select suitable documents for further processing. This process of allocating resources to documents based on partial processing was shown to produce the final results faster than the traditional methods. Although anytime algorithms and partial processing have been used in multi-agent systems, a generic framework that leverage partial results to mitigate the challenges of large and dynamic search spaces (partial-observability, non-determinism and dynamism) do not exist. Formulating this generic framework is the main contribution of our work.

3. MULTI-AGENT FRAMEWORK

Our objective is to design a generic framework to design real-time agents that can utilize partial processing and anytime algorithms to process large and dynamic search spaces. To reiterate, the basic idea behind partial processing is that an agent quickly processes a candidate to get an approximation or partial-value of its evaluation metric. If the evaluation looks promising, the candidate is processed further by the agent. It is evident that the partial-values can be used as a way to quickly and efficiently rank the candidates and help the agents to select promising ones. In the context of our work, resource allocation occurs in the form of agents selecting a candidate and processing it for a certain amount of time. In our experimental section, we assume that the number of computational resources is equivalent to the number of agents. However, our framework can extend to the generic case where the resources are separate from the agents. By the property of anytime algorithms, the quality of the partial-values increases as more computational time is alloted to it. Therefore, by incrementally processing the candidates we can efficiently determine if a candidate becomes less promising and should be given less priority. This results in less wastage of resources on candidates that will ultimately turn out to be non-optimal.

Current frameworks are not designed to leverage the benefits from partial processing. Partial-values provide a unique structure to the search space that can be used to concisely represent the changes in large and dynamic search spaces. Developing a search space model is key to harnessing the advantages of partial processing. As can be seen from the framework architecture(Fig. 1), the search space model is the crucial lynch pin that holds together the different components in the framework. Each component in the framework, dealing with a certain aspect of agent function, interacts with the search space model. For example, the interaction module, that deals with agent communications, relays information about local conditions of the search space to other agents. This is used to update their search space models. The framework architecture has been designed to support plug-n-play. This allows for new algorithms and metrics to be implemented in the framework quickly. We will now describe each component in detail.

3.1 Search Space Model

The search space model mitigates the effects of partial-observability by providing a detailed view of its immediate environment and a coarse grained view of the entire search space. It divides the search space into a grid of dimension (m, n), using the two important parameters for a candidate d: 1) portion of the candidate processed or process-ratio $r_d \in [0, 1]$, and 2) current partial-value $v_d \in [0, 1]$. Each grid-block is denoted by $B(i, j)$ where $1 \le i \le m$ and $1 \le j \le n$. A candidate d is placed in $B(i, j)$ where $(\frac{i-1}{m}) \le r_d < (\frac{i}{m})$ and $(\frac{j-1}{n}) \le v_d < (\frac{j}{n})$. An agent a is assigned to one grid-block $B_{i,j}$ at a time. This is termed the agent's home-block $H_a(i, j)$. The agent can only process candidates in its home-block. Resources are said to be reallocated when the agent re-assigns itself to another grid-block. The initial home-block of an agent is not specified as part of the framework but is decided by the agent designer. The framework supports dynamic partitions where the number of grid-blocks change according to the number of candidates in the search space. Dynamic partitioning can also help in moving from fine grained grouping of candidates to a more coarse grained grouping and vice versa. However, for the implementation presented in the paper, we consider only static partitioning where the search space has $m \times n$ grid-blocks.

For each $B(x, y)$, we define two sets of grid-blocks, $F_B = B(i, n)$ where $x \le i \le m$, and $T_B = B(m, j)$ where $y \le j \le n$, are specified. Grid-blocks of T_B are important as "relevant" candidates will ultimately end up here. A candidate d is said to be relevant if its $v_d \ge \alpha$ where α is a high value ($\sim$0.9). Since partial-values give us an inkling of the final relevancy values, candidates in certain grid-blocks have a higher probability of ending up in T_B. Exploitation region contains candidates that have high likelihood of reaching T_B, whereas exploration region contains candidates with low likelihood of reaching T_B. The search space model uses this heuristic to group candidates into regions of exploration and exploitation, which can then be used by agents to make selections. In order to do this, we quantify the likelihood of candidates in a grid-block $B(x, y)$ using a measure called the risk parameter $R_{risk}(x, y)$. We define $R_{risk}(x, y)$ as follows:

$$R_{risk}(x, y) = 1 - \frac{|T_B(x, y)|}{|F_B(x, y)| + |T_B(x, y)|}\left(1 - \frac{x-1}{m}\right) \quad (1)$$

As a candidate d is processed, its r_d and/or v_d values change. To conform to the search space model, the candidate is moved to a different grid-block. Similarly, the agents move to new home-blocks. This change in the search space caused by the movement of agents and candidates between grid-blocks is termed internal dynamism. This is in contrast to the other dynamism, called external dynamism, which is caused by the entry of new candidates or departure of candidates after they are completely processed. Internal dynamism is modeled in the search space model using a network dynamism graph $G_t(V, E)$, to represent the flow of agents and candidates between the grid-blocks at time t. The set of vertices in $G_t(V, E)$ $V = V_I \cup V_{0,0}$. $V_{0,0}$ represents the external environment from which new candidates are entering the search space and into which completely processed candidates are sent. Vertex set $V_I = \{v_{i,j}\}$ where $v_{i,j}$ represents the grid-block $B(i, j)$. The edge set $E = E_I \cup E_E$. $E_I = \{e_{i,j}^{k,l}\}$ where $e_{i,j}^{k,l}$ is an edge from $v_{i,j}$ to $v_{k,l}$ such that $m \ge k \ge i$, $n \ge l \ge j$, $i \ne 0$, $j \ne 0$. The edges in E_I represent internal dynamism as they contain the flow of agents and candidates between the grid-blocks. $E_E = \{e_{0,0}^{1,1}\} \cup \{e_{m,n}^{0,0}\}$ represents the flow due to external dynamism. Since, the dynamism can be represented as flow in $G_t(V, E)$, we can use a graph based metric such as load to quantify internal and external dynamism. The load $L(i, j, t)$ in a grid-block $B(i, j)$ at time t may be defined as:

$$L(i, j, t) = \frac{N_C(i, j, t)}{N_A(i, j, t)} \quad (2)$$

$N_C(i, j, t)$ and $N_A(i, j, t)$ are the number of candidates and agents, respectively, in grid-block $B(i, j)$ at time t. $N_C(i, j)$ and $N_A(i, j)$ represent the current values of candidates and agents in $B(i, j)$. It may be noted that this definition of load is not part of the framework specifications and may be changed according to application domain requirements.

3.2 Interaction Module

The interaction module deals with communication aspects of agent functionality. Agents can communicate with each other using various forms of communication including point to point and broadcast. An agent sends out information about its home-block to other agents, helping them to update their search space models. In order to keep information of the search space current, the frequency of these messages will increase with dynamism. The module supports a wide range of communication schedules including targeted and broadcast communications.

3.3 Prediction Module

The network dynamism graph G_t provides a snapshot of the search space at time t. Therefore, dynamism can be formally represented as a set of functions $f_{\Delta t} : G_t \rightarrow G_{t+\Delta t}$. These functional mappings can be implemented as prediction algorithms that approximate future states of the search space. Forecasting algorithms, using flow rates of agents and candidates, can be used with G_t to estimate the graph $G_{t+\Delta t}$ at time $t + \Delta t$. By having a formalized representation of dynamism, the framework can support a wide range of candidate selection algorithms that uses both global factors such as future search space states and local factors such as future values of individual candidates. The decomposable architecture of the framework is highlighted by the fact that a combination of search space model and interaction module provides a static snapshot of the search space. On the other hand, a combination of search space model, interaction module and prediction module provides a dynamic snapshot.

3.4 Internal Model

The decision making processes in the agent are implemented in the internal model component using following sub-components:

a) Agent Migration sub-component: contains algorithms used by the agent to decide whether to move to a different home-block. It uses the information of the search space gathered from other agents about current results and load conditions. The decision making for agent mobility is modeled using the following components:
Search space model + Interaction module + Prediction module + Agent migration sub-component ⇒ Agent mobility

b) Agent Communication sub-component: Communications are expensive in parallel/distributed systems and efficient communication schedules can lead to better performance. The agent communication sub-component works with the interaction module to send and receive messages. This sub-component decides on what data to send, its recipients, frequency of messages, etc. The decision making about agent communication is modeled by the following components:
Search space model + Interaction module + Agent Communication sub-component ⇒ Agent communication

c) Candidate Selection sub-component: The algorithms used by the agent to select candidates are implemented in this sub-component.

4. FRAMEWORK IMPLEMENTATION

The previous section provided a detailed description of the framework. We will now implement the framework by developing a set of algorithms and metrics for the various components. They will be used to develop control systems for the simulation experiments that will be conducted to validate the framework. The implementation described in this section is only one of the many configurations supported by the framework.

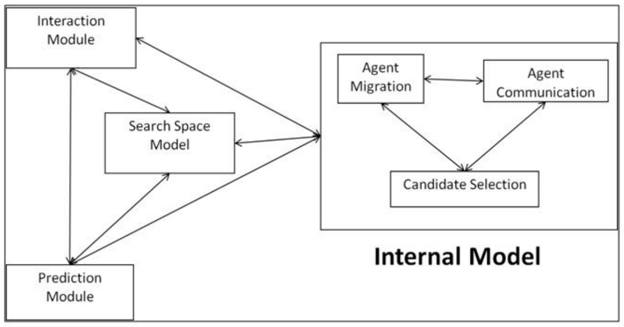

Figure 1: Resource Allocation Framework

4.1 Search Space Model

In our implementation, we use MySQL[1] database for agents to communicate with each other and store search space information that is used in the decision making process. MySQL simplifies the implementation while enabling various types of agent communication such as broadcast and point to point. Each grid-block is associated with a MySQL table that stores details about candidates (e.g. current values of v_d, r_d) and agents (e.g. agent ID) residing in it. Agents have access to the MySQL table of their home-block and use the information to select a suitable candidate for processing. After processing, the candidates are placed in a grid-block appropriate to their new process-ratio and partial-value. The partial-values of the candidates are monotonically non-decreasing. Therefore, the candidates in a grid-block $B(i, j)$ will only move to a grid-block $B(k, l)$ where $i \leq k \leq m$ and $j \leq l \leq n$. This means that the edges in the network dynamism graph $G_t(V, E)$ are unidirectional. For each edge $e_{i,j}^{k,l} \in E$, the flow rate $f_{i,j}^{k,l}$, which is the the number of candidates passing through the edge per second, is maintained. Each node $v_{i,j} \in G_t$ has $E_{in}(i, j)$ and $E_{out}(i, j)$ as the set of incoming and outgoing edges. For each node $v_{i,j}$, we store the following metrics: 1. $g_{in}(i, j)$: current in-flow rate of the agents into the grid-block $B(i, j)$, 2. $g_{out}(i, j)$: current out-flow rate of the agents out of the grid-block $B(i, j)$, 3. $N_A(i, j)$, and 4. $N_C(i, j)$

4.2 Interaction Module

Agent communication is implemented as reading/writing of messages from/into MySQL tables in the grid-blocks. We measure the cost of communication using a distance metric D_G. The distance $D_G(a, b)$ between two agents a and b, in home-blocks $H_a(i, j)$ and $H_b(k, l)$, respectively, is given by $D_G(a, b) = max(|i - k|, |j - l|)$. Using the distance metric, we can also define a neighborhood for the agent. The neighborhood of radius 1 of an agent with home-block $B(x, y)$ are all the grid-blocks $\{B(i, j)\}$ where $max(|i - k|, |j - l|) = 1$. Based on the information content, the messages are of two types:

[1]http://www.mysql.com

4.2.1 Result Messages

The agents will adjust their processing strategy, alternating between being more explorative or more exploitative, based on the number and quality of relevant candidates in T_B. The relevant set of candidates in the system at time t is denoted by R_t. For the simulations in the paper, it is assumed that R_t is known beforehand. In real world scenarios, where R_t is not known, an approximation of R_t may be used. Result messages carry information about R_t and is implemented in the form of agents periodically querying the MySQL tables of the grid-blocks in T_B. Using the values of $R(t)$, the agent calculates the following metrics: a) Result-Age ($R_{age}(t)$): is the time period for which the current result set has remained unchanged, b) Discovery-Time ($T_{disc}(d)$): of a candidate d is the time instant it was found to be relevant or in other words, its v_d was found to be greater than α, and c) Recall-Ratio ($R_{recall}(t)$): is defined as the ratio between the number of candidates in the relevant set R_t and the total number of relevant candidates.

4.2.2 Dynamism Messages

Dynamism messages contain information about changes in the search space and are used by agents to update their individual network dynamism graph G_t. In our implementation, dynamism messages are generated by an agent when it moves to a new home-block or selects a candidate for processing or has finished processing a candidate.

4.3 Prediction Module

One of the novelties of the framework is its explicit modeling of internal and external dynamism. By modeling dynamism, predictions or estimations can be made about the future states of the search space. This allows the agents to pro-actively respond to changes in the search space, potentially leading to more efficient resource allocation. In this implementation, we have designed the following prediction algorithms:

4.3.1 Candidate Prediction

Prediction algorithm for estimating the future partial-values of the candidate is dependent on the domain. Since we use synthetic candidates (with randomly generated partial-values) in our testbed, we use a generic prediction algorithm based on linear regression. The algorithm uses historical values of v_d and their corresponding r_d values, stored as part of the candidate information in the grid-block table, to predict the future partial-value for a given processing time.

4.3.2 Load Prediction

In order to predict loads in the grid-blocks, the number of agents and candidates at some future time $t + \Delta t$ are calculated using the following formulae.

$$N_A(i,j,\Delta t) = N_A(i,j) + (g_{in}(i,j) - g_{out}(i,j)) * \Delta t \tag{3}$$

$$N_C(i,j,\Delta t) = N_C(i,j) + \left(\sum_{e_{x,y}^{i,j} \in E_{in}(i,j)} f_{x,y}^{i,j} - \sum_{e_{i,j}^{x,y} \in E_{out}(i,j)} f_{i,j}^{x,y} \right) * \Delta t \tag{4}$$

$N_A(i,j,\Delta t)$: predicted no. of agents in grid-block $B(i,j)$ after Δt time interval

$N_C(i,j,\Delta t)$: predicted no. of candidates in grid-block $B(i,j)$ after Δt time interval

The in-flow (g_{in}) and out-flow (g_{out}) rates are calculated using the dynamism messages send by agents when it moves between grid-blocks or when it transfers candidates after a process cycle. Load is then calculated using Eqn. 2.

4.4 Internal Model

We implemented the following sub-components in the internal model:

4.4.1 Candidate Selection sub-component

For each candidate, a priority value is calculated after each process cycle. Priority value reflects the potential of a candidate to be relevant and is based on its current and predicted partial-values. The priority of a candidate d that has an allocation of processing time t_k at its k^{th} step is given by

$$P_{d,k} = \beta_1(\delta P_{d,k-1} + (1-\delta)v_d(k)) + \beta_2 q_2 \tag{5}$$

$$q_2 = \begin{cases} \Delta_t & \Delta_t > 0 \\ -\Delta_0 \frac{t_k}{t_0} & Otherwise \end{cases}$$

$$\Delta_k = v_d^{'}(k+1) - v_d(k)$$

In Eqn. 5, β_1, β_2 and δ are constant scaling factors. t_0 is the maximum processing time that can be assigned to a candidate and Δ_0 is the average increase in partial-value when t_0 processing time is applied. Δ_0 is calculated using the increases in partial values of the candidate during previous iterations. When a candidate is being processed for the first time, a default value for Δ_0 is used. More details on the priority function are provided in [10]. Based on the priority values, an agent identifies the top 10 candidates in its home-block and a document is selected at random for processing.

4.4.2 Agent Communication sub-component

This module decides when an agent sends and receives messages. As mentioned before, agents utilize the routines in the interaction module to read and write information in the database. Some of the messages are triggered by events such as agents moving to a different home-block or a candidate being moved to a new grid-block. Other messages, such as reading the list of relevant candidates, are sent periodically.

4.4.3 Agent Migration sub-component

An agent has to decide when and where to move as part of its resource allocation policy. In order to deal with the inherent uncertainty, we use Bayesian Knowledge Bases(BKBs)[11] to model the migration behavior of the agents. BKBs are probabilistic networks that encode if-then rules using random variables (rvs). We use information about the dynamism, in the form of the metrics $R_{age}(t)$ and $R_{recall}(t)$, along with the decision making mechanism to decide whether to be more/less explorative/exploitative. We use a selection function that combines the output of the BKB with the future load values in the neighborhood to select an appropriate grid-block to move to. The algorithms to predict future load values are implemented in the prediction module.

5. VALIDATION

Through simulation experiments, we seek to validate our framework and show that agents can do more efficient resource allocation through explicit modeling of internal and external dynamism. For the set of experiments in this paper, we will use a synthetic testbed of candidates based on the real world data sets generated as part of our previous work in I-FGM[7, 10, 9, 8]. Each candidate is represented by a randomly generated 2-tuple: $\{(r_d(t_1), v_d(t_1)), \ldots\ (r_d(t_n), v_d(t_n))\}$ such that $r_d(t_i) < r_d(t_j)$ and $v_d(t_i) < v_d(t_j)$ where $t_i < t_j$. $r_d(t)$ and $v_d(t)$ are the process-ratio and partial-value respectively, of the candidate d after t amount of processing time has been applied. During the simulation, the instant each candidate enters the search space is its entry time. By manipulating their entry-times, various rates of external dynamism can be simulated. By using a synthetic testbed, we can manipulate properties such as candidate length (short, long), rate of change in the partial-values, proportion of candidates based on size, etc. to create various conditions in the search space. Real world scenario may involve candidates of a particular type and will not enable us to test the framework in a robust manner. In our experiments, we will generate different testbeds to obtain a more rounded assessment of our framework. Such extensive simulations are also required to perform the statistical analysis of system performance, described in the following sections.

The validation consists of two experiments. In Experiment-1, the objective is to validate the fundamental design characteristics of the search space model. We show that grouping the candidates based on their process-ratio and partial-values, and labeling regions as explorative and exploitative in the search space can lead to efficient and dynamic resource allocation. In order to validate the search space model, we compare the performance of three control systems that were implemented using the algorithms and metrics described in Section 4. The control systems are Baseline, Static-Agent and Mobile-Agent systems. They differ in the way they model the search space. In Baseline, the search space is considered to be monolithic - candidates are not grouped. All the agents have access to all the candidates at all times and follow a simple candidate selection policy - select the candidate with the highest priority values for processing. The Static-Agent system on the other hand, uses the grid-block division of candidates in the search space based on their partial-values and process-ratio. However, the agents cannot move between grid-blocks and make their selection among the candidates in their fixed home-block. This control system represents the classic parallel/distributed method of task division. Since agents in Baseline and Static-Agent systems do not communicate with each other, the interaction module is not implemented. Also, both systems do not take dynamism of the search space into account. Therefore, only the candidate selection sub-component of the internal model and candidate prediction algorithm of the prediction module are implemented. Mobile-Agent system is implemented according to the description in Section 4. The agents move in a search space that is partitioned into grid-blocks. The agents communicate with each other to relay information on dynamism and results. Although the system models the dynamism in the search space using the network dynamism graph, it does not predict future states of grid-blocks using metrics such as load. We adopt the following restrictions in order to simplify the implementation. Agents can only migrate to and/or communicate with agents in its immediate neighborhood. Its immediate neighborhood consists of all the grid-blocks that are at a distance of 1 unit as defined by the distance metric, D_G.

In Experiment-2, we design a control system, called the Dynamic-Agent system, that uses the idea of internal and external dynamism, to allocate resources and analyze its performance by comparing it to the Mobile-Agent control system. Mobile-Agent system uses only the current load conditions in the neighborhood to decide on the grid-block to move to. On the other hand, Dynamic-Agent system uses both current and future load values in the neighborhood, and the predictions of future load conditions are used as an input to the agent migration sub-component, along with the result quality and current load information.

5.1 Experiment-1

For Experiment-1, the performance metric used to compare system performance is the total-recall time. Total-recall time is the time taken for the system to retrieve all the relevant candidates. Since real-time search systems deal with getting the results quickly, total-recall time is an important performance measure. The testbed contains 1000 candidates. Each control system is run on the testbed with agents ranging from 8 to 64 and the grid sizes (2, 2) and (4, 4). The experiments are conducted using the randomized block paradigm where a block run consists of running the three systems in a randomized sequence. We perform 3 block runs for each combination of agent number and grid size. The results are collected for each run and analyzed. Analysis of Variance (ANOVA) is performed on the results to ascertain whether the difference in performance of the control systems is statistically significant.

Results and Analysis

On analyzing the bar charts in Fig. 2 & Fig. 3, we see that Mobile-Agent system performs better than the Static-Agent and Baseline systems in all the experimental runs. Baseline performs worse than the other control systems in all the runs. In fact, the total-recall time in the Baseline increases with increase in the number of agents. This is due to the contention between the agents as all agents in the Baseline try to access the candidate with the highest priority. In the Static-Agent system, although the total-recall time decreases with the increase in the number of agents, it decreases at a slower rate than the Mobile-Agent system. The primary reason for the better performance of the Mobile-Agent versus the Static-Agent systems is that the former is able to minimize agent idleness by moving between the grid-blocks. Although the agents in Static-Agent systems do not perform expensive communication operations to interact with each other, the cost savings are not enough to offset the cost due to idleness. For lack of space, only the graphical ANOVA figures for grid size (2,2)(Fig. 4) is shown. ANOVA provides analysis on how significant the performance variations between the systems are. The graphical ANOVA compares the block deviations and residuals with the deviations between the system performances and shows that the block deviations and residual values are much smaller. Therefore, it is clear that the superior performance of the Mobile-Agent system is statistically significant. We have shown that a grid based search space model that categorizes the candidates using their process-ratio and partial-values is indeed viable. Since resource allocation in Mobile-Agent system uses R_{risk}, the experimental results also validates the notion of leveraging exploration and exploitation regions for better performance.

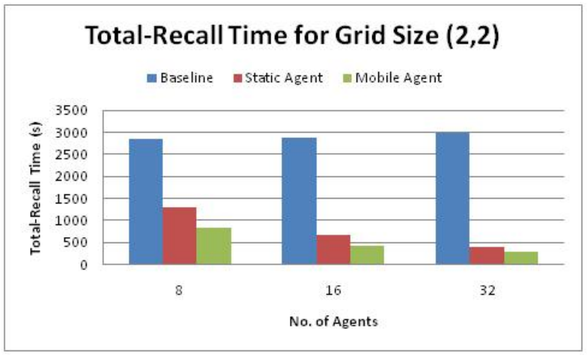

Figure 2: Total-recall time for Experiment-1 with grid size=(2,2)

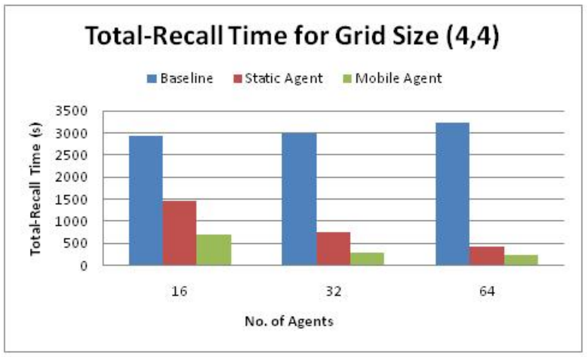

Figure 3: Total-recall time for Experiment-1 with grid size=(4,4)

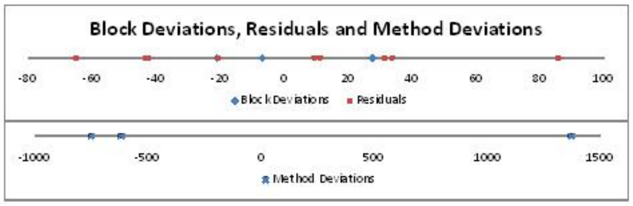

Figure 4: Graphical ANOVA for Experiment-1 with no. of agents = 64 and grid size = 4,4

5.2 Experiment-2

We run the Mobile-Agent and Dynamic-Agent control systems on a large testbed of 2000 candidates. The testbeds are also made more dynamic by introducing groups of 100 candidates at regular intervals of time. The Dynamic-Agent system uses the history of the flow rate values of candidates and agents to predict future values. We introduce a set of 10 relevant candidates at regular intervals in the search space. For each of these sets of relevant candidates, we calculate the proportion of these candidates that are discovered first by the control systems. We call this performance metric the Discovery-Rate of the system. Both the Mobile-Agent and Dynamic-Agent systems are made to run on each testbed 3 times and the Discovery-Rate values are tabulated. Since a document is either discovered by Dynamic-Agent or Mobile Agent systems, if x is the portion of the documents discovered (by definition its discovery-rate) by the Mobile-Agent system, then $(1 - x)$ is the portion discovered by the Dynamic-Agent system. We use multiple testbeds which differ in the proportion of small and large candidates contained in it. This is represented by the parameter β which is defined as the ratio of small documents to large documents. Large candidates have an average processing time of 50s and small candidates have average processing time of 10s. Since the small candidates get processed quickly, the internal dynamism (due to flow of candidates) within the search space will be high when β is large. By using testbeds with different β values, we will be able to test the control systems under different rates of internal dynamism.

Results and Analysis

For brevity, we discuss the Discovery-Rates of the Dynamic-Agent system for two β values: 0.1 (low dynamism) and 0.3 (high dynamism) in Fig. 5 and Fig. 6, respectively. In these graphs, the relevant sets of candidates are labeled 1, 2 and so on. Since the Discovery-Rates of the Mobile-Agent system can be easily ascertained from those of the Dynamic-Agent system, they are not displayed. The Dynamic-Agent system is said to have better performance if its Discovery-Rates are greater than 0.5. From the plot for $\beta = 0.1$ values (Fig. 5), we see that Dynamic-Agent system performs much better than Mobile-Agent in 4 out 5 relevant sets of candidates. However for the higher dynamism (Fig. 6), Dynamic-Agent performs only as good as Mobile-Agent by winning for relevant set no. 2 & no. 5 and tying for set no. 4. The main reason for the relatively poor performance for high dynamism is that the load prediction algorithm uses averaging algorithms to calculate future flow rates of candidates and agents in the network dynamism graph. Developing more sophisticated prediction algorithms based on the dynamism models is part of the future work. However, in the limited case of low dynamism rates, we are able to demonstrate that resource allocation based on predicting future states, using our dynamism model, performs better.

6. CONCLUSION

In this paper, we introduce a novel design framework for cooperative multi-agent systems for anytime processing that seeks to mitigate the effects of complex environments. It uses a simple idea of partitioning the search space into grid-blocks based on the intermediate values generated by the anytime algorithms. Our methodology hinges on the fact that these partial-values can be used as a criteria for allocating resources. We also explicitly model internal and external dynamism as a way to represent, measure and communicate changes in the search space. This also helps the agents to develop both reactive and pro-active resource allocation policies. The framework has a flexible, component based architecture that models important aspects of agent behavior such as agent interaction, agent decision making, etc. We implemented the framework and simulation experiments were conducted to validate the feasibility of using the grid-based search space model for efficient resource allocations. The results were analyzed and interesting trends and observations were discussed.

For future work, we will look at using the framework

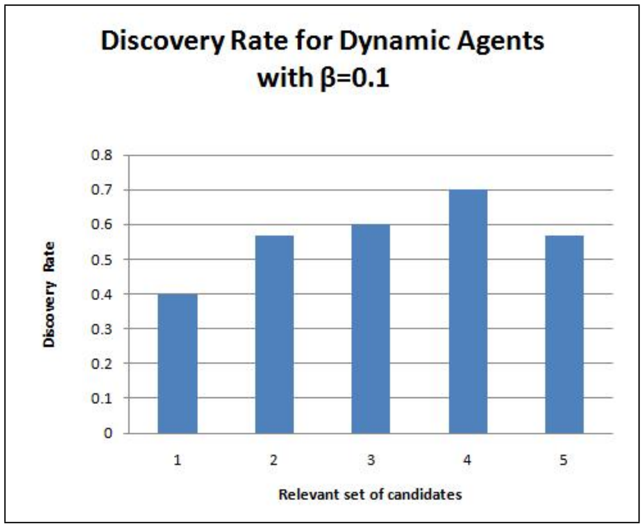

Figure 5: Discovery-Rate for Dynamic-Agent system with = 0.1 in Experiment-2

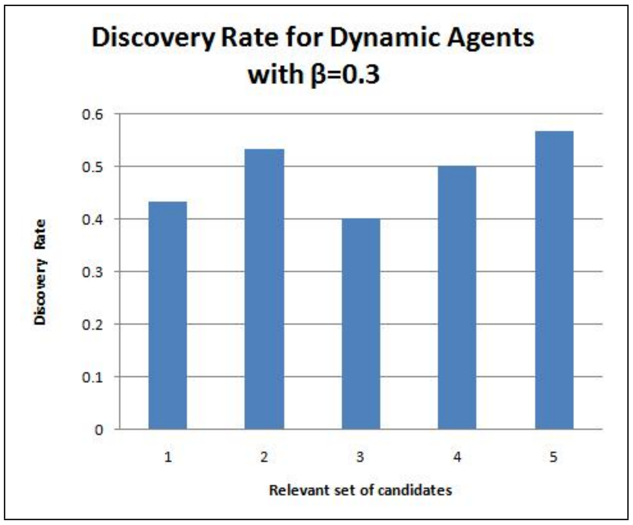

Figure 6: Discovery-Rate for Dynamic-Agent system with = 0.3 in Experiment-2

to develop and analyze more sophisticated agent strategies. We will also look into developing communication schedules that utilize the dynamism in the search space, to optimize communication costs. The framework has functional mappings that can be used to form rigorous mathematical relations, such as between communications and search space dynamism. In the experimental section, we provided a rigorous statistical analysis of the framework under various conditions, that required a synthetic testbed. As part of future work, we will look at applying the framework to concrete scenarios.

7. REFERENCES

[1] C. Carrascosa, J. Bajo, V. Julian, J. Corchado, and V. Botti. Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems with Applications*, 34(1):2 – 17, 2008.

[2] P. Hoen, K. Tuyls, L. Panait, S. Luke, and J. La Poutre. An overview of cooperative and competitive multiagent learning. In K. Tuyls, P. Hoen, K. Verbeeck, and S. Sen, editors, *Learning and Adaption in Multi-Agent Systems*, volume 3898 of *Lecture Notes in Computer Science*, pages 1–46. Springer Berlin / Heidelberg, 2006.

[3] S. Koenig and X. Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3):313–341, 2009.

[4] W. S. Lovejoy. A survey of algorithmic methods for partially observable markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[5] R. L. Raffard, C. J. Tomlin, and S. P. Boyd. Distributed optimization for cooperative agents: Application to formation flight. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, 2004.

[6] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[7] E. Santos Jr, E. Santos, H. Nguyen, L. Pan, and J. Korah. Large-scale distributed foraging, gathering, and matching for information retrieval: Assisting the geo-spatial intelligence analyst. In *Proceedings of SPIE*, volume 5803, page 66, 2005.

[8] E. Santos Jr, E. Santos, H. Nguyen, L. Pan, and J. Korah. A large-scale distributed framework for information retrieval in large dynamic search spaces. *Applied Intelligence*, 35:375–398, 2011.

[9] E. Santos Jr, E. Santos, H. Nguyen, L. Pan, J. Korah, H. Xia, F. Yu, and D. Li. *E-Government Diffusion, Policy and Impact: Advanced Issues and Practices*, chapter Analyst-Ready Large Scale Real Time Information Retrieval Tool for E-Governance, pages 268–294. IGI Global, 2009.

[10] E. Santos Jr, E. E. Santos, H. Nguyen, L. Pan, J. Korah, Q. Zhao, and M. Pittkin. I-FGM information retrieval in highly dynamic search spaces. In *Proceedings of SPIE*, volume 6229, page 622901, 2006.

[11] E. Santos Jr and E. S. Santos. A framework for building knowledge-based under uncertainty. *Journal of Experimental and Theoretical Artificial Intelligence*, 11:265–286, 1999.

[12] E. Santos Jr, S. E. Shimony, and E. M. Williams. Solving hard computational problems through collections (portfolios) of cooperative heterogeneous algorithms. In *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference*, pages 356–360, 1999.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[14] K. Vikhorev, N. Alechina, and B. Logan. The ARTS real-time agent architecture. In *Proceedings of Second Workshop on Languages, Methodologies and Development Tools for Multi-agent Systems (LADS2009)*, Turin, Italy, September 2009.

[15] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.