A Novel Multi-Objective Multi-Constraint Genetic Algorithms Approach for Co-ordinating Embedded Agents

Elias Tawil and Hani Hagras
Department of Computer Science
University of Essex
Colchester CO4 3SQ
{etawil, hani}@essex.ac.uk

Abstract – In this paper, we present a distributed, fault-tolerant and adaptive software architecture for co-operative multi-embedded agent systems that operate in ubiquitous computing environments. The system is based on a novel genetic algorithm that learns to co-ordinate a set of embedded agents whilst satisfying a set of local and global objectives and constraints. The system operates in a lifelong learning mode which adapts to changes in the environment or users' requirements. We experimented on various embedded agents situated in a ubiquitous computing environment testbed which is the Essex Intelligent Dormitory. The results manifested that the system can converge within a short time interval to a co-ordination strategy that satisfies the global and local objectives and constraints.

Keywords: Ubiquitous computing, genetic algorithms, multi-objective optimisation, embedded agents.

1 Introduction

Ubiquitous Computing is a vision where computers would be embedded within everyday devices such as kettles, clothes, and mobile phones. We seem to be heading towards this goal as breakthroughs in technology and manufacturing are resulting in more powerful and smaller processors. As these embedded computers become more abundant, it would be beneficial to have them co-operate with each other to learn and adapt to users’ needs whilst satisfying multiple objectives and constraints.

An example of a Ubiquitous Computing Environment (UCE) is the Intelligent Dormitory (iDorm) at The University of Essex. At the first glance, it looks very much like a standard student accommodation bedroom. Looking under the surface, it is full of sensors such as cameras, thermistors, and pressure pads which are utilised by computerised actuators to control devices such as the window blinds, heating fans and lamps. The iDorm is constantly growing as more sensors and actuators are being introduced to it.

The multitude of computerised devices that make up a UCE can include intelligent embedded agents, which are defined as networked everyday objects that contain intelligent processes that enable them to co-operate to achieve common goals. They can be static like an intelligent refrigerator, mobile like a vacuum cleaning robot, and portable like a smart phone. Since they are expected to enter or leave the UCE anytime, there is a need for an adaptive, online and lifelong learning process for their co-ordination procedure. This process should operate with minimum user interaction and it should aim to satisfy the various goals and constraints imposed on the agents.

The multi-embedded agent system has to deal with different kinds of objectives and constraints [8]. Some of these constraints can be soft, where it would be preferable but not necessary to satisfy them as they may conflict with others. They can also be hard constraints that should be satisfied at all times since they usually involve system and user safety and security as well as user privacy.

There is a need to develop a co-ordination scheme for multi-embedded agent systems that support an ad hoc and highly dynamic (re)structuring of the embedded agents to satisfy the possibly conflicting goals and constraints whilst shielding the laypersons from the need to understand or work directly with the hidden technology. The situation is likely to become much more challenging as the number, varieties, and uses of computer based artefacts increase. One major problem with co-ordinating a system of embedded agents in UCE is that too little information going to a single agent would reduce its potential and might even render it useless. On the other hand, too much information might cause confusion or clog up the network.

There have been many efforts in the fields of multi agents architectures in UCE and multi-robot co-operation (mobile robots may be viewed as a special case of embedded agents). At the MIT Media Laboratory, Minar
presented HIVE, which is a distributed agents platform and a decentralised system for building applications by networking local system resources [5]. The key abstraction in HIVE is the software agent: applications are built out of an ecology of multiple agents interacting over the network. The main drawback of this work is that their agents do not learn and adapt to their environment; furthermore, they have to be configured by someone. *The Aware Home* is a house made to create a laboratory for research in ubiquitous computing for everyday activities [4]. There is a control room in the basement for centralised computing services. Their system is centralised and it is not intelligent in the sense that it does not learn nor adapt to its environment. Its sensors and actuators are fixed and dedicated to performing specific tasks. Mozer et al have been studying a UCE that is controlled by neural networks [6]. This would not be adaptive to changes in the structure of the building’s sensors and actuators. ALLIANCE is a software architecture that facilitates the fault tolerant co-operative control of teams of heterogeneous mobile robots performing missions composed of loosely coupled subtasks that may have ordering dependencies [7]. ALLIANCE’s goal-achieving behaviours do not learn and adapt to the environment.

According to the authors’ knowledge, there is no developed and implemented system that is fault tolerant, adaptive, dynamically-structured, and can learn online the co-ordination scheme of multi-embedded agent systems to satisfy multiple conflicting objectives and constraints in a UCE. Such system will be the focus of this paper.

2 The adaptive co-ordination system

We propose a novel software architecture based on a multi-objective and multi-constraint genetic algorithm approach. It can learn to co-ordinate multiple embedded agents in a UCE and adapt in a lifelong learning manner to changes in the environment and to users’ desires whilst satisfying a set of goals and constraints.

2.1 The multi-objective evolutionary system

The agents utilise a fast elitist multi-objective and multi-constraint genetic algorithm which incorporates a fast non-dominated and parameterless sorting procedure [2]. Figure 1 is a flow chart that shows our evolutionary system. It shows that once the system is started, a parent population is initialised to random individuals. The agent would initially run on a random chromosome. Users, once unhappy with its performance, would *punish* the system by inputting their desires of how they want to change it (e.g., by flipping a light switch or turning on a heating fan). This information is used to calculate the user comfort objective. Whenever the system is punished, a tournament selection procedure, along with single point binary crossover and bitwise mutation, are used to create a child population.

After that, both the parent and child populations are combined into one single population of length $2N$ where $N$ is the length of the parent and the child populations. Next, the combined population is sorted according to non-domination.

The fast non-dominated sorting algorithm [2] works as follows. First, each solution must be compared with every other solution in the population to see if it is dominated. If a solution satisfies more objectives than another, then it dominates it. This process is continued to find the members of the first non-dominated front for all population members. After this, the solutions of the first front are temporarily ignored and the aforementioned procedure is repeated to find the subsequent fronts. The new parent population is formed by adding solutions from the first fronts until the size exceeds the population size $N$. This is when the solutions of the last accepted front are sorted according to the *crowding distance*. The *crowded comparison operator* is then used to pick solutions until the new parent population would be filled.

The *crowding distance* is used to get an estimate of the density of solutions surrounding a particular point in the population. We take the average distance of the two points on either side of this point along each of the objectives [2]. This serves as an estimate of the size of the largest cuboid enclosing the point without including any other point in the population. This is used for increasing the diversity of the Pareto-optimal solutions. The *crowded comparison operator* guides the selection process towards a uniformly spread out Pareto-optimal front [2]. Between two solutions with differing non-dominated ranks, the point with the lower rank is preferred. Otherwise, if both points belong to the same front, then the point which is located in a region with less points (i.e. the size of the cuboid enclosing it is larger) is chosen. The crowded comparison operator is used only when both points that are being compared would be feasible, i.e. satisfy all of their constraints. Otherwise, the feasible solution is preferred over the infeasible one. Should both solutions be infeasible, then the one that violates less constraints would be preferred. After a new parent population of size $N$ is determined; tournament selection, binary crossover, and bitwise mutation are used to create a new child population of the same size. The selection criterion is based on the crowded comparison operator. The whole procedure will repeat itself until the maximum number of iterations is exceeded. The elitist method, along with the crowded comparison operator, is hoped to find multifarious Pareto-optimal solutions. The genetic algorithm will then select a single random solution from the Pareto-optimal front to be used for control.

When the system is punished for the subsequent times, it would run with lower values for the maximum number of generations and the rate of mutation. Its aim
would usually be a subset of the previously evolved set of solutions.

![Main optimisation procedure diagram](image)

Figure 1. Main optimisation procedure

The multi-objective algorithm is useful for finding a diversity of solutions for conflicting objectives of different types that a standard genetic algorithm with a complicated fitness function would not be able to handle. Classical optimisation methods for handling multiple objectives require the user to supply a weight vector or a preference vector. Since such a weight or preference vector scalarises multiple objectives into a single objective, the outcome of the optimisation processes is usually a single optimal solution. In order to obtain a set of Pareto-optimal solutions, these methods must have to be applied many times with different weight or preference vectors. Furthermore, most classical approaches demand that all objectives be of the same type. If an optimisation problem involves an objective that is of a different type, that would have to be converted to match the others’ type. This is a main issue in our case since it would be difficult to compare energy efficiency with network usage or user preference.

The specific multi-objective algorithm that we employ is better than others because it is not computationally complex and thus suitable for embedded agents. In addition, it does not need a sharing parameter, and is elitist. It is easy to implement and use and does not lose good solutions while it is learning online. It is capable of handling discrete and real-valued decision variables together without any special consideration. It is capable of handling constraints in a simple way and without having to set any new parameter, such as a penalty function. It is also capable of handling multiple objectives of mixed types easily. According to the authors’ knowledge, no previous work has employed such a tool to coordinate agents to satisfy many constraints and conflicting objectives.

2.2 The overall system architecture

![Overall system architecture diagram](image)

Figure 2. An intelligent embedded agent architecture

Figure 2 shows the overall software architecture for the proposed system. In this architecture, each embedded agent can provide a set of information services to other agents at request. The information services are in the form of processed sensor readings or internal states. The current implementation which uses the Universal Plug and Play Protocol (UPnP) has several advantages over the previous one which used a simplified version of the DIBAL protocol [1] over socket-to-socket TCP/IP links. The main advantage is that it facilitates the detection of new agents coming into the system, agents that are leaving it and agents that break down. Another benefit is that information is multicasted to subscribed units. This uses less network traffic than peer-to-peer communication and does not infringe on user security and privacy as with broadcasts. Changes will be sent as events when they happen instead of being requested periodically. This
means that the embedded agents will run more efficiently and react more quickly. With UPnP, each agent and its services will have a brief description in the Extensible Markup Language (XML). This is useful for knowing the number and types of services an agent can provide.

The embedded agents are controlled by high level behaviours. These could be Hierarchical Fuzzy Logic Controllers (HFLC) [3] for complex agents like mobile robots, or elementary reactive rules for simple agents like bedside lamps. Each agent is connected to other agents via communication channels which provide the normalised values of the other agents’ services \(v\). The services \(v\) are scaled by weights \(w\) which determine how important each service is for activating the high level behaviour. The weights take values between 0 and 1. The activation function \(\alpha\) determine the degree at which each high level behaviour affects the effectors, \(\alpha\) can be written as follows:

\[
\alpha = \frac{\sum_{i=1}^{n} w_i \cdot v_i}{\sum_{i=1}^{n} w_i}
\]  

where \(n\) is the number of services from the other embedded agents that can be communicated with. The activation level, \(\alpha\), is the normalised sum of products of the weights \(w\) with the normalised values from other agents’ services \(v\). An activation level of zero means that the behaviour is being suppressed. If a weight for a particular service, \(w_i = 0\), it means that this service is not needed by the agent and no communicating link with the embedded agent \(i\) will be established. When all the weights are equal to zero, this means that the agent is not making use of others, and its high level behaviour would be independent of the other agents.

The weights are optimised by the multi-objective and multi-constraint genetic algorithm described in the previous subsection. In our case, all intelligent embedded agents have the same learning procedure shown in Figure 3. Once the agents are started, they search for the services that are available to them and initialise the genetic algorithm based on that. The chromosome size is directly proportional to the number of discovered services. To be able to interpret their chromosomes, they create a genome, which is a dynamic list that maps services onto genes. Each gene would represent a weight value as well as its direction. By direction we mean whether the system should take into account the value of the corresponding service itself or its inverse instead. The chromosomes can be viewed as control rules that contain information on which services to use, how to use them, and to what degree they should affect the activation level. After the initialisation process, the agents would be in running mode where they would run a random solution from the Pareto-optimal front found by the genetic algorithm. In this state, they are subscribed only to services that they need, i.e. with non-zero weights. The inhabitants of the intelligent environment would go on with their everyday lives as with any other mundane inhabited environment.

Once they express a desire to change something that is controlled by an agent, this agent would be punished and switch from running mode to learning mode. When in learning mode, agents would be subscribed to all the services described by the genome. This is when the genes would be optimised according to the objectives and constraints of the agent. Objectives are usually to maximise user-comfort and minimise energy loss and network usage. Constraints involve biasing the system towards the user-comfort objective so users would not be left in the cold and the dark. Other constraints concern preserving user and system safety.

There is a difference between the first time an agent enters learning mode and other times when it resumes learning. On the first time, the multi-objective and multi-constraint evolutionary optimisation procedure runs for a longer time to acquire a large number of diverse optimal individuals. From then on, whenever the agent enters the learning procedure, it would usually narrow down the optimal front to a subset of it. In that sense, the learning process is semi-incremental.

We envisage that an emergent functionality of a system of embedded agents would be to reduce the amount of time users interact with any form of interface, whether it is a switch on the wall or a menu running on a smartphone.
This would be particularly beneficial for the elderly and the disabled.

3 Experiments

Our architecture was implemented on a variety of embedded agents operating in distinct scenarios. Different sets of controlled experiments were performed to demonstrate the functionality of the various aspects of the system as explained in the following subsections.

3.1 iDorm experiments

We have performed many experiments in the iDorm with the aim to demonstrate how the system would operate inside a real UCE. The system was given three global objectives to optimise, which are: to minimise energy consumption and network usage, and to maximise user comfort. Network usage is a count of the number of services an agent would be subscribed to, with fourteen being the maximum number of services available to it. Energy consumption is measured by taking the magnitude of the vector sent to an actuator. User comfort is measured as follows:

\[ e_r = |a_r - d_r| \]  

where \( e_r \) is the error between the activation level and the user’s desire at a certain time instant. The user’s desire is obtained from when the user punishes an agent by expressing a desire to change something that it controls. This could be a lighting agent reading the new state of a light dimmer switch whenever it is turned.

Figure 4. Chromosome fitnesses

We have performed many experiments in the iDorm with the aim to demonstrate how the system would operate inside a real UCE. The system was given three global objectives to optimise, which are: to minimise energy consumption and network usage, and to maximise user comfort. Network usage is a count of the number of services an agent would be subscribed to, with fourteen being the maximum number of services available to it. Energy consumption is measured by taking the magnitude of the vector sent to an actuator. User comfort is measured as follows:

\[ e_r = |a_r - d_r| \]  

where \( e_r \) is the error between the activation level and the user’s desire at a certain time instant. The user’s desire is obtained from when the user punishes an agent by expressing a desire to change something that it controls. This could be a lighting agent reading the new state of a light dimmer switch whenever it is turned.

Figure 5. Number of non-dominated individuals

We have performed several experiments where a user would train a ceiling light to switch off whenever the desk lamp is on and vice versa. In this case, the local goal of a ceiling light agent is to control its light intensity. On average, the system required one resume learning stage to learn to associate the inverse of the value from the state of the desk lamp. Figure 4 shows the fitnesses plotted against each other during one of the runs. The further away a front is from the origin, the more Pareto-optimal it is. From the fifth generation until the 125th, each set of solutions is either a subset or identical to the next generation’s non-dominated front. This also applies to the generations from the 126th until the end of the learning process. The diamond marker at point (100, 100, 13) is the single solution that the system converged to. It is a subset of the front at the 1000th generation before the resume learning stage. It is more fit than it was before because the resume learning stage was when the user was training the system when to switch the light off, and before that the system was being trained when to switch the light on. Therefore, before it resumed learning, the energy preservation objective was conflicting with the user preference. Figure 5 shows how the system was gathering more Pareto-optimal solutions with every generation. The curve plummets at the 126th generation because this is when a more Pareto-optimal set of solutions was found. Other similar experiments were performed in the iDorm with different services in question.

Certain service providing agents were purposely shut down to see if the learning system will compensate for that. For example, if the user wants the ceiling light switched off when they are sitting on the desk, the system could use either the state of the desk lamp or the chair’s pressure pad. If either the chair or desk lamp agent had been previously disconnected, then the system would make use of the other service providing agent. In another experiment when the user was sitting on the bed, we have discovered that a certain light sensor in the room can be
used to detect whether someone is sitting on the bed as their shadow would fall upon it. We did not anticipate this. It entails another aim for our work: users are not only assumed to have insufficient technical expertise to be able to configure the system; they would also not know exactly which services might be useful for an agent, because they cannot see with the eyes of an agent.

### 3.2 Laboratory experiments

![Figure 6. An mDorm's evolutionary system progression](image)

We have performed other experiments in a laboratory environment in which we used a system of heterogeneous embedded agents comprised of mobile robots and mDorms. mDorms are miniature iDorms that can be used to take care of plants. The laboratory was divided into two areas defined by infrared beacons. A mobile robot and an mDorm were assigned to each area. Each robot’s local objective would be to patrol its designated area relaying sonar and beacon alignment information. If a user is detected inside a zone, then the appropriate mDorm should switch from `powerSave` mode to `lifeSupport` high level behaviours. User comfort was coded in the form of constraints on the mDorms. Figure 6 shows how one of the agent’s multi-objective and multi-constraint genetic algorithm converged to a Pareto-optimal set of solutions.

Other laboratory tests were performed to show that the multi-objective evolutionary system is capable of compensating for a faulty component. The attempts were promising as the weight of the service that was sending unreliable values ended up being zero in the set of optimal solutions.

In both the iDorm and the laboratory experiments, the system had converged within a short time interval (few minutes) to an optimal co-ordination strategy that satisfies the global and local objectives and constraints.

### 4 Conclusion

In this paper, we have presented a novel, adaptive, fault tolerant, distributed, unobtrusive and lifelong learning system for co-ordinating multiple embedded agents in a UCE. The system was able to satisfy different local and global objectives and constraints. We have carried out various experiments to measure the system performance in differing circumstances. In our future work, we will try to address how the system would be scalable to handle a large number of embedded agents.

### References


