International Transactions on Systems Science and Applications
Volume 8 ▪ December 2012 ▪ pp. 85-98
sai: itssa.0008.2012.037

ISSN 1751-1461 (print)
ISSN 2051-5642 (online)

A Cyclic Genetic Algorithm to Model Use Patterns in an Intelligent Environment

David T. Alpert and Gary B. Parker

Department of Computer Science
Connecticut College
New London, CT, USA
Email: dalpert@salesforce.com; parker@conncoll.edu

Abstract: This work involves the use of learning algorithms to model the use schedule of an academic building with the intent of intelligently controlling the environment. A sensor network of motion sensors is used to monitor and record the activity of each of the rooms in the building. After a basic preprocessing of the data, the patterns of use of the rooms are determined using a Cyclic Genetic Algorithm (CGA). The CGA has a unique ability to find repetitive cyclic patterns in data, making it ideal for such a problem. Our results show that a CGA has the ability to pick out these patterns and construct a model of the use schedule of a room.

Keywords: evolutionary computation, smart houses, behavioral modeling, ambient intelligence, genetic algorithm.

1. Introduction

Technology has become a large part of our lives and our homes are full of devices designed to make our lives easier. Simple items like a radio or bedside clock, more advanced items like a sound system or game console, and large appliances such as a refrigerator or washing machine can be found in almost any household. Technology permeates the home. Smart home technology is designed to link these individual items so that they can be used in combination to increase their usefulness. Current work in the field focuses on the automation of these features and making the home a dynamic object that can better the comfort of its inhabitants.

These intelligent environments rely on the sensors that are installed in them. Simply looking around our homes reveals that there are many sensors that benefit our daily lives. Some of them are simple as a motion sensor on a garage light or a thermostat in a heating unit, while others are more complicated like a security system with a combination of motion sensors, temperature sensors, window or door sensors, and possibly a camera. Sensor technology has advanced to the point where sensors have become more readily available and can be installed in a home more easily for less cost. These sensors each serve their own purpose, but a method to unify them and harness their power opens new possibilities.

The number of sensors and their uses in automation of simple tasks around the home is increasing. Yet, there is a limit to their capabilities when they are used individually. Linking these sensors into a sensor network and collecting the readings they produce in a central location will allow them to be used in combination and will create more possibilities for the automation of more complex tasks. For example the motion sensor of a security system is only used when it is armed and otherwise idle. If the motion sensor were connected to the thermostat in a heater, the motion detected during its idle periods could be used to determine whether a room is not occupied, and based on that information, turn down the heat to save energy.

Yet, simply networking a collection of sensors will not lead to this kind of advanced behavior. They must be connected to a central processing engine that collects the data, interprets it, makes decisions, and finally takes actions based on it. The combination of a sensor network and a central "brain" will extend the potential of individual sensors and allow them to be more useful in a living environment.

Finally, some form of logic engine would be necessary to automate the control of the environment. While an advanced preprogrammed system could achieve this, incorporating an Artificial Intelligence (AI) system would allow for full automation. The AI will make decisions based on the sensor data, allowing the system to run in the background without constant human supervision. An AI system is the most fit for this task because it has the ability to learn patterned behavior. The events in a living environment tend to exhibit patterns, as humans are creatures of habit. Through the generalization of repeated observations, AI systems are able to learn such patterns from abstract data like those collected from the sensor network.

An AI system is also crucial because of its ability to adapt to various situations as well as changes in the environment. As all living spaces and occupants are unique, it is impossible to design one ideal system that will work in any given setting. Thus its adaptability will allow an AI-based system to adjust to its unique environment and establish control patterns that are adapted to its situation.

Research on such intelligent environments, with various sensors connected to a centralized AI-based controller, has been conducted at various academic institutions around the world. Yet this is still a new area of study and there is a lot to be discovered. AI itself is also a rapidly developing field in Computer Science, and many new techniques have not yet been applied to this problem domain.

2. Related Works

In a brief overview of advances in these technologies, Cook and Das [1] describe the use of artificial intelligence to automate homes, which they refer to as smart environments. They argue for the idea that a sensor network will comprise a necessary backbone to these environments, and will in turn be connected to a centralized intelligence, which makes decisions based on information collected from the sensor network and sends commands to its controlled devices in the home accordingly. They provide several examples of such systems, one of which is based on X-10 technology. X-10 systems control devices using power line communication (PLC) by sending small disturbances along the electrical wiring, a potential advantage over other technologies, since such lines are already present in most all homes. This kind of system allows for easy setup of simple smart home environments. Using the built-in sensor network and artificial intelligence capabilities, such a system can be designed to learn the inhabitants' way of life and adjust the devices in its control to better suit their needs.

In order for a system to understand a person's needs, it must first construct a model of his/her living style and patterns. This is done through the recording and analysis of various forms of data collected by the sensor network. Youngblood et al. [2] describe a system called MavHome (Manage Adaptive Versatile environments), which attempts to model everyday human activities in a home environment, and determine which activity is taking place based on sensor data. This allows the environment to adapt to the specific needs of the occupants. They employ a Hierarchical Hidden Markov Model (HHMM) to learn and predict the activities taking place. This approach is based on the idea that humans are creatures of habit whose actions will show a certain degree of periodicity, thus allowing the HHMM to pick up on the pattern of events leading to a particular activity.

Once a reliable model for an inhabitant is developed, it can be used to make decisions and predictions about the inhabitant's lifestyle. The applications for such a model are numerous, ranging from recreational to environmental and medical. A home that has learned the various unique habits of its occupants can make decisions which enhance the occupant's life in various ways. Home comfort is one of these areas that has various applications. Perhaps on rainy days you like to come home and drink a hot cup of tea while reading a book in the chair next to the heater and listen to your favorite music. An intelligent home would have the ability to learn this pattern of events and control the different aspects of the activity based on certain triggers it receives from its sensors. On a rainy day

the hot water pot would turn on upon your arrival. The heater by your reading chair will also begin warming the room. As you walk towards your chair, the reading lamp will turn on and the stereo will select your favorite songs.

While this scenario is solely for comfort, there are other more practical applications as well. There are certain appliances in the home that could benefit from an intelligent controller with an overall knowledge of the entire home. One could imagine a situation where there are a few appliances competing for a single resource. For example, both the dishwasher and the laundry machine need to use hot water, but the water heater cannot heat water fast enough for both to run simultaneously. If the appliances were controlled by an intelligent system, it would be able to learn to run these at different times. It could also incorporate knowledge about the inhabitant's schedule and run them at convenient and unobtrusive times.

There are an endless number of such scenarios where an embedded intelligent system with access to data collected on a sensor network could increase the comfort and convenience of your home. Some other applications being researched in this field have a more significant impact on people's lives.

One use, described by Helal et al. [3], involves health monitoring for the elderly. A sensor network can be embedded into the home of an elderly person to monitor their wellbeing. The system works by first collecting data in the home and establishing a baseline for the ordinary behavior and lifestyle of the inhabitant. The system can then detect deviations from the predetermined normal behavior patterns and report them to a medical facility. This kind of technology would allow elderly people to live independently in their own homes later into their lives, and would provide peace of mind for their family members. It has also been discovered that in some situations, similar systems were able to detect slight deviations which were the symptoms of diseases, such as Alzheimer's in its early stages.

Environmental issues and natural resources are also topics of great interest. In his overview, Hagras [4] discusses the use of computational intelligence (CI) techniques in living environments to reduce energy consumption. He divides CI into the three predominant areas of fuzzy systems, neural networks (NN), and genetic algorithms (GA), and provides various examples where each is utilized.

CI models have also been used to predict energy consumption of larger environments such as office and academic buildings. Bailey and Curtiss [5] describe the use of a NN to adjust heating and air-conditioning to run efficiently and save energy. Other examples focus more on smaller personal living environments. One such system at the University of Colorado as described by Mozer [6] uses a NN to adaptively control environmental parameters such as lighting and room and water temperatures. These systems are designed to reduce energy consumption while maintaining the inhabitant's comfort level.

Another possible application for a behavior model is to use it as a basis to control different aspects of a living environment in an effort to conserve energy. If an inhabitant's schedule can be determined, the environment can learn to take actions which reduce the waste of energy. If the environment determines that the occupant is consistently not home using certain periods of the day, energy consuming appliances such as the water heater or the air conditioner can be turned off until shortly before the predicted time of his/her arrival home.

One can imagine that on occasion, the inhabitant will unexpectedly come home early or not leave the house at all and will find that the home is not at a desirable temperature or that there is no hot water available immediately. In some situations this could be a deal breaker, but in other cases the minor and often short term discomfort is a reasonable price to pay for the benefits of reduced energy consumption. It would also not be difficult to imagine a simple override mechanism that could be triggered in these situations. A more advanced intelligent system could take into consideration the level of discomfort a particular action could cause the inhabitant. At one extreme end of the spectrum, all appliances would be on constantly and comfort would be maximized at the expense of energy. At the other extreme, all appliances would be always off and energy saving would be maximized at the cost of comfort. An intelligent system would aim to find the ideal balance between the cost of unnecessary energy consumption and the discomfort of the inhabitant.

However, before any of these decisions can take place, a system needs to analyze its collected data and construct a reliable model of the inhabitant's behavior that can be used to accurately predict future actions. Thus methods for determining this model or schedule of activities are of great interest.

The goal of the work presented in this paper is to learn a schedule of use for a particular room so that it can be used as a basis for making predictions on the room's usage. If a system can make such a prediction, it can make better decisions on how to control various aspects of the environment. In some cases, say lighting control, knowing the use of a room in advance is not necessary. Simple systems linking a motion sensor to a light switch already exist, and are very effective because lights can be turned on or off almost instantaneously. Advance knowledge of use becomes more important when the aspect being controlled takes time to reach its desired state. Heating control is one such environmental variable. Because heating a room to a desired temperature takes time, the ability to predict the use of a room in advance can be extremely beneficial. Conversely, if it can be predicted how long a person will be occupying a room, the heat can be shut off and the room can begin to cool without causing much discomfort to the occupant.

While this sort of system would be extremely effective in an ideal situation, human beings do not often stick to a precise schedule and tend to act somewhat unpredictably. Thus it is important for a system to include a reactive component, which can make real-time decisions that can override previous decisions. While this work does not include a real-time component, it attempts to develop a system that can create a schedule for a reliable prediction that a system could use to bias its decisions. It will focus on the use of Cyclic Genetic Algorithms to help construct this model for prediction.

Figure 1. Strider House.

3. The Environment and Data

The environment of interest is Strider House (Figure 1), a building on the Connecticut College campus that was being used by the Computer Science Department. It housed two computer labs and three office spaces. Strider house was an ideal domain for modeling as its use was somewhat consistent and predictable, such as weekly scheduled labs and classes. At the same time its use had enough variation from students coming in to do work or other unscheduled uses, to warrant an adaptive system. Each of the rooms was configured to be monitored by a single motion sensor (Figure 2) that records any activity that takes place. Only one sensor was used per room for a few

reasons. Primarily, the hope was to get a large amount of coverage with the minimal amount of hardware. A system that requires minimal hardware and setup is easier to install and get working. Having only one sensor also reduces the complexity of the data and allows for easier learning. The sensors were strategically placed to capture as much of the motion in the room as possible.

Figure 2. X-10 motion sensor.

The motion sensors are part of an X-10 system, as described previously, and communicate with a base station by radio frequency. The motion is detected using an infrared sensor. The sensor will take readings at intervals to determine the background infrared levels. When a person enters a room, it detects the change in heat and sends a triggered signal to the base station. Data from these motion sensors are recorded on a computer linked to the base station in a simple text format. The values recorded for each trigger event is shown in Table 1. Each sensor event is labeled with a date and time stamp, signal type, and sensor ID. The sensor ID is unique for every sensor and signifies which room the data was received from.

Table 1. Sample Data.

Date	Time	Signal Type	Sensor ID
11/12/2010	09:30:00:359	RFC	b2
11/12/2010	09:30:49:734	RFC	b1
11/12/2010	09:30:50:000	RFC	b1
11/12/2010	09:30:57:562	RFC	b2

As discussed by Galushka et al. [7], raw data collected from a sensor is often too complex or abstract to perform any sort of data mining. Thus some form of preprocessing is necessary to convert it into a form that can be useful. For this work, preprocessing of the data extracted the timing and frequency of the sensor events for each hour. The system logs all sensor events in one file, so firstly, the logs were sorted by room. Next, the time lapses between sensor triggers were calculated. Based on these values an initial cleaning of the data was conducted. An assumption was made that a person moving in a room would trigger the sensor more than once per motion. Thus, if the time between a trigger and its adjacent following trigger was longer than a predetermined threshold, it would be discarded. Once the thresholds for the particular sensor network are determined on an initial data set, they can be used for live analysis. Discarded trigger events were often false triggers or noise in the system.

After the data was cleaned, it was sorted into hour blocks. Each hour block was then classified as “in use” or “not in use” using a set of thresholds. These thresholds were determined manually on a set

of control data where the activity state of certain hours was known. These steps were conducted for each room. The thresholds were determined overall and not on a per room or per hour basis. The final "use status" for every hour in every room was recorded in an easily accessible data structure. The learning algorithm used this data for its analysis.

4. Learning Method

The learning method employed in this work is a form of Genetic Algorithm (GA) known as a Cyclic Genetic Algorithm (CGA). The traditional GA, introduced by Holland [8], is a learning algorithm based on the concepts of heredity and the survival of the fittest. It consists of a population of chromosomes, which are possible solutions to a problem. These chromosomes are often binary strings of fixed length that represent the solution in some way. Each one is assigned "fitness" as to how well it solves the problem. Based on this fitness, the chromosomes of the population undergo the three genetic operators – selection, crossover, and mutation – to create a new generation of chromosomes that theoretically are better solutions. Through this process an optimal, or near optimal solution is learned.

In our work we use a CGA to capture the use patterns of the environment. The cyclic repetitive nature of the data we are working with makes the CGA an ideal candidate.

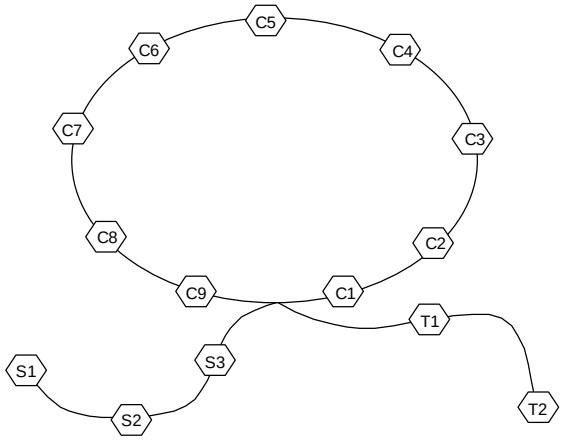

Figure 3. CGA States.

4.1 Cyclic Genetic Algorithm (CGA)

The CGA, as discussed by Parker [9], is unique in that it incorporates time as a factor where each gene is a task and the chromosome represents a sequential solution that is to be carried out linearly in a given span of time.

Figure 3 displays an example of the use of a cycle to form a longer set of tasks. Although the iterative section only has nine possible tasks, the section is run repeatedly a predetermined number of times to achieve the full execution length. The figure also depicts the start and stop sections that may or may not be used based on the use case.

In his work, Parker uses these CGAs to generate an ideal walking gait for hexapod robots. The cyclic portion of the chromosome is ideal for the repetitive nature of this problem. While the robot is walking, the control sequence of the legs must loop through the same commands repeatedly. The cyclic portion of the chromosome learns to represent this pattern, which during execution can be repeated the necessary number of times.

In our work, we use the CGA to pick out the cycles in the use patterns of a room over an extended period of time. The type of environment we work with has a tendency to be used with

regularity, thus creating a pattern that can be detected. For instance a computer lab may be used for a class on Wednesday afternoon every week, while an office space is used in the morning twice a week on Tuesday and Thursday. In these cases, the shortest repeating pattern is a week long. Thus, with this kind of periodicity, our hope is for the CGA to pick a cycle length equal to a week. Since the same use patterns are repeated weekly, the algorithm only needs to learn the week schedule and then repeat it weekly. Although in this particular environment we know that the length of the cycle of use is a week, in other environments where the cycle length is unknown, the CGA has the ability to pick out a cycle of other lengths such as a two or three day cycle. If an office were used every other day, the cycle would be two days. Its ability to pick out the cycle with minimal a priori knowledge is what makes the CGA a powerful tool for this application.

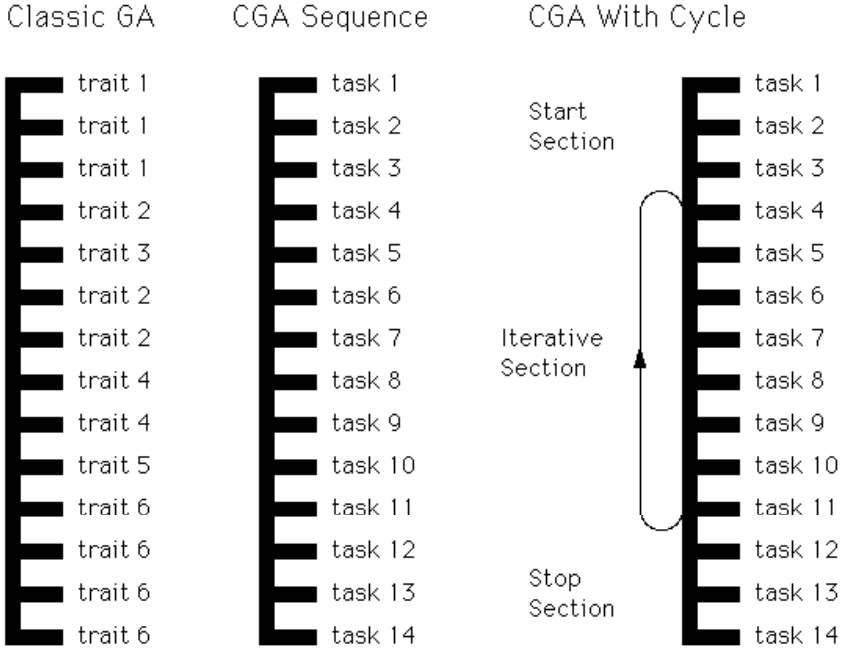

Figure 4. GA and CGA Chromosomes. The genes of the classic GA chromosome (shown on left) typically represent traits of the solution. The CGA chromosome (shown in middle) genes represent tasks to be completed in a sequence. Some portion of the CGA chromosome (shown on right) can be sectioned off to be an iterative section.

4.2 CGA Chromosomes

A CGA chromosome differs from a regular GA chromosome in that its genes represent tasks as opposed to traits of the solution and it has a cyclic portion that is designed for repetition. Figure 4 shows such a chromosome. In the classic GA, sets of bits can be grouped together to represent traits of the solution – for example, the connection weights for an artificial neural network. In the CGA, sets of bits are grouped to represent tasks that are to be completed in a fixed amount of time – for example, signals to servomotors every 25 ms or instructions in a segment of programming code. In this way, the CGA chromosome represents a sequential program that can be executed in order: task 1, then 2, etc. In addition, we can designate some portion of the chromosome as the iterative section and repeat the tasks; creating a cycle. This iterative section is repeated for a set number of times or until some stopping condition is met. The start and stop sections only execute once, but can be used to set up before and after the cycle.

This allows the chromosome to represent a program that has a start section, an iterative section, and a stop section. The start section can contain tasks required to initialize while the iterative section can contain the tasks that are required in a continuous cycle. The trailing stop section can be added to effect a smooth transition back to the at rest state. Each gene, representing a task, can be as simple as a primitive command or as complicated as a subprogram.

In Parker's work with hexapod robots, the start section represents the motions to bring the legs into position to take the first step, and the stop section represents the motions to bring the robot to a standing position. The motions that make the robot walk are represented in the cyclic section, as this part is repeated for each step the robot takes.

A CGA chromosome can also be designed to not use a start or stop section. The bulk of the chromosome is used as a cyclic portion. In our work, we take this approach.

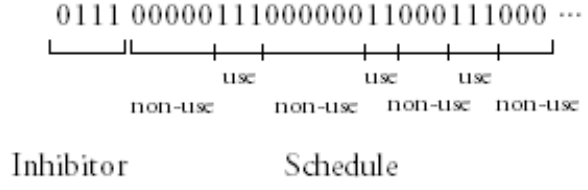

Figure 5. Chromosome representation for learning room usage.

For the CGA used to solve the problem in this work, each chromosome (Figure 5) consists of a 4 bit inhibitor (as described in the next section) and 256 bits to represent the schedule. A chromosome of size 256 bits was chosen because it is long enough to represent 10 days worth of use. This allows the algorithm to pick out cycles of varying lengths ranging from a single day up to ten days. The portion of the chromosome beyond the length of cycle is disregarded. Each bit in the schedule portion represents the use of a particular hour in the schedule, where a 1 stands for a period of use and a 0 for a period of non-use. The chromosome in Figure 5 represents a schedule of five inactive hours, followed by three active hours, followed by six inactive hours, and so on and so forth. The down side to this representation is that, representing cycles longer than a week, for instance a bi-weekly schedule, requires a large number of bits and can slow the computation greatly. While other methods which represent the hours of use in a reduced format were attempted, the full representation proved to be the most effective. Although a CGA can work with both fixed and variable length chromosomes, for the purposes of this work, we only use fixed length chromosomes.

4.3 Inhibitors

The inhibitor portion of the chromosome is used to limit some aspect of its function to prevent it from developing in a particular way that does not suit the problem. In the original application of CGAs to gait generation, inhibitors were used to inhibit designated legs from lifting off the ground at the same time.

In our work, we use the inhibitor to limit the length of the cyclic portion of the chromosome. While the 256 bit chromosome has the ability to represent a schedule just over 10 days long, and will have data for that full length, the algorithm must learn the correct cycle length, which is most often shorter than 10 days. Thus, the value of the inhibitor represents the length of this cycle in days, allowing it to pick shorter cycles. It is a 4 bit binary number, as seen in Figure 5, which is initialized as a random value and is learned over the generations by the algorithm.

When calculating the fitness of a chromosome for selection (as described further in the next section), the value of the inhibitor, in days, is multiplied by 24 to convert it to hours. This value then determines the number of bits from the remaining part of the chromosome that will be used for the cyclic portion. For instance, if the inhibitor has a value of 7 days, the length of the cyclic portion will be 168 bits, which is equal to the number of hours in 7 days. This 168 bit portion is then compared to

the training data to evaluate its accuracy. This portion is repeated as many times as necessary to match the length of the data. If the cycle in the data is actually 7 days, it will have a higher probability of matching and returning a high fitness value. If the cycle is different, it will be less and less accurate and return a poor fitness.

We use days as a unit of cycle length based on the assumption that a cycle should not be shorter than a day. Making this limitation also allows the algorithm to pick out a schedule more effectively without getting confused by minor periodicities in the data.

4.4 Genetic Operators

The three typical operators for both GAs and CGAs are selection, crossover, and mutation. Selection is the means by which parents are chosen to produce offspring. This is usually done several times in the process of producing a new population. Crossover is the method of recombining the two parents to produce an offspring. Mutation is a low probability operator that simulates a mutation taking place in the chromosome of the offspring.

The probability for selection was calculated based on fitness, which is a function of how well the proposed schedule matches the actual data. To determine fitness, the cyclic portion of the selected chromosome is lined up with the training use schedule so that each hour is compared and evaluated. The scoring of correct and incorrect hours is determined by the reward scheme shown in Table 2.

Table 2. Fitness Rewards.

Actual	Learned	Reward
1	1	x
1	0	-1
0	1	-1
0	0	0

In observing the training data it was clear that the number of active hours is highly disproportionate to the number of inactive hours. This causes a problem because, in a simple scoring scheme, a schedule of always inactive scores very well. To counter this, the reward for a correct prediction was weighted to compensate, as denoted by the x in Table 2. This weighting was the proportion of inactive to active hours in the data and was adjusted for every training set to match the proportion in that data. For instance, if the ratio of unused hours to used hours in the data is 10:1, the reward x for a correct use prediction would be 10 points. This scheme, in combination with the -1 reward values for incorrect values, ensured that both the always inactive and always active solutions, on average, would score 0 points.

Once fitness was calculated, the method known as elitism was used, where the highest scoring individual of a population carries directly on to the next, to ensure positive overall fitness growth. All other individuals of the new population are a result of a crossover between two mates stochastically selected where the probability of selection was proportional to the fitness.

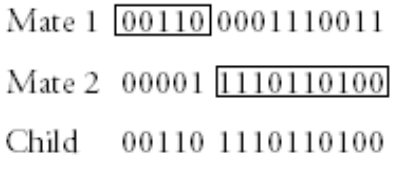

Figure 6. One point crossover.

Two types of crossover, single point and universal, were used with equal probability. Single point crossover involves picking a random point along the length of the chromosome and splitting it into two parts, as seen in Figure 6. The resulting chromosome will consist of the first part of one mate and the second part of the other. The order of the mates is selected randomly to prevent biasing towards one of the mates. Universal crossover is a much simpler process where every bit in the chromosome, including the bits in the inhibitor, has an equal chance of being selected for the new chromosome. Finally, the new chromosome that was created as a combination of its two parents undergoes a basic mutation scheme where every bit in the new chromosome has a 0.3% chance of being flipped.

5. Tests

In a GA, the group of chromosomes – representing possible solutions – that are evolved and mated with each other are referred to as a population. Each repetition of mating chromosomes to produce a full population is known as a generation. In our work, training was conducted for 10,000 generations using populations of size 320. The initial population before learning was created with randomly generated chromosomes where each binary bit was chosen at random.

Training was conducted on both simulated and real data. Two different types of simulated data were used. These are explained in further depth below. The real data used in this work was collected over a period of weeks during normal use of the building. It was during a school semester and the rooms were used frequently on a regular basis. Each test of 10,000 generations was repeated five times.

5.1 Type I Simulated Data

The first type of simulated data, which will be referred to as Type I, was constructed by repeating a single week of real data ten times. This was done to simulate a ten week period where the use schedule was identical from week to week. The Type I data is designed to test the algorithm on an ideal environment where use is consistent, and an accurate schedule could be determined.

5.2 Type II Simulated Data

The second type of simulated data, Type II, was constructed based on a single week of real data. In this case, the second week of data was created by slightly modifying the first week. This simulates a second week with some minor schedule changes. The modified second week was then adjusted further to create a third week. These minor modifications to the use schedule across three weeks simulates a progression of change from week to week while maintaining an overall consistent schedule.

If the algorithm can learn to develop a schedule which best fits the three weeks, it can then be used to predict a fourth week. As more data is collected, a system can continuously train the algorithm on the previous three weeks to make predictions for the current week. An advantage to this method is that the system can adapt more easily to changes in a schedule. While a period of three weeks was chosen for this test, any number of weeks could theoretically be used.

The three simulated weeks were then repeated three times to create 9 weeks worth of data to train the algorithm. This was done to increase the size of the training set. A longer training set increases the variation between the fitness of good and bad solutions. A poor solution with an incorrect cycle will score worse and worse as it is evaluated against more data.

5.3 Varying Cycle Lengths

To test the flexibility of the algorithm on other cycle lengths, similar Type I and Type II data were also constructed to simulate a three day cycle. In this case, three days worth of real data were used.

5.4 Real Data

Finally, experiments were conducted on three consecutive weeks of real data. Similarly to the way in which the simulated Type II data was tested, the three weeks of data were repeated three times to create a training set with a total length of nine weeks.

6. Results

The CGA was successful in determining the use schedule in all five tests.

6.1 Seven Day Cycles

The results from the 7-day simulated Type I and Type II experiments show that the CGA is effective in learning a 7-day schedule of use. Figures 7 and 8 show the fitness growth at selected generations averaged over five trials for the 7 day cycle. The scale for fitness is shown on the y-axis on the right of the figure. The average fitness increases dramatically in the first 100 generations, and continues to grow until about 8000 generations. Note that the x-axis scale on the graph changes from every 100 to every 1000 after the first 1000 generations. The standard deviation between the five trials is shown in the error bars. The y-axis on the left side of the figure shows the scale for these bars.

In all five trials, the algorithm consistently determined the correct cycle length of 7 days for both the Type I and II tests. The use pattern is matched exactly in the Type I test. This shows that the algorithm can learn a schedule exactly, if it is perfectly repetitive. As seen in Figure 7 by the standard deviation bars, this occurred in all five trials by the 8,000th generation.

The deviation for the Type II tests, seen in Figure 8, also drops to zero by the 10,000th generation. This shows that the algorithm consistently produced the same result in all five trials. It is not possible for the learning system to develop a weekly schedule that matches all three weeks in the Type II tests, since there is some variation from week to week in the data. The hope is for the algorithm to pick out the schedule that best fits all three weeks with minimal error. The fact that all five trials produced the same schedule suggests that it is optimized. Direct comparison of the data and the produced schedule supports this as well.

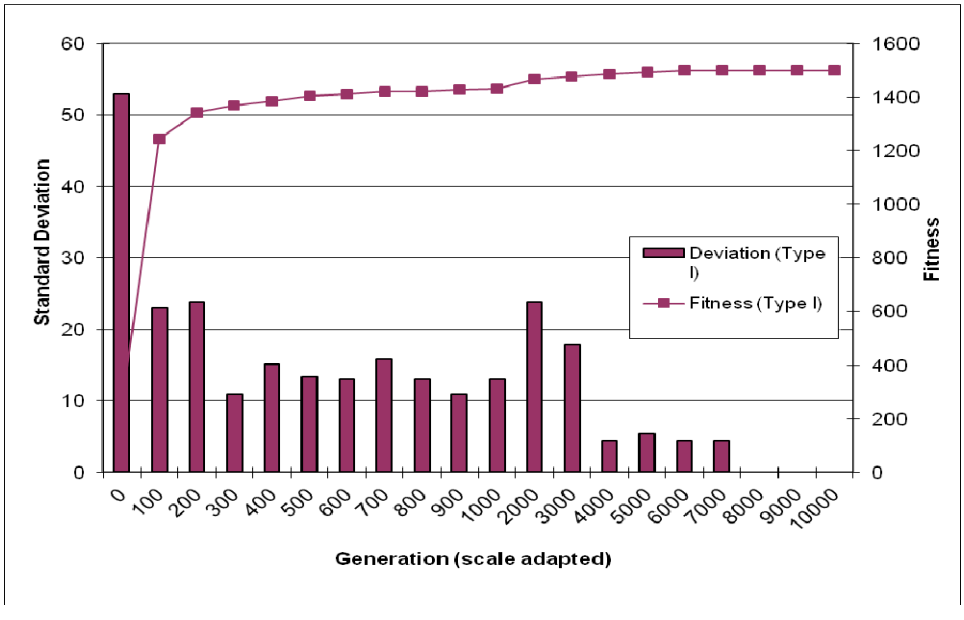

Figure 7. Fitness growth for simulated 7 day cycle (Type I).

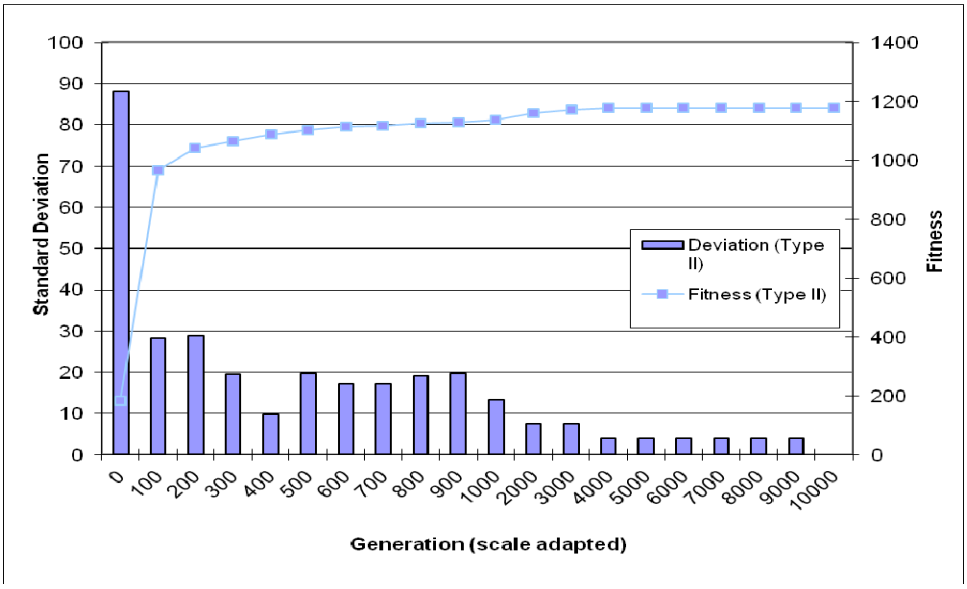

Figure 8. Fitness growth for simulated 7 day cycle (Type II).

6.2 Three Day Cycles

Similar to the test on the 7 day cycle data, the algorithm is able to consistently determine a 3 day schedule for both Type I and Type II tests. These results are shown in Figure 9 and 10. Note that the scale for the x-axis in this Figure is differs to those of Figures 7 and 8. The CGA runs significantly faster on the data for a 3 day cycle and thus learns the pattern in under 1000 generations. The greatest growth in fitness occurs in the first 200 generations, and thus the scale of the figure is adjusted to show this growth in more detail. The values for fitness shown in this graph are again an average of five trials.

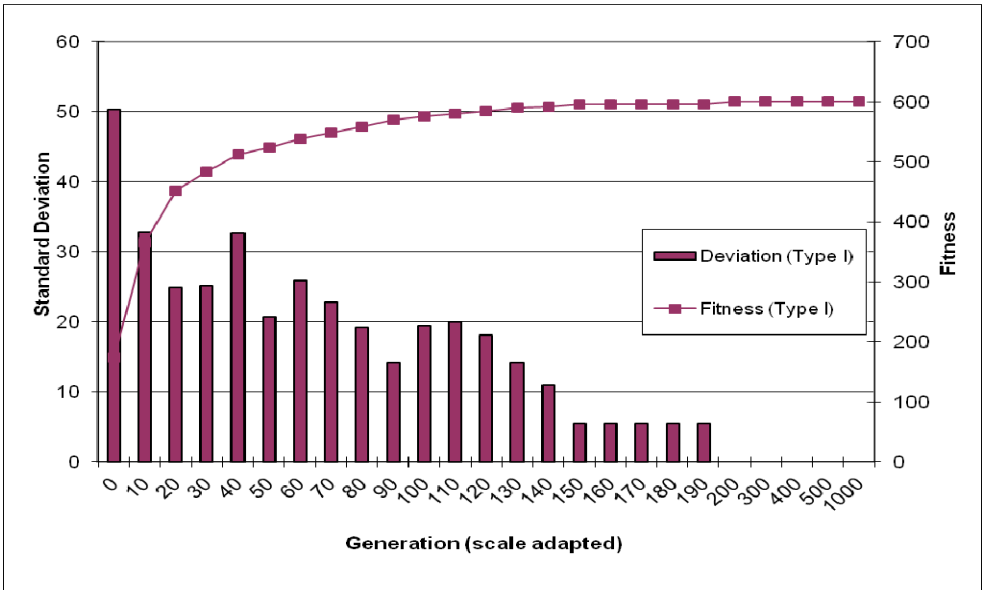

Figure 9. Fitness growth for simulated 3 day cycle (Type I).

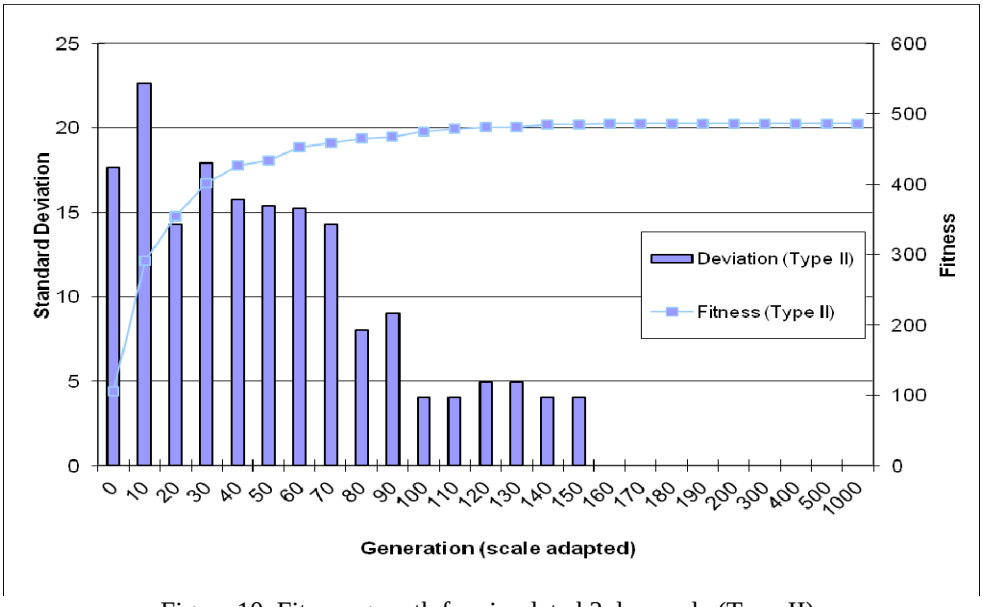

Figure 10. Fitness growth for simulated 3 day cycle (Type II).

In all five trials, the algorithm consistently determined the correct cycle length of 3 days for both the Type I and II tests. Like the previous tests on the 7 day cycles, the use pattern is matched exactly in the Type I test in all five trials by the 200th generation. The deviation for the Type II tests also drops to zero by the 160th generation.

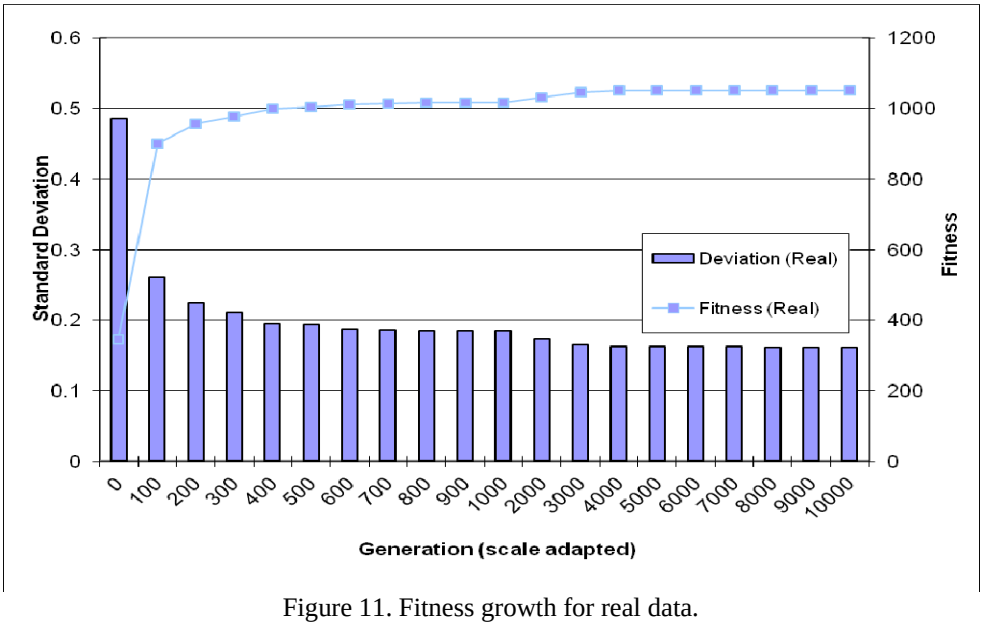

Figure 11. Fitness growth for real data.

The CGA can pick out the exact use schedule for both 7 and 3 day cycles in the Type I tests. On Type II data, the CGA consistently picks out the correct cycle length and produces the best fitting schedule given the variation in the data. These positive results on both the 7 day and 3 day schedules shows that the algorithm has the ability to pick out the correct length of the cyclic pattern with no a priori knowledge.

6.3 Real Data

The experiments on real data show that the CGA can be effective in picking out the ideal schedule. Figure 11 shows the fitness growth across generations for real data averaged over 5 trials. Note again that the x-axis scale on the graph changes after the first 1000 generations.

In three out of five trials, the algorithm correctly determined the cycle length of 7 days and minimized the error. The inconsistency in determining the schedule arises from the fact that real data has significantly more variation from week to week in comparison to the simulated data. Because of this, the perfect single week schedule that fits the data is somewhat ambiguous in itself. More than one pattern of use could also have been in play, which made even the cycle length in question. Nevertheless, the CGA produced seven day cycles that appeared to be reasonable fits for the actual data.

7. Conclusions and Future Work

Our work employs a Cyclic Genetic Algorithm to pick out patterns in the occupancy of a room and construct a schedule that can be used to make future predictions of its use. The results show that this algorithm performs extremely well on two different types of simulated data. It also has the ability to pick out cycles of various lengths and thus does not require any a priori knowledge of the length of the patterns in the data.

We also show that the algorithm performs well on data collected from a real environment. Although the accuracy of the determined schedule is highly dependent on the level of variation in the data, the algorithm does effectively discover a schedule that minimizes the error.

Future work will further analyze the variation in the real data in order to adapt this method and design an algorithm that can prioritize recent data over old. We also hope to incorporate this work into a larger control system as a basis for planning as well as real-time decision making with an aim to control variables in a real environment.

References

[1] D. Cook and S. Das, How smart are our environments? An updated look at the state of the art, Journal of Pervasive and Mobile Computing, Vol. 3, No. 2, 2007, pp. 53-73.

[2] G. M. Youngblood, D. J. Cook, and L. B. Holder, Managing adaptive versatile environments, Journal of Pervasive and Mobile Computing, Vol. 1, No. 4, 2005, pp. 373-403.

[3] S. Helal, B. Winkler, C. Lee, Y. Kaddourah, L. Ran, C. Giraldo, and W. Mann, Enabling location-aware pervasive computing applications for the elderly, Proc. 1st IEEE Pervasive Computing Conference, Fort Worth, TX, 2003, pp. 531-538.

[4] H. Hagras, Employing computational intelligence to generate more intelligent and energy efficient living spaces, International Journal of Automation and Computing, Vol. 5, No. 1, 2008, pp. 1-9.

[5] M. Bailey and P. Curtiss, Neural network modeling and control applications in building mechanical systems, Proc. International Conference of Chartered Institution of Building Services Engineers and American Society of Heating Refrigeration and Air-conditioning Engineers, London, England, 2001.

[6] M. C. Mozer, The neural network house: An environment that adapts to its inhabitants, Proc. American Association for Artificial Intelligence Spring Symposium on Intelligent Environments, Menlo Park, CA, 1998, pp. 110-114.

[7] M. Galushka, D. Patterson, and N. Rooney, Temporal Data Mining for Smart Homes, Lecture Notes in Computer Science, 4008, 2006, pp. 85-108.

[8] J. H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, 1975.

[9] G. B. Parker, Generating Arachnid Robot Gaits with Cyclic Genetic Algorithms, Proc. Third Annual Genetic Programming Conference, 1998.