

**Coevolutionary Approach to Robotics-Tracking and Obstacle
Avoidance**

*Dissertation submitted to Jawaharlal Nehru University, in partial
fulfillments of the requirements for award of the degree of*

Master of Technology

In

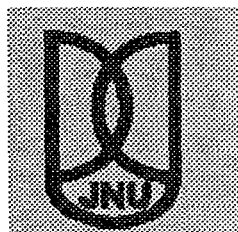
Computer Science and Technology

By

ASHISH KUMAR VERMA

UNDER THE GUIDANCE OF

PROF. K. K. BHARADWAJ



SCHOOL OF COMPUTER & SYSTEMS SCIENCES

JAWARLAL NEHRU UNIVERSITY

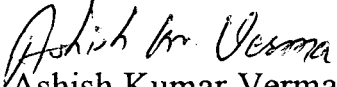
NEW DELHI -110067

January 2004

CERTIFICATE

This is to certify that the dissertation entitled “**Coevolutionary Approach to Robotics-Tracking and Obstacle Avoidance**” being submitted by **Ashish Kumar Verma** to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in Computer Science & Technology, is a bonafide work carried out by him under the guidance and supervision of **Prof. K. K. Bharadwaj**.

The matter embodied in the dissertation has not been submitted for the award of any other degree or diploma.


Ashish Kumar Verma

(Student)

 5-01-09
Prof. K. K. Bharadwaj

(Supervisor)


Prof. Karmeshu

Dean, SC&SS

Jawaharlal Nehru University

New Delhi-67

**Dedicated to
My beloved
Parents**

ACKNOWLEDGEMENTS

I would like to pay obeisance of the feet of my parents for their blessings that are always with me in all my aspirations including my academics.

I would like to sincerely thank my supervisors **Prof. K.K. Bharadwaj**, School of Computer and Systems Sciences, Jawaharlal Nehru University for the help, encouragement and support extended by them in successful completion of this project. Their ideas were innovative and the valuable discussions we had were very much helpful in keeping the project on the right track.

I would like to record my sincere thanks to the Dean Prof. Karmeshu for providing the necessary facilities. I extend my sincere gratitude to my lab mates for their continuous academic as well as morale support through out my dissertation work.

Last, but not the least, I take this opportunity to thank all the faculty, members and friends for their, help and encouragement during the course of the project.

ASHISH KUMAR VERMA

CONTENTS

1. INTRODUCTION	1-2
2. LEARNING	3-12
2.1 Learning at a Glance	3
2.1.1. Classification by Mitchell	4
2.1.2. Classification Based on Environment	6
2.1.3. Classification Based on Algorithm	7
2.2 Coevolutionary Learning	7
2.2.1. Coevolutionay Layout	9
2.2.2 Credit Assignment	11
3. PATH TRACKING AND OBSTACLE AVOIDACE SYSTEM	13-37
3.1. Genetic Algorithm Overview	13
3.1.1. Evaluation Functions	13
3.1.2. Basic Genetic Operators	15
3.2. Zeigler Adaptive System	18
3.2.1. An overview of Zeigler adaptive system	19
3.2.2. A Detailed Structure of Zeigler adaptive system	20
3.2.3. Payoff Allocation	22
3.2.4. Combination Algorithm	27
3.2.5. Weeding Algorithm	28
3.3. Modified Zeigler System	30
3.4. Path tracking and obstacle avoidance system	31
3.4.1. Credit Assignment	32
3.4.2. Conflict Resolution	33
3.4.3 Subcomponents to be Evolved	33
3.4.4 Evolution of Subcomponents	34
3.4.5. Merging the Subcomponents	36

4. IMPLEMENTATION AND RESULT	38-51
4.1. Description of the Simulation Domain	38
4.1.1. Navigation Path	38
4.1.2. Sensors	39
4.1.3. Effectors	41
4.2 Performance Components	42
4.2.1 Knowledge Representation	42
4.2.2. Production System Cycle	44
4.2.3 Merging the SubComponents	44
4.3. Learning Module	46
4.3.1. Credit Assignment	46
4.3.2. Genetic Algorithms	46
4.3.3. Genetic Operators	47
4.4. Experimental/Simulation Result	48
4.4.1. Rules Evolve	49
4.4.2. Performance Measure.	51
5. CONCLUSION AND FUTURE SCOPE	52
References	53

ABSTRACT

Coevolutionary approach encourages the formation of niches representing the simple sub-behaviors. Evolutionary nature of each sub-behavior can be controlled independently, which may evolve the complex behavior without any intermediate step. In this approach, multiple instances of genetic algorithms (GA) are run in parallel, each instance of which evolve one species of individual which are good at particular task. The trainer does not enforce these areas of expertise but rather they evolve through seeding of population. Once the populations are seeded, the evolution of complex behavior proceeds without any human intervention. GA has the capability to solve the technical problem as well as it can provide the simplified scientific model that can answer questions about the nature. Evolutionary paradigm has been proposed to encourage the emergence of niches and species in single population in all these the individual niche competes for the allocation trials.

A cooperative coevolutionary algorithm consists of several evolutionary algorithms instance. Each attempts to evolve species, which are seeded for a particular sub function. Representatives from each sub-populations is selected and are merged together to form a collaborative solution for achieving the required complex behavior. Credit is computed in term of how well each representative cooperates and accordingly feedback is given to each individual species. This helps each individual in evolving better rules.

Rule based adaptive expert systems create and modify the rules sets over the time. The main disadvantage of adaptive rule based system is only best rules are being used always and system does not takes risk to improve over time.

Zeigler provides an elegant solution for the above problem in which the system alternates two rules selection strategies from one generation to the next. During first generation, the best rules are selected while in the second generation the experimental sets of rules are selected. In this approach, the system time is broken into generations.

The main objective of this dissertation work is to coevolve the sequence of decision rules using the Zeigler model, which may lead to successful navigation of the robot. The robot must be able to track the path of an object while avoiding the obstacles.

CHAPTER- 1

INTRODUCTION

Learning of a complex task has always been very difficult to achieve. A technique like “shaping“ [degaries etc page 1 potter] in which a complex problem is divided into a number simpler subtask. These subtasks are easy to learn and are used as building blocks of the complex task. But this technique has a disadvantage that it needs a lot of human support. A proper decomposition of task into subtask mainly depends upon the trainer or the designer. Also an adaptive expert system as presented by the Zeigler [90] provides an elegant way to represent the knowledge by a set of rules.

In Zeigler [3] model the system alternates two rules selection strategies from one generation to the next. During first generation, the best rules are selected while in the second generation the experimental sets of rules are selected. In this approach, the system time is broken into generations and system in a loop.

A slightly modified Zeigler model can be used for evaluating the rules and weeding out the low performer rules. If we use the coevolutionary approach while applying the modified Zeigler model on it, very interesting and useful results are expected.

This work considers two major aspects of the robot navigation namely path tracking and obstacle avoiding. To achieve this task the simulated environment has been used in such a manner so that it consists of dense amount of obstacles and a random moving object and robot has to track the object and while doing so it should not collide with obstacle.

Coevolutionary approach [potter] permits to evolve the sets of subtasks. After merging these subtasks more complex task can be accomplished. The evolutions into sub tasks do not need explicit decomposition of a complex task. Only initial seeding is needed in order to evolve the subcomponents.

The main objective of this dissertation work is to evolve the sequence of decision rules using modified Zeigler adaptive system, which may lead to successful navigation of the robot. To achieve this task two modifications have been made in the Zeigler adaptive system. The first one creates extended problem space and another one modifies the rule representation method. The robot must be able to track the path of an object while avoiding the obstacle at the same time.

Organization of Dissertation: chapter 2 contains concept of coevolutionary learning. Basic frame work related with coevolution lies in this section.

Chapter 3 has discusses the technique used to evaluate the rules and. This chapter creates the background for the implementation.

Chapter 4 and 5 contains the implementation details, conclusion and further extension of this work.

CHAPTER-2

LEARNING

2.1. Learning at a glance

An intelligent system has the capability to adapt and learn. Learning system is not only concerned with the academic but also it is being used in the different areas. Different authors have different views about the learning.

According to Nerendra and Thachar “**Learning is the ability of system to improve their response based on the past experience**” [1]. Michalski, Carbonell and Mitchell define the learning from more cognitive view: “**Learning is processes include the acquisition of new declaration knowledge, the development of motor and cognitiv skill through instruction and practice, the organization of new knowledge into general effective representation and discovery of new facts and theories through the experiment and observation**”[2]. According to a psychologist Simon: “**Learning denotes change in system that are adaptive in the sense that they enable the system to do same task drawn from the same population more efficiently and effective the next time.**”[3].

The reinforcement learning is formally defined as :

Let we have two sets : A , the set of states of environment and b the set of actions the learning system can perform on the environment. The learning system is defined as function $\psi = M(\Phi, t)$ where $\psi \in B$ the action the learning

system performs., $\Phi \in A$ the state of the environment⁵ and $t \in \mathbb{N}^+$ at which this happens.

The reaction of environment is defined by function $p = F(\psi, \Phi, t)$ where $p \in \mathbb{R}^+$ the payoff of the environment and. This measures how good the action of the learning system was.

The higher p is better the learning.

The system is said to be learning if following holds true.

i.e.

$$\{F(M(\Phi_t, t), \Phi_t, t)\} > \{F(M(\Phi_{t-1}, t-1), \Phi_{t-1}, t-1)\}$$

Here $\{ \}$ is representing the expected value. Variables Φ_t and Φ_{t-1} , are the states of the environments at time t and $t-1$ respectively.

Ways of learning: There are various classifications are available about the Ways of learning

2.1.1. Classification by Mitchell

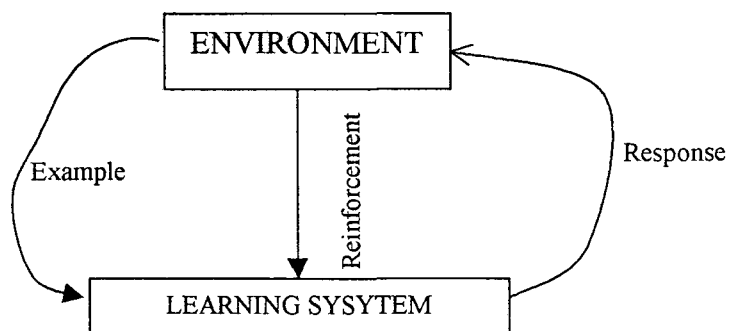
Rote learning and direct implementation of knowledge it is like the education in the primary schools. It needs directly inserting the knowledge in the intelligent system either by programming or putting it into the database system only stores and reproduce them.

Learning from instruction: It is somehow more interesting and in this learning the system must be able to understand the instruction it gets,

store it and integrate it with existing knowledge. But here learning is not much involved.

Learning by analogy: It gives learning system more difficulties. It has to find in its knowledge similar to the task to be learnt and change the knowledge already present until it is applicable to the present situation. Then it has stored this newly generated knowledge.

Learning from example : In this learning the system is presented with samples or examples from an environment together with information to associate with this example. This information can be an indication that whether an example is positive or negative whether the response of the system was good or bad or it can be some action to associate with the example. If the information is given at the same time as the example is called “true learning with examples”. If the information is given after the system has generated a response, it can be called reinforcement learning.



Learning from observation and discovery or unsupervised learning: It is a kind of learning in which the learning system is left on its own to explore its environment and try to make classification of phenomenon. It sees or form theories about it.

2.1.2. Classification based on the environment

Classification of learning system can also be based on the kind of task or environment it learns to operate. The environment can be static or dynamic, a deterministic or stochastic and discrete or continuous.

Static environment doesn't change during the time a learning system is active, while a dynamic environment may change.

A deterministic environment always gives the same response in the same situation and it never gives examples that are flawed. Stochastic environment doesn't have these properties. They can have inherent stochastic features, (as in quantum – dynamic system, gambling problem or prediction of real –life phenomenon, like the weather) or be troubled by noise.

Finite environment consists of a finite number of actions a learning system can take and a finite no of response it can get. While in a continuous

environment both i.e. action a learning system can perform and response it gets can be infinite.

2.1.3. Classifications based on Algorithm

The last distinction between learning systems will mention is the algorithm they use to learn with. The least precise way to make the distinction is to divide them by the line which divides all AI-Research: the symbolic /sub-symbolic distinction. Winston gives a more precise list of different methods. They includes learning by analyzing difference, learning by managing multiple models, learning by correcting mistakes, learning by recording cases, learning by building identification trees and learning by explaining experience. Most of these are symbolic methods, concerned with noise-free static environments. Two other methods of learning are learning by training neural nets and learning by simulating evolution, both sub symbolic and more applicable to changing environment with noise.

2.2. Coevolutionary Learning

In a coevolutionary approach the learning of complex behaviors is achieved by sets of sequential decision rules [4]. This approach requires much less human intervention than the shaping[5] approach. In this approach the multiple instances of genetic algorithms are run in parallel. Each instance of which evolves a collection of interbreeding individuals, which are expert in particular area. This area of expertise is not designed explicitly rather it

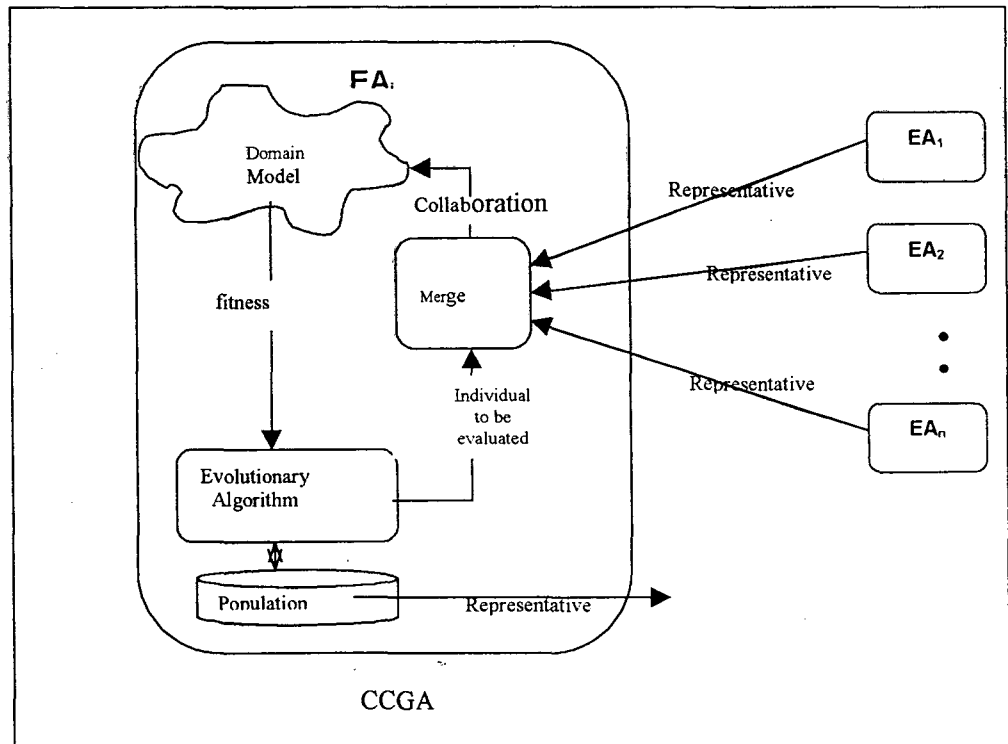
evolves without any human intervention. Only initial seeding is required in order to achieve the desired subbehavior. Seeding provides the very general behavioral rules covering the respect area of expertise. Whereas other techniques like "shaping" are used to construct complex behaviors in stages by breaking them down into simpler behaviors which can be learned more easily, and then using these simpler behaviors as building blocks to achieve more complex behavior [5][6][7]. In these techniques trainer plays a vital role in the decomposition of the complex tasks into simpler subtasks, in the training of the organisms on the subtasks and in the synthesis of the desired behavior from the subtasks. In the case of robot learning it is frequently desirable to minimize such heavy dependency on trainers [4].

In coevolutionary learning, once seeding is done the sub-behavior evolves without any human intervention. More complex task is accomplished by selecting the representatives from each species and merging them together. These composite agents are called "collaboration". After merging the rules together the "collaboration" is evaluated and credit feedback to the individual subcomponents that reflects how well the representatives are cooperating or collaborating with the other representatives. This feedback is then used by the GA instance to evolve better individuals. Such systems may be called as Cooperative Coevolutionary Genetic Algorithms (CCGAs) [4].

2.2.1. Coevolutionary Layout

In order to evolve a complex behavior explicit decomposition of complex task into subtask is needed. But this is very difficult to decide the no of modules needed and there exact nature. But an evolutionary paradigm is that in which such subcomponents "emerge" rather than being hand designed [6].

The use of multiple interacting subpopulations has also been explored as an alternate mechanism for coevolving niches using the so-called island model[9]. In the island model a fixed number of subpopulations evolve competing rather than cooperating solutions. In addition, individuals occasionally migrate from one subpopulation (island) to another, so there is a mixing of genetic material. The previous work that has looked at cooperating rather than competing subpopulations has involved a user-specified decomposition of the problem into species [4].



Cooperative coevolutionary algorithms (CCAs) combine and extend ideas from these earlier systems in several ways. In the figure above, a CCA consists of a collection of evolutionary algorithm (EA) instances, each attempting to evolve species which are useful as modules for achieving higher level goals.

Complete collaborative solutions are obtained by assembling representative members from each of the subpopulations. Credit assignment within each EA instance is defined in terms of the fitness of the collaborations in which its subpopulation members participate. This provides evolutionary pressure for the species from separate subpopulations to cooperate rather than compete. However, competition still exists among individuals within the same subpopulation[4].

Unlike the island model, the individuals from the separate subpopulations do not migrate; therefore, interbreeding does not occur. This lack of migration eliminates haphazard and often destructive recombination between dominate species once niches are established[4].

Mechanisms exist for seeding initial subpopulations with a bias towards evolving certain kinds of species. However, once evolution begins there is nothing to prevent the roles of species from changing considerably. In addition, the general coevolutionary model places no restrictions on the number of subpopulations. In some domains it may be desirable to allow the

dynamic creation of new subpopulations and the elimination of unproductive ones (i.e., the birth and death of species).

In cooperative coevolutionary learning the evolution of each subpopulation is handled by a standard GA.

2.2.2. Credit Assignment

For credit assignment and weeding out the rules we will use the modified Zeigler model [10] of adaptive rule based expert system, described in the next chapter. Since knowledge is represented in the coevolutionary learning as a sequence of decision rules. In Zeigler model rules are represented in the form as follows :

If C then A , i.e., if condition C is true then perform the action A. and this is symbolically represented as C/A. If there is more than one conditions $C_1, C_2, C_3 \dots$ and corresponding actions are $A_1, A_2, A_3 \dots$, then rules are symbolically represented as $C_1 \wedge C_2 \wedge C_3 / A_1 \wedge A_2 \wedge A_3$, it mean

IF C_1 AND C_2 AND C_3 THEN A_1 AND A_2 AND A_3 .

Credit is computed at the end of each navigation cycle and rules are updated to show their strength. These strengths are further used for weeding out the low performers while keeping the higher one.

One more advantage of the Zeigler model [10] is that it not only keep best-rule-so far but also some average rules. This always gives system an opportunity to learn, otherwise system will happy with it's existing rule sets

and will not take a risk to improve over the time. In the next chapter will see some more about the problem domain and the way it can be handled.

CHAPTER 3

PATH TRACKING AND OBSTACLE AVOIDANCE SYSTEM

3.1 Genetic Algorithm overview

Genetic algorithms are the search algorithms based on the mechanism of natural selection and natural genetics [12]. It works with an entire population of different objects, the individuals in the population are assigned fitness-values. This value is an indication of how good a solution a certain individual is. Fitness is calculated by an objective evaluation function.

3.1.1 Evaluation Function

The evaluation function is the only domain-specific part of genetic search. The genetic process itself is not guided by domain-specific knowledge. The individuals are encoded in natural chromosomes, which are usually just bit strings. All genetic operations are conducted on these chromosomes. Because genetic algorithms don't use domain-specific information to guide the search, they are almost universally applicable, especially in domains where very little is known about what is being searched for.

With the old population and the fitness of its members the next generation is created. This goes analogously to natural evolution: "the best individual has the biggest chance of producing offspring". Offspring are like

their parents, but usually not entirely. The newly made individuals eventually replace the older (and hence less well adapted) ones:

In this way we cover a large part of a search space in a short time. It can be proven that fit individuals will grow in number and less fit ones will disappear. The search will end up in an optimum in the search space, and because we work with a large number of individuals, this probably be a good optimum.

There are three basic actions in the genetic algorithm:

1. selection
2. creation of a new individual from parents and
3. Replacement of older individuals.

These actions operate on encoding of the sought parameters, not on the parameters themselves. Thus the genetic algorithm does not have to know anything about the parameters themselves.

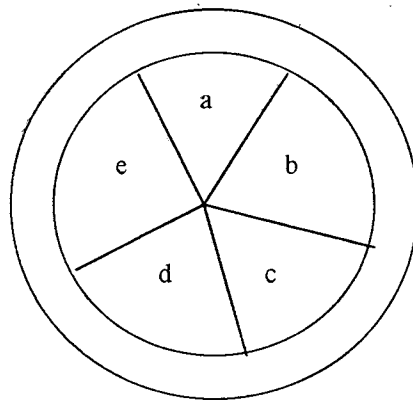
Every parameter can be encoded as a gene on a chromosome. In the technique that will be studied in this paper, genes are characters and chromosomes are strings, but they can be almost anything. The only thing the genetic algorithm must be able to do is to copy individual genes and chromosomes.

First we select the proper parent and then new individuals are created from the selected parents. The new individuals are created mainly by mean of crossover, mutation and inversion operator.

3.1.2. Basic Genetic Operators

Selection :Selection of individuals for pro-creation is the first step. The ones with the highest fitness will have the highest probability of being selected. Picking of individuals is usually done in one of two ways. Roulette-wheel selection or rank-based selection.

In roulette-wheel selection every individual is assigned a certain probability which is proportional to its fitness. All probabilities of all individuals should add up to one. Candidates for procreation are then selected according to these probabilities. This is a bit like spinning roulette -wheel.



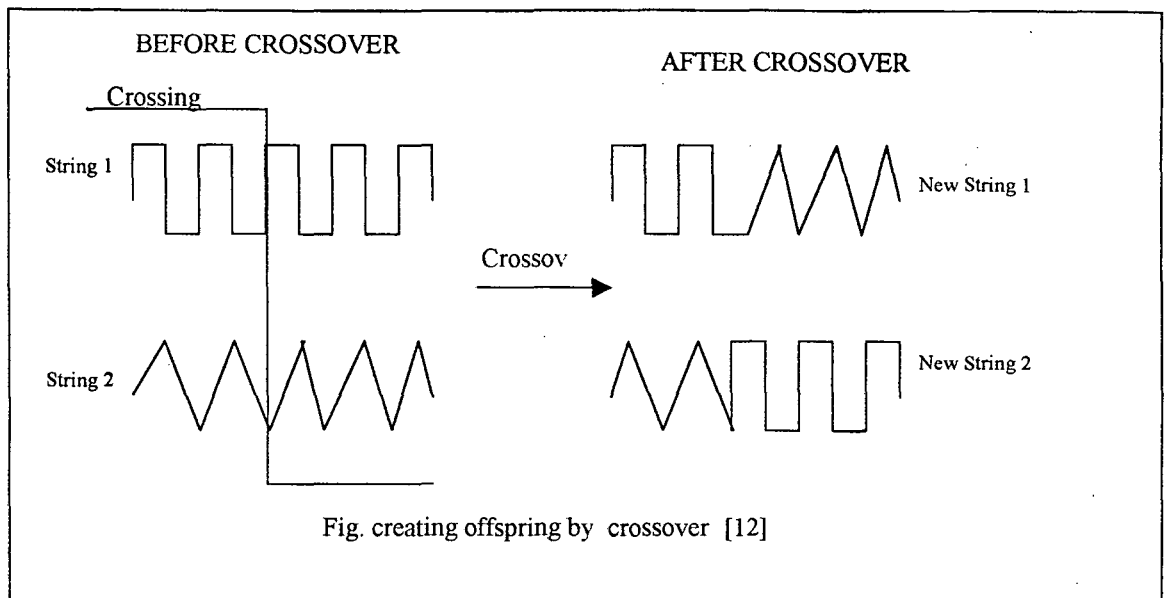
Roulette-Wheel with slot size according to the fitness.[12]

Rank-based selection individuals are ordered according to their fitness. Individuals are then assigned fixed probabilities according to their rank, that is their position in the list. Candidates are again selected with these probabilities.

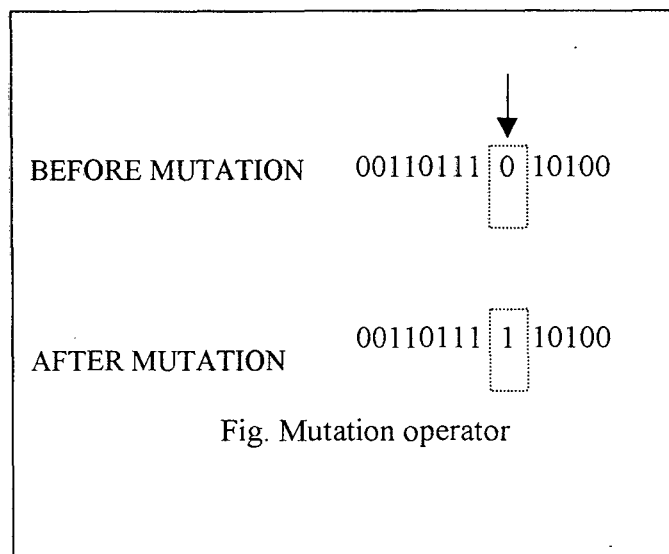
The difference between rank-based selection and roulette-wheel selection is not very big. When using roulette-wheel selection one should be careful that the ratio between the highest and lowest probability is not too great; otherwise certain individuals will very quickly start to dominate the population. It will usually be necessary to scale the probabilities. Goldberg [12] suggests that the ratio should be 1.2 and 2.

Rank-based selection has the problem that all the individuals have to be sorted first. It is much easier to control the different probabilities of the individuals in the population.

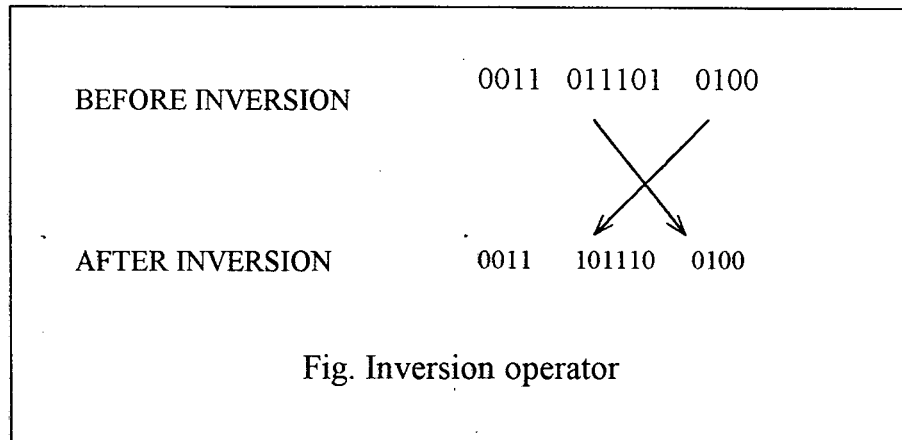
Crossover: Crossover is the mixing of two parents (usually there are two parents per child). The chromosomes of the parents have to be of the same length. A random point (between two genes) in one of the parents is chosen. The child will now consist of the genes before this point of the one parent and the genes after this point from the other parent. The purpose of crossover is to mix the genes of two parents, so good things (partial solutions) from one parent will be mixed with good things from the other parent. This could be considered an exchange of information.



Mutation: Mutation is a very simple operation. The value of a gene in the child is changed at random. This is done with a certain, rather small, probability. Its purpose is to introduce new solutions into the population, which were not present at the outset or were lost during the search.



Inversion: Inversion is a more complex thing. Two points in a chromosome are chosen at random. The sequence is a more complex thing. Two points in a chromosome are chosen at random. This is done to put partial solutions that reinforce each other closer together in the chromosome, so that they are less likely to be separated with.



These technique are not be used with every child. Crossover, mutation and inversion are only used with a certain probability.

3.2. Zeigler Adaptive System

Rule based adaptive expert systems create and modify the rules sets over the time [10][11]. The main disadvantage of adaptive rule based system is only best rules are being used always and system does not takes risk to improve over time.

3.2.1. Basic Zeigler model

Zeigler [10] provides an elegant solution for the above problem in which the system alternates two rules selection strategies from one generation to the next. During first generation, the best rules are selected while in the second generation the experimental sets of rules are selected. In this approach, the system time is broken into generations and system operates following steps in a loop.

1. A solution to all problems in problem set p is attempted using a subset of master rule set.
2. Each solution reached is evaluated and credit is assigned to the rule set and each rule in it.
3. The master rule set has the weeding criteria applied to it, if the criteria are not met low performer are removed.
4. Few new rules are generated from, and placed into the master rule sets.

Figure shown below is high-level description of above algorithm. Rule operator selects the rule by the rule generation process and loads it into the problem space. Using the selected rules, the rule operator attempts to solve the

problem within its own problem space. The final solutions are evaluated and results are saved in the rule manager.

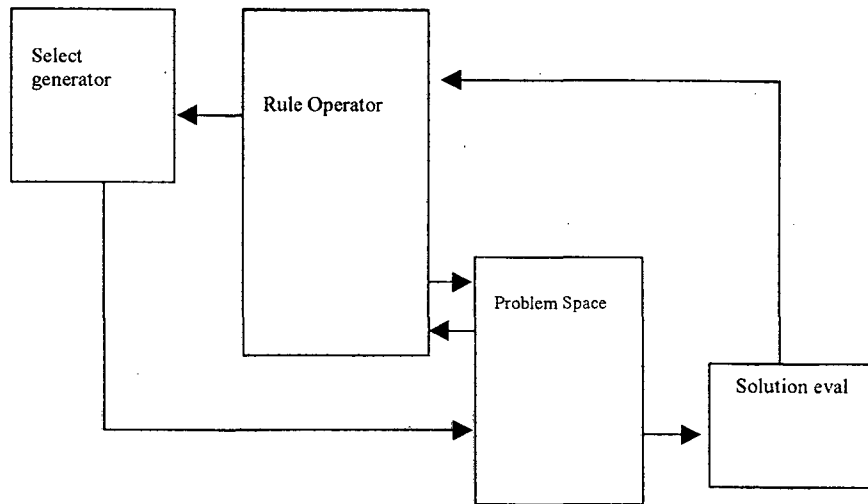
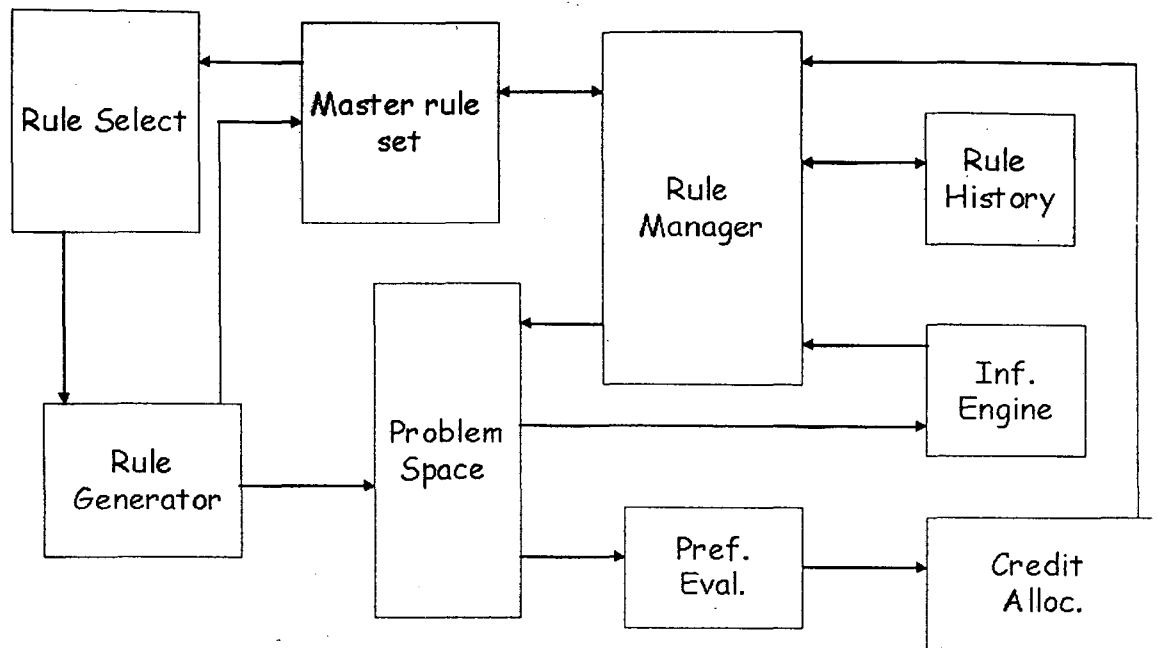


Figure 2: High level details of Ziegler adaptive system.

3.2.2 A detail structure of Zeigler Adaptive System

the following figure is an more detail representation of the zeigler adaptive system. When the system is started the master rule set the rule history and the problem space are initialized to contain the basic rule sets. These three are the only components in the system which are mainly data.

The inference engine is used to forward the chain on rules in the problem space until the termination criterion becomes true. This condition will usually test the current current state for equality to a goal state of the current system.



At any time, if the problem space reaches a state in which the no rule is applied, a subtask is invoked in which is simply a recursive call to the inference process using the original starting rule set. This allows the system to overcome such “frozen” state also allowing lost rules to be reintroduced into a rule set .

Upon termination, the solution is evaluated and by the performance evaluating component. The credit resulting from the is distributed among the rules in the rule set using the algorithm described below. the same rule set is then used to solve the next problem in the problem set.

When the all problems in the problem set have been presented the rule set itself is evaluated i.e. its average performance is calculated. This average value is compared with that of best-so-far rule set. If it is higher, it is appended to the rule history list and become the new best-so-far rule set. In any case the

TH-11281



rules in the master rule set are now updated to select the new credit values of the rules just used.

After update in the rule history and master rule set the weeding criterion is applied. Currently this is done every w generations. Finally a subset of rules are selected from the master rule set and new rules are generated for the next generation. These new rules are added to the master rule set and the problem space. The next generation then begins, using the selected subset and any new rules generated from them.

3.2.3. Payoff Allocation

The payoff enters the system upon reaching the termination condition and rules are only rewarded for being in a set of profitable rules leading directly to payoff. Note that credit enters the system every time a problem is attempted.

As system operates upon a set of problem p , given a rule set R_0 . It is expected that R_0 will be complete in the following sense. R_0 must contain enough condition and action to allow all transitions necessary for an acceptable solution to all problem in p . That is, if the system is expected to evolve a solution from a set of rules which is not a solution then all necessary condition and actions at least be present in the starting rule set, as the system can't create the new ones.

Whenever new rules are generated by the rule manager, they are copied into the master rule set. When weeding takes place, the weeding algorithm is applied to the master rule set. After all undesirable rules have been eliminated the rule manager is loaded with a copy of all the rules from the newly weeded master rule set. Hence the content of the master rule set during the generation g is given by:

$$R_g = \left\{ R_0 \cup \left(\bigcup_{i=1}^g R_{G_i} \right) \right\} - \bigcup_{i=1}^g R_{W_i}$$

Where

U is union of sets.

R_0 is the initial rule set (generation 0)

R_{G_i} is the set of rules generated at generation i .

R_{W_i} is the set of rules weeded at generation i .

The rules in the above formula are identified by the generation in which they are created, so if a rule is weeded and then an identical rule is later created, it is a different rule and will still be counted. The rule set used during generation g to attempt solving the problem set is given by:

$$r_g = \{ r_i : r_i \in \text{select}(R_g) \}$$

Where

$:$ Means "such that"

“Select “ is the selection function

Since weeding takes place before selection, hence R_g is taken as the set obtaining after weeding .

The credit entering the system while attempting problem p is domain dependent and will simply be called C_p for now. We may now compute the credit assigning to a rule during generation g . First the utilization of r_i is calculated:

$$U_{ip} = f_{ip}/tf_p$$

Where

f_{ip} is the number of times r_i , fires while rule set

tf_p is the total number of rule firings which occurred while rule set rg is solving problem p .

Now we can compute the credit assigned to rule $r_i(\in rg)$ for each problem attempted during generation g :

$$C_{ip} = \{ 3 * C_p - 0.1 * \text{complexity}(r_i) \} * u_{ip}$$

Where

C_p is the total credit apportioned to the rule set for its solution to problem P .

Since all the credit obtained by the rule set is divided among its rules according to their utilization, a low performing rule in a high performing rule set will not be able to get a “free ride “ by living off its colleagues. Furthermore a good rule in a mediocre rule set is not highly penalized, since if any credit is obtained it will go to those rules that contributed the most.

This scheme for credit assignment is intended to provide both a penalty and an incentive for complexity. The penalty is the second term in the formula. The incentive will be discussed below.

After computing the credit for one problem, we sum over all the problems to obtain the total credit assigned to rule i during generation g :

$$C_{ig} = \sum_{P=1}^{|p|} C_{ip}$$

In addition, the total number of firing of r_i is saved:

$$f_{ig} = \sum_{P=1}^{|p|} f_{ip}$$

The credit of each rule, r_i is also saved for all generation in which r_i exists in the system. Hence, the credit of rule r_i after generation g is:

$$C_i = \sum_{g \in S} C_{ig}$$

where S is the set of generations in which r_i is selected

A figure of merit for rules which is often used is its rating. This is computed whenever necessary from the above valued. The rating is the average credit per problem set attempted, and is computed as:

$$\text{Rating} = C_i / |S|$$

Information is also saved about rule sets. In particular, each rule set is evaluated; each rule set is evaluated after attempting a solution to a problem set. The value of the best performing rule set is saved as well as the rule set itself. The performance of a rule set is calculated as follows:

$$CR_i = \frac{1}{|P|} \sum_{P=1}^{|p|} C_p$$

This is just the average performance of the rule set over the problems in the problem set.

3.2.4. Combination Algorithm

There are two genetic operators used in the system. They are very simple and described here. However the design of the system is such that any new operator can be easily added.

The first is crossover, but is defined somewhat differently than usual for classifier systems. In this version of crossover, the crossover point, instead of being random, is always chosen as the point between the condition and action. Hence given two rules:

$$C_{a1} \wedge C_{a2} \wedge C_{a3} \wedge C_{a4} \dots\dots\dots C_{aN} / A_{a1} \wedge A_{a2} \wedge A_{a3} \dots\dots A_{aM}$$

$$C_{b1} \wedge C_{b2} \wedge C_{b3} \wedge C_{b4} \dots\dots\dots C_{bN} / A_{b1} \wedge A_{b2} \wedge A_{b3} \dots\dots A_{bM}$$

The two new rules obtained from crossover are given by:

$$C_{a1} \wedge C_{a2} \wedge C_{a3} \wedge C_{a4} \dots\dots\dots C_{aN} / A_{b1} \wedge A_{b2} \wedge A_{b3} \dots\dots A_{bM}$$

$$C_{b1} \wedge C_{b2} \wedge C_{b3} \wedge C_{b4} \dots\dots\dots C_{bN} / A_{a1} \wedge A_{a2} \wedge A_{a3} \dots\dots A_{aM}$$

Note that the multiple actions on the RHS of these rules are performed in order, so the list of actions represents an action sequence.

This second operator is called and-combination and works as follows:

Two rules to combine are selected. From the two conditions, a new condition is made conditions, a new completed by taking the two actions and forming a new action from the sequence of the two. Note that this always results in rules with a higher complexity.

Two parameters of the system are P_c and P , the probability of applying crossover and the probability of applying and –combination respectively. During the generation step of a time slice, each pair of rules in the applied to it Any generated rules are then added to the set which becomes the next generation rule set.

3.2.5. Weeding Algorithm

A parameter to the system is the weeding period, W which is an integer, It specifies the number of generations which the following steps take place:

1. All rules, which have never fired, are removed.
2. Low performing rules are removed.

Although rules which have not fired are not necessarily useless, they are weeded out as excess baggage. Most likely, such a rule has not fired because the state in which it fires has not been reachable using any of the rule sets selected so far. That does not imply that the state would not be reachable using future rule sets, but it was felt that such an event would be relatively rare, and in any case, the system would probably be able to regenerate any needed rule in that case. Also, rules with logically false conditions are eliminated in this way.

A rule is considered to be low performing if its rating is less than $\frac{100}{\text{generation}}$. This is to put a lower bound on each rule's rating. The lower bound is designed to slowly increase with generation, which will gradually force out rules whose rating does not increase.

Selection Algorithms Currently, the system operates in two modes, best and experimental which alternate each generation. The difference between the two modes is the algorithm which is used for selection of that generation's rule set. To select a best rule set, the following algorithm is used.

1. Select the $M+P$ best rules, where M is the number of rules in the best-so-far rule set, and P is a parameter (order of M).
2. Randomly reduce the above list to length M .

The resulting list is the selected best rule set, which is used for the next generation. In experimental rule set is obtained using the following algorithm:

1. Calculate the cutoff point of rule performance ,using the formula

$$C = (7/8) * \text{best} + (1/8) * \text{worst}$$

Where best and worst are respectively the highest and lowest ratings in the rule set

2. Select all rules with rating grater than C.
3. Select P, rules randomly (Pc is a parameter)
4. Take the union of the above two lists, and randomly reduce it to length M (length of the best-so-far rule set).

3.3. Modified Zeigler model

In order to use Zeigler adaptive model efficiently certain modifications have been done. There are basically two modifications that have been tried upon. As Zeigler model assume that there are p sub-problems in the problem space while in coevolutionary model assume that each subtask is logically a single task. Thus in order to evaluate the average performance of the particular generation we take more than one training sets and evaluate the average

performance. In next modification we assume that only one at each step only one ACTION is performed i.e. we will never fire the rule like:

IF C1 and C2 and C3 THEN A1 and A2

Rather we will use a sequence of two different rules in two steps of time i.e.

RULE 1: IF C1 THEN A1

RULE 2: IF C2 THEN A2

Also crossover is simplified and for above rules it is defined as (same as original Zeigler Model).

RULE 3: IF C1 THEN A2

RULE 4: IF C2 THEN A1

And mutation has been defined as:

For any rule: IF X THEN Y, replace the action Y by some other action Z.

Hence new rule after mutation will: IF X THEN Z.

3.4. Path tracking and obstacle avoidance system

The work done in this dissertation implements the coevolutionary learning using modified Zeigler adaptive expert system. Problem domain considered here consists of navigation of robot. During navigation, the robot must be able to follow an object while avoiding the obstacle in the navigating path. The knowledge is represented in the form of sequential decision rules.

We apply modified Zeigler model in each subcomponent as well as in the merging module of sub-components to evaluate the performance of the collaborative solution.

The navigation problem can be considered as the decision-making problem in each time step. Robot consists of a limited set of sensors and effectors. The sensors provide the certain information about the environment to the robot. With the help of these sensors Robot also knows it's state i.e. speed, direction and position etc. at each step of time robot matches it's sensor values for some rules condition to be true and action of the matched condition is performed. If more than one condition matches then rule with higher fitness value is considered.

3.4.1. Credit assignment

If any subcomponent performs the task successfully then total credit assigned to that rule set for that particular problem space will be 100%. Otherwise it will be normalized over 50%.

For example if we want a robot to navigate for 100 steps and if (whatever condition there may be i.e. collision avoidance or path tracking).

Successful then credit =1

Otherwise credit = $((0.5)/99) * (\text{total steps traveled successfully})$

3.4.2. Conflict resolution

During the rule matching and firing the conflict resolution is achieved through the fitness function. All rules are stored in sorted order hence first matching rule should be fired. Also we assume that there are sufficient rules are available in the rule set so that at least one matching rule must be there.

3.4.3. Subcomponents to be evolved

My whole idea is based on the coevolving components i.e. multiple instances of Genetic algorithms, running in parallel using the modified Zeigler model. All rules are evolved in the form of high level rule and in the form of IF *condition* THEN *action*.

The simulated robot considered in this dissertation work, it is assumed that sensors can detect obstacle and object within certain permissible range. The sensors can view by certain degree and can detect the distance of objects and obstacle in discrete values. Sensors used for detecting the obstacle might be different from the sensors needed for detecting an object.

Now two subcomponents, which we are evolving, are:

1. Object's Path tracking components i.e. chasing the object and if object goes out of range to the object detecting sensor before chasing the required number of steps the system will fail.

2. Obstacle avoiding system i.e. the robot must be able to walk in the navigating path for the required no of steps without collision.

The next section dedicated for way evolving the subcomponents and merging the subcomponents.

3.4.4. Evolving Subcomponents

As it is discussed earlier that in coevolutionary learning the subcomponents evolve by initial seeding and no extensive hand held design is needed. Once population is seeded for the particular subfunction the population evolves without any human support.

For evolving the first subcomponent i.e. object's path tracking system, N-object detecting sensors are used. These sensors provide the information about the distance and direction of the object with respect to robot.

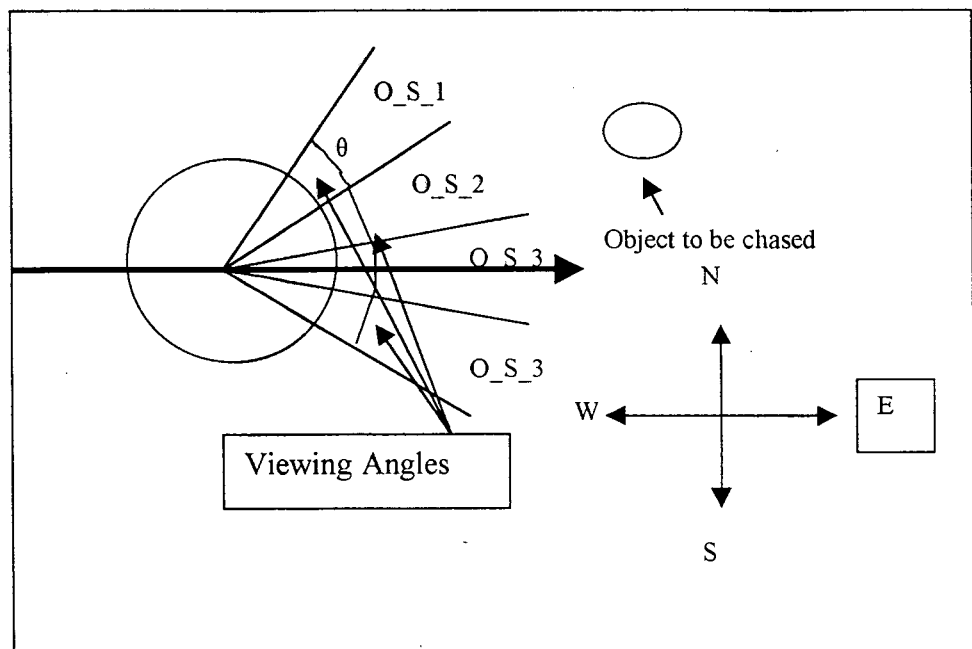


Figure shown above has three object detecting sensors o_s_1 , o_s_2 and o_s_3 . Each can view up to angle θ . Also it can detect the distance of the object let d . all these sensors have a non-overlapping viewing angle hence at any time only one out of N (here 3) object sensors will be active. Depending upon the active sensor and other state variables like speed etc there can be M actions be fire. For example if $d=100$, active o_s_2 ad speed =20 unit the corresponding action can be change direction by 10 degree in clockwise direction and change the speed by 5 unit +ve i.e. increase the speed by 5 unit.

Similarly for obstacle avoiding component there are K obstacle detecting sensors. Each sensor has a fixed viewing angle and sensors can detect obstacle up to a certain permissible distance. Unlike the object sensor there can be more than one active obstacle sensor i.e. more than one sensor can report for obstacle within its range.

For assigning the credit it is assumed that robot need to navigate up to a desired no of steps. Value $STEP_MAX$ controls this factor.

Now following steps are needed in order to evolve the rule set of the first sub component.

- 1 Create the p simulating path. Here p corresponds to the no of sub-problems in the original Zeigler model.

- 2 Create initial population of rules which will serve as the master rule set and set credit=0;
- 3 Apply the modified Zeigler model for credit assignment evolving the new rules up to G generation or until proper performance is achieved.

Similarly the second component will evolve. Only difference will be the set of sensors for obstacle detection may differ from set of sensors used for object seeing.

3.4.5. Merging the subcomponents

This section deals with implementation details of merging.

Merging is one of the vital components of the coevolutionary learning. The genetic material evolved from the individual subpopulation must be merged together by some mean in order to achieve the desired task.

As described in the chapter two that representatives from the subcomponents are selected and are merged together to achieve the higher level task. The feedback about the collaboration is given to the original subcomponent of the representatives.

The merge module has the state information of all components it is intended to merge. Thus while merging the rules from different component the following conditions are possible.

1. If for the given state both rules fire the non contradicting action i.e. for both rule action part is the same then combine these two rules in to a single rule. Credit to the merged rule will be highest .
2. If for the given state the rules fire has the partial contradicting action(change in direction =10 clockwise in both rule but for speed one rule suggests increase in speed while other suggests the decrease in speed) then keep the partial matching action while among contradicting action choose the action from higher fitness population or in other word we choose the action part from the higher fitness holding individual.If there are N matched and M unmatched partial actions then the total credit assigned to the merged rule = $(N/(M+N)) * \text{fitness of best representative}$.

After merging the rules the feedback is floated back in order to improve the subpopulation again.

The merged rule here now can generate a representative to be merged further. Next chapter deals with coding and implementation of these concepts and there performance evaluation.

CHAPTER-4

IMPLEMENTATION AND RESULT

4.1. Description of the Simulation Domain

The simulation environment is a two-dimensional navigation path. The path consists of obstacles and a moving object. The robot must be capable of tracking the object without colliding with the obstacle. Robot has the limited set of sensors including sonar and various types of effectors that maps to sensor reading to actions to be performed. The mapping between the sensor reading to the action to be performed is determined at each time-step. System learns a specific set of rules that reactively decides a move at each time-step and not the specific path.

Whole simulation has been done in C++ using “Standard Template Library”. Rules in the different sub-components have been implemented as objects of different class. Feature like operator overloading from C++ has been used for comparing the two incompatible rules set class without loosing the generality of equality operator and STL has been used as container class of the rules and rule set.

4.1.1. Navigation Path

As told the robot navigates in a 2-D simulating environment. The environment consists of stationary obstacle and a moving object. The

navigation path has been implemented by a mean of 2-D array. The value 0 at any location represents the no obstacle while value 1 represent an obstacle. The obstacle probability is not necessarily be 0.5 rather it is also configurable.

```
#define OBSTALE_PROBABILTIY
```

The navigation area is also configurable i.e.

```
#define MAXX //maximum permissible area in the x direction
```

```
#define MAXY // maximum permissible area in y direction
```

4.1.2. Sensors

The robot is equipped with an active sonar for detecting obstacles in its path and the object to be tracked. The sonar is composed of NO_OF_SONAR cells, each with a resolution of MAX_DEGREE degree, giving the robot a total coverage of NO_OF_SONAR *MAX_DEGREE degrees. The robot also has some internal, or virtual, sensors that give the robot certain information about its own state. The sensors are:

1. Last-turn: the current turning rate of the robot. This sensor can assume n values, ranging from R1 degrees to R2 degrees in MAX_ANGLE_CHANGE degree increments. Where $n = (R2 - R1) / \text{MAX_ANGLE_CHANGE} + 1$.
2. Time: the current elapsed time, an integer.

3. Steps: the no of steps it has moved. Range from zero to MAX_STEP.

4. Speed: the current speed of the robot. It ranges between 0-MAX_SPEED in step of CHANGE_IN_SPEED. Thus can take $\text{MAX_SPEED}/\text{CHANGE_IN_SPEED}+1$ values from 0 to MAX_SPEED. The robot can come to a stop.

5. asonar_n: One of n active sonar cells used for collision avoidance and object tracking. Each cell takes on values from 0 to RANGE_OF_SENSOR in increments of STEP_OF_RANGE unit and represents the distance to an object within that sonar cell's view. If no object is seen, a special value, NO_OBSTACLE, indicates infinity. Each sensor can view the VIEWING_ANGLE_OF_SENSOR degree. Thus total viewing angle achieved is $\text{NO_OF_SENSORS} * \text{VIEWING_ANGLE_OF_SENSOR}$. Also each sensor can have noise added to simulate a more realistic environment. In particular, the sonar readings have both a Gaussian noise added, and a small random probability of "missing" an object, or of reading a "ghost" object that is not really there.

6. object_tracking_sensor_n: n object tracking sensors. Also represented as o_s_n i.e. nth object detecting sensor. For simplicity it has the same characteristics, as that of asonar_n. NO_OF_OBJECT_SENSORS is the total no of object tracking sensors.

4.1.3. Actions (effectors)

There is a discrete set of actions available to control the robot. In this study, we consider actions that specify discrete turning rates and discrete speeds for the robot. The control variable turn has n possible settings, between $R1$ and $R2$ degrees in degree MAX_ANGLE_CHANGE increments. Note that the sonar may cover lesser viewing angle than the turning capability i.e. the $R2-R1$ can be larger than the $NO_OF_SENSORS * VIEWING_ANGLE_OF_SENSOR$, so it is possible for the AUV to turn into an object that is not in sonar range. The control variable speed has n possible settings between 0 and MAX_SPEED (the units are arbitrary) with an increment of $CHANGE_IN_SPEED$, where $n=MAX_SPEED/CHANGE_IN_SPEED + 1$. The learning objective is to develop a reactive plan, i.e., set of decision rules that map current sensor readings into actions, that successfully allows the robot to track the target up to the specified steps i.e. MAX_STEP steps while avoiding the obstacle.

In this simulation, we assume that the robot can change speed a maximum of $CHANGE_IN_SPEED$ units and direction a maximum of MAX_ANGLE_CHANGE degrees within one decision time step.

The robot simulation is divided into episodes that begin with the placement of the robot centered in front of a randomly generated training path with a specified density. The episodes end with either a successful tracking for specified no of steps or, or there is a collision with the obstacle.

4.2. Performance Component

The performance module is a production system interpreter. The primary features of production system are:

1. A restricted but high level rule language;
2. Competition-driven conflict resolution; and
3. Incremental credit assignment methods.

4.2.1 Knowledge Representation

Rules are expressed in a high-level rule language. The use of a high level language for rules offers several advantages over low level binary pattern languages typically adopted in genetic learning systems i.e.

Each rule has the form

if c then a .

where c is a condition on the sensors and action a specifies a setting for the control variables.

The right-hand side of each rule specifies a setting for one or more control variables. For robot, each rule specifies a setting for the variable turn, and a setting for the variable speed. In general, a given rule may specify conditions for any subset of the sensors and actions for any subset of the control variables. Each rule also has a numeric strength, that serves as a

prediction of the rule's utility. The methods used to update the rule strengths is described in the section on credit assignment below. A sample rule in the

```
if ( (speed is [30])
      (asonar_1 is [100])
      (asonar_2 is [150])
      (asonar_3 is [220])
      .....
      (asonar_NO_OF_SENSORS is [100]))
then ((change_in_angle is [10]) (change_in_speed is [-10]))
strength 0.75
```

Path tracking component may produce the rules of the form

```
If((speed is [40])
    (o_s_1 is [30])
    (o_s_2 is [50])
    (o_s_3 is [220])
    .....
    .....
    (o_s_NO_OF_OBJECT_SENSORS is [110]))
then ((change_in_speed is [5])
```

(change_in_direction is [10]))

strength 0.50

4.2.2 Production System Cycle

Production system follows the match/conflict-resolution/act cycle. There is no guarantee that the current set of rules is in any sense complete, it is important to provide a mechanism for gracefully handling cases in which no rule matches neither in problem-space nor in master rule set. This is accomplished by generating new random rule for the particular condition with strength =0. All rules in the match set that agree with the selected action are said to be *active*, and will have their strength adjusted according to the credit assignment algorithm described in chapter 3.

Thus the final rule set will evolve

4.2.3. Merging the Subcomponents

The merge module takes the representatives from both subcomponents i.e. takes the best rule set of obstacle avoidance and best rule set of object tracking. After merging the rule-set of both components the performance is evaluated in combined form. The feedback is again passed to corresponding sub-unit. It is quite possible that even best individual performer scan give

average or poor performance when they are merge together. Thus feedback mechanism cause to evolves some new species if required.

After merging the rules may appear like below.

```
if ( (speed is [ 40])
      (asonar_1 is [100])
      (asonar_2 is [150])
      (asonar_3 is 220)
      .....
      (asonar_NO_OF_SENSORS is [100])
      .....
      (o_s_1 is [10])
      (o_s_2 is [20])
      (o_s_3 is [5])
      .....
      .....
      (o_sr_NO_OF_OBJECT_SENSORS is [220]))
then ((change_in_speed is [10])
      (change_in_direction is [5]))
strength 0..37
```

4.3 Learning Module

Learning is achieved at two different levels namely

1. The credit assignment at the rule and rule set level.
2. Genetic competition at the plan level.

4.3.1. Credit assignment

Total payoff to the rule set =

1.0 if robot is able to follow the object for
MAX_STEP steps.

$(0.5/(MAX_STEP-1))*n$ if there is a collision.

Where n is total no of steps traveled before collision to collision.

Payoff at rule level is achieved according to the algorithm described in the chapter 3rd and weeding are perform based on the strength of the rule.

4.3.2. The Genetic Algorithm

The learning process is a heuristic optimization problem, i.e., a search through a space of knowledge structures looking for structures that lead to high performance. A genetic algorithm is used to perform the search from the

current knowledge base on the basis of fitness, and applying idealized genetic search.

While implementing a random initialization of the first population has been made. Since this approach requires the least knowledge acquisition effort, provides a lot of diversity for the genetic algorithm to work with, and presents the maximum challenge to the learning algorithm. Alternatively we can do adaptive initialization in which rule is specialized according to its early experiences.

4.3.3. Genetic Operators: CROSSOVER and MUTATION

Crossover is used to create new rules from existing rules. Let we consider two rules:

Rule 1: if c1 then a1

Rule 2: if c2 then a2

Then new rules we get through crossover are

Rule 3: if c1 then a2

Rule 4: if c2 then a1

Crossover is simply selecting rules from each parent to create an offspring plan.

Mutation operator introduces the new rules by the random changes. Mutation occurs with very small probability in compare to other genetic operators.

Let there be a rule

Rule 5: if c then a

The mutation operator can define the new rules by changing the action a with some other action namely A . Thus new rule will be

Rule 6: if c then A .

4.4. Experimental Results

Simulation has been performed in the following environment.

```
#define MAX_POPULATION 1024 // initial population in the master rule set
```

```
#define NO_OF_SENSORS 7 // total no of sonars
```

```
#define RANGE_OF_SENSOR 200 // viewing distance of the sensor
```

```
#define STEP_OF_RANGE 10
```

```
#define VIEWING_ANGLE_OF_SENSOR 10 // viewing angle in the degree
```

```
#define INITIAL_ANGLE 0 // all measurement will be taken from this only
```

```
#define W 1 // weeding period
```

```
#define MAX_SPEED 40 //maximum speed
```

```
#define CHANGE_IN_SPEED 5
```

```
#define MAX_ANGLE_CHANGE 10 //maximum permissible change in angle
```

```

#define MAX_STEP 100 // maximum steps of wandering

#define MUTATION .25f // mutation rate

#define NO_OBSTACLE 220 // indicates that there is no obstacle within the
range

#define NO_OF_GENERATION 800

#define OBSTACLE_PROBABILITY .5

#define MAXX 400 // navigation range in X- direction

#define MAXY 400 // navigation range in Y- direction

```

4.4.1. Rules Evolved

Some rules evolve in the generation 789.

```

[speed|sonar_1|sonar_2|sonar_3|sonar_4|sonar_5|sonar_6|sonar_7|o_s_1|o_s_2|o_s_3]/[Δspeed|Δ-
direction]

```

10	220	100	220	100	120	220	220	220	100	220	+5	0
10*	220	100	140	150	220	180	220	220	220	100	+5	-10
10	130	220	220	180	220	70	220	220	220	20	0	-10
10	220	220	50	80	220	220	220	120	220	220	0	0
20#	220	220	220	220	220	220	220	220	40	220	+5	0
20	220	110	150	220	220	220	180	100	220	220	+5	0
20	220	220	30	220	100	130	220	80	220	220	-5	0
20	40	220	220	220	120	80	60	220	30	220	+5	0
25	10	220	20	20	220	220	220	220	40	220	-5	-10
25	120	220	140	80	20	220	220	100	220	220	+5	+10
25	220	130	220	30	220	120	220	220	70	220	0	0
25	220	100	220	220	110	220	220	220	90	220	0	0

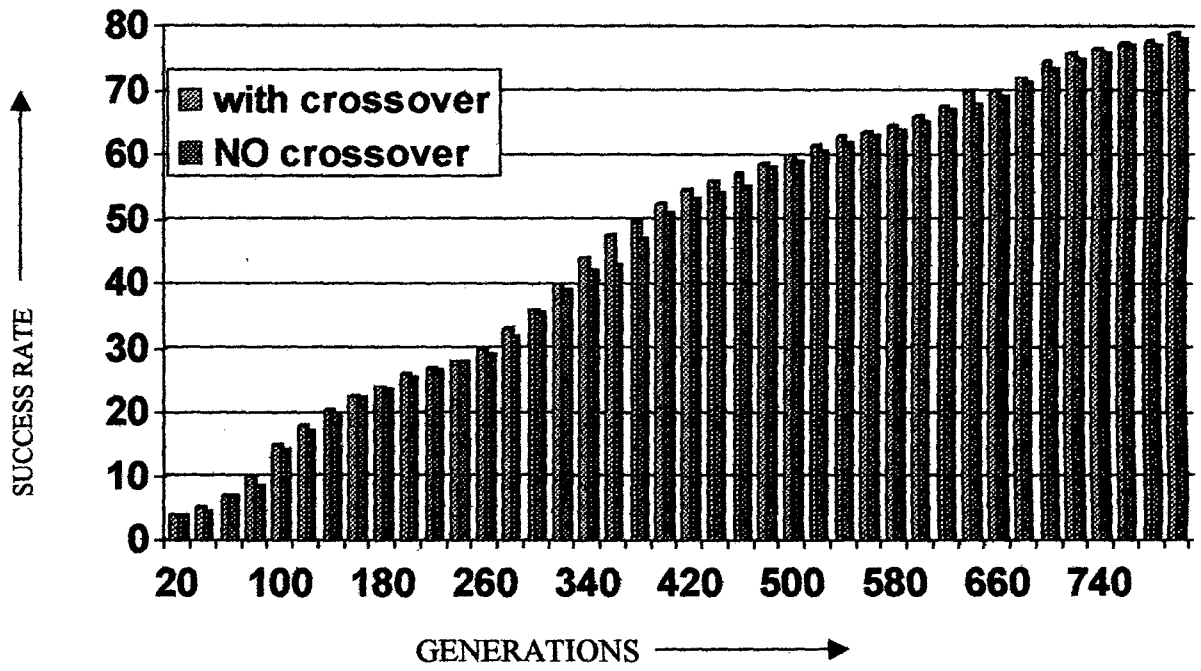
Explanation of Rules:

The rules shown above have two parts. Pipe “|” is representing the logical AND. String before symbol “/” is condition part while part after “/” is action part. Δ -speed and Δ -direction are the change in speed and change in direction respectively. Here +ve change in speed means speed will increase by that amount after firing the rule and -ve means speed should be decreased by that amount after firing rule. change in direction will be clockwise if the value Δ -direction -ve otherwise change in direction will be counter clockwise. Thus first bold faced rule is representing the following rule:

IF(speed=10 AND sonar_1=no obstacle,sonar_2=100 i.e. sonar 2 is detecting obstacle at a distance 100 AND sonar_3=140 AND sonar_4=150 sonar_5=sonar_7= no obstacle AND sonar_6=180 AND object_tracking_sensor_1=object_tracking_sensor_2=No object in vision AND object_tracking_sonar_3=100 i.e. object to be traced is at a distance 100 from the 3rd object tracking sensor .)

THEN (change in speed +5, here +ve means increase AND change in direction 10 clockwise , since value is -ve).

4.4.2 Performance Measurement



CHAPTER - 5

CONCLUSION AND FUTURE WORK

This dissertation work deals with a cooperative coevolutionary approach to robotics. Coevolutionary approach together with modified Zeigler adaptive expert system has been used to evolve the robot navigation. The knowledge has been represented in the high level language, and can be used in reactive systems. This is also very useful in the sense that reactive rules are very difficult to design by hand.

Also this approach forms the stable niches. Each niche represents a particular sub-behavior and never migrates to the other species. This prevents any haphazardness in the overall behavior.

In the present work a simple merging algorithm of the sub tasks to form collaboration has been used. Development of better merging modules to achieve high level collaboration would be an interesting future direction.

Also the credit assignment technique is linear and can be explored further for better credit assignment function. The problem domain in which non-coevolutionary approach doesn't work well would be a logical extension to show the potential of co-operative evolutionary approach. Other extension might be navigating the robot in 3-D space and tracking the object among more than one similar confusing objects.

REFERENCES

1. Kumapati narendra, M.A.L. Thathachar *Learning automata, an introduction*, Prentice Hall International, Englewood Cliff NJ 1989.
2. Ryszard S. Michalski, Jaim. G. Carbonell, Tom M. Mitchell, An Overview of Machine Learning, pp 3-23
3. Herbert A. Simon, Why should a Machine Learn?, pp 25-37.
4. Mitchell A. Potter, Kenneth A. De Jong, and John J. Grefenstette (1995). A Coevolutionary Approach to Learning Sequential Decision Rules. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 366-372. Morgan Kaufmann.
5. de Garis, H. (1990). Building artificial nervous systems using genetically programmed neural network modules. In B. Porter and R. Mooney (Eds.), *Proceedings of the Seventh International Conference on Machine Learning*, pp. 132-139.
6. Singh, S. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning* 8, 323-339.
7. Lin, L.-J. (1993). Hierarchical learning of robot skills by reinforcement. In *Proceedings of the 1993 International Joint Conference on Neural Networks*, pp. 181-186. IEEE Press.
8. Grosso, P. (1985). Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model. Ph. D. thesis, University of Michigan, Ann Arbor, MI.
9. Cohoon, J., S. Hegde, W. Martin, and D. Richards (1987). Punctuated equilibria: A parallel genetic algorithm. In J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 148-154. Lawrence Erlbaum Associates.
10. B.P. Zeigler and Chris P. Stackhouse. An adaptive Expert System. Proc. Int. Symp. Modelling and Simulation Methodology, University of Arizona 1987.
11. B. SOUČEK and M. SOUČEK. Neural and Massively Parallel Computers: The Sixth Generation. A Wiley-Interscience publication (1998). Pp 276-288.
12. David E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*, Pearson education pte. Ltd., 2003, pp 1-20.

