Intelligent Routine Management
For
Open Credit System
Thesis Report

Supervisor: Moin Mostakim

Co-supervisor: Dipan Lal Shaw

Conducted by:

Sifat E Jahan 11201037

Ruksana Akter Brishte 12101121

Sadia Mehrose Islam 13101294

School of Engineering and Computer Science BRAC University

Submitted on 24th August, 2015

Declaration

We, hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole or in part, has been previously submitted for any degree.

Signature of Supervisor

_____

Moin Mostakim

Signature of Author

_____

Ruksana Akter Brishte

Signature of Author

_____

Sadia Mehrose Islam

Signature of Author

_____

Sifat E Jahan

Acknowledgement

This is the work of Ruksana Akter Brishte, Sadia Mehrose Islam and Sifat E Jahan, students of the CSE department of BRAC University, studying CSE starting from the year 2011. The document has been prepared as an effort to compile the knowledge obtained by us during these four years of education and produce a final thesis which innovatively addresses one of the issues of the current practical world. Although there may be more serious and urgent issues that need resolution, still we felt the need to provide a more intelligent routine managing system. We intended to develop a routine, much like BRAC University semester's routine, which would propose effective and efficient routine. The problem was to generate a routine by reducing manual work. Initially this seemed like something achievable and indeed it was. As we began developing the system, our supervisor started adding to it more and more features and functionalities to such a point that the system is capable now to provide a perfect, effective and efficient routine. We had plenty of help from Mr. Moin Mostakim, CSE Dept., BRAC University, with the paperwork. The generation of this report would not be possible without his help.

Table of Contents

List of Figures

Thesis Outline

The thesis consists of eight chapters in all and is outlined below. Each chapter consists of at least one or more sections that describe a specific part of that individual chapter. A detailed description of each of these sections is also outline below.

Chapter One details the purpose, aim and motivation for the development of this thesis. It has three sections - introduction, motivation and objective.
The *Introduction* section describes the purpose and aim for the development.
The *Motivation* section describes the motivation behind the whole development.
The *Objective* section describes our objectives.

Chapter Two describes the problem that this thesis targets to eliminate, the current solutions that exist in the market. It has three sections - problem description, previous works and possible approaches.
The *Problem Description* section describes the problem that this thesis is attempting to solve.
The *Previous Works* section lists the most recent and promising work done by others to provide a similar solution.
The *Possible Approaches* section describes in detail the possible ways to eliminate the problem described in problem description.

Chapter Three describes the literature reviewed to choose the best solution in order to implement the system. This part contains seven sections. They are - *big data*, *data mining*, *routine management*, *job scheduling with constrains*, *back propagation*, *neural networks*, and *decision tree*.

Chapter Four describes briefly about the proposed solution and how we choose it for the system. This chapter has three sections. They are – comparison between different techniques, proposed solution by comparing with an existing system, and proposed solution.
The *Comparison between different techniques* section describes how we compare different ways to accomplish our system.

The *Proposed solution by comparing with an existing system* section describes a comparison between an existing system and our approach.
The *Proposed solution* sections tells us about the way we selected by doing all the comparisons for our system.

Chapter Five describes briefly about the proposed solution and how we customized it for the system. This chapter has three sections. They are – *individual routine, merged routine*, and *room allocation*.
The *Individual routine* section describes how we generated an individual routing by using the customized proposed solution in the system.
The *Merged routine* section informs us about the usage of individual's routine in order to generate a single routine.
The *Room allocation* section allocates room number to final routine.

Chapter Six describes the ways of collecting data and using these and chapter five, how we implemented it. This contains two sections – *data collection* and *methodology*.
The *Data collection* section informs us about the types of data we collected.
The *Methodology* section describes how we implemented the solution along with the data in the system.

Chapter Seven describes the output of the proposed system. There are two sections – *results* and *graph*.
The *Results* section describes in the output of the system.
The *Graph* section describes in detail about the results of the system.

Chapter Eight describes the concluding notes of the authors for this thesis and contains two sections – *limitations* and *future work*.
The *Limitations* section describes in detail all the limitations of the system.
The *Future works* section describes in detail about the features that are to be added in future with the system.

Abstract

Intelligent Routine Management for Open Credit System

In this era of technology, for completing a task one relies heavily on computer. It has become an important part of one's life which helps to reduce manual work. Now a days, time plays a vital role in an individual's life. Hence, we need to keep track of time and distribute it at a work or task accordingly so that it is effective and efficient. In order to achieve this, we need to manage routine efficiently, and intelligently which is a challenge. Taking this importance and challenge into account, we proposed the *Intelligent Routine Management System*, in short *IRM*, for university to repair a semester's routine. This helps us to manage routine in a way such that one can make most out of the time, and to achieve this we used *genetic algorithm (G.A.)* with *constraints* which makes it effective and efficient. It is a user friendly system and proposes a routine based on the main data, the faculty preferences and few constraints. It processes the data according to our designed genetic algorithm and checks the efficiency based on the given constraints. Thus, it proposes a routine which is used as the class schedule for a semester of a university.

Chapter 1

With the goal of building a new way to routine generation system which can be generated automatically, we've come up with our project called, IRM (Intelligent Routine Management).

1.1 Introduction

Scheduling and managing a routine is a difficult task however it is an essential part of human life. Routines help human beings develop self-discipline, make their lives more productive and relieve from stress. Therefore an efficient management of routine is necessary. The investment of time spend in creation of routines is worthless and time consuming. Staring from personal life to corporate life, in education institution or in offices scheduling and managing routine is needed in different ways. For example, making the routine of a University takes a lot of time and doing it manually is really hard as that individual creating it should ensure that the arrangement of the routine should conform every one preferences related to it. In our research we have built an intelligent routine management system using genetic algorithm by which a routine will be automatically generated using many data.

1.2 Motivation

From the very beginning of our university life we have seen one faculty preparing the class schedule for every semester manually. We felt that it was very laborious and difficult to prepare the schedule for a department by single person as it is required to consider a lot of constraints. Such as allocate courses in a way so that faculties don't get different classes in same time, allocates room numbers, and consider theory and lab timing of each course so that they does not clash with each other at the same time. In addition to that, it also checks each faculty's availability to generate the routine. Therefore, making a schedule manually for a semester is a

very tough and challenging task. By conducting some researches and studies, we found that it is possible to build an intelligence routine management system by which the schedule will be prepared automatically using given data. Hence, we decided to build the system in order to prepare the schedule automatically.

1.3 Objective

Each day as we turn the calendar, our lifestyle is getting busier than ever and simultaneously our dependency on modern technology is getting strengthened. To make our life easy using the modern technology, we need to address many mathematical problems and algorithms. Due to the overall influence of the routine in almost every aspect of our lives and the complexity of its arrangement focus our attention for this research. The objective of this research is creating an automated system to build university's class routines in an easy and efficient way to decrease the manual work, by which we want to take our department one step forward of using the modern technology, for saving time and to present an optimized schedule in which no type of clashes will be present.

After some researches and studies, we found that it is possible to build an intelligent routine management system using genetic algorithm. Genetic algorithm is a search heuristic that mimics the process of natural selection. Genetic algorithm belongs to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. This algorithm is more robust than other existing techniques. It does not break its consistency easily even if the inputs changed slightly. Moreover, searching in large space GA offer significant benefits over typical search optimization techniques. Also, it provides huge output variation which cannot be achieved with other existing techniques which is the main reason for us to choose this technique.

Researchers have been working with Genetic algorithm to solve scheduling problem for many years. Many models have been introduced to solve this problem. In this paper we tried to introduce a new approach to build intelligent routine management system using genetic algorithm.

Chapter 2

In this chapter of our report, we are going to discuss briefly about the problem set, previous works related to our topic and some possible approaches to solve the problem.

2.1 Problem Description

It is seen that still individual's tend to make their daily routine manually which takes a lot of time. And also sometimes contains error since it is said that, "To err is human", said by Alexander Pope. If an individual's routine contains error then it will only affect him and does not affect others by that much compared to him. Let's move on to the big picture, where we consider not only a single individual but a big group of individuals. In an organization or company, an individual have lots of responsibilities to fulfill. So if on addition to that, responsibility of making a perfect routine for the organization or company, is given to him/her, it increases pressure. In order to do all task perfectly at a time is a challenge, and this might cause unwanted delays in delivering the tasks. Due to this delay, the other individuals depending on the routine might have problems to face such as fixing important meetings, scheduling tours, and etc. Hence, in the whole process the main sufferers are the individuals and other depending on him. And the routine have a chance of not being efficient, due to human error. Another most important point, the routine should match to others comfortable, free day and time, which is not possible for an individual to do manually. Since taking everyone's preferences and making a perfect routine going in hand to hand with everyone's preferences is tough and if possible then would take a lot of time to complete it. By that time, things will become a mess. From a university's perspective, a delay in producing a perfect routine affects thousands of students studying there. They always eagerly waits for the routine in order to choose subject without clashes and make a routine out of it for their semester. And in terms of a faculties or lecturers, they have a fixed time and day where they are comfortable to take courses especially the part time lecturers. It is important to set the university routine according to each of the lecturers' preferences and be delivered on time and to accomplish this, one should stop doing it manually rather than using technology's help.

For our research and implementation, we then choose a university, BRAC University in order to implement such a routine management system which overcomes these above mentioned and discussed problems.

2.2 Previous Works

Now, there are many ways to manage routine using technology. By technology I mean, using computer we can build a system to generate the routine. So we searched if someone has done a work on this type of field which matches with our idea.

- Firstly, we found out about a project done by a student of North South University. They have proposed to generate an academic routine using Decision tree based Routine Generation (DRG) algorithm. Based on this concept, Exam-time Tabling algorithm (ETA) is developed to implement conflict free exam-time schedule. They have considered many key factors like Teacher Priority, Highest conflicted course, Tolerable conflict and Neighbour slots by ignoring teacher's wish list [1].
- Secondly, a system was build using job scheduling algorithm with constraints to generate routine. By constraints they mean, some conditions are used by them. They generated a routine containing exam date, class room and etc. They also build an interface to show the process using the RGB colours [2].
- Another one was found using genetic algorithm where they build a school timetable. The functions they do are scheduling classes, teachers and rooms into a fixed number of periods, in such a way that no teacher, class or room is used more than once per period [6].
- Some other finding are: Cacchiani et al (2013) is described a new solution through the computation of improved lower boundaries. In the same way, Qauroonia and Akbarzadeh (2013) proposed and studied a specific genetic algorithm. Furthermore, Rahman et al (2014) showed an adaptive linear combination of graph colouring to solve this problem, and Sørensen and Dahms (2014) presented a model based on Integer Programming [3].

2.3 Possible Approaches

At last, we begin with the possible solutions and in order to do that we need to learn about all the ways through which we can generate a routine. And from there, we can pick up a way which would be suitable for our system and also it should be unique, something that is not done by others. So that, we can proudly say that, it's our invention and our own system. The topics for further research are taken from already existing routine generating system, like how they created it, what did they use etc. , and also some other topics that were new and did not used before. So the topics at a glance are: Job Scheduling Algorithm, Back Propagation, Neural Networks, and etc.

Chapter 3

In this chapter we discuss the different ways the routine generation problem can be solved.

3.1 Big Data

Big data is a massive volume of both structured and unstructured data. It is so large and complex that it becomes difficult to process using traditional database and software techniques. It has the potential to help companies improve operations and make faster and more intelligent decisions. Usually includes the data that is too big or it moves too fast or it exceeds current processing capacity. It can be described by following characteristics:

- Volume – It is the size of the data which determines the value and potential of the data under consideration and whether it can actually be considered as Big Data or not.
- Variety – It is the category to which Big Data belongs to. This helps the data analytics to effectively use the data to their advantage and thus upholding the importance of the Big Data.
- Velocity – It refers to the speed of generation of data or how fast the data is generated and processed to meet the demands and the challenges which lie ahead in the path of growth and development.
- Variability – It refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.
- Complexity - Data management can become a very complex process when large volumes of data come from multiple sources. These data need to be linked, connected and correlated in order to be able to grasp the information that is supposed to be conveyed by these data. This situation is known as the ‘complexity’ of Big Data.

An example of big data might be petabytes (1,024 terabytes) or exabytes (1,024 petabytes) of data consisting of billions to trillions of records of millions of people—all from different sources

(e.g. Web, sales, customer contact centre, social media, mobile data and so on). The data is typically loosely structured data that is often incomplete and inaccessible.

Big data is important to business, society and as well as to internet because more data lead to more accurate analysis and accurate analysis may lead to more confident decisions making and better decisions can mean greater operational efficiencies, cost reductions and reduced risk. It requires exceptional technologies to efficiently process large quantities of data within tolerable elapsed times.

3.2 Data Mining

Data mining is the activity of going through big data sets to look for relevant or appropriate information and also used to provide beneficial results. The ultimate goal of data mining is prediction. The process of data mining consists of three stages:

(1) The initial exploration usually starts with data preparation which may involve cleaning data, data transformations.

(2) Model building or pattern identification is considering various models and choosing the best one based on their predictive performance.

(3) Deployment is the final stage which uses the model selected in the previous stage and applying it to new data in order to generate predictions or estimates of the expected outcome.

The idea is to collect massive sets of data that may be homogeneous or automatically collected. Sometimes we need to access smaller, more specific pieces of data from those large sets. So we use data mining to find the relevant data from big data. Data mining can involve the use of different kinds of software packages. Generally, data mining refers to operations that involve all searched operations that return targeted and specific results. For example, a data mining tool may look through dozens of years of accounting information to find a specific column of expenses or accounts receivable for a specific operating year.

3.3 Routine Management

It is managing the works, routines etc. of our day to day life using a program to make our life easier. It simply means the proper management of any type of work or resource used in an organization, that means which way the organization run, which protocol it maintain and why, what is its time schedule etc.

Each and every Organization must follow a routine because, to complete their total tasks in a sequential way .This process make all types of work so much easier because it maintain a schedule. It makes all types of work faster.

3.4 Job Scheduling With Constraints

Scheduling is a process of allocating resources to activities over time. In a typical scheduling problem, resources are scarce and constrained in various ways, and one is looking for a schedule of the activities that both satisfies the constraints and is optimal according to some criterion. In general a resource constrained scheduling problem consists of:

- A set of jobs that must be executed
- A finite set of resources that can be used to complete each job
- A set of constraints that must be satisfied
- Temporal Constraints–the time window to complete the task
- Procedural Constraints–the order each task must be completed
- Resource Constraints - is the resource available
- A set of objectives to evaluate the scheduling performance

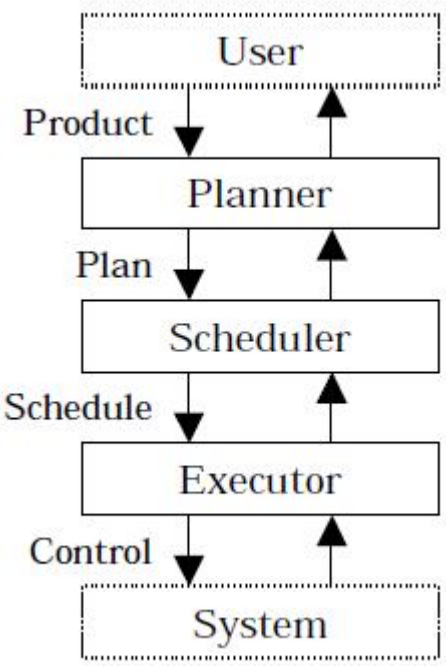


Figure 3.4.1 Scheduling In Context

Positioning of scheduling is between planning and execution. Basically, for a set of desired products, a planner determines the sequence of tasks to get the products, the scheduler decides the resources, operations and timing to perform the tasks and an executor directs the system's resources to perform the operations at the given time to produce the desired products. Scheduling tends to consider higher-level or aggregate operations and take a longer-term view, and thus scheduling problems are decidedly complex, combinatorial problems that are in general NP-hard problem (that is, there are no known algorithms for finding an optimal solution in polynomial time). At the same time, embedded schedulers are expected to have the usual properties of lower-level controllers, such as operating in parallel with the system's execution (on-line scheduling) and in a feedback loop (reactive scheduling), and providing results within hard real-time boundaries (real-time scheduling).

To solve this type of problems, use of **heuristic algorithms** is common. These methods tend to have problems when inputs become more complex and varied (NP-Hard problem).

Genetic algorithms can be used to solve production scheduling problems; these algorithms operate on a population of solution. This population of solutions consists of many answers that may have different sometimes conflicting objectives. For example, one solution may be optimizing for minimum amount of time, other solution may be optimizing for minimum amount of defects. To apply this algorithm to a scheduling problem we must first represent it as a genome. For example, we can define a sequence of tasks and the start times of those tasks as a genome of a scheduling problem, where each task and its start time represent a gene and specific sequence of tasks and its start times represents one genome of the population. For the initial population we can take random start times with precedence constraints. Then we take this initial population and cross it, combining genomes with some mutation (randomness). The offspring is then selected based on the requirements. We can continue this process for a defined amount of

time or until we find the solution that satisfy our minimum criteria. Each generation will take less time and will provide higher quality than the previous generations, and each constraint added will increase the search space and lowering the set of solutions.

Another technique to solve scheduling problems is **Constraint-Based Scheduling (CBS)**. This method separates the model (description of activities, resources, constraints and objectives) from the algorithm that solves it. This separation gives the advantage to deal with a wider variety of constraints, changing the model (even without changing the algorithm) and re-use the model for other tasks (simulation, planning and diagnosis).

Scheduling is a constraint satisfaction and optimization problem. In constraint based scheduling, tasks are represented by resource selection and timing variables, possibly constrained by precedence constraints, resource constraints, routing conditions, and other restrictions. The search space is the space of possible assignments of tasks to resources and the concrete timings of tasks. As tasks are assigned to resources, their timing variables are further constrained by resource constraints (such as limited capacities and setup delays). A solution (schedule) is an assignment to the resource and timing variables that specifies when the tasks have to be executed on what resources. Modern constraint solvers have their roots in three research communities: **Artificial Intelligence (AI)**, which investigated **constraint satisfaction problems (CSPs)** with a focus on general search procedures; Logic Programming, which led to **constraint (logic) programming (CP/CLP)** with a focus on modeling and programmability; and **Operations Research (OR)** with a focus on efficient algorithms for restricted problem formulations. Therefore, CSP research has focused on the basic means for representing and operating on constraints, in particular various forms of backtracking search algorithms.

$$
\begin{aligned}
&Given\ n\ variables\ x_i with\ domains\ D_i and\ m\\
&constraints\ c_j find\ a\ solution\ such\ that\\
&\langle x_1, \dots, x_n \rangle = \langle v_1, \dots, v_n \rangle\\
&v_i \in D_i \qquad i = 1, \dots, n\\
&c_j(v_i, \dots, v_n)\ \ j = 1, \dots, m
\end{aligned}
$$

Figure 3.4.2: Constrained Satisfaction Problem definition

A variable assignment is inconsistent if at least one constraint is not satisfied.

$$
\begin{aligned}
&Given\ n\ variables\ x_i with\ domains\ D_i and\ m\\
&constraints\ c_j and\ objective\ function\ h.\\
&find\ a\ solution\ with\ minimal\ subject\ to\\
&\langle x_1, \dots, x_n \rangle = \langle v_1, \dots, v_n \rangle\\
&h(v_1, \dots, v_n)\\
&v_i \in D_i \qquad i = 1, \dots, n\\
&c_j(v_i, \dots, v_n)\ \ j = 1, \dots, m
\end{aligned}
$$

Figure 3.4.3: Constrained Optimization Problem Definition

To model the CP we need to define variables. These variables may have different types of domains: integer domain, logical domain, enumeration domain, real domain and set domains. Each variable type has its own types of constraints, such as arithmetic constraints for integer and real variables (e.g., 3x+y £ z). The constraints over integer and related variables are also called finite-domain constraints.

Solving constraint problems

The usual way of solving finite-domain CSPs (and COPs) in CP is a combination of domain reduction, constraint propagation, and search.

Domain reduction is the direct application of a unary constraint c(x) to the variable x. For example, if x is an integer variable with current domain [0, 10] and c is x > 3, then the domain of x becomes [4, 10].

Constraint propagation is the propagation of changes in one variable's domain to the domains of other variables connected by constraints. For example, if x and y are integer variables with current domains [0, 10], and a constraint x £ y–3 is added, we can immediately propagate the lower bound of x to y, i.e., y's domain becomes [3, 10], and we can propagate the upper bound of y to x, i.e., x's domain becomes [0, 7].

Propagation can often be attached to or "indexed" by components of a domain representation.

This realization has led to the introduction of indexical (or projection constraints), efficient specialized edges in a constraint graph that replace the original constraints [10]. For example, the constraint x £ y–3 can be represented by the two indexicallb(y):= max (lb(x)+3,lb(y)) and ub(x) :=min(ub(y)–3,ub(x)), where lb and ub denote the lower and upper bounds of the domains, respectively.

Constraint propagation is an incomplete technique, since it does not remove all possible combinations of values that are inconsistent. Because of this incompleteness, domain reduction and constraint propagation have to be complemented by search.

Search methods in constraint programming may be divided into refinement-based methods and repair-based methods.

Refinement-based methods are the common form of search in constraint programming. Each of the variables is assigned a value incrementally until a complete solution is found or a constraint is violated ("labeling"). If a constraint is violated, the last assignment is undone and an alternative value is chosen ("enumeration"). If no value assignment is consistent, search backtracks to a previously assigned variable, and so on. The result is a depth-first tree search.

$set\ X = \langle x_1, \dots, x_n \rangle$
$while\ there\ are\ unassigned\ variables\ in\ X$
  $select\ an\ unassigned\ variable\ x\ from\ X$;
  $select\ a\ value\ v\ from\ the\ domain\ of\ x$;
  $assign\ x = v$;
  $backtrack\ if\ a\ constraint\ is\ violated$;
$end\ while$

Figure 3.4.4: Refinement-based methods

The order in which variables and values are selected can have a significant impact on search efficiency.

Backtracking procedure is applied on these refinement-based methods; one method of the backtracking is **Chronological backtracking** (backtracking to and undoing previous variable and value selections in reverse order of assignment).

If the constraint problem is an optimization problem, a refinement-based search can be augmented with a mechanism that adds a new constraint $h(x1,\dots,xn) < h(v1,\dots,vn)$ every time a new solution $\_v1,\dots,vn\_$ is found. This leads that subsequent solutions to have increasingly better objective values and can be very effective in removing parts of the search tree. The last-found solution is returned as the optimal solution (this kind of optimizing search has been called *branch-and bound search*). A variation of this technique is *binary search*, which keeps progressively narrower lower and upper bounds l and u on $h(x1,\dots,xn)$.

A more recent technique is *limited discrepancy search*, which assumes that the chosen value heuristic makes few mistakes, and which therefore initially limits the number of allowed deviations from the heuristic.

Repair-based methods start with a complete assignment to the variables. If this assignment is inconsistent (at least one constraint is violated), the assignment is "repaired" iteratively by assigning different values to one or more of the variables until a solution is found.

$set\ V = \langle v_1, \dots, v_n \rangle\ as\ initial\ solution\ for\ \langle x_1, \dots, x_n \rangle;$
$while\ V\ is\ inconsistent$
  $select\ an\ inconsistent\ assignment\ x = v\ from\ V;$
  $select\ a\ new\ value\ v'\ for\ x;$
  $assign\ x = v'\ in\ V;$
$end\ while$

Figure 3.4.5: Repair-based methods

These methods can be combined with optimization techniques such as hill climbing or simulated annealing. Repair-based methods typically are not complete (i.e., they are not guaranteed to find the global optimum or even a variable assignment that satisfies all the constraints). Therefore, repair-based methods typically require additional termination criteria (such as an upper limit on the number of repairs).

How to develop a Constraint-based Scheduling:

1. **Modelling scheduling problems**

   Building on the CP representations and techniques introduced above various variable and constraint types have been developed specifically for scheduling problems. Variable domains include interval domains where each value is an interval (e.g., start and duration); resource variables for various classes of resources.

   Scheduling-specific constraints include interval constraints for interval variables (e.g., $t1 \leq t2$ to express that task 1 has to occur before task 2); resource constraints for timing (integer or interval) variables (e.g., allocate(*r*, *t*) for resource *r* and interval *t* to express that the task occupies resource *r* during interval *t*).

   Also higher-level constraints may be defined in terms of lower-level constraints.

Resource constraints define and constrain the available resources by restricting how multiple uses of a resource can be combined. The resources are either renewable or consumable. Unary resources *r* can handle only one task at a time. Volumetric resources (also called multi-unit or n-ary resources) allow tasks to overlap so long as the total amount of resource use at any time does not exceed a given capacity limit. State resources allow tasks to overlap if they require the same state (e.g., the state of a switch). In contrast to renewable resources, *consumable resources* get depleted with each use and have to be renewed explicitly. Finally, tasks may or may not be interruptible, leading to the distinction between preemptive and non-preemptive scheduling.

Some constraint programming systems provide *global constraints, for e*xample all-different constraint (forcing all variables in a given set to have different values) and the cardinality constraint (limiting the number of variables that a given value can be assigned to).

2. **Scheduling-specific propagation techniques**

   There are several techniques: one technique is *resource timetable*, where for each resource a timetable is maintained any time with required and available capacity.

   The propagation between the resources and tasks works in both ways: as a task time becomes fixed, the task resource usage is entered in the timetable; conversely, as available capacity in the timetable is reduced, the interval domains of associated tasks are updated.

   Another technique is *edge finding*, which reasons about the order in which tasks can execute on a given resource. Each task is evaluated with respect to a set of other tasks. If it is determined that the task must or cannot execute before (or after) these tasks, it may be possible to infer new precedence constraints and new bounds on the task's interval domain.

3. **Scheduling**

   Considering the example of job-shop scheduling problem, we can define it as a COP as follows: given a set of jobs and a set of machines. Each job consists of a set of tasks with given durations to be processed in a given order on assigned machines. Each machine can process only one task at a time. The problem is to find a schedule (that is, a start time

for each task) that minimizes the total length of the schedule (i.e., the time at which all jobs are finished). We can represent a task by an interval variable *t*, where *t*.s and *t*.d represent the start and duration times of the task, respectively, and *t*.e = *t*. s + *t*.d is its end time.

$$For\ every\ task\ t: 0 \leq t \leq 1;$$
$$For\ every\ task\ t, given\ duration\ d: t.d = d.$$
$$For\ every\ job\ with\ tasks\ (t_1, \dots, t_n);$$
$$t_i \leq t_{i+1}\ (i = 1, \dots, n-1);$$
$$For\ every\ task\ t, given\ assigned\ machine\ r;$$
$$allocate(r, t)$$

Figure 3.4.6: Applied Constrained

The objective function is defined as *h* = *l*, (the goal is to minimize *l)*.

In traditional, *off-line / predictive scheduling*, this problem is solved by posting the constraints to the constraint solver and then calling the search procedure (e.g., minimize(*X*, *l*), where *X* is the list of all task intervals *t* and the variable *l*).

In *on-line / reactive scheduling*, the tasks and their constraints may become known only incrementally, and the scheduler may run concurrently to the execution of a previous (partial) schedule. Furthermore, the constraints (such as the availability of resources) may change during scheduling and/or execution, in which case part or the entire problem has to be rescheduled. The simplest approach to on-line scheduling is to treat the scheduling problem as a sequence of constraint problems, and to transfer commitments (variable assignments being "executed") from one problem to the next. When embedding scheduling into a control system, a set of unique requirements come into play that require adapting the constraint solver to this environment. These issues include memory management, real-time issues in updating constraints and assigning variables, and the interface to the control application.

4. **On-line scheduling and model-predictive control**

Model-predictive control (MPC) takes a model-based approach in which the model, objectives, and constraints are stated explicitly as a COP. MPC shares with on-line

scheduling the incremental nature of processing incoming requests and the optimization of decisions with respect to a horizon of known or predicted future events.

In scheduling we are given *n* tasks with times *ti*, *m* constraints *cj*, and objective function *h*, resulting in the COP

$$\begin{aligned} &Find\ a\ solution\ with\ minimal\ subject\ to \\ &\langle t_1, \dots, t_n \rangle = \langle v_1, \dots, v_n \rangle \\ &h(v_1, \dots, v_n) \\ &c_j\ (v_1, \dots, v_n) \quad j = 1, \dots, m \end{aligned}$$

In on-line scheduling, this is the COP at a particular time step *k*, and the tasks _*t*1,…,*tn*_ represent the horizon at that time (with *ti*>*k*, *i*= 1,…, *n*). More tasks and constraints may be added in subsequent time steps as they become known. At each time step *k*, the goal, in its simplest form, is to identify which task(s) to start next. Depending on the solution times _*v*1,…,*vn*_, a single, multiple, or no tasks *ti* may be scheduled to execute next (concretely all those with times *vi* = *k*+1 in this example). The variables of these tasks thus become committed and no longer appear as variables at subsequent time steps. In a discrete state-space formulation of MPC, we start from a process model that predicts the evolution of system state *x* and system output *y* over time, given control input *u* and system, input, and output matrices *A*, *B*, and *C*, respectively. (*x*, *y*, and *u* are in general vectors.)

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_{k+1} &= Cx_k \end{aligned}$$

For example a control example, we are given a reference trajectory *ri*, *i*= *k*+1,…,*k*+*n*, for a horizon of *n* steps at time step *k*, together with *m* constraints *cj*, resulting in the COP

$$\begin{aligned} &Find\ a\ solution\ with\ minimal\ subject\ to \\ &\langle u_{k+1}, \dots, u_{k+n} \rangle = \langle v_{k+1}, \dots, v_{k+n} \rangle \\ &h(v_{k+1}, \dots, v_{k+n}) \\ &c_j\ (v_{k+1}, \dots, v_{k+n}) \quad j = 1, \dots, m \end{aligned}$$

Where *h* requires the system output to match the reference trajectory, e.g., with *yi* defined by the process model as a function of the current system state *xk* and the control inputs *ui* = *vi*. The constraints *cj* typically encode limits on control and output values. In contrast to tasks *ti* in scheduling, the order of reference points *ri* and

$$h = \sum_{i=k+1}^{k+n} (r_i - y_i)^2$$

control inputs *ui* is fixed and given. Also, the solution values *vi* in MPC do not represent different tasks, but the values of the same control "task" at different times. Thus, at each time step *k*, the goal is to determine at what value to set the next control input (*uk*+1). The resulting state (*xk*+1) becomes the new "start" state for the next time step, and the horizon is extended accordingly.

From a constraint programming perspective, online scheduling and MPC have several common requirements. Both work incrementally on a stream of requests (tasks, reference points), and have to be reactive in the sense that tasks, states, and reference trajectories may change over time from their original or predicted values. Furthermore, both generally use a *full optimization with minimal commitment* approach: while the variable assignment for the next step to be executed is part of an optimal solution for all variables, the other variables are to be kept open in case more information becomes available (in the form of new or changed tasks and reference points, respectively). In other words, the optimizer is to return a solution for the next step only, but guarantee that it is part of an optimal solution for all known future steps.

3.5 Neural Network

One efficient way of solving complex problems is following the lemma "divide and conquer". A complex system may be decomposed into simpler elements, in order to be able to understand it. Also simple elements may be gathered to produce a complex system. Networks are one approach for achieving this. There are a large number of different types of networks, but they all are characterized by the following components: a set of nodes, and connections between nodes.

The nodes can be seen as computational units. They receive inputs, and process them to obtain an output. This processing might be very simple (such as summing the inputs), or quite complex (a node might contain another network...)

The connections determine the information flow between nodes. They can be unidirectional, when the information flows only in one sense, and bidirectional, when the information flows in either sense.

The interactions of nodes though the connections lead to a global behavior of the network, which

cannot be observed in the elements of the network. This global behavior is said to be *emergent*. This means that the abilities of the network super cede the ones of its elements, making networks a very powerful tool.

Networks are used to model a wide range of phenomena in physics, computer science, biochemistry, ethnology, mathematics, sociology, economics, telecommunications, and many other areas. This is because many systems can be seen as a network: proteins, computers, communities, etc.

One type of network sees the nodes as 'artificial neurons'. These are called artificial neural networks (ANNs). An artificial neuron is a computational model inspired in the natural neurons. Natural neurons receive signals through synapses located on the dendrites or membrane of the neuron. When the signals received are strong enough (surpass a certain threshold), the neuron is activated and emits a signal though the axon. This signal might be sent to another synapse, and might activate other neurons.

Functions: The complexity of real neurons is highly distant when modeling artificial neurons. These basically consist of inputs (like synapses), which are multiplied by weights (strength of the respective signals), and then computed by a mathematical function which determines the activation of the neuron. Another function (which may be the identity) computes the output of the artificial neuron (sometimes in dependence of a certain threshold). ANNs combine artificial neurons in order to process information.

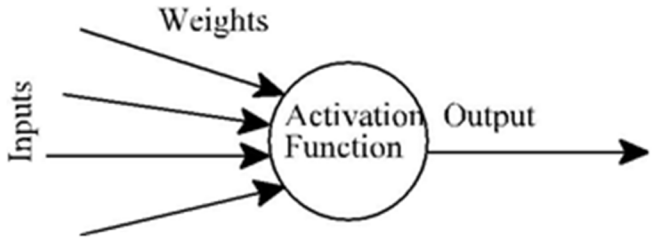


Figure 3.5.1: Neural Network

The higher a weight of an artificial neuron is, the stronger the input which is multiplied by it will be. Weights can also be negative, so we can say that the signal is *inhibited* by the negative weight. Depending on the weights, the computation of the neuron will be different. By adjusting the weights of an artificial neuron we can obtain the output we want for specific inputs. But

when we have an ANN of hundreds or thousands of neurons, it would be quite complicated to find by hand all the necessary weights. But we can find algorithms which can adjust the weights of the ANN in order to obtain the desired output from the network. This process of adjusting the weights is called *learning* or *training*.

The number of types of ANNs and their uses is very high. The differences in them might be the functions, the accepted values, the topology, the learning algorithms, etc. Because of matters of space, we will present only an ANN which learns using the back propagation algorithm for learning the appropriate weights, since it is one of the most common models used in ANNs, and many others are based on it.

3.6 Back Propagation

The back propagation algorithm is used in layered feed-forward ANNs. This means that the artificial neurons are organized in layers, and send their signals "forward", and then the errors are propagated backwards. The network receives inputs by neurons in the *input layer*, and the output of the network is given by the neurons on an *output layer*. There may be one or more intermediate *hidden layers*. The back propagation algorithm uses supervised learning, which means that we provide the algorithm with examples of the inputs and outputs we want the network to compute, and then the error (difference between actual and expected results) is calculated. The idea of the back propagation algorithm is to reduce this error, until the ANN *learns* the training data. The training begins with random weights, and the goal is to adjust them so that the error will be minimal.

The activation function of the artificial neurons in ANNs implementing the back propagation algorithm is a weighted sum (the sum of the inputs x_i multiplied by their respective weights w_{ji}):

$$A_j(\bar{x}, \bar{w}) = \sum_{i=0}^{n} x_i w_{ji} \qquad (1)$$

We can see that the activation depends only on the inputs and the weights.

If the output function would be the identity (output=activation), then the neuron would be called

linear. But these have severe limitations. The most common output function is the sigmoidal function:

$$O_j(\bar{x}, \bar{w}) = \frac{1}{1 + e^{A_i(\bar{x}, \bar{w})}} \tag{2}$$

The sigmoidal function is very close to one for large positive numbers, 0.5 at zero, and very close to zero for large negative numbers. This allows a smooth transition between the low and high output of the neuron (close to zero or close to one). We can see that the output depends only in the activation, which in turn depends on the values of the inputs and their respective weights.
Now, the goal of the training process is to obtain a desired output when certain inputs are given. Since the error is the difference between the actual and the desired output, the error depends on the weights, and we need to adjust the weights in order to minimize the error. We can define the error function for the output of each neuron:

$$E_j(\bar{x}, \bar{w}, d) = (O_j(\bar{x}, \bar{w}) - d_j)^2 \tag{3}$$

We take the square of the difference between the output and the desired target because it will be always positive, and because it will be greater if the difference is big, and lesser if the difference is small. The error of the network will simply be the sum of the errors of all the neurons in the output layer.

$$E_j(\bar{x}, \bar{w}, d) = \sum_j (O_j(\bar{x}, \bar{w}) - d_j)^2 \tag{4}$$

The back propagation algorithm now calculates how the error depends on the output, inputs, and weights. After we find this, we can adjust the weights using the method of *gradient descendent*:

$$\Delta \mathrm{w}_{ji} = -\eta \frac{\partial \mathrm{E}}{\partial w_{ji}} \tag{5}$$

This formula can be interpreted in the following way: the adjustment of each weight ()w_{ji}) will be the negative of a constant eta (0) multiplied by the dependence of the previous weight on the error of the network, which is the derivative of E in respect to w_i. The size of the adjustment will depend on 0, and on the contribution of the weight to the error of the function. This is, if the

weight contributes a lot to the error, the adjustment will be greater than if it contributes in a smaller amount. Equation (5) is used until we find appropriate weights (the error is minimal). If you do not know derivatives, don't worry, you can see them now as functions that we will replace right away with algebraic expressions. If you understand derivatives, derive the expressions yourself and compare your results with the ones presented here. If you are searching for a mathematical proof of the back propagation algorithm, you are advised to check it in the suggested reading, since this is out of the scope of this material.

So, we "only" need to find the derivative of E in respect to w_{ji}. This is the goal of the back propagation algorithm, since we need to achieve this backwards. First, we need to calculate how much the error depends on the output, which is the derivative of E in respect to O_j (from (3)).

$$\frac{\partial \mathrm{E}}{\partial O_j} = 2(O_j - d_j) \tag{6}$$

And then, how much the output depends on the activation, which in turn depends on the weights (from (1) and (2)):

$$\frac{\partial O_j}{\partial w_{ji}} = \frac{\partial O_j}{\partial A_j}\frac{\partial A_j}{\partial w_{ji}} = O_j(1 - O_j)x_i \tag{7}$$

And we can see that (from (6) and (7)):

$$\Delta v_{ik} = -\eta \frac{\partial E}{\partial v_{ik}} = -\eta \frac{\partial E}{\partial x_i}\frac{\partial x_i}{\partial v_{ik}} \tag{8}$$

And so, the adjustment to each weight will be (from (5) and (8)):

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial w_{ji}} = 2(O_j - d_j)O_j(1 - O_j)X_i \tag{9}$$

We can use (9) as it is for training an ANN with two layers. Now, for training the network with one more layer we need to make some considerations. If we want to adjust the weights (let's call them v_{ik}) of a previous layer, we need first to calculate how the error depends not on the weight, but in the input from the previous layer. This is easy, we would just need to change x_i with w_{ji} in (7), (8), and (9). But we also need to see how the error of the network depends on the adjustment

of v_{ik}. So:

$$\Delta w_{ji} = -2\eta(O_j - d_j)O_j(1 - O_j)X_i \quad (10)$$

Where:

$$\frac{\partial E}{\partial w_{ji}} = 2(O_j - d_j)O_j(1 - O_j)w_{ji} \quad (11)$$

And, assuming that there are inputs u_k into the neuron with v_{ik} (from (7)):

$$\frac{\partial x_i}{\partial v_{ik}} = X_i(1 - x_i)v_{ik} \quad (12)$$

If we want to add yet another layer, we can do the same, calculating how the error depends on the inputs and weights of the first layer. We should just be careful with the indexes, since each layer can have a different number of neurons, and we should not confuse them.

For practical reasons, ANNs implementing the back propagation algorithm do not have too many layers, since the time for training the networks grows exponentially. Also, there are refinements to the back propagation algorithm which allow a faster learning.

How the training is done: The network is first initialized by setting up all its weights to best all random numbers–say between–1and+1.Next, the input pattern is applied and the output calculated (this is called the forward pass).The calculation gives an output which is completely different to what you want (the Target), since all the weights are random. We then calculate the error of each neuron, which is essentially: Target-Actual Output (What you want –What you actually get). This error is then used mathematically to change the weights in such a way that the error will get smaller. In other words, the Output of each neuron will get closer to its Target (this part is called the reverse pass). The process is repeated again and again until the error is minimal.

When to stop training: Then network keeps training all the patterns repeatedly until the total error falls to some pre-determined low target value and then it stops. Note that when calculating the final error used to stop the network (which is the sum of all the individual neuron errors for each pattern) you need to make all errors positive so that they add up and do not subtract (an error of -0.5 is just as bad as an error of +0.5).

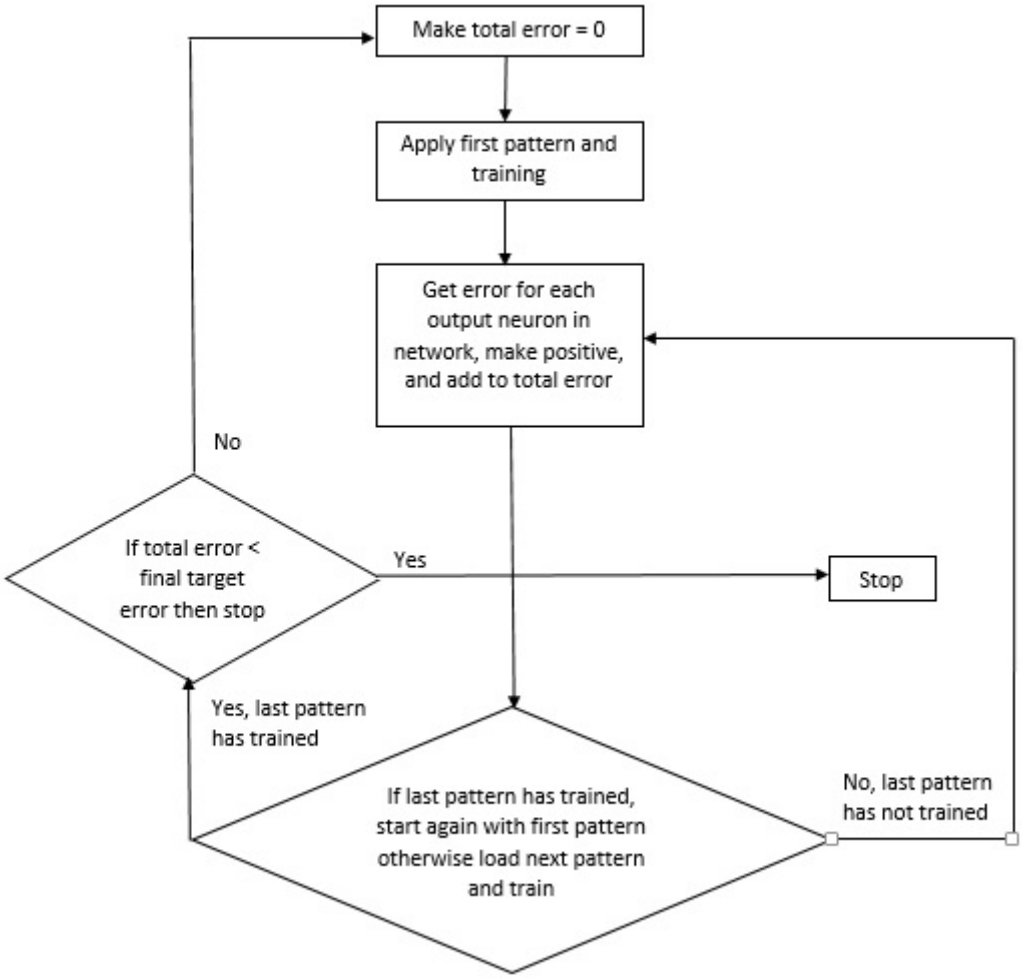


Figure 3.6.1: Back Propagation Training Diagram

A better way to stop training: It has been trained; it would be able to recognize not just the perfect patterns, but also corrupted or noisy versions. There is a better way of knowing when to stop the network training–by using a Validation Set. This stops the network over training (becoming too accurate, which can lessen its performance). It does this by having a second set of patterns which are noisy versions of the training set (but aren't used for training themselves). Each time after the network has trained; this set is called the Validation Set and is used to calculate an error. When the error becomes low the network stops.

Problems with this algorithm : The best known is called "Local Minima", and occurs because the algorithm always changes the weights in a way which causes the error to fall. But the error might briefly have to rise as part of a more general fall. If this is the case, the algorithm will "gets stuck" (because it can't go up hill) and the error will not decrease further. It contains

other problems as well. These tend to manifest themselves as the network gets larger, but many can be overcome by reinitializing the weights to different starting values.

Solutions of problems: There are several solutions to the problem found. One is very simple and that is to reset the weights to different random numbers and try training again (this can also solve several other problems). Another solution is to add “momentum” to the weight change. This means that the weight change this iteration depends not just on the current error, but also on previous changes done.

3.7 Decision Tree

A decision tree is a decision support tool that uses a tree-like graph. A schematic tree-shaped diagram used to determine a course of action or show a statistical probability. Each branch of the decision tree represents a possible decision or occurrence. The tree structure shows how one choice leads to the next, and the use of branches indicates that each option is mutually exclusive. The tree also shows the possible consequences, including utility, chance event outcomes, and resource costs.

A decision tree is used as a visual and analytical decision support tool. A decision tree can be used to clarify and find an answer to a complex problem. On a decision tree, the expected values of competing alternatives are calculated. The structure allows users to take a problem with multiple possible solutions and display it in a simple, easy-to-understand format that shows the relationship between different events or decisions. The furthest branches on the tree represent possible end results.

A decision tree consists of 3 types of nodes:

1. Decision nodes - These are commonly represented by squares. Decision nodes are used when a decision needs to be made between at least two alternatives.

2. Chance nodes - These are represented by circles. Chance nodes represent a point on the decision tree in which there is a degree of uncertainty about the outcomes of a decision, so there must be at least two possible outcomes represented.

3. End nodes - These are represented by triangles. An end node is where a decision is made and its value or utility is identified.

When to Consider Decision Trees

Each instance consists of an attribute with value pairs. The classification is over discrete values (e.g. yes/no). It can have disjunctive descriptions –each path in the tree represents a disjunction of attribute combinations. Any Boolean function can be represented. It is okay for the training data to contain errors – decision trees are robust to classification errors in the training data. The training data can contain missing values – decision trees can be used even if instances have missing attributes.

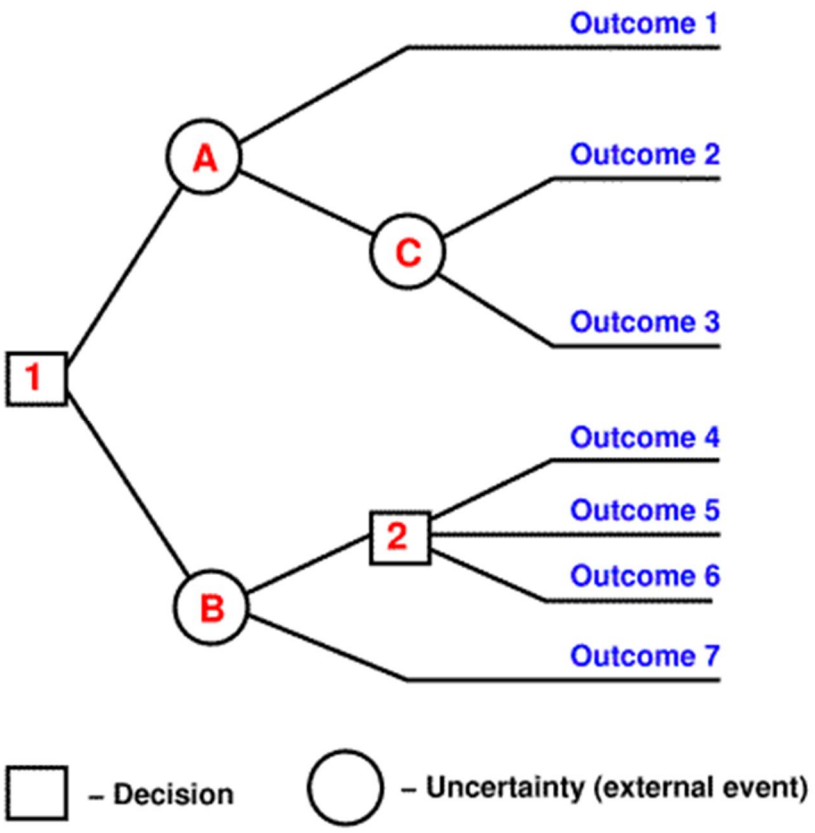


Figure 3.7.1: Decision Tree

Decision trees have several advantages. Decision trees:

- Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.
- Have value even with little hard data. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes.
- Allow the addition of new possible scenarios
- Help determine worst, best and expected values for different scenarios
- Use a white box model. If a given result is provided by a model.

White box model is a subsystem whose internals can be viewed, but usually cannot be altered.

Disadvantages of decision trees:

- For data including categorical variables with different number of levels, information gain in decision trees are biased in favour of those attributes with more levels.
- Calculations can get very complex particularly if many values are uncertain and/or if many outcomes are linked.

Decision trees used in data mining are of two main types:

- **Classification tree** analysis is when the predicted outcome is the class to which the data belongs.

- **Regression tree** analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).

The term **Classification and Regression Tree (CART)** analysis is an umbrella term used to refer to both of the above procedures. Trees used for regression and classification have some similarities - but also some differences, such as the procedure used to determine where to split.
Some techniques, often called *ensemble* methods, construct more than one decision tree:

- Bagging decision trees, an early ensemble method, builds multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction.

- A Random Forest classifier uses a number of decision trees, in order to improve the classification rate.
- Rotation forest - in which every decision tree is trained by first applying principal component analysis (PCA) on a random subset of the input features.

Top – down induction of decision tree:

Basic Algorithm:

1. A ← the "best" decision attribute for a node N.

2. Assign A as decision attribute for the node N.

3. For each value of A, create new descendant of the node N.

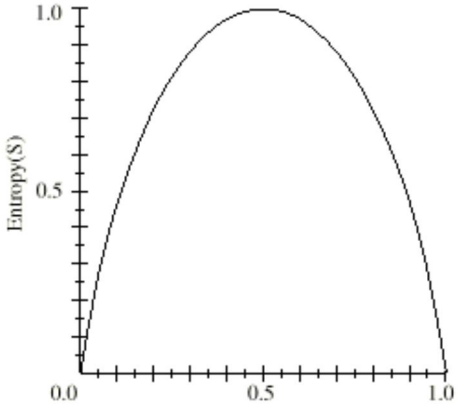
4. Sort training examples to leaf nodes.

5. IF training examples perfectly classified, THEN STOP.

6. ELSE iterate over new leaf nodes

Entropy:

Entropy is a measure of how much we know about a particular class. The more we know, lower the entropy.



Let S be a sample of training examples, and
p+ is the proportion of positive examples in S and
p-is the proportion of negative examples in S.
Then: entropy measures the impurity of S:
$E(S) = -p+ \log_2 p+ - p-\log_2 p-$

Figure 3.7.2: Entropy

Information gain:

$$Gain(S, A) = E(S) - \sum_{v|eValues(A)} \frac{|S_v|}{|S|} E(S_v)$$

Gain(S, A) = expected reduction in entropy due to sorting A. Where SV= {s ∈ S| A(s) = V}

ID3 Algorithm

ID3 algorithms determine the attribute with the highest information gain on the training set.

It uses this attribute as the root; create a branch for each of the values the attribute can have.

For each branch, repeat the process with subset of the training set that is classified by that branch.

Hypothesis Space Search in ID3

- The hypothesis space is the set of all decision trees defined over the given set of attributes. ID3's hypothesis space is a compete space; i.e., the target description is there.
- ID3 performs a simple-to-complex.
- ID3The evaluation function is the information gain.
- ID3 maintains only a single current decision tree.
- ID3 performs no backtracking in its search.
- ID3 uses all training instances at each step of the search.

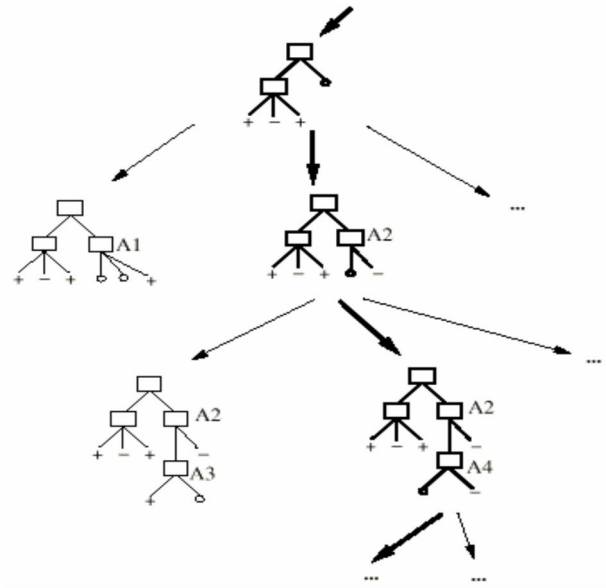


Figure 3.7.3: ID3 Algorithm

Chapter 4

This chapter describes briefly about the proposed solution and how we choose it for the system.

4.1 Comparison with Different Techniques

To choose the best way to accomplish our proposed solution, we compare all the available techniques. First of all, decision tree and CSP always give solution if exists, where as G.A. does not always gives solution. Secondly, we store all branches in decision tree, and in G.A. we only store the states, and for CSP we store all the states. Decision tree and CSP uses large amount of space, where as G.A. uses less amount of space. Lastly, Decision tree and CSP both cannot provide variation and randomness, but G.A. provides variation and randomness.

4.2 Proposed Solution by Comparing with an Existing System

There exists another system similar to our system and is built by the NSU students. They used decision tree for generating routines but we used genetic algorithm instead. Decision tree requires a lot of time compared to genetic algorithm to reach to a solution. It does not provide randomness and variation, thus always gives the same solution for a problem. And also there exists a possibility of taking a wrong decision. Whereas genetic algorithm works at a faster rate and is more effective and efficient compared to that of decision tree. Also it provides randomness and variation, which give various results of same problem. There is a disadvantage of using genetic algorithm that is it sometime falls in an infinite loop and does not give solution, but this is minor compared to all the advantages it provides. After looking at all the sides of the glitches, we choose genetic algorithm for generating routines in our system which is the intelligent routine management system.

4.3 Final Proposed Solution

To implement our Intelligent Routine Management (IRM) we studied the existing different approaches and comparing among all of them, we have chosen Genetic Algorithm (GA) to build our system. The main reason behind selecting GA is because of its great ability to generate different outputs while executing the same program several times which provides great variability and a larger range of output.

Chapter 5

This chapter describes briefly about the proposed solution and how we customized it for the system.

To implement our *Intelligent Routine Management* (IRM) we studied the existing different approaches; among all of them we have chosen *Genetic Algorithm* (GA) to build our system. The main reason behind selecting GA is because of its great ability to generate different outputs while executing the same program several times which provides great variability and a larger range of output.

Our main focus in order to build the system it was generate a routine where each and individual faculty's preferences is the primary constraint. We try to build a system where everyone will have their preferable routine.

In order to build the required routine we create each faculty's individual routine; in this part we mainly focus on each faculty's preferable days and times to take classes, according to that we build our fitness function of the system. After that, we merge all the individual routines and generate the whole routine.

Basically we have used three classes to build our system: one for generating the individual faculty's routine, one for merging the individual routines and another to allocate rooms to the generated routine courses. We are also using a database to access to the required information for generating routine. The given diagram below is the system architecture:

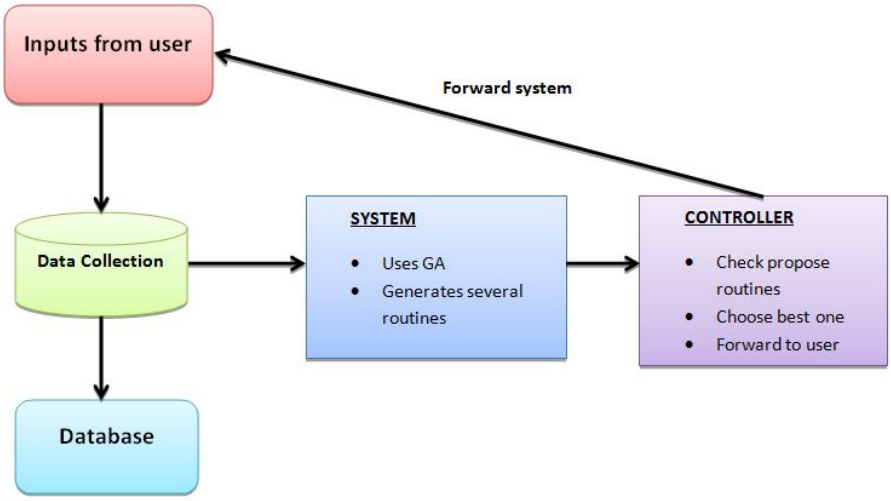


Figure 5.0.1: System Architecture

5.1 Individual Routine

We take as input a faculty's initial, courses taken and sections and his or her preferable days and times. We collect all the required data by using a database. Here, we use our newly created GA to generate individual faculty's routine.

- *Generating random routines:*

  After having all the required data, we generate two different random routines, one assigned as *parent1* and another as *parent2.*

- *Fitness Function:*

  The fitness function checks the random generated routine and gives it a fitness value according to its acceptance to the faculty's preferences. Basically, the faculty's preferable days and times are the keys factors in this method and as our purpose is to achieve a routine where each faculty's preferences has to be considered as the most important, this function is the more impactful function of our system.

In this function, the given routine is checked according to the faculty's preferences. If any course appears in a day which is not preferable by the faculty or the time slot assigned is not preferable then the fitness value is incremented by 1. All the courses in the routine are checked in this function and according to the criteria the final fitness value is given.

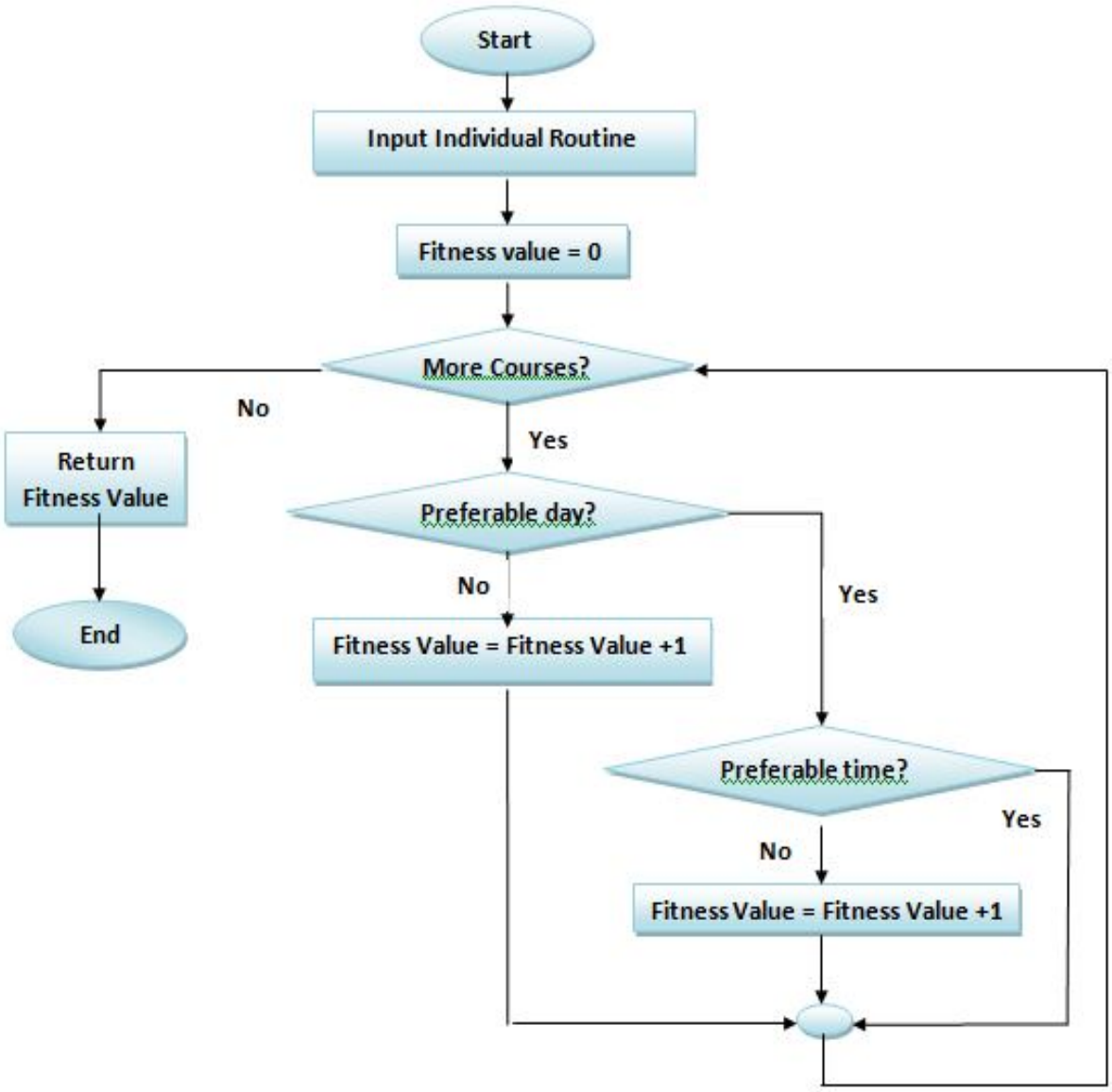


Figure 5.1.1: Fitness Function Flowchart

The *parent1* and *parent2* after their generation, the fitness function is applied to them. Their fitness values are checked then, if any of them has the lowest fitness value which is zero then we have found a solution and didn't proceed further otherwise we go to the next step.

- *Crossover Function:*

  In this function, *parent1* and *parent2* routines are partially modified to generate new two routines: *child1* and *child2*.

  In our system, we have chosen two-point crossover technique to implement the crossover function of our GA. Basically, this process consists of recombination of the two parents to generate new children. In our system, we are taking each parent array from the two indexes left from the half to one index more than the half.

$$h = \frac{array\ length}{2}$$

$$Condition: start\ c = h - 2\ until\ c = h + 1$$

  Then we swap the *parent1* content with the *parent2* content to generate the children. Both children execute the fitness function to get their fitness values. After getting all the fitness values for the parents and the newly created children we find two routines with the minimum fitness value among all of them and defined them as the new parents.

  This crossover function continues until we found a routine with fitness value zero or after executing it twenty times is doesn't give the required output then we apply mutation function.

- *Mutation Function:*

  In this function, the given parent's content is changed randomly. In our system, we have chosen boundary mutation technique to implement the mutation function of our GA. Basically, we randomly generate two indexes and swap content between them. In our system, when we apply the mutation function we applied it to both parents.

As a matter of fact, GA is a technique which provides great variability of output, however sometimes is cannot provide output to meet the required conditions. In behalf of that in our system there is a counter which counts till one hundred thousand if the solution is not found until that we discard the existent parents and generate two new parents randomly and continue the steps again. Using this counter we are ensuring that we will get a result to generate the routine.

5.2 Merged Routine

After having all the individual routines generated, we merge them all together to form the final routine. In this process, we take all the routines and add the content of the each routine with the other.

5.3 Room Allocation

In the generated merged routine there are not rooms assigned for the classes. In this process, we allocate rooms to the classes. In our system database we have the rooms available for the classes and we retrieve that information to allocate them. This allocation process is done randomly where each course will have a randomly chosen room but taking in count that rooms does not clash in any time slot or days.

5.4 System Diagram

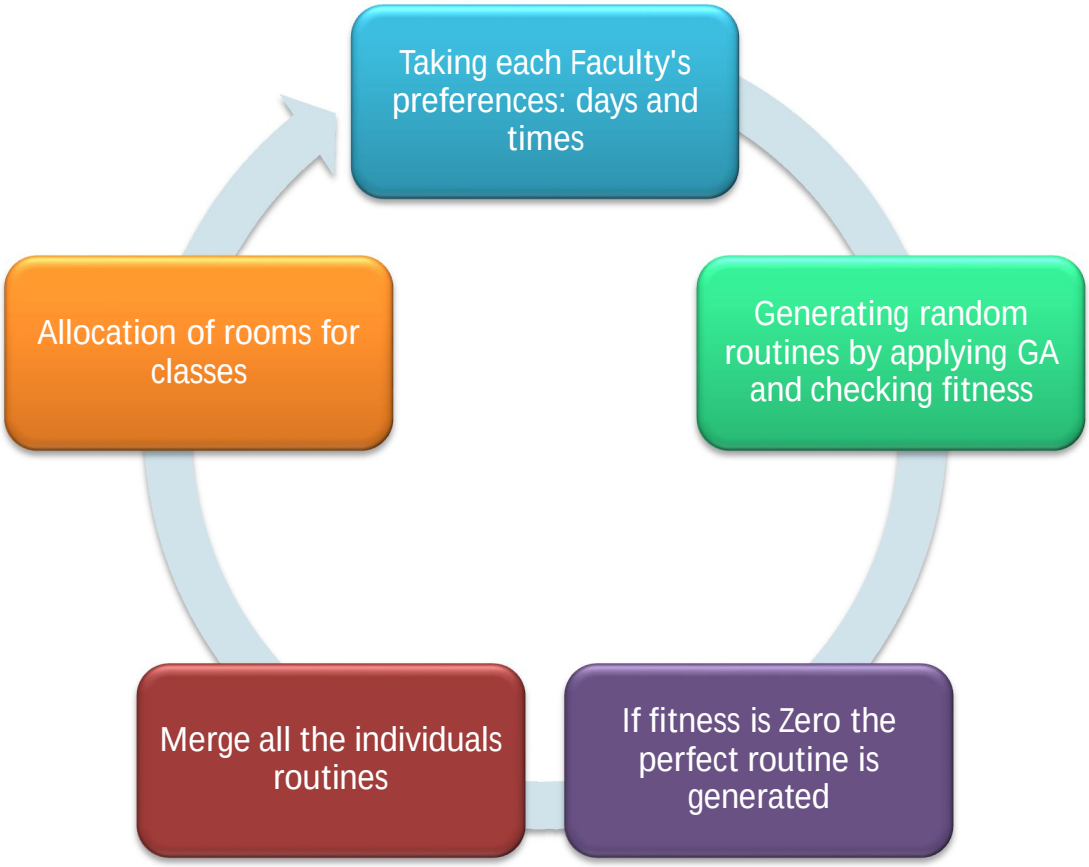


Figure 5.4.1: System Diagram

Chapter 6

This chapter describes the ways of collecting data and using these and chapter four, how we implemented it.

6.1 Data Collection

To begin our work, we needed data, so the first step is data collection. We collected the data of faculties, courses, and room numbers of Computer Science and Engineering Department of BRAC University. For each faculty, we took their full name, initial, designation, preferred day and time. For courses, we took course title and section. And on addition to that, we linked the section number of courses with the faculty taking that section. Now putting everything together, we started our process of building the Intelligent Routine Management System (IRM).

- Faculty initial, name and designation
- Preferred days from the faculty
- Suitable times of taking classes from the faculty
- Course title and id
- Room numbers
- NSU routine as sample

6.2 Methodology

Firstly, we made different tables using MySQL, for faculty, courses, and another table joining section with faculties. After doing this, we created database user access so that we can use it for further process.

Secondly, we created routines for individuals according to the tables, and using the customized genetic algorithm. Inside the G.A. we do crossover, mutation, fitness and etc. as explained

earlier, and by doing so we achieved a perfect, effective and efficient routine for an individual. And the routine contains an individual's day, time, course id, and section of course.

Now, we combine all the routines generated for individual faculties, and combine them to create one merged routine and allocate rooms to the routine which is then used as a semester routine. And lastly, we created an excel output of the final merged routine so that it is easy to use. This describes our whole process at a glance.

Chapter 7

This chapter describes the contents of this chapter. There are two sections – *results* and *graph*.

7.1 Results

Our system uses several inputs to generate a routine. In the input we took in count faculties' initials, courses taken with their respective section numbers, faculties preferable days and times and rooms numbers. In our system, we use as sample of BRAC University's CSE department.
This is the output format of our system for each individual faculty:

Example: CSE423[1]-MHA-UB30602

Here, first is the course code, within the brackets is the section number, next is the faculty initial and room assigned.

Our system provides two outputs: one is the output generated in the IDE and another is in excel format which is handier to use.

```
Individual Routine: MHA
CSE___[_]-___ , CSE423[2]-MHA , CSE___[_]-___ , CSE423[1]-MHA , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ ,
CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ ,
CSE___[_]-___ , CSE423[2]-MHA , CSE___[_]-___ , CSE423[1]-MHA , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ ,
CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ ,
CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ ,
```

Figure 7.1.1: Individual Routine Output

```
Merge Routine:
CSE111[1]-AAR , CSE423[2]-MHA , CSE423[1]-MHA , CSE___[_]-___ , CSE111[2]-MSA/CSE110[2]-AAR/CSE471[1]-MMM , CSE320[1]-SKZ , CSE_
CSE110[1]-MSN/CSE250[1]-SUP , CSE___[_]-___ , CSE___[_]-___ , CSE260[2]-MSN , CSE260[1]-MSN/CSE220[1]-NUS , CSE470[1]-HOS/CSE22:
CSE111[1]-AAR , CSE423[2]-MHA , CSE423[1]-MHA , CSE___[_]-___ , CSE111[2]-MSA/CSE110[2]-AAR/CSE471[1]-MMM , CSE320[1]-SKZ , CSE_
CSE110[1]-MSN/CSE250[1]-SUP , CSE___[_]-___ , CSE___[_]-___ , CSE260[2]-MSN , CSE260[1]-MSN/CSE220[1]-NUS , CSE470[1]-HOS/CSE22:
CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ , CSE___[_]-___ ,
```

Figure 7.1.2: Merged Routine Output

```
Merge Routine after room allocation:
CSE111[1]-AAR-UB30301, CSE423[2]-MHA-UB30401, CSE423[1]-MHA-UB30701, CSE___[_]-___, CSE111[2]-MSA-UB30601/CSE110[2]-AAR-UB30602/
CSE110[1]-MSN-UB30503/CSE250[1]-SUP-UB30502, CSE___[_]-___, CSE___[_]-___, CSE260[2]-MSN-UB30603, CSE260[1]-MSN-UB30602/CSE220[1
CSE111[1]-AAR-UB30301, CSE423[2]-MHA-UB30401, CSE423[1]-MHA-UB30701, CSE___[_]-___, CSE111[2]-MSA-UB30601/CSE110[2]-AAR-UB30602/
CSE110[1]-MSN-UB30503/CSE250[1]-SUP-UB30502, CSE___[_]-___, CSE___[_]-___, CSE260[2]-MSN-UB30603, CSE260[1]-MSN-UB30602/CSE220[1
CSE___[_]-___, CSE___[_]-___, CSE___[_]-___, CSE___[_]-___, CSE___[_]-___, CSE___[_]-___, CSE___[_]-___, CSE___[_]-___,
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 7.1.3: Final Routine IDE Output

	A	B	C	D	E	F	G
1	Sl no.	Faculty	Course	Section	Day	Time	Room
2	1	AAR	CSE111	1	Sun,Tues	08:00 - 09:20	UB30301
3	2	MHA	CSE423	2	Sun,Tues	09:30 - 10:50	UB30401
4	3	MHA	CSE423	1	Sun,Tues	11:00 - 12:20	UB30701
5	4	MSA	CSE111	2	Sun,Tues	02:00 - 03:20	UB30601
6	5	AAR	CSE110	2	Sun,Tues	02:00 - 03:20	UB30602
7	6	MMM	CSE471	1	Sun,Tues	02:00 - 03:20	UB30402
8	7	SKZ	CSE320	1	Sun,Tues	03:30 - 04:50	UB30502
9	8	MSN	CSE110	1	Mon,Wed	08:00 - 09:20	UB30503
10	9	SUP	CSE250	1	Mon,Wed	08:00 - 09:20	UB30502
11	10	MSN	CSE260	2	Mon,Wed	12:30 - 01:50	UB30603
12	11	MSN	CSE260	1	Mon,Wed	02:00 - 03:20	UB30602
13	12	NUS	CSE220	1	Mon,Wed	02:00 - 03:20	UB30501
14	13	HOS	CSE470	1	Mon,Wed	03:30 - 04:50	UB30603
15	14	NUS	CSE221	1	Mon,Wed	03:30 - 04:50	UB30403

Figure 7.1.4: Excel Output

As we can see in this output, we can easily differentiate each faculty's courses, his or her respective sections, assign rooms and time slots.

From the both outputs, we can visualize that the first format is difficult to understand and less readable than the second output which is more readable and understandable.

Our system generate a routine randomly however its complexity is less than the other existent methods to generate routine using GA. The reason behind is that we are mainly focusing on individual routines rather than running the GA for the whole merged routine which ultimately takes less time as the problem is divided in small parts and merged later, that is, we are using "divide and conquer" approach in our system.

7.2 Graph

From the excel output, we plotted a graph against the total number of classes and working days of our university. We plotted an individual faculty's routine on a particular day, and altogether show loads for a particular day. For each individuals bar, we used the excel output to find the courses taken by him and at which day. As said earlier, we used sample data containing 9 different faculties and their preferred date and time. This graphs also helps us to know individual faculty's load per day. And also the total number of classes that will be taken on that day by the faculties of our sample.

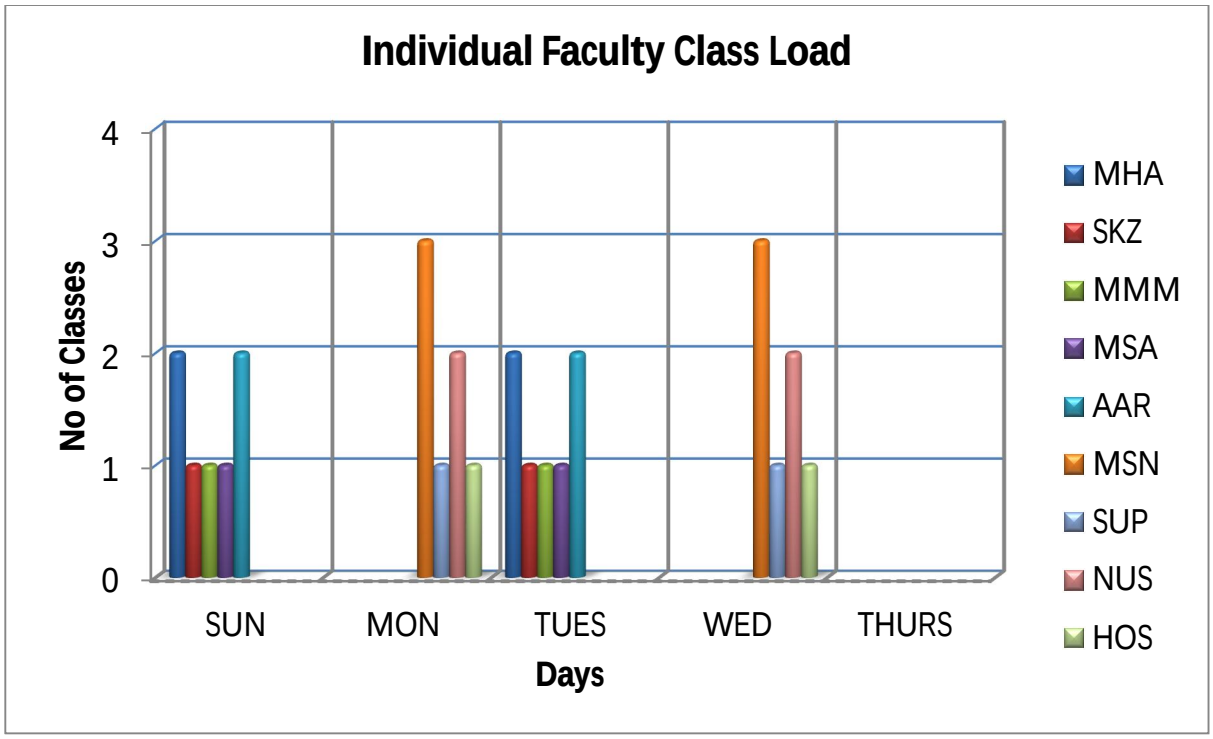


Figure 7.2.1: Individual Faculty Class Load

Chapter 8

This chapter describes the concluding notes of the authors for this thesis.

8.1 Limitations

Although this research was carefully prepared, we are still aware of its limitations and shortcomings. First of all, we have used sample data for our thesis. It would have been better if we had used large amount of data's which would increase the accuracy of our system. Secondly, in our intelligent routine management system input format retrieved from the database are defined. The format is the input have to come in a fixed sequence, like course id, section and etc. Thirdly, for the thesis we have made the routine for only one department. We have used the information of CSE department, since it would take a long time to gather information and data of all the departments. Our system prepares the routine for theory courses of CSE department.

8.2 Future Works

We would like to take our thesis intelligent routine management system to a whole new level. Firstly, we would like to add the lab timings with our routine. This would result in a perfect routine for the Computer Science and Engineering department (CSE).

Secondly, we would like to implementation it for all the departments. So that it will be helpful for the entire department to make the routine very easily and effectively.

In addition to that, including the examination routine of all courses will take it to a different level. It can be used to help a student selecting courses based two parameters, routine and examination, rather than one which makes it a complex process.

Lastly, we would like to design the interface such that, all the important information appears in a synchronized way. Like, each department routine would be kept differently not together. And each department's routine will contain the course id, room number assigned and faculty initials, rather than just the course id.

To summarize it all, we would like to add some features with the current system to make it a complete, efficient, effective and informative system.

Conclusion

Managing a routine is a difficult and important task for a university and many universities do this difficult task manually. To overcome this difficulty, we have tried to build an intelligence routine management system in our thesis by which a routine will be automatically generated using many data's. To implement our system we studied the existent different approaches; among all of them we have chosen *Genetic Algorithm* (GA) to build our system. The main reason behind selecting GA is because of its great ability to generate different outputs while executing the same program several times which provides great variability and a larger range of output. We have successfully built a system by which every faculty is able to have their preferable routine. Firstly, our system took faculty's full name, initial, designation, preferred day and time as faculty information and for courses, course title and section as input. With all this information we made different data tables, after that we created database user access so that we can use it for further process. Then using the tables and the customized genetic algorithm our system creates routines for individuals. Inside the G.A. we do crossover, mutation, fitness and etc. as explained earlier, and by doing so we achieved routine for an individual. After that we merge all the routines generated for individual. Finally, in the merge routine we allocate the rooms to the classes, and then create one final routine which is a semester routine. At the end, the system gives an excel output of the merged routine.

By using the intelligence routine management system it is easy to create a routine automatically which is perfect, effective and efficient routine and less time consuming. In our IRM system we have some limitation. We have used a sample set of inputs, our input formats retrieved from the database are defined and this system work for one department. In future, we would like to implementation it for all the departments including the examination routine of all courses and we would like to design the interface for the system.

References

---

1. A. Rahman, S. Giasuddin & R. Rahman. “Decision Tree Based Routine Generation (DRG) Algorithm: A Data Mining Advancement to Generate Academic Routine and Exam-time Tabling for Open Credit System.” Academy Publisher. *Journal of Computers, Vol 5, No 1 (2010), 12-22,* Jan 2010.

2. *G. D.* Smet. *Course Scheduling with OptaPlanner*. 2013.

3. “Genetic Algorithms.” *Surprise 96 Journal, vol 1, article1*. 1996.

4. J. Herrmann. & C. Lee. “Solving a Class Scheduling Problem with a Genetic Algorithm.” *ORSA Journal on Computing. Vol.7, No 4*, Fall 1995.

5. M. Fromhez. *Constraint-based Scheduling.* Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA, 2001.

6. M. Jankovic. *Making a Class Schedule Using a Genetic Algorithm*. January 22, 2008.

7. O. Castrillon. (2014). “A New software for the university class schedules based on evolutionary algorithms and cognitive rhythms.” *International Journal on Advances in Education Research*. ISSN: 2340-2504.

8. S. K. Jha. Exam Timetabling Problem Using Genetic Algorithm. *International Journal of Research in Engineering and Technology*. 2014.

9. S. Sayad . *An Introduction to Data Mining*. 2010.

10. *The Back Propagation Algorithm.* Robert Gordon University Aberdeen. Scotland, UK. 2015.

11. T. M. Mitchell. *Lecture slides for textbook Machine Learning.* McGraw Hill. 1997.