ANALYSIS AND VISUALIZATION OF TIME-VARYING DATA

USING THE CONCEPT OF 'ACTIVITY MODELING'

by

Salil R Akerkar

A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL & COMPUTER ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

2004

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation form or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed us of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED_____

APPROVAL BY THESIS DIRECTORS

This thesis has been approved on the date shown below:

_____ _____

Dr Bernard Zeigler Date

Professor of Electrical and Computer Engineering

ACKNOWLEDGEMENT

I would like to thank my parents for the belief and confidence they had me. Their happiness was a major driving force for me to excel in my educational career.

I would like to thank my sister for always setting challenging (and realistic) goals for me throughout my schooling days. I always followed her career path, and I am happy to do so.

I would like to thank my advisor, Dr Zeigler for his immense support throughout the course of my Masters. It was a pleasure working under him and I truly have learned a lot under his tutelage.

I would like to thank Dr. Hariri and Dr. Nutaro for being in my thesis defense panel.

I would like to thank Hans-Berhard Broeker and Cristina Siegerist for their help with my thesis and Glenn Jorden for his wonderful mentoring during my Co-Op tour at AMD.

I would like to thank everyone at ACIMS lab for creating a wonderful environment during my research assistantship over here. Thanks to Xiaolin Hu and Alex for providing data for my thesis.

Lastly, thanks to all my friends here at Tucson, for making my stay a memorable one. A special thanks to Harish Krishnamoorthy and Harinarayan Venkatramani for being the best roommates ever.

TABLE OF CONTENTS

LIST OF CHARTS AND FIGURES

LIST OF TABLES

ABSTRACT

Scientific visualization, which transforms raw data into vivid 2D or 3D images and movies, has been recognized as an effective way to understand the large-scale datasets. However, most existing visualization methods do not scale well with growing data size. At present there are a number of techniques to analyze data belonging to a particular time-step, but not much research has been made into analysis of data with respect to correlation between time-steps. An attempt is made in this thesis to develop a system, extending the concept of *Activity* in DEVS, based on a simplified theory of finding the active regions in a cellular space of a *spatio-temporal* process. The process of analyzing and visualizing the time-varying data using the system is called *Activity Modeling*. Monitoring of activity would aid in analyzing the process with respect to it's computationally efficiency, dynamically allocating resources to the then-active regions. Analysis of data using *Activity Modeling* gives a different perspective of the process under consideration and focuses only on the active regions in the cellular domain. An overall analysis of the process is presented, in the form of images and movies, of various results computed in the cellular and temporal domain The '*Activity modeling*' system, apart from detecting active regions in the time-varying datasets, also computes statistical results and introduces a concept to predict a *possible pattern* based on the temporal correlation extracted from the data analyzed during the present time step.

1 INTRODUCTION

1.1 CURRENT TECHNIQUES IN VISUALIZATION

Several disciplines, including medicine, computational fluid dynamics, molecular dynamics and oceanography need to analyze large time-varying data-sets using effective visualization and data analysis methods. However, visual exploration of the data is a complicated task because often data produced is large, remote, multi-dimensional and featureless.

There are organizations like ITR [1] which focus on improving the interactivity and explorability of large-scale, time-varying data visualization through the study and development of innovative data reduction methods. ICASE and NASA LaRC [2] have come up with novel methods to visualize time-varying data and organize frequent symposiums to tackle the problem of analyzing large time-varying data. There are papers published [3] to develop an end-to-end low-cost solution for visualizing time-varying data on a parallel computer located at a remote site, in essence making use of large storage space and parallelization techniques to increase the processing power.

Current techniques are more focused on scalability issues and interactive techniques of visualization like volume rendering (ray-tracing algorithms for volume data) and Iso-surfaces. At present the amount of data from scientific and engineering simulations is so large that overwhelming percentage of it is never inspected. There are a few efforts being actually made on doing the analysis of data at the right locations. One such effort [4] deals with developing a client/server model for visual analysis of iso-surfaces and cross-sections in time-varying data and to set a control plane based on it.

The control plane would guide the user with possible starting locations for inspection of data.

It is also found that most of the present techniques are focused in visualization of a single time-step at a time giving more importance to the spatial correlation in the data. There is some research [5] going on in progress to de-correlate the data into a range of spatial and temporal levels of details using a concept called as Wavelet-Based Time-Space Partitioning tree for large-scale time-varying data.

1.2 RESEARCH MOTIVATION

At present there a number of simulations in the scientific and engineering community to produce time-varying data which needs to be visualized in an efficient way. Many applications producing data actually are interested in visualizing the change going on in a particular cell in the temporal domain. It would be possible to find this change, if we have a minimum of at least two time-steps of data values at any given time.

The concept of *Activity* was introduced in DEVS [6] to compute the number of threshold crossings for a cell. It computes the activity of the cell for a pre-defined quantum by using the correlation found in data, in the temporal domain. If the concept of *Activity* is extended to visualization of time-varying data to generate the change in data values for successive time-steps, it would be possible to visualize the activity of the process.

Visualizing activity for a process would help in tracing the region, which needs to be brought in focus for the analysis of large time-varying data-sets. It is also possible for the system to detect the shift in activity found in the cellular domain, which would be of

immense help for resource allocating algorithms.

Presently most of the techniques concentrate on visualizing data in only the cellular domain (one image frame consists of one time-step). The concept of *Activity Modeling* would make it possible to visualize temporal information for an image frame in a cellular domain.

Based on this concept, we can not only generate the activity based information but make use of it to find out other interesting results related to the pattern of activity in a process, the shift of activity found with respect to time and the behavior of the process with respect to its activity.

1.3 THESIS OUTLINE

Chapter 2 introduces the term '*Activity Modelin*g' with respect to various disciplines in today's world. It overviews the '*Activity'* concept in DEVS and finally describes how '*Activity Modeling*' would be used in visualization.

Chapter 3 provides an overview of the system (*Activity Modeler*). It explains in brief the three stages of implementation generic to all the models. The first stage: *Pre-visualization processing* is described in the same chapter.

The concept of '*The Activity Engine*' is introduced in Chapter 4, which forms the second stage of our system. Various concepts related to the visualization of the Activity information are introduced here.

Chapter 5 dives into the Visualization modules implemented in the third (final) stage of our system. Visualization modules developed in GNUPLOT and AVS-Express, used to generate images and movies, are explained here.

Chapter 6 presents observations found from a variety of time-varying data obtained from scientific simulations. It introduces a way to analyze the process from a different perspective (*Activity Domain*).

Automation and Optimization methods used are described briefly in Chapter 7. Benchmarking of the modules with respect to CPU time and memory usage is also covered in this chapter.

Chapter 8 concludes the thesis discussing the future work, present limitations and suggests enhancements, which could be made to the present *Activity Modeler* system.

2 ACTIVITY MODELING

2.1 ACTIVITY MODELING: RELATED CONCEPTS

'*Activity Modeling*' in essence is a very generic term, which can be applied to a variety of topics in different disciplines. Before applying the concept of Activity Modeling in the field of visualization, I would like to discuss a few fields where the concept has already been deeply rooted.

Most of the early work in modeling activity comes from the field of Artificial Intelligence [7]. Many uncertainty reasoning models have been actively pursued in AI and image understanding literature, including Belief networks [9] and Dempster-Shafer Theory.

There have been few '*Activity Modeling*' algorithms in the field of Computer vision for applications such as video surveillance. The basis depends on generating data using various types of networks and formulating algorithms to recognize events. There has been an interesting research [10] in Activity Modeling and Recognition in this field using Shape Theory.

The '*Interaction Lab*' [11] in University of Southern California is dedicated to research in modeling human and robot group activity ranging from one-to-one interaction to small groups to large crowd. Motion capture mechanisms including vision-based and laser-based methods are used to develop robot-tracking system, which is used to gather activity data over long periods. These data are used to derive interaction features and patterns for '*Activity Modeling*'.

'*Activity modeling*' is a term very common in the '*Work-Flow management*

systems' [12] and other Business related topics. However, in this case the term activity is more related to an '*organizational unit performing a specific function*'.

2.2 ACTIVITY: A DEVS CONCEPT

The DEVS specification introduces a way to model systems as mathematical objects using DEVS formalisms. It is used for discrete event modeling and simulation and has considerable advantages over the conventional discrete time (DTSS) approach.

DEVS introduces a concept called Activity [13]. A cell is said to more active than another cell if its value crosses the quantum greater number to times than the other one.

A mathematical definition of Activity over a continuous segment follows:

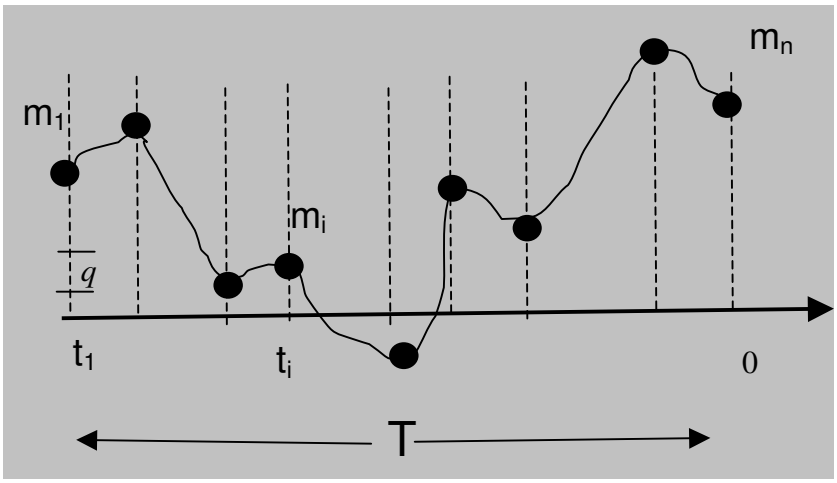


Figure 1: Definition of Activity

$$ActivityPattern(T) = (m_1, t_1)...(m_i, t_i)...(m_n, t_n) \quad \ldots(i)$$

$$Activity(T) = \sum | m_{i+1} - m_i | \quad \ldots(ii)$$

$$AvgActivity(T) = Activity / T \quad \ldots(iii)$$

$$NumberOfThresholdCross(T,q) = Activity(T)/q \quad \ldots(iv)$$

$$AverageDerivative(t_i, t_{i+1}) = | m_{i+1} - m_i | / t_{i+1} - t_i \quad \ldots(v)$$

From the above equations we can conclude the following:

- Activity at any given time step cannot be negative.
- Activity during a given time interval is directly proportional to the number of threshold crossings in that interval.

A quantum (q) is defined to provide quantization supported by DEVS simulator, which is used to detect the change in output (event). In other words, quantization is a method that allows DEVS simulator to track the activity of the cells. From equation (iv), we find that the quantum size is inversely proportional to the number of threshold crossings. It has been proved that to achieve computational efficiency, the number of threshold crossings should be roughly equal to the number of transitions in DEVS.

2.3 VISUALIZATION WITH ACTIVITY MODELING

The concept of 'Activity Modeling' introduced here is with reference to visualization of time-varying data. Two terms related to Activity; *Instantaneous Activity* and the *Accumulated Activity* are introduced here. Both the terms are defined with respect to a particular cell spanning the range of temporal domain.

The *Instantaneous Activity* of a cell for a particular time-step is defined as the absolute difference between values of that cell for successive time steps and is illustrated in the equation below.

$$\textit{Instantaneous Activity (t) (IA)} = \left|Value(t) - Value(t-1)\right|$$

The *Accumulated Activity* of a cell for a particular time-step is defined as the sum of *Instantaneous Activities* of that cell for that particular time-step as well as for all the preceding time-steps.

$$\textit{Accumulated Activity (T) (AA)} = \sum_{t}^{T} |\, Value(t) - Value(t-1) \,|$$

The data is initially said to be present in *Value domain*. When the data is represented in the form of Activity related information, it is said to be present in *Activity domain.* The data can be processed either for a particular cell at a time (*Cellular domain*) or for a particular time-step (*Temporal domain*).

The Activity concept is based on certain guidelines which all the processes normally abide to. The most common of such guidelines are:

- Spatial coherency: If a cell is active, there is a high probability that its neighboring cells are active. Similarly, if a cell is passive, there is a high probability that its neighboring cells are passive.
- Temporal Coherency: If a cell is active for some time-step, there is a high probability that it would remain active for the next time-step.

3 IMPLEMENTATION

3.1 AN OVERVIEW OF THE 'ACTIVITY-MODELING' SYSTEM

The '*Activity-Modeling*' system can be considered as a black-box, which takes raw time-varying data, generated from some process, as an input and provides us with a detailed analysis of the process, with regards its '*activity*'. The results generated are visualized using the visualization modules, which are part of the system. An effort has been made to automate the whole process so that the system would run without human intervention.

The system consists of three stages, which are illustrated in Figure 2. The first stage, which is described later in this chapter, has a primary purpose of checking the integrity of the data and converting the raw-data to a format known to its successive stages. It consists of a formatter written in PERL, which also provides results related to the structure of the data-file used as input in the next stage.

The second stage comprises the '*Activity Engine'* generates the activity related information and analyzes the data-pattern of the process. This stage is implemented with C++ modules and also provides optimization flags and command-line arguments, which help in the automation of the process as described in Chapter 7.

The third stage is dedicated to visualization of the results using pre-built visualization modules in GNUPLOT and AVS-Express.

The system is embedded with benchmarking modules to trace the computation time for various modules and also has a pre-defined directory structure and a parent '*Activity_modeler*' script which provides automation of the system via an easy user

interface. At present the system supports time-varying data which is 1D or 2D in the spatial domain.

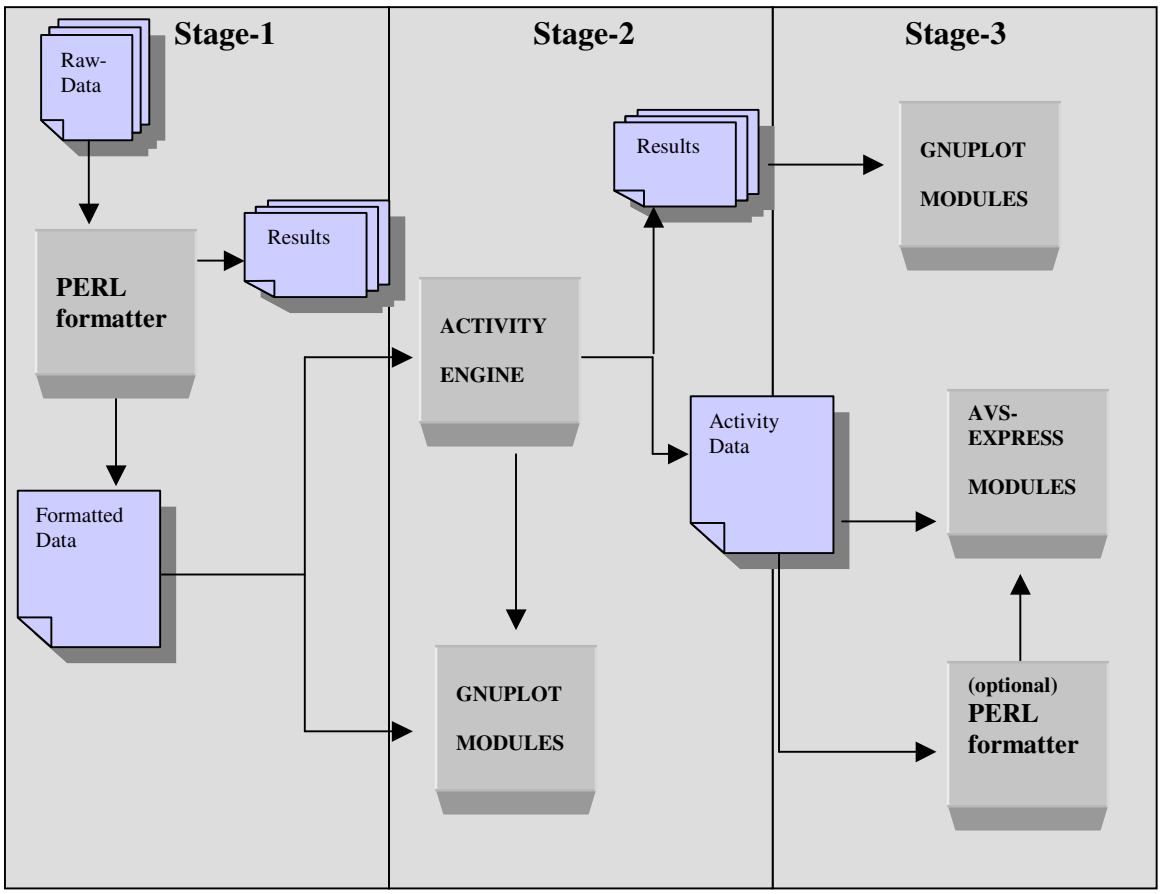


Figure 2: '*Activity-Modeling*' System

3.2 PRE-VISUALIZATION PROCESSING BY PERL MODULES

The raw-data provided at the input of the system can take various formats and needs to be processed for using in our system. It may be corrupted during its transfer from some remote-location. The user may not be provided with any information related to the data file. Due to the large variety of formats the raw-data can be present in, there are few limitations present for our PERL formatter.

- Data-files should be present in ASCII format.

- The cells should be of a regular structured format, which is described in 3.2.1.
- The comments should be present only before the start of data.
- In case of a 2D space, data for different time steps should be present in separate files with (file) naming convention of the form '*<fnamebase>#timeStep.fnameEnd*'.

This stage has mainly three functions. Firstly, it provides information regarding the number of cells and time-steps to the *Activity Engine* module. Secondly, it formats the data in desired format (required by the Activity Engine), for 1D and 2D space (cellular). Third, it provides an error-checking module, which automatically provides *correction logic* and corrects the data file incase it has been corrupted.

The PERL modules also perform auxiliary functions like extracting only a specific part of the data file or generating a transpose of the file.

3.2.1 REGULAR STRUCTURED FORMAT

For the sake of standardization, the data is required to be in a particular format as described below. This format is called the '*Regular Structured format*', following the naming conventions of AVS-Express. It is characterized by the following features:

- Every value corresponds to a particular cell
- Every cell is same in size, is quad-shaped; has eight neighbors (4 on sides + 4 on diagonals) for 2D and two neighbors for 1D.
- Connectivity information is implicit in the data and should not be explicitly mentioned.

With the above features as a base, there are some significant variations in the way data is presented for 1D and 2D space.

For 1D space, all the data is specified in one single file, with rows representing the cells and columns representing the time-steps as shown in **Figure 3**. The format was designed for providing efficiency in computational speed considering the fact the C++ processes arrays in a row-major order and the activity information is present in time-steps.

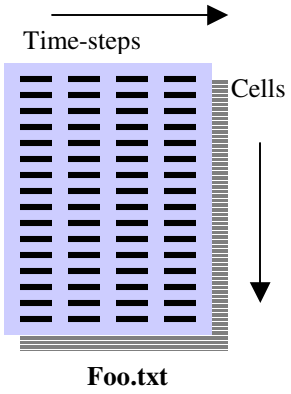


Figure 3: Regular structured format for 1D data

For 2D space, normally a data for each time-step is present in a separate file. Since the Activity Engine requires data for successive time-steps, the PERL formatter concatenates the data from every file to a single file with the data present in the *Regular structured format*. This format essentially helps the *Activity Engine* to handle the trade-off between CPU utilization and memory usage wisely. For larger time-steps it also has the advantage of decreasing the computation time by eliminating the time spent in IO operations for files. The disadvantage would be that this method would impose restrictions to visualization for a large number of time-steps, however this disadvantage is taken care of by the visualization scripts and post processing of resultant data using PERL modules. The *concatenation* and *expansion* operation for 2D data, illustrated in **Figure 4**, is carried out the *pre-processing* and *post-processing* modules in PERL.

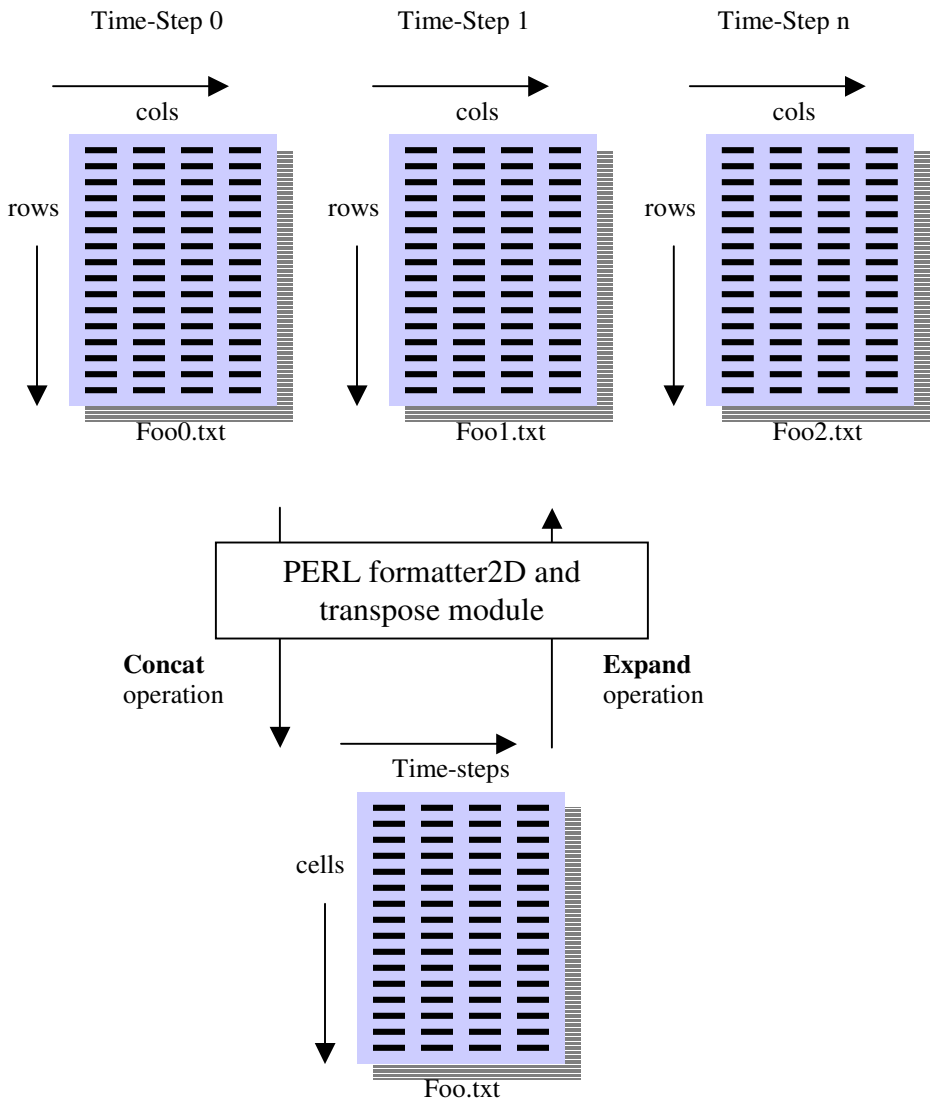


Figure 4: Operation of 2D formatter

3.2.2 CORRECTION LOGIC (PRE-VISUALIZATION)

The Correction logic provides an error-checking algorithm described below. It is built on few assumptions about the structure of the data file and can be changed when data files are of a different structure than our assumptions.

By default, we assume that the data file abides by the rule that the rows of our data files represent the cells and columns the increasing time-steps, as described in 3.2.1. The correction-logic algorithm is as follows:

Algorithm 3.1:

Step1:
Compute the number of lines (*nCells*)
Compute the number of values (*nValues*)
If(*nValues%nCells* is 0)
Correction logic not required, *nTSteps = nValues / nCells*
Else
Correction logic required, goto Step2

Step2: (*Correction Logic*)
Generate a 1D diagnostic array:
For(*cell:= 0-nCells*)
diagArr[cell] = number of time-steps found for that cell
Generate a frequency table (*freq_table(time-steps)*) which calculates the number of occurrences of the total number of time-steps.
Maximum(*freq_table*) := *maxAt*
For (each line in data)
If(number of Elements eq *maxAt*)
leave the row unmodified
Else if((number of Elements < *maxAt*)
Replicate the last time-step value for the missing values
Else
Truncate the values till *maxAt*.

The PERL scripts to implement the above logic are *formatter.pl* (1D) and *formatter2D.pl* (2D) and have various command line options to make automation of the process easier as discussed in Chapter 7. Extensive use of PERL's pattern-matching, regular expressions and array processing has been made in the formatter to increase the computational speed.

3.2.3 DATA GENERATION USING PERL SCRIPTS

A PERL script called *dataGenerator.pl* is used to generate 1D and 2D data files, which were used as inputs to the Activity Engine during the training stage of the system. They produce test data generated by mathematical equations and have great flexibility in terms of the nature of data to be produced.

The script generates data based on,

1. Dimensions of data (1D, 2D. 3D)
2. data-type to output – float OR integer
3. nature of data to output – random OR user-defined function *f(x,t)* and *f(x,y,t)*

The significance of this script was to generate known patterns of data and verify if the *Activity Engine* produced desirable results during the training stage of development of our system.

4 THE ACTIVITY ENGINE

4.1 THE ACTIVITY ENGINE- AN OVERVIEW

The Activity Engine, written in C++, is the heart of the visualization system. It is used primarily to generate the activity related information from the data-file, which is later on visualized using modules in GNUPLOT and AVS-Express. It is also used to extract statistical information from the data and analyze the activity-pattern for the whole process.

A functional block diagram of the *Activity Engine* is shown in Figure 5 with regards to its interaction with the other modules in the system.

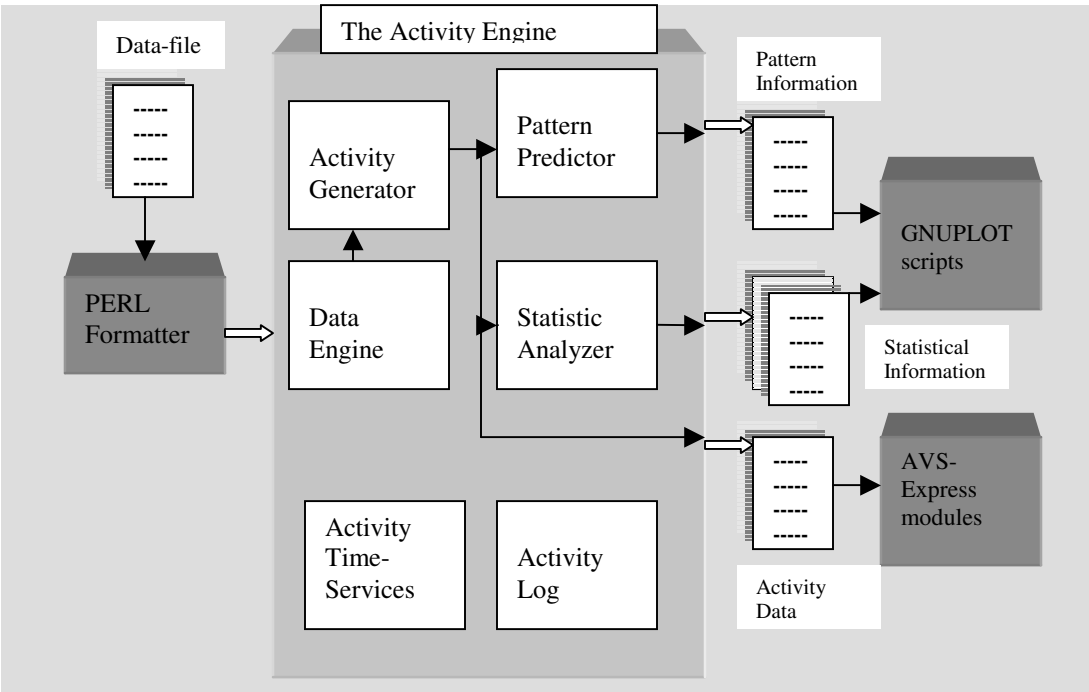


Figure 5: *The Activity Engine*

The Activity Engine was first conceived with the purpose of generating *Instantaneous Activity* and *Accumulated Activity*. Later on two additional modules were incorporated for a thorough analysis of the data in the *Activity Domain.*

As seen in Figure 5, the *Data Engine* module forms the input interface to the *Activity Engine*. It passes the extracted data to the *Activity Generator* module, which switches the mode of operation from the *Value domain* to the *Activity domain*. The *Statistic Analyzer* and *Pattern Predictor* modules, then, act in the *Activity domain* to produce a number of results in the *Cellular and temporal* domain.

4.2 THE DATA ENGINE

The *Data Engine* forms the interface to the *Activity Engine* and deals with file handling, memory allocation, de-allocation and data-processing for the other modules in the *Activity Engine*. Input data is read by this module either sequentially or via random access logic (reading a particular row out-of-sequence). Apart from acting as a reader it also performs the following functions:

- Generating transpose to switch between domains (*Temporal* and *Cellular*).
- Transformation between 2D and 1D arrays for analysis of 2D cellular data.
- Tracking the size of files in bytes to keep track of memory usage.

4.3 THE ACTIVITY GENERATOR

Activity Generator is the primary module in the system, used to switch the mode of operation from *Value domain* to *Activity domain.* In *Activity domain* it is used to

calculate the *Instantaneous Activity* and the *Accumulated Activity* of the cells. Based on the *Instantaneous Activities* in the process, it is also used to calculate the *Time Advances* for a cell by finding the reciprocal of the *Instantaneous Activities.* Apart from that it is also used to find out the *Activity Factor*.

4.3.1 ACTIVITY FACTOR

Activity Factor is defined as the fraction of total time-steps the *Instantaneous Activity (IA)* remains greater than a particular threshold set by the user. The threshold is generally a fraction of the maximum IA values of the process. *Activity Factor* is used to find the process behavior in the *Cellular domain*. In other words, it denotes where the activity is concentrated in the *Cellular domain*.

$$Activity\ Factor\ (AF) = \frac{nTimeSteps(IA(t) > threshold)}{TotalTimeSteps}$$

In the above equation, the *threshold* value can also be used to eliminate insignificant amounts of activity values in the process generated due to errors in simulations, which generate the data. A meaningful threshold value, normally set oblivious to the user, is the global average of *Instantaneous Activity* taken over both the *Temporal* and *Cellular* domain.

$$Threshold\ (AF) = \sum_{t}\sum_{x} IA(x,t) \Big/ NT$$

4.3.2 SIGNED INSTANTANEOUS ACTIVITY

The concept of *signed Instantaneous Activity (sIA)* is introduced here, which violates the definition of *Activity* in some extent.

$$signed\ Instantaneous\ Activity\ (sIA) = Value[t + \Delta t] - Value[t]$$

As seen above the sign is preserved. *Signed Instantaneous Activity* is used to generate find information related to Peaks in the *temporal domain* using a simple algorithm illustrated below.

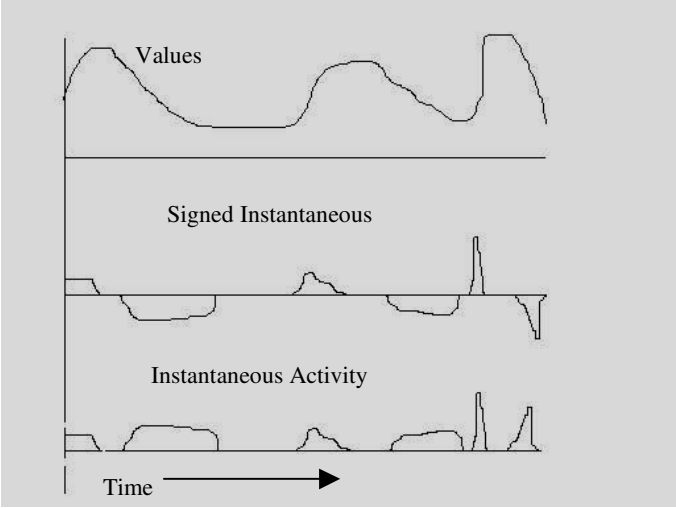


Figure 6: *signed Instantaneous Activity*

Algorithm 4.1:

```
if(   (signed_activity[i-1] >=0)  && (signed_activity[i] < 0)     )
        peaks++; // Increase the peak count
else if(   (signed_activity[i-1] <=0)  && (signed_activity[i] > 0)     )
        bases++; // Increase the base count
```

The concept of *signed Instantaneous Activity* is used to simplify the problem of finding the peaks. As seen from the figure it is evident that whenever the activity curve

cuts the time axis, depending on the direction in which the curve cuts the axis, there is a corresponding peak or base in the output values.

4.4 THE STATISTIC ANALYZER

The Statistic Analyzer module, as the name suggests is used to extract various statistics from the data-file. This module is expandable and at present supports three specific groups of statistics namely:

1. Maximum, Minimum, Range
2. Average, Standard deviation, Mean, Geometric mean, harmonic.
3. Living Factor
4. Histogram of Time Advances

A flag present in the command-line options of the *Activity Engine,* called the *fast* flag, is used to increase the computational speed. One way to achieve the increase in speed is by disabling the group of statistics present in the *Statistic Analyzer* module, which tend to be compute-intensive for floating-point values.

The *Statistic Analyzer* acts in the *Activity domain* processing values for each cell at a time (*Cellular domain*) or each time-step at a time (*Temporal domain*), depending on which group of information it is generating. Statistics Analyzer is also used to provide the visualization system with a set of known values, like the global and local average of IA curves, which are provided as inputs to other modules in the system. For example the *Living Factor* is normally provided by a local average of IA values while the *Activity*

Factor is provided by a global average of activity values.

4.4.1 LIVING FACTOR

Living Factor is defined as the fraction of total cells, the *Instantaneous Activity (IA)* remains greater than a particular threshold for a particular time-step. The threshold is generally a fraction of the maximum IA values of the process. *Living Factor* is similar to *Activity factor*, only that the two act in different domains.

$$LivingFactor\ (LF) = \frac{nCells(IA(t) > threshold)}{TotalCells}$$

The *Living factor* is used in the *temporal domain* and the gives results for a particular time-step. In simple words, as the name suggests, it signifies the fraction of cells living for a particular time-step.

In the above equation, a *threshold* value has been introduced to eliminate the effect of possible noise in the data (*insignificant activities*). The process can be termed as *flattening the IA curve*, considering that applying threshold would remove (flatten) the insignificant maxima and minima in IA curve. A meaningful threshold would be the local average of IA values taken over the cellular domain.

4.4.2 HISTOGRAM OF TIME ADVANCES

The *Statistical Analyzer* module also computes the histogram of *time advances (Tavd)* for the process to give an overall picture of the speed of the process, with respect to the time taken to produce the next event (Tadv). The user can give the number of

histogram steps at the input and the module will compute the distribution of cells over the *temporal domain.* Since various simulations can take a very large range of *time advances* the histogram is plotted over the logarithmic scale. *Time advances* are calculated from the IA curve as they are nothing but the reciprocal of *instantaneous activities*. Algorithm 4.1 computes the histogram as shown below

Algorithm 4.2: (*Histogram for tadv*)

```
For(cell:=0; cell<Cells; cell++)
        Tadv[cell] = 1/IA[cell]
Initialize an array for Histogram (hist[STEP_SIZE]) to all zeros
MaxLog := log10(Max(Tadv)/Min(Tadv))
MinLog := log10(Min(Tadv)/Min(Tadv)) = 1
If(Tadv[cell] == 0)
        hist[STEP_SIZE-1]++
Else
        hist[int (log10 (Tadv[cell]/Min) ) ]++
```

The formula in the above algorithm enables us to visualize a very high range of *time advances* of the order of $Max(tadv)/Min(tadv) \approx 10^9$

A visual representation of the working of the Statistic Analyzer is shown in Figure 7, with the help of the domain concept.

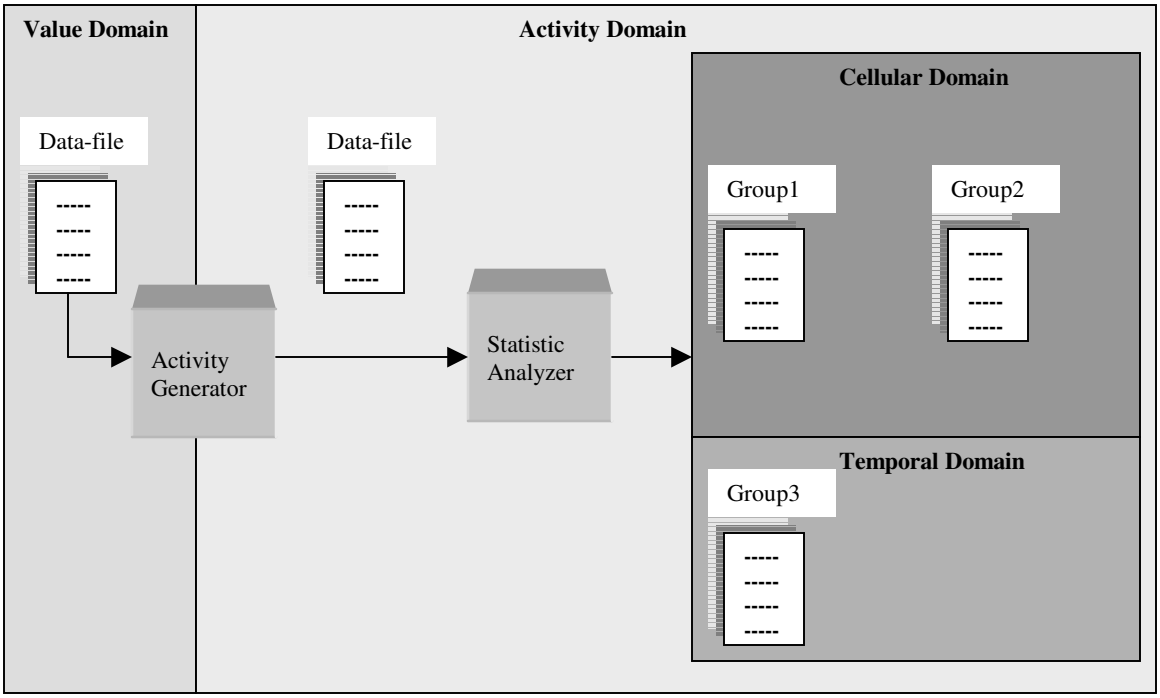


Figure 7: The Statistic Analyzer

The results generated for each group are present in a particular format, which are fed along with other related information to the visualization scripts to visualize the results in the form of images and movies.

4.5 THE PATTERN PREDICTOR

The Pattern Predictor module works only in the t*emporal domain* and is most intelligent of all the modules in the system. It analyzes the activity of the process using the phenomenon of spatial coherence and makes and attempt to predict futuristic behavior of the process depending on the phenomenon of temporal coherence.

Pattern predictor introduces two new concepts called as *trace locator* and *Region of Imminence (ROI),* which are used to monitor the activity for the present time-step and

give predictive results regarding, what the activity at the next the time-step, would be in the future. It also introduces a novel method to find out the Activity related information using peaks, applied in the *Value domain*. The *Pattern predictor* module in essence is not meant to generate data for visualization, but to predict futuristic behavior of the process and analyze the present behavior in the *temporal domain.*

4.5.1 PEAK-BASE METHOD

The conventional method to find Accumulated Activity is to add the Instantaneous Activity for successive time-steps as show below:

$$Accumulated\ Activity\ (T) = \sum_{1}^{T} | m_i - m_{i-1} |$$

As seen clearly, the operation would require 2T operations. Since accumulated activity is normally required only for the final time-step, there can be a more efficient way to find out the accumulated activity if we are presented with some additional information related to the Instantaneous Activity (IA) curve. This additional information should be provided in form of peaks and bases present in the IA curve.

As seen in Figure 8, a significant savings in computations can be achieved by the presence of the Peak-Base information of the process.

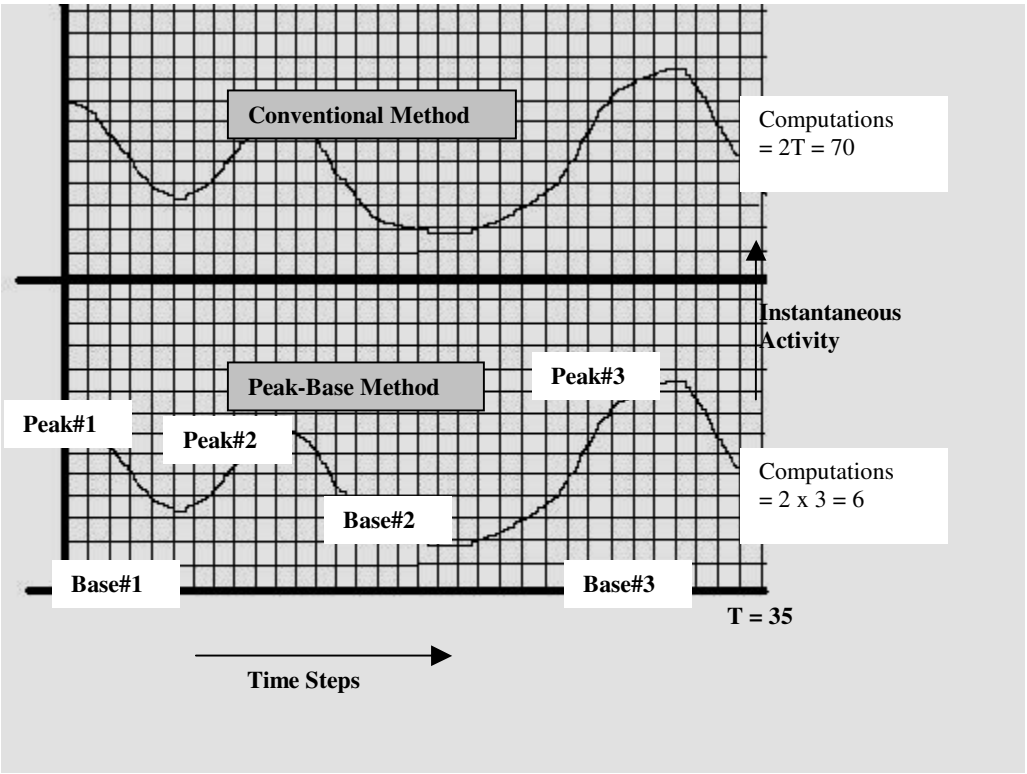


Figure 8: Peak-Base Method to find Accumulated Activity

In Figure 8, we see that there are three pairs of Peak and Base in the *Instantaneous Activity* curve. The *Accumulated Activity* can be calculated by the adding the difference in the successive Peaks and Bases as shown by the formula below

$$Accumulated\ Activity\ (T) = \sum_i (|\ Peak_i - Base_i\ | + |\ Peak_{i+1} - Base_i\ |)$$

The case for boundary condition is not shown in the above formula. For example the formula denoted the computations going on for the intermediate Peaks and Bases, while it has no idea whether the IA curve starts or ends in a Peak or Base.

Another constraint to the above formula is that the Peaks and Bases should always alternate (NPeaks = NBases Or NBases +/- 1). The presence of false peaks would cause an error as seen in **Figure 9**, since there is expected to be a Base in between Peak#2 and

Peak#3.

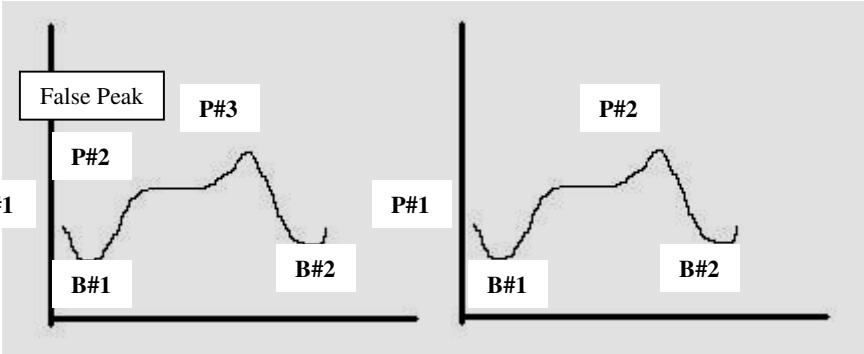


Figure 9: False-Peak problem

The false peak problem can be eliminated by slightly modifying the algorithm to find the Peaks. However, in the actual implementation of the Peak-Base formula, both the problems mentioned above are taken care off by using a single array consisting of the Peak-Base Values, which takes care of the boundary conditions as well as the false Peak problem.

One thing to note is that the information provided by Peaks and Bases causes some overhead in terms of memory usage. In processes where the IA curve is not characterized by frequent fluctuations, the *Peak-Base method* would not be recommended to calculate the *Accumulated Activity*. However, if the fluctuations in the IA curve are known to be caused by noise, the concept of threshold can be used to flatten the IA curve as described above.

4.5.2 REGION OF IMMINENCE (ROI)

Region of Imminence is the region of cells defined in the temporal domain, which

is predicted to be active for the following time-step, considering normal process behavior. In other words, ROI consists of the cell numbers whose *Instantaneous Activity* is predicted to be greater than the threshold set for the following time-step.

ROI is mainly used to analyze process behavior and requires information regarding the Peak locations and Values.

Figure 10 illustrates the concept of ROI and how it is related to the Peak information of an IA curve in the temporal domain. Any process that attains a state of equilibrium is defined to be normal. For such a process, the Instantaneous Activity should approach zero as the process ends. This would mean that the ROI should comprise of all the cells at $t = T$ for even a single peak present.

Since the ROI information is extracted from data in the *Cellular domain* and *Activity domain*, it makes use of both; spatial and temporal coherency described in Chapter 2.

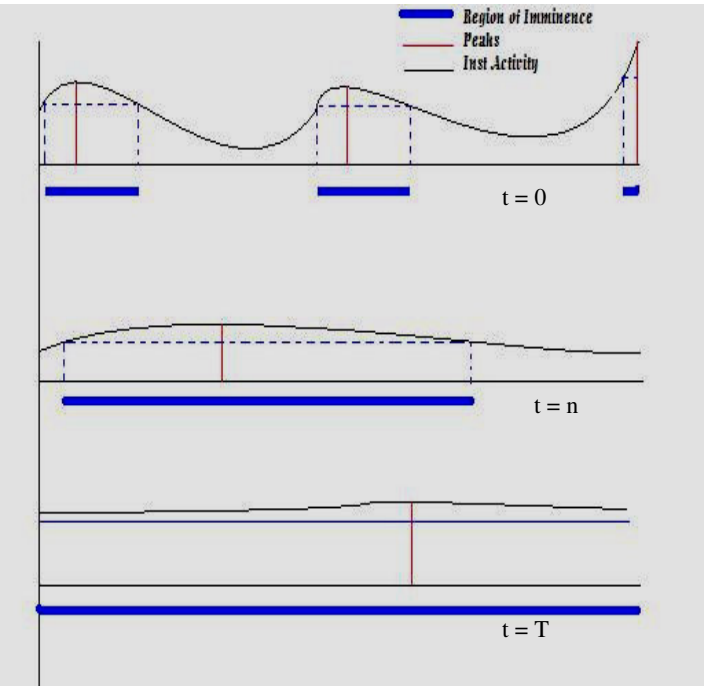


Figure 10: Region of Imminence

The algorithm to find the ROI for a 2D process would be:

Algorithm 4.3: (*ROI*)

Compute Peak Values and locations as described in Equations 4.1from IA curve
For(peak := 1 – Number of Peaks)
    Threshold = (*e*PeakValue[peak]*)
    leftNeighbor = rightNeighbor = PeakLocation[peak] ..For 1D
    left = right = Peaks2D[peak].y, up = down = Peaks2D[peak].x ..For 2D
Compute the neighboring locations based on the threshold conditions and the boundary conditions described in scanning algorithm4.4
For(*cell := leftNeighbor - rightNeighbor*) ..For 1D
    ROI[cell] = 1
For (*row:=left - right*) ..For 2D
    For (*col := up - down*)
        If(*IA[row][col] > Threshold*)
            ROI2D[row][col] = 1

The above algorithm requires a way to find the Peaks and to scan the IA curve (1D) or the IA surface (2D) to find out the ROI. Finding the peaks is described below in 4.5.4, while scanning the IA curve (and IA surface) would be described now.

For 1D space, finding the ROI from the IA curve requires scanning the curve only in 1 dimension since every cell has only 2 neighbors (and the ones at corners have 1). The procedure of scanning the IA curve is described using the **Figure 11**, which uses the following algorithm. Since every peak in the IA surface needs to be scanned for imminent regions, this algorithm is compute-intensive and is more optimized for CPU time then memory.

Algorithm 4.3 (a)

For 1D:
Scan to the left of Peak and compute *leftNeighbor* based on
    Boundary condition (*leftNeighbor* > 0) and
    Threshold condition (*IA[leftNeighbor] >= Threshold*)

&& (*IA[leftNeighbor-1] < Threshold*)
Scan to the right of Peak and compute *rightNeighbor* based on
Boundary condition (*rightNeighbor < nCells-1*) and
threshold condition (*IA[rightNeighbor] >= Threshold*)
&& (*IA[rightNeighbor-1] < Threshold*.

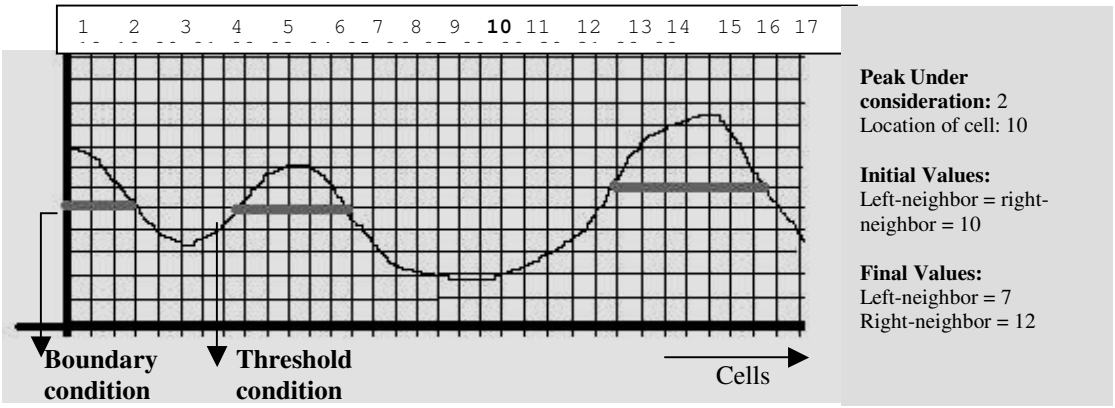


Figure 11: Scanning IA curve – to find ROI in 1D

For 2D space, scanning the ROI surface uses a similar approach, however, since the space is 2D, the number of neighbors present are 4 (if you consider only the edges) and additional 4 (considering the neighbors diagonally at the corners). Also the boundary conditions need to be taken care of at the four edges of the bounding box defining the 2D space. The algorithm used to find the ROI is on the similar lines to that for 1D:

Algorithm 4.3 (b)

For2D:
While(*IA[row][left] > threshold*) and (*left*>0)
Decrement *left*
While(*IA[row][right] > threshold*) and (*right<Col*)
Increment *right*
While(*IA[up][col] > threshold*) and (*up>0*)
Decrement *up*
While(*IA[down][col] > threshold*) and (*down<Row-1*)

```
        Increment down
While( (IA[row-upleft][col-upleft] > threshold) and
        ((row-upleft)>0) and ((col-upleft)>0))
        ROI2D[row-upleft][col-upleft] = 1; upleft++;
While((Values2D[row+downleft][col-downleft] > threshold) and
        ((col-downleft)>0) and ((row+downleft)<Rows-1))
        ROI2D[row+downleft][col-downleft] = 1; downleft++;
While((Values2D[row-upright][col+upright] > threshold) and
        ((row-upright)>0) and ((col+upright)<Cols-1))
        ROI2D[row-upright][col+upright] = 1;upright++;
While((Values2D[row+downright][col+downright] > threshold)
        and ((row+downright)<rows-1) and ((col+downright)<Cols-1))
        ROI2D[row+downright][col+downright] = 1;downright++;
```

Figure 12 shows the scanning algorithm for IA surface. The cells colored in black are the Peak values at that particular time-step. The cells colored in dark-grey are the cells covered in the ROI and those colored light-grey are the cells not traced by the ROI scanning algorithm, however theoretically speaking they are in the ROI. Lastly, the uncolored cells are the also the cells not belonging to the ROI, however the should not be mistaken to be inactive.

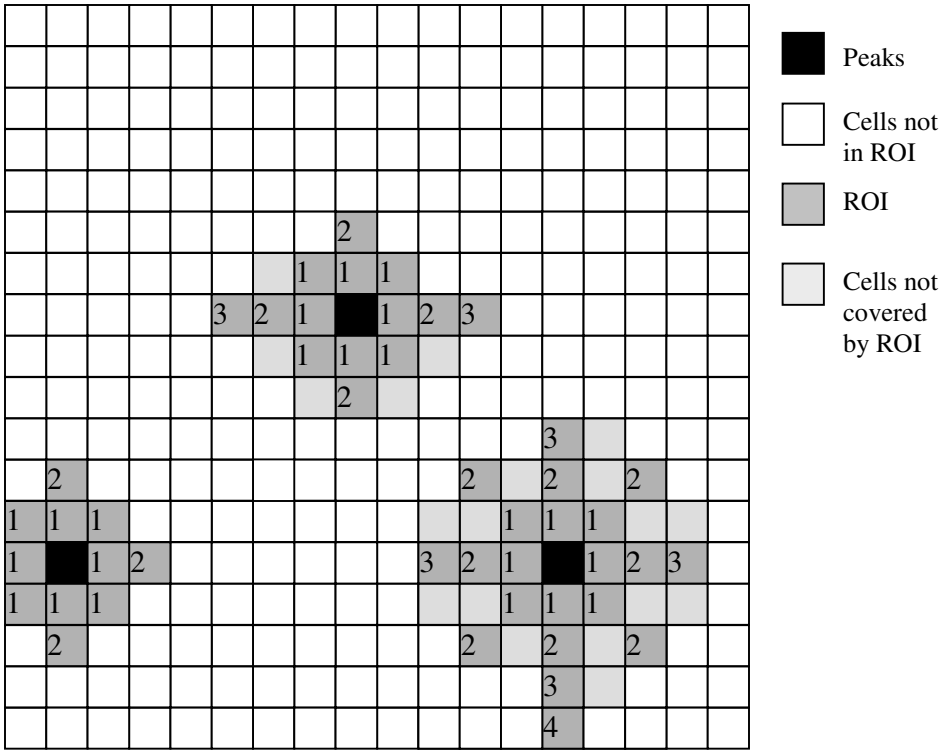


Figure 12: Scanning IA surface – to find ROI in 2D

The Step 4 in ROI algorithm can be modified to encompass the cells colored in light-grey too, however that would be at the cost of the simplicity of the present algorithm.

Finding the ROI is normally a compute-intensive process in case a significant number of peaks present. There are some modifications made to the above scanning algorithm for a 2D spatial data to speed up the algorithm depending on few assumptions. There are in all four levels of scanning, based on how the algorithm is implemented. Based on those algorithms there are four levels of tuning in ROI:

i. Coarse tuning: Only the horizontal and vertical neighbors are scanned.

ii. Normal tuning: In addition to the horizontal and vertical even the diagonal neighbors are scanned. Described in Figure 12.

iii. Fine tuning: After normal tuning, the region with in the horizontal and vertical limits is scanned to cover the light-grey cells shown in Figure 12. This algorithm depends on the assumption that most of the cells not covered in normal tuning are within the limits of the horizontal and vertical ROI limits.

iv. Finest tuning: This is the most comprehensive algorithm of all and depends on a recursive method to consider every imminent neighbor as a peak. However, this turns out to be highly inefficient for data with large number of cells and produces results slightly better than the previous algorithm.

A visual comparison of the first three scanning algorithms for a classical 2D diffusion problem (time step -15) is given for in Figure 13 as well as a comparison with respect to the Imminent Factor and computation speed is given in Table 1.

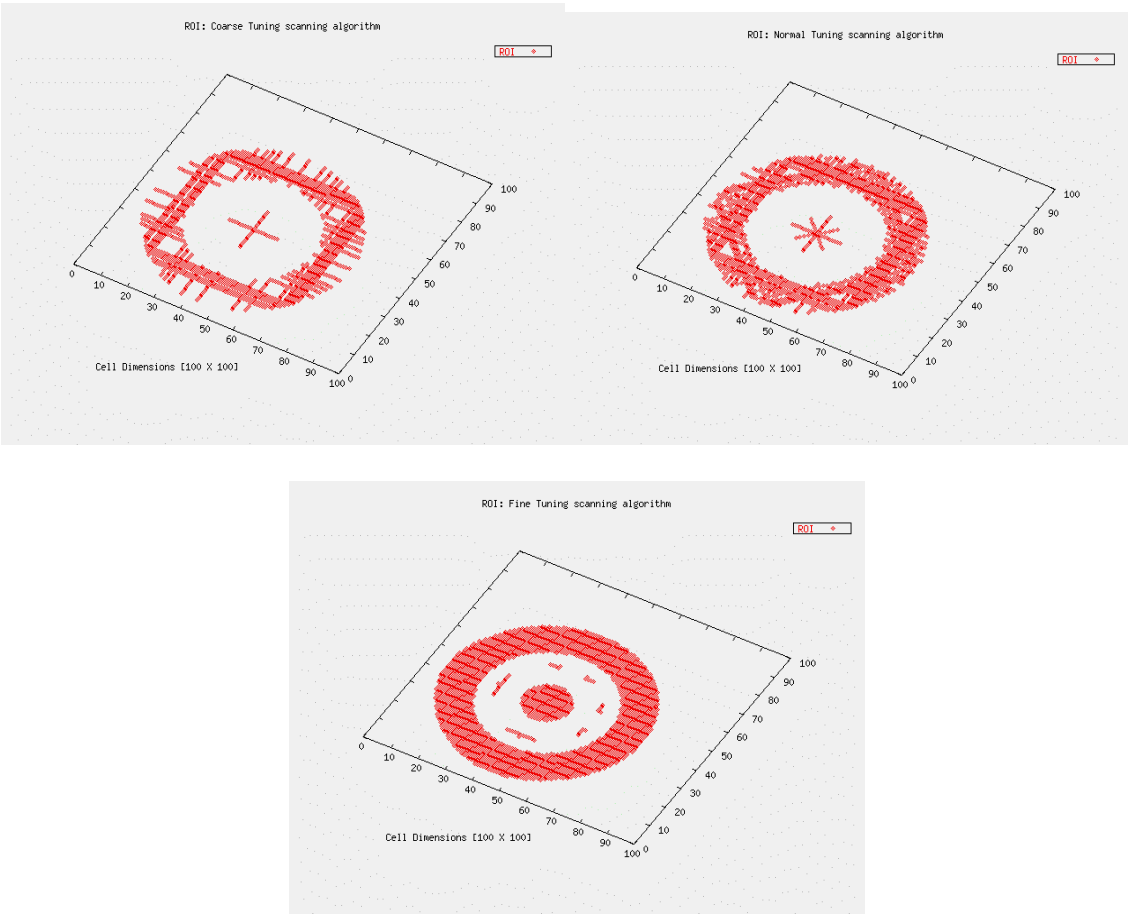


Figure 13: Visual Comparison of scanning algorithms for 2D diffusion problem

Scanning Algorithm	Computation Time (ms)	Imminence Factor (t=5)	Imminence Factor (t=5)	Imminence Factor (t=5)
Coarse	5393	0.070900	0.143200	0.097400
Normal	6224	0.098500	0.189600	0.139500
Fine	6456	0.150500	0.254700	0.332700

Table 1: Computation time and Imminence Factor for scanning algorithms

ROI not only serves as a predictive characteristic of the process but also provides information regarding the region where bulk of Activity is going on for a particular time-step. This information would be of use in resource allocation algorithms where the system needs to find the active regions for a particular time-step.

4.5.3 TRACE LOCATOR

Trace Locator is an algorithm used to trace the locations of the cells having the same values over the period of time. In many applications it may be useful to trace the locations of the maximum value over a period of time-step, to see where the activity of a process is concentrated.

We can get the similar information monitoring the Peak Values. However, it may be possible that the peak values do not represent the actual concentration of activity when there is a significant difference in the Peak Values for a particular time-step. Figure 14 shows an example when *Trace Locator* for maximum values can be more useful to represent the activity of the process (as compared to a Peak locator)

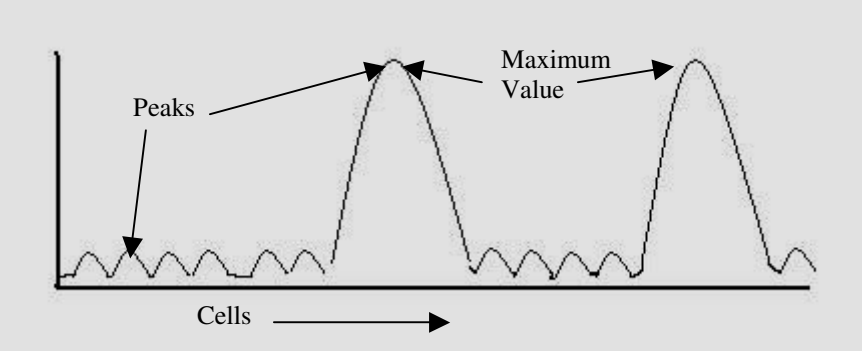


Figure 14: Difference between Trace Locator and Peak Locator

As illustrated in Figure 14, if the values of peaks are differing significantly or if the peaks with smaller values are appearing significantly more times, spread out over the whole range of space, it would be more sensible to track only the values greater than a particular threshold set in the *Trace Locator* modules (for e.g. 0.9 x maximum values).

It is mainly used to find the most active regions during a particular time-step. It can be used for a number of applications, which require knowledge regarding the shift of activity and to characterize the whole process on the basis of activity.

4.5.4 PEAK LOCATOR

The Pattern Predictor module computes information required for various modules like ROI, based on the peaks found in the IA curve (or surface). It is one of most basic functionality provided by the Pattern Predictor module. The Peak Locator as the name suggests stores the values and locations of the peaks in dynamically formed arrays. This requires two runs through the values, one for finding the number of peaks, and the other to store the Peak Values and locations. For 1D space, the algorithm to find peaks should make two comparisons since every cell has 2 neighbors, while in 2D the algorithm should

make eight comparisons since every cell has 8 neighbors. The cells present at the boundaries should be treated differently with lesser comparisons being made as shown in the equations below.

Equations 4.1:

For 1D peak is present if

i. $(Val[cell] > Val[cell-1]) \;\&\; \&(Val[cell] \geq Val[cell+1])$ … for internal cells

ii. $Val[0] > Val[1]$ … boundary condition 1

iii. $Val[Cells-1] > Val[Cells-1]$ … boundary condition 2

For 2D peak is present if

i. $(Val2D[row][col] \geq Val2D[row][col+1]) \;\&\; \&(Val2D[row][col] > Val2D[row][col-1]) \;\&\; \&$
$(Val2D[row][col] > Val2D[row-1][col]) \;\&\; \&(Val2D[row][col] > Val2D[row-1][col-1]) \;\&\; \&$
$(Val2D[row][col] > Val2D[row-1][col+1]) \;\&\; \&(Val2D[row][col] \geq Val2D[row+][col]) \;\&\; \&$
$(Val2D[row][col] > Val2D[row+1][col-1]) \;\&\; \&(Val2D[row][col] > Val2D[row+1][col+1])$
… for internal cells

ii. $(Val2D[0][0] > Val2D[0][1]) \;\&\; \&(Val2D[0][0] > Val2D[1][0]) \;\&\; \&(Val2D[0][0] > Val2D[1][1])$
… boundary conditions (corner)

iii. $(Val2D[0][col] > Val2D[0][col-1]) \;\&\; \&(Val2D[0][col] > Val2D[0][col+1]) \;\&\; \&$
$(Val2D[0][col] > Val2D[1][col-1]) \;\&\; \&(Val2D[0][col] > Val2D[1][col]) \;\&\; \&$
$(Val2D[0][col] > Val2D[1][col+1])$
… boundary condition (edge)

The Peak Locator algorithm takes care of the *false peak* problem mentioned above. One common problem faced in tracing peaks is detection of a region of cells, which all have some maximum values and are located adjacent to each other. In such case a center cell will have all its eight neighbors with the same values and the equalities in the above equation will not hold true. In this case, we ignore the peak from the peak locator module, however it is found out by the ROI module during another run of comparison through the cells. The *false peak* problem is illustrated in Figure 16 below for a 1D heat diffusion problem. Since the IA curve for our 1D diffusion problem, was found

to be characterized with a very small slope in the *cellular domain*, it was hard to detect the true peaks values using Equation 4.1 (i-a)

$$(Val[cell] > Val[cell-1]) \,\&\, \&(Val[cell] \geq Val[cell+1]) \qquad \ldots\text{Equation 4.1 (i)-a}$$

The above equation detected peaks, biased to the cells with larger locations as shown in Figure 15 (a). If we change the above equation with respect to its inequalities as

$$(Val[cell] \geq Val[cell-1]) \,\&\, \&(Val[cell] > Val[cell+1]) \qquad \ldots\text{Equation 4.1 (i)-b}$$

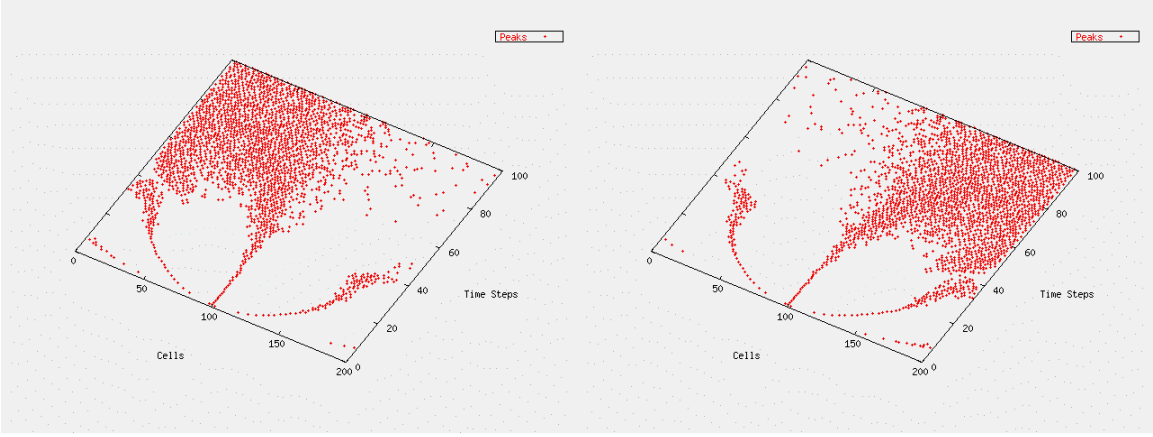
, we find that it detects false peaks biased to the cells with smaller locations as shown in Figure 15 (b). If the above equation is changed to

$$(Val[cell] > Val[cell-1]) \,\&\, \&(Val[cell] > Val[cell+1]) \qquad \ldots\text{Equation 4.1 (i)-c}$$

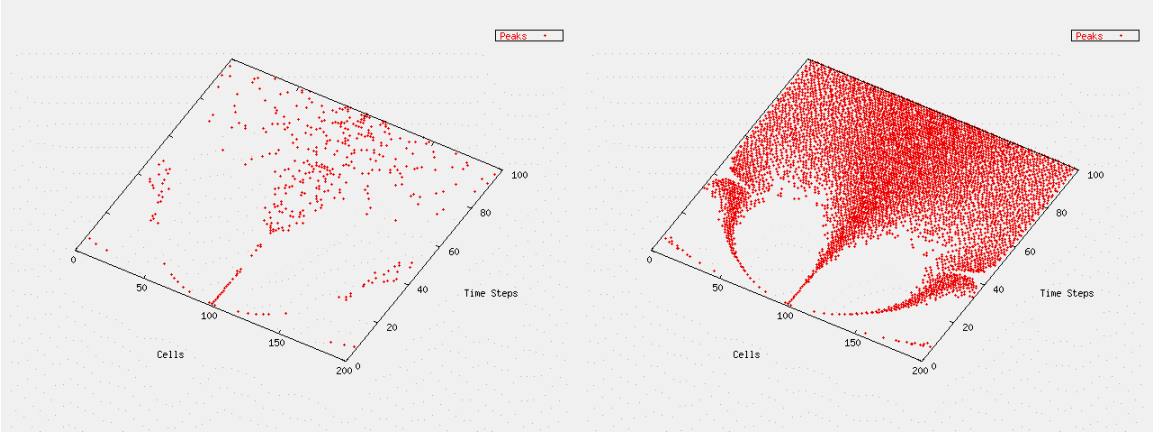
, we find that detection algorithm becomes very conservative and some of the true peaks are also missed out as seen in Figure 15 (c), while equation

$$(Val[cell] \geq Val[cell-1]) \,\&\, \&(Val[cell] \geq Val[cell+1]) \qquad \ldots\text{Equation 4.1 (i)-d}$$

, detects the *false peaks* for the full range of *cellular domain* as seen in Figure 15 (d).



(a) (b)



(c) (d)

Figure 15: False Peak problem for 1D diffusion problem

The following algorithm takes care of the problem illustrated above and gives a true representation of the ROI as seen in the Figure 16

Algorithm 4.4
Step 1: Find the *ROI-a* using Algorithm 4.3 and Equation 4.1 (i)-a
Step 2: Find the *ROI-b* using Algorithm 4.3 and Equation 4.1 (i)-b
Step 3: *ROI*: = *ROI-a && ROI-b* (over the cellular domain)

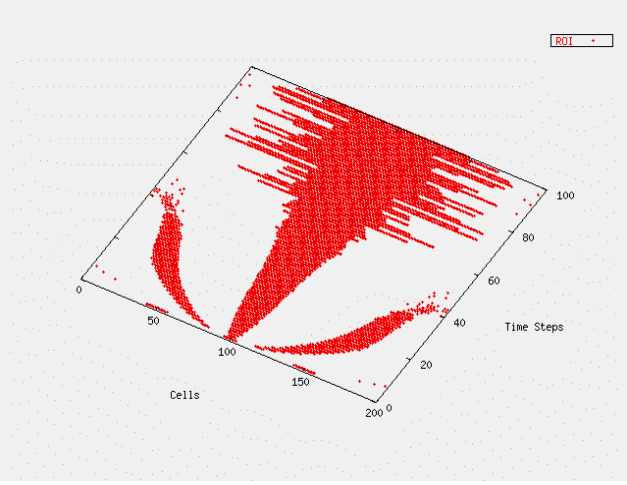


Figure 16: Approximated ROI using Algorithm 4.4

4.5.5 PREDICTING THE PATTERN FOR 1D SPACE

The *Pattern Predictor* module also has the logic to predict the activity pattern of the process under consideration for a future time-step on the basis of the activity pattern at the present time-step. The theory of operation is based on extracting possible patterns in activity from two successive time-steps at an earlier time and cross-checking the activity pattern with an activity pattern in the future.

The activity pattern fed to the input of this module is generated by the ROI module with an epsilon (e) value in the range [0.9-0.95]. The reason why the e is set at the higher end is to discard the unwanted peaks which tend to predict false patterns. Since the data present in ROI module is in the form of 0s and 1s, it is preprocessed by a *Linear Span* module which detects the region of 1s in the data and saves them in the form of (*start Cell, length of region*) as illustrated in algorithm 4.4.

Algorithm 4.5: (*Linear Span*)

```
startCell = 0, cell = 0, Intialize Lspan[cell]
While(cell < nCells)
        If (ROI[cell])
                startCell := cell
                while(ROI[cell] and (cell < nCells) )
                        Increment cell
                        Lspan[startCell]++
        Else
                Increment cell
```

During the first step of the algorithm, the output of the *Linear Span* modules for *t=t0 and t=t1* are processed using the pattern predictor algorithm and the detected patterns are saved in a data structure. In the second step of the algorithm, a possible pattern is extracted for the time *t=tn (for n- [t2-T])*. In the third step of the algorithm a

sanity check is made by comparing the active region predicted by the module and the actual active region computed by the ROI at *t=t2*. If the actual ROI is significantly different than the predicted ROI, it is re-computed for the next sample (*t=t1 and t=t2*). The process of predicting the temporal pattern can be illustrated with the algorithm.

Algorithm 4.6: (*Predict Pattern*)

Step1:
ta = t1, tb = t2, set ROI (e=0.9)
Compute *ROI(ta)* and *ROI(tb)* as illustrated in algorithm 4.2
Compute *LinearSpan(ta)* and *LinearSpan(tb)* as shown in algorithm 4.4
Step2:
Start the prediction process by detecting patterns from *LinearSpan(ta)* and *LinearSpan(tb)* to produce *predictedROI(t)* as shown in equation 4.2.
Step3:
Compare the *ROI(tb+1)* and the *predictedROI(tb+1)*.
If(results are satisfactory)
End
Else
ta = ta+1, tb = tb+1, check boundary conditions and goto Step1

The prediction process consists of two steps. First step consists of scanning for predefined pattern present in the repository of patterns and the second step consists of computing the ROI for the later time-steps. The pattern found is stored in a data structure characterized by the attributes such as its direction, offset in location and type of pattern (increasing or decreasing). *First order patterns* are defined to be the ones in which either of the neighboring cells at *t=tb* posses a non-zero value and may contribute to a pre-defined pattern. *Second order pattern*s are defined to be those which have all its neighboring cells with zero values but has some high non-zero value at t=tb which may contribute to a pre-defined pattern from our repository. The concept of the order in patterns in illustrated in Figure 17 below.

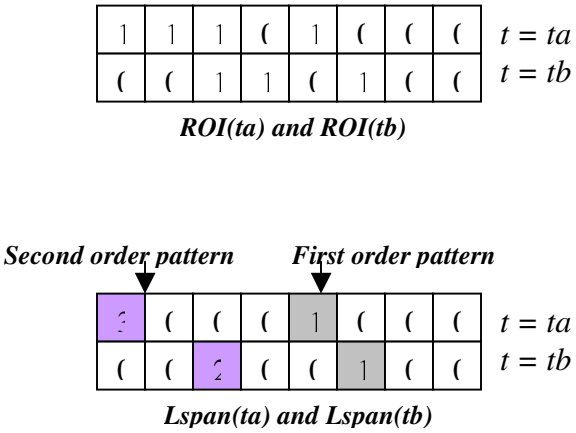


Figure 17: Concept of *order* in the *Predict Pattern* module

Based on the order, direction and type of patterns there are 5 first order patterns and 2 second order patterns. The attributes of a pattern are explained with the help of the examples in Figure 18.

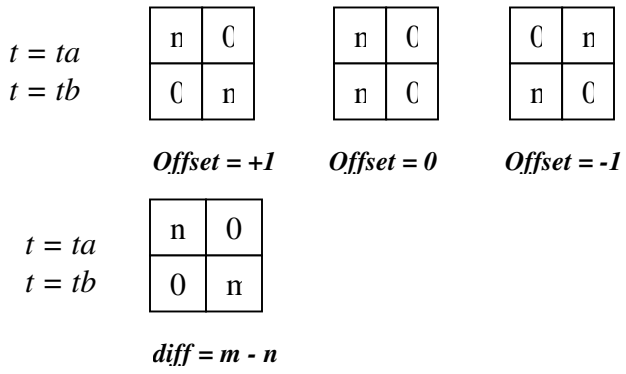


Figure 18: Attributes of the *Pattern* data-structure

Table 2 and Table 3 provide a detailed description of the pre-defined possible pattern in the repository and its classification on the basis of order, type and direction of

the pattern. Note that the data structure in which the pattern is stored also has information about the location (*x0*) and the activity value at the cell (*m0*), however since these values are independent of the type of pattern detected they are not covered in the tables below.

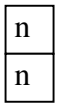
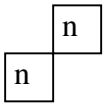
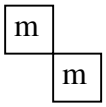
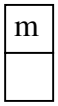
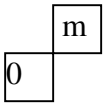
Type	Example	Constraint	Direction (dir)	Offset (off)	Difference (d)
1		no constraints	Down	0	0
2		no constraints	Left	-1	0
3		|m - n| <= 2	Right	1	d = n – m
4		|m-n| <= 2	Right	0	d = m – n
5		|m-n| <= 2	Right-Left	-1	d = m - n

Table 2: Type of First order patterns

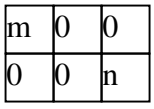
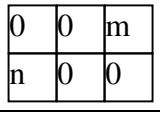
Type	Example	Constraint	Direction (dir)	Offset (off)	Difference (d)
1		|m - n| <= 2	Right	L(n) – L(m)	n – m
2		|m - n| <= 2	Left	L(n) – L(m)	n- m

Table 3: Type of Second order patterns

The second step in the module consists of generating the ROI for a time-step t=tn; it can be generalized for all the patterns depending on attributes such as difference and direction. The algorithm 4.6 describes the process of generating the ROI(t = tn).

Algorithm 4.6:

```
If(difference eq 0)
        If(direction is down or right)
                cell = x0 + off*(tn-t0)
                while( (cell < Cells) and (cell < x0+off*(tn-t0)+m0) )
                        ROI[tn][cell] = 1, Increment cell
        Else If(direction is left)
                cell = x0 + off*(tn-t0) + m0
                while( (cell >= 0) and (cell >  x0 + off*(tn-t0))
                        ROI[tn][cell] = 1, Decrement cell
Else If(difference > 0)
        If(direction is down or right)
                cell = x0 + off*(tn-t0)
                while( (cell < Cells) and (cell < x0+off*(tn-t0)+d*(tn-t0)) )
                        ROI[tn][cell] = 1, Increment cell
        Else If(direction is left)
                cell = x0 + off*(tn-t0) + d*(tn-t0)
                while( (cell >= 0) and (cell >  x0 + off*(tn-t0))
                        ROI[tn][cell] = 1, Decrement cell
        Else If(direction is undefined)
                cell = x0
                while( (cell >= 0) and (cell >  x0 + off*(tn-t0))
                        ROI[tn][cell] = 1, Decrement cell
                cell = x0
                while( (cell < Cells) and (cell <  x0 + off*(tn-t0) + d*(tn-t0))
                        ROI[tn][cell] = 1, Increment cell
Else //If(difference < 0)
        If(direction is down or right)
                cell = x0 + off*(tn-t0)
                while( (cell < Cells) and (cell < x0+off*(tn-t0)+d*(tn-t0) + m0) )
                        ROI[tn][cell] = 1, Increment cell
        Else If(direction is left)
                cell = x0 + off*(tn-t0) + d*(tn-t0) + m0
                while( (cell >= 0) and (cell >  x0 + off*(tn-t0))
                        ROI[tn][cell] = 1, Decrement cell
        Else If(direction is undefined)
                cell = x0 + off*(tn-t0) + m0 + d*(tn-t0)
                while( (cell >= 0) and (cell >  x0 + off*(tn-t0))
                        ROI[tn][cell] = 1, Decrement cell
```

It is found that for processes whose ROI is characterized by a non-linear curve, the predictor module doesn't work well. However, it gives significantly close results for processes with more or less linear ROI curves.

5 VISUALIZATION

Two visualization softwares- GNUPLOT and AVS-Express were used for visualization purpose. Both have their advantages and reasons why they are used during that particular stage. A brief description follows regarding the two softwares and how they were used to visualize the data files.

5.1 GNUPLOT

5.1.1 INTRODUCTION

GNUPLOT is a portable command-line driven interactive data-file (text or binary) and function plotting utility for UNIX, IBM OS/2, MS Windows, DOS, Apple Macintosh, VMS, Atari and many other platforms. The software is copyrighted but freely distributed.

It was originally intended as graphical program which would allow scientists and students to visualize mathematical functions and data. In 2D, it can draw line, point, dot, box, histogram graphs or vector fields. In 3D, it supports line, point and dot surfaces, with or without hidden line removal. It supports color or grayscale surfaces and maps.

The GNUPLOT version 3.7 for 32-bit Windows was used for visualization. GNUPLOT was selected for the preliminary and intermediate visualization of the data files due to the following reasons:

- Simplicity of use and portable over variety of platforms
- Presence of interactive mode and batch mode to provide flexibility of usage

- GNUPLOT scripting which allows automation of the system and customization for a particular data-file.
- GNUPLOT v3.7 onwards has an additional flag to the 'splot' function called the 'matrix' flag, which is used to visualize data represented in a 2D format as a matrix. Incidentally, this is the same format used as input to the Activity Engine module.
- Variety of features can be enabled and disabled by a simple '*set*' commands.
- Lastly, the visualized images can be saved in a variety of different formats.

5.1.2 SPLOT FUNCTION

The *splot* (surface plot) function is used for the preliminary visualization of data-files generated by the *Activity Engine*. *splot* can display a surface as a collection of points, or by connecting those points. The points may be read from a data file (as in our case) or result from evaluation of a function at specified intervals. The surface may be approximated by connecting the points with straight line segments, in which case the surface can be made opaque with set hidden3d. The orientation from which the 3d surface is viewed can be changed with set view.

Additionally, for points in a grid format, splot can interpolate points having common amplitude and can then connect those new points to display contour lines, either directly with straight-line segments or smoothed lines. Functions are already evaluated in a grid format, determined by set isosamples and set samples, while file data must either be in a grid format, as described in data-file, or be used to generate a grid. Contour lines may be displayed either on the surface or projected onto the base. The base projections of the contour lines may be written to a file, and then read with plot, to take advantage of

plot's additional formatting capabilities.

The matrix flag in *splot* function is used to visualize the data, which is represented in the same format used as input to the Activity Engine module. The matrix flag indicates that the ASCII data are stored in matrix format. The z-values are read in a row at a time as shown below

z11 z12 z13 z14 ...

z21 z22 z23 z24 ...

z31 z32 z33 z34 ...

and so forth. The row and column indices are used for the x- and y-values.

5.1.3 GNUPLOT SCRIPTS

Sometimes, several commands are typed to create a particular plot, and it is easy to make a typographical error while entering a command. To stream- line our plotting operations, several GNUPLOT commands may be combined into a single script file.

Consider a case in which we have to change the viewing angles to get a closer look of a particular cell or a specific time-step. This would require a means to rotate the image around the central axis to an angle which would give a more clear perspective of that required portion of the image. A script file can be written for exactly that purpose, which would use the '*set view*' command from GNUPLOT to change the default viewing angle such that all the hidden portions of the image a brought to light.

Figure 19 and Figure 20 show how the hidden portions of the image are made

visible by changing the viewing angles.

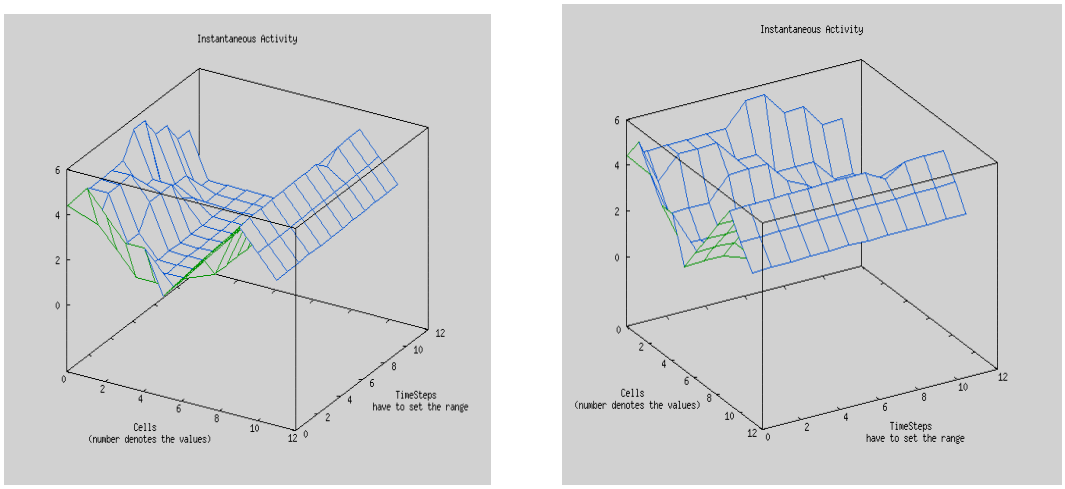


Figure 19: Rotating the vertical angle

Another useful aspect of the writing scripts is automating the visualization system.

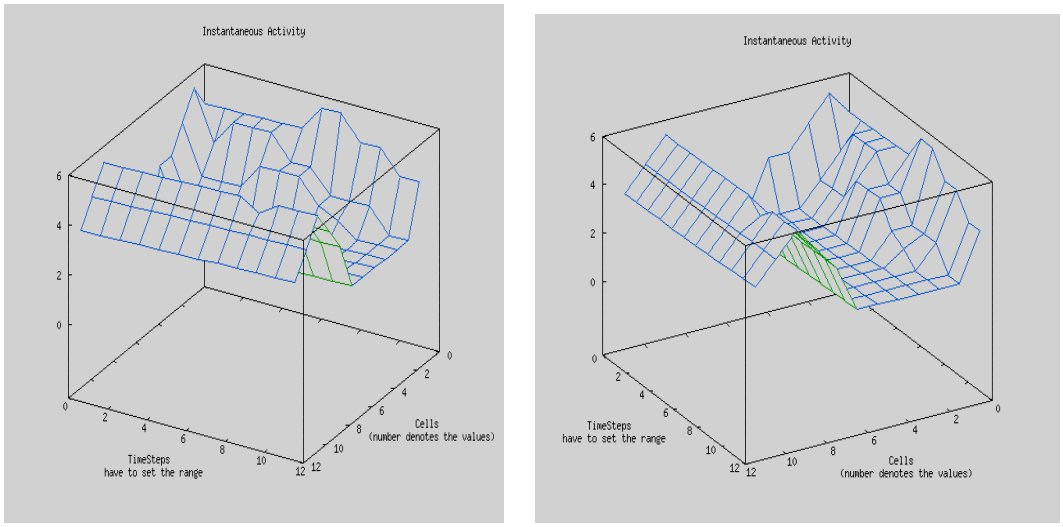


Figure 20: Rotating the horizontal angle

5.1.4 GNUPLOT ON WINDOWS

As mentioned above the *Activity Modeler* system is implemented on the Windows

platform and GNUPLOT was essentially built for the UNIX platform. To enable intelligent scripting in GNUPLOT on Windows, we have to open a pipe to GNUPLOT and stream-line commands to it using some scripting language like PERL. The Windows version of GNUPLOT has a utility program called *pgnuplot*, which supports piping of commands.

Since 2D visualization requires movie-making from image files, there should be an efficient way to generate a number of images from the resultant data files. One of the auxiliary PERL script perform the function of generating images by opening a pipe to the *pgnuplot* program. While generating a large number of images (> 20), the script allows automatically closes the pipe and re-opens it after GNUPLOT finishes the processing to prevent buffer overflow problem.

5.2 AVS-EXPRESS

5.2.1 INTRODUCTION

AVS/Express is an object-oriented, visual development tool that enables you to build reusable objects, application components, and sophisticated data visualization applications. It has the following features:

- Object Oriented - AVS/Express's development approach is object-oriented; it supports the encapsulation of data and methods; class inheritance; templates and instances; object hierarchies; and polymorphism.

- Visual development - The Network Editor is AVS/Express' main interface. It is a visual development environment that is used to connect, define, assemble, and manipulate objects through mouse-driven operations.
- Visualization application - AVS/Express provides a number of predefined application components (objects) that process, display, and manipulates data. The objects and application components that you connect and assemble in the Network Editor control how data is processed and how it is displayed. If you choose, you can compile and package those objects and even add a user interface to create a complete application that can be delivered as a stand-alone application.

AVS-Express has a Network Editor, which is the primary development tool consisting of libraries of objects and a workspace where we can build visualization modules. The underlying code in which objects are written is called the V code, which can be modified using a text editor or the Network editor. AVS-Express is built on component technology, which is defined as the ability to create an application from reusable, modular pieces of components. Lastly, the most important is that it gives you the flexibility to create our own components and make them re-usable modules adding them to the pre-existing libraries. AVS-Express is by far more advanced then GNUPLOT and used for developing movies and fancy visualization images, while GNUPLOT is used for making quick 1D and 2D graphs with limited visualization modules like contour and surface plot.

5.2.2 READERS

AVS-Express supports a generalized data-structure called a *field* which supports many different types of data sets. Almost all of the visualization modules in AVS-Express supports *field* as an input. To gain the capabilities of these AVS-Express visualization modules we have to write *Reader* for our data-sets. This process is called '*Importing the data'*. Apart from an *import* module there are other module called the *readfilename, animfilename and trigger* which comprise the reader for our data.

AVS-Express supports four different types of data based on the grid (connectivity) information. They are summarized in Table 4.

MESH TYPE	**GRID**	**CONNECTIVITY**
Uniform	Supply max/min nodes only	Implicit based on dimension
Rectilinear	Specify axis nodes only	Implicit based on dimension
Structured	Specify all nodes	Implicit based on dimension
Unstructured	Specify all nodes	Specify connectivity using cell-sets

Table 4: Data types supported by AVS-Express

As seen in Table 4 our data falls in the Uniform Category and essentially there is no need to even process it before importing. However, since our data is present in the *matrix* format and Express expects the data to be in a column arrays, an *Import* module is written which formats the data in the desired format and later the AVS-Express *field mappers* are used to output the desired data in the *field* format. Figure 21 shows a 3D data with 2000 values being imported to form a field structure. Note that the *import* module which is the heard of the Reader is not shown in the figure.

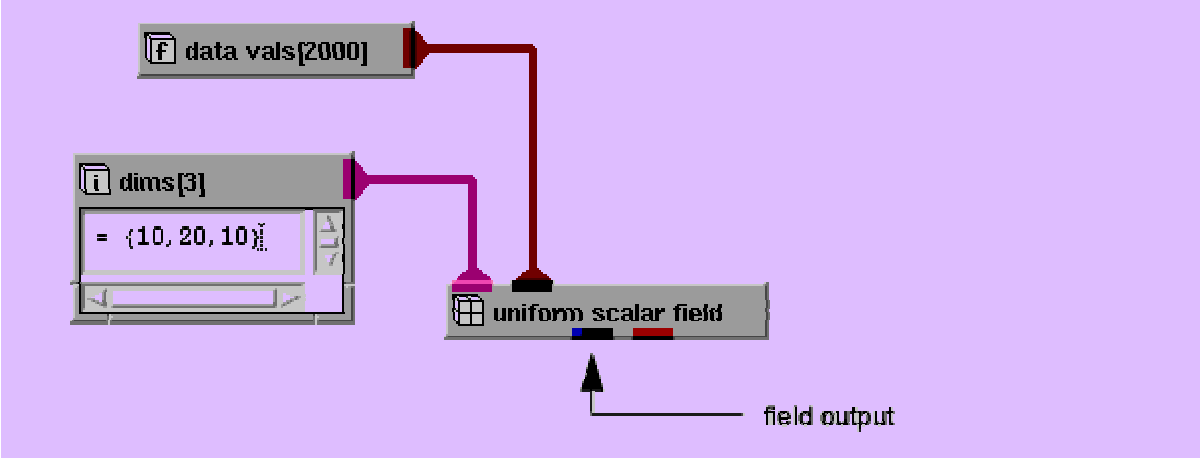


Figure 21: File Import module for AVS-Express

The *readfilename, animfilename* and *trigger* modules are developed to use the *Animator* and *ImageCapture* modules, which contains the movie-making capabilities.

5.2.3 AVS-MODULES

AVS-Express has a set of libraries. The Data Visualization library was used for our data. The Data visualization library has a rich set of modules which allow the user to use some of the most advanced visualization techniques. Some of the techniques used are listed below:

- Contours, Isolines, Isosurfaces: Dealing with your data as a volume. Best way to render a volumetric data making sense to human eye.
- Slices and cross-sections: Examining volume data as a separate slice.
- Colormaps: Attaching meaningful graphical cues to the data-sets
- CitySpaces and Surface plots: Method designed to effectively show histogramic data-sets in 3D.

- Animation: The best way to visualize time-varying data having a spatial dimension greater than two.

There are a number of other modules present, however explaining those is beyond the scope of this thesis.

5.3 VISUALIZATION CONCEPTS

Apart from using the built-in capabilities of the visualization softwares, there are a few things to be added to generate images and develop movies from the resultant data. The most common feature which needs to be developed is a *Reader* for the data to be visualized. The implementation of *Reader* for AVS-Express is described above. GNUPLOT does not require any reader module and reads directly from files, which are already present in the required format (*Activity Engine* generates data-files in those formats to eliminate the overhead in visualization)

Briefly, any visualization system consists of three stages. The first is the *Reader* stage, which reads the file whose data, needs to be visualized. Internally the reader stage is software-specific; some softwares make copies of the data, which others (AVS-Express) create soft-links to the files. The second stage forms the deals with the core visualization process and consists of the built-in modules like Surface Plot, Iso-Surface, volume rendering, arbitrary slicer etc. This stage also consists of the bulk of computation in the visualization process. The third stage is the writing stage; in which the generated results can be stored in the form of images and movies. AVS-Express has some sophisticated modules to make movies in different formats, however GNUPLOT does not have any known method to make movies and they are made in a two-step process;

generating images for every time-step and using some other utility program to generate movies from the image-files (for 3D visualization).

In visualization of time-varying data, the dimension of the data needs to be increased by one, as time itself contributes to one dimension. In short, 1D spatial data is actually 2D in the spatio-temporal domain, while 2D spatial data is 3D in the spatio-temporal domain. There are always two ways to visualize any data, either using movies or using images. A movie consists of frames corresponding to time-steps and is an interactive way to monitor the changes going on per time-step for the variable visualized. Images are static in nature. One can generate an image for a particular time-step or for a range of time-steps where the variable is visualized in the spatio-temporal domain as described above. However, one should note that generating an image consisting of data for all time-steps is only possible for 1D space. In our system, for 2D spatial data, visualization will be supported mainly in the form movies in the cellular domain and images in the temporal domain. For 1D spatial data, visualization will be supported in the form of images for both the domains (cellular and temporal).

5.3.1 VISUALIZATION OF 1D TIME-VARYING DATA

For 1D data, the whole process can be captured by a single image; time forms one axis of the system as shown in Figure 22. In this case, the cellular space is 1D consisting of 50 cells and the process has 50 time-steps. An easy and sensible way to visualize the whole process would be adding a time axis to the graph as shown in the Figure 20. These

types of images are used for results generated from the *Activity Generator* module, which are 2D in nature.

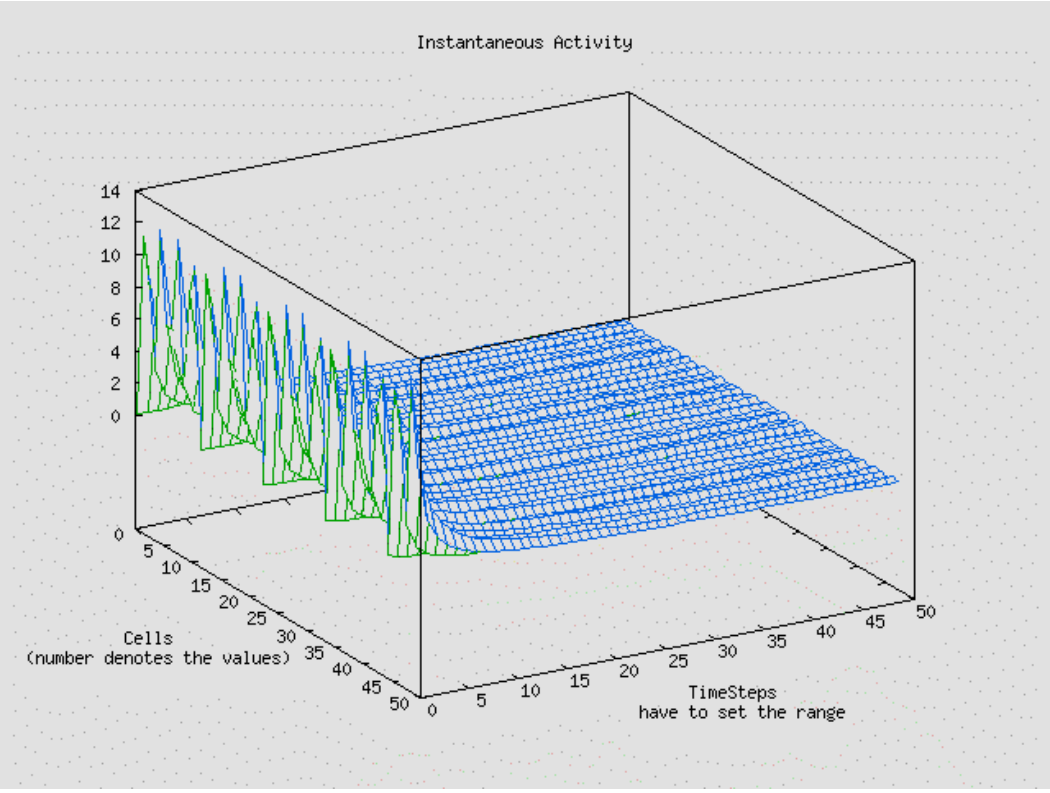


Figure 22: Visualization of 2D spatio-temporal process

For 1D data, results like *Living Factor* and *Activity Factor* reduce a dimension of the (spatio-temporal) data as the results compute a single value for all the cells (*in temporal domain*) and a single value for all the time-steps (*in cellular domain*). Such results are visualized as 1D graph using the *plot* and *multiplot* function from GNUPLOT. Figure 23 shows the *Statistic Analyzer* results for a test data using the *multiplot* command. These are the second type of results, which are in essence 1D in nature and normally generated by *Statistic Analyzer* for 1D data.

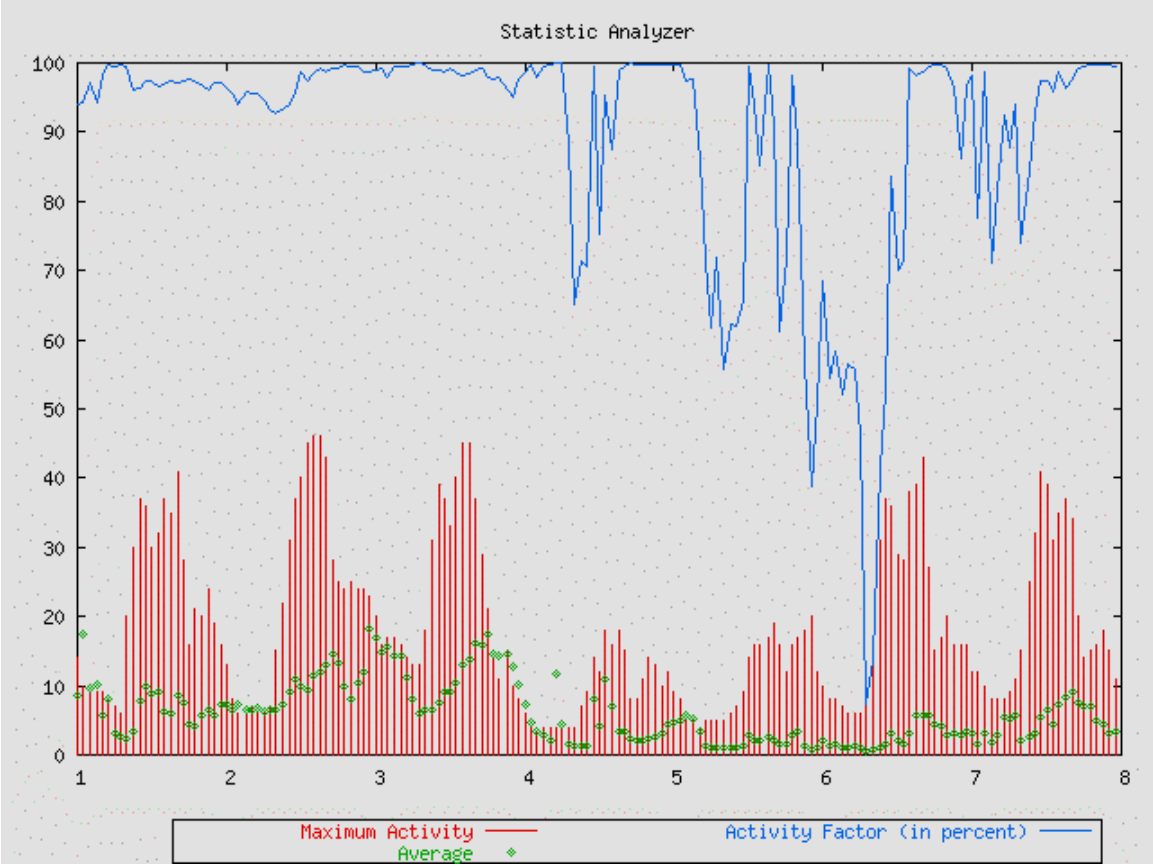


Figure 23: Visualization using multi-plot graphs

The third type of result to be visualized is 2D in nature and spans the whole spatio-temporal domain. These results are generated mainly by the *Pattern Predictor* module and use a technique called *zero padding* and *binary visualization*. Some of the results which fall in this category are that for the *Max-Locator, Peak locator* and *Region of Imminence.*

5.3.2 ZERO PADDING AND BINARY VISUALIZATION

The concept of *zero padding* and *binary visualization* was introduced specifically for two purposes:

- Focusing on the results and eliminating unwanted data.
- Significant reduction of resultant data-files.

Zero padding is essentially replacing the unwanted data with zeros, and retaining the values for the data of interest. Binary visualization is an enhanced form of zero padding which replaces the data of interest by 1s and unwanted data with 0s. However, it should be noted that almost all the visualization softwares visualize zero values by placing a dot at that location. To overcome this unwanted effect, the range of the z-axis can be changed to a value greater than zero and less than the minimum value to be visualized. This can be accomplished by a simple command in GNUPLOT; *set zrange [0.5:],* which sets the minimum value on z axis to 0.5 for visualizing the *ROI*.

Since, the zero can be printed to the file as a *short integer* data-type there is a considerable savings in the file-size, which serves our second purpose. Also eliminating the visualization of unwanted data by changing the range enables us to focus on the region of interest and serves our first purpose. Figure 22 shows the image generated from the data obtained by the *Max-Locator* module. Clearly, the image focuses only on the maxima found in the data, which is an indication of the most active cells in the spatio-temporal domain.

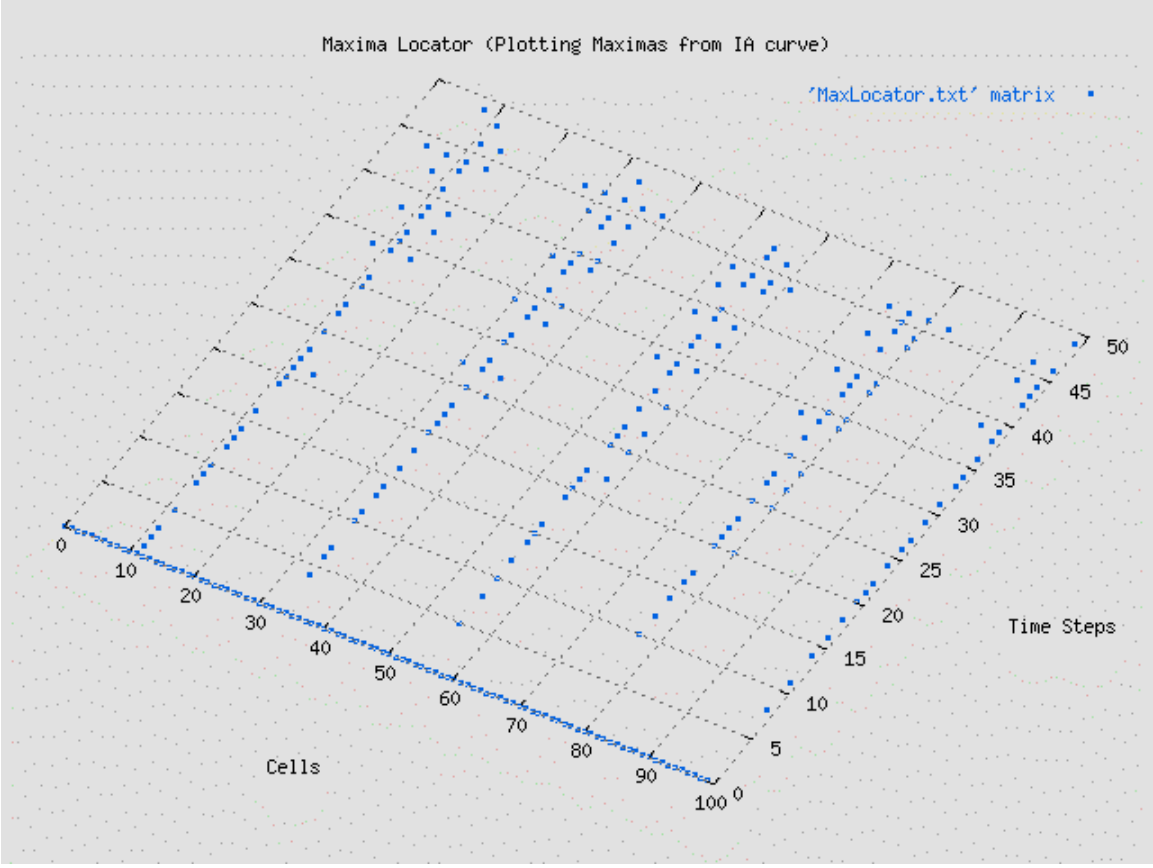


Figure 24: Binary Visualization for Max-Locator module

The three types of resultant data generated by the *Activity Engine* are summarized below in Table 5.

Type of result	**Domain**	**Dim**	**Visualization technique**
Instantaneous Activity	Spatio-temporal	2D	Surface Plot images (GNUplot)
Accumulated Activity	Spatio-temporal	2D	Surface Plot images (GNUplot)
Activity Factor	Temporal	1D	1D single graph (GNUplot)
Statistics	Temporal	1D	1D multi graphs (GNUplot)
Living Factor	Cellular	1D	1D single graph (GNUplot)
Maxima Locator	Temporal	1D	1D multi graph (GNUplot)
Peak Locator	Cellular	1D	1D multi graph (GNUplot)
Region Of Imminence	Spatio-temporal	2D	Binary visualization (GNUplot)
Peak Bars	Spatio-temporal	2D	Zero padding (GNUplot)
Max Bars	Spatio-temporal	2D	Zero Padding (GNUplot)

Table 5: Visualization of results for 1D space

5.3.3 VISUALIZATION OF 2D TIME-VARYING DATA

The types of resultant data generated by the *Activity Engine* from 2D time-varying data are similar to that generated from 1D data. They are summarized in below in the Table 6.

The only difference is that the dimension has increased by one correspondingly and the techniques of visualization have changed accordingly. The visualization techniques change with respect to the dimension of the data as seen from Table 6.

In case of 3D data, it should be noted that every cell is defined in 3D space with three variables, two dictating its position in the 2D cellular space, while an additional variable for the time-step.

Type of result	**Domain**	**Dim**	**Visualization technique**
Instantaneous Activity	Spatio-temporal	3D	Surface Plot images (GNUplot) and movies (AVS-Express)
Accumulated Activity	Spatio-temporal	3D	Surface Plot images (GNUplot) and movies (AVS-Express)
Activity Factor	Temporal	2D	Surface plot images (GNUplot)
Statistics	Temporal	2D	Surface plot images (GNUplot)
Living Factor	Cellular	2D	Surface plot images (GNUplot)
Maxima Locator	Temporal	2D	Surface plot images (GNUplot)
Peak Locator	Cellular	2D	Surface plot images (GNUplot)
Region Of Imminence	Spatio-temporal	3D	Binary visualization (GNUplot)
Peak Bars	Spatio-temporal	3D	Zero padding (GNUplot)
Max Bars	Spatio-temporal	3D	Zero Padding (GNUplot)

Table 6: Visualization of results for 2D space

Visualization of 2D data uses the z-axis to represent the value of the variable to be visualized. However, in 3D data if we need to represent the whole process in one image,

the z-axis needs to represent the time-step in this case. Thus, we can represent values of the variable to be visualized by colors. This would however, make the image opaque restricting the visibility only to the boundary variables. There are techniques in sophisticated visualization software like AVS-Express to increase the transparency of the image or volume rendering of the image, but it would not give a clear insight to the inner details. A better way to visualize the 3D data is using movies, with each time-frame corresponding to a time-step. *Activity Generator* and *Pattern Predictor* produces 3D data, which needs to be visualized by movies as described above or by surface plot images for a particular time-step of interest as shown in Figure 25.

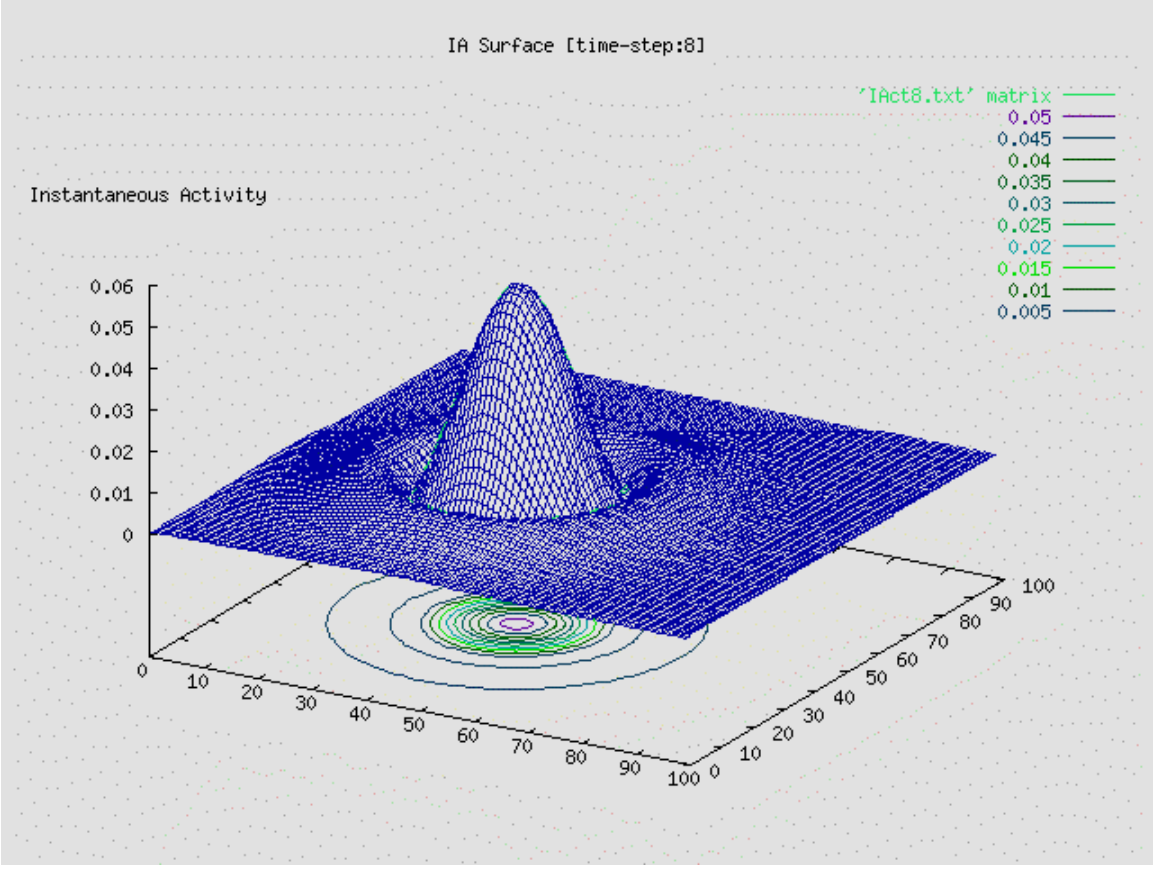


Figure 25: Surface Plot Images for 2D spatial visualization

Figure 25 above is a surface plot for the *Instantaneous Activity* of a 2D diffusion process at a particular time-step. Contours are drawn on the base to show the regions of equal values.

The data generated by the *Pattern Predictor* modules (*ROI, Peak Bars and Max Bars*) are also 3D and are visualized in the same fashion as described above, however since zero padding and binary visualization is used for such data, it is a good idea to visualize the values as discrete point as shown in Figure 24.

The *Statistic Analyzer* for 2D time-varying data produces resultant files which itself is 2D in nature. These results are visualized with GNUPLOT surface plot function as images.

6 OBSERVATIONS

This chapter presents the observations found by processing data obtained by a number of different types of simulations. For data in 1D space we have:

- data generated by PERL script as shown in 3.2.3
- 1D heat diffusion
- activity of state of robots

For data in 2D space we have:

- 2D heat diffusion
- Fire spread model

6.1 DATA FOR 1D SPACE

6.1.1 TEST DATA GENERATED BY *dataGenerator* SCRIPT

For 1D space, first the *Activity Engine* was tested with data generated by the mathematical equations shown in Equations 6.1, to model the behavior of the process in terms of a function represented by *f(x,t)* (N=100, T=100):

Equation 6.1:

Type 1 $$\frac{(NCells)}{(0.2*x+1)}\left|0.1*\sin(0.2*t)\right| \quad \ldots \text{(a)}$$

Type 2 $$\frac{(0.5*NCells)}{(x+1)}\left|\sin(t)*\cos(t)\right| \quad \ldots \text{(b)}$$

Type 3 $$\frac{(NCells)}{(0.2*x+1)}\left|\sin\left(\frac{t}{x+1}\right)\right| \quad \ldots \text{(c)}$$

The first two types of test data have activity concentrated in the same cellular area throughout the temporal domain; they differ in two aspects, the rate of decrease of activity and the number of peaks at t=0 as seen in Figure 26.

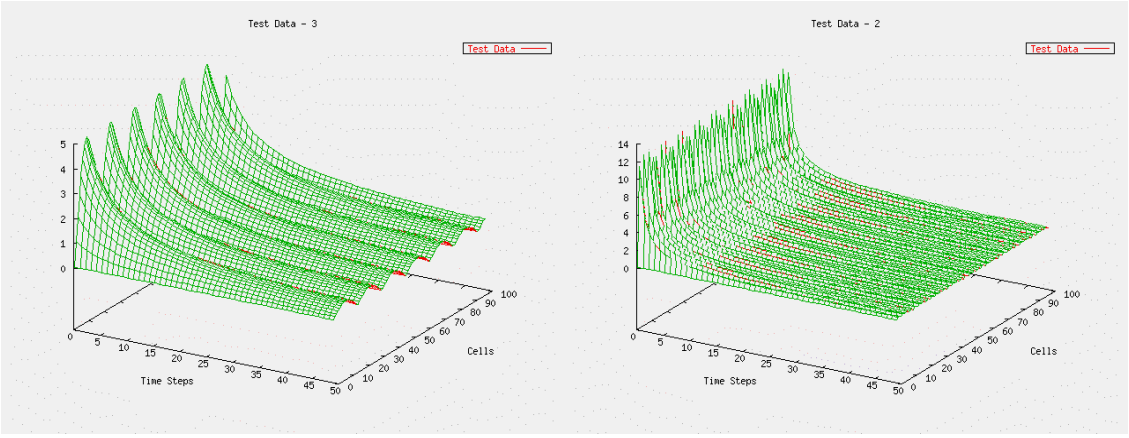


Figure 26: Test data 1,2 in value domain

As seen from the figure the type– 2 stabilizes sooner than type - 1. The *IA* curves are found to be similar to plots above, while the *AA* curve for the first type has a gradual rise and the second type of data is characterized by a steeper rise of AA curves as expected.

The *Activity Factor* is found to form a similar pattern fairly independent of the threshold set for it, due to the periodic nature of the function over the cellular domain. The plots obtained by the statistical analyzer are seen to be characterized distinctly by the periodic behavior of the sine and cosine functions in the Equation 6.1, as seen in Figure 27.

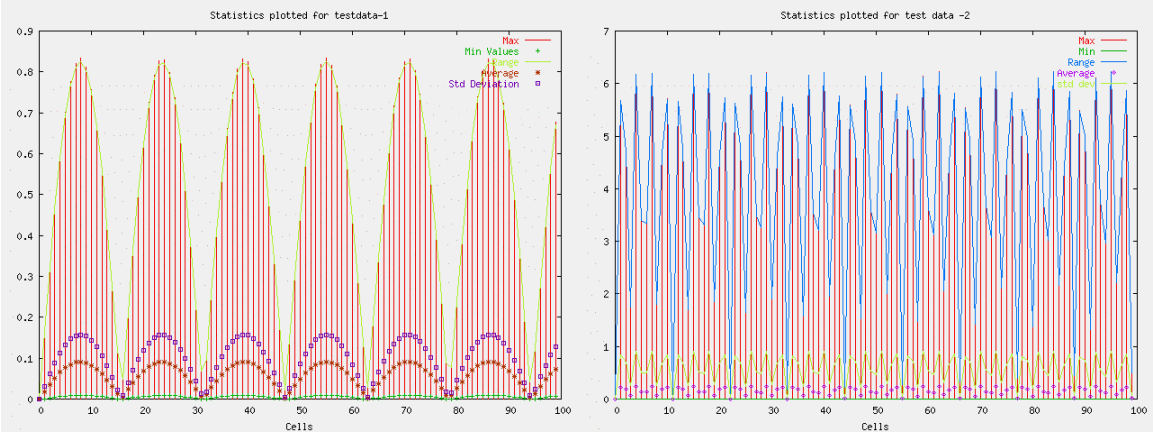
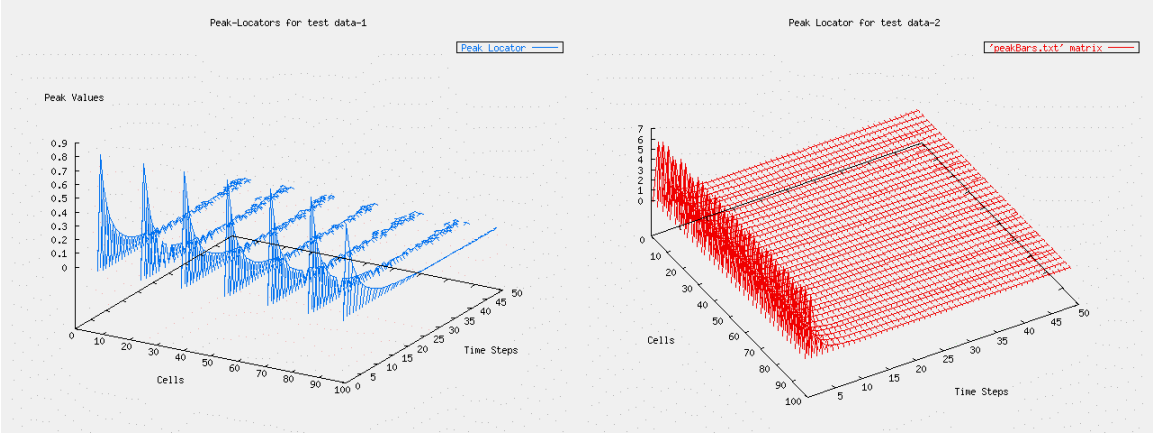
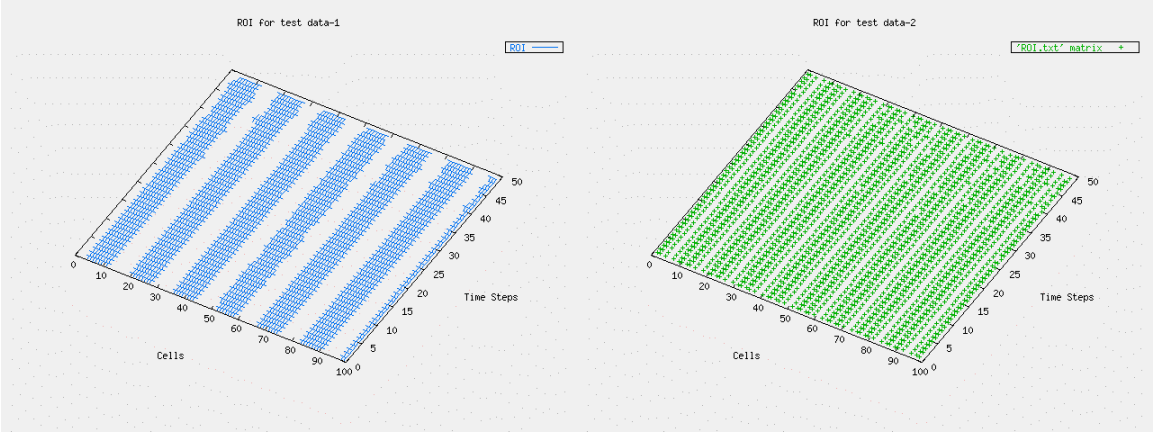


Figure 27: Results from Statistical Analyzer for test-data 1,2

The Pattern predictor module computes the peaks present in the IA curve and predicts the ROI based on it. As seen from Figure 28, the first type of data is characterized by lesser peaks and consequently the ROI is much smaller than that of second type.



(a) (b)



(c) (d)

Figure 28: Results from Pattern Predictor module for test-data1,2

Finally, to test the results obtained from the Pattern Predictor module, a graph is plotted in the temporal domain, with the sum of IA values for the whole cellular domain, the Max locations and the imminent locations. If the graph of the imminent locations closely follows the first graph, the ROI is a true measure of the active region.

The first two types of data were not characterized by any shift of activity in the process and the active regions pretty much remained the same throughout the time of the process. The third type of test data was generated to test the case of activity shift in the temporal domain. Figure 29 shows the plot of IA curve and AA curve for the 3rd type:

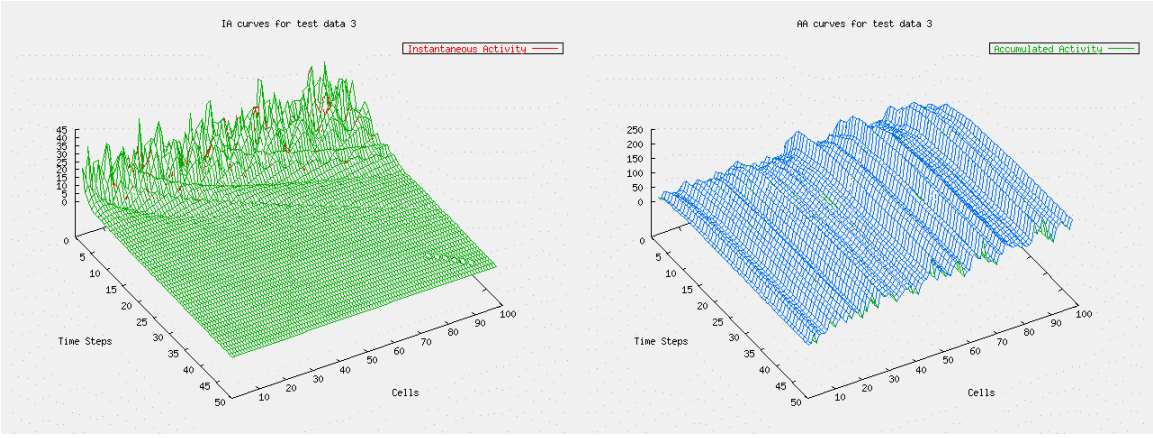


Figure 29: IA/AA curve for test data-3

As seen from Figure 29 there is a distinct shift in activity from one side to other during the course of time. The peaks are not intuitive from the Activity curves nor is the shift of activity seen clearly. Once the IA values are processed by the *Activity Engine* we can clearly see the presence of Peaks and the imminent region signifying the shift of activity in the process. The results are shown in Figure 30 below.

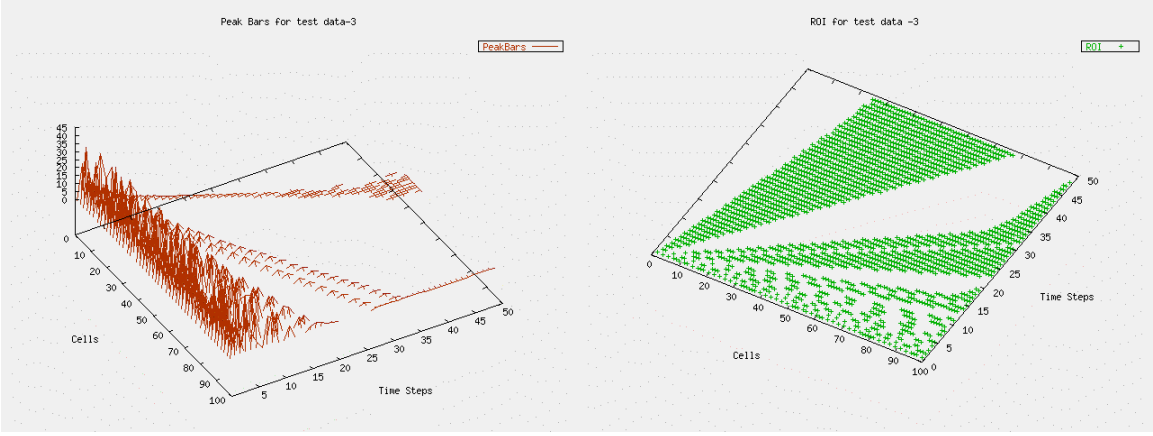


Figure 30: Results from Pattern Predictor module for test-data3

Figure 31 shows that activity detected by the *Max Locator* module would not be an appropriate measure of the Activity for this process since the impulses, signify the sum of the IA values for the cells detected by the *Max-Locator* module, the line denotes the IA values computed by the *Peak Locator* module and the points is the actual representation of the activity of the process.

It is clearly seen that the *Peak Locator* gives a much true measure of the activity of the process in this case.

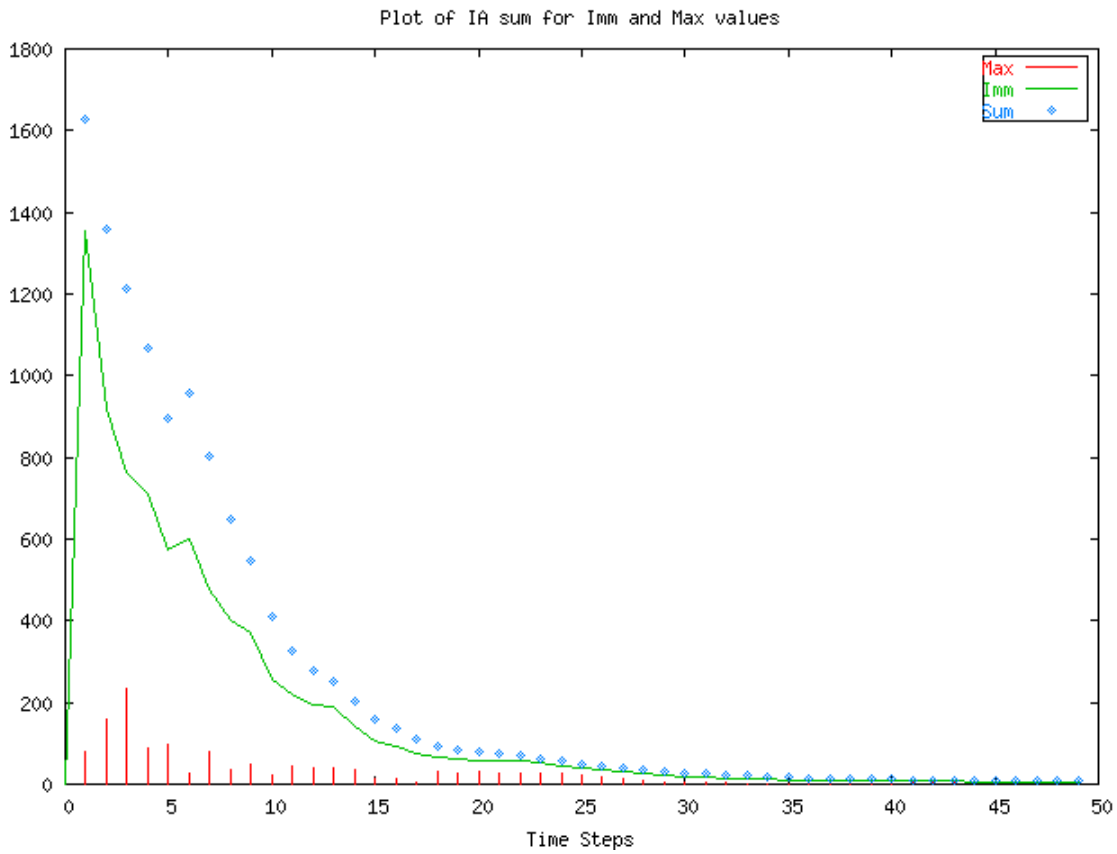


Figure 31: Check for Peak-Locator and Max-Locator module for test data-3

6.1.2 1D DIFFUSION PROCESS FOR N=10, N=100, N =200

Data obtained from 1D diffusion process was analyzed by the system with the parameters shown in Table 7.It also shows the results such as the global average activity and the total accumulated activity for the whole process.

Parameters	**N=10**	**N=100**	**N=200**	**N=200**
T	100	100	100	50
ROI (e)	0.9	0.9	0.9	0.9
LFac (e)	Local	Local	Local	Local
AFac (e)	Global (0.002186)	Global (0.000375)	Global (0.00194)	Global (0.004098)
Global Max(Tadv)	1001624	1000225	1000225.	1016801
Global Min(Tadv)	3.800186	1.102626	1.037858	3.8713
Global Max(IA)	0.263145	0.90693	0.9635	0.2538
Global Min(IA)	9.98377e-7	9.99e-7	9.99e-7	9.8e-7
Total AA	2.428327	3.8296	3.92825	2048.887

Table 7: Process Characteristics for 1D/2D diffusion

The IA plot for the diffusion process is shown in the Fig; as seen from the figure not much information about the process can be obtained from this plot. However the results obtained from the *Activity Factor* module shown in Fig distinctly signifies the region of cells which have been more active the

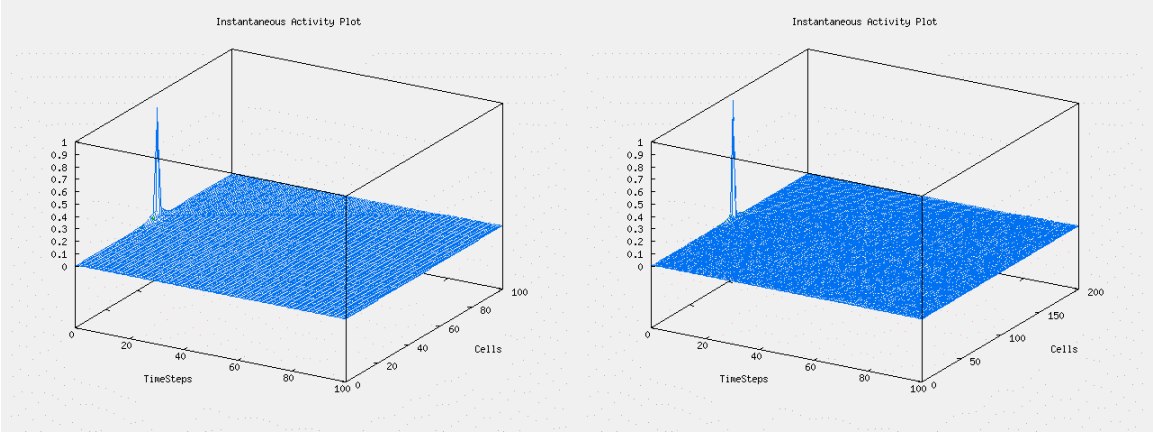


Figure 32: IA curve for 1D diffusion process (N=100, N=200)

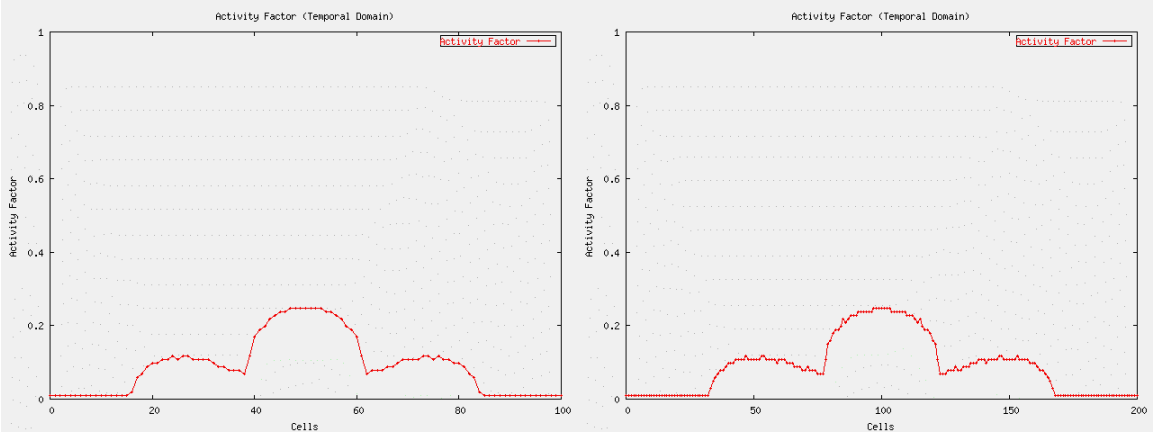
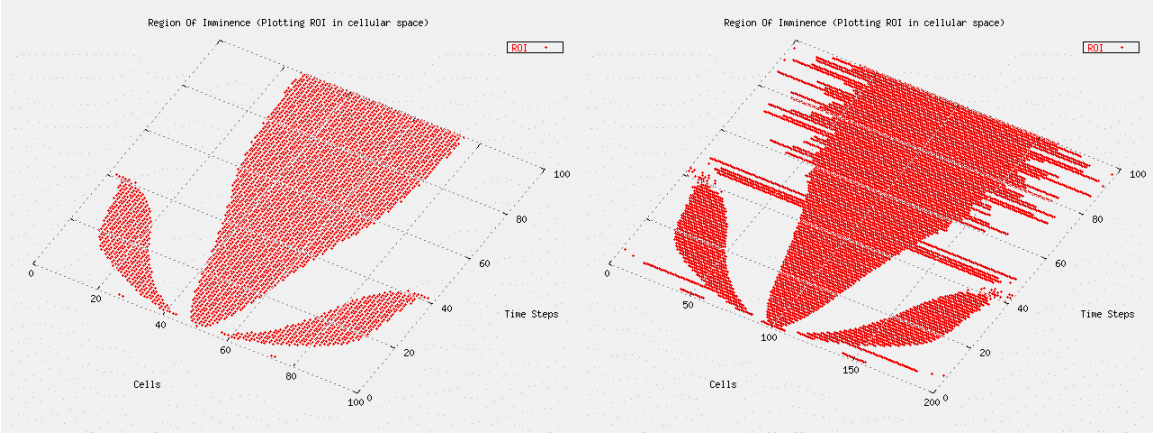


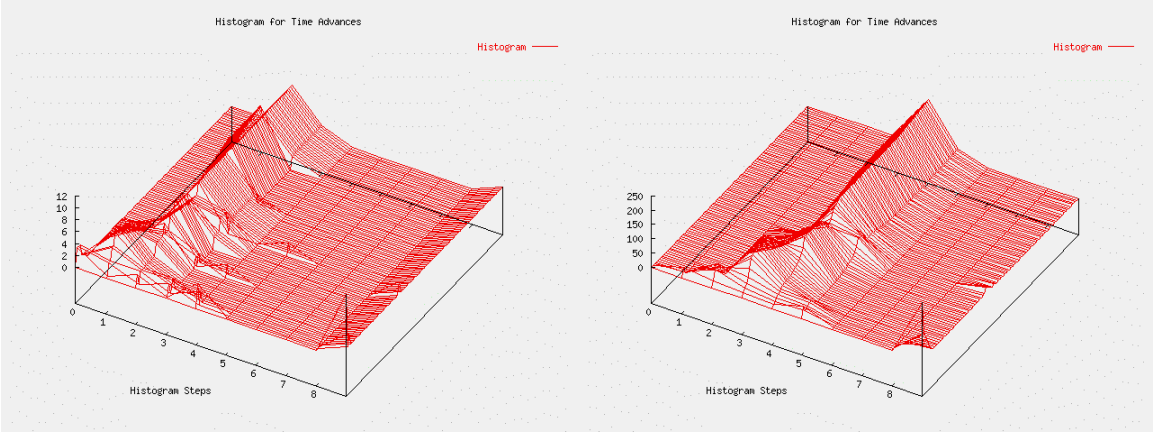
Figure 33: Activity Factor plotted for 1D diffusion (N=100. N=200)

The ROI computed by Pattern Predictor module is shown in Figure 34. Three distinct regions are observed from the module signifying the shift of activity in three different directions. Finally, Figure 35 plots the histogram of the process for N=10 and N=200. From the plot the speed of the process can be computed based and it is found that the process for N=200 is characterized by larger *time advances*, clearly signifying that it takes more time to schedule events. In other words, the IA curve is characterized by a steeper curve.



(a) N=100 (b) N=200

Figure 34: ROI for 1D diffusion (N=100, N=200)



(a) N=10 (b) N = 200

Figure 35 Histogram of Time-Advances for 1D diffusion (N=10, N=200)

6.1.3 ROBOT ACTIVITY – (MODELING TRANSITION OF STATE)

Data was visualized for robot activity modeled at ACIMS [14] using DEVS simulation, where a robot is either said to be active (moving) or passive (stopped). Every robot communicates with its successor and predecessor in form of events. The whole process is dictated by an algorithm, which defines some rules regarding the state of a robot. For example, if a robot is lost all the other robots stop moving. To get an overall picture of the activity of the process, the data generated by the simulation was processed by the *activity modeler* system.

The data obtained from modeling the activity of robots had to go through a separate pre-processing stage, as the data was obtained was present with a simulation time instead of real-time. Thus, the data had to be converted to real-time with equal intervals in time domain. The data present was binary, with a 1 present for a moving robot and a 0 present for a stopped robot. Since the data is just binary we don't have to

compute the thresholds for the various factors and can simply set them to any value in between (0,1) (since maximum(IA) = 1, minimum(IA) = 0 for binary data). The threshold was set to 0.5. The data was present for around 2000 simulation time steps (around 2300 real-time steps) and was the data was analyzed using the *Activity Modeler* system.

The activity in this case would be the change in state for the robot. Also the cells were represented by the robots (N=30, the data was presented for 30 robots). The total number of transitions was found to be 5324. It was found that the average IA for all the robots was pretty much the same (< 20%) and equal to their *Activity Factor* as shown Figure 36. The *imminent factor* and *living factor* were found to be the same (as seen Figure 37), since for binary data, the ROI for a particular time step.

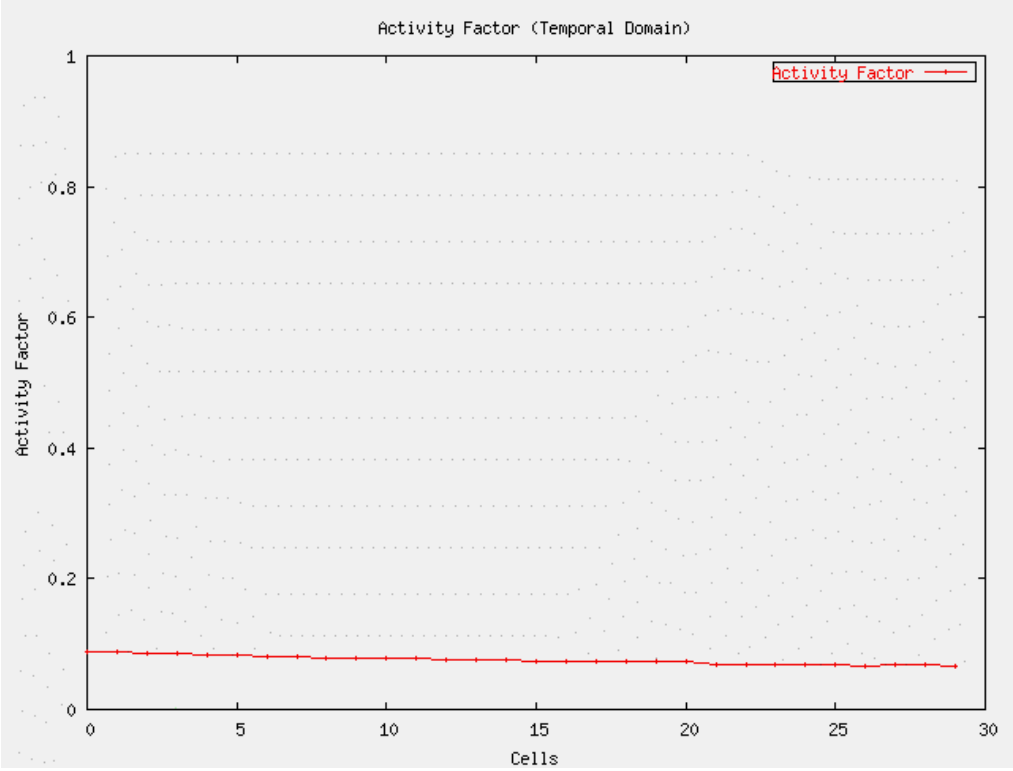


Figure 36: Activity Factor for robot activity

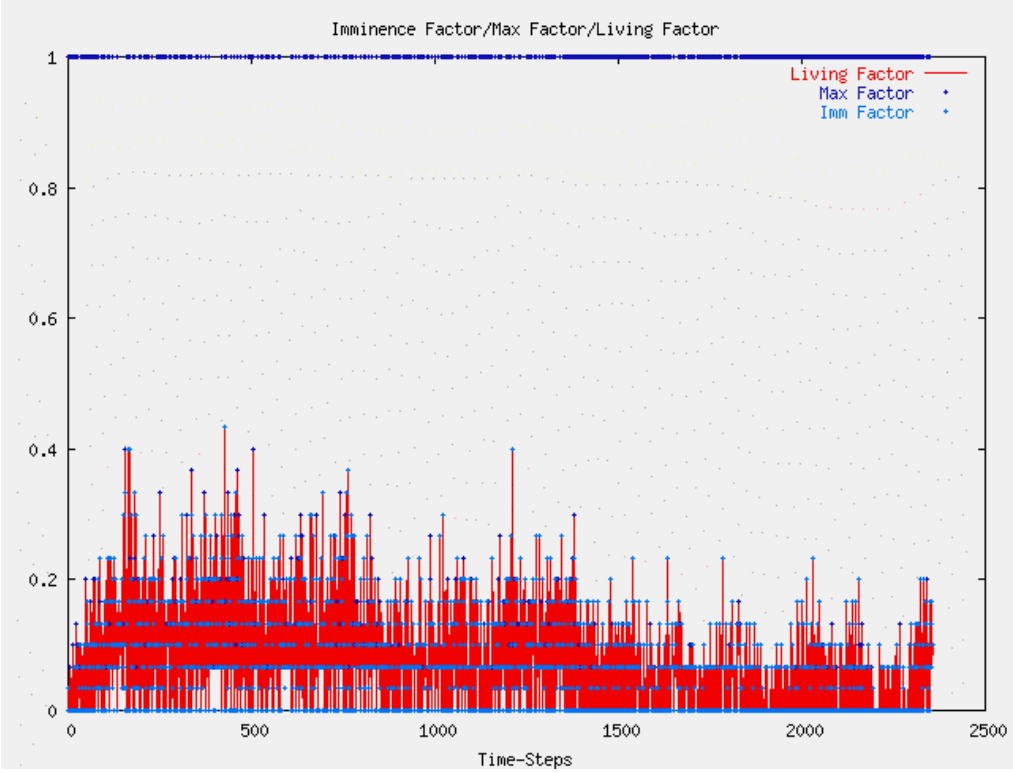


Figure 37: Imminent Factor and Living Factor for robot activity

6.2 DATA FOR 2D SPACE

6.2.1 2D DIFFUSION PROCESS

The parameters set for the system for 2D diffusion and the results obtained for the process can be found from Table 7. The histogram for *time advances* was found to be concentrated in the first two steps of the histogram as seen from Figure 38, signifying clearly that the process was not characterized by steep changes in the IA curve and the diffusion process was gradual unlike the 1D diffusion. The results from the *statistic analyzer* are now for a 2D space and have to be represented as surface plot images for

each of the result (For example the standard deviation for the IA values is represented by the surface plot image in Figure 39)

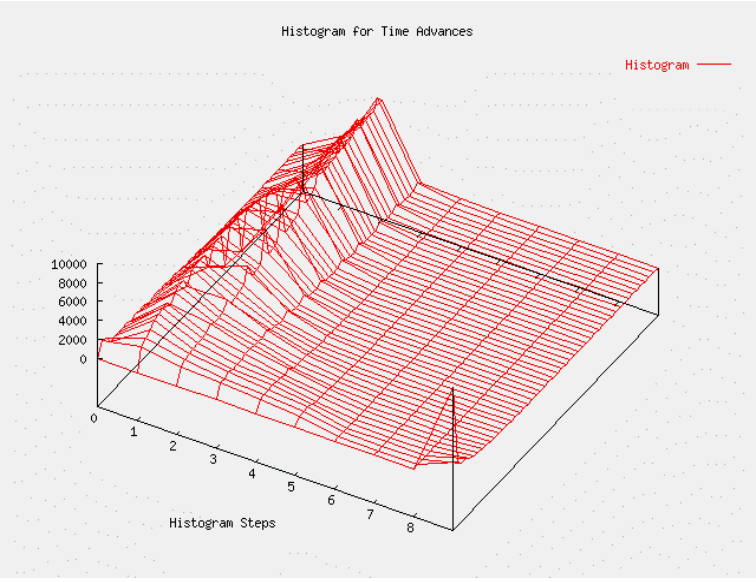


Figure 38: Histogram of Time advances for the 2D diffusion problem

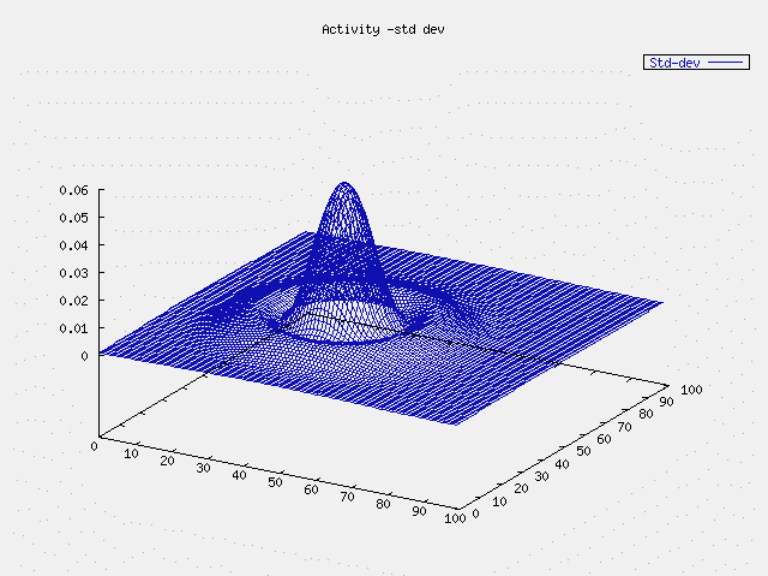
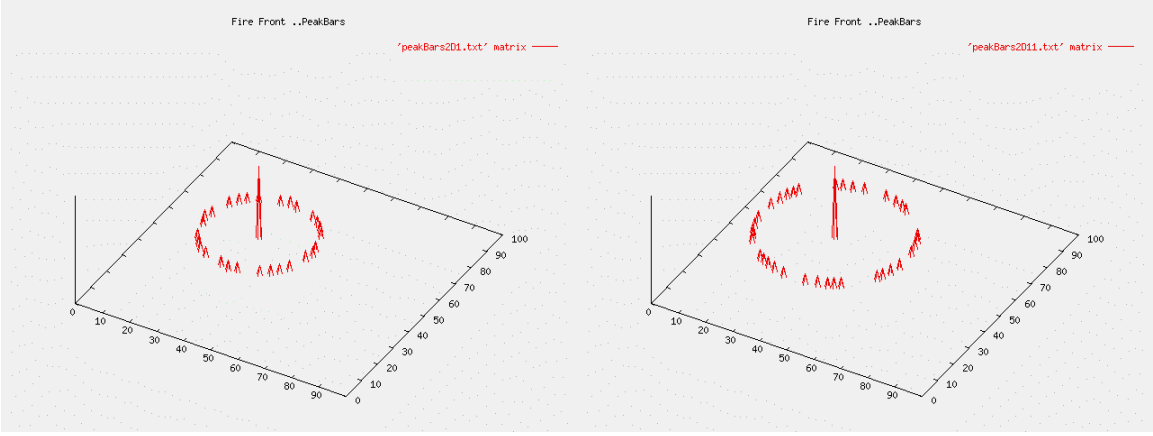


Figure 39: Standard Deviation for the 2D diffusion problem

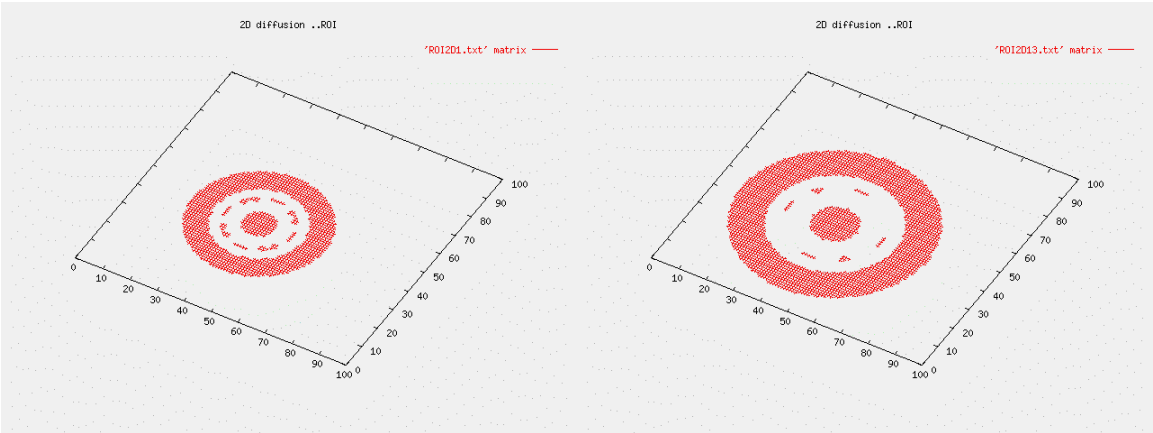
Most of the results from the pattern predictor module were in the form of movies

and were characterized by two concentric circles moving out over from the center towards the boundary over the period of time. This happens because the first derivative of temperature (rate of change of the temperature) was found to be higher at those locations. Based on the IA surface, it was found that the peaks formed a concentric circle moving in the direction of diffusion and so did the ROI move out specifying the imminent cells in the process.

Figure 40 shows the peaks in the IA surface and the ROI calculated from them at t=2 and t = 20.



(a) t =2 (peaks) (b) t =20 (peaks)



(c) t = 2 (ROI) (d) t = 20 (ROI)

Figure 40: ROI and Peak Bars for 2D diffusion process (t=2, t=20)

6.2.2 FIRE FRONT MODEL

A fire-front model [15] was simulated to model the propagation of fire in forests. There was a need to study the behavior of two phenomena, namely, ignition and burning. By theory we know that as a fire-front approaches a cell, it raises the temperature of the cell to a certain ignition threshold, beyond which the burning process begins. The burning process is characterized by a steep increase in temperature till some maximum value, followed by a steep decrease in temperature. If the temperature of the cell did not reach that threshold, it doesn't go through the burning process.

With the above knowledge of a burning procedure, it is possible to analyze the fire-front model. The data present was for 300 time steps. The results for the process are tabularized in Table 8.

Global average *IA* (*AFac (e)*)	1.791918
Global Max (*Tadv*)	1.00
Global Min (*Tadv)*	0.004673
Global Max (*IA*)	213.995
Global Min (*IA)*	1
Total Accumulated Activity	5321979

Table 8: Results for Fire-Front model (100 x 100, T =300)

The data was processed by the *Activity Modeler* system to get an overview of the whole process. It was found that around t=180, the fire fronts reached the cellular boundary, which was distinctly seen by a drop in the *Living Factor* curve from Figure 41. It is also seen from Figure 41 that no more than 20% of the cells were ever active throughout the duration of the process. Figure 42 shows that although the ROI module

did not gave a true representation of the active regions in the process from t [50-150] for e=0.7, it wasn't accurate enough, since the total activity of the imminent cells is considerable lower than the actual activity. This was the time when the fire-fronts reaches the boundary.

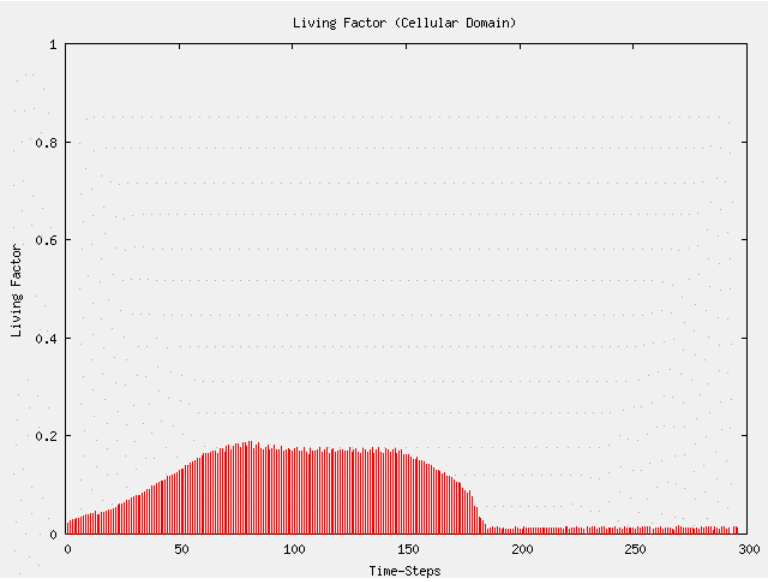


Figure 41: Living Factor for Fire-Front data (100 x 100, t=300)

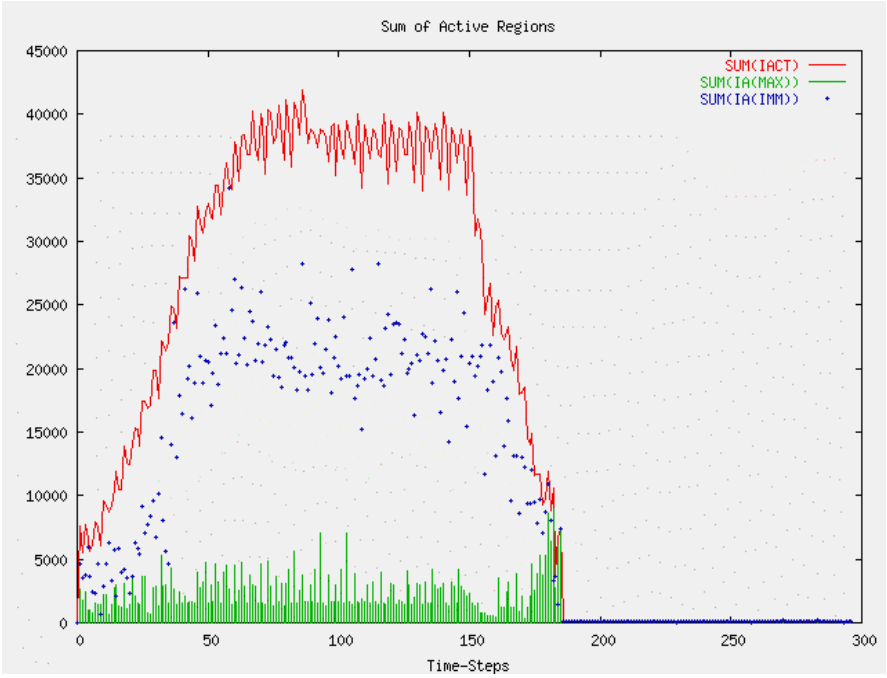
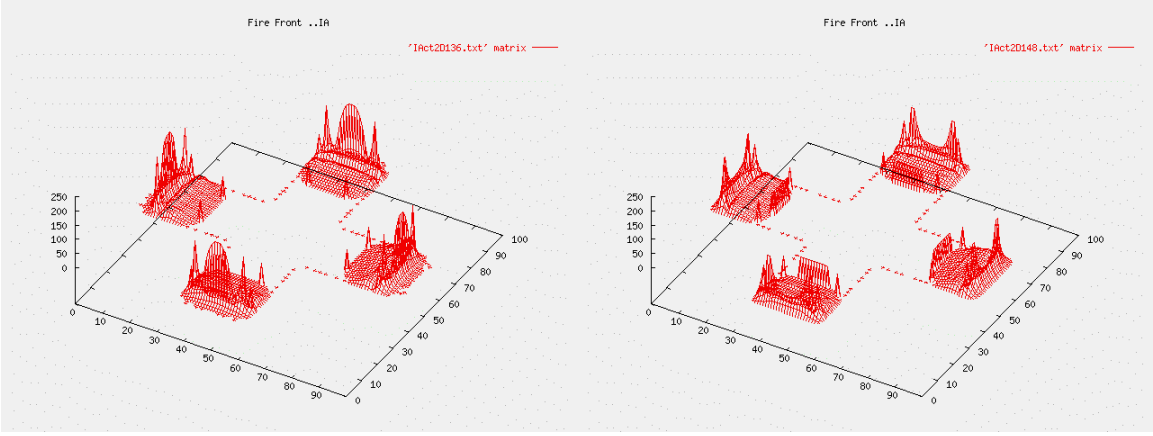


Figure 42: Sum of IA values for all cells and imminent cells

The movie of the data generated by the *Activity generator*, for the *instantaneous activity* surfaces, was characterized by two distinct activity patterns along the fire fronts. The first activity pattern is due to the igniting process, while the second pattern following it is due to the burning process. There were distinct oscillations around the high activity patterns found on the igniting fronts. Figure 43 shows the activity patterns for two time-steps (t=136, t=148), after the fire breaks in four linear fronts.



(a) t = 136 (b) t = 148

Figure 43: Instantaneous Activity Surface for Fire-Front model

Finally, to detect only the igniting process, the *Accumulated Activity* was modeled, setting a lower and a higher threshold to focus only on the desired temperature range of burning [373K]. This helped to characterize the igniting process in terms of the direction of ignition and length of fire-front.

7 AUTOMATION AND OPTIMIZATION

7.1 AUTOMATION OF THE SYSTEM

Since the System consists of three distinct stages, starting from the formatting of raw-data to the visualization of fancy images, with a number of scripts and executables processing data at various stages, there is a need for automation in the process.

It is possible to automate the whole system using a script making system calls to other scripts and executables of the system. The following features are required to aid the automation process:

- Standardization of filenames for 1D and 2D data.
- Setting correct path for the executables and data generated from intermediate results.
- Use of command line arguments to customize the system for the data under consideration.

Apart from standardization of name for filenames generated by the modules, the raw-data files also need to be named in specific manner for 2D data files. They should of the form '*filenameBase+<time-step>+.filenamebase'*.

Regarding the path of the executables, the system has a pre-defined directory structure as shown in Figure 44. Note that the root path is relocatable. The relative paths of the code and data should not be changed at any time.

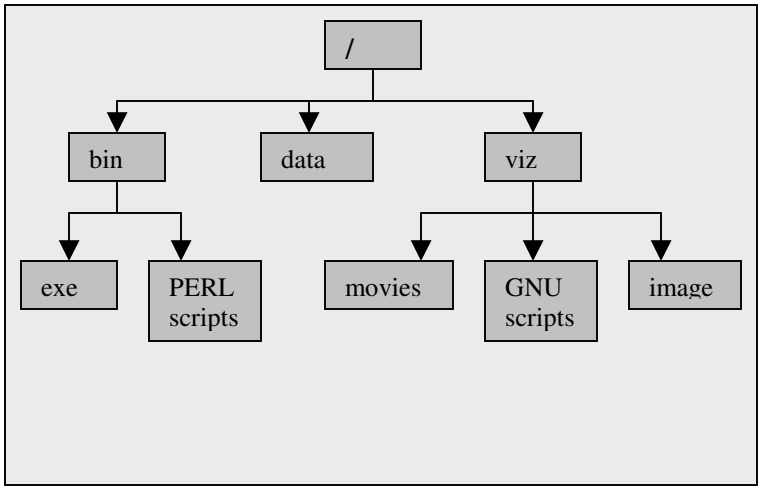


Figure 44: Directory structure of system

Command-line arguments for the *formatter* and *activity engine* are given below (can be found by a '–h' command). The parent script, which facilitates automation, needs to have command-line arguments, which in turn are mapped to appropriate commands of the internal scripts and executables. For example, for visualizing 2D spatial data the *Activity Engine* (*karma*) needs to turn on its '-2D' flag and the *formatter2D* PERL script should be used instead of the *formatter* script. Thus, the user needs to provide sufficient enough information to parent script for correct automation process.

```
Formatter.pl
For correction      -f fname
For extraction      -f fname -e rows

Formatter2D.pl      -C [<concat>||<expand>]
To concatenate      -f fnamebase
                    -s time-steps
To expand           -f fnamebase
                    -r <row-size>
                    -c <col-size>

Karma.exe           -extract [<stats>||<activity>||<pattern>||<all>]
                    -file <filename>
                    -2D
                    -fast
                    -transpose
                    -noactivity
                    -predict <#steps>
                    -cells <#cells>
                    -steps <#steps>
                    -rows <#rows>
                    -cols <#cols>
```

Figure 45: Command-line arguments for automation

At present automation is present only for GNUPLOT visualization modules on Windows platforms, due to cross-platform development issues discussed which will be discussed in the Chapter 8.

The parent script, which provides automation, is called the *Activity Modeler* script and its flow path is given in Figure 46.

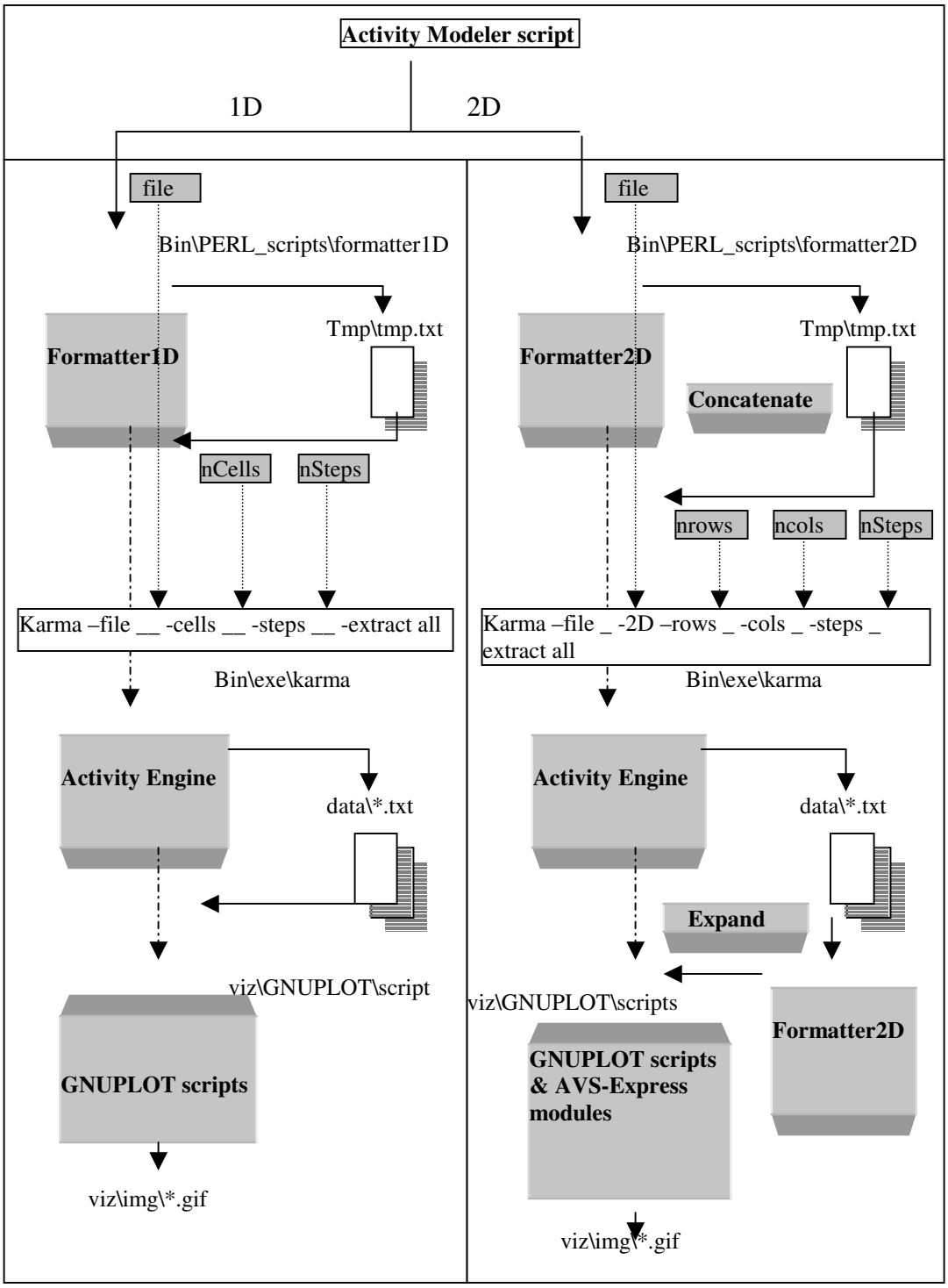


Figure 46: Automation Process for the ‘Activity Modeler’ system

7.2 OPTIMIZATION OF THE SYSTEM

Run-time visualization of time-varying data creates a necessity for the modules of the system to process data as quick as possible. Although, the visualization system described here is not strictly run-time, it can be used to process time-varying data at periodic intervals, say every n time-steps dumped by the simulation generating the data (discussed in Chapter 9). This would impose a certain restriction on the data-processing of our visualization system, which in turn would create a necessity to make a more efficient use of the computation power, thus necessitating the optimizing of the code. Most of the optimization techniques discussed here are inspired by those used in the real-time systems and embedded software. They are discussed below.

- *Local Variables:* Minimize local variables so that the compiler can fit them in registers (improving performance over accessing them from memory) avoiding frame pointer operations on local variables that are kept on stack (reducing overhead of restoring frame pointer). Declare local variables in innermost loop.
- *Function Calls:* Reducing the number of parameters due to expensive parameter pushes on the stack call. Passing parameters by reference using pointers and references. Specifying a return values only when needed. Declare small functions inline.
- *Data types:* Prefer the use of int over char and short as C++ perform arithmetic operation and parameter passing at integer level.
- *Constructors:* Defining light weight constructors specially helps for an array of objects. Using constructor initialization lists helps in increasing the performance by eliminating

the slower assignment operations over the initialization operations.

- *Loop optimization:* Eliminate loop independent expressions in the body of loop. Merge loops if possible. Use loop un-rolling for smaller number of iterations. If possible decrement the *for* loops as the condition (*i!=0*) is executed much faster than (*i<max*) Use early loop breaking if condition is satisfied.

It is a well-known fact that there is always a trade-off between speed and memory usage. Since, we have to process large time-varying data-sets optimization not only in CPU-time but also memory usage should be considered equally important. Although, the memory bottleneck depends significantly on the RAM capacity, significant optimization of memory usage can be realized by the following techniques:

- Eliminate *memory leak problems* by de-allocating the memory whenever necessary and pointing unused pointers to NULL to combat the *problem with dangling pointers*.
- For *Statistic Analyzer and Pattern Predictor* modules, where the number of peaks or maxima are not known until run-time, there two ways to allocate space for the results
    - Using linked-lists by adding a peak as it is found. The advantage is that we have to traverse the Instantaneous Activity array only once.
    - Using dynamically allocated arrays. In this case we would have to traverse twice through the array; first to find out the number of Peaks (for eg) and second time to store the values at those particular locations.

It is found that although one had to traverse the array twice, it was much more efficient to use dynamically-allocated arrays over linked lists for even a small number

of peaks due to simpler logic and random access of array structures while using them at a later use.

- Allocating space for the array under consideration only once by the *Data Engine* module and referencing it later by other modules in the system.

Apart from the standard techniques described above, there are a few other techniques implemented to provide optimization in time and memory which are turned on by the '*-fast'* flag in the *Activity Engine* module.

1. If it is known to the user that the data values are in integer format or it is known if the fractional part of the floating point can be ignored, he can turn on the *–fast* flag and the resultant data processing would be done in integer arithmetic as much as possible. Since this would eliminate the use of floating point unit, there would be a considerable increase in the performance of the system as well as reduction in the memory usage during run-time. This point has been illustrated by comparing the time taken to generate test data for integer data-type and floating point data-type for the same function.

The Table 9 illustrates the time taken by the *dataGenerator* module to compute a simple function $z = f(x, y) = x^2 y$ with the integer flag turned on and later with the flag turned off. The code is benchmarked using PERL's benchmark module.

Cell Dimension (x, y)	**Integer Flag (ON/OFF)**	**Wall Clock time(sec)**	**System Time (sec)**	**CPU Time (sec)**
100 X 100	ON	1	0.01	0.04
	OFF	1	0.01	0.07
500 X 1000	ON	3	0.09	1.61
	OFF	7	0.12	3.44

Table 9: Optimization in time using the integer flag

2. The user can also turn on the *-fast* flag to save considerable amount of time that the *Statistic Analyzer* spends in generating results like standard deviation. Since standard deviation needs to compute the square root of a number, which is compute-intensive operation, it can be bypassed, for larger data-sets, by turning on the *–fast* flag.

8 SUMMARY AND FUTURE WORK

8.1 SUMMARY

The *Activity Modeler* system gives a new perspective for analysis of data based on the concept of *Activity* in DEVS. The system was used to generate results from a variety of data produced by DTSS and DEVS simulations. The results were found to give a true representation of the process and helped to study the behavior of the process in the activity domain. Results were produced in both, the temporal and the cellular domain to give a better insight of the process as a whole as well as for a particular range of time steps or cells.

The *statistical analyzer* is used to compute a number of results, which were inherently static in nature, like the global and local average, maxima and minima of *instantaneous activity* and *time advances.* The *pattern predictor* module is based on the fundamental that the whole process can be characterized on the basis of its activity curve (1D) or surface (2D), by detecting its localized peaks. It introduces a concept to predict the imminent regions based on the detected peaks. The *pattern predictor* also introduces a concept to predict the imminent regions in the future based on the present active regions, by detecting the direction and shape of activity pattern.

For 1D diffusion process, it was found that the *living factor* curve was similar for N=100 and N=200, and it peaked to 0.4 around t=10 and decreased gradually, while the *activity factor* was characterized by a distinct curve immaterial of the size of cells, and was found to never exceed 20%.

While modeling the robot activity, it was found that the *activity factor* for all the robots was more or less equal to each other (approximately 10%). The *living factor* modeled the activity for all the robots in the temporal domain and showed the time-steps of high active regions (t=500, around 40% of the robots were active) as well as the time-steps when a robot got lost when the *living factor* dropped to zero. It was also found from the *linear span* and *ROI* module that the maximum group of imminent robots was never greater than 6.

Data analyzed by fire-front model showed the time of maximum activity (t – [70,140]) when around 20% of the cells were active. It successfully modeled the regions where the fire first ignited the cells and also brought into focus the oscillatory behavior of the ignition.

The concept of *activity* introduced from DEVS also proves from the following calculations that choosing a proper quantum for the simulation, DEVS can be much efficient than DTSS simulations. Based on the concept of Activity, the number of output transitions in DEVS, is given by the number of threshold crossings as

$$Number\ of\ transitions = \sum_{x}\sum_{t} IA[x,t] \Big/ q$$

For DTSS simulations the number of output transitions is given by

$$Number\ of\ transitions = NT \Big/ \Delta t$$

The maximum *instantaneous activity* in the cellular and temporal domain, which is the maximum slope (derivative) of the output values of a simulation in DEVS, is given by

$$Maximum\ (IA) = q \Big/ \Delta t$$

From the above three equations the we can compare the DEVS and DTSS simulations on the basis of the number of transitions taken by the process.

$$\frac{DEVS}{DTSS} = \sum_{x}\sum_{t} IA[x,t] \Big/ NT \times Maximum(IA)$$

The above ratio is computed for a number of processes and the results are presented in the Table 10. Clearly, it is seen that DEVS is much more efficient than DTSS simulations.

Model	**Cells**	**Time**	**Max (IA)**	**Total AA**	**DEVS / DTSS**
1D diffusion (N=10)	10	100	0.263184	2.4283	0.0093
1D diffusion (N=100)	100	100	0.9069	3.8296	4.22e-4
1D diffusion (N=200)	200	100	0.9635	3.9285	0.0002
2D diffusion	10000	50	0.2583	2048.77	0.819
Fire-Front	10000	297	213.995	5321979	0.0083

Table 10: DEV v/s DTSS based on activity concept

8.2 FUTURE WORK

At present, the *activity modeler* system supports data only for 1D and 2D spatial domain. It can be extended in the future for 3D space. It would impose some problems for visualization stage, as it is would be a certain problem to visualize data in 4D (spatio-temporal process).

The *activity engine* presented in this thesis has the *data engine* module developed in C++ with MFC classes. Hence it would not work on a UNIX platform. The *data engine* module needs to be coded using a cross-platform compatible library to enable the

use of the system on both, Windows and UNIX platforms. Also, at present the third stage (visualization) of the system at present is implemented only using the GNUPLOT scripts. If the visualization of 3D data is desired, the visualization stage of the system can be re-built with more sophisticated visualization software.

The ability of the *pattern predictor* module to compute the imminent regions for the whole process before hand, on the basis of only two time steps needs is unrealistic to some extent. At present, it is implemented only for 1D space and was tested for 1D diffusion process and test data-3 from the *datagenerator* script. The percentage error in the predicted and the actual ROI is given in the Table 11 and plotted in the Figure 47.

Time Steps (t)	Cell in error (%Error)			
	Cells = 10 (1D diffusion)	Cells = 100 (1D diffusion)	(1Dtest data3)	Cells = 200 (1D diffusion)
8	1 (10)	13 (13)	39(39)	26 (13)
15	1 (10)	21 (21)	13(13)	22 (11)
22	3 (30)	19 (19)	13(13)	24 (12)
29	5 (50)	27 (27)	13(13)	48 (24)
36	3 (30)	12 (12)	12(12)	23 (11)
43	2 (20)	15 (15)	18(18)	26 (13)
50	1 (10)	16 (16)	7(7)	28 (14)
57	1 (10)	8 (8)	7(7)	18 (9)
64	6 (60)	8 (8)	2(2)	14 (7)
71	2 (20)	10 (10)	2(2)	14 (7)
78	2 (20)	8 (8)	0(0)	12 (6)
85	2 (20)	8 (8)	0(0)	12 (6)
92	2 (20)	8 (8)	0(0)	12 (6)

Table 11: Percentage error in predicted and actual ROI for 1D processes.

For process whose activity curve is characterized by a non-linear curve (second order processes) the predictor module doesn't not work well. However, even for second order processes it is found that as the number of cells increases, the percentage error

decreases, even though the region of imminence increases.

The dotted curve in the Figure 47 clearly shows that the percentage error for the test data-3 (characterized by linear activity curve) is much lesser and diminishes to zero as time progresses

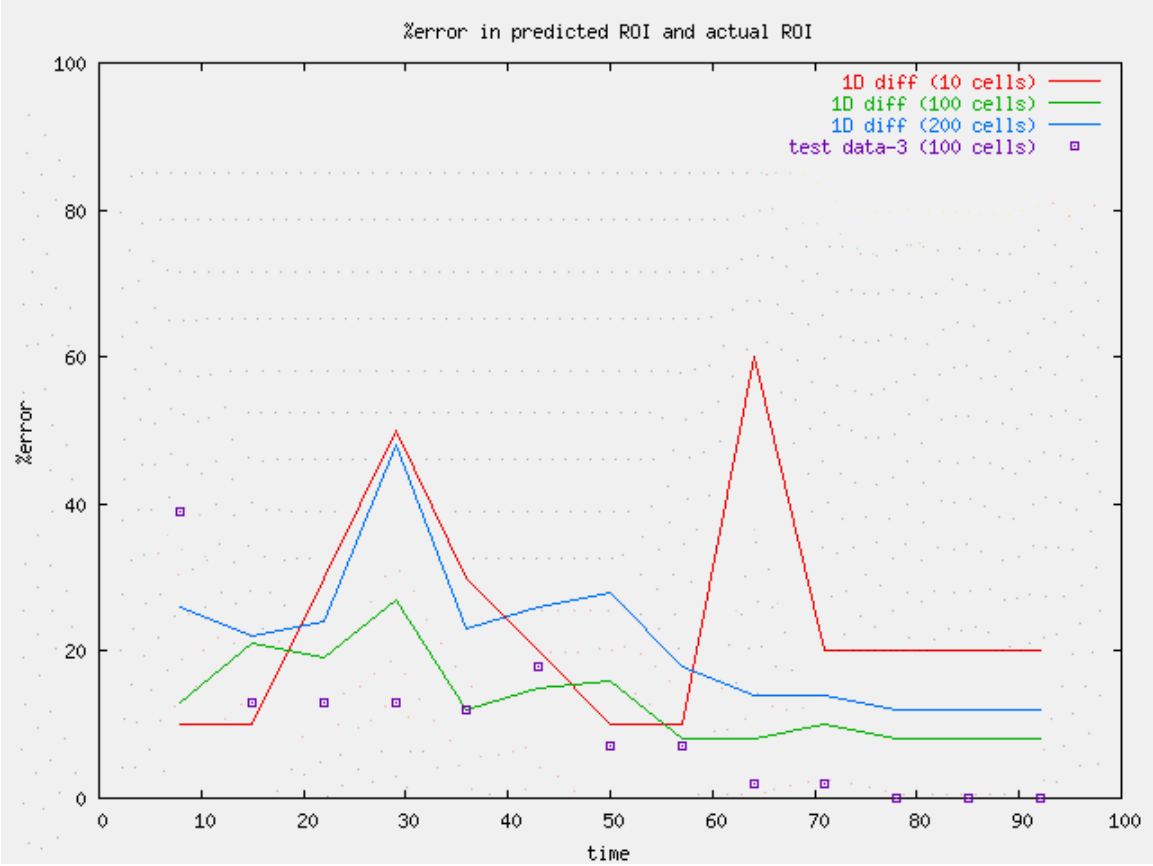


Figure 47: Percentage Error in the Predicted and the actual ROI for 1D process

REFERENCES

[1] Guangeng Ji and Han-Wei Shen, "*Time-Varying Iso-surface tracking with global optimization*", IEEE Visualization 2004 Conference

[2] Pak Chung Wong and R. Daniel Bergeron, "A tool for hierarchical representation and visualization of Time-Varying data", ICASE/LaRC Symposium, 1995

[3] Kwa-Liu Ma and David Camp, "High Performance Visualization of Time-Varying volume data over Wide-Area network", Proceedings of Super Computing 2000

[4] Jack Snoeyink, "A geometric basis for visualizing time varying volume data", IEEE Visualization 2003 Conference

[5] Chaoli Wang, Jinzhu Gao and Han-Wei Shen, "A Framework for rendering large time-varying data Using Wavelet based Time-Space Partitioning Tree (WTSP)", proceedings of Eurographics symposium on Visualization, 2004

[6] Zeigler, Bernard P., Herbert Praehofer, Tag Gon Kim, Theory of Modeling and Simulation: Second Edition. San Diego: Academic Press, 2000

[7] J. Davis and A. Bobick, "The representation and recognition of Action using temporal templates", in CVPR 1997

[8] B. F. Bremond and M. Thonnat, "Analysis of Human Activities described by Image sequences", in *Proc. Intl. Florida AI Research Symp,* 1997

[9] J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan Kauffman, 1976

[10] Rama Challapa, Namrata Vaswani and Amit Roy Choudhary, "Activity Modeling and Recognition using Shape Theory", in CVPR 2003

[11] Kristina Lerman and Aram Galystan, "Agent Memory and Adaptation in multi-agent system", AAMAS, 2003, Melbourne, Australia

[12] R. Bastos, D. Dubugras and A. Ruiz, "Extending Activity Diagram for Worflow Modeling", 35th Annual Hawaii International Conference of System-Sciences

[13] J. Nutaro, B.P. Zeigler, R. Jammalammadaka, S. Akerkar, "Discrete Event Solution of Gas Dynamics with the DEVS framework: Exploiting Spatio-temporal heterogeneity", ICCS, Melbourne, Australia, 2003

[14] Xiaolin Hu, B.P. Zeigler, "Model Continuity to Support Software Development for Distributed Robotic Systems: A Team Formation Example".

[15] Alex Muzy, "Fire Front Model"