A SKETCH RECOGNIZER FOR GRAPHS

by

Hamdi DİBEKLİOĞLU

Engineering Project Report

Yeditepe University
Faculty of Engineering an Architecture
Department of Computer Engineering
2006

A SKETCH RECOGNIZER FOR GRAPHS

APPROVED BY:

Assist. Prof. Dr. Ender Özcan (Supervisor) ……

Assoc. Prof. Dr. Raşit Eskicioğlu ……....

Assist. Prof. Dr. Cem Ünsalan ……..

DATE OF APPROVAL: 24/05/2006

ACKNOWLEDGEMENTS

I would like to thank to my advisor, Ender Özcan, my cousin Hüseyin Hışıl, Cem Ünsalan and Tevfik Metin Sezgin for their advice and support all through my project.

Additionally, I want to thank my family; Mehmet Dibeklioğlu, Sevim Dibeklioğlu and Sami Dibeklioğlu for their unconditional love and support during my education.

ABSTRACT

The current trend on the use of touch-screens has introduced the need for keyboard-less solutions to graphical user interfaces. Gesture-based interfaces as a piece of pie, receive the scientific interest because of its theoretical diversity. Even näive approaches require a common knowledge about linear algebra, artificial intelligence, sketch and handwriting recognition within the context. The main emphasis of this project is to develop an intelligent pen-based human-computer interface and propose a new online hand-drawn digit recognition algorithm which runs with linear time complexity, where one of the most recognized algorithms, ICP (Iterative Closest Point) halts with quadratic time complexity. In return, we show how the proposed algorithm, together with ICP, can be applied to increase the success ratio of digit recognition significantly without the inclusion of a neural network. Moreover, the derived information is represented by an adjacency matrix to supply an abstract view of any kind of input graph. As sample applications, Kruskal and Dijkstra algorithms are fed using the output generated by the implemented sketch recognizer system. In conclusion, this work outlines a new solution to the graph creation problem with hand-drawn sketch and digit recognition.

ÖZET

Günümüzde dokunmatik ekran teknolojisine olan talep, grafik kullanıcı arayüzlerini klavyesiz kullanabilme ihtiyacını doğurmuştur. Pastanın bir parçası olan, çizim tabanlı arayüzler, teorik çeşitliliği sebebiyle bilimsel ilgiyi üstüne çekmektedir. Bu konudaki basit uygulamalar bile genel olarak, lineer cebir, yapay zeka, çizim ve el yazısı tanımlama bilgisi gerektirmektedir. Bu araştırma projesinin temel amacı, çizim tabanlı zeki bir insan-bilgisayar arayüzü geliştirmek ve lineer zamanda çalışan bir çizim tanımlama algoritması önermektir. En ünlü çizim tanımlama algoritmalarından ICP yönteminin karesel zaman karmaşıklığında çalıştığı unutulmamalıdır. Bununla beraber, bu çalışmada, ICP yöntemi ile birlikte önerilen algoritmanın da kullanılmasıyla, sisteme herhangi bir yapay sinir ağı eklemeden, sayı çizimlerinin tanımlanmasının başarı oranındaki ciddi artış da gösterilmektedir. Ek olarak, geliştirilen uygulamada, girilen herhangi bir çizgenin sanal gösterimi bir komşuluk matrisi ile sağlanmaktadır. Geliştirilen çizim tabanlı kullanıcı arayüzünün oluşturduğu veriler ile örnek olarak eklenen Kruskal ve Dijkstra çözümleme uygulamaları da kullanılabilmektedir. Sonuç olarak, bu çalışma, bilgisayar ortamında elle çizilmiş şekiller ve sayılar ile çizge oluşturma ve tanımlamaya yeni bir çözüm getirmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SYMBOLS / ABBREVIATIONS

E	Edge
G	Sequence of Nodes
V	Vertex
ASKER	A Sketch Recognizer
ASSIST	A Shrewd Sketch Interpretation and Simulation Tool
DENIM	serge de Nimes
SILK	Sketching Interfaces Like Krazy
GP	Genetic Programming
HMM	Hidden Markov Model
ICP	Iterative Closest Point
MMD	Minimum Mean Distance
NN	Nearest Neighbor
OCR	Optical Character Recognition
PSA	Parallel Sampling Algorithm
SVM	Support Vector Machines

1. INTRODUCTION

Think of a graph drawing on a notebook sheet which is so simple to prepare and understand. Also, it can be changed by erasing some parts of the graph and drawing then again. But actually applying an algorithm to a graph on a paper is very hard and time consuming. At each step, the graph figure should be redrawn or changed partially. If this graph can be drawn on a computer screen using a digital pen, it would become very simple to visualize and modify it. Also, it can be used as an input to an application or an implemented graph algorithm.

As computational manner, visualizing a drawing on a computer monitor by using human-machine interaction tools, i.e. a touch-screen or a pen mouse, is so simple. But the visualized figure is only raw data which consists of 2-dimensional points and has no extra meaning for a machine. At this point, gesture-based information generation gains importance. Gesture means “hand movements” as a word but it has a meaning of recognizable physical movements which forms a meaning for a system as a technical definition. A gesture-based application is a piece of software which gets physical data by a digital pen, a mouse or a touch-screen as input and understands what it means as information.

In this work, a gesture-based interface is developed. Implemented system is an online sketch recognizer for graphs which could understand hand-drawn circles, lines and hand-written digits simultaneously. After recognition of drawings, this application generates an adjacency matrix which specifies the hand-drawn graph figure. Another capability of the application is to provide flexible usage properties like removing or moving selected graph components and changing weight of a particular link at any time.

Developed system is not only a gesture-based interface, but it is also a graph visualization tool. As a good example it could be used efficiently in “Analyses of Algorithms” classes instead of using traditional white-board sketching. By this way, the lecturer could

easily draw and modify a graph, assign weights to the edges. Additionally, user can demonstrate a graph algorithm on the drawn figure as an animation. The generated adjacency matrix can be fed as an input into another application. As an example, animated solvers for Dijkstra and Kruskal algorithms have been embedded in the system. An alternative usage environment of developed application should be tablet pc's. They are new technology computers with sensitive hybrid screens and there are not special applications for them enough. Developed recognizer system should be easily used with a tablet pc as drawing on paper.

This work is also a project. In general, several handwriting recognition systems use ICP (Iterative Closest Point) method as a core algorithm with a neural network. In this research a new method is developed by using ICP and also new core algorithm. This new method increases the digit recognition success ratio incredibly without using a neural network. Another important thing is about the time complexity of the new core recognition algorithm, which has O (n) despite the ICP method has a time complexity of O (n^2).

In Section 2, a general knowledge about graphs and sketch recognition is given. In Section 3, the implemented system is described in detail. Finally, in Section 4, the conclusions on this work are provided.

2. PRELIMINARIES

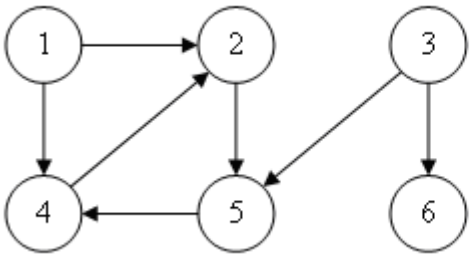
2.1. Graphs

A graph G=<V, E> consists of a set of *vertices* (also known as nodes) V and a set of *edges* (also known as arcs) E. An edge connects two vertices u and v; v is said to be *adjacent* to u. In a *directed* graph, each edge has a sense of direction from u to v and is written as an ordered pair <u,v> or u->v. In an *undirected* graph, an edge has no sense of direction and is written as an unordered pair {u,v} or u<->v. An undirected graph can be represented by a directed graph if every undirected edge {u,v} is represented by two directed edges <u,v> and <v,u> [5, 6].

A *path* in G is a sequence of vertices $<v_0, v_1, v_2, ..., v_n>$ such that $<v_i,v_{i+1}>$ (or $\{v_i,v_{i+1}\}$), for each i from 0 to n-1, is an edge in G. The path is *simple* if no two vertices are identical. The path is a *cycle* if $v_0=v_n$. The path is a *simple cycle* if $v_0=v_n$ and no other two vertices are identical.

2.1.1. Graphs by Adjacency Matrices

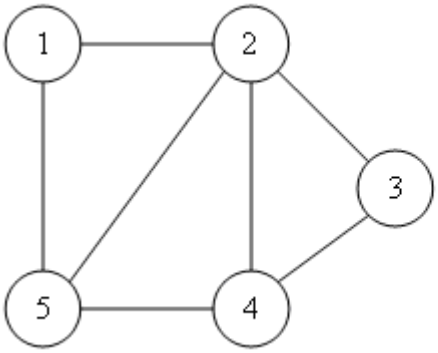
A graph G can be represented by a |V|*|V| *adjacency* matrix A. If G is directed, A_{ij}=true if and only if $<v_i,v_j>$ is in E. There are at most $|V|^2$ edges in E.



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0

Figure 2.1. Adjacency Matrix of a Directed Graph

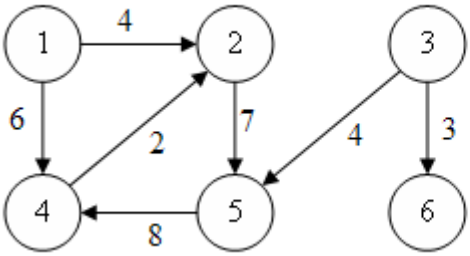
If G is undirected, $A_{ij}=A_{ji}$=true if $\{v_i,v_j\}$ is in E and $A_{ij}=A_{ji}$=false otherwise. In this case there are at most |V|*(|V|+1)/2 edges in E, A is symmetric and space can be saved by storing only the upper triangular part A_{ij} for i>=j.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Figure 2.2. Adjacency Matrix of an Undirected Graph

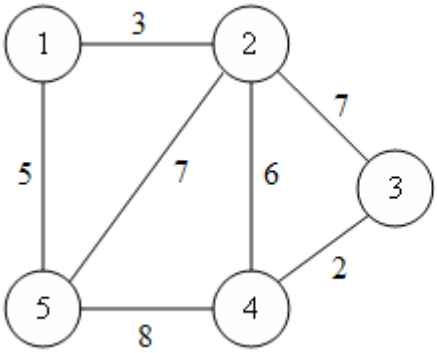
An adjacency matrix is easily implemented as an array. Both directed and undirected graphs may be weighted. A weight is attached to each edge. This may be used to represent the distance between two cities, the flight time, the cost of the fare, the electrical capacity of a cable or some other quantity associated with the edge. The weight is sometimes called the length of the edge, particularly when the graph represents a map of some kind. The weight or length of a path or a cycle is the sum of the weights or lengths of its component edges.



	1	2	3	4	5	6
1	0	4	0	6	0	0
2	0	0	0	0	7	0
3	0	0	0	0	4	3
4	0	2	0	0	0	0
5	0	0	0	8	0	0
6	0	0	0	0	0	0

Figure 2.3. Adjacency Matrix of a Weighted Directed Graph

The adjacency matrix of a weighted graph can be used to store the weights of the edges. If an edge is missing a special value, perhaps a negative value, zero or a large value to represent "infinity", indicates this fact.



	1	2	3	4	5
1	0	3	0	0	5
2	3	0	7	6	7
3	0	7	0	2	0
4	0	6	2	0	8
5	5	7	0	8	0

Figure 2.4. Adjacency Matrix of a Weighted Undirected Graph

It is often the case that if the weights represent distances then the natural distance from vi to itself is zero and the diagonal elements of the matrix are given this value. A weighted adjacency matrix is easily defined in any imperative programming language.

A graph is complete if all possible edges are present. It is dense if most of the possible edges are present. It is sparse if most of them are absent, |E|<<|V|2. Adjacency matrices are space efficient for dense graphs but inefficient for sparse graphs when most of the entries represent missing edges. Adjacency lists use less space for sparse graphs.

2.1.2. Graphs by Adjacency Lists

In a sparse directed graph, |E|<<|V|2. In a sparse undirected graph |E|<<|V|*(|V|-1)/2. Most of the possible edges are missing and space can be saved by storing only those edges that are present, using linked lists.

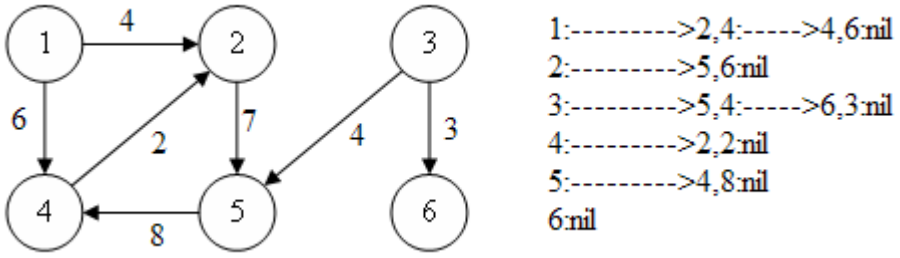


Figure 2.5. Adjacency Lists for Weighted Directed Graph

An edge <vi,vj> is placed in a list associated with vi. The edge is represented by the destination vj and the weight.

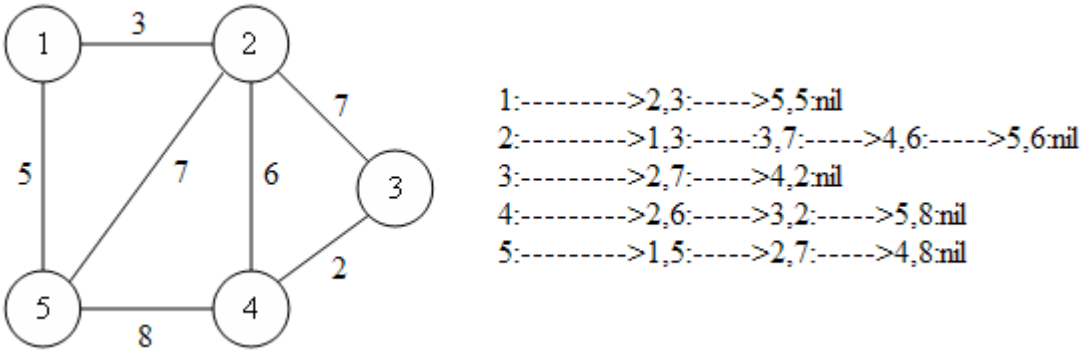


Figure 2.6. Adjacency Lists for Weighted Undirected Graph

Adjacency lists can be defined using records (structs) and pointers. Note that some questions, such as "are vi and vj adjacent in G", take more time to answer using adjacency lists than using an adjacency matrix as the latter gives random access to all possible edges.

2.2. Sketch Recognition

This work is settled on stroke based sketches. Stroke based sketches are drawings made by mouse pressed from beginning to end. Sketches are started with mouse press and finished if mouse released. Mainly there are two types of sketches for this system. First one is shape sketch and other is sketched digits as handwriting.

2.2.1. Sketched Shape Recognition

In sketched shape recognition, collected data -especially in terms of points(x,y)- is processed by using several classification techniques. In many recognition systems, special moment descriptors are also used to represent symbols such as Zernike moment descriptors [7]. The most common tree classification methods are Support Vector Machines (SVM), Minimum Mean Distance (MMD), and Nearest Neighbor (NN).

2.2.1.1. Support Vector Machines (SVM). Support Vector Machines are learning machines that can perform binary classification (pattern recognition) and real valued function approximation (regression estimation) tasks. Support Vector Machines non-linearly map their n-dimensional input space into a high dimensional feature space. In this high dimensional feature space a linear classifier is constructed [8].

Support Vector Machines are based on the concept of decision planes that define decision boundaries [9]. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in the illustration below. In this example, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).

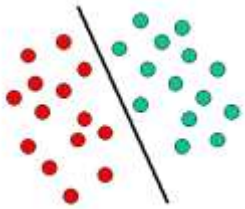


Figure 2.7. Different colored points which can be separated by a line

The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). This situation is depicted in the illustration below. Compared to the previous schematic, it is clear that a full separation of the GREEN and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyper-plane classifiers. Support Vector Machines are particularly suited to handle such tasks.

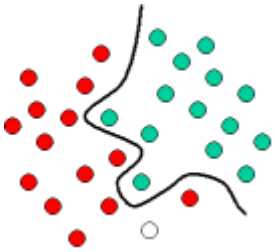


Figure 2.8. Different colored points which can be separated by a curve

The illustration below shows the basic idea behind Support Vector Machines. Here it is seen that the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of

the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the GREEN and the RED objects.

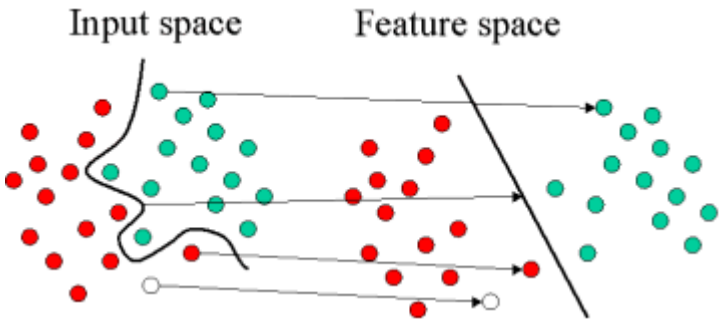


Figure 2.9. Examples of input and feature spaces for a SVM

2.2.1.2. Minimum Mean Distance (MMD). Despite Minimum Mean Distance is so basic; it is very important method to understand similarity of two multi-dimensional shapes. For sketch recognition, generally, drawn shape and all prototypes in the database are compared and a fitting value calculated for each comparison. After all tests the prototype or sample with the *minimum fitting error* is selected as recognized shape. In this procedure, MMD is used to find the minimum distance between each point of a sample and sketched figure. The average of minimum distances of all points in a comparison can be defined as *fitting error*.

2.2.1.3. Nearest Neighbor (NN). The nearest neighbor algorithm in pattern recognition is a method for classifying phenomena based upon observable features. In the algorithm, each feature is assigned a dimension to form a multidimensional feature space. A training set of objects with *a priori* known class are processed by feature extraction and plotted within the multi-dimensional feature space. The offsets in each dimension are referred to as the feature vector. This is the training or learning stage. Because the engine can be retrained to classify various phenomena, pattern recognition is part of machine learning [10].

The testing phase begins with phenomena to be classified (the class not being known *a priori*) and extracts the same set of features. The geometric distance is computed between the new feature vector and each *a priori* feature vector from the training set. The shortest distance thus computed is to the nearest neighbor. The *a priori* class of the nearest neighbor is now assigned to the phenomena to be classified.

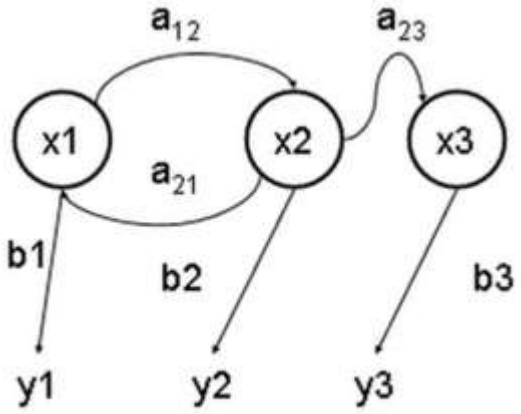
Obviously, this algorithm will be more computationally intensive as the size of the training set grows. Many optimizations have been given over the years; these generally seek to reduce the number of distances actually computed. Some optimizations involve partitioning the feature space, and only computing distances within specific nearby volumes.

Other variations of the algorithm include the *k*-nearest neighbor algorithm where several of the nearest feature vectors are computed, and the classification is made with the highest confidence only if all of the nearest neighbors are of the same class.

Nearest neighbor has some strong consistency results. As the amount of data approaches infinity, nearest neighbor is guaranteed to yield an error rate no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data). *K-nearest* neighbor is guaranteed to approach the Bayes error rate, for some value of *k* (where k increases as a function of the number of data points).

2.2.2. Handwritten Digit Recognition

2.2.2.1. On-Line Digit Recognition Using Hidden Markov Model. As one of the major research directions for on-line handwriting recognition, Hidden Markov Model (HMM) is widely used because of the time sequential nature of online scripts as well as its capability of modeling shape variability in probabilistic terms [11]. A Hidden Markov Model (HMM) is a statistical model where the system being modeled is assumed to be a Markov process with unknown parameters, and the challenge is to determine the hidden parameters from the observable parameters. The extracted model parameters can then be used to perform further analysis [4, 12].



x — hidden states

y — observable outputs

a — transition probabilities

b — output probabilities

Figure 2.10. State transitions in a hidden Markov model (example)

The preceding diagram emphasizes the state transitions of a HMM. It is also useful to explicitly represent the evolution of the model over time, with the states at different times t_1 and t_2 represented by different variables, $x(t_1)$ and $x(t_2)$.

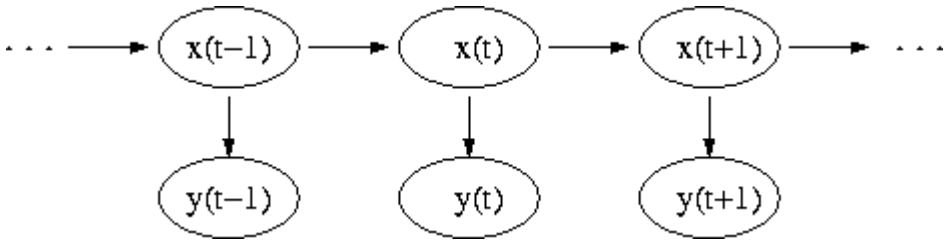


Figure 2.11. Evolution of a Markov model

In this diagram, it is understood that the time slices ($x(t)$, $y(t)$) extend to previous and following times as needed. Typically the earliest slice is at time t=0 or time t=1.

2.2.2.2. On-Line Digit Recognition using Off-Line Features. One of the fundamental aspects of handwritten character recognition methodologies is the manner in which data is collected. The data acquisition process can be on-line or off-line. On-line handwritten data is collected using a digitizer or an instrumented pen to capture the pen-tip position(x_t, y_t) as a function of time. Contrary to on-line, off-line handwritten data is collected using a scanner resulting in the generation of the signal as an image I(x, y). Both areas of handwritten character recognition have been in existence for more than three decades. This technique is based on using off-line features extracted from the on-line data. By this manner also Genetic Programming (GP) classifiers can provide good recognition results [13].

There are two main considerations for the use of offline features in online recognition: a) Hierarchical feature space has proved to be very effective for digit classification and pair wise discrimination of confusing allograph. b) Genetic Programming based classification has helped identify the important features for classifier development. Specifically, GP has several advantages over classical feature selection techniques when used for off-line handwritten digit classification using hierarchical features.

2.2.2.3. Online Digit Recognition using Alignment to Prototypes. Nearest neighbor classifiers are simple to implement, yet they can model complex non-parametric distributions, and provide state-of-the-art recognition accuracy in OCR databases.
At the same time, they may be too slow for practical character recognition, especially when they rely on similarity measures that require computationally expensive pair wise alignments between characters. But there are several efficient methods for computing an approximate similarity score between two characters based on their exact alignment to a small number of prototypes. [14]

2.2.2.4. Iterative Closest Point Method (ICP). The ICP (Iterative Closest Point) algorithm is widely used for geometric alignment of two-dimensional and three-dimensional models and also used for matching and recognition of 2D shapes. There are many variants of ICP, affecting all phases of the algorithm from the selection and matching of points to the minimization strategy [2]. But the general form of ICP can be defined easily with the following pseudo-code.

***ICP*_algo (shape1, shape2)**

```
Set initial value of total_minimum_distance to zero
From initial point to last point of shape1
    From initial point to last point of shape2
        Calculate distance between current points of shape1 and shape2
    End
    Find and save the minimum distance as minimum_distance
    Add the minimum_distance to total_minimum_distance variable
End
average_min_distance  = the total_minimum_distance / the number of points in shape1
```

Figure 2.12. Pseudo-code of ICP

By using this algorithm, average minimum distance of drawn digit for every digit sample is calculated and the minimum of all digit samples is selected as recognized digit.

2.3. Related Work

In this section, related work in gesture-based interfaces, which use sketch editing and sketch recognition, is briefly reviewed. Previous research in sketch understanding has generally chosen one of two paths: online interfaces that require an instrumented drawing surface that captures stroke information; or off-line interfaces that allow the user to sketch using pen-and-paper. Among all these systems, we will focus on online recognition systems because developed gesture-based interface is also an online system [1].

SILK (Landay & Myers 1995) is one of the earliest tools in this category. It recognizes sketches of user interfaces [3]. The sketch could then be converted into an interactive demo of the sketched GUI.

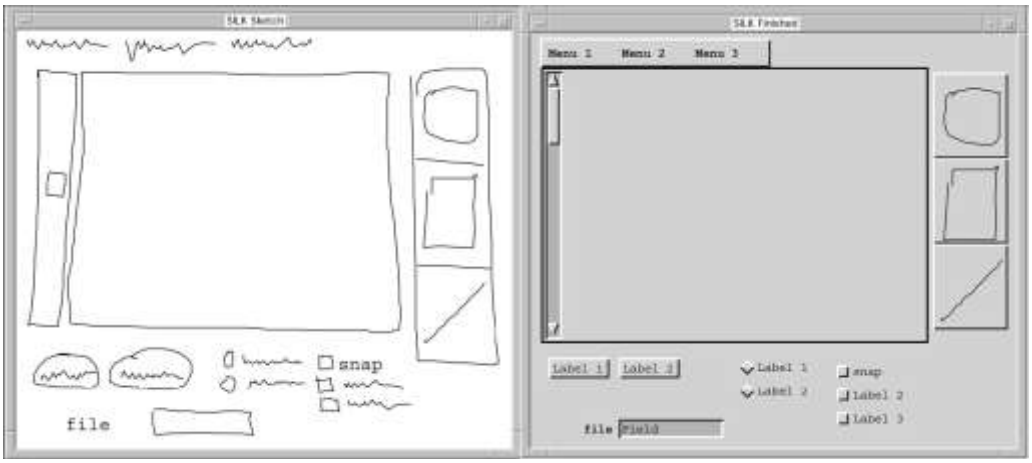


Figure 2.13. Screenshots of SILK

Jorge and Fonseca presented a novel approach to recognizing geometric shapes interactively (Jorge & Fonseca 1999), using fuzzy logic and decision trees to recognize multi-stroke sketches of geometric shapes [15].

ASSIST (Alvarado 2000) is an online tool which allows a user to sketch simple mechanical systems and see simulations of her drawings in a two-dimensional kinematic simulator [16].

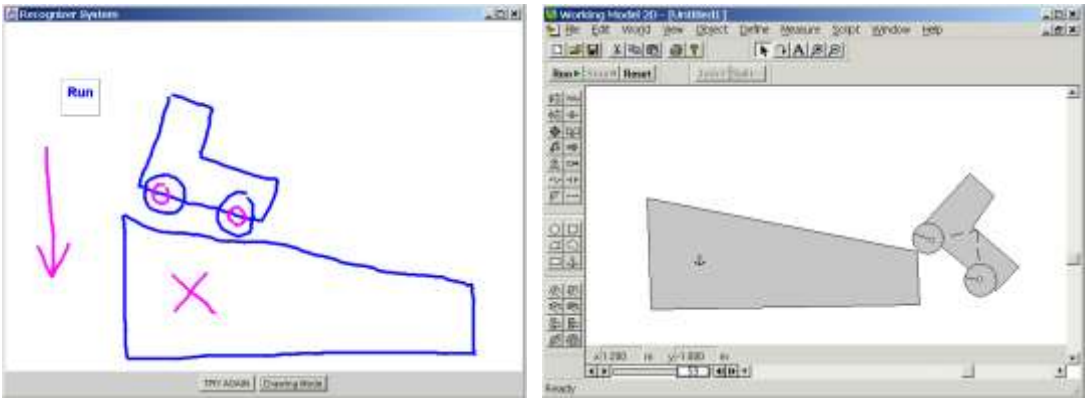


Figure 2.14. Screenshots of ASSIST

Denim (Lin et al. 2000) is an online system for informal sketching of web site designs, in which the user can sketch boxes to represent web pages and lines to indicate links between the pages. Different kinds of links are recognized based on their context, rather than on their shape. Denim uses a unistroke recognizer (Rubine 1991), so each symbol must be sketched with a single stroke [17].

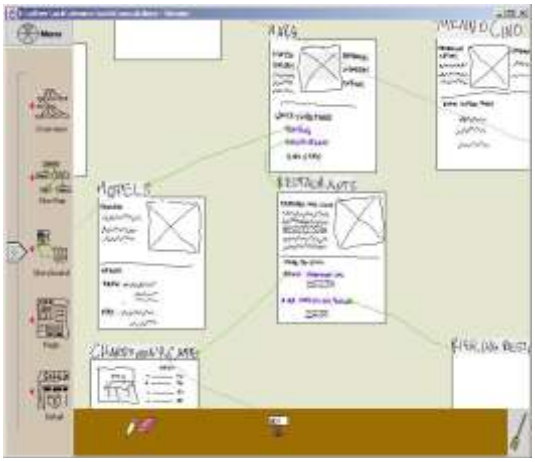


Figure 2.15. A screenshot of Denim

Sezgin et al. (Sezgin & Davis 2001) have created an online system that recognizes geometric primitives. The system uses a three-phase technique (approximation, beautification, basic recognition) to transform sketches into sharpened diagrams. Its recognition process uses shape and curvature of strokes, among other information [18].

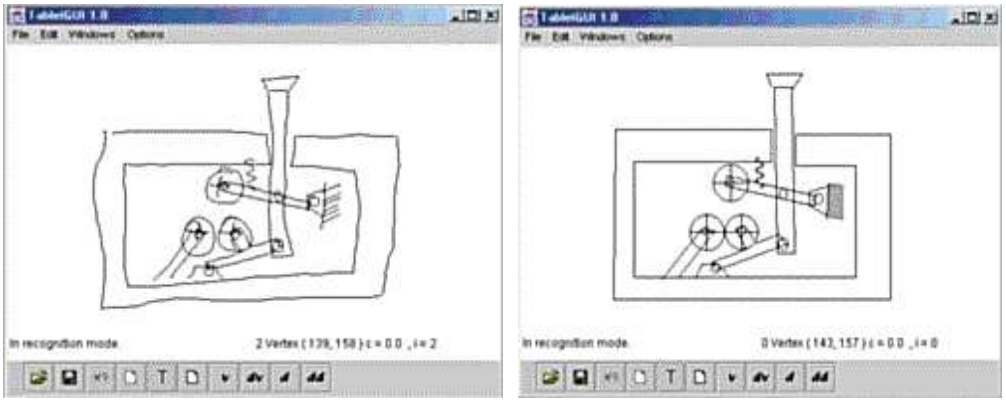


Figure 2.16. Screenshots from "Early Processing in Support of Sketch Understanding"

Tahuti (Hammond & Davis 2002) is another online system for UML diagram recognition, which uses both geometric and contextual constraints to recognize UML symbols. Tahuti is non-modal: users can interleave editing and drawing with no mode switching, which requires editing gestures to be distinguishable from sketched symbols [19].

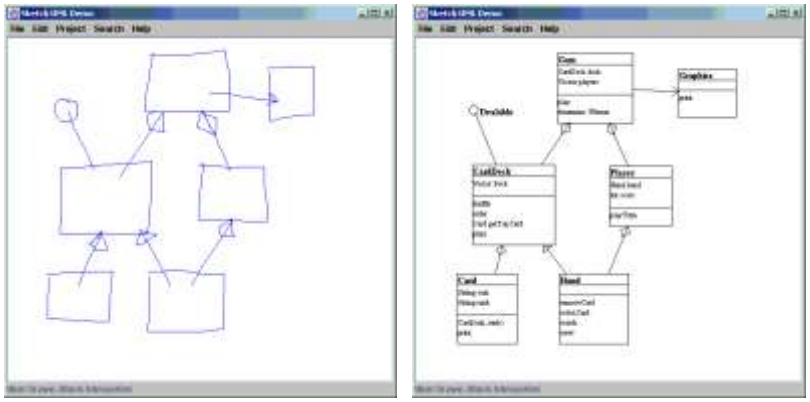


Figure 2.17. Screenshots of Tahuti

Lank et al. (Lank, Thorley, & Chen 2000) used online stroke information to recognize UML diagrams. UML symbols were recognized by heuristics such as stroke count and the ratio of stroke length to bounding-box perimeter. The Lank system tries to distinguish between UML symbols and character data. Its user interface allows the user to correct the system at each stage of understanding: acquisition, segmentation, and recognition [20].

The Natural Log (Nicholas, Matsakis 1999) is a tool recognizes many common mathematical forms such as digits, lowercase Latin letters, certain Greek letters, relational and binary operators, fractions, summations, square roots, and many accents. By using this tool Latex and MathML Expressions are generated [21].

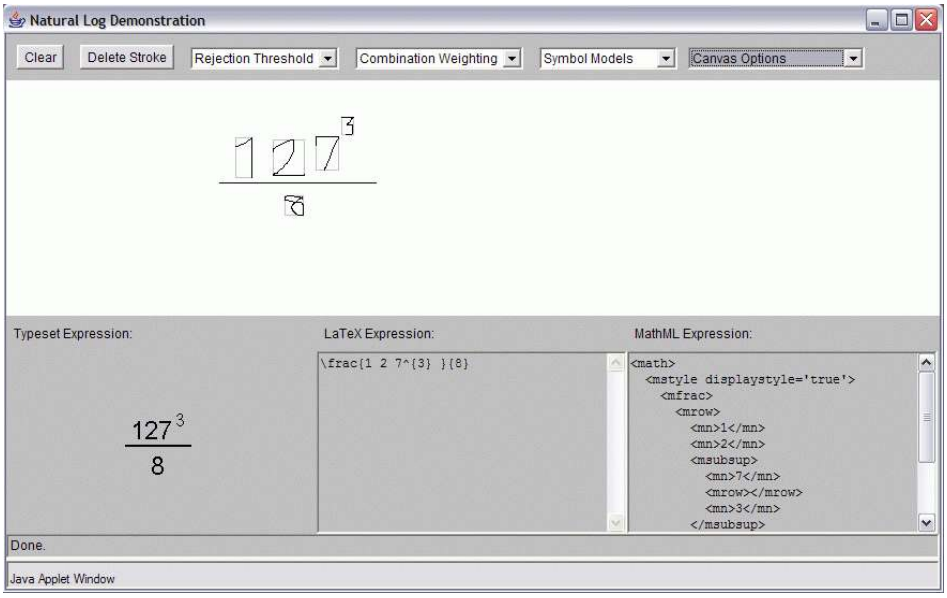


Figure 2.18. A screenshot of Natural Log

The Natural Log has some fundamental limitations. The primary limitation is that the character models have been created from examples of Nicholas' handwriting, collected on a digitizing tablet. This means that other users' handwriting will often be misinterpreted. In addition, the system currently does not support integral or subscript notation at all.

As it is seen by all these work, sketch based interface is a common research area but there is not any sketch based interface for graphs. The developed application “ASKER” is a natural sketch recognition environment for graphs (directed, undirected, weighted, unweighted). In this system, the user can sketch graphs on a tablet or whiteboard in the same way they would on paper, but the graphs would then be recognized by the computer and the related adjacency matrices be generated.

3. A SKETCH RECOGNIZER FOR GRAPHS

This work combines sketch recognition, hand-written digit recognition and graph theory knowledge and forms an intelligent interface. Implemented application is developed in Java SDK 1.5.06 environment, using Eclipse as an IDE. This project also called as ASKER which means "**A Ske**tch **R**ecognizer".

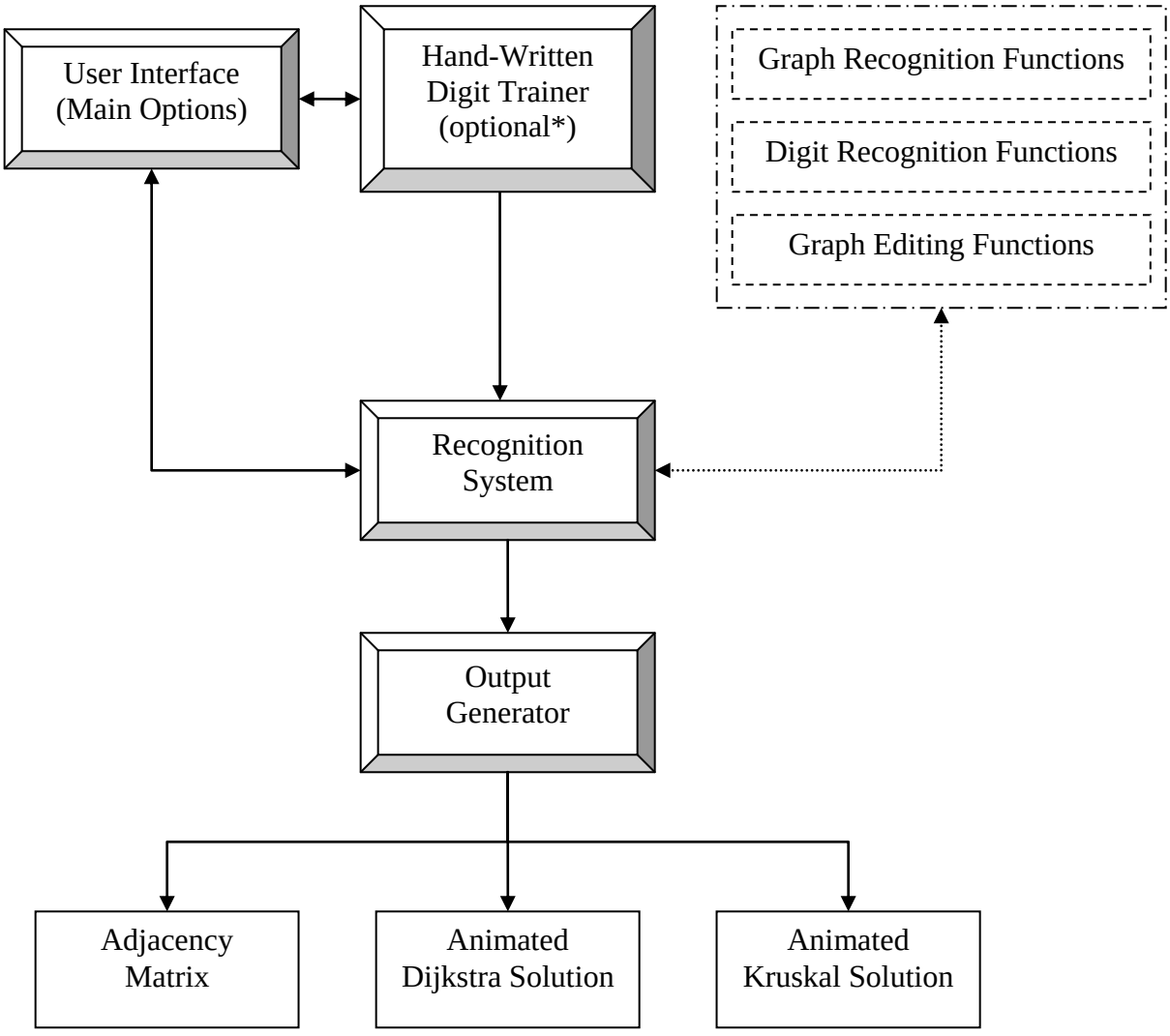


Figure 3.1. Skeleton Flowchart of ASKER

The preceding figure shows the main skeleton of ASKER. As it is seen, it consists of 4 main parts, which are User Interface (Main Options), Hand-Written Digit Trainer, Recognition System and Output Generator. In this section all these four sub-systems will be described in detail.

3.1. User Interface

The user interface of ASKER provides to control the main options of the system. It is the starting step of the whole application. Before starting to sketch a graph, main menu is needed to be adjusted for required usage details.

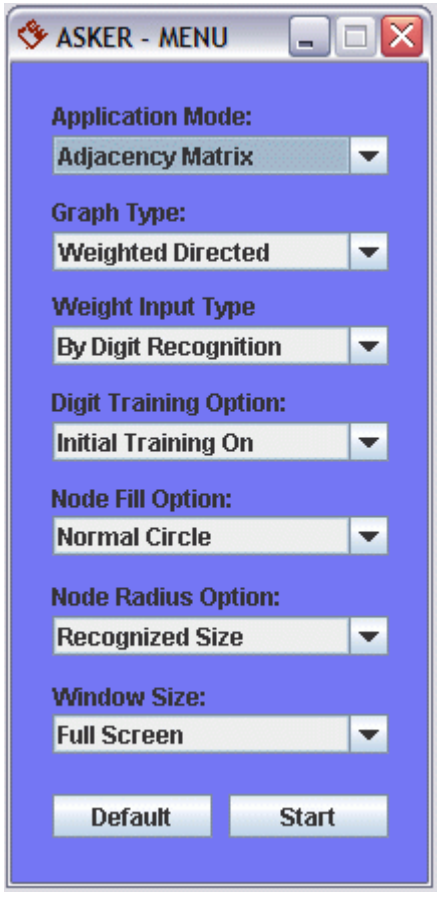


Figure 3.2. User Interface: ASKER - MENU

Application Mode option provides to select required output type. There are 3 options can be adjusted such as Adjacency Matrix, Dijkstra Demo and Kruskal Demo. Adjacency matrix option generates an Adjacency matrix of the sketched graph, in addition Dijkstra and Kruskal Demo options provide animated solutions of related algorithm.

Graph Type option is only used for Adjacency Matrix mode, if any demo mode is selected this options will be disabled. By using this option, required graph type can be adjusted such as Weighted Directed, Unweighted Directed, Weighted Undirected and Unweighted Undirected. Properties of the selected graph type are adjusted automatically by the system.

Weight Input Type option controls the weight input method you want to use for weighted graphs such as digit recognition and numerator. When an unweighted graph type is selected, this option will be disabled.

Digit Training Option can be used to enable or disable the initial handwritten digit training for a new user. Because of the recognizer system doesn't have a neural network; initial digit training is an essential function for ASKER to learn user's handwritten digits. If a new user doesn't train the digit recognizer, recognition success ratio may be decreased. This option is automatically disabled if weight input by digit recognition won't be used in graph sketching.

Node Fill Option provides to fill inside of nodes with default color or to visualize nodes as basic circles.

Node Radius Option can be used to adjust graph nodes radius such as recognized size and fixed size. Recognized size will be calculated by user's node sketch, but fixed size automatically sets the radius of all nodes to 40 pixels.

Window Size option controls the application window size. It can be adjusted manually such as 640*800, 800*600, 1024*768 and 1280*1024 or directly as Full Screen.

Default button sets all options to initial values. And last of all *Start* button starts application by configuring the selected options.

3.2. Handwritten Digit Trainer

Handwritten Digit Trainer provides ASKER to learn user's handwritten digits. This module shows which digit user would sketch such as 0, 1, 2, ..., 9 on the screen.

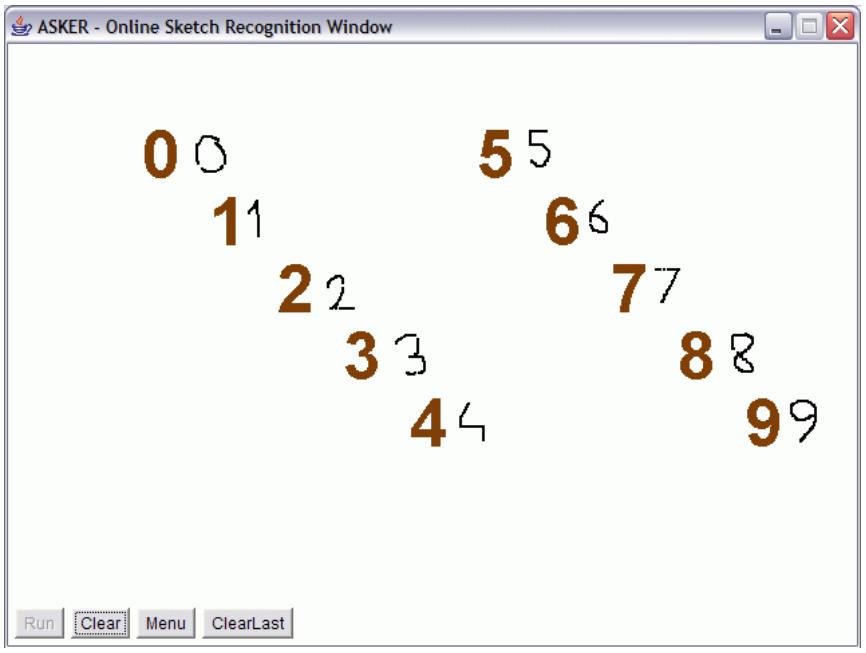


Figure 3.3. Digit Training Screen of ASKER

After user sketched each of the cited digits digit trainer calculates bounding box and normalize all points' coordinates of that sketched digit sample. Whole calculated prototype details are saved on a file; user_digits.txt. This text file would be used as user digit samples database in the process of Recognition System. Following figure shows the main format of the "user_digits.txt" file for a sample digit.

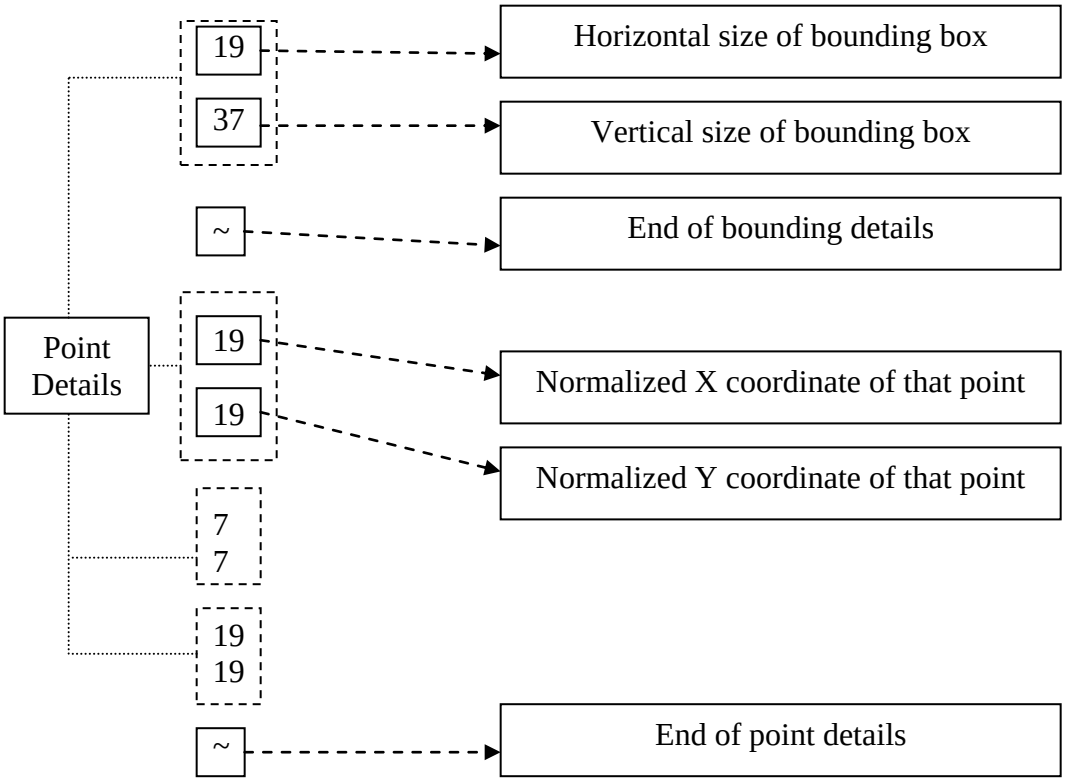


Figure 3.4. Format of user_digits.txt for one sample digit.

"user_digit.txt" file consists of digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 by the figured format. At the end of the file an extra tilde "~" is added which means that all sample digits are finished.

3.2.1. Normalization

In training module all sketched digit samples are processed and normalized. In normalization, the point with minimum x coordinate of that sample is found and its x value subtracted from all points' x values. By the same manner y normalization is also done. The following figure shows the pseudo-code of the normalization process.

---

*Normalization*_algo(sample_digit[points])
Find the point with minimum x value and save it as *x_min*.
Find the point with minimum y value and save it as *y_min*.
From initial point to end point of sketched digit sample
    Current point's x = Current point's x – *x_min*
    Current point's y = Current point's y – *y_min*
End loop
Return normalized sample_digits

Figure 3.5. Normalization Pseudo-code

User can clear and change his/her last sketch or all sketched digits can be cleared, each can be drawn again, in digit training. *ClearLas*t button clears the last sketch of the user and *Clear* button clear of sketched digits as stated above.

Figure 3.6. Confirmation box which asks if a new training is required

Also at the end of the training, a confirmation box appears and asks if a new training is required for further recognition success ratio. After user finished training step, ASKER starts recognition system automatically.

3.3. Recognition System

Recognition System of ASKER combines all of the recognition and editing functions. It provides user a natural environment to sketch and edit all details of any graph. This sub-system of ASKER will be described in three parts, which are graph recognition, handwritten digit recognition and graph editing functions.

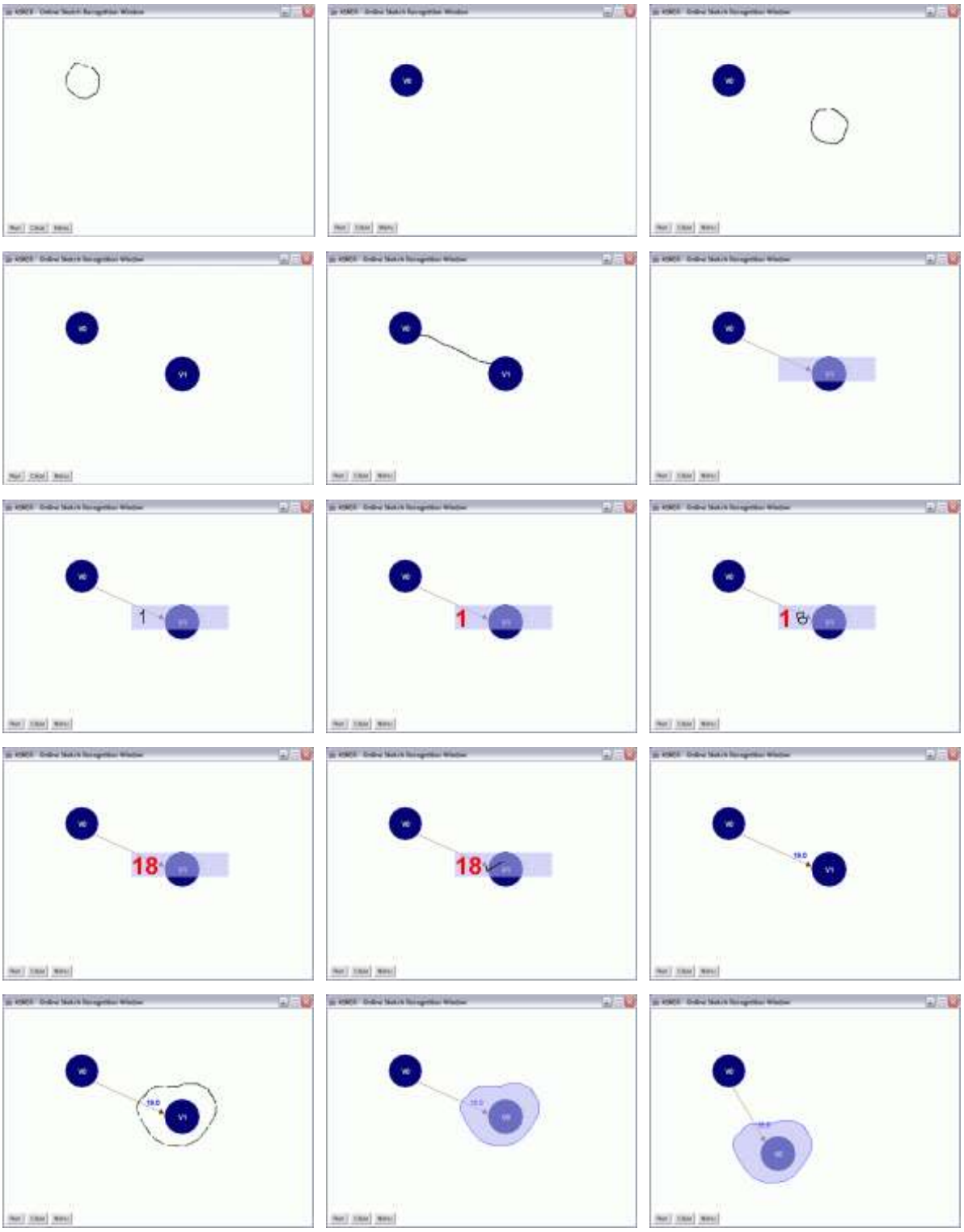


Figure 3.7. An example usage of recognition and editing properties of ASKER

3.3.1. Graph Recognition Functions

These functions initially test and decide if the drawn figure is a edge or a node. For this decision drawn points are collected in a linked-list and by using these points' coordinates; one line, one circle equation are formed.

3.3.1.1. Determination of Line Equations. First and last point coordinates of the stroke are used to determine a line equation. It is determined by using *the equation of a line with two known points.*

There are three different situations according to the coordinates of points A= (x_1 , y_1) and B=(x_2 , y_2).

In the first case, $x_1 \neq x_2$ and $y_1 \neq y_2$.

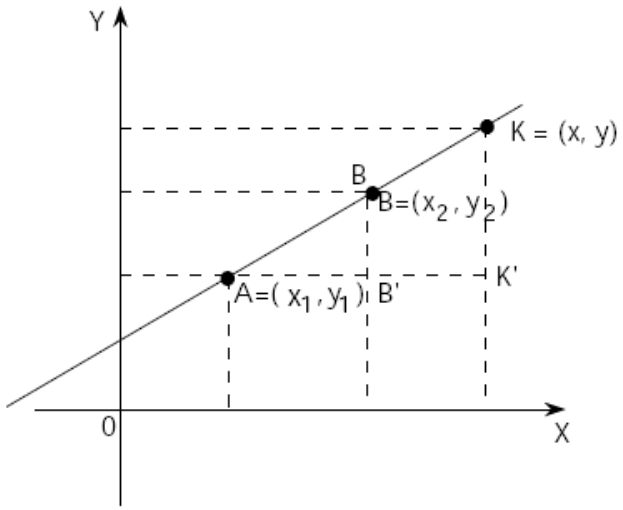


Figure 3.8. Line for first situation

As it is seen in Figure 3.8; by using the similarity of triangles ABB' and AKK', $\frac{|KK'|}{|K'A|} = \frac{|BB'|}{|B'A|}$ so, $\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$ is derived.

In the second case, $x_1 = x_2$ and $y_1 \neq y_2$.

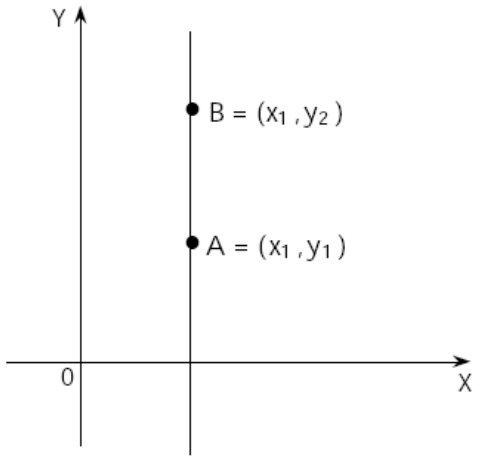


Figure 3.9. Line for second situation

As it is seen in Figure 3.9, the line intersects these two points is parallel to Y-axis. Apsises of all points on this line is equal to x_1. So the equation of this line is x= x_1.

In the third case, $x_1 \neq x_2$ and $y_1 = y_2$. The equation which is derived in the first situation is substituted by $y_1 = y_2$. By this way; y= y_1 is derived.

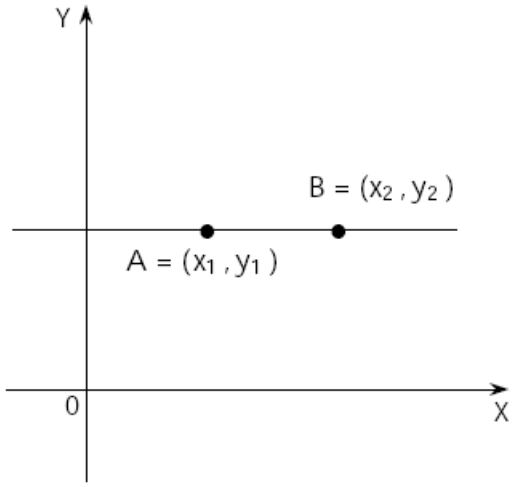


Figure 3.10. Line for third situation

3.3.1.2. Determination of Circle Equations. Equation of a circle is $(x - h)^2 + (y - k)^2 = r^2$ where h and k are the x and y coordinates of the center of the circle and r is the radius.

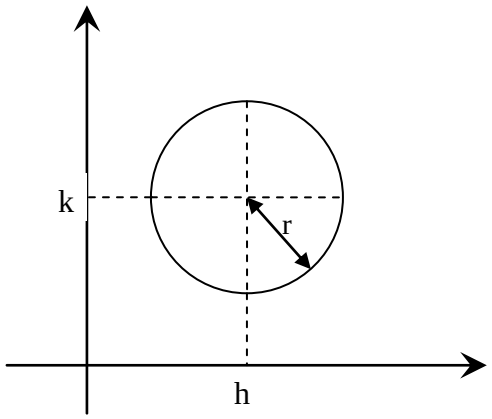


Figure 3.11. A circle

Collected points of the sketched shape are searched for finding minimum x, minimum y, maximum x and maximum y coordinates of all points. They are saved as x_min, x_max, y_min and y_max respectively. k, h and r values which are required for circle equation determination are derived by the following equations.

$$k = \frac{\text{y\_min} + \text{y\_max}}{2}$$

$$h = \frac{x\_\min + x\_\max}{2}$$

$$r = \frac{(x\_\max - x\_\min) + (y\_\max - y\min)}{2}$$

After the determination of the line and circle equations, collected points are tested by using these equations.

3.3.1.3. Line Test. Line test returns the *average_fitting_error* of the test. In this step, points which are collected from sketch of user are tested by the determined line equation with the LineTest algorithm which is shown in following pseudo-code.

***LineTest*_algo (sketched_points, line_equation)**

Set initial value of *total_fitting_error* to zero

line_equation => ax + by + c = 0

From initial point to last point of **sketched_points**

px = x-coordinate value of current point

py = y-coordinate value of current point

$$\text{distance} = \left| \frac{(a \times px) + (b \times py) + c}{\sqrt{a^2 + b^2}} \right|$$

total_fitting_error = total_fitting_error + distance

End loop

$$\textit{average\_fitting\_error} = \frac{\text{total\_fitting\_error}}{\text{total\_number\_of\_points}}$$

return *average_fitting_error*

Figure 3.12. Pseudo-code of LineTest

3.3.1.4. Circle Test. Circle test returns the *average_fitting_error* of the test. In this step, collected points are tested by the determined circle equation with the CircleTest algorithm which is shown in following pseudo-code.

***CircleTest*_algo (sketched_points, circle_equation)**

Set initial value of *total_fitting_error* to zero

circle_equation => $(x - h)^2 + (y - k)^2 = r^2$

From initial point to last point of **sketched_points**

px = x-coordinate value of current point

py = y-coordinate value of current point

distance = $\left|\sqrt{(h - px)^2 + (k - py)^2} - r\right|$

total_fitting_error = total_fitting_error + distance

End loop

average_fitting_error =

return *average_fitting_error*

Figure 3.13. Pseudo-code of CircleTest

Recognition system compares the results of CircleTest and LineTest operations and the shape with the minimum fitting error is selected as recognized shape. If the recognized shape is a circle, it is controlled by several methods to learn what it means as a shape. By analytic tests, ASKER determines if it is a node or it means an area selection on the previous sketched part of the graph. Also these tests controls if the sketched shape is fully inside of the drawing area, or not. If the recognized shape is a line, by the similar controls, sketch meaning is determined. It can be either a new edge or a deletion command. All these processes can be summarized by the following flowchart.

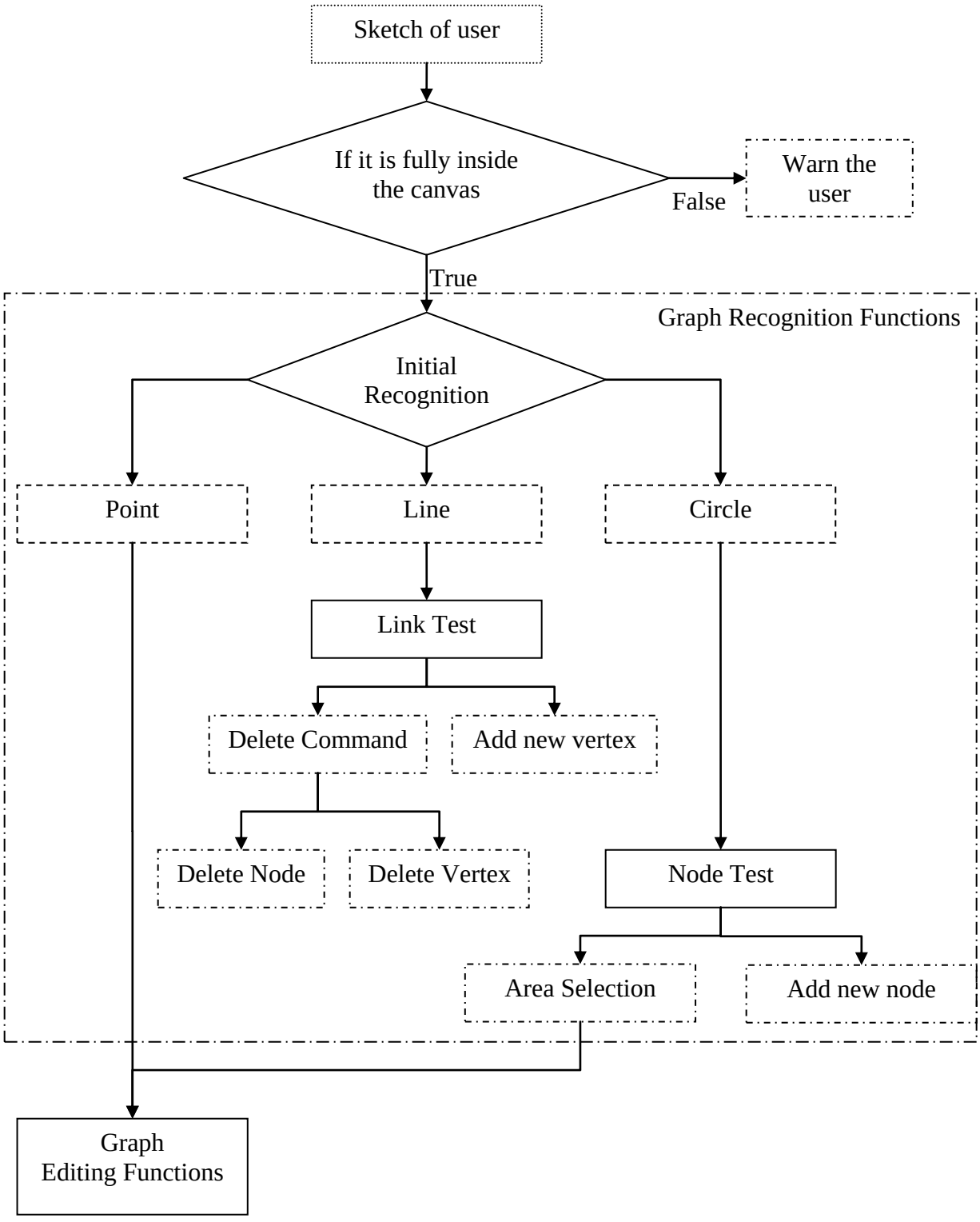


Figure 3.14. Flowchart for Graph Recognition Processes

3.3.1.5. Link Test. It is a complex control mechanism which determines the meaning of the sketched line. There are several controls inside Link Test. The general procedure of this step is described by the following pseudo-code.

***LinkTest*_algo (line_equation, first_point_of_the_line, last_point_of_the_line)**

```
If this line's start and end points are in different nodes
    If the graph is directed
        If there is no edge between these two nodes in the same direction
            Add new edge
        Else warn user
    Else if the graph is undirected
        If there is no edge between these two nodes
            Add new edge
        Else warn user
Else if this line's start and end points are outside of all nodes
    If this line intersects any edges
        Delete those edges
    If this line intersects any nodes
        Delete those nodes
Else do nothing
```

Figure 3.15 Pseudo-code of LinkTest

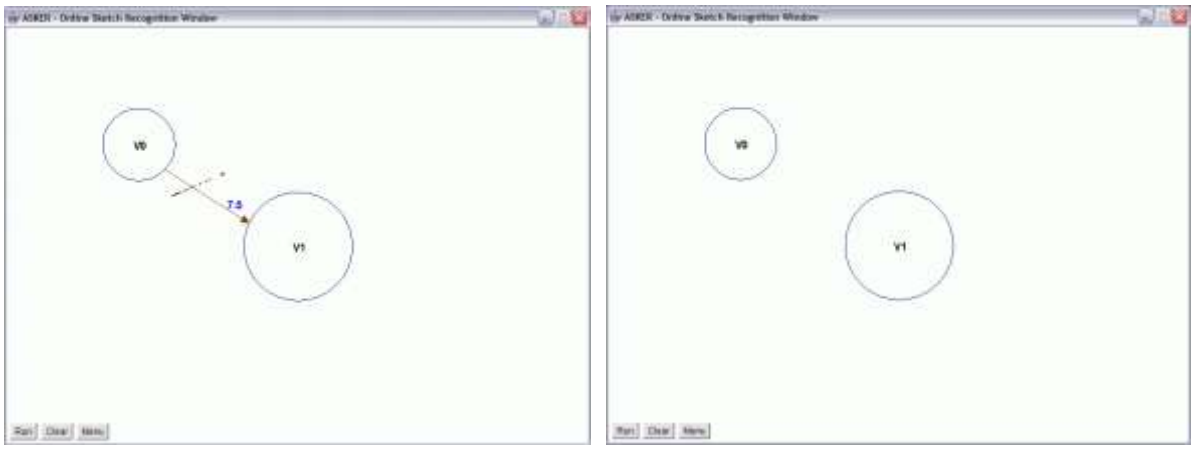


Figure 3.16. Edge deletion using a line

3.3.1.6. Node Test. In this step sketched circle' meaning is determined. The main flow of this mechanism is shown in the following pseudo-code of the NodeTest.

***NodeTest*_algo (circle_equation, sketched_shape_points)**

Determine *circle_bounding_box*

Determine *sketched_shape_bounding_box*

If *sketched_shape_bounding_box* covers at least one node's bounding box

    Sketched area is selected

    Graph editing move function is called

Else if *circle_bounding_box* doesn't intersect any node's bounding box

    If *circle_bounding_box* isn't inside of any node's bounding box

        Add new node

    Else warn user

Else if *circle_bounding_box* intersects at least one node's bounding box

    Warn user

Else do nothing

Figure 3.17. Pseudo-code of NodeTest

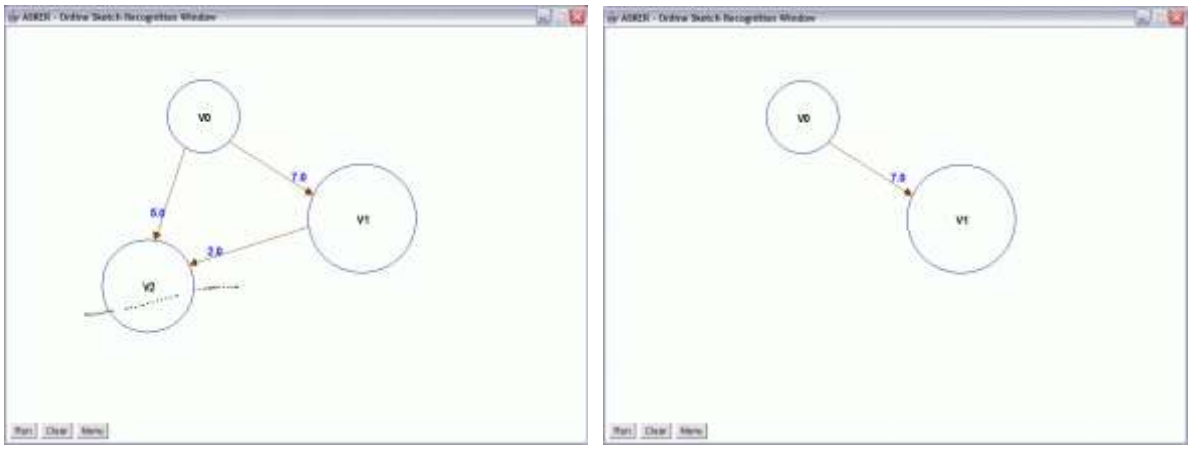


Figure 3.18. Node deletion using a line

In addition if the recognized shape is a node or edge which is added to the related list, else related editing functions are called. In ASKER, nodes and edges are stored in different linked-lists.

3.3.2. Handwritten Digit Recognition Functions

Handwritten digit recognition is a complex process. ASKER recognizes the sketched digits by using a new method which is proposed in this research project. In this step, check sign and deletion sign [Figure 3.21. and Figure 3.22.] are also recognized by the same way. The main flow of this method is shown by the following flowchart.

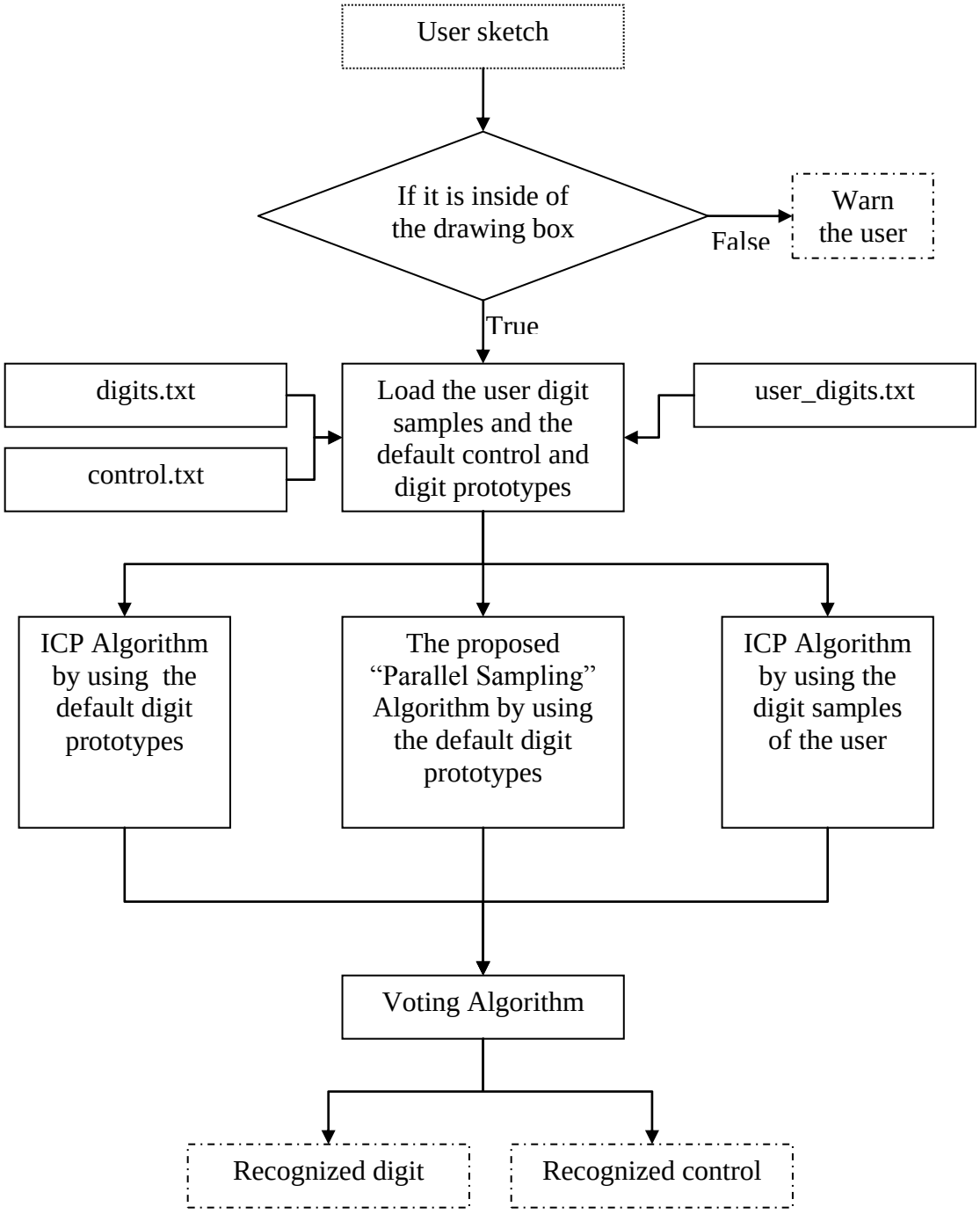


Figure 3.19. Flowchart of the proposed handwritten digit and sign recognition method

As it is seen in the preceding flowchart, proposed method uses two types of prototypes such as user digit samples and default prototypes. All these information is firstly normalized [Figure 3.5.] and scaled then processed by the proposed *Parallel Sampling Algorithm (PSA)* and *ICP Algorithm* which is described before [Figure 2.12.]. In this section, *Parallel Sampling Algorithm* and *Voting Algorithm* will be described in detail.

For digit and sign recognition, required prototypes are loaded from digits.txt, control.txt and user_digits.txt. user_digits.txt's structure details are explained before. Files; digits.txt and control.txt are formed by the same manner with the user_digits.txt [Figure 3.4.].

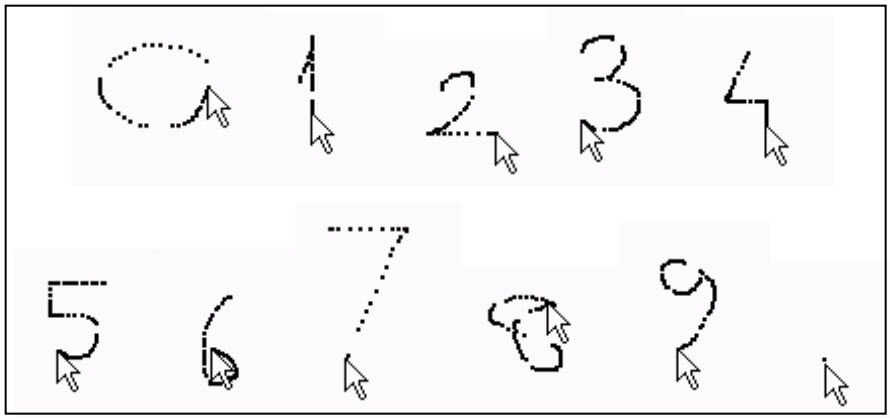


Figure 3.20. Examples of sketches; 0,1,2,3,4,5,6,7,8,9 and period respectively

Figure 3.21. Deletion sign is used clear the last recognized digit

Figure 3.22. Check sign is used to complete the weight related digit recognition

3.3.2.1. Scaling. Scaling is a prerequisite step for PSA and ICP algorithm as normalization is. After normalization is done to sketch points, it is time for scaling which is described by the following pseudo-code.

***Scaling*_algo (prototype_digit_points, normalized_sketched_points)**

Determine *prototype_bounding_box*

Determine *sketch_bounding_box*

$$\text{x\_scale} = \frac{\text{x-axis size of prototype\_bounding\_box}}{\text{x-axis size of sketch\_bounding\_box}}$$

$$\text{y\_scale} = \frac{\text{y-axis size of prototype\_bounding\_box}}{\text{y-axis size of sketch\_bounding\_box}}$$

From first point to last point of *normalized_sketched_points*

scaled_current_point_x = x_scale × *current_normalized_sketched_point_x*

scaled_current_point_y = y_scale × *current_normalized_sketched_point_y*

End loop

Return *scaled_points*

Figure 3.23. Pseudo-code of Scaling Algorithm

3.3.2.2. Parallel Sampling Algorithm (PSA). Proposed algorithm, PSA, gets the prototype digit and sketch as inputs. After that PSA calculates average fitting error of these inputs with a parallel sampling method by using their points' sequence. It is described in detail by the following pseudo-code.

***PS_algo*(prototype_digit[points], scaled&normalized_sketch[points])**

Set *total_distance* to 0

s_num = number of sketched points

p_num = number of prototype's points

If $s\_num \leq p\_num$

$counter = s\_num$

s_sampling_factor = 1

$$\text{p\_sampling\_factor} = \left\lfloor \frac{\text{s\_num}}{\text{p\_num}} \right\rfloor$$

Else

$counter = p\_num$

$$\text{s\_sampling\_factor} = \left\lfloor \frac{\text{p\_num}}{\text{s\_num}} \right\rfloor$$

$p\_sampling\_factor = 1$

For $i = 0$ to *counter*

$current\_ip = i \times p\_sampling\_factor$

$current\_is = i \times s\_sampling\_factor$

Find the distance between prototype_digit[ip] and scaled&normalized_sketch[is]

Save it as *current_distance*

$total\_distance = total\_distance + current\_distance$

End loop

$$\text{average\_fitting\_error} = \frac{\text{total\_distance}}{\text{counter}}$$

Figure 3.24. Pseudo-code of PSA

PSA is executed for all of the digit prototypes. At the end the prototype with the minimum average fitting error is selected as recognized digit.

3.3.2.3. Voting Algorithm. After all recognition tests are finished, there are three results by using ICP with user digit samples, ICP and PSA with default prototypes. With a simple mechanism all these 3 recognition results are voted and the sketched digit is recognized. Voting algorithm can be summarized by the following pseudo-code.

***Voting*_algo (recognition_1, recognition_2, recognition_3)**

If all recognition results are same

    Recognized digit is that result

Else if two of the results are same

    Recognized digit is that result

Else all results are different

    Recognized digit is the input recognition which has the minimum *average_fitting_error*

Return recognized digit

Figure 3.25. Pseudo-code of Voting Algorithm

3.3.3. Graph Editing Functions

There are two types of graph editing function in ASKER. First one is the “moving the selected area functions” and the second is the “changing the entered weight functions”.

3.3.3.1. Moving the selected area functions. By these functions, the selected area on the graph can be moved dynamically. Changes on the edges related to the move process is automatically done. Moving the selected area functions provide keeping the selected moving nodes from intersecting with other nodes. All these processes can be described by the following pseudo-code.

***Moving*_algo (selected_area, current_stroke)**

Determine the covered nodes by selection

If the current_stroke's first point is inside the selected_area

    From the first point to last point of the curren_stroke

        Mode selected nodes

        Reformulate the lines which are related with selected nodes

        Repaint the screen

    End loop

Else cancel the selection

Figure 3.26. Pseudo-code of the Moving Algorithm

3.3.3.2. Weight editing functions. These functions are used to change the weight of an edge in any time. These procedures can be described by the following pseudo-code.

***WeightEdit*_algo (current_stroke)**

If the current_stroke is a point

    If this point is inside of the bounding box of any string which represents a weight

        Weight enterance box is opened and if necessary,

        handwritten digit recognition functions are called

    Else do nothing

Figure 3.27. Pseudo-code of the Weight Editing Algorithm

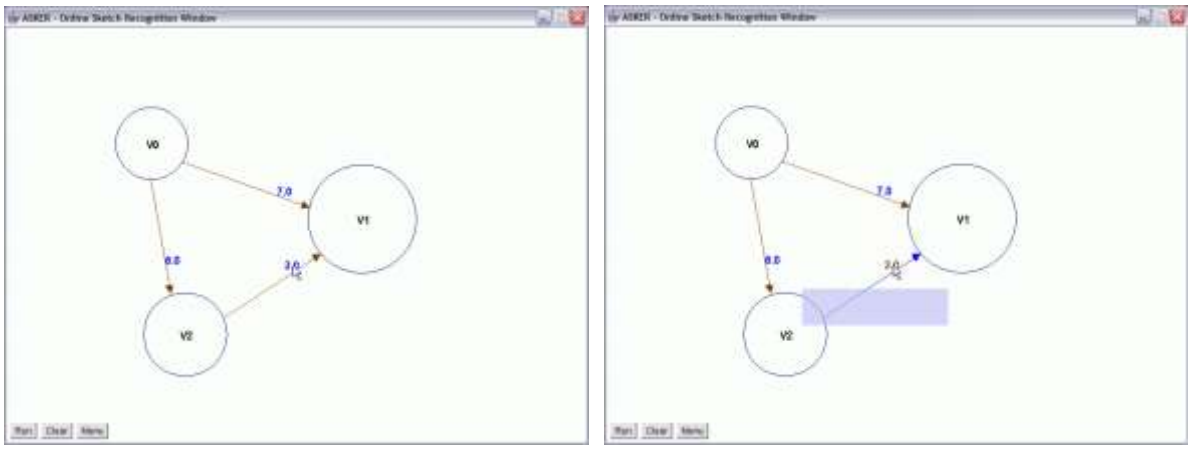


Figure 3.28. Weight Editing

3.4. Output Generator

ASKER stores all recognized nodes and edges in Node and Edge linked-lists respectively. Also the coordinates and the sizes of the recognized graph elements are stored as well. When the graph design is completed and the run button which is on the right bottom of the canvas, is pressed and adjacency matrix is formed by using stored weights in the Edge linked-list. For weighted graphs adjacency matrix represents the weights of the edges in addition zeros shows that there is no edge for that nodes. If the system is adjusted as unweighted, generated adjacency matrix consists of zeros and ones. Ones show that there is a link between those nodes; in addition zero shows that there is no related edge.

If application mode of ASKER is adjusted as Dijkstra or Kruskal Demo, the related demo application starts after the generation of the adjacency matrix. All these output types of ASKER can be seen by the following figures.

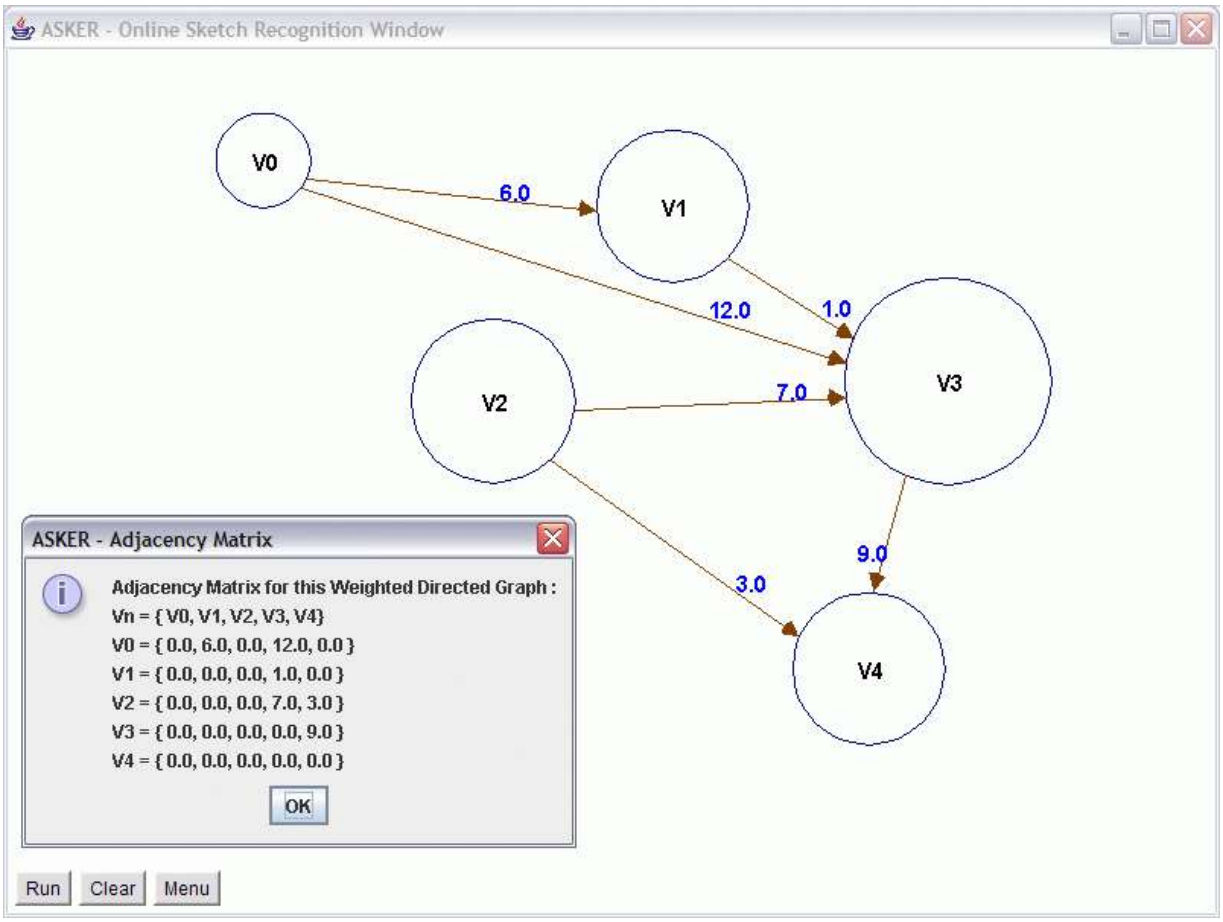


Figure 3.29. Generated Adjacency Matrix

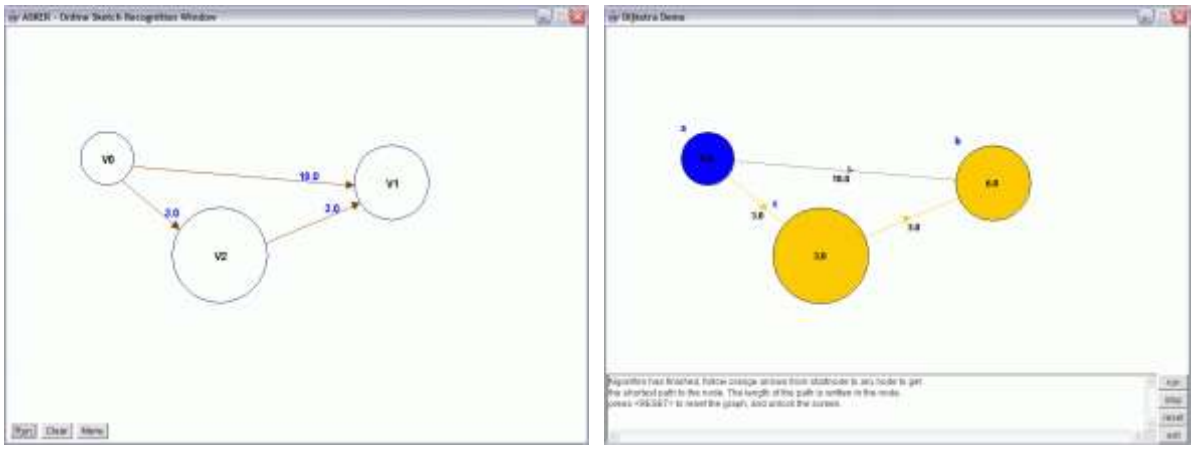


Figure 3.30. Dijkstra Demo of sketched graph

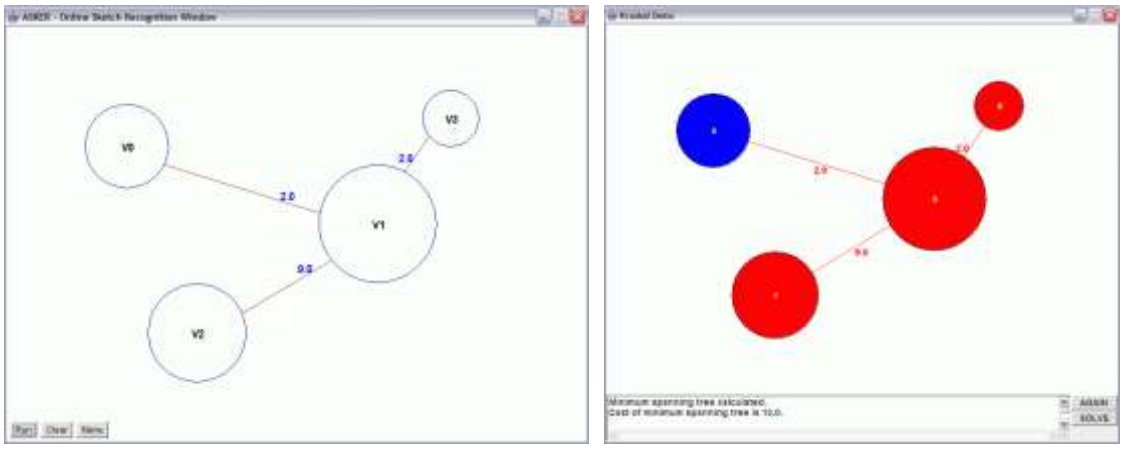


Figure 3.31. Kruskal Demo of sketched graph

3.5. Warnings

ASKER provides several warnings for an efficient usage of the system. In this section all these warnings are described and shown by figures.

Warning 1 shows that the sketched circle intersects at least one of the nodes.

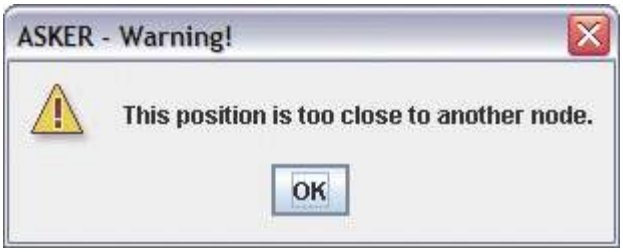


Figure 3.32. Warning 1

Warning 2 shows that the sketched line starts inside of a node but it is finished in free area.

Figure 3.33. Warning 2

Warning 3 shows that the sketched circle is fully inside of a node.

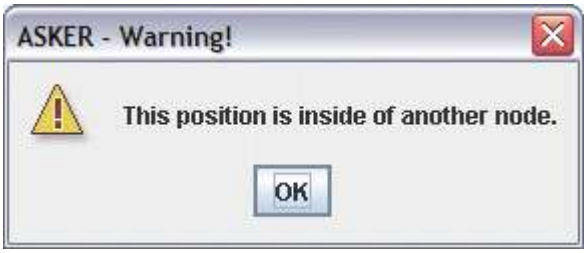


Figure 3.34. Warning 3

Warning 4 shows that the sketched line links two nodes which are also have a common edge, and also this is a undirected graph.

Figure 3.35. Warning 4

Warning 5 shows that the sketched line links two nodes which are also have a common edge in this direction, and also this is a directed graph.

Figure 3.36. Warning 5

Warning 6 shows that the sketched circle is too small to visualize efficiently.

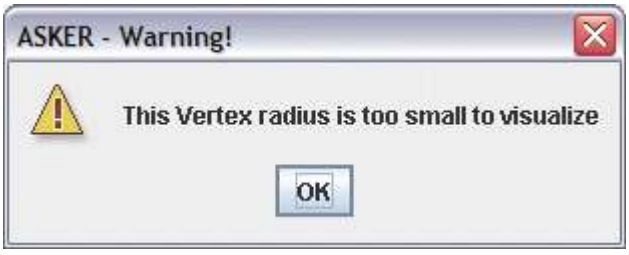


Figure 3.37. Warning 6

Warning 7 shows that the handwritten digit recognizer is active for weight entrance and also user tries to draw digits outside the recognition box or tries to sketch another shapes.

Figure 3.38. Warning 7

Warning 8 shows that the sketched circle is intersects with the borders of the drawing window.

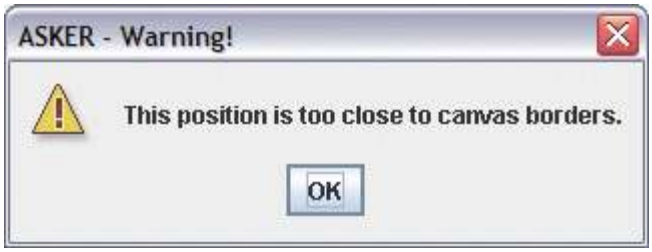


Figure 3.39. Warning 8

3.6. Test Results of PSA and Voting Algorithm

Parallel Sampling Algorithm (PSA), is a new approach to calculate average fitting error of a sketch with respect to a prototype. PSA is a core method, which can be used in several sketch recognition systems. Sequence of points obtained during a sketch determines the performance of PSA. Modified versions of PSA can be generated by sorting the points with increasing y-coordinate or increasing x-coordinate order. Modified version can be used for sequence-independent pattern or sketch recognition.

Digit and sign recognition tests for determination of the proposed PSA and Voting Algorithm were done. Totally 15 people sketched 2 samples for every digit and every sign.

Digit domain = {0,1,2,3,4,5,6,7,8,9}.

Sign domain = {deletion sign, check sign}.

In these tests, Wacom Pen Partner which is shown by the following picture, is used as a sketched point collector.

Figure 3.40. Wacom Pen Partner

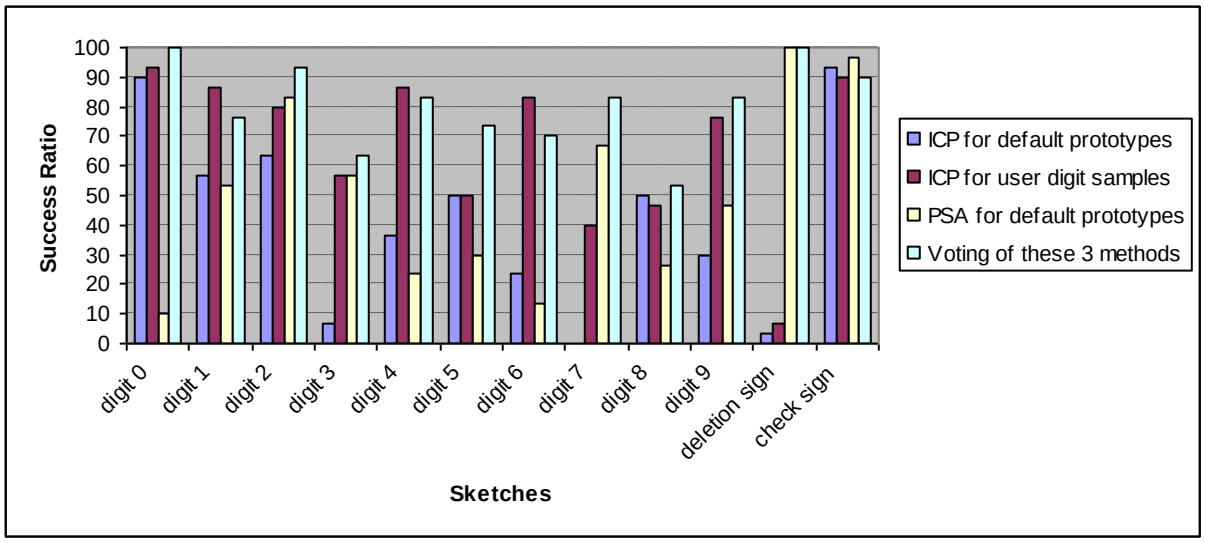


Figure 3.41. Recognition Success ratio of sketched samples by indicated four methods

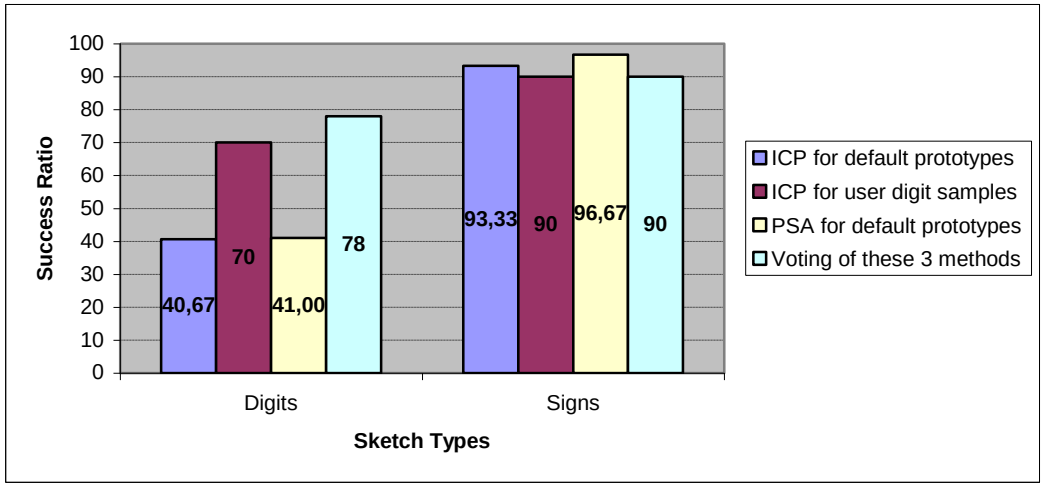


Figure 3.42. Recognition success ratio of digits and signs by indicated four methods

As the charts show, indicated ICP methods and PSA have dissimilar success ratios for different digits and signs. Generally ICP is better for digits, and PSA is better for signs. It is because ICP is sequence independent but PSA is a sequence dependent method.

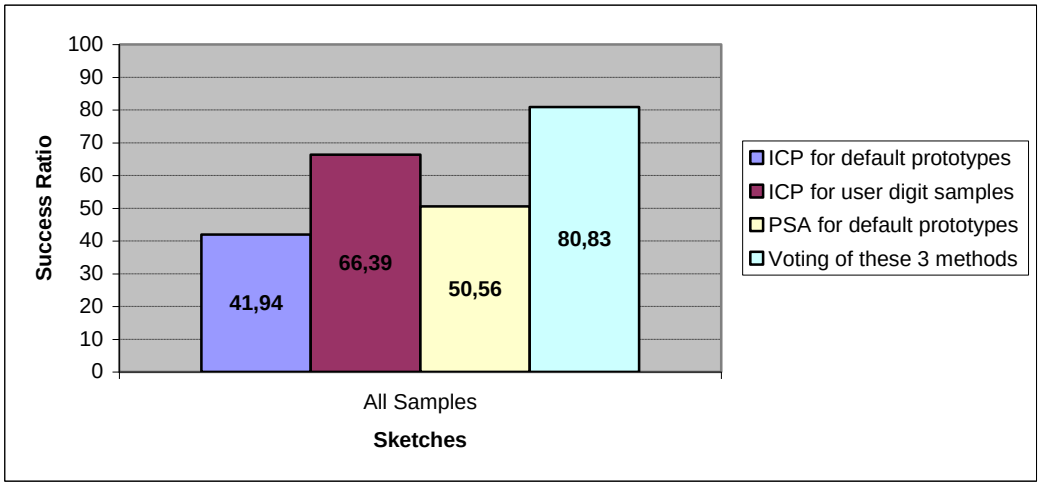


Figure 3.43. Average recognition success ratio of all samples by indicated four methods

In addition, the method with the best success ratio can be different for different sketches. But the main thing is that PSA gives good results for the situations in which ICP fails. By this manner Voting Algorithm can be used to increase success ratio of the recognition.

As a result, in sequence dependent systems, PSA is better than ICP and Voting Algorithms. But the best solution for digit recognition systems and hybrid recognition systems which consist of digit and sign samples as ASKER, is the proposed Voting Algorithm.

4. CONCLUSION

Graphs are indispensable for representing networks, maps of roads, railways, airline routes, pipe systems, telephone lines, electrical connections, prerequisites amongst courses, dependencies amongst tasks in a manufacturing system and a host of other data. There are a large number of important results and structures that are computed from graphs. Generally, initial graph designs are sketched on paper, because of the simplicity of drawing and editing. Designed graphs are later transferred to computers to test or to solve the related problems. To be useful in early design, computers must allow the designer to sketch as he does on paper and provide him with benefits, such as the ability to simulate his design that he does not have on paper.

Our goal is to develop an intelligent system which provides designers to sketch their graph design as if they were drawing on a paper. Furthermore, we expect that the system will represent the drawn graph in terms of edges, vertices and weight matrix, internally. Consequently, designed graph would also be converted into an input for any graph based application. By this manner, we created an intelligent system for graph designing and representing. Its all functions are designed and implemented suitable for natural drawing and writing. In this application, the user can easily draw and edit his sketch; moreover, the adjacency matrix representing the sketched graph is generated. Another feature of the tool is to provide animated solutions to well-known shortest path and minimum spanning tree algorithms such as Dijkstra and Kruskal. Any graph related simulator can be also embedded into the code of the developed tool.

As a future work, the digit recognition method will be optimized and its success ratio will be improved based on the test result statistics. In addition, a neural network can be added to this core recognition method. To sum up, the implemented online graph recognizer would be preferred by designers, instead of sketching on a paper, since it is a very basic, functional and a natural tool.

REFERENCES

1. Matt Notowidigdo and Robert C. Miller, "Off-line Sketch Interpretation", American Association of Artificial Intelligence, 2004.

2. Szymon Rusinkiewicz and Marc Levoy, "Efficient Variants of the ICP Algorithm", Third International Conference on 3D Digital Imaging and Modeling, 2001.

3. http://www.cs.berkeley.edu/~landay/research/publications/SILK_CHI/jal1bdy.html

4. J. Li, A. Najmi, R. M. Gray, Image classification by a two dimensional hidden Markov model, *IEEE Transactions on Signal Processing*, 48(2):517-33, February 2000.

5. Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, "Introduction to Algorithms", pp. 465-467, 8th Edition, 1992.

6. http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Graph/

7. Heloise Hse and A. Richard Newton, Sketched Symbol Recognition using Zernike Moments, Technical Memorandum UCB/ERL M03/49, Dec 2003.

8. http://svm.dcs.rhbnc.ac.uk/

9. Pawel Lewicki and Thomas Hill, Statistics Methods and Applications, Tulsa, 2006 http://www.statsoft.com/textbook/stsvm.html

10. http://en.wikipedia.org/wiki/Nearest_neighbor_(pattern_recognition)

11. Jay J. Lee, Jahwan Kim, And Jin H. Kim, "Data Driven Design Of HMM Topology For On-Line Handwriting Recognition", Applications In Computer Vision, pp. 107–121, 2001.

12. http://en.wikipedia.org/wiki/Hidden_Markov_model

13. A.Teredesai, E. Ratzlaff, J. Subrahmonia and V. Govindaraju, "On-Line Digit Recognition using Off-Line Features", International Conference on Computer Vision, Graphics and Image Processing, 2002.

14. Jonathan Alon, Vassilis Athitsos, and Stan Sclaroff, "Online and Offline Character Recognition Using Alignment to Prototypes", 2005.

15. Jorge, J. A., and Fonseca, "A simple approach to recognize geometric shapes", 1999.

16. Christine J. Alvarado, "A Natural Sketching Environment: Bringing the Computer into Early Stages of Mechanical Design", Master Thesis, MIT, May 2000.

17. Lin, J., Newman, M. W., Hong, J. I. and Landay, "Denim: Finnding a tighter fit between tools and practice for web site design", J. A. 2000.

18. Sezgin, T. M., and Davis, R., "Early processing in sketch understanding", Proceedings of 2001 Perceptive User Interfaces Workshop, 2001.

19. Hammond, T., and Davis, R. 2002. "Tahuti: A geometrical sketch recognition system for UML class diagrams. In Proceedings of AAAI Spring Symposium on Sketch Understanding", pp. 59-68.

20. Lank, E.; Thorley, J. S.; and Chen, S. J.-S. 2000. Interactive system for recognizing hand drawn UML diagrams. In Proceedings of CASCON 2000.

21. Nicholas Matsakis, "The Natural Log", Master Thesis, MIT, 1999.

REFERENCES NOT CITED

D. Chetverikov, D. Svirko, and D. Stepanov, “The Trimmed Iterative Closest Point Algorithm”, 2002.

Xiaolei Huang, Dimitris N. Metaxas, “Matching and Recognition of Shapes using Chord-based Point Density Graphs”, 2004.

James V. Mahoney and Markus P. J. Fromherz, “Three main concerns in sketch recognition and an approach to addressing them”, 2002.

Dean Rubine, “Specifying Gestures by Example”, Proceedings of the 18th annual conference on Computer graphics and interactive techniques pp. 329 – 337, 1991.

Tevfik Metin Sezgin and Randall Davis, “Handling Overtraced Strokes in Hand-Drawn Sketches”, MIT, 2004.

Tevfik Metin Sezgin and Randall Davis, “Scale-space Based Feature Point Detection for Digital Ink”, MIT, 2004.