# GLOBAL APPROACHES TO SOLVING RECOGNITION PROBLEMS OF NOISY IMAGES

A THESIS
SUBMITTED TO THE FACULTY OF ATLANTA UNIVERSITY
FOR PARTIAL FULFILLMENT OF THE REQUIREMENT FOR
THE DEGREE OF MASTER OF SCIENCE

BY
CHARLES THOMPSON

DEPARTMENT OF MATHEMATICAL AND COMPUTER SCIENCES

ATLANTA, GEORGIA

JULY 1989

ABSTRACT

COMPUTER SCIENCE

THOMPSON, CHARLES B.S. MOREHOUSE COLLEGE, 1985

GLOBAL APPROACHES TO SOLVING RECOGNITION PROBLEMS OF NOISY IMAGES

Advisor: Dr. Nazir A. Warsi

Thesis dated June, 1989

An important problem in the area of pattern recognition is automatic detection of certain pre-assigned elements of an image distorted by noise. In this research, a global approach will be used. One such approach is to use an optimal smoothing algorithm which depends on efficient dynamic programming computational techniques. The basic purpose of this research is to make this dynamic programming process efficient in terms of storage requirement and computational effort. Our goal, using the objective function, is to find an optimal order of optimization and then design an efficient computational technique.

Two global techniques will be presented in this paper. Included is a graph-searching technique and the above mentioned technique using dynamic programming. Emphasis will be on the development of an algorithm using dynamic programming.

ACKNOWLEDGMENT

I wish to express my deepest appreciation and sincere gratitude to those who have contributed their time, energy, and support to make this study possible.

Thanks are especially due Dr. Nazir A. Warsi, my thesis advisor. His instruction, suggestions, and patience were essential to the completion of this thesis.

Further thanks are also due Nasa Langley for providing financial, technical, and general support to help make this study possible. Special thanks are offered to Mr. Micheal Goode, Technical Monitor, for his technical support and to Dr. Samuel E. Massenberg, University Affairs Officer, for his general support.

TABLE OF CONTENTS

LIST OF FIGURES

# CHAPTER I

## Introduction

The purpose of this paper is to describe the development of an optimal smoothing algorithm which will recognize images in a digitized noisy environment. The problem is to automatically recognize certain pre-assigned elements of an image which is distorted by noise. The main objective is to find an image as similar as possible to the given image. A global approach of optimal smoothing algorithm may be used for this purpose.

The optimal smoothing algorithm involves, first, digitizing the original image and dividing it into regions which contain points called pixels. At each pixel, we measure a property "P," such as verticality, dottedness, linearity, rectangularity, etc. We assign the most reliable property value to each pixel. Then we use a smoothing algorithm to develop an objective function which is used to obtain a smoothed (optimal) value of the property "P." This problem amounts to unconstrained dynamic programming optimization. Also, any optimization problem which can be posed as dynamic programming can also be stated as a problem of finding the shortest path on a weighted graph. That is, a graph searching method can be used to find the optimal path (value). Graph searching methods have advan-

tages over dynamic programming, mainly because there are many algorithms for finding the shortest path which use heuristic information to speed up the search.

Heuristics can be thought of as "rules of thumb," that is, heuristic information can be embedded in a objective function or figure of merit. This objective function would contain information about the edges or contours of the given image to help speed up the search by rejecting image components that do not meet certain criteria.

Graph searching, in general, involves starting with a directed graph network **G** = **<V,E>** where **V** represents a set of nodes and **E** represents a set of directed edges with positive lengths. One of the nodes will be designated the start node. The problem is to find the shortest path from the start node to other nodes. The shortest path or length involving directed edges, can also be represented as a cost. Our problem can then be stated as finding the cheapest path from the start node to the other node. To find the cheapest path, we use one of the many algorithms which has been developed for this purpose. We begin with the start node and generate step by step our graph **G** until a goal node is found. At each step a node, already generated, is selected and the successors for that node are generated and pointers are setup from each successor back to its parent node. The nodes selected depend on the cost estimate as determined by a function used to evaluate the cost at each step. To speed

up the search, global information can be used. That is, information concerning the shape or the contour of our image can be put into a figure of merit function to aid in searching for the shortest path thereby minimizing the search time.

Dynamic programming can best be described as recursive optimization. We take a large problem and break it up into a number of smaller problems. Then the smaller problems are solved sequentially such that the results from the preceding problem is used recursively to solve the subsequent problem.

The simplest form of dynamic programming is the serial type. Basically a serial system consists of two or more stages connected or joined together in series where the output of one stage becomes the input to the next stage. A typical Dynamic programming stage is characterized by five factors:

1. An input or initial state (X) which contains all the necessary information about the input.
2. An output or final state (Y) which contains all the relevant information about the output.
3. A decision variable which contains the information to be manipulated to achieve the desired result.
4. A return variable, scalar, to measure effectiveness using the objective function.
5. A transformation function which expresses each

component of the output as a function of the input state and decision variable.

A more complex form of dynamic programming is the nonserial type. There are four basic nonserial structures. All nonserial structures, have one thing in common: from at least one of the stages there are two or more inputs and/or outputs. The implications are that one would have two or more serial systems within a nonserial dynamic programming structure. The problem then becomes that of finding the optimal path or optimal serial structure. If an exhaustive search procedure is employed, the required storage and computation time become exponential for large problems. Therefore, we must develop a special procedure or heuristic to minimize the storage and computation requirements of these problems.

The dynamic programming aspect of this research consists of two parts. First, an optimal order in which the nonserial dynamic programming stages can be solved must be found. Second, the design and use of an efficient computational technique must be effected to process the linearized dynamic programming once an optimal order has been obtained. This paper describes a method or technique which will help to solve the problems caused by dimensionality, increased time, and increased storage due to multiple inputs at each stage. Using this method, the goal is to make the storage and central processing unit time manageable for

practical applications.

It is important to note that as the dimensions of state variables in dynamic programming increase, the problem reduces to that of the Traveling Salesman Problem[107]. Since this problem is **np-complete**, generally, heuristics are utilized in its solution. We briefly discuss the concepts of heuristics and their applications along with the Traveling Salesman Problem in Chapter II.

As alluded to earlier, the dynamic programming formulation of the smoothing procedure has an equivalent graph searching counterpart. Consequently, we present the basic concepts of graph searching in Chapter III.

The computational techniques of dynamic programming applied to the smoothing algorithm form the core of this study. In Chapter IV, we introduce such a computational technique for the serial case to highlight various demands and issues. In Chapter V, we present an unconstrained dynamic programming and related computational methods for a nonserial case.

Chapter VI deals with the details of the smoothing algorithm and its formulation as dynamic programming optimization. Various techniques of dynamic programming computation are discussed in relation to the optimal smoothing algorithm.

Finally, in Chapter VII, a comparative summary of graph searching and dynamic programming is presented.

# Chapter II

# Intelligent Techniques For Complex Problems

## 2.1 Heuristics

Heuristics may be described as criteria for selecting courses of action from among many alternatives based on which the most efficient results can be obtained for reaching a particular goal. One could view heuristics as rules of thumb used in determining what actions to take. For example, a popular method for determining if a cake has completely baked involves inserting a toothpick into the cake then removing it to see if crumbs remain. This method does not always determine if the cake is completely baked but is effective most of the time.

The requirements for good heuristics is that they should not only provide a simple means for determining the best course of action but also, sufficiently often, guarantee to identify the most effective course.

Most complex problems require the evaluation of an astronomical number of possibilities to determine an exact solution. In many of these complex problems, the time required to find an exact solution takes more than a lifetime. Heuristics are used to indicate a way to reduce the number of evaluations and obtain solutions within reasonable time constraints.

## 2.2 The Traveling Salesman Problem

An example of a complex problem is the **Traveling Salesman Problem.** In this problem, the object is to find the cheapest path for visiting every city, given a number of cities, once and only once and then returning to the initial city. This problem is represented by a graph, with each node representing a city and the edges representing the cost (distance) between nodes (cities). This problem is usually stated for a complete graph, that is, a graph in which every node is connected to each other node.

The **Traveling Salesman Problem** belongs to a class of problems known as **np-complete** in which all known algorithms for obtaining exact solutions require exponential time in the worst case. Using a good bounding function makes the determination of optimal paths a lot faster and a lot cheaper.

Figure 2.1 represents a graph for the Traveling Salesman Problem. This graph shows two paths marked ABC and ADE, with ABC and ADE representing two partially completed paths which are to be considered in our search to find the optimal path. By using the two marked paths, the final solution will ultimately be determined by the cost of the initial path (ABC or ADE) plus the cost of the final path through the remaining nodes. Even while using the partial paths, the final solution still remains almost as difficult as

finding the entire solution. Therefore, we need heuristics to come up with an estimate of the completion cost. Once we have this estimate, we can determine which partial path to explore first, and then combine the cost of the explored path with the estimate of its completion (the one that offers the lower overall cost estimate).

Using this heuristic estimate at each stage, we select for expansion that partial path which gives the lowest estimated completed path cost. If our heuristic function gives a good estimate, then the partial path that is selected and found to be complete is also the cheapest path.

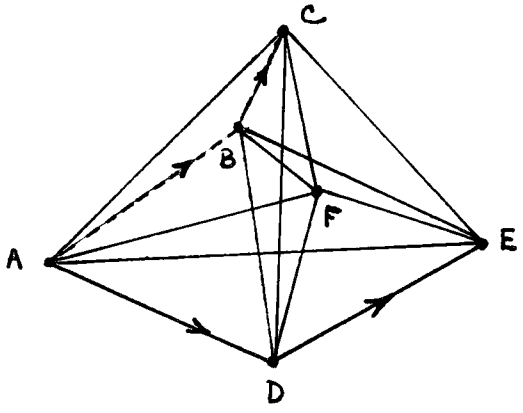
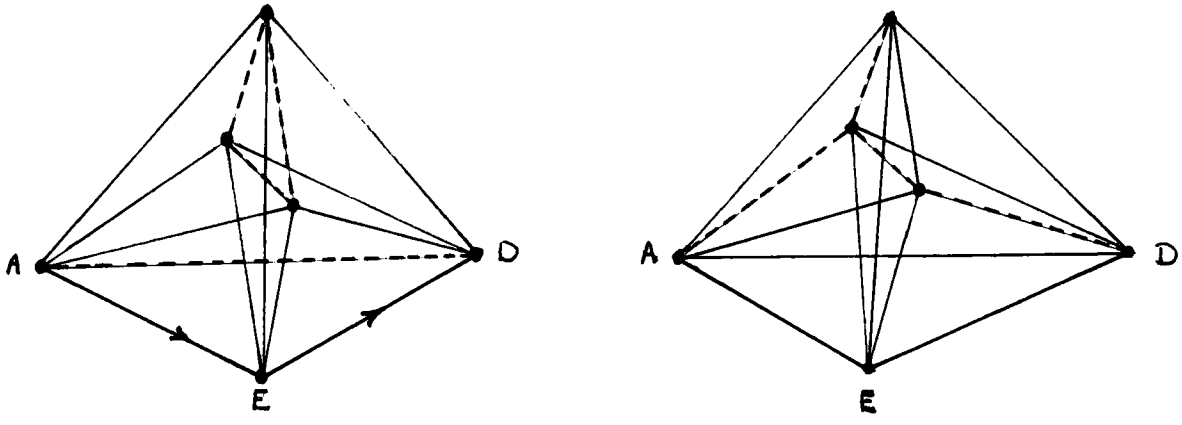



**Figure 2.1 Graph Representing the Traveling Salesman Problem.**

Two of the more popular bounding functions used to solve the Traveling Salesman Problem involves finding the cheapest **Second Degree Graph** going through the remaining nodes and finding the **Minimum Spanning Tree** through all remaining nodes. These functions are obtained by solving the

optimal assignment problem. The second degree graph function requires computational steps of the order $N^3$ and the minimum spanning tree computational requirement is of the order $N^2$. The fact that these two functions provide optimistic estimates of the completion cost is apparent when we consider that completing the tour requires selecting a path that goes through all unvisited cities. We have a path which is a special case of a **Two Degree Graph** and a path which is a special case of a **Spanning Tree.** Since the set of all completion paths is included in the optimization over the set of path only, the cost of the solution found must be lower than the optimistic estimate. Figure 2.2a shows the shortest **Two Degree Graph** with sub-path AED. Figure 2.2b shows a **Minimum Spanning Tree** completion of the sub-path AED.




(a) (b)

**Figure 2.2 (a) Two Degree Graph and (b) Minimum Spanning Tree.**

Even though the shapes of these graphs do not resemble the desired tour, the costs of their solutions represent good optimistic estimates of the optimal tours. Therefore, they can be used as heuristics for improving the search efficiency.

## 2.3 Optimizing, Satisficing, and Semi Optimizing Tasks

By optimization we mean, as mentioned in the Traveling Salesman Problem, finding a tour (path) at least as cheap as any other tour. Satisficing refers to finding a qualified object or solution with as little search effort as possible.

Most problems can be posed as both constraint-satisfaction and optimization tasks. As an example, suppose we take a chess board and attempt to place eight queens on the board in such a way that no queen can attack another. That is, each queen is placed on the board such that no two queens are located on the same row, column, or diagonal. The goal is to find the cheapest placement of the queens on the board where each capturable queen represents a cost of one unit. Presenting the problem in this manner generally makes it more difficult since we are not certain if a zero-cost solution exists. Even if we could prove that such a solution exists, we would still have to search other higher configurations to determine the optimal solution. However, using heuristics can be very instrumental in reducing the

effort of finding a quality solution.

In theorem proving, we would normally be satisfied with the first proof found, however ugly. By focusing the search effort toward finding the shortest proof, prevents us from making aimless explorations in a large space of possibilities, which leads to smaller search efforts in finding a more elegant proof. This principle, called the **small-is-quick**, plays a major role in heuristic methods for satisficing search.

Often the difference between the complexity of satisficing and task optimization is substantial. For example, in the Traveling Salesman Problem, finding some tour through a set of cities is trivial , but finding the optimal tour is **np-hard.** In order to minimize, search time one may be forced to relax optimality requirements and settle for finding a good solution using a reasonable amount of search effort. Whenever our criterion allows us to accept a solution close to optimal or within a neighborhood of optimal, the problem becomes one of **semi-optimization.**

**Semi-optimization** falls into one of two categories. If the neighborhood for our solution is sharply defined (which means the solution must be very near optimal), our task is called **near-optimization.** If the criterion is further relaxed and the required solution must only be near optimal with a sufficiently high probability, the problem then becomes an **approximate-optimization** task.

Most practical problems are of the **semi-optimization** type, that is, a balance is required between the quality of the solution and the cost of searching for a solution. Since searching for the optimal solution being np-hard can very easily reach astronomical proportions in terms of time and cost, we must, out of economical necessity, relax the optimality requirement. Therefore, our problem is to devise algorithms which place bounds on search efforts to the extent we compromise our optimization. The following represents a description of the required tools:

1. A symbol structure or code which can represent each candidate object in the space;
2. Computational tools that are capable of transforming the encoding of one object into another in order to scan the space of candidate objects systematically; and
3. An effective method of scheduling these transformations so as to produce the desired object as quickly as possible.

In artificial intelligence terms, these requirements are known as the **database**, the **operators** or **production rules**, and the **control strategy**. A systematic control strategy is one in which every object is examined, and examined only once. This is for completeness and efficiency.

# Chapter III

# Intelligent Graph-Searching Procedures

## 3.1 Graph-Searching

An optimal smoothing algorithm basically involves search strategies with graphs. To make the discussion easier, we will define some nomenclature regarding graphs and graph searching.

A graph 'G' is a pair of finite sets {V,E}, where 'V' is called the set of nodes (or vertices) and 'E' (which may be empty) is called the set of edges. With each edge e in E, we associate an unordered pair of nodes (a,b) that are called the end nodes of 'e'. Each graph under consideration will have a unique node 's' called the **start node**. As an example, consider a Graph 'G' which has the set V = {a,b,c,s} as its set of nodes and the set E = $\{e_1,e_2,e_3,e_4\}$ as its set of edges, where $e_1$ = (a,b), $e_2$ = (b,c), $e_3$ = (b,s), and $e_4$ = (a,c) (Figure 3.1).

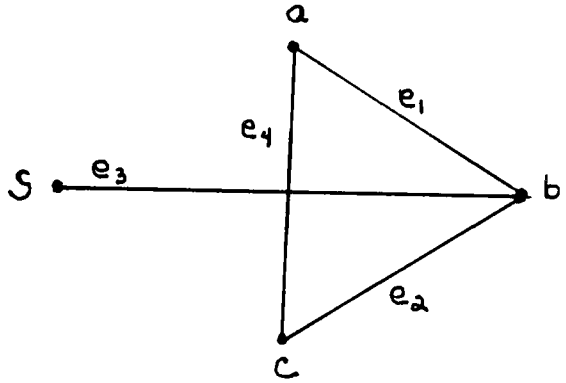



**Figure 3.1 Graph G.**

If an edge is directed from node **n** to node **n'**, node **n'** is said to be a successor of **n** and node **n** is the parent of **n'**. The number of successors emanating from a given node is called the degree of that node. A complete graph on **n** nodes is a simple graph where every pair of distinct nodes is connected by an edge. This graph is represented by $C_n$. Figure 3.2 shows the representation for complete graphs $C_3$ and $C_5$.

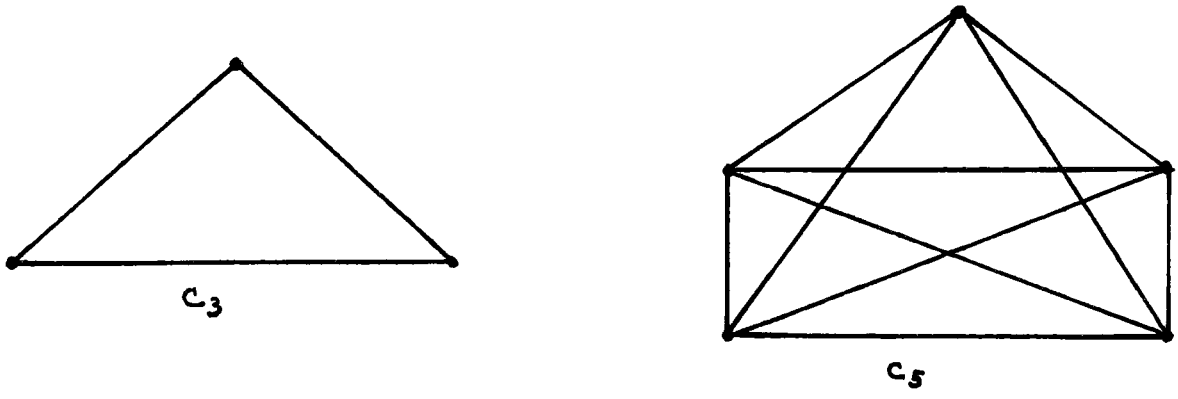



**Figure 3.2 Complete Graphs for $C_3$ and $C_5$.**

A digraph (directed graph) **D** is a finite set **A**, which contains the nodes or vertices of a graph, together with a subset **E** of **AxA**. Each ordered pair **(a,b)** in **E** is called a directed edge from a to b. Edges in digraphs are represented by arrows. Figure 3.3 (a) and (b) are examples of digraphs.

Let **D** be a digraph and let **a** be a node in **D**. The out-degree of **a** is the number of directed edges (arrows) leaving **a** and the in-degree of **a** is the number of edges entering **a**. The total degree of **a** is the sum of its in-degree and its

out-degree. In Figure 3.3a the in-degree of b is 2 and the out-degree of b is 1. The total degree of b is 3 (out-degree 1 and in-degree 2). In any digraph, a node with in-degree 0 is called a **source** and a node with out-degree 0 is called a **sink**. Figure 3.3b is an example of a graph where a has an in-degree of 0 and c has an out-degree of 0.

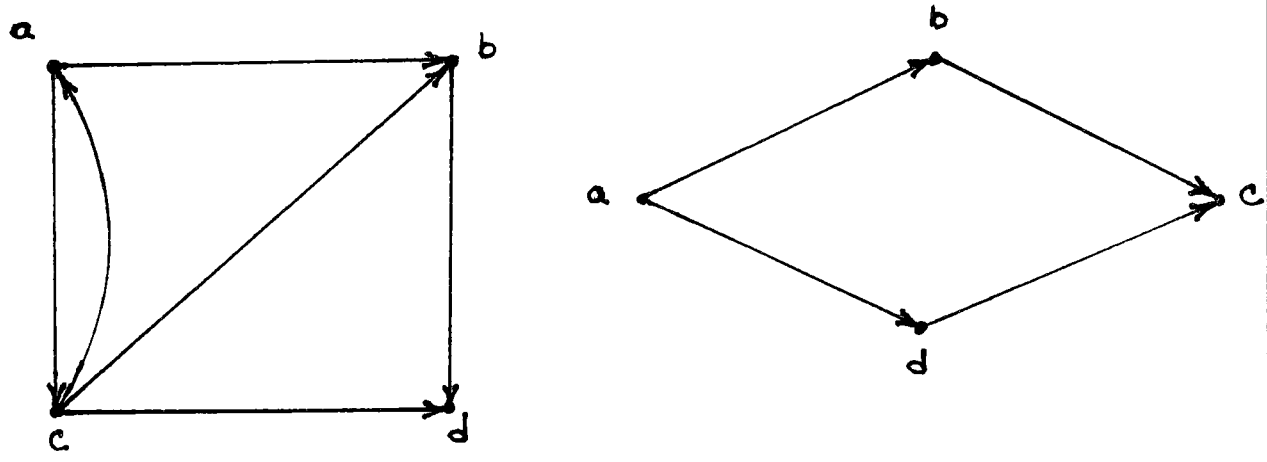



**FIGURE 3.3 Digraphs.**

Consider a set of edges, $(a_0,a_1),(a_1,a_2),\ldots,(a_{n-1},a_n)$ where $a_n = a_0$. Such a set is called a directed cycle denoted by $a_0 \Rightarrow a_1 \Rightarrow a_2 \Rightarrow \ldots a_{n-1} \Rightarrow a_n = a_0$. A digraph is acyclic if it contains no cycles. If D is an acyclic digraph, then D has at least one and only one sink. Figure 3.4 shows examples of acyclic graphs. In Figure 3.4a we have a tree structure which clearly does not have a cycle. In figure 3.4b the circuit (a,b,c,e,a) does not form a directed cir-

cuit due to the violation of edge orientation (c,e). Therefore Figure 3.4b is also acyclic.

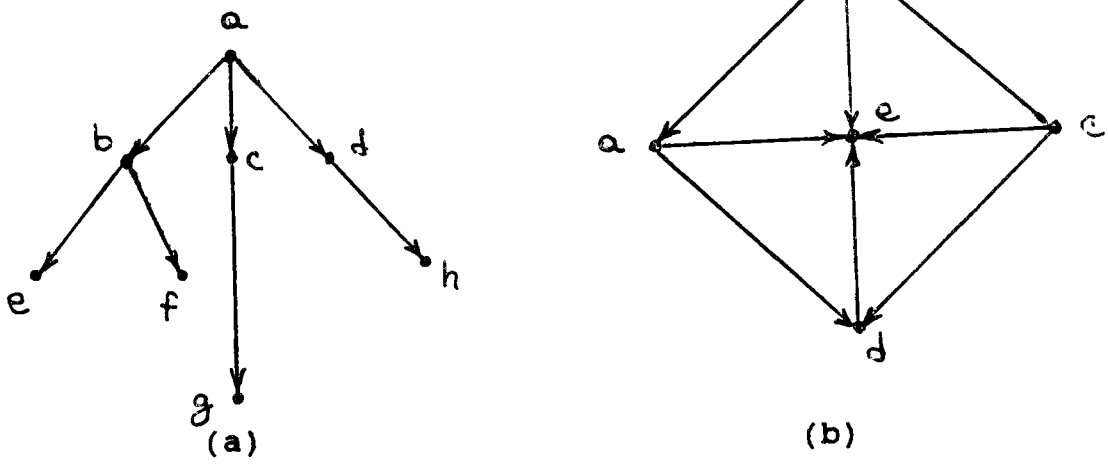



**FIGURE 3.4 Acyclic Digraphs.**

A tree is a graph in which each node (except one root node) has only one parent. A node that has no successor is called a leaf or a terminal node. Often the edges on the tree are assigned weights representing either costs or rewards determining whether they will be a part of the final solution in procedures involving graph traversals.

The most elementary step of graph searching is **node generation**, that is, generating or computing a code representing a node with respect to its parent. The new successor is then said to be generated and its parent **explored**.

Another computational step of great importance is called **node expansion**, which consists of generating all the

successors of a parent node. The parent is then said to be expanded. Pointers are usually set up from each successor node back to the parent node. These pointers provide for tracing a path from a goal node n back to the start node or some other ancestor node.

A search procedure is a prescription for determining the order in which nodes are to be generated. These search procedures can be either uninformed (blind), informed, guided, or directed. In the uninformed situation, the order in which the nodes are expanded depends only on the information gathered by the search, but remains unaffected by the unexplored nodes or by the goal criterion. The informed search uses partial information about the problem domain and the nature of the goal criterion for making the search as efficient as possible.

The set of nodes in a graph being searched can at any time be divided into four disjoint subsets:

(1) Nodes that have been expanded;

(2) Nodes that have been explored, but not yet expanded;

(3) Nodes that have been generated, but not yet explored; and

(4) Nodes that are still not generated.

In the search procedures to be subsequently discussed, it is necessary that a distinction be made between the first and third groups of nodes. Nodes that are expanded, making

their successors available to the search procedure, are called **closed**; nodes which have been generated and are awaiting expansion are called **open**.

## 3.2 Search Algorithms

Search algorithms used for trees includes, **Depth-First Search, Breadth-First Search, Hill-climbing,** and their variations. Many np-hard problems use other more complex techniques such as heuristic search using shortest path algorithms such as Dijkstra's algorithm (also called Greedy Algorithm), A* algorithm and Dynamic Programming. Many problems in Artificial Intelligence are reduced to finding an optimal path in a weighted graph.

Let **G(N,A)** be a directed graph with $n_i$ belonging to **N** and edge $(n_i,n_j)$ belonging to **A**. If an edge is directed from node $n_i$ to $n_j$, then $n_j$ is the successor of $n_i$ and $n_i$ is the parent of $n_j$. A cost $c(n_i,n_j)$ is assigned to each edge $(n_i,n_j)$. A sequence of nodes $n_{i1},n_{i2}, \ldots,n_{ik}$ with each node $n_{ij}$ a successor of $n_{i,j-1}$ for $j = 2,\ldots,k$ is called a path from $n_{i1}$ to $n_{ik}$ and the value

$$c = c(n_{i,j-1},n_{ij})$$

is its cost.

A procedure for finding a shorter path (using minimal

cost), starts with a specified or initial node 's' and then finds the shortest path (goal nodes). An algorithmic description of the procedure is as follows:

(1) Start with the initial node 's';

(2) Expand successors nodes and set pointers from each successor back to its parent node; and

(3) Check the successor nodes to see if they are goal nodes

(4) When a goal node is found trace back from the goal node to the start node to produce a solution path.

To speed up the search, special information called heuristic information, is used to direct the search. Let f'(n) be an estimate of the cost of the minimal cost path from 's' to a goal node 'n'. The node having the smallest value of f' is the node estimated to be on a minimal cost path; therefore it is expanded next. The estimate f'(n) can be expressed as the estimate of a minimal cost path from node **n** to a goal node:

$$f'(n) = g'(n) + h'(n)$$

Let g'(n) be the cost from 's' to 'n' given by the sum of all edge costs encountered in tracing the pointers from 'n' to 's' (this represents the lowest cost path); and h'(n) obtained from the heuristic information. The following is the **A* Algorithm** which makes use of the f' cost estimate.

## Algorithm A*

(1) Mark the start node 's' open and set g'(s) = 0.

(2) If no node is open, exit with failure; otherwise continue.

(3) Mark the open node 'n' whose total estimate

$$f'(n) = g'(n) + h'(n)$$

is smallest as closed. (resolve ties for minimal f' values arbitrarily, but always in favor of any goal node).

(4) If 'n' is a goal node, exit with the solution path obtained by tracing back through the pointers; otherwise continue.

(5) Expand node 'n', generating all if its successors. (if there are no successors go to (2)).

(6) If a successor $n_i$ is not marked, set

$$g'(n_i) = g'(n) + c(n,n_i);$$

mark it open; and direct pointers from it back to n.

(7) If a successor $n_i$ is marked closed or open, update its value:

$$g''(n_i) = \min (g'(n_i), g'(n_i) + c(n,n_i)).$$

Mark those closed successors open, whose g' values were lowered, and redirect to 'n' the pointers from all nodes whose g' values were lowered.

(8) Go to (2).

In general, this algorithm is not guaranteed to find a minimal cost path to a goal. Hart, Nilsson, and Raphael have shown that, if h(n) is a lower bound on the cost of the minimal cost path from node 'n' to a goal node, then algorithm **A*** will find an optimal path to a goal node [101]. If h' = 0 (no heuristic estimate available) then algorithm **A*** is called the uniform cost algorithm and it coincides with the Dijkstra algorithm (see next algorithm).

The following is a greedy algorithm called Dijkstra's algorithm. The sets 'C' and 'S' are the set of available candidate nodes and the set of nodes already chosen. At every moment, 'S' contains the nodes with the known minimal distance with respect to the source and 'C' contains all the other nodes. Initially 'S' contains only the source. Once the algorithm is finished, 'S' contains all the nodes of the graph and the solution to the problem. The length of the shortest of the nodes in S is stored in an array 'D'.

The nodes of the graph are numbered from 1 to n, and N = {1,2,..., n}, where node 1 is the source or start, and Matrix L gives the length of each directed edge: L[i,j] => 0 if edge (i,j) exists and L[i,j] = infinity otherwise.

## Dijkstra's Algorithm

```
Function dijkstra[1..n, 1..n]): array[2..n]
1.  (initialization)
    C <= (2,3,....n) (S = N\C  exists only  by implica-
    tion);
2.  For i <= 2 to n do D[i] <= L[1,i];
3.  (greedy loop)
    Repeat n-2 times
      v <= some element of C minimizing D[v];
      C <= C\(v)  (and implicitly S <= S u (v));
      For each w ∈ C do
        If D[w] > D[v] + L[v,w] then
        D[w] <= D[v] + L[v,w];
        P[w] <= v;
```

# Chapter IV

# Serial Dynamic Programming Calculation

## 4.1 Three Stage Serial Dynamic Programming Problem

The following is an example of a simple serial dynamic programming problem. Figure 4.1 is a block diagram which represents a one stage system characterized by five factors which includes: the input state, the output state, the decision variable, the stage return, and stage transformation. Next we describe the steps involved in solving a

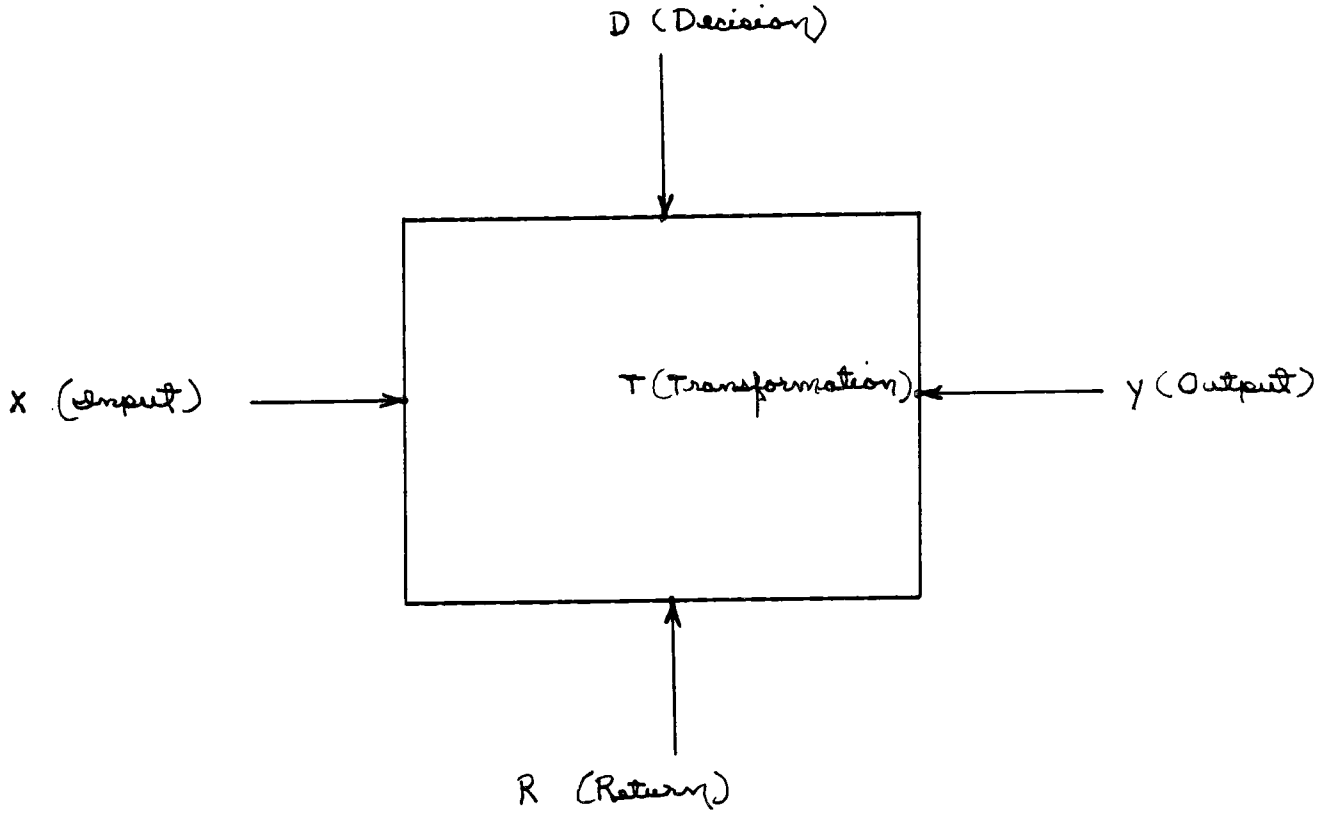



**Figure 4.1 One Stage in a Dynamic Programming Problem.**

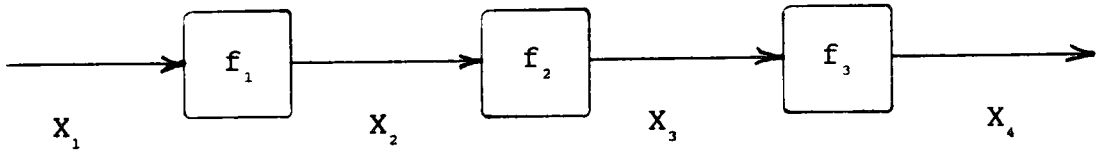
three stage Serial Dynamic Programming problem:

$$\min_{X_1,X_2,X_3} [f_1(X_1,X_2) + f_2(X_2,X_3) + f_3(X_3, X_4)],$$

where $X_1$, $X_2$, $X_3$ take discrete values and functional values ($f_1$, $f_2$, and $f_3$) are known. This can be optimized as follows using serial dynamic programming as follows:

1. $F_3(X_3) = \min_{X_4} f_3(X_3,X_4)$
2. $F_2(X_2) = \min_{X_3}[F_3(X_3) + f_2(X_2,X_3)]$
3. $F_1(X_1) = \min_{X_2}[F_2(X_2) + f_1(X_1,X_2)]$
4. $F_1(X_1^*) = \min_{X_1} F_1(X_1)$

Figure 4.2 is a graphical representation of the above three stage optimization process. Figure 4.3 includes three tables which contains the input values for each stage.




stage by stage

**Figure 4.2 Graphical Representation of a Three Stage Dynamic Programming System.**

Example

Given

| $X_1$ | $X_2$ | $f_1$ | $X_2$ | $X_3$ | $f_2$ | $X_3$ | $X_4$ | $f_3$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 4 | 1 | 1 | 1 | 1 | 1 | 2 |
| 1 | 2 | 3 | 1 | 2 | 2 | 1 | 2 | 2 |
| 2 | 1 | 1 | 2 | 1 | 3 | 2 | 1 | 3 |
| 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 1 |

**Figure 4.3 Input Values for Three Stage Dynamic Programming System.**

Calculations

Use table 1 to calculate $F_3(X_3)$ for $X = 1,2$:

$$F_3(1) = \min_{X_4} f_3(1,X_4) = \min\{f_3(1,1), f_3(1,2)\}$$

$$= \min\{2,2\} = 2 \text{ for } X_4 = 1;$$

$$F_3(2) = \min\{f_3(2,1), f_3(2,2)\}$$

$$= \min\{3,1\} = 1;$$

Calculate $F_2(X_2)$ for X2 = 1,2 (use table 2):

$$F_2(1) = \min_{X_3}\{F_3(X_3) + f_2(1,X_3)$$

$$= \min\{F_3(1) + f_2(1,1), F_3(2) + f_2(1,2)\}$$

$$= \min\{2 + 1, 1 + 2\} = 3;$$

$$F_2(2) = \min\{F_3(1) + f_2(2,1), F_3(2) + f_2(2,2)\}$$

$$= \min\{2 + 3, 1 + 1\} = 2;$$

Calculate $F_1(X_1)$ $X_1 = 1,2$:

$$F_1(1) = \min\{F_2(1) + f_1(1,1),\ F_2(2) + f_1(1,2)\}$$

$$= \min\{3 + 4,\ 2 + 3\} = 5;$$

$$F_1(2) = \min\{F_2(1) + f_1(2,1),\ F_2(2) + f_1(2,2)\}$$

$$= \min\{3 + 1,\ 2 + 2\} = 4;$$

| $X_3$ | $F_3$ | $X_4$ |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 1 | 1 |

TABLE 1

| $X_2$ | $F_2$ | $X_3$ |
|---|---|---|
| 1 | 3 | 1 |
| 2 | 2 | 2 |

TABLE 2

| $X_1$ | $F_1$ | $X_2$ |
|---|---|---|
| 1 | 5 | 2 |
| 2 | 4 | 1 |

TABLE 3

**Figure 4.4 Tables Used to Calculate or Determine the Optimal Value and Optimal Variables.**

Now from table 3, min $F_1(X_1) = 4$ with $X_1 = 2$. But Table 3 gives $X_2 = 1$ for $X_1 = 2$. Table 2 gives $X_3 = 1$ for $X_2 = 1$. Table 1 gives $X_4 = 1$ for $X_3 = 1$. The optimal variables are $X_1 = 2$, $X_2 = 1$, $X_3 = 1$, and $X_4 = 1$. Hence the Optimal value is given by $f_1(2,1) + f_2(1,1) + f_3(1,1) = 4$. The order of optimization is obvious: $f_3 < f_2 < f_1$.

## 4.2 A Simple Algorithm for Serial Dynamic Programming

1. Find the order: $f_3 < f_2 < f_1$ as in the above example, i.e. the optimization of $f_3$ must precede that of $f_2$ and the optimization of $f_2$ must precede that of $f_1$.
2. Start with $f_3$ (the last one) and execute stage-by-stage optimization until $f_1$ has been processed. At each stage, before optimization, add the result of the previous stage.
3. Store results, when you optimize a stage, for use at the next stage. Optimal variables will be stored so that after the final optimization, the optimal variables at various stages can be retraced.

# CHAPTER V

## Computational Techniques in Nonserial Dynamic Programming

### 5.1 Nonserial Dynamic Programming

Consider the following optimization problem:

$$\min [f_1(X_1,X_{13},X_{12}) + f_2(X_{12},X_{24}) + f_3(X_{13},X_{34}) + f_4(X_{34},X_{24},X_4)].$$

An equivalent graph representation is given by Figure 5.1.

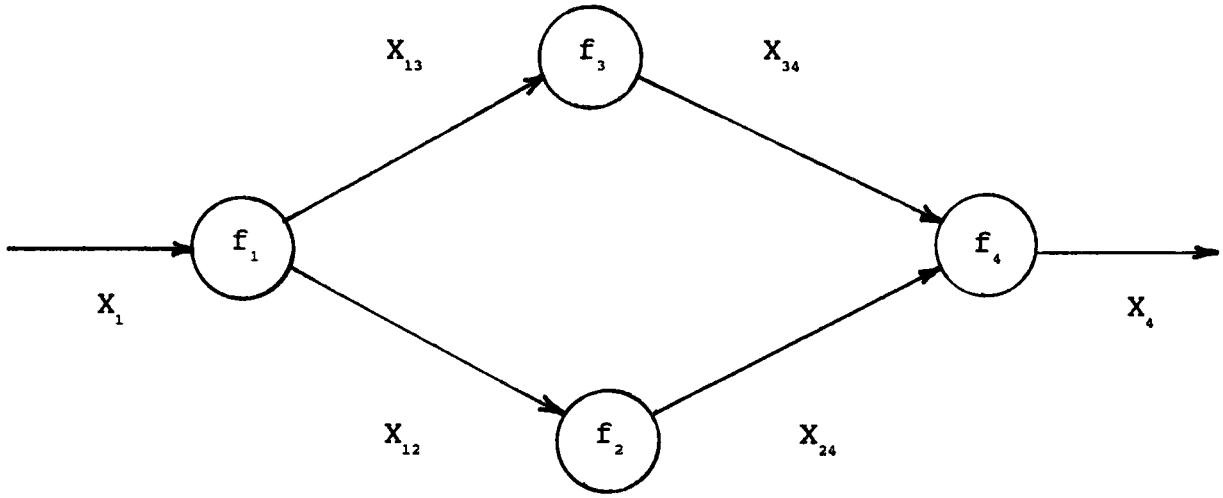



**Figure 5.1 Simple Nonserial Dynamic Programming System.**

The order of optimization is not obvious in this case; however, an ad hoc examination of interacting variables

gives the following:

$$1)\ F_4(X_{34}, X_{24}) = \min_{X_4} f_4(X_{34}, X_{24}, X_4)$$

$$2)\ F_3(X_{13}, X_{12}) = \min_{X_{34}, X_{24}} [F_4(X_{34}, X_{24}) + {}_3(X_{13}, X_{34}) + f_2(X_{12}, X_{24})]$$

$$3)\ F_1(X_1) = \min_{X_{13}, X_{12}} [F_3(X_{13}, X_{12}) + f_1(X_1, X_{13}, X_{12})]$$

$$4)\ F(X_{1*}) = \min_{X_1} F_1(X_1)$$

Therefore the order is $f_4 < f_3 + f_2 < f_1$.

In a complex nonserial case, the optimal order is rather difficult to determine by mere observation. For the sake of efficiency, the process of finding the optimal order must be automated. In the following section we consider a complex nonserial dynamic programming, whose ordering issues are discussed in details. We shall see in the next chapter that this example is exactly the formulation of the smoothing algorithm.

## 5.2 Example of Nonserial Dynamic Programming

Consider the following nonserial dynamic programming:

$$\min \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} f_{ij}(X_{ij}, X_{i+1,j}, X_{i,j+1}, P_{ij}) \text{ where}$$

$$f_{ij}(X_{ij}, X_{i+1,j}, X_{i,j+1}, P_{ij}) =$$
$$h\ (X_{ij}, X_{i+1,J}) + v(X_{ij}, X_{i,j+1}) + q(X_{ij}, P_{ij})$$

Figure 5.2 shows a possible graphical representation of the problem.

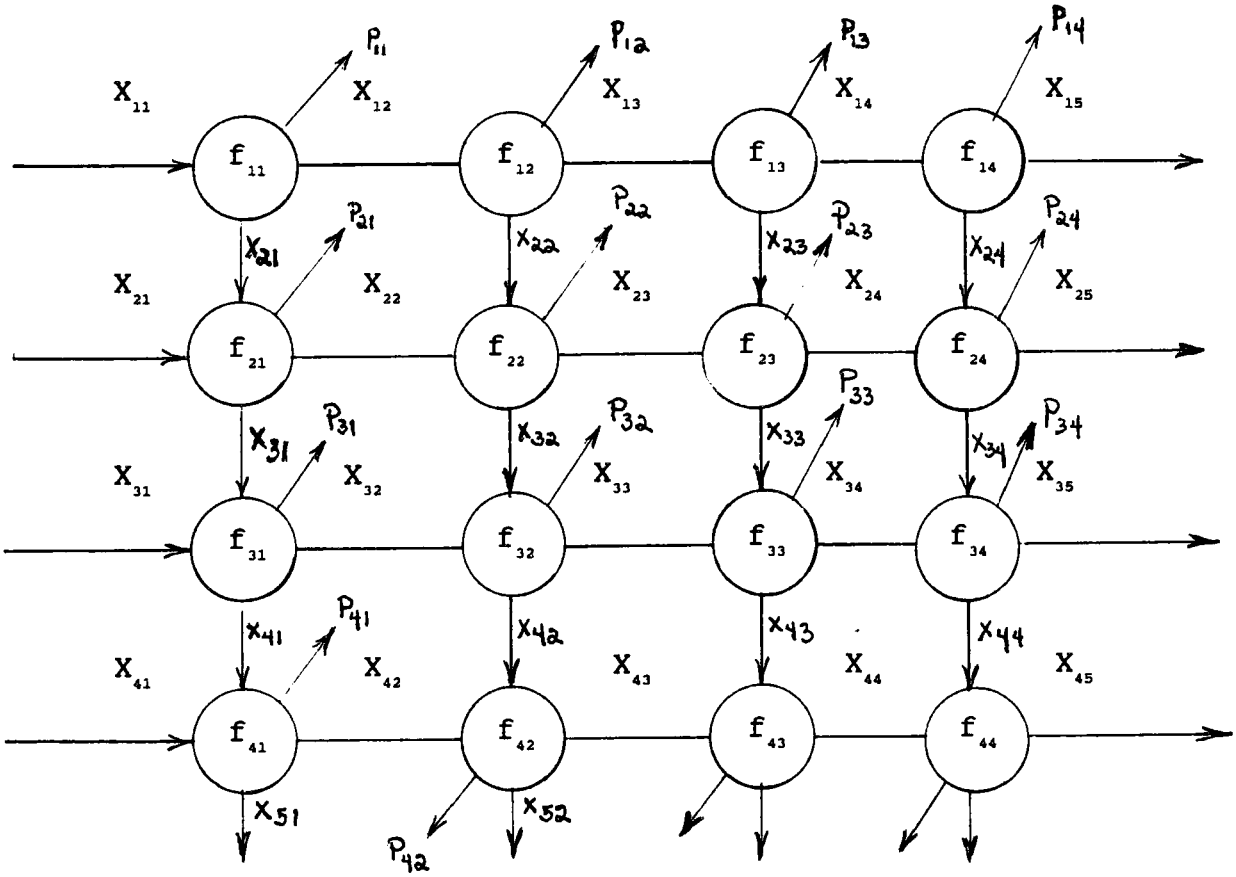



**Figure 5.2 Graphical Representation of Nonserial Dynamic Programming.**

An observation reveals that there are several optimal orders, one of which is given below.

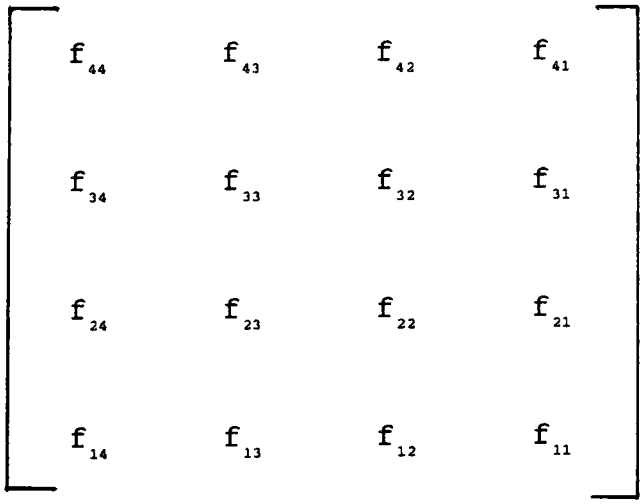



**Figure 5.3 One Order of the Graph in Figure 5.2.**

Here first, second, third, and fourth rows must be processed in that order, step by step. Such an ordering gives:

1. $F_{44}(X_{44}) = \min_{X_{45}, X_{54}, P_{44}} f_{44}(X_{44}, X_{45}, X_{54}, P_{44})$

2. $F_{43}(X_{43}) = \min_{X_{44}, X_{53}, P_{45}} [F_{44}(X_{44}) + f_{43}(X_{43}, X_{44}, X_{53}, P_{45})]$

Obviously we need to store only $F_{43}(X_{43})$ and $F_{44}(X_{44})$. In general, when we optimize: $F_{ij}(X_{ij})$ then we need to store $X_{i+1,j}(X_{ij})$ and $X_{i,j+1}(X_{ij})$.

## 5.3 Problems

Generally, in complex nonserial dynamic programming,

the order of optimization is not clear. Even if we find an order, we need to worry whether it is optimal. Yet there is another issue: can we reduce computational effort and storage.

In the literature there exist numerous attempts for finding optimal orders [14]. However, the computational and storage requirements of all existing algorithms are formidable. In the following we give a method akin to topological sorting of graphs. Because of the nature of applications (e.g., smoothing algorithm), the graph to be sorted will be assumed to have no cycles.

## 5.4 Topological Sorting

The process of finding the topological ordering of nodes of an acyclic digraph is called topological sorting. We will apply this method to the digraph in Figure 5.2. Once the graph has been sorted using the sorting procedure, the result will be a linearized ordering to which we can apply serial dynamic programming techniques.

This algorithm proceeds in a straightforward manner by listing out a vertex in the network with in-degree zero (note there always exists one because of no cycles). Then this vertex and all edges leading away from it are deleted from the network. These two steps are repeated until all nodes have been listed and then deleted from the network. A

simple or crude form of the topological sorting algorithm is as follows:

1. Input the network and let "n" be the number of vertices;
2. Output the vertices (in sorted order);
3. Put all nodes with in-degree zero on a stack;
   If stack is empty then end the procedure, else go to 4;
4. Pick a vertex with in-degree zero by popping the stack;
5. Output the vertex and remove the vertex and all out edges from the graph;
6. In removing an out edge consider the vertex to which this edge is leading. If its in-degree has reduced to 0 by removal of the edge then push the vertex on the stack and repeat 4.

The algorithm can be implemented by representing the network by adjacency lists. The adjacency lists for the graph in Figure 5.2 is given in Figure 5.4. The head nodes of these lists contain two fields: **COUNT** and **LINK**. The **COUNT** field contains the in-degree of that vertex and **LINK** is a pointer to the first node on the adjacency list. Each list has two fields: **VERTEX** and **LINK**. Count fields are easily set up at the time of input. When edge <i,j> is input, the count of vertex **"j"** is incremented by 1. As seen

above, the list of vertices with zero count is maintained in a stack. The strategy of maintaining a stack of zero-degree nodes makes the procedure fast. We note that when the removal of an edge reduces the degree of a node to zero, it can be pushed onto the stack without much effort; otherwise each call for a vertex with in-degree 0 will require a search of n elements in the list.

The details of the sorting algorithm using appropriate data structures are provided below.

## 5.5 Topological Sorting Procedure

```
procedure TOPOLOGICAL_SORT(COUNT,VERTEX,LINK,n)
    {The "n" vertices of the network are listed in topo-
     logical order. The network is represented as a set
     of adjacency lists with COUNT(i) = the in-degree of
     vertex i}
1.  top := 0      { initialize stack }
2.  for i := 1 to n do      { create a linked stack of ver-
                              tices with no predecessors }
3.   if COUNT(i) = 0 then ( COUNT(i) := top: top := i )
4.  end
5.  for i := 1 to n do  {print the vertices in topological
                         order}
6.    if top = 0 then [print ('network has a cycle'):stop]
7.    j := top: top := COUNT(top):print(j)
```

```
                                                { unstack a vertex}
8.      ptr := LINK(j)
9.      while  ptr # 0 do
           { decrease the count of successor vertices of j }
10.        k := VERTEX(ptr)     { k is a successor of j }
11.        COUNT(k) := COUNT(k) - 1     { decrease count }
12.        if COUNT(K) = 0         { add vertex k to stack }
13.          then [ COUNT(k) := top: top := k ]
14.        ptr := LINK(ptr)
15.     end
16.  end
17.  end TOPOLOGICAL_SORT
```

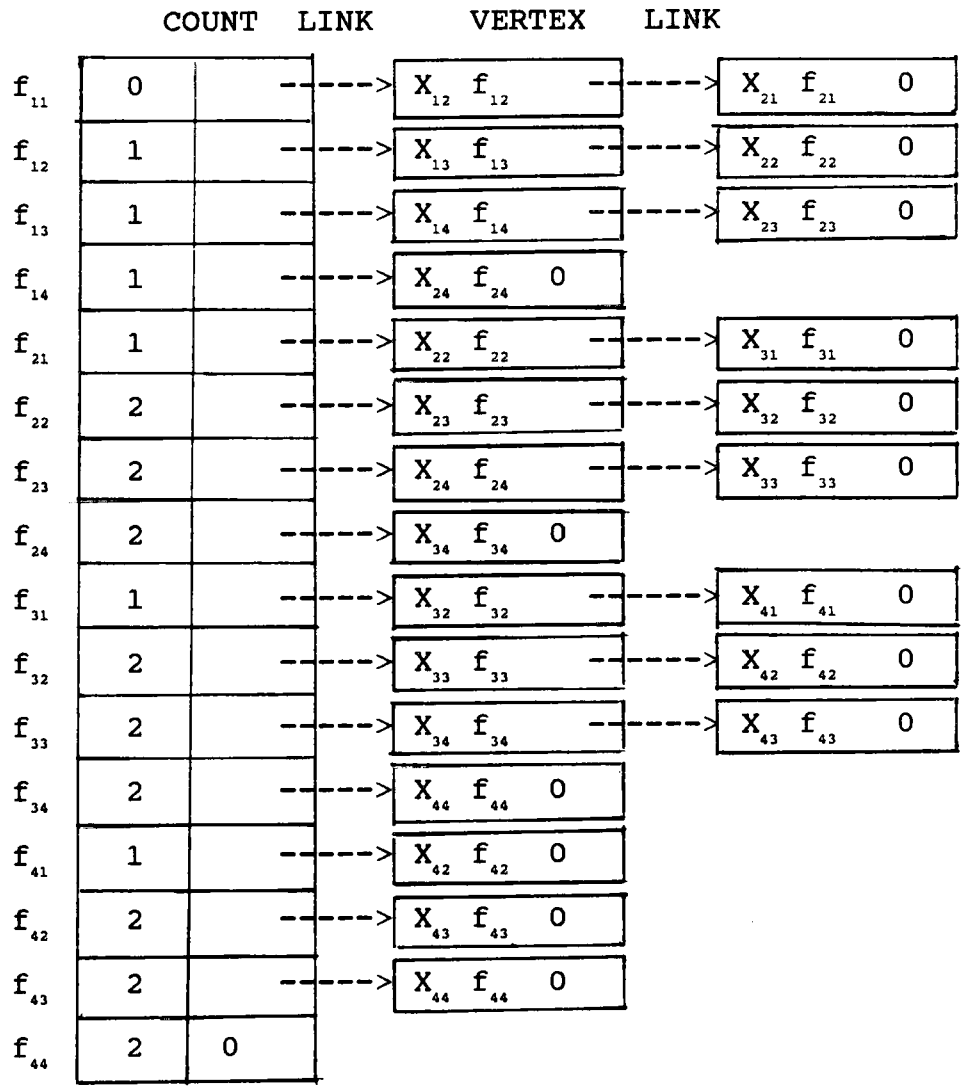



**FIGURE 5.4 Adjacency Structure or List for Graph in Figure 5.2.**

Using Figure 5.4 as input to the Topological Sort algorithm we see that $f_{11}$ has an in-degree of zero and $f_{44}$ has an out-degree of zero. This is the minimum requirement for a digraph to be acyclic. As the algorithm proceeds, the first

node, $f_{11}$, will be listed on the stack. Then all the edges leading out from $f_{11}$ will be deleted along with the vertex itself. Next we decrement the count of the successors of $f_{11}$, $f_{12}$ and $f_{21}$. The count (in-degree) of $f_{12}$ and $f_{21}$ is now zero, therefore either $f_{12}$ or $f_{21}$ could be selected as the next vertex to be evaluated or added to the stack. Suppose we select $f_{21}$, then $f_{21}$ is added to the stack and the vertex $f_{21}$ with its edges are deleted. Afterward we decrement the count of the successors of $f_{21}$, $f_{31}$ and $f_{22}$. The count of $f_{31}$ is decremented to zero and the count of $f_{22}$ is decremented to one. We now have two nodes or vertices with in-degree of zero, $f_{12}$ and $f_{31}$. We continue this process until all vertices have been selected or a cycle has been detected. The topological order selected is stored in the stack in reversed order. This order is; $f_{11}$, $f_{21}$, $f_{31}$, $f_{41}$, $f_{12}$, $f_{22}$, $f_{32}$, $f_{42}$, $f_{13}$, $f_{23}$, $f_{33}$, $f_{43}$, $f_{14}$, $f_{24}$, $f_{34}$, $f_{44}$, one of many possible orders. We now have a linearized order in which to perform our dynamic programming.

## CHAPTER VI

### Optimal Smoothing Algorithm

The problem is to automatically recognize certain pre-assigned elements of an image which is mixed or distorted by noise. The main objective is to find an image as similar as possible to the given image. A global approach of optimal smoothing algorithm may be used for this purpose. Generally, global information (heuristic) is needed or used to reduce the search or computation time. The global information contains knowledge about edges, contours, or some property relating to the given image.

A high level description of such a pattern recognition process for rectangular images is as follows:

1) Digitize the original image into pixels. The image or picture can be represented or stored in the form of a rectangular array, $[Y_{ij}]$.
2) Let "P" denote a property such as linearity, verticality, rectangularity, dottedness, etc. Measure a given property "P" at each pixel. In this smoothing algorithm, the properties being measured are verticality, horizontality, etc.
3) Reduce regions to points: A region $R_{ij}$, $1 \leq i \leq m$, $1 \leq j \leq n$ shrinks to a single point $Y_{ij}$.

4) Assign the most "reliable" property value $P_{ij}$ of "P" to $Y_{ij}$.

5) Use smoothing algorithm:

Suppose $x_{ij}$ denotes P-values in $p_{ij}$. Let **h** and **v** be the costs of horizontal and vertical irregularities respectively, and **q** the function penalizing the difference between $x_{ij}$ and $p_{ij}$. Now $x_{ij*}$, minimizing the following objective function gives the smoothed value of $x_{ij}$:

$$\min \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} [h(x_{ij},x_{i+1,j}) + v(x_{ij},x_{i,j+1}) + q(x_{ij},P_{ij})]$$

This is an example of unconstrained dynamic programming. The problem is potentially np-complete in terms of computation and storage. To reduce the computational efforts, we use global information which contains information about the given image. A picture has many horizontal and vertical levels and it is also noisy. To aid in the recognition of an image, global information relating to the edges or contours can be embedded in the objective function. In the absence of convexity conditions, this problem amounts to dynamic programming optimization.

To reduce the intermediate storage problem, all redundant and blocked states will be eliminated and a pseudo-

memory technique will be used.

Various techniques found in the literature [107] to reduce dimensionality, storage, and computation time includes; Fathoming (Elimination by Bounds), Reaching, Nearest Neighbor Technique, Minimum State Representations, and Data Structures (Computer Techniques and Codes).

Fathoming is a simple, yet extremely effective, technique of branch and bound used to eliminate states in dynamic programming algorithms. This technique or approach can be applied to finite dynamic programming problems. Basically, if we can demonstrate that no completion of an optimal sub policy for a state can lead to an optimal policy, then that state can be eliminated with no chance of overlooking any optimal policy. The computer storage requirements are reduced since it is not necessary to store either f(x) or the decision(s) which led to f(x) for any fathomed state "x." When compared to pure dynamic programming, applying a branch and bound technique results in a storage saving factor of $10^3$ and computational time saving factor of $10^2$.

Reaching is a label-setting technique for solving the functional equation of dynamic programming. This differs from the usual recursive computation procedure of dynamic programming (termed pulling) due to the order in which the calculations are performed. The use of reaching allows the exploitation of special structure in a manner that is not

possible with the usual recursive computation procedure of dynamic programming. Reaching can often be accelerated as in the case of dynamic programming with bounds. Even when there is no special structure to exploit or any means of accelerating reaching, it requires no more effort than the usual recursive computation.

The Nearest Neighbor Techniques are those techniques which reduces dimensionality by characterizing pairs of subsets. The state space of these subsets has the property that the only allowable transitions from the states of one subset are to the states of the other. An example of this type of reduction of dimensionality is a control problem with **M** state variables and **m** transition equations. Since the transition equations govern the motion of the system, it is often possible to reduce an **M** dimension problem to an **m** dimensional problem.

Dimensionality can often be reduced by exploiting special structures. The resulting dynamic programming algorithm is a special case of "reaching" in which candidates for current states are first temporarily labeled from the set of permanently labeled states of the previous embedded state space. The new labels are not declared permanent until tentative labeling has been completed and all infeasible and dominated states have been discarded. In this manner the embedded state space approach uses reaching to sequentially generate the complete family of undominated feasible

solutions to the resource allocation problem.

Finally, to improve the overall storage and efficiency of the algorithm, the use of data structures are of fundamental importance. In dynamic programming, the structures of primary importance are linked lists (dynamic structures), trees, and heaps. Effective and efficient implementation of techniques such as reaching and elimination by bounds necessitates linked lists (as opposed to arrays) storage. In linked lists, storage is allocated as needed.

The development of parallel computers may greatly influence future techniques for reducing dimensionality. These parallel machines have a number of different processing units which can execute instructions simultaneously. The problem is to isolate those computational components which can be processed in parallel.

## Chapter VII

### Summary of Graph-Search vs. Dynamic Programming

Graph-search techniques using global information about the given image can be very effective in finding a solution. If the optimal solution is not important, that is, if a semi-optimal solution will suffice, then several heuristics can be used to reduce the search for a solution and consequently the search time.

Even when the noise level is high, the use of good heuristics (global information) facilitates the detection of objects in the picture. The computing time for detecting images is directly related to the amount of noise associated with the image. As the noise level increases the computing time increases.

Dynamic programming techniques are considered blind. That is, the search time does not vary with the amount of noise associated with the image. Dynamic programming is very effective in recognizing patterns in a noisy environment. The use of global information (heuristics) will greatly reduce the search time. In our problem, we have applied a technique which reduces the search to O(n+k) time where n is the number of variables and k is the number of edges in the graph. In any event, hopefully our technique , in the worst case, will be as good as any graph-search tech-

nique.

Whether one uses a graph-search or a dynamic programming technique, for maximum efficiency, the global information should be embedded in a figure of merit function (objective function) and not in the whole program.

## BIBLIOGRAPHY


[1] Ahuja, N., and B. J. Schachter. Pattern Models, John Wiley and Sons, New York, 1983.

[2] Al-Alaoui, M. A. Some Applications of Generalized Inverse to Pattern Recognition. Georgia Institute of Technology, Atlanta, 1974.

[3] Albrecht, D. G. Recognition of Pattern and Form: Proceedings of a conference held at the University of Texas at Austin. Springer - Verlag, New York, 1982.

[4] Allen, P. K. Robotic Object Recognition Using Vision and Touch. Kluwer Academic Publishers, 1987.

[5] Andrews, H. C. Introduction To Mathematical Techniques in Patterns. Wiley-Interscience, 1982.

[6] Arkadev, A. G., E. M. Braverman. Computers and Pattern Recognition. Thompson Book Co. Washington, D.C. 1967.

[7] Baird, H. S. Model-based Image Matching Using Location. MIT Press, Cambridge, Mass, 1985.

[8] Baird, W.A. Adaptation of Pattern Recognition Software. Engineering Experiment Station, Georgia Institute of Technology, 1980.

[9] Batchelor, B. G. Practical Approach to Pattern Classification. Plenum Press, New York, 1974.

[10] Batchelor, B.G. Pattern Recognition: Ideas in Practice. Plenum Press, New York, 1978.

[11] Bayer, U. Pattern Recognition Problems in Geology and Paleontology. Springer - Verlag, New York, 1985.

[12] Bellman, R. Dynamic Programming. Princeton University Press, Princeton, New Jersey, 1957.

[13] Bellman, R. and S. E. Dreyfus. Applied Dynamic Programming. Princeton University Press, Princeton, New Jersey, 1962.

[14] Bertele, U. and F. Brioshi. Nonserial Dynamic Programming. Academic Press, New York, 1972.

[15] Bezdek, J. C. Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press, New York, 1981.

[16] Biberman, L. M. Perception of Displayed Information. Plenum Press, New York, 1973.

[17] Bongard, M. Pattern Recognition. Spartan Books, 1970.

[18] Boulay, B. D., D. Hogg and L. Steels. Advances In Artificial Intelligence. Elsevier Science Publishing, 1987.

[19] Bossel, H. and S. Klaczko. Systems Theory in the Social Sciences: Stochastic and Control Systems, Pattern Recognition, Fuzzy Analysis, Simulation, Behavioral Models. Birkhauser, Basal, 1976.

[20] Bow, S. Pattern Recognition: Applications to Large Data-Set Problems. M. Dekker, New York, 1984.

[21] Briot, M. Proceedings of the 6th International Conference on Robot Vision and Sensory Controls. IFS, Bedford, 1986.

[22] Caianiello, E. R. and G. Musso. Cybernetic Systems: Recognition, Learning, Self Organization. John Wiley and Sons, New York, 1984.

[23] Cappellini, V. and R. Marconi. Advances in Image Processing and Pattern Recognition: Proceedings of the International Conference, Pisa, Italy. Elsevier Science Publishing, New York, 1985.

[24] Casasent, D. P. Optical Pattern Recognition. Society of Photo-optical Instrumentation Engineers, Bellingham, Washington, 1979.

[25] Cathey, W. T. Optical Information Processing and Holography. John Wiley and Sons, New York, 1974.

[26] Caulfield, H. J. Analog Optical Processing and Computing. SPIE---The International Society for Optical Engineering, Bellingham, Washington, 1985.

[27] Chagas, C., R. Gattass, and C. Gross. Pattern Recognition Mechanisms: Proceedings of a Study Week. Springer-Verlag, New York, 1985.

[28] Charniak, E. and D. Mcdermott. Introduction to Artifical Intelligence. Addison-Wesley Publishing Com-

pany, 1985.

[29] Chen, C. H. Statistical Pattern Recognition. Hayden Book Co., Rochelle Park, New Jersey, 1973.

[30] Chen, C. H. Pattern Recognition and Signal Processing. Sijthoff & Noordhoff, 1978.

[31] Chen, C. H. Nonlinear Maximum Entropy Spectral Analysis Methods for Signal Recognition. Research Studies Press, New York, 1982.

[32] Chen, C. H. Digital Waveform Processing and Recognition. CRC Press, Boca Raton, Florida, 1982.

[33] Chen, C.H. Pattern Recognition and Artificial Intelligence: Joint Workshop on Pattern Recognition and Artificial Intelligence. Academic Press, New York, 1976.

[34] Cheng, G. C. Symposium on Automatic Photointerpretation, Washington, D.C., Pictorial Pattern Recognition: Proceedings. Thompson Book Company, Washington, D.C., 1968.

[35] Chiaraviglio, L. Toward a Syntax-Monitored Semantic Pattern Recognition. School of Information Science, Georgia Institute of Technology, Atlanta, Georgia, 1969.

[36] Chien, Y. Interactive Pattern Recognition. M. Dekker, New York, 1978.

[37] Clark, C. Image Understanding System II. Society of Photo-Optical Instrumentation Engineers, Bellington, Washington, 1980.

[38] Conference on Computer Graphics, Pattern Recognition, and Data Structure, UCLA, IEEE Computer Society, and ACM Special Interest Group on Computer Graphics. Proceedings of the Conference on Computer Graphics, Pattern Recognition, & Data Structure. Institute of Electrical and Electronics Engineers, New York, 1975.

[39] Corbett, F. J. Image Pattern Recognition: Algorithm Implementations, Techniques, and Technology. SPIE--- The International Society for Optical Engineering, Bellingham, Washington, 1987.

[40] De Mori, R. Computer Models of Speech Using Fuzzy Algorithms. Plenum Press, New York, 1983.

[41] Devijver, P. and J. Kettler. Pattern Recognition: A Statistical Approach. Prentice-Hall International, Englewood Cliffs, New Jersey, 1982.

[42] Devijver, P. A. and J. Kittler. Pattern Recognition Theory and Applications. Springer-Verlag, New York, 1987.

[43] Dodwell, P. C. Visual Pattern Recognition. Holt, Rinehart and Winston, New York, 1970.

[44] Dodwell, P. C. Perceptual Processing; Stimulus Equivalence and Pattern Recognition. Appleton-Century-Crofts, New York, 1971.

[45] Ferrari, L. A. and Society of Photo-Optical Instrumentation Engineers. International Symposium on Pattern Recognition and Acoustical Imaging. SPIE---The International Society for Optical Engineering, Bellingham, Washington, 1987.

[46] Fu, K. S. and Japan-U.S. Seminar on the Learning Process in Control Systems. Pattern Recognition and Machine Learning; Proceedings. Plenum Press, New York, 1971.

[47] Fu, K. S. Sequential Methods in Pattern Recognition and Machine Learning. Academic Press, New York, 1968.

[48] Fu, K. S. Syntactic Methods in Pattern Recognition. Academic Press, New York, 1974.

[49] Fu, K. S. and T. M. Cover. Digital Pattern Recognition. Springer-Verlag, New York, 1976.

[50] Fu, K. S. and J. E. Albus. Syntactic Pattern Recognition,Applications. Springer-Verlag, New York, 1977.

[51] Fu, K. S. and A. B. Whinston. Pattern Recognition Theory and Application: (Proceedings of the NATO Advanced Study Institute on Pattern Recognition Theory and Application). Noordhoff, Leyden, 1977.

[52] Fu, K. S. and T. M. Cover. Digital Pattern Recognition. Springer-Verlag, New York, 1980.

[53] Fu, K. S. and T. S. Yu. Statistical Pattern Classification using Contextual information. Research Studies Press, New York, 1980.

[54] Fu, K. S. Applications of Pattern Recognition. CRC

Press, Boca Raton, Florida, 1982.

[55] Fu, K. S. Syntactic Pattern Recognition and Applications. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[56] Fu, K. S. and D. P. Agrawal. VLSI for Pattern Recognition and Image Processing. Springer-Verlag, New York,1984.

[57] Fukunaga, K. Introduction to Statistical Pattern Recognition. Academic Press, New York, 1972.

[58] Gelsema, E. and L. N. Kanal. Pattern Recognition in Practice: Proceedings of an International Workshop Held in Amsterdam. Elsevier Science Publishing Company, New York, 1980.

[59] Gelsema, E. S. and L. N. Kanal. Pattern Recognition in Practice II: Proceedings of an International Workshop Held in Amsterdam. Elsevier Science Publishing Company, New York, 1986.

[60] Getty, D. J. and J. H. Howard. Auditory and Visual Pattern recognition. L. Erlbaum, Hillsdale, New Jersey 1981.

[61] Gilmore, J. Applications of Artificial Intelligence. SPIE---The International Society for Optical Engineering, Bellingham, Washington, 1984.

[62] Gonzalez, R. C. and M. C. Thomason. Syntactic Pattern Recognition: An introduction. Addison-Wesley Publishing Company, Reading, Mass., 1978.

[63] Grasselli, A. Automatic Interpretation and Classification of Images; A Nato Advanced Study Institute. Academic Press, New York, 1969.

[64] Grenander, U. Pattern Synthesis. Springer-Verlag, New York,1976.

[65] Grenander, U. Pattern Analysis. Springer-Verlag, New York, 1978.

[66] Griffiths, M. and C. Palissier. Algorithmic Methods for AI. Chapman and Hall, 1987.

[67] Haken, H. Pattern Formation by Dynamic Systems and Pattern Recognition: Proceedings of the International Symposium on Synergetics at Schloss Elmau, Bavaria. Springer-Verlag, New York, 1979.

[68] Hall, E. L. Computer Image Processing and Recognition. Academic Press, New York, 1979.

[69] Hansen, J. C. Emphasis on Individual Frame Distances in isolated word recognition. Georgia Institute of Technology, Atlanta, Georgia, 1981.

[70] Haralick, R. M. Pictorial Data Analysis. Springer-Verlag, New York, 1983.

[71] IEEE Computer Society and Machine Intelligence and Pattern Analysis Committee. IEEE Computer Society Conference on Pattern Recognition and Image Processing: Proceedings at Rensselaer Polytechnic Institute. The Society, Long Beach, California, 1977.

[72] IEEE Computer Society Conference on Pattern Recognition and Image Processing (3d: 1979: Chicago). Proceedings, PRIP, 79, Chicago. Institute of Electrical and Electronics Engineers, New York, 1979.

[73] IEEE Computer Society. IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management. IEEE Computer Society Press, Los Angeles,California, 1981.

[74] IEEE Computer Society Conference on Pattern Recognition and Image Processing (1981: Dallas, Texas.). Proceedings / PRIP 81, August 3-5, 1981, Dallas, Texas Texas. IEEE Computer Society Press, Los Angeles, California, 1981.

[75] IEEE Computer Society Conference on Pattern Recognition and Image Processing (1982: Las Vegas, Nevada). Proceedings / PRIP 82. June 14-17, 1982, Las Vegas, Nevada. Institute of Electrical and Electronics Engineers, New York, 1982.

[76] IEEE Computer Society International Symposium on Medical Imaging and Image Interpretation (1st: 1982: Berlin, Germany). Proceedings / ISMIII. IEEE Computer Society Press, Silver Spring, MD., 1982.

[77] IEEE Computer Society. 1983 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management: Pasadena, California, October 12 - 14, 1983. Computer Society Press, Silver Springs, MD, 1983.

[78] IEEE Computer Society Conference on Computer Vision and Pattern Recognition (1983: Washington, D.C.). Proceedings /CVPR 83. IEEE Computer Society Press,

Silver Springs, MD, 1983.

[79] IEEE Computer Society Workshop on Visual Languages (1984: Hiroshima-shi, Japan), Technical Committee on Machine Intelligence and Pattern Analysis, Software Engineering Technical Committee, and IEEE Computer Society, Tokyo Chapter. 1984 IEEE Computer Society Workshop on Visual Languages: December 6-8,1984, Hiroshima, Japan. IEEE Computer Society Press, Silver Springs, MD, 1984.

[80] IEEE Computer Society Conference on Computer Vision and Pattern Recognition (1985: San Francisco, California). Proceedings / CVPR '85. IEEE Computer Society Press, Silver Springs, MD, 1985.

[81] IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management (3rd 1985: Miami Beach, Florida). 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, Miami Beach, Florida, November 18-20, 1985. IEEE Computer Society Press, Washington, D. C.,1985.

[82] IEEE Computer Society Conference on Computer Vision and Pattern Recognition (1986: Miami Beach, Florida). Proceedings / CVPR '86. IEEE Computer Society Press, Washington, D. C., 1986.

[83] IEEE Computer Society and Machine Intelligence and Pattern Analysis Committee. IEEE Computer Society Conference on Pattern Recognition and Image Processing, May 31-June 2, 1978. Institute of Electrical and Electronics Engineers, New York, 1978.

[84] Instytut, K. and V. A. Kovalevskii. Character Readers and Pattern Recognition. Spartan Book, New York, 1968.

[85] International Conference on Computer Vision (1st:1987: London, England) and IEEE Computer Society. Proceedings, first International Conference on Computer Vision: June 8-11, 1987. IEEE Computer Society Press, Washington, D.C., 1987.

[86] International Conference on Robot Vision and Sensory Controls (1st: 1981 Stratford-Upon-Avon, U.K.), IFS LTd., and British Pattern Recognition Association. Proceedings of the 1st International Conference on Robot Vision and Sensory Controls, April 1-3, 1981. IFS (Publications)LTD., Bedford, England, 1981.

[87] Joint Workshop on Pattern Recognition and Artificial Intelligence (1976: Hyannis, Mass.) and IEEE Computer Society. Conference Record / 1976 Joint Workshop on Pattern Recognition and Artificial Intelligence. The Society, Long Beach, California, 1976.

[88] Jurs P. C. and T. L. Isenhour. Chemical applications of Pattern Recognition. John Wiley and Sons, New York, 1975.

[89] Kanade, T. Computer Recognition of Human Faces. Birkhauser, Basel, 1977.

[90] Kanal, L. N. and IEEE Workshop on Pattern Recognition (1st: 1966: Dorado P. R.) Pattern Recognition: Proceedings. Thompson, Washington D. C., 1968.

[91] Kanal. L. N. and A. Rosenfeld. Progress in Pattern Recognition. North-Holland Publishing Company, New York, 1981.

[92] Kandel, A. Fuzzy Techniques in Pattern Recognition. John Wiley and Son, New York, 1982.

[93] Kittler, J. and M.J. Duff. Image Processing System Architectures. John Wiley and Sons, New York, 1985.

[94] Klinger, A., K. S. Fu, and T. Kunii. Data Structures, Computer Graphics, and Pattern Recognition. Academic Press, New York, 1977.

[95] Kovalevskii, V. A. Image Pattern Recognition. Springer-Verlag, New York, 1980.

[96] Krishnaiah, P. R. and L. N. Kanal Classification, Pattern Recognition, and Reduction of Dimensionality. Elsevier Science Publishing Company, New York, 1982.

[97] Lainiotis, D. G. and N. S. Tzannes. A Selection of Papers from INFO II, the Second International Confererence on Information Sciences and Systems. D. Reidel Publishing Company, Boston, 1980.

[98] Latombe, J. C. Artificial Intelligence and Pattern Recognition in Computer Aided Design: Proceedings of the IFIP Working. North-Holland Publishing Company, New York, 1978.

[99] Liu, H. and P. S. Schenker. Optical and Digital Pattern Recognition. The Society, Bellingham, Washington, 1987.

[100] Martelli, A. Edge Detection Using Heuristic Search Methods. Computer Graphics and Image Processing. (April 1972) Vol. 1: 169-182.

[101] Martelli, A., An Application of Heuristic Search Methods to Edge and Contour Detection. Communication of The ACM. (February, 1976), Vol. 19: 2: 73-83.

[102] McGillem, C. D. and D. B. Morrison Symposium on Machine Processing of Remotely Sensed Data. Institute of Electrical and Electronics Engineers, New York, 1975.

[103] Meisel, W. S. Computer-Oriented Approaches to Pattern Recognition. Academic Press, New York, 1972.

[104] Mendel, J. M. and K. S. Fu. Adaptive, Learning, and Pattern Recognition Systems; Theory and Application tions. Academic Press, New York, 1970.

[105] Miclet, L. Structural Methods in Pattern Recognition. Springer-Verlag, New York, 1986.

[106] Montanari, U. On the Optimal Detection of Curves in Noisy Pictures. Communications of the ACM. (May 1971) 14: 5: 335-345.

[107] Morin, T. L. Computational Advances in Dynamics Programming. Academic Press, New York, 1978.

[108] Nagao, M. A Structural Analysis of Complex Aerial Photographs. Plenum Press, New York, 1980.

[109] Nemhauser, G. L. Introduction to Dynamic Programming. John Wiley and Sons, 1966.

[110] Nevatia, R. Computer Analysis of Scenes of 3-Dimensional Curved Objects. Birkhauser, Basel, 1976.

[111] Niemann, H. Pattern Analysis. Springer-Verlag, New York, 1981.

[112] North Atlantic Treaty Organization. Advisory Group for Aerospace Research and Development. Aerospace Medical Panel. Pattern Recognition. Body Armour and Aircrew Equipment Assemblies, Current Space Medical Problems. Aeromedical Evacuation. North Atlantic Treaty Organization, 1968.

[113] Oja, E. Subspace Methods of Pattern Recognition. John Wiley and Sons, New York, 1983.

[114] Pal, S. K. and D. K. Dutta Majumder. Fuzzy Mathematical Approach to Pattern recognition. John Wiley and Sons, New York, 1986.

[115] Paris D. T. Computer Aided Detection/Classification Identification for Sonar in Mine Hunting. School of Electrical Engineering, Georgia Institute of Technology, Atlanta, Georgia, 1986.

[116] Patrick, E. A. and J. M. Fattu. Artificial Intelligence with Statistical Pattern Recognition. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[117] Patrick, E. A. Fundamentals of Pattern Recognition. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[118] Pau, L. F. Failure Diagnosis and Performance Monitoring. M. Dekker, New York, 1981.

[119] Pau, L. F. An Introduction to Infrared Image Acquisition and Classification Systems. John Wiley and Sons, New York, 1983.

[120] Pavlidis, T. Algorithms for Graphics and Image Processing. Computer Science Press, Rockville, MD, 1982.

[121] Pavlidis, T. Structural Pattern Recognition. Springer-Verlag, New York, 1977.

[122] Pearl, J. Heuristics Intelligent Search Strategies for Computer Problem Solving. Addison -Wesley Pub lishing Company, Reading, Massachusetts, 1984.

[123] Preston, K. and M. J. B. Duff. Modern Cellular Automata: Theory and applications. Plenum Press, New York, 1984.

[124] Pugh, A. Robot Sensors. Springer-Verlag, New York, 1986.

[125] Pungor, E. and G. E. Veress. Modern Trends in Analytical Chemistry. Elsevier Science Publishing Company, New York, 1984.

[126] Rand, R. S. and Army Engineer Topographic Laboratories. Texture Analysis and Cartographic Feature Extraction. U.S. Army Corps of Engineers, Fort Belvoir, Virginia 1985.

[127] Reed, S. K. Psychological Processes in Pattern Recognition. Academic Press, New York, 1973.

[128] Ronse, C. and P. Devijver. Connected Components in Binary Images: The Detection Problem. John Wiley and Sons, New York, 1984.

[129] Rosenfeld, A. Picture Languages: Formal Models for Picture Recognition. Academic Press, New York, 1979.

[130] Rosenfeld, A. Techniques for 3-D Machine Perception. Elsevier Science Publishing Company, New York, 1986.

[131] Schwab, E. C. and H. C. Nusbaum. Pattern Recognition by Humans and Machines. Academic Press, Orlando, Florida, 1986.

[132] Sebestyen, G. S. Decision-making Processes in Pattern Recognition. Macmillan, New York, 1962.

[133] Sklansky, J. and G. N. Wassel. Pattern Classifiers and Trainable Machines: with 117 Illustrations. L. Erlbaum, Hillsdale, New Jersey, 1981.

[134] Suen, C. Y. and R. De Mori. Computer Analysis and Perception. Crc Press, Boca Raton, Florida, 1982.

[135] Taylor, J. R. Tactile Sensing: A Case Study of the Lord Corporation LTS-300T. Georgia Institute of Technology, Atlanta, Georgia, 1987.

[136] Tendam, I. M. and D. B. Morrison. Fifth Annual Symposium (on) Machine Processing of Remotely Sensed Data. Institute of Electrical and Electronics Engineers, New York, 1979.

[137] Tou, J.T. and R. C. Gonzalez. Pattern Recognition Principles. Addison-Wesley Publishing Company, Reading, Mass.,1974.

[138] Uhr, L. M. Pattern Recognition, Learning , and Thought: Computer-Programmed Models of Higher Mental Processes. Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[139] Uhr, L. M. Pattern Recognition: Theory, Experiment, Computer Emulations, and Dynamic Models of Form Perception and Discovery. John Wiley and Sons, New York, 1966.

[140] Ullmann, J. R. Pattern Recognition Techniques. Crane,Russak, New York, 1973.

[141] Varmuza, K. Pattern Recognition in Chemistry. Springer-Verlag, New York, 1980.

[142] Watanabe, S. Frontiers of Pattern Recognition: The Proceedings. Academic Press, New York, 1972.

[143] Watanabe, S. Methodologies of Pattern Recognition; {Proceedings}. Academic Press, New York, 1969.

[144] Watanabe, S. Pattern Recognition: Human and Mechanical. John Wiley and Sons, New York, 1985.

[145] Wolff, D. D. and M. L. Parsons. Pattern Recognition Approach to Data Interpretation. Plenum Press, New York, 1983.

[146] Young, T. Y. and K. S. Fu. Handbook of Pattern Recognition and Image Processing. Academic Press, Orlando, Florida, 1986.

[147] Young, T. Y. and T. W. Calvert. Classification, Estimation, and Pattern Recognition. American Elsevier Publishing Company, 1974.

[148] Zimmerman, N. J. Proceedings of the 5th International Conference on Robot Vision and Sensory Controls. Elsevier Science, New York, 1985.