Finite State Properties of Bounded Pathwidth Graphs

by

Xiuyan Lu

B.Sc., Beijing University of Aeronautics and Astronautics, 1984
M.Sc., Beijing University of Aeronautics and Astronautics, 1987

A Project Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this project report as conforming
to the required standard

[redacted]

Dr. M. Fellows, Supervisor (Department of Computer Science)

[redacted]

Dr. M. Serra, Departmental Member (Department of Computer Science)

[redacted]

Dr. V. Bhargava, Outside Member (Dept. of Electrical & Computer Engineering)

Supervisor: Dr. Michael Fellows

Abstract

The concepts of the *treewidth* and *pathwidth* of combinatorial structures have come to play a central role in a number of important developments in discrete mathematics and combinatorial algorithm design. In particular, they are of fundamental importance to several general algorithm design techniques that provide an interesting line of attack for many problems that are NP-complete.

This approach to algorithm design involves two basic steps: (1) the representation of the combinatorial structure by a string or tree of symbols over a finite alphabet (termed a *parse* of the structure), and (2) finite-state recognition of these representations.

The project reported on here explores several issues regarding this approach to algorithm design that must be adequately resolved in order for these general theoretical developments to deliver useful algorithms. In particular, these issues concern several junctures where large hidden constants may hinder this approach. Since most of the tools of theory are relatively insensitive to constants, the project has centered on an experimental methodology, and on developing general software to support the exploration of these issues. In this setting, the experiments necessarily involve some preliminary theoretical work.

There are three main results of this project. First, we identify a set of structural operators that are sufficient to parse graphs of bounded pathwidth, and prove this sufficiency. Secondly, based on this parsing scheme we have implemented a general software system for

studying finite-state recognition of families of graphs of bounded pathwidth represented as structural parses. Finally, we have used this software to explore the finite-state recognition of Hamiltonian graphs, obtaining data on the size of the resulting finite-state automata for the pathwidth bounds 2 and 3.

Examiners:

[redacted]

Dr. M. Fellows, Supervisor (Department of Computer Science)

[redacted]

Dr. M. Serra, Departmental Member (Department of Computer Science)

[redacted]

Dr. V. Bhargava, Outside Member (Dept. of Electrical & Computer Engineering)

Acknowledgements

I would like to express my greatest gratitude and appreciation to my supervisor, Dr. Michael Fellows for his encouragement and guidance throughout my graduate studies and this project work. I would like to thank him for the financial assistance made available by him without which this research may not have been possible.

A special acknowledgement is due to Michael Dinneen for his helpful suggestions.

Thanks to all my friends for their friendship and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The project that is the subject of this report addresses several problems concerning graphs of bounded pathwidth. The main goal has been to develop a software platform for the experimental study of finite-state graph properties and recognizers.

This introductory chapter is organized as follows. In the first section we review the background and motivation for the project in general terms, and survey the main results. In succeeding sections we review basic concepts and notation concerning graphs and automata that are employed in the chapters that follow.

1.1 The motivation for the project and the main results

In recent years, the concepts of the *treewidth* and *pathwidth* of graphs (and other more general combinatorial structures) have assumed a role of central importance in graph theory, combinatorics and graph algorithms. There are basically two reasons for this:
(1) The concepts of *treewidth* and *pathwidth* play a central structural role in the proof of

the *Graph Minor Theorem* (as the solution of Wagner's conjecture has come to be called), by Robertson and Seymour [25,26,27,28]. This is clearly one of the deepest and most beautiful results of mathematics in this century, having (on the side) profound implications for computational complexity theory. Moreover, this major breakthrough has opened up an area of research with tremendous prospects for further development.

(2) A fundamental way of coping with NP-completeness is to restrict the problem instances. For graph problems, the restriction to bounded treewidth (or pathwidth) has proved to be a general notion that unifies a wide variety of approaches to restricting input for problems about graphs. Moreover, while "most graph problems are NP-complete in general," it has been shown, as an accumulated result of a very large recent literature [4,5,6,9,13,15,17,23] that, "most problems are easy for graphs of bounded treewidth or pathwidth."

The starting point for this project are the results of [2] showing that essentially all of the wide variety of approaches in the very large literature on algorithms for graphs of bounded treewidth and pathwidth can be viewed in an elegant unified framework based on finite automata. The results of [2] also show that well-quasi-ordering results such as the Graph Minor Theorem are intrinsically related to issues concerning finite-state recognition for graphs of bounded treewidth and pathwidth.

There is the distinct possibility that these recent theoretical developments may lead to novel and useful algorithmic approaches for a number of natural problems that exhibit "bounded combinatorial width" including problems in VLSI layout, computational biology, and natural language processing. This possibility is being active explored both

theoretically and in prototype system implementations [10]. The most significant remaining difficulties concern "hidden constants" in the theoretical basis. Theoretical tools for addressing questions about hidden constants tend to be both pessimistic and insensitive. The primary purpose of this project has been to explore some of these issues experimentally, and most importantly, to create a software environment to support the needed empirical exploration, in the setting of bounded pathwidth.

The finite-state point of view concerning algorithms for graphs of bounded pathwidth can be given the following high level description. The algorithms proceed according to the following two steps:

Step 1: The input graph is *parsed* into a representation as a string of symbols. The symbols represent a sequence of structural operations by which the graph can be built.

Step 2: The graphs having the property of interest (for example, Hamiltonicity) are recognized by means of a finite-state automata operating on the parsed representations. The automaton should accept precisely, and only, the strings that are parses of graphs having the property.

This project has focused on the following questions that immediately arise in this program:

Question 1: How many symbols are needed to parse the graphs of a given pathwidth?

Question 2: For familiar graph properties for which the above algorithmic program can in principle be carried out, how large are the resulting automata?

We remark that while these are clearly important questions, there are also other important questions which this project does not address. In investigating these questions,

this project has achieved the following.

Main Result 1. A sufficient set of parsing operators is identified for graphs of bounded pathwidth. This basic result is described in Chapter 2.

Main Result 2. Based on the identified parsing operators, a system has been implemented by which the automaton for a graph property can be automatically compiled from the information: (1) a decision subroutine for the property, and (2) a finite-state test set for the property. Test sets are explained in Chapter 3, and pathwidth 2 and pathwidth 3 test sets for the property of Hamiltonicity are identified. The organization and capabilities of the system are discussed in Chapter 4.

Main Result 3. Numerical results on Hamiltonicity automata for pathwidth bounds 2 and 3 are obtained by using the automata compiler. These results are described in Chapter 4.

Chapter 5 concludes this report with a discussion of some of the remaining open problems and areas needing further investigation.

1.2 Graph theoretic preliminaries

We will use the standard notation $G = (V, E)$, where G is a *graph* with vertex set V and edge set E. We refer to elements of V as *vertices* and to elements of E as *edges.* Each edge is an unordered pair of vertices. The *order* of a graph $G = (V, E)$ is $|V|$, and the *size* of this graph is $|E|$. If both $|V|$ and $|E|$ are finite, then the graph is said to be *finite graph.* A graph is called *simple* if it has no loop, and if there is never more than one edge between any two vertices. Two different vertices are called *adjacent* if they are both incident with the same edge. For each vertex x we denote by $N(x)$ the *neighborhood* of

x, the set of vertices which are adjacent to x. Unless stated otherwise we assume that all graphs in this project are simple and finite.

Let $G = (V, E)$ be a graph. A *subgraph* of G is a graph $H = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. If $W \subseteq V$ is a subset of vertices then $G[W]$ denotes the subgraph induced by W, i.e., $G[W]$ is the subgraph with vertex set W in which two vertices are adjacent whenever they are adjacent in G. If $W = \{v_1, ..., v_s\}$ then we also use the notation $G[v_1, ..., v_s]$ for $G[W]$. If W is a subset of vertices such that every pair of vertices is adjacent, then W is called a *clique* in G. The *clique size* of a clique W in G is the number of vertices of W.

A *walk* is an alternating sequence of vertices and edges, starting and ending with a vertex, in which every edge is incident with the two vertices immediately preceding and following it. If all vertices of a walk are distinct (and hence also all edges in the walk are distinct) the walk is called a *path* . If the number of vertices in a path is equal to the order of the graph, then the path is called a *Hamiltonian path.* If the walk is closed, i.e., the starting and ending vertex are the same, and if all vertices in the walk are distinct, then the walk is called a *cycle.* If the number of different vertices in a cycle is equal to the order of the graph, then the cycle is called a *Hamiltonian cycle.* If a graph has a Hamiltonian cycle, then the graph is called a *Hamiltonian graph.*

1.3 Background on languages and automata

An *alphabet* is a finite, nonempty set of elements. The elements of the alphabet Σ are usually called *symbols* . A *word over an alphabet* Σ is a finite sequence of symbols from Σ. The *empty word* is the empty sequence and is denoted by λ. The empty word is a word

over every alphabet. The *universal language* of all words over an alphabet Σ is denoted by Σ^*. The set Σ^* is always infinite. The *length* of a word x is denoted by $|x|$ and is simply the number of symbols in the given word. Let x be a word over an alphabet Σ and let a be in Σ. Then the *a-length* of x is denoted by $|x|_a$ and is the number of times a occurs in x. Given two words x and y over an alphabet Σ, the *catenation of x with y,* denoted by xy, is the word obtained by appending the word y to x. Let Σ be an alphabet. For all words, x, y, and z over Σ, $(xy)z = x(yz) = xyz$ where parentheses are used to indicate which catenation operation is carried out first. In other words, catenation is associative.

Since a language is not an arbitrary set, but is a subset of Σ^*, for some alphabet Σ, we might expect that there is definitional method peculiar to languages. A recognizer defines a language by providing an algorithm to recognize words in the language. A black box view of a recognizer is given in Figure 1.1.

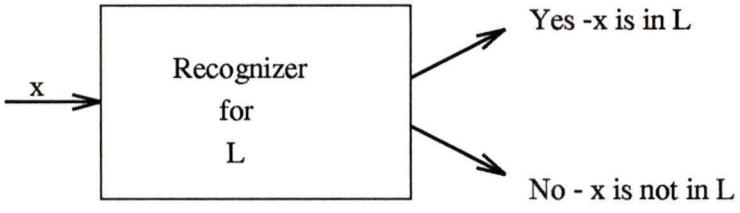


Figure 1.1: A black box view of a recognizer

There are several ways to describe the recognizer, such as finite automata, pushdown automata and Turing machines. In this project, we describe the recognizer as a finite automata.

A *deterministic finite automata (DFA), M,* is specified by a quintuple $M = (Q, \Sigma, \delta, s, F)$, where Q is an alphabet of state symbols; Σ is an alphabet of input symbols; δ: $Q \times \Sigma \rightarrow Q$

is a transition function; s in Q is the start state; and $F \subseteq Q$ is a set of final states.

Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. We say that a word in $Q\Sigma^*$ is a *configuration* of M. It represents the current state of M and the remaining unread input of M. If px and qy are two configurations of M, then we write $px \vdash qy$, if $x = ay$ for some a in Σ, and $\delta(p, a) = q$. We say px *yields* qy. Observe that $\vdash$ is a binary relation over $Q\Sigma^*$. For $k \geq 1$, we write $px \vdash^k qy$ (k-steps of M on px) if either $k = 1$ and $px \vdash qy$ or $k > 1$ and there exists a configuration rz such that $px \vdash rz$ and $rz \vdash^{k-1} qy$. We write $px \vdash^+ qy$, where $\vdash^+$ is the transitive closure of $\vdash$, if $px \vdash^k qy$, for some $k \geq 1$. In a similar manner we define $\vdash^*$, the reflexive closure of $\vdash$. We write $px \vdash^* qy$ if either $px = qy$ or $px \vdash^+ qy$. We say that the sequence of configurations given by $px \vdash^* qy$ is a *configuration sequence.*

Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. We say a word x in Σ^* is *accepted* by M if $sx \vdash^* f$, for some f in F. We say that $sx \vdash^* f$ is an *accepting configuration sequence.* A word that is not accepted is said to be *rejected.* The set of words accepted by M, called the *language accepted, defined, or recognized by* M is denoted by $L(M)$ and is defined as $L(M)$= $\{x$: x is in Σ^* and $sx \vdash^* f$, for some f in F $\}$.

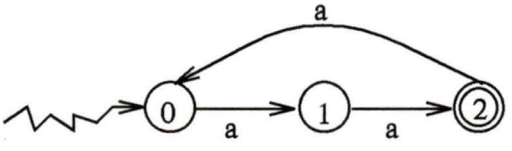


Figure 1.2: An example of finite state automaton

Let $M = (Q, \Sigma, \delta, s, F)$ be defined by $Q = \{0, 1, 2, 3, 4\}, \Sigma = \{a, b\}$, $s = \{0\}$, $F = \{2\}$, and $\delta(0, a) = 1, \delta(1, a) = 2, \delta(2, a) = 0$. The state diagram of M is displayed in Figure 1.2, where the start sate is indicated by an incoming wavy arrow and each final state is

drawn as a double circle. We get $L(M) = a^{2+3n}$ for all $n \geq 0$.

1.4 Background on pathwidth

A *path-decomposition* of a graph $G = (V, E)$ is a sequence X_1, X_2,..., X_r of subsets of V that satisfy the following conditions: $\bigcup_{1 \leq i \leq r} X_i = V$; for every edge $(u, v) \in E$, there exists an X_i, $1 \leq i \leq r$, such that $u \in X_i$ and $v \in X_i$; for all i, j, k and $1 \leq i < j < k \leq r$, $X_i \bigcap X_k \subseteq X_j$. The *pathwidth of a path-decomposition* X_1, X_2,...,X_r of a graph G is $\max_{1 \leq i \leq r} |X_i| - 1$. The *pathwidth of a graph* G, denoted pw(G), is the minimum pathwidth over all path-decomposition of G.

In figure 1.4, an example of a graph with pathwidth 2 is given, together with a path-decomposition of it.

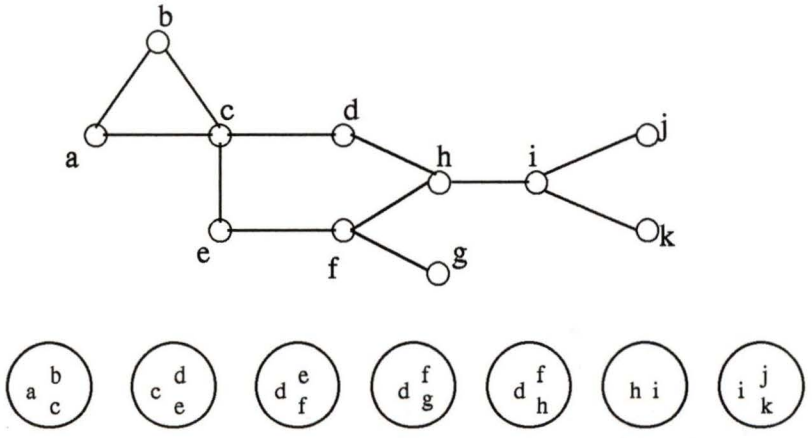


Figure 1.3: Example of a graph with path-decomposition

The problems of determining the pathwidth of a graph, and of computing a path-decomposition of optimal width are NP-hard, but if k is a fixed constant the problems are solvable in polynomial time. The first algorithms developed to solve these problems for

fixed k were based on dynamic programming and required $O(n^{k+2})$ [5] and $O(n^{2k^2+4k+8})$ [21] time, respectively. As a by-product of their work on Wagner's conjecture, Robertson and Seymour showed the existence of $O(n^2)$ algorithms (for fixed k) [26]. Ton Kloks showed in his Ph.D. Dissertation [24] that for each k there exists an $O(nlogn)$ algorithm that, given a graph $G = (V, E)$, decides whether the pathwidth of G is at most k and, if it is, constructs a path-decomposition of width at most k. Recently, Bodlaender has developed a linear time algorithm for this problem [8].

Chapter 2

Structural Parsing of a Bounded Pathwidth Graph

In this chapter we describe a scheme for parsing bounded pathwidth graphs. We prove that the operator set identified is sufficient to parse all graphs of a given pathwidth bound. As a consequence, every bounded pathwidth graph can be represented by a string of symbols representing a sequence of operators. For the set of operators that we identify we will show conversely that every sequence of operators over this set generates a bounded pathwidth graph. This representational mechanism provides the fundamental basis for the finite-state approach to graph decision algorithms for graphs of bounded pathwidth, and is utilized in the algorithm compilation system that is the focus of this project.

2.1 Preliminaries

Definition 2.1 *For a positive integer k, a k-simplex S of a graph $G = (V, E)$ is an injective map $\partial_S : \{1, 2, ..., k\} \rightarrow V$.*

Definition 2.2 *A k-boundaried graph $B = (G, S)$ is a graph G together with a k-simplex S for G. Vertices in the range of ∂_S are called boundary vertices.*

Definition 2.3 *An empty k-boundaried graph is a k-boundaried graph $B = (G, S)$ where $G = (V, E)$ and $|V| = k$, $|E| = 0$.*

A fundamental operation (denoted $\oplus$) on k-boundaried graphs that we consider is that of gluing them together along their boundaries by identifying like-labeled vertices. Formally:

Definition 2.4 [19] *If $B = (G, S)$ and $B' = (G', S')$ are k-boundaried graphs, then $B \oplus B'$ denotes the k-boundaried graph obtained from the disjoint union of the graphs $B = (G, S)$ and $B' = (G', S')$ by identifying each vertex u in the range of ∂_S with the vertex v in the range of $\partial_{S'}$ for which $\partial_S(u) = \partial_{S'}(v)$. (In some situations, which will be clear from the context, we consider $B \oplus B'$ to be an ordinary graph, by "forgetting" the boundary.)*

At the heart of the automata-theoretic approach to bounded pathwidth is the representation of graphs of bounded pathwidth as strings of symbols, where the symbols are taken from a finite alphabet and represent structural primitives and operations for generating the graph. Such a string we refer to as a *structural parse* for the graph. A general useful notion for such representation schemes is that of a composition operator for boundaried graphs.

The idea of a boundaried graph composition operator is quite simple: the operator is described by a fixed *operator graph* that has several boundaries (not necessarily disjoint). For example, for an unary operator, the operator graph is equipped with 2 boundaries, one for the argument to the operator, and the other to be the boundary of the resulting graph.

Definition 2.5 [19] *An n-ary composition operator* $\otimes$ *is defined by the data*

1. *A k-boundaried graph* $B_\otimes = ((V_\otimes, E_\otimes), S_\otimes)$;

2. *Injective maps* $(\partial_S)_i$: $\{1, ..., k\} \to V_\otimes$ *for* $i = 1, 2, ..., n$.

For the binary case, if B_i, for $i = 1, 2$, is a pair of k-boundaried graphs $B_i = ((V_i, E_i), S_i)$ then $B_1 \otimes B_2$ is defined to be the k-boundaried graph for which the ordinary underlying graph is formed from the disjoint union of B_1, B_2 and $B_\otimes$ by identifying each vertex u of S_i (for $i = 1, 2$) with its image $\partial_{S_i}(u)$ in $V_\otimes$. The boundary set and the labeling for $B_1 \otimes B_2$ is given by $S_\otimes$ and $\partial_{S_\otimes}$.

Definition 2.6 *Suppose* $\mathcal{F}$ *is a class of graphs. For two k boundaried graphs* X *and* Y, $X \sim_{\mathcal{F}} Y$ *if and only if, for every k-boundaried graph* Z, $X \oplus Z \in \mathcal{F} \iff Y \oplus Z \in \mathcal{F}$.

Definition 2.7 *A family of* $\mathcal{F}$ *of graphs is finite state for a parsing scheme* Ω *(operator set* Σ*) if and only if there exists a finite state machine* M *that accepts* $x \in \Sigma^*$ *if and only if* x *is a parse of a graph in* $\mathcal{F}$.

Theorem 2.1 [2] *If* Ω_k *is a parsing scheme of k-boundary operators (operator set* Σ_k*) for graphs of pathwidth at most k, then a graph family* $\mathcal{F}$ *is finite state if and only if* $\sim_{\mathcal{F}}$ *has finite index.*

Theorem 2.2 [2] *The family $\mathcal{F}$ of Hamiltonian graphs is finite state.*

We remark that the property of Hamiltonicity is by no means special in being finite-state for bounded pathwidth. In fact, there are only a few known natural properties of graphs which are *not* finite-state for bounded pathwidth. The reference [2] surveys most of what is known in this regard.

2.2 The parsing scheme

In this section, we describe a set of operators, Σ_k, which has the following properties: For any given graph $G = (V, E)$, if the pathwidth of G, $PW(G) \leq k$, then the graph G can be generated by some parse $(g_1, g_2, ..., g_n) \in \Sigma_k^*$; and for any given parse $(z_1, z_2, ..., z_m) \in \Sigma_k^*$, if we let H be the graph which is generated by the parse, then the pathwidth of H, $PW(H) \leq k$.

2.2.1 Definitions and lemmas

Graphs of bounded pathwidth have been defined in Chapter 1. For the purposes of this Chapter, it is more convenient to work with an equivalent recursive characterization of graphs of bounded pathwidth.

Definition 2.8 *A graph G is a k-path if there exists a k-boundaried graph $B = (G, S)$ in the following family $\mathcal{F}$ of recursively generated k-boundaried graphs.*

1. *$(K_{k+1}, S) \in \mathcal{F}$ where S is any k-simplex of the complete graph K_{k+1}.*

2. *If* $B = ((V, E), S) \in \mathcal{F}$ *then* $B' = ((V', E'), S') \in \mathcal{F}$ *where* $V' = V \cup \{v\}$ *for* $v \notin V$ *and (a)* $E' = E \cup \{(\partial_S(i), v) \backslash 1 \leq i \leq k\}$;

(b) S' *is defined in one of the following two ways:*

(1) $\partial_{S'}(i) = \partial_S(i)$ *for all* i, *or*

(2) selecting any $j \in \{1, 2, ..., k\}$

$$\partial_{S'}(i) = \begin{cases} v & \text{if } i=j \\ \partial_S(i) & \text{otherwise.} \end{cases}$$

Definition 2.9 *A partial k-path is a k-path with some edges removed.*

Lemma 2.1 *If* G *is a* k*-path then the pathwidth of* G, $PW(G) = k$. *Further if* H *is a partial* k*-path, then the pathwidth of* H, $PW(H) \leq k$.

Proof: We first prove that any k-path G, with defining k-boundaried graph (G, S), has a path-decomposition $X_1, X_2, ..., X_r$ of width k. We argue by structural induction on the definition of a k-path. For the initial case of G being a $(k+1)$-clique, the hypothesis holds, since $X_1 = \{1, 2, ..., k+1\} = V(G)$ is a path-decomposition of width k. For the inductive step, assume $X_1, X_2, ..., X_r$ is the required path-decomposition for G. Let G' be the k-path built from some (G, S) by applying case 2 of the recursive definition. If we set $X_{r+1} = range(S) + v$, and v is a vertex that never occurs in $X_1, X_2, ..., X_r$, then $X_1, X_2, ..., X_r, X_{r+1}$ is the required path-decomposition of G', since the following hold: (1) the new vertex v is in X_{r+1}; (2) all new edges (u, v) in $E(G')\backslash E(G)$ have $u \in X_{r+1}$ and $v \in X_{r+1}$; (3) $X_i \bigcap X_{r+1} \subseteq X_i \bigcap X_r$.

To see that any k-path G has a lower bound of k for its pathwidth, we first notice that G contains a clique C of size $k+1$. For any path-decomposition $X = X_1, X_2, ..., X_r$

of G, let $Y_i = X_i \bigcap V(C)$ for $1 \leq i \leq r$. If any $|Y_i| > k$ then the width of X is at least k. Suppose $|Y_i| \leq k$ for all Y_i. There must then be a Y_i and a Y_j, $1 \leq i < j \leq r$, such that $u \in Y_i \backslash Y_j$ and $v \in V_j \backslash V_i$ for different vertices u and v. Since $(u, v) \in E(C)$, there must be some Y_k, $1 \leq k \leq r$, such that $u \in Y_k$ and $v \in Y_k$. Now $\{u, v\} \not\subseteq Y_i \bigcap Y_j$ so either $k < i$ or $k > j$. But $v \notin Y_k$ for $k \leq i$, and $u \notin Y_k$ for $k \geq j$. This contradicts that X is a path-decomposition (since it fails the edge covering requirement). Therefore, the pathwidth of any k-path is k.

For the second part of the lemma, we begin with a path-decomposition $X_1, X_2, ..., X_r$ of width k for a k-path G from which the partial k-path H has been obtained by some edge deletions. We may simply note that the original path-decomposition X is still a path-decomposition of the partial k-path H. So the pathwidth of H, $PW(H) \leq k$. This proves the lemma. □

Definition 2.10 [10] *A path-decomposition $X_1, X_2, ..., X_r$ of pathwidth k is smooth, if for all $1 \leq i \leq r$, $|X_i| = k + 1$ and for all $1 \leq i < r$, $|X_i \cap X_{i+1}| = k$.*

Theorem 2.3 [10] *Any path-decomposition of minimal width can be transformed to a smooth path-decomposition of the same pathwidth.*

Lemma 2.2 *Every graph of k-bounded pathwidth (as defined in Chapter 1) is a partial k-path.*

Proof: Suppose G is a graph with pathwidth bounded by k. By Theorem 2.3 there is a smooth path-decomposition $X_1, X_2, ..., X_r$ of G, where for all $1 \leq i \leq r$, $|X_i| = k + 1$ and for all $1 \leq i < r$, $|X_i \cap X_{i+1}| = k$.

we first construct a k-path from this smooth path-decomposition of G as follows:

(1) For the initial step:

$(K_{k+1}, S) \in \mathcal{F}$ where S is any k-simplex of the complete graph K_{k+1}.

Let $V(K_{k+1})$=X_1 and

$E(K_{k+1})$=$E(X_1) \cup \{(u,v)\}$. (vertices $u, v \in X_1$ and edge $(u,v) \notin E(X_1)$).

(2) For the inductive step:

If $B = ((V,E),S) \in \mathcal{F}$ then $B' = ((V',E'),S') \in \mathcal{F}$ where $V' = V \cup \{v\}$.

Let $v \notin (X_1 \cup X_2 \cup, ..., \cup X_i)$ and $v \in X_{i+1}$;

(a) $E' = E \cup \{(u,v)\}$ where (edge $(u,v) \in E(X_{i+1})$ and $(u,v) \notin E$) or (vertices $u, v \in X_{i+1}$ and edge $(u,v) \notin E(X_{i+1})$));

(b) $S' = X_{i+1} \cap X_{i+2}$ for $1 \leq i < (r-1)$ or $S' = S$ for $i = r-1$.

Then we can easily get a partial k-path representation of the graph G by deleting all the edges $e \notin E(X_i)$ for $1 \leq i \leq r$ in the above constructed k-path. □

Definition 2.11 [7] *The graphs G and H are isomorphic, $G \simeq H$, if there is a one to one mapping of the vertices of G onto the vertices of H which maps adjacent pairs of vertices in G to adjacent pairs of vertices in H and which also maps nonadjacent pairs of vertices in G to nonadjacent pairs of vertices in H.*

Definition 2.12 *A parse is a sequence of operators $(g_1, g_2, ..., g_n) \in \Sigma_k^*$.*

Definition 2.13 *A graph G is generated by the parse $(g_1, g_2, ..., g_n) \in \Sigma_k^*$ if there is a k-boundaried graph $(B,S) = (g_1, g_2, ..., g_n)$ such that $B \simeq G$.*

Definition 2.14 *Let* $G_n = (g_1, g_2, ..., g_n)$ *be a graph represented as a parse over* Σ_k. *A prefix graph (of length m) of* G_n *is* $G_m = (g_1, g_2, ..., g_m)$, $1 \leq m \leq n$.

Definition 2.15 *Let* $G = g_1, g_2, ..., g_n$ *and* $Z = z_1, z_2, ..., z_m$ *be any two sequences of operators over* Σ_k. *The catenation* $(\cdot)$ *of* G *and* Z *is defined as*

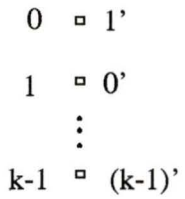
$G \cdot Z = (g_1, g_2, ..., g_n, z_1, z_2, ..., z_m)$

Definition 2.16 *Let* $B\partial_{S_i}$ *be boundary vertices of graph* $G_i = (g_1, g_2, ..., g_i)$. *A prefix boundary vertices* $B\partial_{S_{i-1}}$ *of the operator* $g_i \in G_i$ *are the boundary vertices of graph* $G_{i-1} = (g_1, g_2, ..., g_{i-1})$.

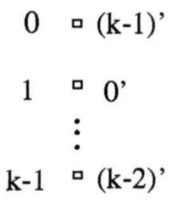
2.2.2 The operator set for the parsing scheme

The operator set Σ_k, by means of which we will represent partial k-paths, consists of the following unary composition operators. Our notational convention is that the argument boundary of an operator graph is labeled 0,1,2,...,k-1, and the boundary of resulting graph is labeled 0',1',2',...,(k-1)'.

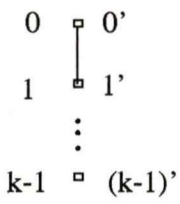
[a]: Switching vertex labels of vertex 0 and vertex 1.



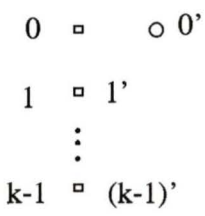
[b]: Cycling vertex labels of vertex 0, vertex 1, ...,vertex $k-1$.



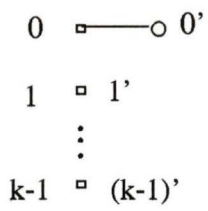
[c]: Adding an edge between vertex labeled as 0 and vertex labeled as 1



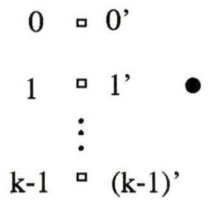
[d]: Extending vertex labeled as 0 without any edge



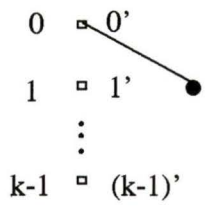
$\boxed{e}$: Extending vertex labeled as 0 with an edge



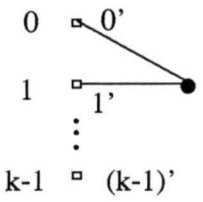
$\boxed{f_0}$: Bulbing a pendant vertex with zero edge.



$\boxed{f_1}$: Bulbing a pendant vertex with one edge.

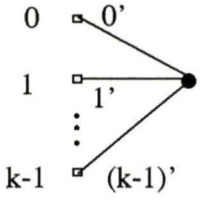


$\boxed{f_2}$: Bulbing a pendant vertex with two edges.



.....

$\boxed{f_k}$: Bulbing a pendant vertex with k edges.



The above parsing scheme involves altogether $k + 6$ operators.

2.2.3 The relation of the operator set to boundaried pathwidth graphs

Lemma 2.3 *Every parse* $G_n = (g_1, g_2, ..., g_n) \in (\Sigma_k)^*$ *represents a graph with a path-decomposition of width* $\leq k$.

Proof: We begin by dividing the operators into two types:

1. O_1={ $\boxed{\text{d}}$, $\boxed{\text{e}}$, $\boxed{f_0}$, $\boxed{f_1}$,...,$\boxed{f_k}$ };

2. O_2={ $\boxed{\text{a}}$, $\boxed{\text{b}}$, $\boxed{\text{c}}$ }.

Let $G_n = (g_1, g_2, ..., g_n)$ be a k-boundaried graph and $|G_n| = m$; $(h_1, h_2, ..., h_m)$ be a

sequence of operators in which $h_i = g_{i_1} \in G_n$ and $h_i \in O_1$, and v_i denote the index of the i-th operator $h_i \in (h_1, h_2, ..., h_m)$. Let I be the set of these v_i. For each $i \in I$, let

X_i = { the prefix boundary vertices $B\partial_{S_{i-1}}$ of operator $h_i = g_{i_1}$ }

$\cup$ { the new vertex introduced by the operator $h_i = g_{i_1}$}

We claim that $X = X_1, X_2, ..., X_{|I|}$ is a path-decomposition of width k.

By the definition of X_i, we get

(1) $|X_i| = k + 1$;

(2) $\bigcup X_i = \{1, 2, ..., |I|\} = m = |G_n|$;

We notice that edges can be added in the following ways

1. between prefix boundary vertices $B\partial_{S_{i-1}}$ of the operator $h_i = g_{i_1} \in O_1$.

2. between prefix boundary vertex $B\partial_{S_{i-1}}$ of the operator $h_i = g_{i_1} \in O_1$ and the new vertex introduced by the operator $h_i = g_{i_1} \in O_1$.

By the definition of X_i, we can get

(3) For each edge $E_i = (v_i, v_j)$ in G_n, there is X_i such that v_i in X_i and v_j in X_i.

$X_i \bigcap X_j$ ={ the boundary vertices that have not been changed from the operator $h_i = g_{i_1} \in O_1$ to the operator $h_j = g_{j_1} \in O_1$, $i < j$ and $i_1 < j_1$ }

So for any k, $i < k < j$, we have $X_i \bigcap X_j \subseteq X_i \bigcap X_k$. This implies that

(4) $X_i \bigcap X_j \subseteq X_k$.

From the definition of the pathwidth of a graph, we know that the pathwidth of any graph described by a parse sequence of operators is at most k. $\square$

Lemma 2.4 *Every partial k-path can be represented by some parse* $G_n = (g_1, g_2, ..., g_n) \in (\Sigma_k)^*$

Proof: First we show that every k-path can be represented by a parse over the described operator set Σ_k.

We prove this inductively, from the recursive definition of k-paths. Suppose we start from an empty k pathwidth graph. The initial $(k+1)$ clique can be represented by the following sequence of operators over Σ_k.

$\boxed{f_k}\ (\boxed{\text{b}})^*\ (\boxed{\text{a}})^*\ \boxed{\text{c}}\ (\boxed{\text{b}})^*\ (\boxed{\text{a}})^*\ \boxed{\text{c}}$,..., $(\boxed{\text{b}})^*\ (\boxed{\text{a}})^*\ \boxed{\text{c}}$

where the number of $\boxed{\text{c}}$ operator is equal to k.

For the inductive step, assume that any k-path $G = (g_1, g_2, ..., g_n)$ with $|G| = m$ can be represented as the sequence of operators in the given operator set Σ_k. We will prove that for a k-path G' with $|G'| = m + 1$, it can also be represented as the sequence of operators in the given operator set Σ_k.

From the definition of k-paths, there are two ways to construct G' from G.

1. boundary vertices keep unchanged.

   $G' = G + \boxed{f_k}$

2. one of boundary vertices is changed to interior vertex and the new added vertex is added to boundary vertices.

   $\boxed{\text{e}}\ (\boxed{\text{b}})^*\ (\boxed{\text{a}})^*\ \boxed{\text{c}}\ (\boxed{\text{b}})^*\ (\boxed{\text{a}})^*\ \boxed{\text{c}}$,..., $(\boxed{\text{b}})^*\ (\boxed{\text{a}})^*\ \boxed{\text{c}}$

   where the number of $\boxed{\text{c}}$ operator is equal to $k-1$.

Next, we need to prove that every partial k-path can be represented by a sequence of operators from the described operator set. We use the fact that any partial k-path graph can be obtained from a k-path by edges deleting.

For an edge deleting, if the edge is introduced by a $\boxed{\text{c}}$ operator, we can simply delete it; if the edge is introduced by an $\boxed{\text{e}}$ operator, then we can use a $\boxed{\text{d}}$ operator to replace it; if the edge is introduced by a $\boxed{f_i}$ $(0 \leq i \leq k-1)$ operator, then we can use a $(\boxed{\text{b}})^*$ $(\boxed{\text{a}})^*$ $\boxed{f_j}$ $(\boxed{\text{b}})^*$ $(\boxed{\text{a}})^*$ $(0 \leq j < i)$ operator to replace it;

Thus we reach the conclusion that every partial k-path can be represented by the operator set. □

Theorem 2.4 *The family of graphs generated by Σ_k equals the family of partial k-paths.*

Proof: The theorem is proved from Lemma 2.3 and Lemma 2.4. □

2.3 Examples of the parsing scheme

To illustrate our parsing representation of bounded pathwidth graphs, Figure 2.1 shows how the cycle with six vertices can be parsed with bounded pathwidth-three by the parsing scheme. Notice that many possible pathwidth-three decompositions are possible. (The square vertices denote the final set of boundary vertices.)

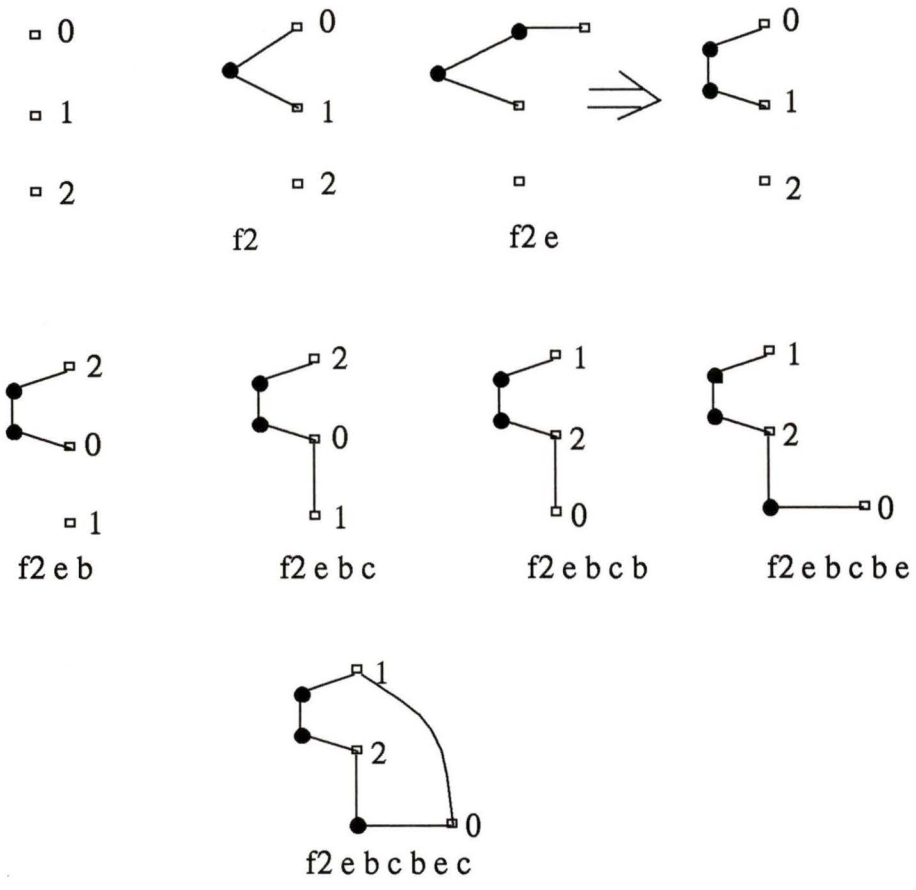


Figure 2.1: Parsing a cycle with six vertices

Chapter 3

Finite State Automata and Test Sets

In this chapter, we describe the theoretical foundation of the algorithm compilation system. A central role is played by a graph-theoretic analog of the well-known Myhill-Nerode theorem of formal language theory. Another key concept is that of a *test set* for a regular language, which is explained here both for formal languages and for bounded pathwidth graphs. Test sets for the graph property of Hamiltonicity for the pathwidth bounds 2 and 3 are identified. These test sets form part of the input to the algorithm compilation system, from which the Hamiltonicity automata are produced. The results of the compilation are described in Chapter 4.

3.1 Preliminaries

We first describe the principles of finite-state algorithm compilation in the setting of regular formal languages, and then describe how this can be generalized to the setting of finite-state algorithms for bounded pathwidth graphs.

Definition 3.1 [30] *Let $L \subseteq \Sigma^*$ be an arbitrary formal language over the finite alphabet Σ. We define the canonical (right) congruence for L to be the equivalence relation on Σ^* defined: $x \sim_L y$ if and only if for all $z \in \Sigma^*$, $xz \in L \leftrightarrow yz \in L$.*

It is a standard fact (and easily verified) that: (1) $\sim_L$ is an equivalence relation; (2) for all $x, y, z \in \Sigma^*$, $x \sim_L y$ implies $xz \sim_L yz$; and (3) L is a union of equivalence classes of $\sim_L$. The property (2) is that of being a *right congruence*, which we define generally as follows.

Definition 3.2 [30] *Let $\sim$ be an equivalence relation on Σ^*. We say that $\sim$ is a right congruence if for all $x, y, z \in \Sigma^*$, $x \sim y$ implies $xz \sim yz$.*

When discussing an equivalence relation $\sim$ on Σ^*, we will use the notation $[x]_\sim$ to denote the equivalence class of $x \in \Sigma^*$ with respect to $\sim$, or simply the notation $[x]$ where $\sim$ is understood.

Definition 3.3 [30] *The index of an equivalence relation $\sim$ is the cardinality of the set of equivalence classes. We say that $\sim$ has finite index if the index of $\sim$ is finite.*

The classic Myhill-Nerode theorem of formal languages is usually stated as follows.

Theorem 3.1 [30] *The following are equivalent for a language L over a finite alphabet Σ:*

(1) L is finite-state.

(2) The canonical congruence $\sim_L$ has finite index.

(3) L is a union of equivalence classes of a right congruence of finite index on Σ^*.

It is a standard fact that if L is a finite-state language, then there is a unique DFA for L having a minimum number of states. This is termed the *minimal automaton* for L. The equivalence relation $\sim_L$ provides an implicit description of the minimal automaton for L in the following way.

1. The states of the minimal automaton are the equivalence class of $\sim_L$.

2. The start state of the minimal automaton is $[\lambda]$, the equivalence class of the empty word.

3. The transition function δ is defined for $a \in \Sigma$ by $\delta([x], a) = [xa]$.

4. The accept states of the minimal automaton are those equivalence classes $[x]$ for which $[x] \subseteq L$.

The following "computational form" of the Myhill-Nerode theorem provides the conceptual engine for the finite-state algorithm compilation system.

Theorem 3.2 [30] *Let $L \subseteq \Sigma^*$ be a regular language, and suppose that the following are supplied as inputs:*

(1) A decision algorithm for L.

(2) A decision algorithm for a finite index right congruence $\sim$ on Σ^* with respect to which L is a union of equivalence classes.

Then from this information a DFA for L can be computed.

The algorithm described by the above theorem (described in more detail for the bounded pathwidth setting in Chapter 4) progressively identifies a set of words R that are representatives of the equivalence classes of $\sim$ that implicitly constitute the states of the DFA. Each equivalence class (state) is represented by a unique word in R. and the transition function for the DFA represented as a function $\delta : R \times \Sigma \to R$.

The process starts with R containing only the empty word λ which represents the start state $[\lambda]$. The transition function is defined (implicitly) by $\delta([x], a) = [xa]$. Initially we know that the start state is $[\lambda]$, represented by λ, and that on the symbol $a \in \Sigma$ there is a transition from $[\lambda]$ to $[\lambda a = a]$, which, if it is a new state, could be represented by the word a. The question arises, however, whether this "new state" has already been identified. That is, it may be the case that $[\lambda] = [a]$ (i.e. $\lambda \sim a$). We use the input decision algorithm (2) to determine whether this is the case. In general, we perform a breadth-first search of Σ^* beginning from λ, at each step of this search determining, by the use of algorithm (2), whether a representative of a new state has been identified, and in any case progressively determining the transition function. Since there are only finitely many states, this process must eventually complete. At that point, we use the input decision algorithm (1) to identify the accept states of the automaton.

We remark that it is important to understand that the input (1) to the compilation algorithm described above is generally *not* finite-state. In the case of Hamiltonicity of graphs, we obviously know an effective but inefficient decision algorithm for the property that consists in simply trying all possibilities for a Hamilton tour of the vertex set. The

interesting thing is that this inefficient algorithm can be used to compile an extremely efficient finite-state algorithm (for bounded pathwidth). The algorithm compilation system accesses the input decision algorithms (1) and (2) as subroutines.

We next describe the notion of a *test set* for a regular language L, which is used to give us a handle on $\sim_L$ for the purposes of finite-state algorithm compilation.

Definition 3.4 [30] *Let $L \subseteq \Sigma^*$ be a regular language and let $T \subseteq \Sigma^*$ be a set of words. We define the equivalence relation on Σ^*: $x \sim_T y$ (with respect to L) if $\forall t \in T(xt \in L \leftrightarrow yt \in L)$.*

Definition 3.5 [30] *A test set for a regular language $L \subseteq \Sigma^*$ is a set of words $T \subseteq \Sigma^*$ such that $\sim_T = \sim_L$.*

In other (more intuitive) words, a test set for L is a set of "extensions" t such that if x and y agree on these extensions, then they agree on all extensions (compare the definition of $\sim_L$). Of course, this is most interesting when the test set T is finite.

Lemma 3.1 *Let $M = (Q, \Sigma, q_0, \delta, F)$ be the minimal automaton for a regular language $L \subseteq \Sigma^*$. If T is a set of words with the property that for every pair of states $q, q' \in Q$, there is a word $t \in T$ such that $\delta(q,t) \in F$ and $\delta(q',t) \notin F$ (or vice versa) then T is a test set for $L = L(M)$.*

Proof: We claim that $\sim_T = \sim_L$. Clearly $x \sim_L y$ implies $x \sim_T y$. Conversely, suppose $x \not\sim_L y$. But then $[x]$ and $[y]$ are distinct states in the minimal automaton for L, and so there is a word $t \in T$ such that (without loss of generality) $\delta([x], t) \in F$ while $\delta([y], t) \notin F$, that is, $xt \in L$ while $yt \notin L$, and therefore $x \not\sim_T y$. □

Note that necessarily if $[x]$ and $[y]$ are two distinct states of the minimal automaton, then necessarily there is a word t such that $\delta([x], t) \in F$ while $\delta([y], t) \notin F$ (or *vice versa*). From this one easily proves the following.

Lemma 3.2 [12] *If $L \subseteq \Sigma^*$ is a regular language, then there is a test set T for L that consists of $\binom{n}{2}$ words of length at most n^2, where n is the number of states in the minimal automaton for L.*

The next lemma shows that if we have a test set T for a regular language L then we have a handle on the canonical congruence for L. The practical consequence is that finite-state algorithm compilation based on a test set immediately yields the minimal automaton for the language L.

Lemma 3.3 *If T is a test set for a regular language L, then $\sim_T = \sim_L$.*

Proof: It follows from Definition 3.5 of a test set. □

All of the above generalizes to the setting of finite-state algorithms for bounded pathwidth graphs, and the proofs are essentially identical. The analog of the Myhill-Nerode theorem for the bounded pathwidth setting has already been stated in Chapter 2 as Theorem 2.1.

The relevant notion of a *test set* for a finite-state graph property is defined as follows.

Definition 3.6 *Let k be a pathwidth bound, and let $\mathcal{F}$ be a family of graphs that is finite-state for graphs of pathwidth at most k. A test set for $\mathcal{F}$ for pathwidth k is a set T of k-boundaried graphs such that $\sim_T = \sim_{\mathcal{F}}$, where $\sim_T$ is the equivalence relation on k-boundaried graphs defined: $X \sim_T Y$ if and only if $\forall Z \in T$ $(X \oplus Z \in \mathcal{F} \leftrightarrow Y \oplus Z \in \mathcal{F})$.*

The algorithm compilation system for bounded pathwidth graphs that is the principal result of this project embodies the following "algorithmic version" of the graph-theoretic analog of the Myhill-Nerode theorem.

Theorem 3.3 [2] *Let k be a pathwidth bound, and let $\mathcal{F}$ be a family of graphs that is finite-state for graphs of pathwidth at most k. Suppose that the following are supplied:*

(1) A decision algorithm for $\mathcal{F}$.

(2) A test set for $\mathcal{F}$ for pathwidth k.

Then from this information a DFA for $\mathcal{F}$ for pathwidth k can be computed.

3.2 Test sets for Hamiltonicity

3.2.1 A complete test set for Hamiltonicity on partial 2-paths

In this section we identify a complete test set for building a finite state automaton for Hamiltonicity on partial 2-paths (graphs with pathwidth ≤ 2).

Definition 3.7 *A class of graphs, $\mathcal{F}$, that have a pathwidth decomposition ≤ 2 and have a Hamilton cycle are considered to be the language of L_{ham2}.*

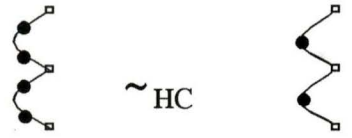
Definition 3.8 *Two partial 2-paths G and H are Hamiltonicity equivalent, $G \sim_{HC} H$, if for every 2-boundary graph $Z \in \Sigma^*$, $G \oplus Z$ is a member of L_{ham2} if and only if $H \oplus Z$ is a member of L_{ham2}.*

Without loss of generality, if $X \not\sim_{HC} Y$ we assume that there exists a test $Z \in \Sigma^*$ such

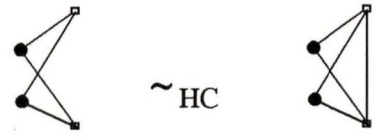
that $X \oplus Z$ is in L_{ham2} while $Y \oplus Z$ is not in L_{ham2}.

For the remainder of this chapter, let BV_G denote the set of boundary vertices of G and IV_G denote the set of interior vertices of G. Let BE_G denote the set of the edges between boundary vertices of G and IE_G denote the set of the edges between interior vertices or between an interior vertex and a boundary vertex of G. Let C denote a Hamilton cycle in the graph of $X \oplus Z$.

Fact 1: For a boundary graph G, if there are no other vertices but $k > 1$ degree two interior vertices in a path between two boundary vertices, we can remove $k-1$ such interior vertices to get a subgraph G' of G for which $G' \sim_{HC} G$. One example is as follows:



Fact 2: If a boundary graph G has a Hamilton cycle and every edge in the cycle is $e \in IE_G$, we delete or add an edge $e' \in BE_G$ to get a subgraph or supergraph G' of G for which $G' \sim_{HC} G$. One example is as follows:



For the remainder of this section, we assume that all graphs are boundary graphs with 2 boundary vertices.

Theorem 3.4 *The set of boundary graphs in Figure 3.1,* $T_2 = \{$ *Test*0, *Test*1, *Test*2, *Test*3 $\}$, *is a complete test set for Hamiltonicity on partial* 2*-paths.*

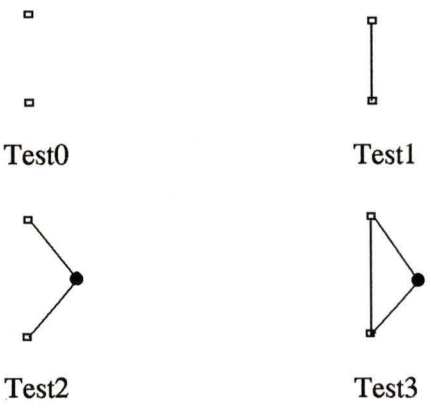


Figure 3.1: A complete test set for Hamiltonicity on partial 2-paths

Proof: Let $\sim_{HC}$ be a Hamiltonicity equivalence class. Given two graphs X and Y, if $X \not\sim_{HC} Y$, we say that X and Y are not congruent graphs for Hamiltonicity. We will show that there is a test $T \in T_2$ such that ($X \oplus T \in L_{ham2}$ and $Y \oplus T \notin L_{ham2}$) or ($X \oplus T \notin L_{ham2}$ and $Y \oplus T \in L_{ham2}$).

- Case 1: X is a Hamiltonian graph, and Y is not a Hamiltonian graph.

  There is a test $T = Test0 \in T_2$ that leaves X in L_{ham2} and throws Y out of L_{ham2}, i.e., $X \oplus T \in L_{ham2}$ and $Y \oplus T \notin L_{ham2}$.

- Case 2: X is a Hamiltonian graph, and Y is a Hamiltonian graph too.

  By Fact 1 and Fact 2 described above, the only possible pair of graphs X, Y are as follows:

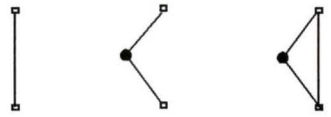
There is a test $T = Test2 \in T_2$ that leaves Y in L_{ham2} and throws X out of L_{ham2}, i.e., $X \oplus T \notin L_{ham2}$ and $Y \oplus T \in L_{ham2}$.

- Case 3: X is not a Hamiltonian graph, and Y is not a Hamiltonian graph either.
  - Case 3.1: $C \in X \oplus Z$ and $IV(X) = \phi$.

    The possible X is as follows:

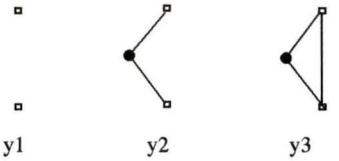
    We select $T = Test3 \in T_2$ as a candidate test. So we have $X \oplus T \in L_{ham2}$. If $Y \oplus T \notin L_{ham2}$, then we finish this case proof. Else if $Y \oplus T \in L_{ham2}$, then there are two cases: (1) $X = x1$; (2) $X = x2$.

    (1) If $X = x1$, Y can be one of the followings:



    and there is a test $T' = Test2 \in T_2$ that leaves Y in L_{ham2} and throws X out of L_{ham2}, i.e., $X \oplus T' \notin L_{ham2}$ and $Y \oplus T' \in L_{ham2}$;

    (2) If $X = x2$, then Y can be one of the followings:



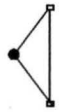
    If $Y = y1$, then there is a test $T' = Test2 \in T_2$ that leaves X in L_{ham2} and throws Y out of L_{ham2}, i.e., $X \oplus T' \in L_{ham2}$ and $Y \oplus T' \notin L_{ham2}$.

If ($Y = y2$ or $Y = y3$), then there is a test $T' = Test1 \in T_2$ that leaves Y in L_{ham2} and throws X out of L_{ham2}, i.e., $X \oplus T' \notin L_{ham2}$ and $Y \oplus T' \in L_{ham2}$.

- Case 3.2: $C \in X \oplus Z$ and $IV(X) \neq \phi$.

  The following graph is the only X that satisfies the conditions.

We select $T = Test1 \in T_2$ as a candidate test. So we have $X \oplus T \in L_{ham2}$. If $Y \oplus T \notin L_{ham2}$, then we finish this case proof. Else if $Y \oplus T \in L_{ham2}$, then Y is as follows:



and there is test $T' = Test0 \in T_2$ that leaves Y in L_{ham2} and throws X out of L_{ham2}, i.e., $X \oplus T' \notin L_{ham2}$ and $Y \oplus T' \in L_{ham2}$. □

3.2.2 A complete test set for Hamiltonicity on partial 3-paths

In this section we identify a complete test set for building a finite state automaton for Hamiltonicity on partial 3-paths (graphs with pathwidth ≤ 3).

Definition 3.9 *A class of graphs, $\mathcal{F}$, that have a pathwidth decomposition ≤ 3 and have a Hamilton cycle are defined to be the language of L_{ham3}.*

Definition 3.10 *Two partial 3-paths G and H are Hamiltonicity equivalent, $G \sim_{HC} H$, if for every 3-boundary graph $Z \in \Sigma^*$, $G \oplus Z$ is a member of L_{ham3} if and only if $H \oplus Z$*

is a member of L_{ham3}.

Without loss of generality, if $X \not\sim_{HC} Y$ we assume that there exists a test $Z \in \Sigma^*$ such that $X \oplus Z$ is in L_{ham3} while $Y \oplus Z$ is not in L_{ham3}.

For the remainder of this section we assume that all graphs are boundary graphs with 3 boundary vertices. We notice that Fact 1 and Fact 2 which have been discussed in section 3.2.1 are also applied in this section.

Theorem 3.5 *The set of boundaried graphs described in Figure 3.2,* T_3=$\{Test0, Test1, Test2, Test3, Test4, Test5, Test6, Test7, Test8, Test9, Test10, Test11, Test12, Test13, Test14\}$, *is a complete test set for Hamiltonicity on partial* 3*-paths.*

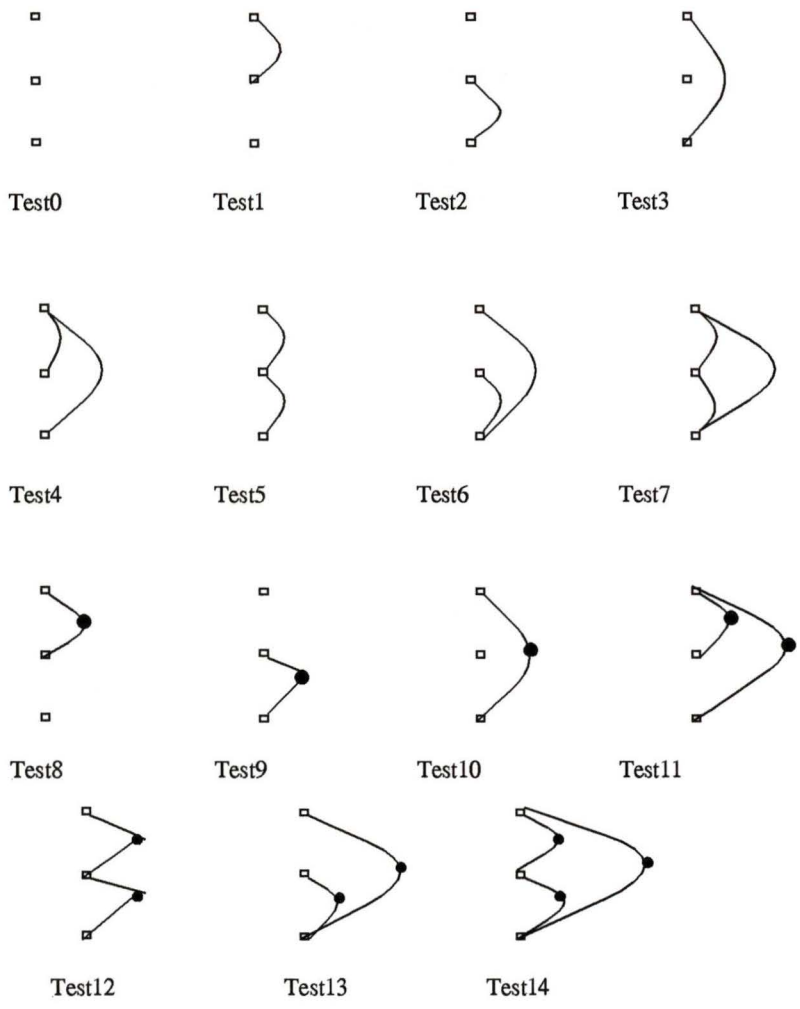
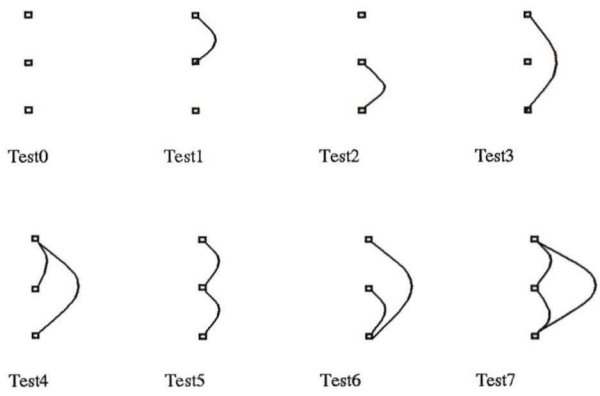


Figure 3.2: A complete test set for Hamiltonicity on partial 3-paths

Proof: Let $\sim_{HC}$ be a Hamiltonicity equivalence class. Given two graphs X and Y, if $X \not\sim_{HC} Y$, we say that X and Y are not boundary congruent graphs for Hamiltonicity. We will show that we have a test $T \in T_3$ such that ($X \oplus T \in L_{ham3}$ and $Y \oplus T \notin L_{ham3}$) or ($X \oplus T \notin L_{ham3}$ and $Y \oplus T \in L_{ham3}$).

- Case 1: $C \subseteq X$.

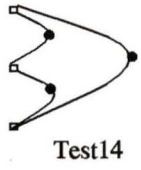
  $C \subseteq X \Rightarrow IV_Z = \phi$, and $IV_Z = \phi \Rightarrow Z \in$ {Test0, Test1, Test2, Test3, Test4, Test5, Test6, Test7} of the follows:



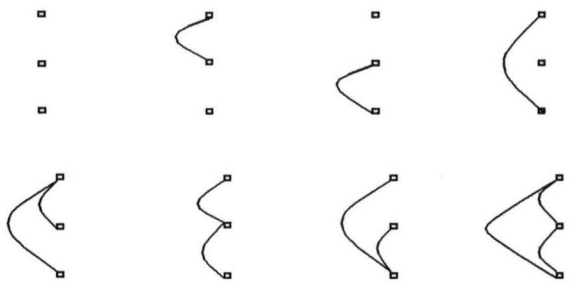
  So we can always select a corresponding $T \in T_3$ to replace Z, such that $X \oplus T \in L_{ham3}$ and $Y \oplus T \notin L_{ham3}$.

- Case 2: $C \subseteq Z$.

  $C \subseteq Z \Rightarrow IV_X = \phi$. We select $T' = \{Test14\} \in T_3$ of the following as a candidate test.

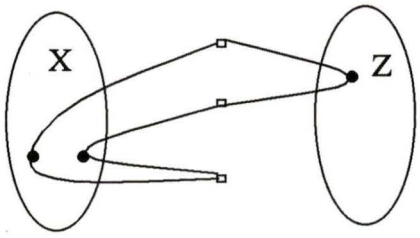


  So we get $X \oplus T' \in L_{ham3}$. If $Y \oplus T' \notin L_{ham3}$, then we finish this case proof. Else if $Y \oplus T' \in L_{ham3}$, then $IV_Y = \phi$. By $IV_X = \phi$ and $IV_Y = \phi$, we get all the possible X and Y as follows:



Since for a combination of any two graphs above, there is a test $T \in T_3$ which leaves one of the pair in the family and throws the other of the pair out of the family, we always can find a test $T \in T_3$ such that ($X \oplus T \in L_{ham3}$ and $Y \oplus T \notin L_{ham3}$) or ($X \oplus T \notin L_{ham3}$ and $Y \oplus T \in L_{ham3}$).

- Case 3: $C \subseteq X \oplus Z$ and its Hamilton cycle $C \in X \oplus Z$ looks like the following:

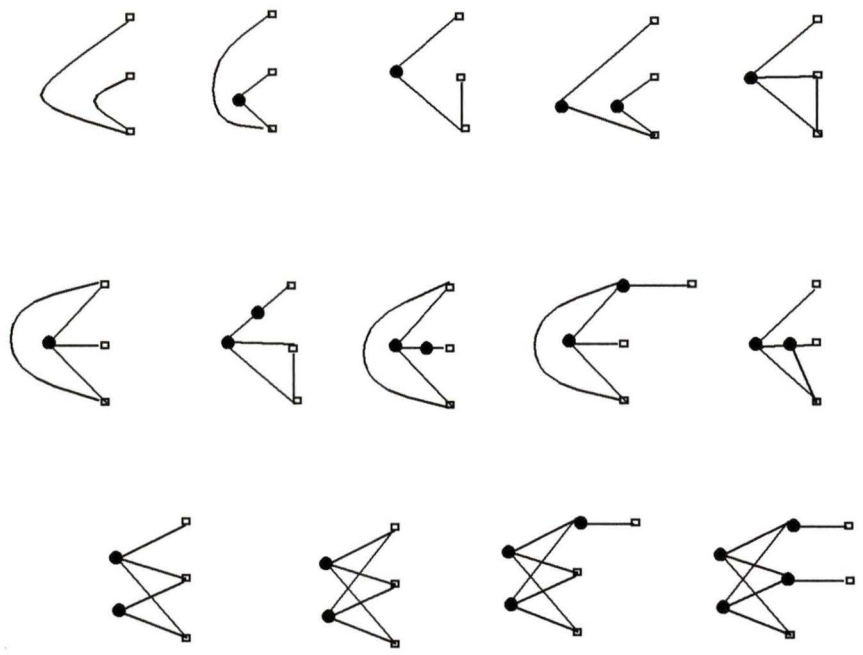


$C \subseteq X \oplus Z \Rightarrow (IV_X \neq \phi$ and $IV_Y \neq \phi)$. We select $T' = \{Test8\} \in T_3$ of the following as a candidate test.

Test8

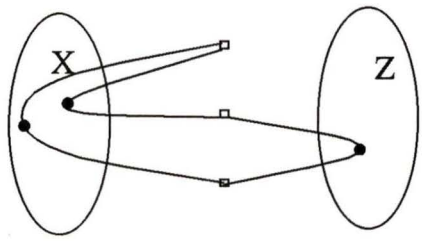
So we get $X \oplus T' \in L_{ham3}$. If $Y \oplus T' \notin L_{ham3}$, then we finish this case proof. Otherwise if $Y \oplus T' \in L_{ham3}$, then $IV_Y \neq \phi$. By ($IV_X \neq \phi$, $IV_Y \neq \phi$) and

($X \oplus T' \in L_{ham3}$, $Y \oplus T' \in L_{ham3}$), we get all the possible different X and Y as follows:

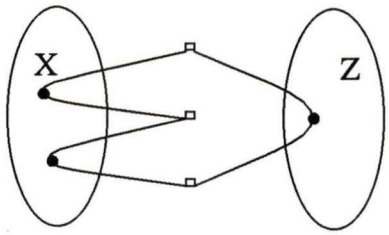


Since for a combination of any two graphs above, there is a test $T \in T_3$ which leaves one of the pair in the family and throws the other of the pair out of the family, we always can find a test $T \in T_3$ such that ($X \oplus T \in L_{ham3}$ and $Y \oplus T \notin L_{ham3}$) or ($X \oplus T \notin L_{ham3}$ and $Y \oplus T \in L_{ham3}$).

- Case 4: $C \subseteq X \oplus Z$ and its Hamilton cycle $C \in X \oplus Z$ looks like the following:

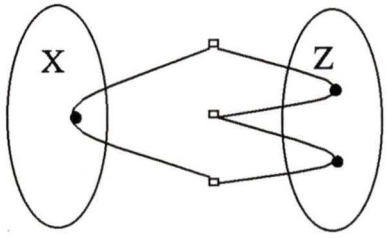


- Case 5: $C \subseteq X \oplus Z$ and its Hamilton cycle $C \in X \oplus Z$ looks like the following:



To prove case 4 and case 5, we can symmetrically use the same way as in case 3.

- Case 6: $C \subseteq X \oplus Z$ and its Hamilton cycle $C \in X \oplus Z$ looks like the following:

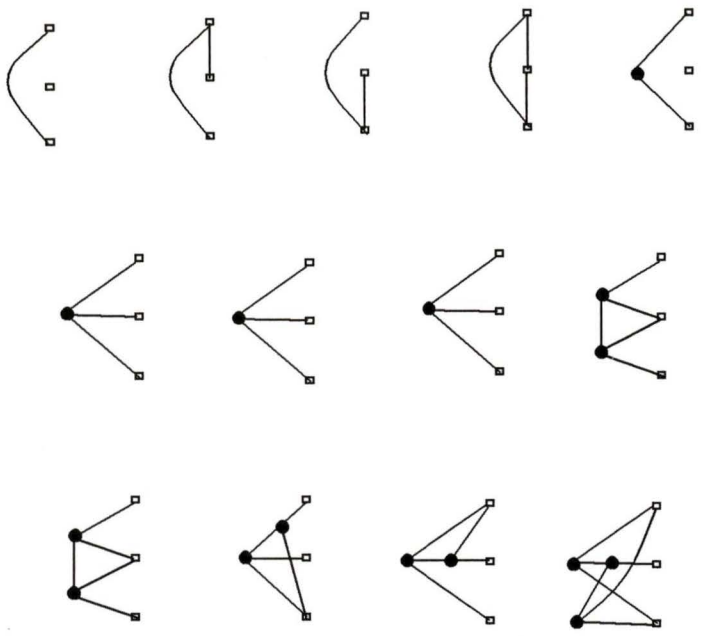


$C \subseteq X \oplus Z \Rightarrow (IV_X \neq \phi$ and $IV_Y \neq \phi)$.

We select $T^{'} = \{Test12\} \in T_3$ of the following as a candidate test.

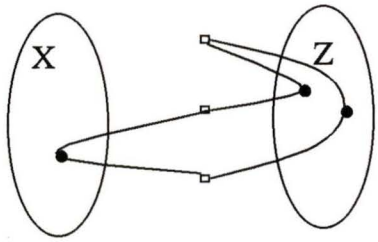
Test12

So we get $X \oplus T^{'} \in L_{ham3}$. If $Y \oplus T^{'} \notin L_{ham3}$, then we finish this case proof. Otherwise if $Y \oplus T^{'} \in L_{ham3}$, then $IV_Y \neq \phi$. By ($IV_X \neq \phi$, $IV_Y \neq \phi$) and ($X \oplus T^{'} \in L_{ham3}$, $Y \oplus T^{'} \in L_{ham3}$), we can get all the possible different X and Y as follows:

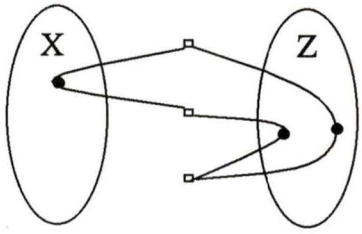


Since for a combination of any two graphs above, there is a test $T \in T_3$ which leaves one of the pair in the family and throws the other of the pair out of the family, we always can find a test $T \in T_3$ such that ($X \oplus T \in L_{ham3}$ and $Y \oplus T \notin L_{ham3}$) or ($X \oplus T \notin L_{ham3}$ and $Y \oplus T \in L_{ham3}$).

- Case 7: $C \subseteq X \oplus Z$ and its picture looks like the following:



- Case 8: $C \subseteq X \oplus Z$ and its picture looks like the following:



To prove case 7 and case 8, we can symmetrically use the same way as in case 6. □

Chapter 4

A Brief Description of the Algorithm & Experiment Results

The conceptual engine is described as the following in Chapter 3 for the finite-state algorithm compilation system for the regular language $L \subseteq \Sigma^*$.

(1) A decision algorithm for L.

(2) A decision algorithm for a finite index right congruence $\sim$ on Σ^* with respect to which L is a union of equivalence classes.

In this chapter, based on Theorem 3.3 of chapter 3, we describe how to implement this finite-state algorithm compilation system for the bounded pathwidth setting. In section 4.1 we present a principal algorithm for the compilation. In section 4.2 we give our experiment results of finite-state automaton for recognizing the property of Hamiltonicity on partial 2-paths. And in section 4.3 we give our experiment results of finite automaton for recognizing the property of Hamiltonicity on partial 3-paths.

4.1 The finite-state compilation algorithm for bounded pathwidth graphs

In this section we present a compilation algorithm for constructing a finite-state automaton to recognize a finite-state property of bounded pathwidth graphs. We first give some definitions which are necessary to understand our algorithm and then present our compilation algorithm. Let $\mathcal{F}$ be a family of graphs that is finite state for graphs of pathwidth at most k.

Definition 4.1 *For every $y = xa$, where $x, y \in \Sigma_k^*$ and $a \in \Sigma_k$, if y is a member of the set S' if and only if x is a member of the set S', then we say that the set S' is a prefix-closed set. If a prefix-closed set S' is not empty and has finite elements then we say that S' is a non-empty finite prefix-closed set.*

Definition 4.2 *The observation table is denoted by (S, T, f) where S is a nonempty finite prefix-closed set, T is a complete test set for $\mathcal{F}$ for pathwidth k and f is a finite function mapping of $((S \cup S \cdot \Sigma_k) \oplus T)$ to $\{0,1\}$.*

An observation table can be visualized as a two-dimensional array with rows labeled by elements of $(S \cup S \cdot \Sigma_k)$ and columns labeled by elements of T, with the entry for row s and column t equal to $f(s,t)$. Where $u = s \oplus t$, $f(u) = 1$ if and only if $u \in \mathcal{F}$. The $row(s)$ denotes the finite function f_s from T to $\{0,1\}$, defined by $f_s(t) = f(s,t)$. Our algorithm of compilation for constructing a DFA for $\mathcal{F}$ is as follows:

Input: (1) A decision algorithm A for $\mathcal{F}$;

(2) A complete test set T for $\mathcal{F}$ for pathwidth k.

Output: A DFA for $\mathcal{F}$ for pathwidth k.

Algorithm:

Initialize S to $\{\lambda\}$.

Do membership testing for $\lambda \oplus t$ for each element of $t \in T$ by calling A.

Do membership testing for $\lambda \cdot a \oplus t$ for each $a \in \Sigma_k$ and each element of $t \in T$ by calling A.

Construct the initial observation table (S,T,f).

Repeat

While (S,T,f) is not closed

Find $s_1 \in S$ and $a \in \Sigma_k$ such that

$row(s_1 \cdot a)$ is different from row(s) for all $s \in S$.

Add $s_1 \cdot a$ to S.

Extend f to $(S \cup S \cdot \Sigma_k) \oplus T$ by doing membership testing by calling A.

Until (S,T,f) is closed

Make the DFA from this closed observation table.

Definition 4.3 *An observation table is called closed provided that for each element* $s_1 \in S \cdot \Sigma_k$ *there exists an* $s \in S$ *such that* $row(s_1) = row(s)$.

Theorem 4.1 *If* (S, T, f) *is a closed observation table, the following automaton which is defined over* Σ_k *with state set* Q, *initial state* q_0, *accept state set* F *and transition function* δ *is an automaton for* $\mathcal{F}$ *for pathwidth* k.

1. $Q = \{row(s) : s \in S\}$;

2. $q_0 = row(\lambda)$;

3. δ: $\delta(row(s), a) = row(s, a)$ *where* $a \in \Sigma_k$;

4. $F = \{row(s) : s \in S \text{ and } f(s, \lambda) = 1\}$.

Proof:

1. $Q = \{row(s) : s \in S\}$, the states of the DFA are the equivalence classes of $\sim_{\mathcal{F}}$.
2. The start state of the DFA is $[\lambda]$, the equivalence class of the empty word.
3. The transition function δ of the DFA is defined for $a \in \Sigma_k$ by $\delta([x], a) = [xa]$.
4. The accept states of the DFA are just those equivalence classes $[x]$ for which $[x] \subseteq \mathcal{F}$. □

4.2 Experiment results on partial 2-paths

Input: (1) A decision algorithm for Hamiltonicity – trying all possibilities for a Hamilton tour of the vertex set of the graph; and (2) the test set T= $\{Test0, Test1, Test2, Test3\}$ of Figure 3.1.

Output: A DFA for recognizing the property of Hamiltonicity of graphs with pathwidth ≤ 2.

Starting from the input we get the closed observation table (S, T, f) as showed in Table 4.1.

$S \setminus T$	Test0	Test1	Test2	Test3
λ	0	0	0	1
$\boxed{\text{c}}$	0	0	1	1
$\boxed{\text{d}}$	0	0	0	0
$\boxed{f_2}$	0	1	1	1
$\boxed{\text{c}} \cdot \boxed{f_2}$	1	1	1	1
$\boxed{f_2} \cdot \boxed{f_2}$	1	1	1	0

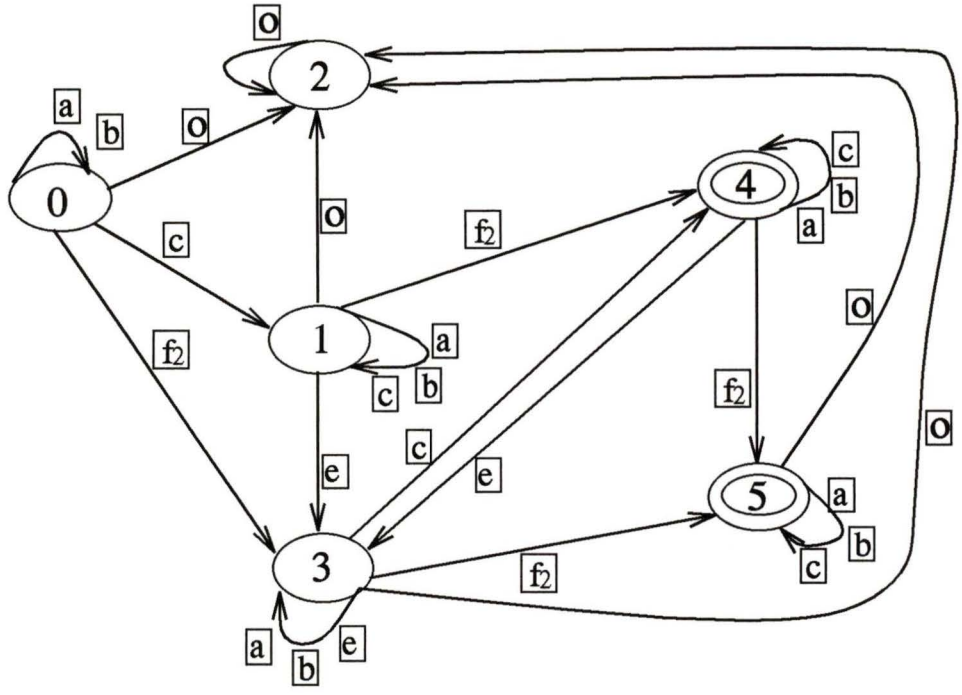
Table 4.1: A closed observation table (S,T,f) for constructing a DFA to recognize the property of Hamiltonicity of graphs with pathwidth ≤ 2

states $\setminus \Sigma$	$\boxed{\text{a}}$	$\boxed{\text{b}}$	$\boxed{\text{c}}$	$\boxed{\text{d}}$	$\boxed{\text{e}}$	$\boxed{f_0}$	$\boxed{f_1}$	$\boxed{f_2}$
0	0	0	1	2	2	2	2	3
1	1	1	1	2	3	2	2	4
2	2	2	2	2	2	2	2	2
3	3	3	4	2	3	2	2	5
4	4	4	4	2	3	2	2	5
5	5	5	5	2	2	2	2	2

Table 4.2: A transition table for constructing a DFA to recognize the property of Hamiltonicity of graphs with pathwidth ≤ 2, final states: 4 and 5

By the closed observation table showed in Table 4.1 we get the automaton transition table as showed in Table 4.2.

By the automaton transition table showed in Table 4.2 we get our DFA as showed in Figure 4.1.



Note: o = all other symbols not marked in each state's transitions

Figure 4.1: A DFA for recognizing the property of Hamiltonicity of graphs with pathwidth ≤ 2

4.3 Experiment results on partial 3-paths

Input: (1) A decision algorithm for Hamiltonicity – trying all possibilities for a Hamilton tour of the vertex set of the graph; and (2) and the test set T=$\{Test0, Test1, Test2, Test3,$

$Test4$, $Test5$, $Test6$, $Test7$, $Test8$, $Test9$, $Test10$, $Test11$, $Test12$, $Test13$, $Test14\}$ of Figure 3.2.

Output: A DFA for recognizing the property of Hamiltonicity of graphs with pathwidth ≤ 3.

Starting from the input we can get the closed observation table (S, T, f). By the closed observation table (S, T, f) we can get the automaton transition table. By the automaton transition table we can get our DFA. We just list our resulting finite state automaton transition table in Table 4.3 since the automaton has 58 states. It is too big to draw its closed observation table and DFA here.

states $\backslash\Sigma$	a	b	c	d	e	f_0	f_1	f_2	f_3
0	0	0	1	2	2	2	2	3	4
1	1	5	1	2	3	2	2	3	6
2	2	2	2	2	2	2	2	2	2
3	3	7	3	2	3	2	2	2	8
4	4	4	6	2	9	2	2	8	10
5	11	11	12	2	7	2	2	13	14
6	6	14	6	15	16	2	2	8	17
7	15	15	18	2	7	2	2	13	19
8	8	19	8	15	20	2	2	2	21
9	22	23	18	2	9	2	2	13	10
10	10	10	17	15	24	2	2	21	21
11	5	1	25	2	2	2	2	26	27
12	25	28	12	15	4	2	2	13	29
13	26	30	13	15	22	2	2	2	31
14	27	27	29	15	32	2	2	31	17
15	7	3	20	2	2	2	2	26	33
16	34	34	6	2	35	2	2	8	17
17	17	17	17	15	24	2	2	21	21
18	20	36	18	15	23	2	2	13	37
19	33	33	37	15	36	2	2	21	21

states $\backslash\Sigma$	a	b	c	d	e	f_0	f_1	f_2	f_3
20	18	13	20	2	26	2	2	26	28
21	21	21	21	2	2	2	2	2	2
22	9	9	20	2	3	2	2	26	10
23	23	22	6	2	7	2	2	8	10
24	39	8	40	2	33	2	2	41	38
25	12	12	25	2	26	2	2	26	29
26	13	18	26	2	26	2	2	2	31
27	14	6	29	2	42	2	2	31	17
28	28	25	43	2	30	2	2	31	29
29	29	29	29	15	27	2	2	31	17
30	36	20	44	2	30	2	2	45	37
31	31	37	31	15	20	2	2	2	21
32	32	16	29	2	46	2	2	31	17
33	19	8	38	2	33	2	2	21	21
34	16	32	6	15	4	2	2	8	17
35	47	34	48	2	35	2	2	13	17
36	30	26	40	2	49	2	2	41	38
37	38	38	37	15	36	2	2	21	21
38	37	31	38	2	33	2	2	21	21
39	24	24	44	15	50	2	2	45	37

states $\setminus\Sigma$	a	b	c	d	e	f_0	f_1	f_2	f_3
40	44	31	40	2	33	2	2	41	38
41	45	45	41	2	26	2	2	2	21
42	51	6	52	2	42	2	2	45	17
43	43	43	43	15	27	2	2	31	29
44	40	40	44	15	50	2	2	45	37
45	41	53	45	15	15	2	2	2	21
46	54	16	44	2	46	2	2	45	17
47	35	46	55	15	22	2	2	26	17
48	55	14	48	15	16	2	2	13	17
49	49	26	53	2	49	2	2	21	21
50	50	20	29	2	30	2	2	31	17
51	42	42	56	15	54	2	2	41	17
52	56	29	52	15	27	2	2	45	17
53	53	41	53	2	49	2	2	21	21
54	46	35	40	2	57	2	2	41	17
55	48	39	55	15	20	2	2	26	17
56	52	52	56	15	24	2	2	41	17
57	57	35	53	2	57	2	2	21	31

Table 4.3: A closed observation table for constructing a DFA to recognize Hamiltonian property of graphs with pathwidth ≤ 3, final states: 17 21 29 31 37 38 40 41 43 44 45 52 53 56

Chapter 5

Summary and Discussion of Open Problems

The main achievement of this project has been to create a software package for empirical exploration of the problem of the sizes of finite automata for solving graph recognition problems for bounded pathwidth. The use of the package has been demonstrated on the well-known problem of graph Hamiltonicity, and the sizes of the automata for the pathwidth bounds 2 and 3 determined. This effort involved also the identification and proof of completeness for a set of parsing operators, and the identification of tests sets for the property of Hamiltonicity.

A large number of interesting open problems remain. We note some of the most important.

(1) It will be interesting to compare the automata size results for Hamiltonicity with results for other graph properties. It can be shown that there are graph properties for

which the minimal automata has arbitrarily large size, for any given pathwidth bound, but it may be that most of the "natural" graph problems have a similar growth function $f(k)$ for the size of the minimal automata for a pathwidth bound of k.

(2) The size function discussed in (1) depends on the particular choice of parsing operators. It would also be interesting to see how the size of the function is influenced by this choice for the property of Hamiltonicity.

(3) There are several basic open problems concerning this computational perspective. Among these: the complexity of computing tests sets of minimal size for finite-state languages over a fixed-size alphabet is presently undetermined. One can show that there is a single "universal" test set $U_n \subseteq \Sigma^*$ of size $u(n)$ that is adequate for the all regular languages over Σ accepted by automata with at most n states. If a polynomial bound on $u(n)$ can be shown, then this would have interesting implications for both computational learning theory and structural complexity. The problem of bounding the size of a smallest universal test set is therefore interesting. Little is presently known about this question.

(4) For some important graph properties, such as having *bandwidth* bounded by a fixed constant t, it can be shown that the decision problem is *not* finite-state for bounded treewidth or pathwidth. However, it may be possible to compile a finite-state automata that is "usually correct" (perhaps according a naturally occurring input distribution). As an example of this possibility in the relatively simple setting of formal languages over the 2-letter alphabet $\{a, b\}$, it is well-known that the language $\{x : |x|_a = |x|_b\}$ is not finite-state. Yet, if one chooses a large test set at random, the resulting compiled automata can be observed to perform to some interesting degree of approximation. Some such finite-

state approximation methods could conceivably be useful for some graph problems having applications in VLSI layout. This could be explored with the software package developed by this project.

Bibliography

[1] Abrahamson, K., Ellis, J., Fellows, M. and Mata, M., On the complexity of fixed-parameter problems. *Proceedings F. O.C.S. (1989), 210-215.*

[2] Abrahamson, K., and Fellows, M. R., Finite automata, bounded treewidth and well-quasiordering. *Technical Report DCS-185-IR, 1992, University of Victoria.*

[3] Arnborg, S., Corneil, D. G., and Proskurowski, A., Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth. 8, (1987), pp. 227-284.*

[4] Arnborg, S., Lagergren J. and Seese D., Problems easy for tree-decomposable graphs. *Proc. 15th Int. Coll. Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 317 (Springer, Berlin, 1988), 38-51.*

[5] Arnborg S. and Proskurowski A., Linear time algorithms for NP-hard problems on graphs embedded in k-trees. *Technical Report TRITA-NA-8404, The Royal Institute of Technology, Sweden, 1984.*

[6] Bern M. W., Lawler E. L. and Wong A. L., Linear time computation of optimal subgraphs of decomposable graphs. *J. of algorithms 8 (1987), 216-235.*

[7] Bodlaender H. L., Dynamic programming on graphs with bounded treewidth. *Ph.D dissertation, M.I.T., 1987.*

[8] Bodlaender H. L., A linear time algorithm for finding tree-decompositions of small treewidth. *In Proc. 25th Annual ACM Symposium on Theory of Computing. ACM Press, 1993.*

[9] Bodlaender, H. L., and Kloks, T., Better algorithms for the pathwidth and treewidth of graphs. *In proceedings of the 18th International Colloquium on Automata, Languages and Programming, pages 544-555. Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.*

[10] Cattell, K. and Dinneen, M., Pathwidth and k-paths. *Manuscript, University of Victoria.*

[11] Courcelle, B., The monadic second-order logic of graphs I: Recognizable sets of finite bgraphs. *Information and Computation, 85:12-75, 1990*

[12] Dinneen M., Finite state automata and test sets. *Manuscript, University of Victoria.*

[13] Dinneen M., A linear feedback vertex set algorithm for bounded pathwidth graphs. *Manuscript, University of Victoria.*

[14] Fellows, M. R., The Robertson-Seymour theorems: A survey of applications. *Contemporary Mathematics 89 (1989), 1-18.*

[15] Fellows, M. R., and Langston M. A., Nonconstructive proofs of polynomial-time complexity. *Information Processing Letters 26(1987/88), 157-162.*

[16] Fellows, M. R., and Langston M. A., Fast self-reduction algorithms for combinatorial problems of VLSI design. *Proceedings 3rd Aegean Workshop on Computing (1988) 278-287.*

[17] Fellows, M. R., and Langston M. A., Nonconstructive tools for proving polynomial-time complexity. *Journal of the Association for Computing Machinery 35 (1988) 727-739.*

[18] Fellows, M. R., and Langston M. A., An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. *In proceedings of the 30th Annual Symposium on Foundations of Computer Science, pages 520-525, 1989.*

[19] Fellows, M. R., and Abrahamson, K. R., Cutset-regularity beats well-quasi-ordering for bounded treewidth. *Contemporary Mathematics, Volume 147, 1993.*

[20] Fellows, M.R., and Langston, M. A., On search, decision and the efficiency of polynomial-time algorithms. *In proceedings of the 21rd Annual Symposium on Theory of Computing, pages 501-512, 1989.*

[21] Ellis, J. A., Sudborough, I. H., and Turner, J., Graph separation and search number. *Technical report DCS-66-IR, University of Victoria, (1987).*

[22] Gold, E. M., Complexity of automaton identification from given data. *Information and Controls 37, pages 321-333 (1978).*

[23] Hedetniemi S. T., Bibliography of algorithms on partial k-trees. *Manuscript, Clemson University, 1989.*

[24] Kloks, T., Treewidth, *PhD thesis, Universiteit Utrecht, The Netherlands, 1993.*

[25] Robertson, N., and Seymour, P. D., Graph minors – a survey. *Surveys in Combinatorics, pages 153-171, 1985.*

[26] Robertson, N., and Seymour, P. D., Graph minors. II: algorithmic aspects of treewidth.*J. Algorithms 7, (1986), pages 309-322..*

[27] Robertson, N., and Seymour, P. D., Graph minors. IV. Tree-width and well-quasi-ordering. *J. Comb. Theory Series B, 48:227-254, 1990.*

[28] Robertson, N., and Seymour, P. D., Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory Series B, 52:153-190, 1991.*

[29] Ord-Smith, R. J., Generation of permutation sequences: part1. *The Computer Journal Volume 13, Number 2, pages 152-139.*

[30] Wood, D., Theory of Computation. *HARPER & ROW, PUBLISHERS, New York.*

VITA

Surname: LU Given Names: XIUYAN

Place of Birth: Jilin, China Date of Birth: Dec. 4 , 1963

Educational Institutions Attended:

University of Victoria	1991 to 1993
Beijing University of Aeronautics and Astronautics	1984 to 1987
Beijing University of Aeronautics and Astronautics	1980 to 1984

Degrees Awarded:

B.Sc.	Beijing University of Aeronautics and Astronautics	1984
M.Sc.	Beijing University of Aeronautics and Astronautics	1987

PARTIAL COPYRIGHT LICENSE

Title of Project Report:

Finite State Properties of Bounded Pathwidth Graphs

Author: [redacted]

Xiuyan Lu

November 10, 1993