PRECEDENCE THINNESS OF GRAPHS AND RESTRICTED HAMMING-HUFFMAN TREES

Moysés da Silva Sampaio Júnior

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Jayme Luiz Szwarcfiter
Fabiano de Souza Oliveira

Rio de Janeiro
Novembro de 2022

PRECEDENCE THINNESS OF GRAPHS AND RESTRICTED HAMMING-HUFFMAN TREES

Moysés da Silva Sampaio Júnior

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Jayme Luiz Szwarcfiter
Fabiano de Souza Oliveira

Aprovada por: Prof. Jayme Luiz Szwarcfiter
Prof. Fabiano de Souza Oliveira
Profa. Flavia Bonomo-Braberman
Prof. Fábio Happ Botler
Prof. Min Chih Lin
Profa. Cláudia Linhares Sales

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2022

da Silva Sampaio Júnior, Moysés

Precedence Thinness of Graphs and Restricted Hamming-Huffman Trees/Moysés da Silva Sampaio Júnior. – Rio de Janeiro: UFRJ/COPPE, 2022.

XI, 57 p.: il.; 29, 7cm.

Orientadores: Jayme Luiz Szwarcfiter

Fabiano de Souza Oliveira

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2022.

Referências Bibliográficas: p. 55 – 57.

1. (proper) k-thin graphs. 2. Precedence (proper) k-thin graphs. 3. Recognition algorithm. 4. Characterization. 5. Threshold graphs. 6. Hamming–Huffman codes. 7. Hypercube graphs. 8. Minimum neighborhood. I. Luiz Szwarcfiter, Jayme *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Primeiramente, gostaria de agradecer ao professor Paulo Eustáquio Duarte Pinto, meu orientador durante a graduação e o mestrado, pelo apoio, por tudo que me ensinou e por ter despertado em mim o amor pela ciência.

Agradeço imensamente aos meus orientadores Jayme Luiz Szwarcfiter e Fabiano de Souza Oliveira, por terem aceitado me orientar e por toda a paciência e generosidade que tiveram para comigo. Ter tido a oportunidade de trabalhar com os dois foi uma experiência muito enriquecedora, da qual eu sempre me lembrarei com carinho e gratidão.

Agradeço aos professores Flavia Bonomo-Braberman, Fábio Happ Botler, Min Chih Lin e Cláudia Linhares Sales por terem aceitado avaliar este trabalho. Em especial, aos professores Flavia Bonomo-Braberman e Min Chih Lin por terem me recebido em Buenos Aires em setembro de 2018, período no qual eu tive a oportunidade de trabalhar, e aprender, com os dois. Sem dúvidas, visitar a Universidad de Buenos Aires foi muito importante para a minha formação como doutor.

Um agradecimento especial aos meus amigos do LabAC, por todo o apoio que eu recebi durante o período deste doutorado. Em especial, aos colegas Alexsander Andrade de Melo, Alesom Zorzi, Edinelço Dalcumune e Wanderson Douglas Lomenha Pereira.

Agradeço aos amigos que tive a oportunidade de fazer na UFRJ, em especial à Danielle Alves Castelo Branco da Silva.

Agradeço à CAPES pelo suporte financeiro, se não fosse a bolsa de doutorado, certamente essa formação não seria possível.

Agradeço à UFRJ por ter me possibilitado essa formação, especialmente aos professores e demais funcionários do Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

FINURA DE PRECEDÊNCIA EM GRAFOS E ÁRVORES DE HAMMING-HUFFMAN RESTRITAS

Moysés da Silva Sampaio Júnior

Novembro/2022

Orientadores: Jayme Luiz Szwarcfiter
Fabiano de Souza Oliveira

Programa: Engenharia de Sistemas e Computação

Grafos de intervalo e grafos de intervalo próprio são classes de grafos bem conhecidas e para as quais existe uma ampla literatura relacionada. Como consequência, algumas generalizações foram propostas com o passar dos anos, nas quais grafos em geral são expressos por meio de k grafos de intervalo.

Um exemplo recente desse tipo de generalização são as classes dos grafos k-fino e k-fino próprio, que generalizam grafos de intervalo e grafos de intervalo próprio, respectivamente. Neste trabalho, é introduzido uma subclasse dos grafos k-fino (resp. k-fino próprio), chamada de grafos k-fino de precedência (resp. k-fino próprio de precedência). Com relação aos grafos k-fino de precedência particionados, é apresentado um algoritmo polinomial de reconhecimento baseado em árvores PQ. Considerando os grafos k-fino próprio de precedência particionados, é provado que o problema de reconhecimento relacionado é NP-completo para um k arbitrário e polinomial quando k é fixo. Além disso, é apresentado uma caracterização baseada em grafos de limiar para ambas as classes.

Na década de oitenta, Hamming propôs uma estrutura, denominada de árvore Hamming-Huffman (AHH), que une compressão de dados e detecção de erros.

Considerando as AHHs, esta tese define uma versão mais restrita desse tipo de estrutura, denominada $[k]$-AHH, que admite símbolos em no máximo k níveis distintos. Neste trabalho, é apresentado um algoritmo polinomial para a construção de [2]-AHHs ótimas. Para símbolos com frequências uniformes, é provado que uma AHH ótima é sempre uma [5]-AHH e que sempre existe uma AHH ótima que também é uma [4]-AHH. Por fim, considerando os resultados experimentais, é conjecturado que sempre existe uma AHH uniforme ótima que também é uma [3]-AHH.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

PRECEDENCE THINNESS OF GRAPHS AND RESTRICTED HAMMING-HUFFMAN TREES

Moysés da Silva Sampaio Júnior

November/2022

Advisors: Jayme Luiz Szwarcfiter
Fabiano de Souza Oliveira

Department: Systems Engineering and Computer Science

Interval and proper interval graphs are very well-known graph classes, for which there is a wide literature. As a consequence, some generalizations of interval graphs have been proposed, in which graphs in general are expressed in terms of k interval graphs, by splitting the graph in some special way.

As a recent example of such an approach, the classes of k-thin and proper k-thin graphs have been introduced generalizing interval and proper interval graphs, respectively. In this work, we introduce a subclass of k-thin graphs (resp. proper k-thin graphs), called precedence k-thin graphs (resp. precedence proper k-thin graphs). Concerning partitioned precedence k-thin graphs, we present a polynomial time recognition algorithm based on PQ trees. With respect to partitioned precedence proper k-thin graphs, we prove that the related recognition problem is NP-complete for an arbitrary k and polynomial-time solvable when k is fixed. Moreover, we present a characterization for these classes based on threshold graphs.

In the eighties, Hamming proposed a data structure, called Hamming-Huffman tree (HHT), in which both data compression and data error detection are tackled.

Considering Hamming-Huffman trees, we define a restricted version of this structure, called $[k]$-HHT, which admits symbol leaves in at most k different levels. We present an algorithm to build optimal [2]-HHTs. For uniform frequencies, we prove that an optimal HHT is always a [5]-HHT and that there exists an optimal HHT which is a [4]-HHT. Then, considering experimental results, we conjecture that there exists an optimal uniform tree which is a [3]-HHT.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis tackles problems of two research areas of computer science. The first one concerns to thinness in graphs, which is a concept that generalizes interval graphs. In particular, we study a subclass of k-thin (resp. proper k-thin) graphs. The second problem is related to Hamming-Huffman trees, which is a variation of Huffman trees that unites both data compression and data error detection.

This chapter aims to present an introduction for both these topics. It describes some related literature and the outline of the thesis. Furthermore, the list of publications that resulted from the work of the thesis.

1.1 Thinness in graphs

The class of k-thin graphs has been introduced by Mannino, Oriolo, Ricci and Chandran in [1] as a generalization of interval graphs. Motivated by this work, Bonomo and de Estrada [2] defined the class of proper k-thin graphs, which generalizes proper interval graphs. A *k-thin graph* G is a graph for which there is a k-partition $(V_1, V_2, \ldots, V_k)$, and an ordering s of $V(G)$ such that, for any triple (p, q, r) of $V(G)$ ordered according to s, if p and q are in a same part V_i and $pr \in E(G)$, then $qr \in E(G)$. Such an ordering and partition are said to be *consistent* with each other. For instance, for $C_4 = v_1, v_2, v_3, v_4, v_1$, both the partitioning $\mathcal{V} = \{\{v_1\}, \{v_2, v_3, v_4\}\}$ and the ordering $s = v_1, v_2, v_4, v_3$ are consistent with each other.

A graph G is called a *proper k-thin graph* if $V(G)$ admits a k-partition $(V_1, \ldots, V_k)$, and an ordering s such that both s and its reversal are consistent with the partition $(V_1, \ldots, V_k)$. An ordering of this type is said to be *strongly consistent* with the partition. For example, for $C_4 = v_1, v_2, v_3, v_4, v_1$, both the partitioning $\mathcal{V} = \{\{v_1, v_3\}, \{v_2, v_3\}\}$ and the ordering $s = v_1, v_2, v_3, v_4$ are strongly consistent with each other.

The interest on the study of both these classes comes from the fact that some NP-complete problems can be solved in polynomial time when the input graphs

belong to them [1–3]. Some of those efficient solutions have been exploited to solve real world problems as presented in [1].

On a theoretical perspective, defining general graphs in terms of the concept of interval graphs has been of recurring interest in the literature. Firstly, note that these concepts measure "how far" a given graph G is from being an interval graph, or yet, how G can be "divided" into interval graphs, mutually bonded by the existence of an ordering of the vertices holding a certain property. Namely, the vertices can be both partitioned and ordered in such a way that, for every part V' of the partition and every vertex $v \in V(G)$, the vertex ordering obtained by removing all vertices except v and those in V' that precede v is a canonical ordering of an interval graph. Characterizing general graphs in terms of the concept of interval graphs, or proper interval graphs, is not new. A motivation for such an approach is that the class of interval graphs is well-known, having several hundreds of research studies on an array of different problems on the class, and formulating general graphs as a function of interval graphs is a way to extend those studies to general graphs. Given that, both k-thin and proper k-thin graphs are also generalizations of this kind.

Other generalizations of interval graphs have been proposed. As examples, we may cite the k-interval and k-track interval graphs. A *k-interval* is the union of k disjoint intervals on the real line. A *k-interval graph* is the intersection graph of a family of k-intervals. Therefore, the k-interval graphs generalize the concept of interval graphs by allowing a vertex to be associated with a set of disjoint intervals. The *interval number* $i(G)$ of G [4] is the smallest number k for which G has a k-interval model. Clearly, interval graphs are the graphs with $i(G) = 1$. A *k-track interval* is the union of k disjoint intervals distributed in k parallel lines, where each interval belongs to a distinct line. Those lines are called *tracks*. A *k-track interval graph* is the intersection graph of k-track intervals. The *multitrack number* $t(G)$ of G [5, 6] is the minimum k such that G is a k-track interval graph. Interval graphs are equivalent to the 1-interval graphs and 1-track interval graphs. The problems of recognizing k-interval and k-track interval graphs are both NP-complete [7, 8], for every $k \geq 2$.

In this work, we define subclasses of k-thin and proper k-thin graphs called *precedence k-thin* and *precedence proper k-thin* graphs, respectively, by adding the requirement that the vertices of each class have to be consecutive in the order. In both cases, when the vertex order is given, it can be proved that a greedy algorithm can be used to find a (strongly) consistent partition into consecutive sets with minimum number of parts. When, instead, the partition into k parts is given, the problem turns out to be more interesting. We will call *partitioned precedence k-thin graphs* (resp. *partitioned precedence proper k-thin graphs*) the graphs for which there exists a (strongly) consistent vertex ordering in which the vertices of each part

are consecutive for the given partition. Concerning partitioned precedence k-thin graphs, we present a polynomial time recognition algorithm. With respect to partitioned precedence proper k-thin graphs, we provide a proof of NP-completeness for arbitrary k, and a polynomial time algorithm when k is fixed. Also, we provide a characterization for both classes based on threshold graphs.

The condition of precedence in thinness may favor the solution of some problems that might be harder for more general graphs. This is due to the fact that the precedence condition imposes a stronger and richer structure between vertices belonging to distinct parts of the partition. Property 3, in Section 3.3, is an example of a consequence of such a richer structure, which can be applied to the Hamiltonian path problem. If the input graph is a connected precedence proper 2-thin graph, having a partition for which each part induces a connected graph, it is straightforward to show that the graph admits a Hamiltonian path. Moreover, this path can be obtained in polynomial time.

1.2 Hamming-Huffman trees

In information theory, there is a common trade-off that arises in data transmission processes, in which two goals are usually tackled independently: data compression and data error detection. Paradoxically, these two goals have conflicting natures: while data compression shrinks the message as much as possible, data preparation for error detection adds redundancy to messages so that a receiver can detect corrupted bits, and possibly fix them. Data compression can be achieved using different strategies, often depending on the type of data being compressed. One of the most traditional methods is that of Huffman [9], which uses ordered trees, known as Huffman trees, to encode the symbols of a given message.

A Huffman tree assigns each symbol found on the message to be compressed to a new binary string, such that the total amount of data associated with the message, using this new encoding scheme, is the smallest as possible. Huffman trees achieve this goal by observing the frequencies of each symbol in the original message and assigning smaller codifications to higher-frequency symbols. A relevant aspect of Huffman trees is that this type of tree is proved to yield optimal codes.

In 1980, Hamming proposed the union of both compression and error detection features through a data structure called Hamming-Huffman tree [10]. This data structure compresses data similarly to Huffman trees with the additional feature of enabling the detection of any 1-bit error due to error transmission. In contrast to Huffman trees, building optimal Hamming-Huffman trees is still an open problem.

Due to its importance and relevance for practical applications, we aim to study exact algorithms to build optimal Hamming-Huffman trees. Since the problem is

still wide open, our approach is by constraining the number of levels of the tree at which its leaves can appear. This approach was employed in [11] to study Hamming-Huffman trees in which the leaves lie at a single level.

Therefore, in this work, we tackle the problem of building optimal Hamming-Huffman trees in which the leaves lie in exactly k distinct levels. If $k \leq 2$, we provide a polynomial time algorithm to solve the problem. Otherwise, we provide an algorithm to evaluate a lower bound on the optimal cost of such trees when the symbols have a uniform probability of occurrence. In this case, we also prove that there exists an optimal Hamming-Huffman tree having their symbol leaves lying on at most 4 consecutive levels.

1.3 Published results

This section presents conference participations and journal publications derived from the work of this doctorate.

1.3.1 Participation in Conferences

2022 Min C. Lin, Fabiano S. Oliveira, Paulo E. D. Pinto, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter
Restricted Hamming-Huffman trees,
10th Latin American Workshop on Cliques in Graphs.

2020 Flavia Bonomo-Braberman, Fabiano S. Oliveira, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter,
A complexidade do reconhecimento de grafos k-fino próprio de precedência,
V Encontro de Teoria da Computação, ETC 2020.

2020 Min C. Lin, Fabiano S. Oliveira, Paulo E. D. Pinto, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter
Two Level Hamming-Huffman Trees,
9th Latin American Workshop on Cliques in Graphs.

2019 Flavia Bonomo-Braberman, Fabiano S. Oliveira, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter,
Reconhecimento de Grafos Fino de Precedência,
IV Encontro de Teoria da Computação, ETC 2019.

2018 Flavia Bonomo-Braberman, Fabiano S. Oliveira, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter,
Sobre Finura Própria de Grafos,
III Encontro de Teoria da Computação.

2018 Fabiano S. Oliveira, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter,
On a Class of Proper 2-thin Graphs,
VIII LATIN AMERICAN WORKSHOP ON CLIQUES IN GRAPHS.

1.3.2 Publications in Journals and Submitted Papers

2022 Flavia Bonomo-Braberman, Fabiano S. Oliveira, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter
Precedence thinness in graphs,
Discrete Applied Mathematics, Volume 323, P. 76-95, 15 December 2022

2022 Min C. Lin, Fabiano S. Oliveira, Paulo E. D. Pinto, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter
Restricted Hamming-Huffman trees,
RAIRO-Oper. Res., Volume 56, Number 3, P. 1823 - 1839, May-June 2022

2022 Flavia Bonomo-Braberman, Carolina L. Gonzalez, Fabiano S. Oliveira, Moysés S. Sampaio Jr., Jayme L. Szwarcfiter
Thinness of product graphs,
Discrete Applied Mathematics, Volume 312, P. 52-71, 15 May 2022

This thesis focus mainly on the results of the papers entitled Precedence thinness in graphs and Restricted Hamming-Huffman trees. It is worth mentioning that parts of these works were developed in the course of a scientific visit to the university of Buenos Aires during September of 2018, where I worked with the professor Flavia Bonomo-Braberman and the professor Min Chih Lin. Also, this visit resulted in the paper entitled Thinness of product graphs where other types of (proper) thinness have been defined and the behavior of (proper) thinness over some graph products have been studied.

1.4 Outline of the work

This work is organized in 5 chapters. Chapter 2 introduces the necessary concepts and basic terminology employed throughout the text.

Chapter 3 presents some results for the class of precedence k-thin graphs, as described next. Section 3.1 presents a polynomial time recognition algorithm for the class of precedence k-thin graphs. Section 3.2 proves the NP-completeness for the recognition problem of precedence proper k-thin graphs. Moreover, it proves that if the number of parts of the input graph is fixed, then the problem is solvable in polynomial time. Section3.3 presents a characterization for (proper) k-thin graphs based on threshold graphs.

Chapter 4 presents some results regarding Hamming-Huffman trees, as following described. In Section 4.1, the problem of building Hamming-Huffman trees in which all symbol leaves lie on the same level is tackled. In Section 4.2, the problem of building Hamming-Huffman trees in which the leaves are distributed in two distinct levels is discussed. In Section 4.3, its is proved that, for symbols with a uniform probability of occurrence, there is always an optimal Hamming-Huffman tree such that its symbol leaves lie on at most four consecutive levels. Also, it is presented an algorithm to evaluate a lower bound on the cost of such trees. In Section 4.4, some experimental results are presented.

Finally, in Chapter 5, some concluding remarks and proposals concerning the problems to be further investigated.

Chapter 2

Preliminaries

All graphs in this work are finite and have no loops or multiple edges. Let G be a graph. Denote by $V(G)$ its vertex set, by $E(G)$ its edge set. Denote the *size* of a set S by $|S|$. Unless stated otherwise, $|V(G)| = n$ and $|E(G)| = m$. Let $u, v \in V(G)$, u and v are *adjacent* if $uv \in E(G)$.

Let $V' \subseteq V(G)$. The *subgraph* of G *induced* by the subset of vertices V', denoted by $G[V']$, is the graph $G[V'] = (V', E')$, where $E' = \{uv \in E(G) \mid u, v \in V'\}$. An *induced subgraph* of G is a subgraph of G induced by some subset of $V(G)$. A graph G' obtained from G by *removing* the vertex $v \in V(G)$ is $G' = G[V(G) \setminus \{v\}]$.

Let $u \in V(G)$. The *(open) neighborhood* of u, denoted by $N_G(u)$, is defined as $N_G(u) = \{v \in V(G) \mid (u, v) \in E(G)\}$. The *closed neighborhood* of u is denoted by $N_G[u] = \{u\} \cup N_G(u)$. Let $U \subseteq V(G)$. Define $N_G(U) = \bigcup_{u \in U} N_G(u)$ and $N_G[U] = \bigcup_{u \in U} N_G[u]$. When G is clear in the context, it may be omitted from the notation.

We define $u, v \in V(G)$ as *true twins* (resp. *false twins*) if $N[u] = N[v]$ (resp. $N(u) = N(v)$). A vertex v of G is *universal* if $N[v] = V(G)$. We define the *degree* of $v \in V(G)$, denoted by $d(v)$, as the number of neighbors of v in G, i.e. $d(v) = |N(v)|$.

A *clique* or *complete set* (resp. *stable set* or *independent set*) is a set of pairwise adjacent (resp. nonadjacent) vertices. We use *maximum* to mean maximum-sized, whereas *maximal* means that such a set is not properly contained in any other set. The use of *minimum* and *minimal* is analogous. A vertex $v \in V(G)$ is said to be *simplicial* if $G[N(v)]$ is a clique.

A *coloring* of a graph is an assignment of colors to its vertices, each vertex assigned to a color, such that any two adjacent vertices are assigned different colors. The smallest number t such that G admits a coloring with t colors (a *t-coloring*) is called the *chromatic number* of G and is denoted by $\chi(G)$. A coloring defines a partition of the vertices of the graph into stable sets, called *color classes*.

Let $G = (V, E)$, the *complement* of G is defined as $G' = (V, E')$ such that $E' = \{uv \mid u, v \in V \text{ and } uv \notin E\}$.

A graph $G = (V, E)$ is a *comparability graph* if there exists an ordering $v_1, \ldots, v_n$ of V such that, for each triple (r, s, t) with $r < s < t$, if $v_r v_s$ and $v_s v_t$ are edges of G, then so is $v_r v_t$. Such an ordering is a *comparability ordering*. A graph is a *co-comparability graph*

if its complement is a comparability graph.

A *tree* T is a connected graph that has no cycles. A *rooted tree* T_v is a tree in which a vertex $v \in V(T)$ is labeled as the *root* of the tree, and all vertices, known as *nodes* of T_v, are classified in relation to the root. Let $u, w \in V(T_v)$, w is a *descendant* of u if the path from w to v includes u. The node w is said to be a *child* of u if it is a descendant of u and $wu \in E(T_v)$. The *children* of u is defined as the set containing all child nodes of u. The node u is said to be a *leaf* of T_v if it has no child in T_v. The *subtree* T_u of T_v *rooted at* the node u is the *rooted* tree that consists of u as the root and its descendants in T_v as the nodes.

A *directed graph*, or *digraph*, D is formed by a set of vertices $V(D)$ and a set of *directed edges*, or *arcs*, $E(D)$ such that $E(D)$ consists of ordered pairs of $V(D)$. Analogously to graphs, unless stated otherwise, $|V(D)| = n$ and $|E(D)| = m$. A *directed cycle* of a digraph D is a sequence $v_1, v_2, \ldots, v_i$, $1 \leq i \leq n$, of vertices of $V(D)$ such that $v_1 = v_i$ and, for all $1 \leq j < i$, $(v_j, v_{j+1}) \in E(D)$. A *directed acyclic graph* (*DAG*) D is a digraph with no directed cycles. A *topological ordering* of a DAG D is a sequence $v_1, v_2, \ldots, v_n$ of $V(D)$ such that there are no $1 \leq i < j \leq n$ such that $(v_j, v_i) \in E(D)$. Determining a topological ordering of a DAG can be done in time $O(n + m)$ [12].

An *ordering* s of elements of a set $\mathcal{C}$ consists of a sequence $e_1, e_2, \ldots, e_n$ of all elements of $\mathcal{C}$. The set $\mathcal{C}$, corresponding to all elements in an ordering s, is denoted by $V(s)$. We define $\bar{s}$ as the *reversal* of s, that is, $\bar{s} = e_n, e_{n-1}, \ldots, e_1$. We say that e_i *precedes* e_j in s, denoted by $e_i < e_j$, if $i < j$. An ordered tuple $(a_1, a_2, \ldots, a_k)$ of some elements of $\mathcal{C}$ is *ordered according to* s when, for all $1 \leq i < k$, a_i precedes a_{i+1} in s. Given orderings s_1 and s_2, the ordering obtained by *concatenating* s_1 and s_2, i.e., where a precedes a' if either $a < a'$ in s_1 or $a < a'$ in s_2 or $a \in V(s_1)$ and $a' \in V(s_2)$, is denoted by $s_1 s_2$.

2.1 Interval graphs

An *interval graph* G is a graph in which $V(G)$ is a family of intervals on the real line such that for any two intervals $I, J \in V(G)$, $IJ \in E(G)$ if and only if $I \cap J \neq \emptyset$. A family $\mathcal{R}$ of intervals that can be represented by $G(V)$ is said to be an *interval model*, or a *model*, of G. We say that $\mathcal{R}$ is *associated with* G and vice-versa. It is worth mentioning that an interval graph can be associated to several models but an interval model can be associated to a unique graph. Concerning interval graphs, there are some characterizations that are relevant to this work, which we present next.

Theorem 1 ([13])**.** *A graph G is an interval graph if, and only if, there is an ordering s of $V(G)$ such that, for any triple (p, q, r) of $V(G)$ ordered according to s, if $pr \in E(G)$, then $qr \in E(G)$.*

The ordering described in Theorem 1 is said to be a *canonical* ordering. Figure 2.1(a) depicts an interval graph in which the vertices are presented from left to right in one of its canonical orderings. An interval model of this graph is presented in Figure 2.2.

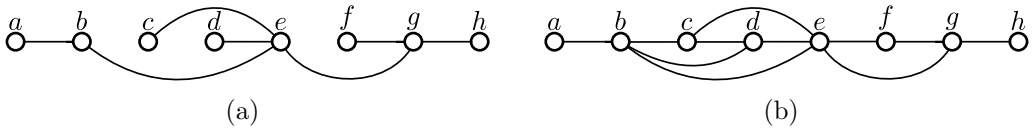


Figure 2.1: (a) Canonical ordering and (b) proper canonical ordering.

Let $\mathcal{R}$ be an interval model of an interval graph G. Note that, if we consider a vertical line that intersects a subset of intervals in $\mathcal{R}$, then these intervals form a clique in G. This is true because they all contain the point in which this line is defined. Moreover, if the given line traverses a maximal set of intervals, then the corresponding clique is maximal. Figure 2.2 depicts an interval model and the vertical lines that correspond to maximal cliques of the graph in Figure 2.1(a). The following is a characterization of interval graphs in terms of the maximal cliques of the graph.

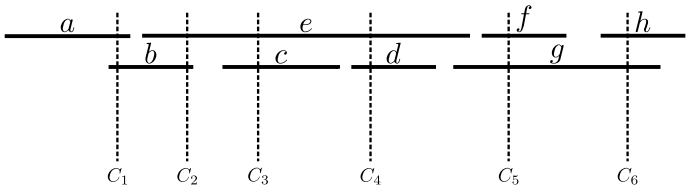


Figure 2.2: An interval model and the maximal cliques of the graph in Figure 2.1(a).

Theorem 2 ([14]). *A graph G is an interval graph if, and only if, there is an ordering $s_C = C_1, C_2, \ldots, C_k$ of its maximal cliques such that if $v \in C_i \cap C_z$ with $1 \leq i \leq z \leq k$, then $v \in C_j$, for all $i \leq j \leq z$.*

The ordering s_C described in Theorem 2 is said to be a *canonical clique* ordering. The ordering of the maximal cliques depicted in Figure 2.2, read from left to right, represents a canonical clique ordering. Let $v \in V(G)$, we define $s_C(v)$ as the index of the first maximal clique in the ordering s_C containing v.

A *proper interval graph* is an interval graph that admits an interval model in which no interval properly contains another. There is a characterization of proper interval graphs which is similar to that presented in Theorem 1, as described next.

Theorem 3 ([14]). *A graph G is a proper interval graph if, and only if, there is an ordering s of $V(G)$ such that, for any triple (p, q, r) of $V(G)$ ordered according to s, if $pr \in E(G)$, then $pq, qr \in E(G)$.*

The ordering specified in the previous theorem is defined as a *proper canonical* ordering. The ordering of the vertices of the graph depicted in Figure 2.1(b) from left to right is proper canonical. Note that a proper canonical ordering is also a canonical ordering. An important property of proper canonical orderings is the following:

Lemma 4 ([14]). *If G is a connected proper interval graph, then a proper canonical ordering of G is unique up to reversion and permutation of mutual true twin vertices.*

It can be proved that the vertices of each connected component of an interval graph constitute a block of consecutive vertices in every canonical ordering of the vertices of the graph. As a consequence, if a (proper) interval graph G is disconnected, then every canonical ordering of $V(G)$ consists of a permutation of canonical orderings of each of its components. For the sake of simplicity, we often refer to (proper) canonical orderings of $V(G)$ as (proper) canonical orderings of G.

Let s be an ordering of $V(G)$ and $s_C = C_1, C_2, \ldots, C_k$ an ordering of the maximal cliques of G. The sequence s is said to be *ordered according* to s_C if for all $u, v \in V(G)$ such that $u < v$ in s, there are no $1 \leq i < j \leq k$ such that $v \in C_i \setminus C_j$ and $u \in C_j$. As an example, note that the canonical ordering of the Figure 2.1(a) is ordered according to the maximal clique ordering $s_C = C_1, C_2, \ldots, C_6$ of the Figure 2.2. The following lemma relates the characterization of canonical orderings and canonical clique orderings.

Lemma 5. *Let G be a graph and s be an ordering of $V(G)$. The ordering s is a canonical ordering of $V(G)$ if, and only if, s is ordered according to a canonical clique ordering of G.*

Proof. Consider s_C a canonical clique ordering of G. We will show that it is possible to build a canonical ordering s from s_C, respecting its clique ordering. To achieve this, first start with an empty sequence s. Iteratively, for each element X of the sequence s_C, choose all simplicial vertices of X, adding them to s in any order and removing them from G. Note that the removal of the simplicial vertices of X can turn other non simplicial vertices of G into simplicial ones. Clearly, at the end of the process, s will contain all the vertices of G. Suppose s is not a canonical ordering, that is, there are $p, q, r \in V(G)$, $p < q < r$ in s, such that $pr \in E(G)$ and $qr \notin E(G)$. Let C_p, C_q and C_r be the maximal cliques of s_C being processed at moment p, q and r were choose, respectively. Note that $C_p < C_q < C_r$ in S_c and, as C_p is the last maximal clique in s_C that contains p and $pr \in E(G)$, $r \in C_p$. Besides, as $qr \notin E(G)$, $r \notin C_q$. Therefore, $r \in C_p \cap C_r$ and $r \notin C_q$. A contradiction with the fact that s_C is a canonical clique ordering. Hence, s is a canonical ordering.

Let $s = v_1, v_2, \ldots, v_n$ be a canonical ordering of G. We prove by induction on $|V(G)| = n$ that s is ordered according to a canonical clique ordering. Clearly, the statement is true for $n = 1$. Suppose that the statement is true for any $1 \leq n' < n$. Let s' be the sequence obtained from s by removing v_n. Clearly s' is also a canonical ordering and, by the induction hypothesis, s' is ordered according with a canonical clique ordering $s'_C = C_1, C_2, \ldots, C_k$. Let C_i, with $1 \leq i \leq k$, be the first clique of s'_C such that $v_n \in N(C_i)$. Let $C'_j = \{C_j \cap N(v_n)\} \cup \{v_n\}$, $i \leq j \leq k$. Note that, as s is a canonical ordering, for all C_j of s'_C, $i < j \leq k$, $\{C_j \cap N(v_n)\} \cup \{v_n\}$ is a maximal clique of G. If $C_i \subseteq N(v_n)$, then $s'_C = C_1, C_2, \ldots, C'_i, C'_{i+1}, \ldots, C'_k$ is a canonical clique ordering that matches s. Otherwise, $s'_C = C_1, C_2, \ldots, C_i, C'_i, C'_{i+1}, \ldots, C'_k$ is a canonical clique ordering that matches s. Therefore, s is ordered according to a canonical clique ordering of G. □

2.2 PQ trees

A *PQ tree* [15] T is a data structure consisting of an ordered tree that describes a family $\mathcal{F}$ of permutations of elements from a given set $\mathcal{C}$. In a PQ tree T, the set of leaves is $\mathcal{C}$ and the permutation being represented by T is the sequence of the leaves from left to right. Regarding internal nodes, they are classified into two types, the P and the Q nodes. A PQ tree T' is *equivalent* to T if it is obtained from T by any sequence of consecutive transformations, each consisting of either permuting the children of a P node, or reversing the children of a Q node. The family $\mathcal{F}$ of permutations of elements from $\mathcal{C}$ represented by T is that of permutations corresponding to all PQ trees equivalent to T.

Graphically, in a PQ tree, leaves and P nodes are represented by circles and Q nodes by rectangles. In representations of schematic PQ trees, a node represented by a circle over a rectangle will denote that, in any concrete PQ tree conforming the scheme, such a node is either a P node, or a Q node, or a leaf. Figure 2.3 depicts the described operations. In Figure 2.3(a), we have a partial representation of a PQ tree. Figure 2.3(b) depicts an equivalent PQ tree obtained from a permutation of the children of a P node of this tree and Figure 2.3(c) exemplifies an equivalent PQ tree obtained from the reversion of the children of a Q node.

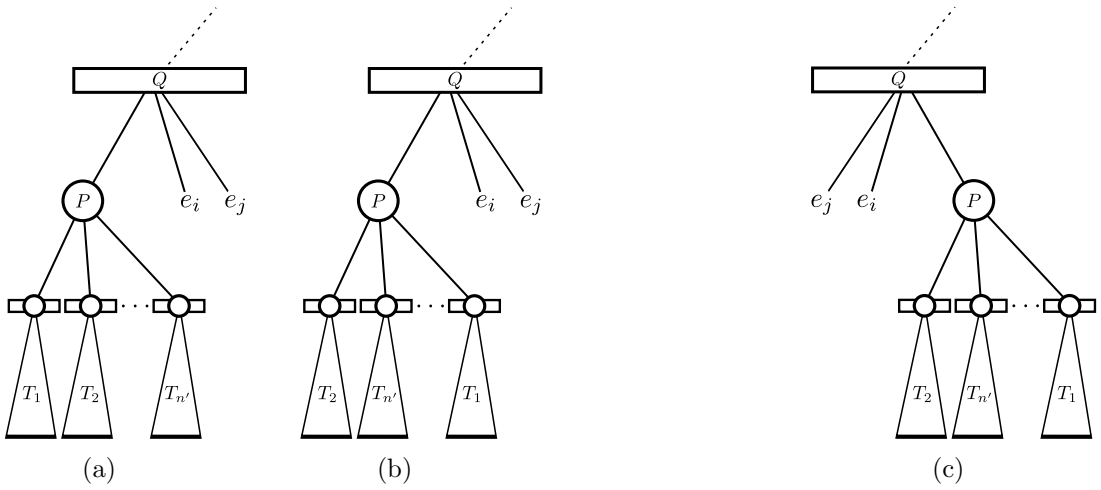


Figure 2.3: A PQ tree (a) and an example of permutations (b) and reversions (c) of the children of its nodes.

One of the applications from the seminal paper introducing PQ trees is that of recognizing interval graphs. In such an application, each leaf of the PQ tree is a maximal clique of an interval graph G and the family of permutations the tree represents is precisely all the canonical clique orderings of G [15]. A PQ tree can be constructed from an interval graph in time $O(n + m)$ [15].

We will say that a vertex v belongs to a node X of a PQ tree, and naturally denote by $v \in X$, if it belongs to any leaf that descends from this node. Figure 2.4 depicts a PQ tree of the interval graph of Figure 2.1(a) according to the maximal cliques in Figure 2.2. The permutations implicitly represented by this PQ tree are:

- $C_1, C_2, C_3, C_4, C_5, C_6$
- $C_1, C_2, C_3, C_4, C_6, C_5$
- $C_1, C_2, C_4, C_3, C_5, C_6$
- $C_1, C_2, C_4, C_3, C_6, C_5$
- $C_6, C_5, C_4, C_3, C_2, C_1$
- $C_5, C_6, C_4, C_3, C_2, C_1$
- $C_6, C_5, C_3, C_4, C_2, C_1$
- $C_5, C_6, C_3, C_4, C_2, C_1$

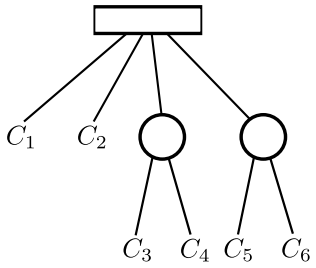


Figure 2.4: A PQ tree of the interval graph of Figure 2.1(a) according to the maximal cliques in Figure 2.2.

2.3 Thinness and proper thinness

A graph G is called a *k-thin graph* if there is a k-partition $(V_1, V_2 \ldots, V_k)$ of $V(G)$ and an ordering s of $V(G)$ such that, for any triple (p, q, r) of $V(G)$ ordered according to s, if p and q are in a same part V_i and $pr \in E(G)$, then $qr \in E(G)$. An ordering and a partition satisfying that property are called *consistent*. That is, a graph is k-thin if there is an ordering consistent with some k-partition of its vertex set. The *thinness* of G, denoted by $thin(G)$, is the minimum k for which G is a k-thin graph.

A graph G is said to be a *proper k-thin graph* if G admits a k-partition $(V_1, \ldots, V_k)$ of $V(G)$ and an ordering s of $V(G)$ consistent with the partition and, additionally, for any triple (p, q, r) of $V(G)$, ordered according to s, if q and r are in a same part V_i and $pr \in E(G)$, then $pq \in E(G)$. Equivalently, an ordering s of $V(G)$ such that s and its reverse are consistent with the partition. Such an ordering and partition are called *strongly consistent*. The *proper thinness* of G, or $pthin(G)$, is the minimum k for which G is a proper k-thin graph.

Figures 2.5(a) and 2.5(b) depict two 2-partitions of a graph, in which the classes are represented by distinct colors, and two different vertex orderings. The ordering of Figure 2.5(a) is consistent with the corresponding partition but not strongly consistent, while the ordering of Figure 2.5(b) is strongly consistent with the corresponding partition.

Note that k-thin graphs (resp. proper k-thin graphs) generalize interval graphs (resp. proper interval graphs). The 1-thin graphs (resp. proper 1-thin graphs) are the interval graphs (resp. proper interval graphs). The parameter thin(G) (resp. pthin(G)) is in a way a measure of how far a graph is from being an interval graph (resp. proper interval graph).

For instance, consider the graph C_4. Since C_4 is not an interval graph, pthin$(C_4) \geq$ thin$(C_4) > 1$. Figure 2.5 proves that thin$(C_4) =$ pthin$(C_4) = 2$.

The complexity of deciding whether the thinness of a graph is at most k is NP-complete (cf [16]). On the other hand, the complexity of recognizing whether a graph is

proper k-thin, is still an open problem even for a fixed $k \geq 2$. For a given vertex ordering, there are polynomial time algorithms that compute a partition into a minimum number of classes for which the ordering is consistent (resp. strongly consistent) [2, 3]. However, given a vertex partition, the problem of deciding the existence of a vertex ordering which is consistent (resp. strongly consistent) with that partition is NP-complete [2].

A characterization of k-thin or proper k-thin graphs by forbidden induced subgraphs is only known for k-thin graphs within the class of cographs [2]. Graphs with arbitrary large thinness were presented in [1], while in [2] a family of interval graphs with arbitrary large proper thinness was used to show that the gap between thinness and proper thinness can be arbitrarily large. The relation of thinness and other width parameters of graphs like boxicity, pathwidth, cutwidth and linear MIM-width was shown in [1, 2]. In [17] the behavior of (proper) thinness over graph products are studied. Moreover, some other types of thinness are presented, which differs from each other by the constraints applied to each partition.

Let G be a graph and s an ordering of its vertices. The graph G_s has $V(G)$ as vertex set, and $E(G_s)$ is such that for all $v < w$ in s, $vw \in E(G_s)$ if and only if there is a vertex z in G such that $v < w < z$, $zv \in E(G)$ and $zw \notin E(G)$. Similarly, the graph $\tilde{G}_s$ has $V(G)$ as vertex set, and $E(\tilde{G}_s)$ is such that for all $v < w$ in s, $vw \in E(\tilde{G}_s)$ if and only if either $vw \in E(G_s)$ or there is a vertex x in G such that $x < v < w$, $xw \in E(G)$ and $xv \notin E(G)$.

Theorem 6. *[2, 3] Given a graph G and an ordering s of its vertices, a partition of $V(G)$ is consistent (resp. strongly consistent) with the ordering s if and only if the partition is a valid coloring of G_s (resp. $\tilde{G}_s$), which means that each part corresponds to a color class in the coloring under consideration.*

(a) (b)

Figure 2.5: (a) A consistent ordering and a (b) strongly consistent ordering of $V(C_4)$, for the corresponding 2-partitions.

2.4 Hamming-Huffman trees

An *n-cube* or a *hypercube* with *dimension* n, is the graph Q_n having $V(Q_n)$ as the set of all binary strings with size n (and, therefore, $|V(Q_n)| = 2^n$). Moreover, $(u, v) \in E(Q_n)$ if the binary strings of u and v differ exactly in one position. Let $u, v \in V(Q_n)$, the *Hamming distance* between u and v, denoted by $d(u, v)$, is the number of positions in which the binary strings of u and v differ.

The *parity* of a binary string v is the parity of the number of 1's in v.

We define the *minimum neighborhood* over independent sets with size ℓ of Q_n as

$$\varphi(\ell, n) = \min\{|N(L)| \mid L \subset V(Q_n),\ |L| = \ell \text{ and } L \text{ is an independent set of } Q_n\}$$

A *strict binary tree* is a rooted tree such that each node has either two or zero children. The *level* of a node is the number of edges on the path from this node up to the root of the tree. A *full binary tree* is a strict binary tree in which all the leaves are at the same level. The *height* of a tree is the maximum level over all its nodes.

In the context of data compression techniques, an important data structure is the *Huffman tree*. A *Huffman tree* (HT) T is a rooted strict binary tree in which each edge (u, v), v being a left (resp. right) child of u, is labeled by 0 (resp. 1) and the set of leaves of T is Γ, the set of all distinct symbols of which a message M to be sent consists. Given T, each symbol a of M is sequentially encoded into a binary string $c(a)$. Such encoding is given by the sequence of 0's and 1's found on the edges of the directed path from the root of T to the leaf corresponding to a. In Figure 2.6(a), for instance, the leaves are encoded, reading them from left to right, as 00, 010, 011, 10, and 11. Over all possible trees, the HT for M is a tree T such that its cost

$$c(T) = \sum_{a \in \Gamma} p(a)|c(a)|$$

is minimum, where $p(a)$ stands for the probability of occurrence of a in the message and $|c(a)|$ is the length of the string $c(a)$. We say that an HT is *uniform* if all of its symbols have a uniform probability of occurrence, that is, each symbol has a probability of occurrence of $\frac{1}{|\Gamma|}$. Figure 2.6(a) depicts a uniform HT T with $c(T) = 2.4$ on 5 symbols.

The concept of Hamming-Huffman trees generalizes that of Huffman trees. A *Hamming-Huffman tree* (HHT) is a strict binary tree holding the same properties of an HT, except that the set of leaves is partitioned into symbol and error leaves, such that the following properties hold:

- every node e of T such that $d(c(e), c(a)) = 1$, for some symbol leaf $a \in \Gamma$, is a leaf of T called an *error leaf*;
- every node of T is either an error leaf or an ancestor of a symbol leaf.

HHTs can be applied to detect errors which occurs in the transmission of messages. Under the assumption that when data is transmitted at most one bit can accidentally be flipped, HHTs detect such errors during the decoding process: if an error leaf is hit, the data has been corrupted during transmission. Optimal HHTs are defined exactly the same as (optimal) HTs. Figure 2.6(b) depicts an optimal uniform HHT T with $c(T) = 3.8$ on 5 symbols. In the presented figures, error leaves are colored black.

Although HTs can be built efficiently in a greedy fashion [9], the construction of optimal HHTs is open since defined by Hamming in the eighties [10]. An approximation algorithm having time $O(n \log^3 n)$ was presented in [18, 19] with a low additive error with respect to the entropy.

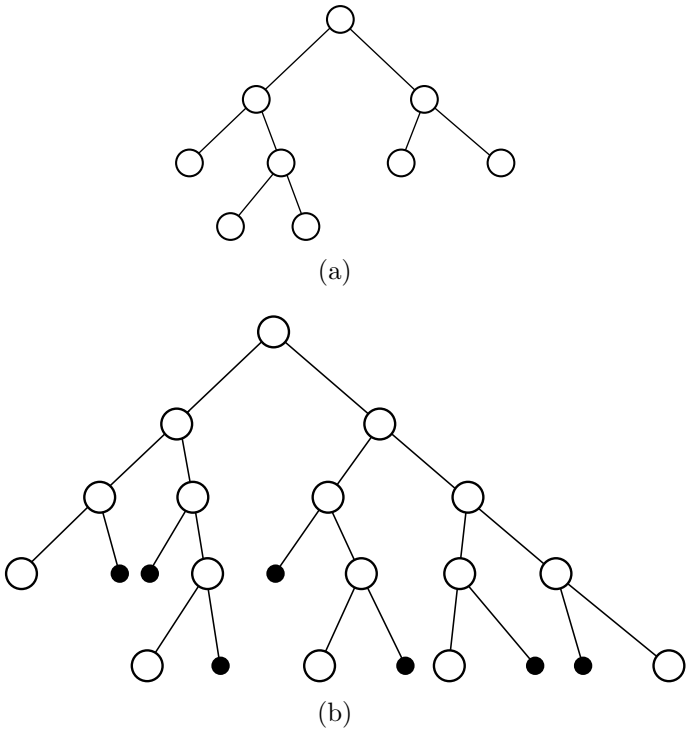


Figure 2.6: Examples of (a) uniform Huffman and (b) optimal uniform Hamming-Huffman trees, for 5 symbols.

Hamming-Huffman trees inspired other codes, such as in [20, 21], in which the authors proposed a code called even codes. Even codes are obtained by a special type of Huffman tree in which symbols are allowed to be encoded only with an even number of ones. In this type of tree, nodes associated with an odd number of ones are either internal nodes or error leaves.

The authors of that code provided an algorithm to build such a code in time $O(n^3 \log n)$. In [22], the algorithm was improved to have time complexity of $O(n^3)$. They also presented two approximation algorithms for even codes having time complexity of $O(n \log n)$, the second achieving a code having cost 16.7% higher than that of Huffman trees. Related/constrained problems are studied in [23].

Figure 2.7 depicts an optimal Huffman tree, an optimal even tree and an optimal Hamming-Huffman tree for three symbols with uniform frequencies.

In this work, we will weaken the concept of HHTs to enlarge the number of trees which could be called HHTs. This is convenient to our algorithms and proofs, without losing the motivation associated with the application related to detecting error capabilities.

If a tree T is an HHT as previously defined, we will say that T is a *strict HHT*. The weaker definition of HHTs is the following. Given a strict binary tree T, in which the set of leaves is partitioned into symbol and error leaves, T is an HHT if $contract(T)$ is a strict HHT, where $contract(T)$ is a tree transformation defined as follows:

- if T has no sibling error leaves, then no modification to the tree is done.

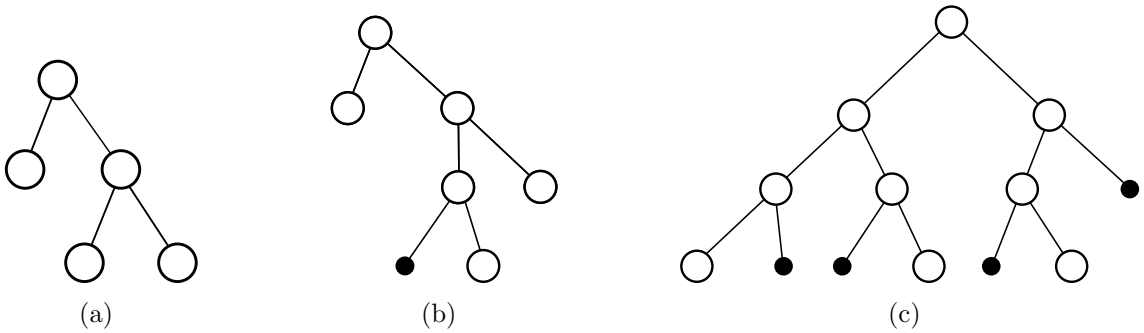


Figure 2.7: Optimal uniform HT(a), uniform even tree (b) and uniform HHT(c) for three symbols.

- if the sibling node of an error leaf e is an error leaf, remove both e and its sibling from T and make the node p, parent of e in T, into an error leaf. Let T' be the resulting tree. Apply $contract(T')$ recursively to obtain the final transformation.

Figure 2.8(a) presents an HHT T in this weaker sense, and Figure 2.8(b) shows the strict HHT resulting of $contract(T)$. Note that $contract(T)$ can be done in time $O(\ell)$.

The number of error leaves in HHTs is directly related to the encodings associated with the symbol leaves. In Figure 2.8(a), symbol leaves are encoded as 000 and 111, resulting in 6 error leaves, whereas in Figure 2.8(c), symbol leaves are encoded as 000, 011, 101, and 110, resulting in 4 corresponding error leaves. In this second HHT, two more symbols are being encoded using the same full binary tree as the one used in the first HHT.

In this work, we approach the problem by defining a constrained version of Hamming-Huffman trees, namely that of determining an optimal HHT T in which the symbol leaves are placed in exactly k distinct levels. A tree for which this property holds will be called a *k-Hamming-Huffman tree*, or *k-HHT*. In Chapter 4, the problem of k-HHT is discussed.

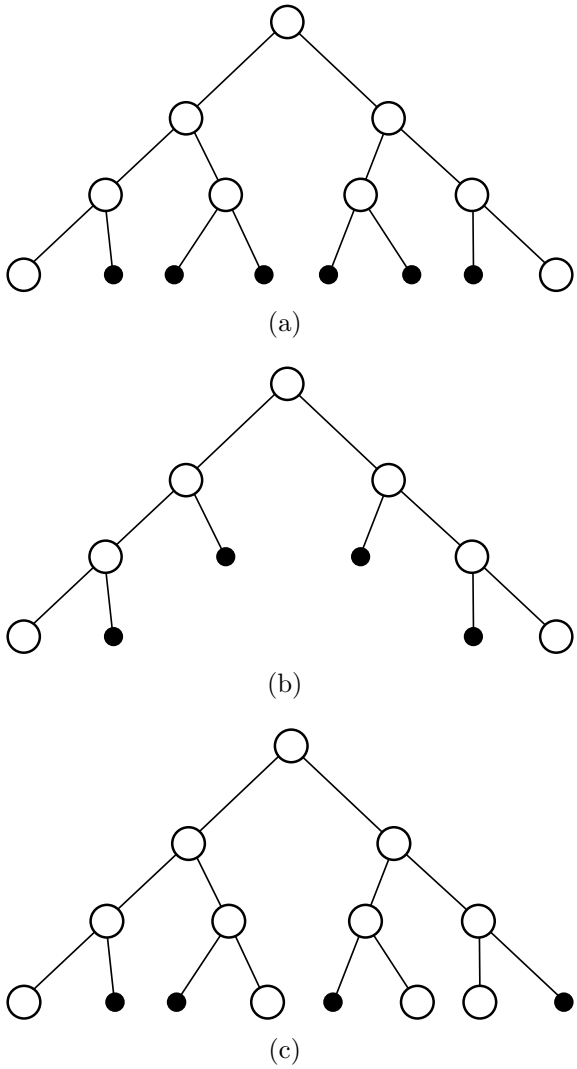


Figure 2.8: Examples of Hamming-Huffman trees.

Chapter 3

Precedence Thinness

In this chapter, we consider a variation of the concept of (proper) thinness in graphs by requiring that, given a vertex partition, the (strongly) consistent orderings hold an additional property. This class is defined as follows.

A graph G is *precedence k-thin* (resp. *precedence proper k-thin*), or *k-PT* (resp. *k-PPT*), if there is a k-partition of its vertices and a consistent (resp. strongly consistent) ordering s for which the vertices that belong to a same part are consecutive in s. Such an ordering is called a *precedence consistent ordering* (resp. *precedence strongly consistent ordering*) for the given partition. We define *pre-thin*(G) (resp. *pre-pthin*(G)) as the minimum value k for which G is k-PT (resp. k-PPT). A k-PT (resp. k-PPT) graph is a k-thin (resp. proper k-thin) graph. Therefore, the class of k-PT (resp. k-PPT) graphs is a proper subclass of that of k-thin (resp. proper k-thin) graphs.

Figure 3.1 illustrates a graph that is a 2-PPT graph. The convention assumed is that the strongly consistent ordering being represented consists of the vertices ordered as they appear in the figure from bottom to top and, for vertices arranged in a same horizontal line, from left to right. Therefore, the strongly consistent ordering represented in Figure 3.1 is $s = a, b, c, a', b', c'$. The graph C_4 is not 2-PPT, despite $\text{pthin}(C_4) = 2$. It can be easily verified by brute-force that, for all possible 2-partitions of its vertex set and for all possible orderings s in which the vertices of a same part are consecutive in s, the ordering and the partition are not strongly consistent.

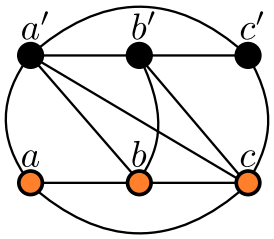


Figure 3.1: A 2-PPT graph.

If a vertex order s is given, by Theorem 6, any partition which is precedence (strongly) consistent with s is a valid coloring of G_s (resp. $\tilde{G}_s$) such that, additionally, the vertices on each color class are consecutive according to s. A greedy algorithm can be used to

find a minimum vertex coloring with this property in polynomial time. Such method is described next, in Theorem 7.

Theorem 7. *Let G be a graph and s be an ordering of $V(G)$. It is possible to obtain a minimum k-partition $\mathcal{V}$ of $V(G)$, in polynomial time, such that s is a precedence (strongly) consistent ordering with respect to $\mathcal{V}$.*

Proof. Consider the following greedy algorithm that obtains an optimum coloring of G_s (resp. $\tilde{G}_s$) in which vertices having a same color are consecutive in s. That is, s is a precedence consistent (resp. precedence strongly consistent) ordering concerning the partition defined by the coloring. Color v_1 with color 1. For each v_i, $i > 1$, let c be the last color used. Then color v_i with color c if there is no v_j, $j < i$, colored with c such that $v_jv_i \in E(G_s)$ (resp. $v_jv_i \in E(\tilde{G}_s)$). Otherwise, color v_i with color $c + 1$. We show, by induction on $|s| = n$, that the algorithm finds an optimal coloring in which each vertex has the least possible color.

The case where $n = 1$ is trivial. Suppose that the algorithm obtains an optimum coloring of G_s (resp. $\tilde{G}_s$), for orderings having size less than n. Remove the last vertex v_n from s and G_s (resp. $\tilde{G}_s$) and use the given algorithm to color the resulting graph. By the induction hypothesis, the chosen coloring for $v_1, v_2, \ldots, v_{n-1}$ is optimal. Moreover, the colors are non-decreasing and each vertex is colored with the least possible color. Now, add the removed vertex v_n to s and to the graph G_s (resp. $\tilde{G}_s$), with its respective edges, and let the algorithm choose a coloring for it. If the color of v_n is equal to the color of v_{n-1} the algorithm is optimal by the induction hypothesis. Otherwise, v_n is colored with a new color c'. Suppose the chosen coloring is not optimal. That is, it is possible to color v with an existing color. This implies that there is at least a neighbor v_j of v_n, in G_s (resp. $\tilde{G}_s$), that can be recolored with a smaller color. This is an absurd because the algorithm has already chosen the least possible color for all the vertices of $G_s \setminus \{v_n\}$ (resp. $\tilde{G}_s \setminus \{v_n\}$), relative to s. □

In the following sections, we will deal with the case where the vertex partition is given and the problem consists of finding the vertex ordering. From now on, we will then simply call *precedence consistent ordering* (resp. *precedence strongly consistent ordering*) to one that is such for the given partition.

3.1 Precedence thinness for a given partition

In this section, we present an efficient algorithm to precedence k-thin graph recognition for a given partition. This algorithm uses PQ trees and some related properties to validate precedence consistent orderings in a greedy fashion, iteratively choosing an appropriate ordering of the parts of the given partition that satisfies precedence consistence, if one does exist. Formally, the problem addressed in this section is the following.

Problem:	PARTITIONED k-PT (Recognition of k-PT graphs for a given partition)
Input:	A natural k, a graph G and a partition $(V_1, \ldots, V_k)$ of $V(G)$.
Question:	Is there a consistent ordering s of $V(G)$ such that the vertices of V_i are consecutive in s, for all $1 \leq i \leq k$?

It should be noted that a precedence consistent ordering s of G consists of a concatenation of canonical orderings of each $G[V_i]$, for all $1 \leq i \leq k$. That is, $s = s_1 s_2 \ldots s_k$, where $s_1, s_2, \ldots, s_k$ is a permutation of $s'_1, s'_2, \ldots, s'_k$ and s'_i is a canonical ordering of $G[V_i]$, for all $1 \leq i \leq k$. The following property is straightforward from the definition of a precedence consistent ordering.

Property 1. *Let $(V_1, V_2, \ldots, V_k)$ be a partition of $V(G)$, s a precedence consistent ordering and $1 \leq i, j \leq k$. If V_i precedes V_j in s, then, for all $u, v \in V_i$ and $w \in V_j$, if $uw \notin E(G)$ and $vw \in E(G)$, then u precedes v in s.*

Property 1 shows that, for any given consistent ordering $s = s_1 s_2 \ldots s_k$, the vertices of s_j impose ordering restrictions on the vertices of s_i, for all $1 \leq i < j \leq k$. This relation is depicted in Figure 3.2. This property will be used as a key part of the greedy algorithm to be presented later on.

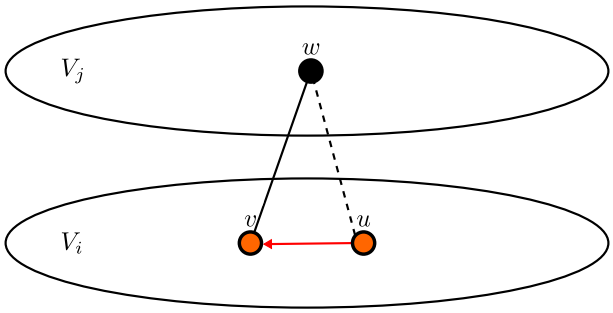


Figure 3.2: Precedence relations among the vertices in a precedence consistent ordering.

Let $s_T = C_1, C_2, \ldots, C_q$ be an ordering of the maximal cliques of an interval graph G obtained from a PQ tree T. Recall from Chapter 2 that it is possible to obtain a canonical ordering s ordered according to s_T. Let $u, v \in V(G)$. We define T as *compatible with the ordering restriction* $u < v$ if there exists a canonical ordering s ordered according to s_T such that $u < v$ in s. The following theorem describes compatibility conditions between a PQ tree and an ordering restriction $u < v$.

Theorem 8. *Let G be an interval graph and T be a PQ tree of G. Let X be a node of T with children $X_1, \ldots, X_k$ and $u, v \in X$. Denote by T_X the subtree rooted at X. The following statements are true.*

(i) if v belongs to all leaves of T_X, then T_X is compatible with $u < v$ (see Figure 3.3(a)).

(ii) Let $X_i, X_j \in \{X_1, \ldots, X_k\}$. If $u \in X_i$, $v \notin X_i$, $v \in X_j$ and T_X is compatible with $u < v$, then X_i precedes X_j in T_X (see Figure 3.3(b)).

Proof. Let G_X be the graph induced by the union of the leaves of T_X.

(i) Let s_x be a canonical ordering of G_X and s'_x the ordering obtained from s_x by moving v to the last position. As v is a universal vertex from G_X, s'_x is also a canonical ordering of this graph. Consequently, T_X is compatible with $u < v$.

(ii) Suppose X_j precedes X_i in T_X and T_X is compatible with $u < v$. As $v \notin X_i$, then by Theorem 2, there is no X_z such that X_i precedes X_z and $v \in X_z$. Otherwise, there would exist three maximal cliques C_i, C_j, C_z such that $C_j < C_i < C_z$ in the ordering of cliques represented in T_X and such that $v \notin C_i$, $v \in C_j \cap C_z$. Therefore $v < u$ in any canonical ordering s_X of G (Lemma 5), a contradiction because T_X is compatible with $u < v$. Thus, X_i precedes X_j in T_X. □

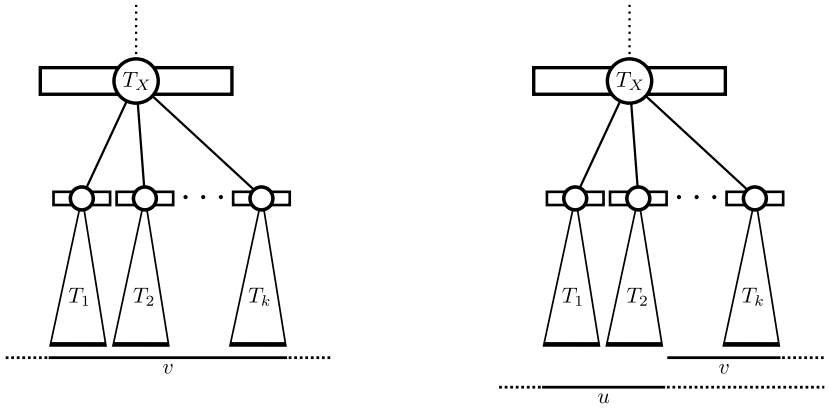


Figure 3.3: Ordering imposed by Theorem 8 item *i* (*a*) and item *ii* (*b*).

Theorem 8 can be used to determine the existence of a PQ tree compatible with an ordering restriction $u < v$. This task can be achieved by considering as the node X of Theorem 8, each one of the nodes of a given PQ tree T. If T violates the conditions imposed by the theorem, T is "annotated" in a way that the set of equivalent PQ trees is restricted, avoiding precisely the violations. This procedure continues until all nodes produce their respective restrictions, in which case any equivalent PQ tree allowed by the "annotated" tree T is compatible with the given ordering restrictions. If there is no equivalent PQ tree to the "annotated" tree T, which means there is no way to avoid the violations, then there is no tree which is compatible with such an ordering restriction.

The general idea to "annotate" a PQ tree is to use an auxiliary digraph to represent required precedence relations among the vertices. This digraph is constructed for each part and any topological ordering of it results in a precedence consistent ordering concerning the vertices of this part. Property 1 is applied to determine the ordering restrictions of the vertices and Theorem 8 to ensure that at the end of the algorithm the vertices are ordered according to a canonical clique ordering. Such a procedure is detailed next.

First, the algorithm validates if each part of the partition induces an interval graph. This step can be accomplished, for each part, in linear time [15]. If at least one of these parts does not induce an interval graph, then the answer is NO. Otherwise, the algorithm tries each part as the first of a precedence consistent ordering. For each candidate part V_i, $1 \leq i \leq k$, it builds a digraph D to represent the order conditions that the vertices of V_i must satisfy in the case in which V_i precedes all the other parts of the partition. That is, V_i must be ordered in such a way that it is according to a canonical clique ordering and respects the restrictions imposed by Property 1. In this strategy, the vertex set of D is V_i and its directed edges represent the precedence relations among its vertices. Namely, $(u, v) \in E(D)$ if, and only if, u must precede v in all precedence consistent orderings that have V_i as its first part. The algorithm uses Property 1 to find all the ordering restrictions $u < v$ among the vertices of V_i imposed by the others parts, adding the related directed edges to D. Then, by building a PQ tree T of $G[V_i]$, Theorem 8 is used to transform T into an equivalent PQ tree T' ensuring that all those ordering restrictions $u < v$ are satisfied. If there is a PQ tree T' of $G[V_i]$ that is compatible with all the ordering restrictions imposed by Property 1, then the algorithm adds directed edges to D according to the canonical clique ordering represented by T'. This step is described below in the Algorithm 1 and it is similar to the one described in Lemma 5. At this point, D is finally constructed and the existence of a topological ordering for its vertices determines whether V_i can be chosen as the first part of a precedence consistent ordering for the given partition. If that is the case, V_i is chosen as the first part and the process is repeated in $G \setminus V_i$ to choose the next part. If no part can be chosen at any step, the answer is NO. Otherwise, a feasible ordering of the parts and of the vertices within each part is obtained and the answer is YES. Next, the validation of the compatibility of T is described in more detail.

Algorithm 1: Adding edges from a PQ-tree to D

Input: G: an interval graph; D: a digraph; T: a PQ-tree;

```
procedure addEdgesFromPQTree(G, D, T)
    Let s_C be the canonical clique ordering relative to T
    for each C_i ∈ s_C do
        Let S be the set of simplicial vertices of C_i
        for each v ∈ S do
            for each u ∈ C_{i+1} do
                E(D) ← E(D) ∪ {(v, u)}
        G ← G \ S
```

For each imposed ordering $u < v$ (that is, $(u, v) \in E(D)$), T is traversed node by node, applying Theorem 8. As a consequence of such an application, if the order of the children of some node X of T must be changed to meet some restrictions, directed edges are inserted on T to represent such needed reorderings. Those directed edges appear among nodes that are children of a same node in T. At the end, to validate if T is compatible with all needed reorderings of children of nodes, a topological ordering is applied to the

children of each node. If there are no cycles among the children of each node, then there is an equivalent PQ tree T' compatible with all the restrictions. In this case, T' is obtained from T applying the sequence of permutations of children of P nodes and reversals of children of Q nodes which are compliant to the topological orderings. Otherwise, if a cycle is detected in some topological sorting, then there is no PQ tree of $G[V_i]$ which is compatible with all the set of restrictions. In other words, V_i cannot be chosen as the first part in a precedence consistent ordering for the given partition. Algorithm 2 formalizes the procedure.

To illustrate the execution of Algorithm 2, consider the graph G as defined in Figure 3.4 and the 3-partition $\mathcal{V} = (V_1, V_2, V_3)$ of $V(G)$ where $V_1 = \{a, b, c, d, e, f, g, h, i, j, k, l\}$, $V_2 = \{a', b', c', d', e', f', g, h', i', j', k', l'\}$ e $V_3 = \{a'', b'', c'', d'', e'', f'', g'', h'', i'', j'', k'', l''\}$. For the sake of clearness, in Figure 3.4, the edges with endpoints in distinct parts are depicted in black, the edges with endpoints in a same part are in light gray and the vertices belonging to distinct parts are represented with different colors. Moreover, the vertices of each part, read from the left to right, consist of a canonical ordering of the graph induced by that part. Each part of $\mathcal{V}$ induces the interval graph G' depicted in Figure 3.5(a). Figure 3.5(b) depicts a model of G'. In this model, all maximal cliques are represented by vertical lines. Figure 3.5(c) represents a PQ tree of G' in which each maximal clique is labeled according to the model in Figure 3.5(b).

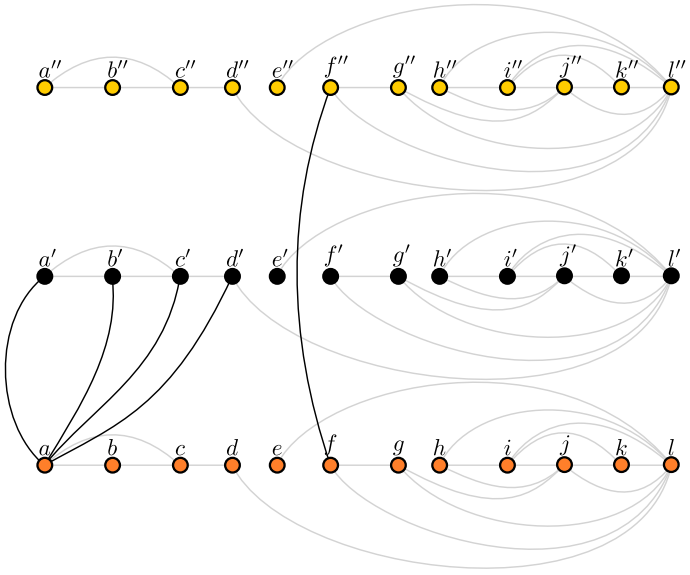


Figure 3.4: A graph G and a 3-partition $\mathcal{V} = (V_1, V_2, V_3)$ of its vertices where $V_1 = \{a, b, c, d, e, f, g, h, i, j, k, l\}$, $V_2 = \{a', b', c', d', e', f', g, h', i', j', k', l'\}$ e $V_3 = \{a'', b'', c'', d'', e'', f'', g'', h'', i'', j'', k'', l''\}$.

Suppose that, at the first step, the algorithm tries to choose V_1 as the first part of the precedence consistent ordering. As mentioned, $G[V_1] \cong G'$ and, according to the model in Figure 3.5(b), $G[V_1]$ has maximal cliques $C_1 = \{a, b, c\}$, $C_2 = \{c, d\}$, $C_3 = \{d, l\}$, $C_4 = \{e, l\}$, $C_5 = \{f, g, l\}$, $C_6 = \{g, j, l\}$, $C_7 = \{h, i, j, l\}$ and $C_8 = \{i, j, k, l\}$. Concerning the edges between V_1 and V_2 and according to Property 1, the vertex a of V_1 must suc-

Algorithm 2: PARTITIONED k-PT

Input: G: a graph; k: a natural number; $\mathcal{V}$: a k-partition $(V_1, V_2, \ldots, V_k)$ of $V(G)$;

function partitioned-k-PT(G, k, $\mathcal{V}$)

- $s \leftarrow \emptyset$
- **for each** $V_i \in \mathcal{V}$ **do**
  - **if** $G[V_i]$ is not an interval graph **then**
    - **return** (NO, $\emptyset$)
- **while** $\mathcal{V} \neq \emptyset$ **do**
  - **for each** $V_i \in \mathcal{V}$ **do**
    - $foundFirstPart \leftarrow$ **TRUE**
    - Create a digraph $D = (V_i, \emptyset)$
    - Build a PQ tree T_i of $G[V_i]$
    - **for each** $V_j \in \mathcal{V}$ such that $V_j \neq V_i$ **do**
      - Let S be the set of precedence relations among the vertices of V_i concerning V_j (Property 1)
      - **for each** $(u < v) \in S$ **do**
        - $E(D) \leftarrow E(D) \cup \{(u, v)\}$
        - **for each** node X of T_i **do**
          - Add the direct edges, deriving from $(u < v)$, among the children of X (Theorem 8)
      - **for each** node X of T_i **do**
        - Let $D_X = (V', E')$ be the digraph where V' is the set of the children of X and E' are the directed edges added among them
        - **if** there is a topological ordering s_X of D_X **then**
          - Arrange the children of X according to s_X
        - **else**
          - $foundFirstPart \leftarrow$ **FALSE**
    - **if** *foundFirstPart* **then**
      - `addEdgesFromPQTree`($G[V_i]$, D, T_i)
      - **if** there is a topological ordering s_i of D **then**
        - $s \leftarrow ss_i$
        - $\mathcal{V} \leftarrow \mathcal{V} \setminus V_i$
        - **break**
      - **else**
        - $foundFirstPart \leftarrow$ **FALSE**
  - **if not** $foundFirstPart$ **then**
    - **return** (NO, $\emptyset$)
- **return** (YES, s)

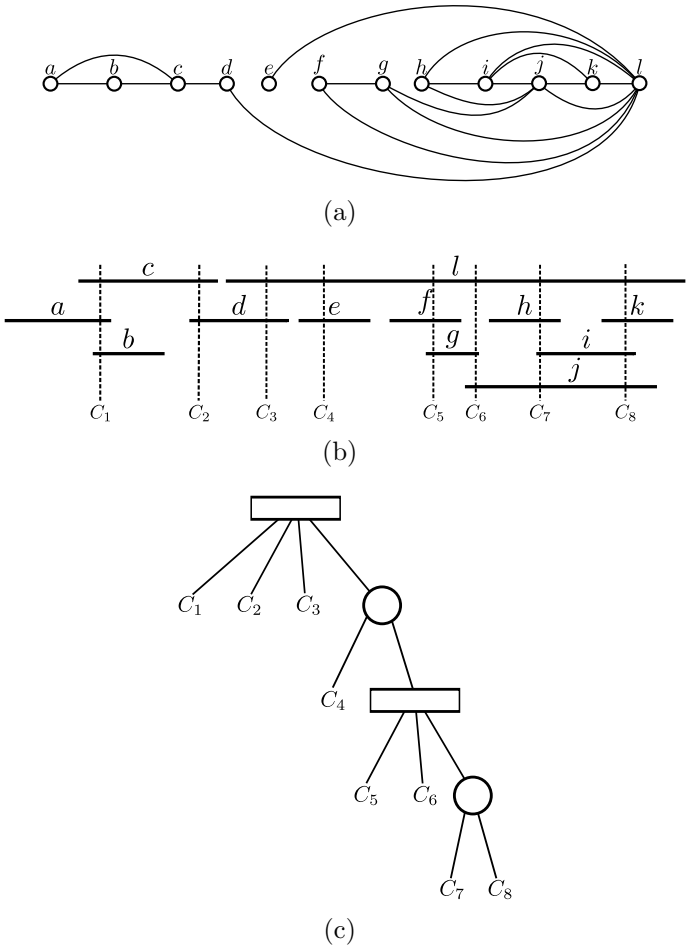


Figure 3.5: An interval graph G' (a); an interval model (b) and a PQ tree (c) of G'.

ceed all the other vertices of this part in any valid canonical ordering. This requirement is translated into the corresponding PQ tree through directed edges as depicted in Figure 3.6(a). Let X be the current node of T. Note that if X is a Q node, then any imposed ordering of a pair of its children implies an ordering for all of them. In Figure 3.6(a), the oriented edges deriving from Property 1 are represented in blue and the edges deriving from the orientation demanded by Q nodes, due to the presence of the blue ones, are represented in orange. Clearly, there is a valid PQ tree that satisfies such orientations. Now the algorithm adds the directed edges to the tree deriving from fact that V_1 must also precede V_3. Considering the edges between V_1 and V_3, and according to Property 1, the vertex f of V_1 must succeed all the other vertices of V_1 in any valid canonical ordering. This requirement is translated into the PQ tree in Figure 3.6(a), resulting in the PQ tree in Figure 3.6(b). Clearly, no PQ tree can satisfy the given orientation due to the directed cycle in the tree. Then, V_1 cannot precede both V_2 and V_3.

As V_1 cannot be chosen as the first part, the algorithm tries another part as the first of the precedence consistent ordering. Suppose it now chooses V_2 as the first part. The interval graph $G[V_2]$ has as maximal cliques $C_1 = \{a', b', c'\}$, $C_2 = \{c', d'\}$, $C_3 = \{d', l'\}$, $C_4 = \{e', l'\}$, $C_5 = \{f', g', l'\}$, $C_6 = \{g', j', l'\}$, $C_7 = \{h', i', j', l'\}$ and $C_8 = \{i', j', k', l'\}$. Note that, as there are no edges between V_2 and V_3, V_2 can precede V_3 in any valid

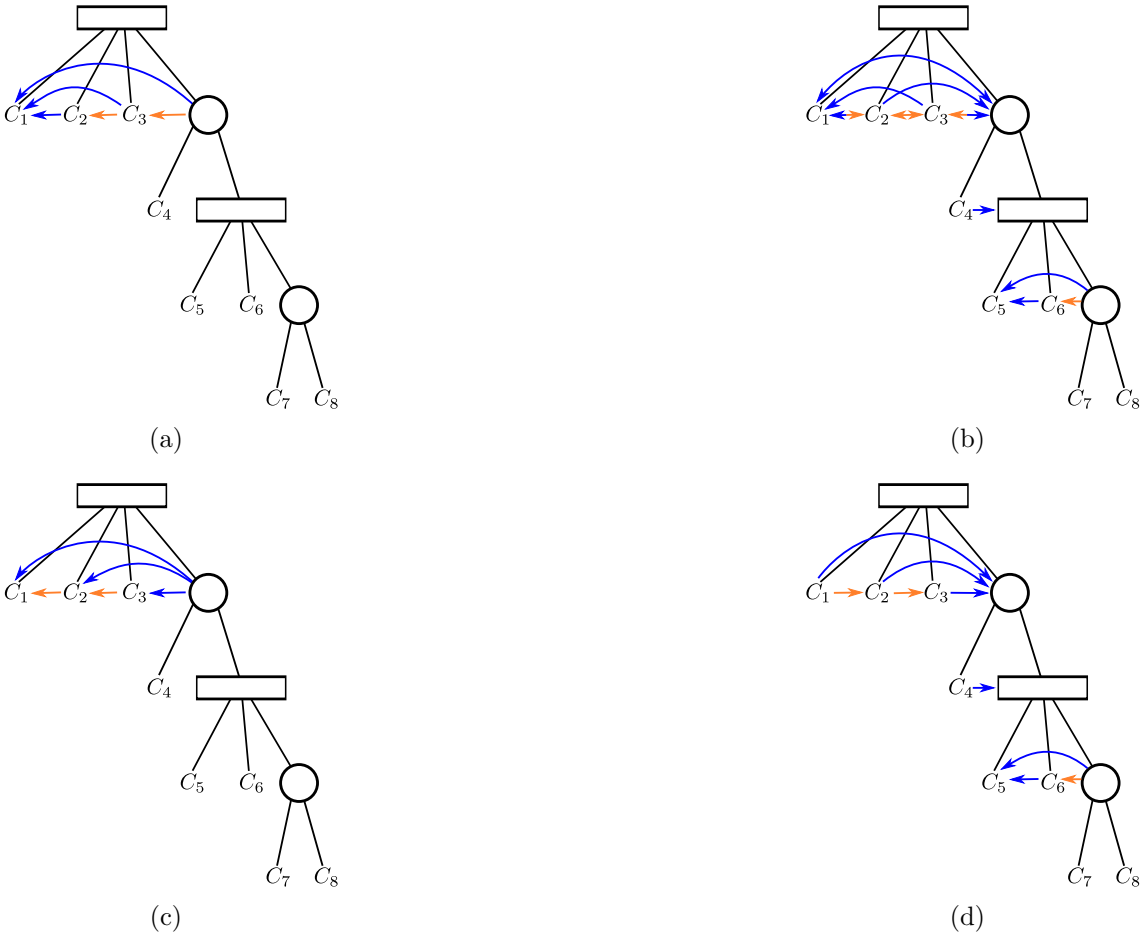


Figure 3.6: The edges added to the PQ tree of Figure 3.5(c) through the example of execution of the Algorithm 2.

consistent ordering. The algorithm must decide whether V_2 can precede V_1. According to the Figure 3.4 and Property 1, the vertices $\{a', b', c', d'\}$ of V_2 must succeed all the other vertices of the same part in any valid consistent ordering. This requirement is again translated into directed edges in the PQ tree of V_2 as depicted in Figure 3.6(c). Clearly, there is a PQ tree T' satisfying those directed edges. Then, the algorithm adds the edges deriving from the canonical clique ordering represented by T' to D. Figure 3.7(a) depicts the final state of D once the necessary edges have been added. In this figure, the edges deriving from Property 1 are presented with an orange color and the edges related to T' are presented in a blue color. For readability, in these figures the edges that can be obtained by transitivity are omitted. As there are no cycles in D, V_2 can precede V_1 and V_3. A topological ordering of D leads to the canonical ordering $s = e', f', g', h', i', j', k', l', d', a', b', c'$ of $G[V_2]$.

After deciding V_2 as the first part, the algorithm uses the same process to choose the second part. Suppose it tries V_1 as the second part. Figure 3.6(d) depicts the edges added to the PQ tree of $G[V_1]$. As there are no cycles in the edges added, there is a tree which is compatible with the precedence relations associated with $V_1 < V_3$. Figure 3.7(b) depicts the final state of the digraph D related to V_1. As there are no cycles in D, V_1 can precede

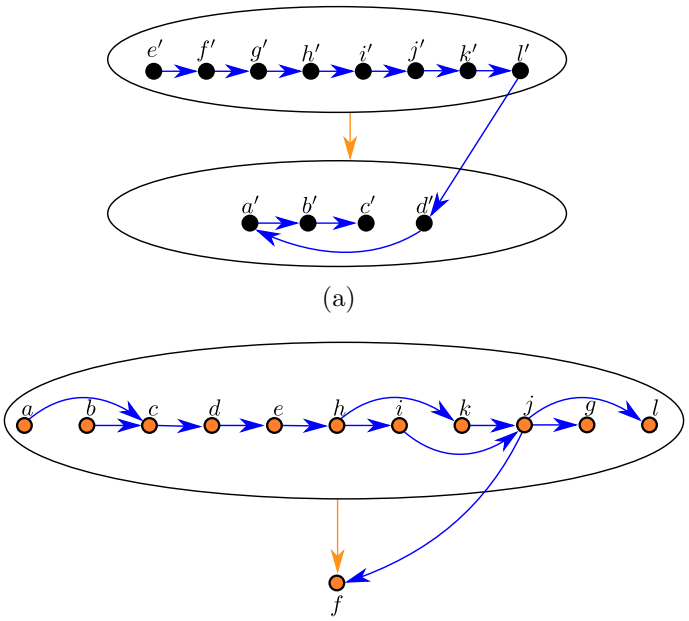


Figure 3.7: Related digraph D when V_2 is chosen as the first part (a) and when V_1 is chosen as the second part (b) in Algorithm 2.

V_3, so the algorithm chooses it as the second part. Finally, the algorithm chooses V_3 as the last part and determines that there is a precedence consistent ordering such that $V_2 < V_1 < V_3$.

Concerning the complexity of the given strategy, each time one of the k parts is tried to be the first, we build a new digraph D, a new PQ tree T and obtain the precedence relations according to Property 1. Enumerating all the precedence relations requires at most $O(n^3)$ steps, which is the time that it takes to iterate over all triples of vertices of the graph. Moreover, each one of these relations must be mapped to D, which takes time $O(1)$, and T. First, note that the number of nodes of T is asymptotically bounded by its number of leaves, that is, by the number of maximal cliques of the part being processed. As the number of maximal cliques is bounded by the number of vertices of the given part, the number of nodes of T is $O(n)$. Consequently, it is possible to model a precedence relation of type $u < v$ into T, using Theorem 8, in time $O(n^2)$. To achieve this, first T is traversed, in order to decide which nodes contain (resp. not contain) u and v. A traversal of T can be done in time $O(n)$, and T can be constructed in $O(n + m)$ time. Additional steps will be necessary and generate new traversals in T following the tree levels, with the purpose to add the necessary directed edges among the vertices that are children of the same node. This step can be done in

$$\sum_{v \in V(T)} d^2(v) \leq \sum_{v \in V(T)} d(v)|V(T)| = O(|E(T)||V(T)|) = O(|V(T)|^2) = O(n^2)$$

as $|E(T)| = O(|V(T)|)$ and $|V(T)| = O(n)$. Aiming to verify the existence of a compatible

tree, the algorithm applies a topological ordering to the children of each node of T, which takes overall time $O(n(n+m))$. Then, the ordering in T is translated to D through directed edges. By using the Algorithm 1, this step requires no more than $O(n^2+m)$ operations. Finally, a topological ordering is applied to D. Thus, the algorithm has

$$O(k^2(n+m+n^3n^2+n^2+nm+n^2+m+n+m)) = O(k^2n^5)$$

time complexity.

3.2 Precedence Proper Thinness for a Given Partition

In this section, we discuss precedence proper thinness for a given partition. First, we prove that this problem is NP-complete for an arbitrary number of parts. Then, we propose a polynomial time algorithm for a fixed number of parts based on the one presented in Section 3.1. Formally, we will prove that the following problem is NP-complete.

Problem:	PARTITIONED k-PPT (Recognition of k-PPT graphs for a given partition)
Input:	A natural k, a graph G and a partition $(V_1, \ldots, V_k)$ of $V(G)$.
Question:	Is there a strongly consistent ordering s of $V(G)$ such that the vertices of V_i are consecutive in s, for all $1 \leq i \leq k$?

The NP-hardness of the previous problem is accomplished by a reduction from the problem NOT ALL EQUAL 3-SAT, which is NP-complete [24]. The details are described in Theorem 9.

Problem:	NOT ALL EQUAL 3-SAT
Input:	A formula φ on variables $x_1, \ldots, x_r$ in conjunctive normal form, with clauses $\mathcal{C}_1, \ldots, \mathcal{C}_s$, where each clause has exactly three literals.
Question:	Is there a truth assignment for $x_1, \ldots, x_r$ such that each clause $\mathcal{C}_i$, $i \in \{1, \ldots, s\}$, has at least one true literal and at least one false literal?

Theorem 9. *Recognition of k-PPT graphs for a given partition is NP-complete, even if the size of each part is at most 2.*

Proof. A given precedence strongly consistent ordering for the partition of $V(G)$ can be easily verified in polynomial time. Therefore, this problem is in NP.

Given an instance φ of NOT ALL EQUAL 3-SAT, we define a graph G and a partition of $V(G)$ in which each part has size at most two. The graph G is defined in such a way that φ is satisfiable if, and only if, there is a precedence strongly consistent ordering of $V(G)$ for the partition. The graph G is constructed as follows.

For each variable x_i appearing in the clause $\mathcal{C}_j$, create the part

$$X_{ij} = \{x_{ij}^T, x_{ij}^F\}$$

For each variable x_i, create the parts

$$X_i^T = \{x_i^T\} \text{ and } X_i^F = \{x_i^F\}$$

The edges of the graph between these parts are (x_i^T, x_{ij}^T) and (x_i^F, x_{ij}^F) for every i, j such that variable x_i appears in clause $\mathcal{C}_j$.

Notice that in any strongly consistent ordering, part X_{ij} must be between parts X_i^F and X_i^T. Moreover, if $x_i^F < x_i^T$, then $x_{ij}^F < x_{ij}^T$, and conversely. In particular, in any valid vertex order, for each $i \in \{1, \ldots, r\}$, either $x_{ij}^F < x_{ij}^T$ for every $j \in \{1, \ldots, s\}$ or $x_{ij}^T < x_{ij}^F$ for every $j \in \{1, \ldots, s\}$.

The Partitioned k-PPT instance will be such that if there is a precedence strongly consistent ordering for the vertices with respect to the given parts, then the assignment $x_i \leftarrow (x_i^F < x_i^T)$ (that is, x_i is true if x_i^F precedes x_i^T in such an ordering and x_i is false otherwise) satisfies φ in the context of Not all equal 3-SAT and, conversely, if there is a truth assignment satisfying φ in that context, then there exists a strongly consistent ordering for the Partitioned k-PPT instance in which $x_i^F < x_i^T$ if x_i is true and $x_i^T < x_i^F$ otherwise.

In what follows, if the k-th literal ℓ_{ij} of $\mathcal{C}_j$ is the variable x_i (resp. $\neg x_i$), we denote by O_{ij} the ordered part $\{x_{ij}^F, x_{ij}^T\}$ (resp. $\{x_{ij}^T, x_{ij}^F\}$).

Given a 2-vertex ordered part C, we denote by C^1 and C^2 the first and second elements of C. By $\pm C$, we denote "either C or $\bar{C}$".

For each clause $\mathcal{C}_j = \ell_{1j} \vee \ell_{2j} \vee \ell_{3j}$, we add the 2-vertex ordered parts Y_{1j}, Y_{2j}, and Y_{3j}, and the edges (O_{1j}^2, Y_{1j}^1), (O_{1j}^1, Y_{2j}^1), (O_{1j}^2, Y_{2j}^1), (O_{2j}^1, Y_{1j}^2), (O_{2j}^1, Y_{1j}^1), (O_{2j}^2, Y_{2j}^1), (O_{2j}^2, Y_{2j}^2), (O_{2j}^1, Y_{3j}^1), (O_{2j}^1, Y_{3j}^2), (O_{3j}^1, Y_{1j}^2), (O_{3j}^2, Y_{1j}^2), (O_{3j}^1, Y_{2j}^2), (O_{3j}^2, Y_{2j}^2), (O_{3j}^2, Y_{3j}^1), (O_{3j}^2, Y_{3j}^2). These edges ensure the following properties in every strongly consistent ordering of the graph with respect to the defined partition.

1. Since (O_{1j}^2, Y_{1j}^1) is the only edge between O_{1j} and Y_{1j}, their only possible relative positions are $O_{1j} < Y_{1j}$ and its reverse $\bar{Y}_{1j} < \bar{O}_{1j}$.

2. Since (O_{1j}^1, Y_{2j}^1) and (O_{1j}^2, Y_{2j}^1) are the edges between O_{1j} and Y_{2j}, their possible relative positions are $\pm O_{1j} < Y_{2j}$ and $\bar{Y}_{2j} < \pm O_{1j}$.

3. Since (O_{2j}^1, Y_{1j}^1) and (O_{2j}^1, Y_{1j}^2) are the edges between O_{2j} and Y_{1j}, their possible relative positions are $\bar{O}_{2j} < \pm Y_{1j}$ and $\pm Y_{1j} < O_{2j}$.

4. Since (O_{2j}^2, Y_{2j}^1) and (O_{2j}^2, Y_{2j}^2) are the edges between O_{2j} and Y_{2j}, their possible relative positions are $O_{2j} < \pm Y_{2j}$ and $\pm Y_{2j} < \bar{O}_{2j}$.

5. Since (O_{2j}^1, Y_{3j}^1) and (O_{2j}^1, Y_{3j}^2) are the edges between O_{2j} and Y_{3j}, their possible relative positions are $\bar{O}_{2j} < \pm Y_{3j}$ and $\pm Y_{3j} < O_{2j}$.

6. Since (O^1_{3j}, Y^2_{1j}) and (O^2_{3j}, Y^2_{1j}) are the edges between O_{3j} and Y_{1j}, their possible relative positions are $\pm O_{3j} < \bar{Y}_{1j}$ and $Y_{1j} < \pm O_{3j}$.

7. Since (O^1_{3j}, Y^2_{2j}) and (O^2_{3j}, Y^2_{2j}) are the edges between O_{3j} and Y_{2j}, their possible relative positions are $\pm O_{3j} < \bar{Y}_{2j}$ and $Y_{2j} < \pm O_{3j}$.

8. Since (O^2_{3j}, Y^1_{3j}) and (O^2_{3j}, Y^2_{3j}) are the edges between O_{3j} and Y_{3j}, their possible relative positions are $O_{3j} < \pm Y_{3j}$ and $\pm Y_{3j} < \bar{O}_{3j}$.

Notice that, by items 1 and 6 (resp. 2 and 7), the vertices of Y_{1j} (resp. Y_{2j}) are forced to lie between those of O_{1j} and those of O_{3j}. More precisely, the possible valid orders are $O_{1j} < Y_{1j}, Y_{2j} < \pm O_{3j}$ and their reverses $\pm O_{3j} < \bar{Y}_{2j}, \bar{Y}_{1j} < \bar{O}_{1j}$.

By items 3 and 4, the vertices of O_{2j} are forced to be between those of Y_{1j} and those of Y_{2j}. More precisely, the possible valid orders are $\pm Y_{1j} < O_{2j} < \pm Y_{2j}$ and their reverses $\pm Y_{2j} < \bar{O}_{2j} < \pm Y_{1j}$.

By items 3 and 5, the vertices of Y_{1j} and Y_{3j} are forced to be on the same side with respect to the vertices of O_{2j}, either $\bar{O}_{2j} < \pm Y_{1j}, \pm Y_{3j}$ or $\pm Y_{1j}, \pm Y_{3j} < O_{2j}$.

Hence, taking also into account item 8, the possible valid orders are

- $O_{1j} < Y_{1j}, \pm Y_{3j} < O_{2j} < Y_{2j} < \bar{O}_{3j}$
- $O_{1j} < Y_{2j} < \bar{O}_{2j} < Y_{1j}, \pm Y_{3j} < \bar{O}_{3j}$
- $O_{1j} < Y_{2j} < \bar{O}_{2j} < Y_{1j} < O_{3j} < \pm Y_{3j}$

and their reverses,

- $O_{3j} < \bar{Y}_{2j} < \bar{O}_{2j} < \bar{Y}_{1j}, \pm Y_{3j} < \bar{O}_{1j}$
- $O_{3j} < \bar{Y}_{1j}, \pm Y_{3j} < O_{2j} < \bar{Y}_{2j} < \bar{O}_{1j}$
- $\pm Y_{3j} < \bar{O}_{3j} < \bar{Y}_{1j} < O_{2j} < \bar{Y}_{2j} < \bar{O}_{1j}$

and will correspond to truth assignments that make true, respectively,

a) $\ell_{1j} \wedge \ell_{j2} \wedge \neg \ell_{3j}$

b) $\ell_{1j} \wedge \neg \ell_{2j} \wedge \neg \ell_{3j}$

c) $\ell_{1j} \wedge \neg \ell_{2j} \wedge \ell_{3j}$

d) $\neg \ell_{1j} \wedge \neg \ell_{2j} \wedge \ell_{3j}$

e) $\neg \ell_{1j} \wedge \ell_{2j} \wedge \ell_{3j}$

f) $\neg \ell_{1j} \wedge \ell_{2j} \wedge \neg \ell_{3j}$

Suppose first that there is a precedence strongly consistent ordering of $V(G)$ with respect to its vertex partition. Define a truth assignment for variables $x_1, \ldots, x_r$ as $x_i \leftarrow (x_i^F < x_i^T)$, for $i \in \{1, \ldots, r\}$.

As observed above, if the value of x_i is true (resp. false), then for every $j \in \{1, \ldots, s\}$, the part X_{ij} is ordered $x_{ij}^F\ x_{ij}^T$ (resp. $x_{ij}^T\ x_{ij}^F$). So, for each clause $\mathcal{C}_j$, the part corresponding to its k-th literal will be ordered as O_{kj} if the literal is assigned true and as $\bar{O}_{kj}$ if the literal is assigned false. Since for each valid order of the vertices there exist $k, k' \in \{1, 2, 3\}$ such that the part corresponding to the k-th literal is ordered O_{kj} and the part corresponding to the k'-th literal is ordered $\bar{O}_{k'j}$, the truth assignment satisfies the instance φ of Not all equal 3-SAT.

Suppose now that there is a truth assignment for variables $x_1, \ldots, x_r$ that satisfies the instance φ of Not all equal 3-SAT. Define the order of the vertices in the following way. The first r vertices are $\{x_i^F : x_i \text{ is true}\} \cup \{x_i^T : x_i \text{ is false}\}$, and the last r vertices are $\{x_i^T : x_i \text{ is true}\} \cup \{x_i^F : x_i \text{ is false}\}$. Between these first and last r vertices, place all the parts X_{ij}, Y_{1j},Y_{2j} and Y_{3j} associated with each clause $\mathcal{C}_j$, $j = 1, \ldots, s$. In particular, the parts X_{ij}, Y_{1j}, Y_{2j} and Y_{3j} are ordered accordingly to which of the conditions (a)–(f) is satisfied. By the analysis above, this is a precedence strongly consistent ordering of the vertices of G, with respect to the defined parts. □

As an example of the NP-completeness reduction presented in Theorem 9, Figure 3.8 depicts the instance of the Partitioned k-PPT problem built from the instance $\varphi = \{(x_1 \vee x_2 \vee \neg x_3)\}$ of Not all equal 3-SAT problem. In this example, the sequence $s = x_1^F, x_2^F, x_3^F, x_{11}^F, x_{11}^T, Y_{11}^1, Y_{11}^2, Y_{31}^1, Y_{31}^2, x_{21}^F, x_{21}^T, Y_{21}^1, Y_{21}^2, x_{31}^F, x_{31}^T, x_1^T, x_2^T, x_3^T$, related to the true assignment $x_1 = x_2 = x_3 = T$, is a solution to (G, φ).

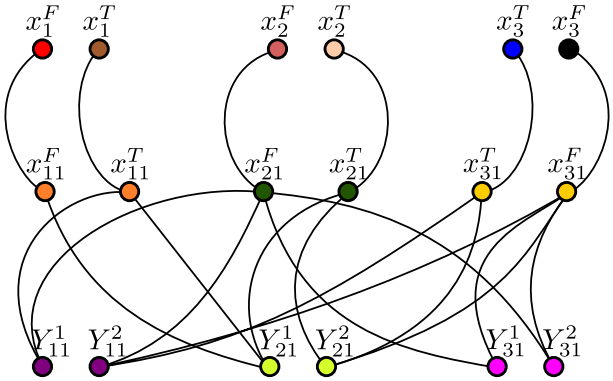


Figure 3.8: Instance of the Partitioned k-PPT problem built from the instance $\varphi = \{(x_1 \vee x_2 \vee \neg x_3)\}$ of Not all equal 3-SAT problem.

The remaining of this section is dedicated to discuss a polynomial time solution to a variation of the Partitioned k-PPT problem. This variation consists in considering a fixed number of parts for $V(G)$, that is, k is removed from the input and taken as a constant for the problem. The strategy that will be adopted is the same used for Partitioned k-PT problem. It is not difficult to see that Property 1 is not sufficient to describe the requirements that must be imposed in the ordering of vertices in a precedence strongly

consistent ordering. This is so because, unlike what occurs in a precedence consistent ordering, in a precedence strongly consistent ordering the vertices of each part V_i may impose an ordering to vertices that belong to parts that precede and succeed V_i. Given this fact, we observe the following property to describe such relation.

Property 2. *Let $(V_1, V_2, \ldots, V_k)$ be a partition of $V(G)$, s a precedence strongly consistent ordering and $1 \leq i, j \leq k$. If V_i precedes V_j in s, then for all $u, v \in V_i$ and $w \in V_j$, if $uw \notin E(G)$ and $vw \in E(G)$, then u precedes v in s. Moreover, for all $u \in V_i$ and $w, x \in V_j$, if $uw \notin E(G)$ and $ux \in E(G)$, then x precedes w in s.*

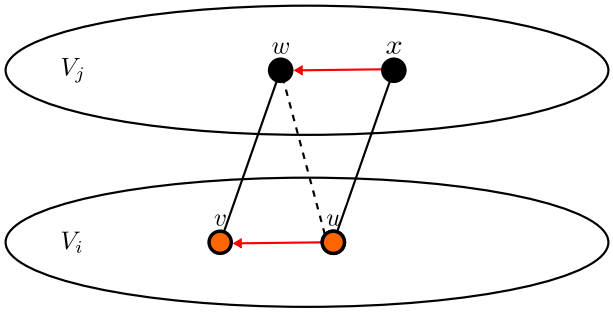


Figure 3.9: Precedence relations among the vertices in a precedence strongly consistent ordering.

Notice that the greedy strategy used in Section 3.1 does not work in the problem being considered. This is so because, according to Property 2 and visually depicted in Figure 3.9, the ordering of vertices of V_i in a precedence strongly consistent ordering s is influenced by both the parts that precede and succeed V_i in s. Despite this, the method described in the Section 3.1 to validate whether a part can precede a set of parts is also useful to present a solution to this problem. In the given solution, the addition of edges from a PQ-tree to the digraph D (Algorithm 1) has to be specialized for the case of proper interval graphs, which is provided in Algorithm 3. In this algorithm, when the simplicial vertices are determined for each maximal clique, the ones that have appeared first in the clique ordering will be chosen first in the final ordering. The time complexity of Algorithm 3 remains the same as that of Algorithm 1. The strategy to solve the Partitioned k-PPT problem, for a fixed k, is described next.

Let G be a graph, $\mathcal{V} = (V_1, V_2, \ldots, V_k)$ a partition of $V(G)$ and s a precedence strongly consistent ordering of $V(G)$ for the given partition. Clearly, for all $1 \leq i \leq k$, $G[V_i]$ must be a proper interval graph for s to be a precedence strongly consistent ordering. Verifying whether $G[V_i]$ is a proper interval graph can be accomplished in linear time. If one of the parts does not induce a proper interval graph, then the answer is NO. Otherwise, each part has a PQ tree associated to it.

For a given sequence $s_{\mathcal{V}}$ of parts of $\mathcal{V}$, suppose that $V_j < V_i < V_z$ in $s_{\mathcal{V}}$, for $1 \leq j, i, z \leq k$. Let T_i be a PQ tree of $G[V_i]$. Notice that, considering the Property 2, if we apply Theorem 8 to get the ordering constraints imposed by V_j and V_z to T_i, and add the directed edges to T_i in the same way that has been done in Section 3.1 and T_i can meet

Algorithm 3: Adding edges from a PQ-tree to D

Input: G: a proper interval graph; D: a digraph; T: a PQ-tree;

```
procedure addEdgesFromPQTree(G, D, T)
    Let s_C be the canonical clique ordering relative to T
    for each C_i ∈ s_C do
        Let S be the set of simplicial vertices of C_i
        for each v_j ∈ S do
            for each v_z ∈ S such that s_C(v_j) < s_C(v_z) do
                E(D) ← E(D) ∪ {(v_j, v_z)}
            for each v_z ∈ C_{i+1} do
                E(D) ← E(D) ∪ {(v_j, v_z)}
        G ← G \ S
```

the constraints, then T_i is compatible to being at that position. That is, the vertices of V_i can precede the vertices of V_z and succeed the vertices of V_j in any precedence strongly consistent ordering. We show that for any $s_{\mathcal{V}}$, it is possible to verify whether there is a strongly consistent ordering s in which the ordering of the parts in s is precisely $s_{\mathcal{V}}$.

To solve the problem, we will test all $k!$ possible permutations $s_{\mathcal{V}} = V'_1, V'_2, \ldots, V'_k$ among the parts of $\mathcal{V}$ and validate, using a digraph and PQ trees, if each part V'_i can precede V'_z and succeed V'_j, for all $1 \leq j < i < z \leq k$. This validation is done exactly as described in Section 3.1, except for using Property 2 instead of Property 1. If there is some s that satisfies this condition, then there is a precedence strongly consistent ordering with respect to s and G is a k-PPT graph concerning $\mathcal{V}$. Otherwise, G is not a k-PPT graph with respect to $\mathcal{V}$. Algorithm 4 formalizes the procedure.

Concerning the time complexity of the algorithm, first note that the creation of the directed edges derived from Property 2 related to V'_j (resp V'_z) in T_i can be done in $O(n^5)$ time. Also, for each V_i we apply this property considering all the other parts, that is, $O(k)$ times, and therefore $O(k^2)$ times overall considering each V_i. As this operation must be executed for all $k!$ possible permutations, and considering the analysis of this same method in Section 3.1, the given strategy yields a worst case time complexity of $O(k!k^2n^5) = O(n^5)$ as k is fixed.

We end this section by mentioning an even more restricted case of the problem. Namely, the recognition of k-PPT graphs for a fixed number of parts such that each part induces a connected graph. Note that, as each part induces a connected graph, the proper interval graph induced by each part has a unique proper canonical ordering up to reversion or mutual true twins permutation. This fact implies that the PQ tree related to each one of these proper interval graphs is formed by one node of type Q, which is the root, that has all the maximal cliques as its children. That is, there are only two possible configurations for each one of these PQ trees. As the number of possible configurations is a constant, this property leads to a more efficient algorithm. Instead of using Theorem 8 to map restrictions to the PQ tree in order to obtain a compatible tree, the algorithm can

Algorithm 4: PARTITIONED k-PPT

Input: G: a graph; $\mathcal{V}$: a k-partition $(V_1, V_2, \ldots, V_k)$ of $V(G)$ for some fixed k

function Partitioned-k-PPT(G, $\mathcal{V}$)
- **for each** $V_i \in \mathcal{V}$ **do**
  - **if** $G[V_i]$ is not a proper interval graph **then**
    - **return** (NO, $\emptyset$)
- **for each** permutation $s_{\mathcal{V}}$ of $\mathcal{V}$ **do**
  - $s \leftarrow \emptyset$
  - $foundValidPermutation \leftarrow$ **TRUE**
  - **for each** $V_i \in s_{\mathcal{V}}$ **do**
    - Create a digraph $D = (V_i, \emptyset)$
    - Build a PQ tree T_i of $G[V_i]$
    - **for each** $V_j \in s_{\mathcal{V}}$ such that $V_j \neq V_i$ **do**
      - Let S be the set of precedence relations among the vertices of V_i concerning V_j (Property 2)
      - **for each** $(u < v) \in S$ **do**
        - $E(D) \leftarrow E(D) \cup \{(u, v)\}$
        - **for each** node X of T_i **do**
          - Add the direct edges, deriving from $(u < v)$, among the children of X (Theorem 8)
      - **for each** node X of T_i **do**
        - Let $D = (V', E')$ be the digraph where V' is the set of the children of X and E' are the directed edges added among them
        - **if** there is a topological ordering s of D **then**
          - Arrange the children of X according to s
        - **else**
          - $foundValidPermutation \leftarrow$ **FALSE**
    - **if** $foundValidPermutation$ **then**
      - `addEdgesFromPQTree`($G[V_i]$, D, T_i)
      - **if** there is a topological ordering s_i of D **then**
        - $s \leftarrow ss_i$
      - **else**
        - $foundValidPermutation \leftarrow$ **FALSE**
        - **break**
  - **if** $foundValidPermutation$ **then**
    - **return** (YES, s)
- **return** (NO, $\emptyset$)

check both configurations of the PQ tree independently. As the step which uses Theorem 8 is no longer required, this approach leads to an algorithm that yields a worst case time complexity of $O(k!k^2 2^k n^3) = O(n^3)$. This strategy is presented in Algorithm 5.

Algorithm 5: PARTITIONED k-PPT

Input: G: a graph; $\mathcal{V}$: a k-partition $(V_1, V_2, \ldots, V_k)$ of $V(G)$, for some fixed k, such that $G[V_i]$ is connected for all $1 \leq i \leq k$

```
function Partitioned-k-PPT(G, 𝒱)
    for each V_i ∈ 𝒱 do
        if G[V_i] is not a proper interval graph then
            return (NO, ∅)
    for each permutation s_𝒱 of 𝒱 do
        s ← ∅
        foundValidPermutation ← TRUE
        for each V_i ∈ s_𝒱 do
            foundValidTree ← FALSE
            Build a PQ tree T_i of G[V_i]
            Let T_i' be the PQ tree obtained from T_i by reversing the order of
              the children of the root
            for each T ∈ {T_i, T_i'} do
                Create a digraph D = (V_i, ∅)
                for each V_j ∈ s_𝒱 such that V_j ≠ V_i do
                    Let S be the set of precedence relations among the
                      vertices of V_i concerning V_j (Property 2)
                    for each (u < v) ∈ S do
                        E(D) ← E(D) ∪ {(u, v)}
                addEdgesFromPQTree(G[V_i], D, T)
                if there is a topological ordering s_i of D then
                    s ← ss_i
                    foundValidTree ← TRUE
                    break
            if not foundValidTree then
                foundValidPermutation ← FALSE
                break
        if foundValidPermutation then
            return (YES, s)
    return (NO, ∅)
```

3.3 Characterization of k-PT and k-PPT Graphs

This section describes a characterization of k-PT and k-PPT graphs for a given partition. First, we define some further concepts.

A graph G is a *split* graph if there is a 2-partition (V_1, V_2) of $V(G)$ such that V_1 is a clique and V_2 a stable set of G. A graph G is called a *threshold* graph if G is a split graph and there is an ordering of V_1 (resp. V_2), named *threshold ordering*, such that the neighborhood of vertices of V_1 (resp. V_2) are ordered by inclusion, in the following way: if $u, v \in V_1$ (resp. $u, v \in V_2$), u preceding v in the threshold ordering, then $N[u] \subseteq N[v]$ (resp. $N(u) \subseteq N(v)$).

For the following characterization, we will define the split graph $S_G(V_1, V_2)$ with respect to a 2-partition (V_1, V_2) of $V(G)$. Such a graph is obtained from G by the completion of edges among the vertices of V_1 and the removal of all edges among the vertices of V_2, hence transforming V_1 into a clique and V_2 into a stable set. Figures 3.10(b), 3.10(d) and 3.10(f) illustrate the corresponding split graphs of the graphs in the Figures 3.10(a), 3.10(c) and 3.10(e), respectively.

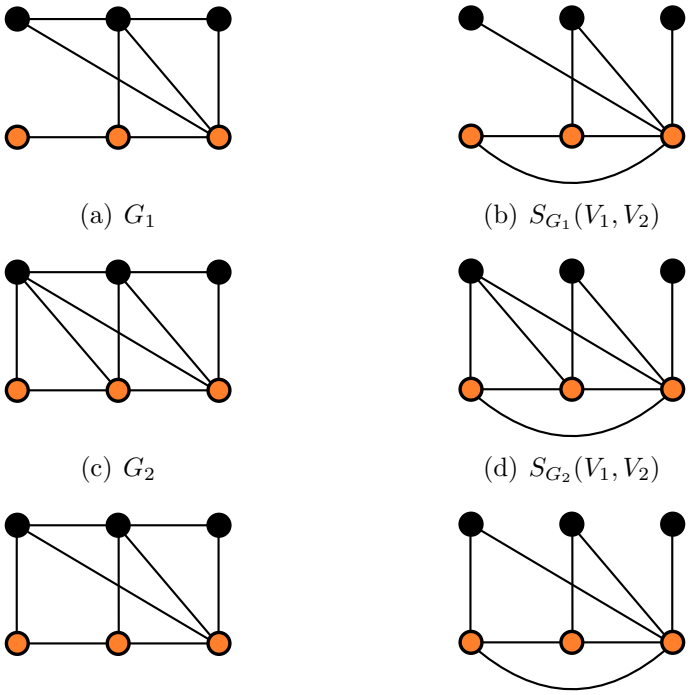
Let $s = s_1 s_2$ be an ordering of $V(G)$. We define (s_1, s_2) as *in accordance* with G if s_1 is a threshold ordering of $S_G(V(s_1), V(s_2))$, and if s_1 and s_2 are canonical orderings of $G[V(s_1)]$ and $G[V(s_2)]$, respectively. Additionally, we define (s_1, s_2) as *strongly in accordance* with G if s_1 and s_2 are proper canonical orderings of $G[V(s_1)]$ and $G[V(s_2)]$, respectively, and both s_1 and $\bar{s_2}$ are threshold orderings of $S_G(V(s_1), V(s_2))$.

As an example, let s_1 and s_2 be the orderings represented in the Figure 3.10 by reading the vertices of each part, of each graph, from left to right. The related pair (s_1, s_2) of Figure 3.10(a) is in accordance, but is not strongly in accordance, with the given graph. In the order hand, Figure 3.10(c) depicts a pair (s_1,s_2) which is strongly in accordance with the associated graph. Finally, Figure 3.10(e) exemplifies a case where the given orderings are neither in accordance nor strongly in accordance with its correlated graph. In fact, since both V_1 and V_2 in Figure 3.10(e) induce subgraphs that admit only four canonical orderings each, it can be easily verified that there is no (s_1, s_2) which is in accordance, or strongly in accordance, with the graph considering the same partition (V_1, V_2).

Lemma 10. *Let G be a graph. Then, G is 2-PT if, and only if, there is a consistent ordering $s = s_1 s_2$ for which $\mathcal{V} = (V(s_1), V(s_2))$ is a 2-partition of $V(G)$ such that (s_1, s_2) is in accordance with G.*

Proof. Let $\mathcal{V} = (V_1, V_2)$ be a 2-partition of $V(G)$ and s_1 and s_2 be two total orderings of V_1 and V_2, respectively.

Assume G is 2-PT concerning $\mathcal{V}$. Let $s = s_1 s_2$ be a precedence consistent ordering of $V(G)$. Thus, s_1 is a canonical ordering of $G[V_1]$. Proceeding by contradiction, suppose that s_1 is not a threshold ordering of $S_G(V_1, V_2)$. That is, there are $u, v \in V_1$ and $w \in V_2$, with $u < v$ in s_1, such that $w \in N[u]$ and $w \notin N[v]$. As $u < v < w$ in s, there is a contradiction with the fact that s is a precedence consistent ordering of $V(G)$. Therefore, (s_1, s_2) is in accordance with $G(V_1, V_2)$.



(a) G_1 (b) $S_{G_1}(V_1, V_2)$

(c) G_2 (d) $S_{G_2}(V_1, V_2)$

(e) G_3 (f) $S_{G_3}(V_1, V_2)$

Figure 3.10: Corresponding split graphs (V_1 is the set of orange vertices and V_2 the black ones).

On the other hand, assume that (s_1, s_2) is in accordance with G. Thus, both s_1 and s_2 are canonical orderings of $G[V_1]$ and $G[V_2]$, respectively, and s_1 is a threshold ordering of $S_G(V_1, V_2)$. Now we prove that $s = s_1 s_2$ is a precedence consistent ordering of $V(G)$ with respect to $\mathcal{V}$. In order to reach a contradiction, suppose that this statement does not hold. That is, there are $u, v \in V_1$ and $w \in V_2$, with $u < v$ in s, such that $uw \in E(G)$ and $vw \notin E(G)$. This results in a contradiction because s_1 is a threshold ordering of $S_G(V_1, V_2)$. Hence, $s = s_1 s_2$ is a precedence consistent ordering of $V(G)$ with respect to $\mathcal{V}$. □

Lemma 11. *Let G be a graph. Then, G is 2-PPT if, and only if, there is a strongly consistent ordering $s = s_1 s_2$ for which $\mathcal{V} = (V(s_1), V(s_2))$ is a 2-partition of $V(G)$ such that (s_1, s_2) is strongly in accordance with G.*

Proof. Let $\mathcal{V} = (V_1, V_2)$ be a 2-partition of $V(G)$ and s_1 and s_2 be two total orderings of V_1 and V_2, respectively.

Assume G is 2-PPT concerning $\mathcal{V}$. Let $s = s_1 s_2$ be a precedence strongly consistent ordering of $V(G)$. Thus, s_1 is a proper canonical ordering of $G[V_1]$ and, as G is also a 2-PT graph, s_1 is a threshold ordering of $S_G(V_1, V_2)$, according to Lemma 10. In order to reach a contradiction, suppose that $\bar{s_2}$ is not a threshold ordering of $S_G(V_1, V_2)$. That is, there are $u, v \in V_2$ and $w \in V_1$, with $u < v$ in s_2, such that $w \in N(v)$ and $w \notin N(u)$. This is a contradiction with the fact that s is a precedence strongly consistent ordering, as $w < u < v$ in s. Hence, (s_1, s_2) is strongly in accordance with $G(V_1, V_2)$.

Now assume (s_1,s_2) is strongly in accordance with G. That is, both s_1 and $\bar{s_2}$ (resp. s_1 and s_2) are threshold orderings (proper canonical orderings) of $S_G(V_1, V_2)$ (resp. $G[V_1]$ and $G[V_2]$, respectively). By Lemma 10, $s = s_1 s_2$ is a precedence consistent ordering of $V(G)$. Next, we prove that $s = s_1 s_2$ is also a precedence strongly consistent ordering of $V(G)$ with respect to $\mathcal{V}$. For the sake of contradiction, suppose that the statement does not hold. That is, there are $u, v \in V_2$ and $w \in V_1$, with $u < v$ in s, such that $vw \in E(G)$ and $uw \notin E(G)$. This is an absurd, as $\bar{s_2}$ is a threshold ordering of $S_G(V_1, V_2)$. Consequently, $s = s_1 s_2$ is a precedence strongly consistent ordering of $V(G)$ with respect to $\mathcal{V}$. □

The above lemmas can be generalized to an arbitrary number of parts as follows.

Theorem 12. *Let G be a graph. For all $k > 2$, G is k-PT (resp. k-PPT) if, and only if, there is a precedence consistent (resp. strongly consistent) ordering $s = s_1 \dots s_k$ for which $\mathcal{V} = (V(s_1), V(s_2), \dots, V(s_k))$ is a k-partition of $V(G)$ such that for all $1 \leq i < j \leq k$, (s_i, s_j) is in accordance (resp. strongly in accordance) with $G[V(s_i) \cup V(s_j)]$.*

Proof. Let $s = s_1 \dots s_k$ be a total ordering and $\mathcal{V} = (V(s_1), V(s_2), \dots, V(s_k))$ be a partition of V(G). Notice that s is a precedence consistent (resp. strongly consistent) ordering if, and only if, for all $1 \leq i < j \leq k$, $s_i s_j$ is a precedence consistent (resp. strongly consistent) ordering of $G[V_i \cup V_j]$ concerning the 2-partition (V_i, V_j). By Lemma 10 (resp. Lemma 11), it holds if, and only if, (s_i, s_j) is in accordance (resp. strongly in accordance) with $G[V(s_i) \cup V(s_j)]$. □

A *biclique* of a graph G is a subgraph of G whose vertices can be partitioned into two sets A, B such that every vertex of A is adjacent to every vertex of B. Theorem 12 leads to the following property with relation to k-PPT graphs and bicliques.

Property 3. *Let $(V_1, V_2, \dots, V_k)$ be a partition of $V(G)$, s be a precedence strongly consistent ordering and $1 \leq i, j \leq k$. If V_i precedes V_j in s and $v \in V_i$ is adjacent to $w \in V_j$, then the partition $\{z \in V_i \mid z = v$ or z succeeds v in $s\} \cup \{z \in V_j \mid z = w$ or z precedes w in $s\}$ forms a biclique of G.*

Chapter 4

Restricted Hamming-Huffman trees

In this Chapter, we define a more restricted version of the problem of building optimal Hamming-Huffman trees. We tackle the problem of building optimal Hamming-Huffman trees in which the leaves lie in exactly k distinct levels. These type of trees are defined as *k-Hamming-Huffman trees*, or *k-HHTs*. Formally, we define the k-HHT problem as

Problem:	k-HHT
Input:	A set of symbols Γ and, for all $a \in \Gamma$, the probability p(a) of occurrence of a in a message.
Output:	An HHT T in which all symbol leaves lie at exactly k levels of T and such that $c(T)$ is minimum.

We denote as *$[k]$-Hamming-Huffman tree* (*$[k]$-HHT*) a minimum cost HHT over all optimal k'-HHTs, for all $1 \leq k' \leq k$.

If $k \leq 2$, we provide a polynomial time algorithm to solve the problem. Otherwise, we present an algorithm to evaluate a lower bound on the optimal cost of such trees when its symbols have a uniform probability of occurrence. In this case, we also prove that there exists an optimal Hamming-Huffman tree having their symbol leaves lying on at most 4 consecutive levels. Finally we compare the efficiency, considering both data compression and data error detection capabilities, of [2]-HHTs over HTs an even trees (defined in Chapter 2).

4.1 Hamming Huffman trees with leaves in one level

In this section, we tackle the 1-HHT problem. This problem can be reduced to that of deciding the minimum height of the full binary HHT for which the symbol leaves can be arranged in the last level.

First, note that there is an important relation between optimal 1-HHTs with ℓ symbols and minimum neighborhoods of independent sets with ℓ elements of Q_n. Consider the one-to-one mapping between the leaves of a full binary 1-HHT T having height n to the vertices of Q_n, in which a leaf a corresponds to $c(a) \in V(Q_n)$. The problem of finding the minimum number of error leaves, over all possible trees T, is equivalent to that of finding, over all independent sets L of cardinality $\ell = |\Gamma|$ in $V(Q_n)$, one that minimizes $|N(L)|$. This is so because

$$\begin{aligned} T \text{ is a 1-HHT} &\iff d(c(u), c(v)) \geq 2 \text{ for all distinct } u, v \in \Gamma \\ &\iff L = \{c(u) \mid u \in \Gamma\} \text{ is an independent set of } Q_n \end{aligned}$$

Thus, for a given 1-HHT T, the set of errors leaves of T is precisely $N(L)$ in Q_n and $L = \{c(u) \mid u \in \Gamma\}$.

The efficient computation of $\varphi(\ell, n)$ is possible with the aid of Theorem 14. Before presenting the theorem, we have to state the following auxiliar lemma.

Lemma 13 ([25], [26])**.** *For any given non-negative integers ℓ and n, $\ell < 2^n$, the number m has a unique representation*

$$\ell = \binom{n}{n} + \binom{n}{n-1} + \ldots + \binom{n}{k+1} + \binom{a_k}{k} + \ldots + \binom{a_t}{t}$$

such that

$$n > a_k > a_{k-1} > \cdots > a_t \geq t \geq 1$$

The representation presented in Lemma 13 is defined as the *n-bounded canonical representation* of ℓ.

Also, in [26], Katona defined the function $G(\ell, n)$ as follows:

$$G(\ell, n) = \begin{cases} 0, & \text{if } \ell \leq 0 \\ \binom{n}{n} + \binom{n}{n-1} + \cdots + \binom{n}{k+1} + \binom{n}{k} + \binom{a_k}{k-1} + \cdots + \binom{a_t}{t-1}, & \text{otherwise} \end{cases}$$

where the a's are those from Lemma 13 with respect to the given ℓ.

The following theorem was proven in [27].

Theorem 14 (Theorem 2 in [28] and [27])**.** *For every $\ell \leq 2^{n-1}$*

$$\varphi(\ell, n) = G(\ell, n-1).$$

A consequence of this theorem in [28], is that all the vertices belonging to the independent set L yielding $|N(L)| = \varphi(\ell, n)$ can be assumed to be, without loss of generality, of the same parity. This fact can be used directly to solve the 1-HHT problem, as shown in the following theorem. Let $h(\ell) = \lceil \log_2 \ell \rceil + 1$.

Theorem 15. *Let Γ be a set of ℓ symbols, each $a \in \Gamma$ having probability $p(a)$. The height and the cost of an optimal 1-HHT T are, respectively, $h(\ell)$ and*

$$c(T) = h(\ell) \sum_{a \in \Gamma} p(a)$$

Proof. To find an optimal 1-HHT T, it is necessary to determine the minimum height h of T such that the symbols and their corresponding error leaves lie all at this level, that is, $\ell + \varphi(\ell, h) \leq 2^h$. By [28], one may consider without loss of generality that the set of ℓ codifications consists of elements having a same parity. In order to choose ℓ vertices with a same parity, one may use at most half the symbol leaves of that level. That is, a full binary 1-HHT must have at least height $\lceil \log_2 \ell \rceil + 1$ to be able to contain ℓ leaves with a same parity thus, $h \geq \lceil \log_2 \ell \rceil + 1$. On the other hand, let L be a set of codifications having a same parity. First note that L is an independent set of Q_n and the corresponding error leaves have the opposite parity. Therefore, any set of ℓ symbols leaves at level $\lceil \log_2 \ell \rceil + 1$ having a same parity consists of a valid 1-HHT. Consequently, $h \leq \lceil \log_2 \ell \rceil + 1$, yielding that $h = \lceil \log_2 \ell \rceil + 1$ and $c(T) = (\lceil \log_2 \ell \rceil + 1) \sum_{a \in \Gamma} p(a)$. □

4.2 Hamming Huffman trees with leaves in two levels

In this section, we discuss optimal 2-HHTs. We show that, similarly to the 1-HHTs, it is possible to build optimal 2-HHTs efficiently. We provide an algorithm for building optimal 2-HHTs that runs in time $O(\ell \log^2 \ell)$, where $\ell = |\Gamma|$.

A motivation to study this specific case lies in the fact that, for symbols with uniform probabilities of occurrence, there is always a Huffman tree with symbols in at most two different levels. This follows from Section 2.3.4.5 of Knuth's Vol. 1 [29]. It is not known whether this is also the case for Hamming-Huffman trees. Experimental results were designed to investigate this hypothesis and the results are presented in Section 4.4. Furthermore, recall that the problem of building optimal general HHTs is open since the eighties. Thus, the approach of studying more restrictive cases is worthy, since a solution for a particular case may have practical value or lead to a solution for general HHTs.

The strategy for determining an optimal 2-HHT with leaves at two levels h_1 and h_2, with $h_1 < h_2$, is devised as follows. Note that by Theorem 15, a full binary tree having height $h(\ell)$ is enough to build a 1-HHT. Therefore, $h(\ell)$ is a natural upper bound on h_1, that is, $1 \leq h_1 < h(\ell)$.

First, consider any specific value for h_1. For such a value, let ℓ_1 be the number of symbol leaves that are placed at level h_1. Note that $1 \leq \ell_1 \leq \min\{\ell - 1, 2^{h_1 - 1}\}$, since 2^{h_1} is the maximum number of nodes at level h_1, and half of them have the same parity.

Once ℓ_1 symbol leaves are chosen to be placed at level h_1, there will be error nodes corresponding to such symbol leaves at this same level, and the remaining nodes will be free nodes from which the tree can grow to achieve larger levels (in particular, to achieve

level h_2, where the remaining symbol leaves must lie). As seen in Figure 2.8, distinct sets of leaves lead to distinct sets of error leaves, the latter varying considerably in size. Clearly, to minimize the cost of the solution for the fixed values of (h_1, ℓ_1), it suffices to minimize the value of h_2. To do that, it suffices to distribute as uniformly as possible the remaining $\ell_2 = \ell - \ell_1$ symbols leaves over the subtrees rooted at the free nodes. Indeed, it is possible to arrange all symbol leaves at level h_2 of each subtree all having the same parity, ensuring that the leaves at level h_2 pairwise have Hamming distance of at least 2. Therefore, the aim is to choose the set of symbol leaves at level h_1 in such a way that the number of free nodes is maximized or, equivalently, that the number of error leaves is minimized. In other words, the algorithm must select a set of symbol leaves at level h_1 which produces $\varphi(\ell_1, h_1)$ corresponding error nodes. For this choice, the maximum number of free nodes, that will be denoted by $\rho(\ell_1, h_1)$, is given by

$$\rho(\ell_1, h_1) = 2^{h_1} - \ell_1 - \varphi(\ell_1, h_1) \tag{4.1}$$

For the cost $c(T) = c(\ell, \ell_1, h_1)$ of this particular way of building a 2-HHT T having ℓ_1 symbols at level h_1, we present an algorithm to evaluate this value. If $\rho(\ell_1, h_1) = 0$, then it is not possible to grow the tree to allocate the remaining ℓ_2 symbols at level h_2. So, the choices of h_1, ℓ_1 turned out to lead to an unfeasible solution. If $\rho(\ell_1, h_1) > 0$, then the remaining ℓ_2 symbols are uniformly distributed in $\rho(\ell_1, h_1)$ subtrees rooted at the free nodes, each receiving at most $\left\lceil \frac{\ell_2}{\rho(\ell_1, h_1)} \right\rceil$ symbols, and one of them receiving exactly such an amount. Minimizing the common height in each subtree rooted at each free node is a 1-HHT problem. By using the result $h(\ell)$ of Theorem 15, we have that the minimum height $h'(\ell_1, h_1)$ required for each subtree to accommodate those symbols is given by

$$h'(\ell_1, h_1) = h\left(\left\lceil \frac{\ell_2}{\rho(\ell_1, h_1)} \right\rceil \right)$$

To determine $c(\ell, \ell_1, h_1)$ in this case, it is needed to assign the set of symbols Γ to the set of chosen symbols leaves. But to minimize such a cost, it clearly suffices to place at level h_1 the ℓ_1 symbols with the highest probability of occurrence. Assuming that $\Gamma = \{a_1, a_2, \ldots, a_\ell\}$ is ordered decreasingly according to their respective probability of occurrence, we have that

$$
\begin{aligned}
c(\ell, \ell_1, h_1) &= h_1 \sum_{i=1}^{\ell_1} p(a_i) + h_2 \sum_{i=\ell_1+1}^{\ell} p(a_i) \\
&= h_1 \sum_{i=1}^{\ell_1} p(a_i) + (h_1 + h'(\ell_1, h_1)) \sum_{i=\ell_1+1}^{\ell} p(a_i) \\
&= h_1 \sum_{i=1}^{\ell} p(a_i) + h'(\ell_1, h_1) \sum_{i=\ell_1+1}^{\ell} p(a_i) \\
&= h_1 + h'(\ell_1, h_1) \sum_{i=\ell_1+1}^{\ell} p(a_i)
\end{aligned}
$$

Figure 4.1 depicts this strategy. The nodes labeled with "s" represent symbol leaves, the black nodes represent the error leaves, and the dashed nodes represent the free nodes.

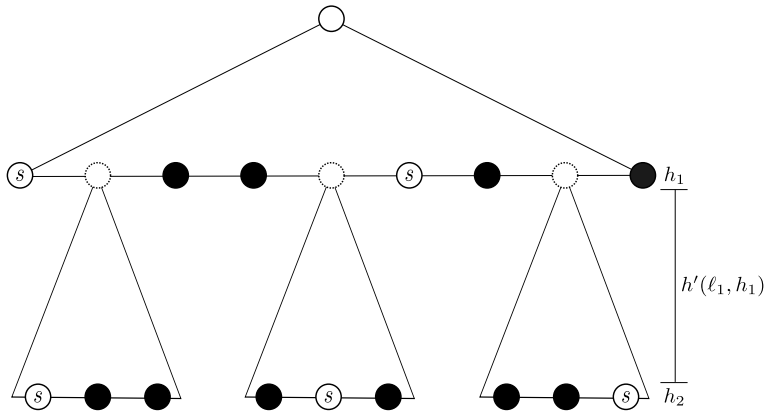


Figure 4.1: A Hamming-Huffman tree with leaves on two levels.

The optimal cost is the minimum cost obtained by varying h_1 and ℓ_1 over all possible values. Formally, the cost of an optimal 2-HHT T is given by

$$
c(T) = \min\{c(\ell, \ell_1, h_1) \mid 1 \leq h_1 < h(\ell)\ ,\ 1 \leq \ell_1 \leq \min\{\ell - 1, 2^{h_1 - 1}\}\}
$$

where

$$
c(\ell, \ell_1, h_1) = h_1 + h'(\ell_1, h_1) \sum_{i=\ell_1+1}^{\ell} p(a_i)
$$

Concerning the computational complexity for determining the optimal cost, for each $1 \leq h_1 < h(\ell)$, there are at most $2^{h_1 - 1}$ possible values for ℓ_1. Therefore, there are at most $1 + 2 + 2^2 + \ldots + 2^{\lceil \log_2 \ell \rceil - 1} = 2^{\lceil \log_2 \ell \rceil} - 1 = \Theta(\ell)$ distinct pairs (h_1, ℓ_1). Moreover, for the computation of each $c(\ell, \ell_1, h_1)$, the evaluation of $\varphi(\ell_1, h_1)$ is required. This evaluation can be computed in time $O(h_1^2) = O(\log^2 \ell)$ with the aid of a precomputed Pascal triangle. Besides that, a precomputed sum of values $\sum_{j=1}^{i} p(a_i)$ for all $1 \leq i \leq \ell$, which can be done in time $\Theta(\ell)$, can be used to obtain the summation present in $c(\ell, \ell_1, h_1)$ in constant time. Therefore, the complexity of evaluating the cost of an optimal tree is $O(\ell \log^2 \ell)$.

The results of this section can be summarized by the following theorem.

Theorem 16. *Let Γ be a set of ℓ symbols, each $a \in \Gamma$ having probability $p(a)$. The cost of an optimal 2-HHT T is*

$$c(T) = \min\{c(\ell, \ell_1, h_1) \mid 1 \leq h_1 < h(\ell)\ ,\ 1 \leq \ell_1 \leq \min\{\ell - 1, 2^{h_1 - 1}\}\}$$

where

$$c(\ell, \ell_1, h_1) = h_1 + h'(\ell_1, h_1) \sum_{i=\ell_1+1}^{\ell} p(a_i)$$

and

$$h'(\ell_1, h_1) = h\left(\left\lceil \frac{\ell_2}{\rho(\ell_1, h_1)} \right\rceil \right)$$

Furthermore, this cost can be computed in time $O(\ell \log^2 \ell)$.

4.3 Uniform Hamming-Huffman trees

In this section, we discuss the problem of building optimal uniform HHTs, that is, trees in which all symbols have the same probability of occurrence. In contrast to 2-HHTs, even for the more restrictive case of uniform probabilities, an efficient algorithm for building an optimal uniform HHT will remain open. Let $\lambda(T)$ be the difference between the last and the first levels of T which have at least one symbol leaf at that level. For instance, for T as in Figure 2.6(a), $\lambda(T) = 1$, where $\lambda(T) = 0$ for any tree T of Figure 2.8. We prove that $\lambda(T) \leq 4$ for all optimal uniform HHT T. Moreover, we show that there is always an optimal uniform HHT T in which $\lambda(T) \leq 3$. In addition, all optimal uniform HHTs are [5]-HHTs, and there exists an optimal uniform HHT which is a [4]-HHT. Finally, we present a dynamic programming algorithm to evaluate a lower bound on the cost of such a tree.

Recall that (optimal) HTs for symbols with uniform frequencies have all leaves in at most two levels. It is unknown whether the same holds for HHTs. We consider the conjecture that there is always an optimal uniform HHT in which the symbol leaves are distributed in $k \leq 3$ distinct levels, and we provide empirical evidence in favor of it. Section 4.4 compares the lower bound of this section with the cost of optimal 2-HHTs, the latter being computed as presented in Section 4.2.

Let T be a uniform HHT on ℓ symbols. Consider the following operation over a symbol leaf s of T:

- *descend*(s) (see Figure 4.2): replace the leaf s by a full binary HHT T_s having height two in such a way that this leaf becomes the root of T_s. The tree T_s is such that one of its leaves is the symbol leaf s. Note that there are exactly two error leaves associated with s among the leaves of T_s, besides one free node s', regardless which leaf corresponds to s. Next, transform s' into a symbol leaf associated with any

symbol s'' that appears in the last level of T. Finally, transform the node of s'' into an error leaf. Let T' be the resulting tree. Apply $contract(T')$ to obtain the final transformation.

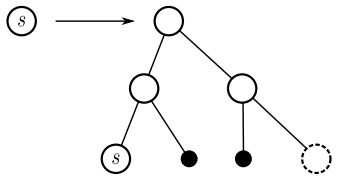


Figure 4.2: Operation $descend(s)$, over a symbol leaf. The dotted node represents a free node to be used in the encoding of another symbol.

Let T' be the resulting tree after applying $descend(s)$. The following lemma proves that T' is also an HHT.

Lemma 17. *Let T be an HHT, s be one of its symbol leaves and T' be the tree obtained by the operation descend(s). The tree T' is an HHT.*

Proof. We shall prove that, in T', all the leaves with Hamming distance one to s are error leaves. As the root of T_s comes from a symbol leaf in T, all nodes with Hamming distance one to it in T' are error leaves. Moreover, as T_s is an HHT by construction, all the nodes in T' with Hamming distance one to s and s' are also error leaves in T'. Finally, the transformation carried out in the last step ensures that T' does not contain two sibling leaves which are both error leaves. That is, every node of T' is either an error leaf or an ancestor of a symbol leaf. Therefore, T' is an HHT. □

Let $p = \frac{1}{\ell}$ be the probability of occurrence of the symbol leaves of T and T' be the tree obtained by the operation $descend(s)$, for some symbol leaf s of T. Let l_1 be the level of s and l_2 be the last level of T. Note that, the symbol of s'' was moved from level l_2 to level $l_1 + 2$. Moreover, the symbol of s was moved from level l_1 to level $l_1 + 2$. Therefore, the cost of T' can be written as a function of the cost of T as

$$c(T') = c(T) - p(\lambda(T) - 4) \tag{4.2}$$

Theorem 18. *Let T be a uniform HHT. If T is optimal, then $\lambda(T) \leq 4$. Moreover, there is always an optimal T for which $\lambda(T) \leq 3$.*

Proof. We prove that when $\lambda(T) > 4$ is always possible to obtain an HHT T' from T such that $c(T') < c(T)$, contradicting the optimality of T.

Let l_1 and l_k be the first and the last level of T containing symbol leaves, respectively. Let p be the probability of occurrence of each symbol associated to T. Apply $descend(s)$ to some symbol leaf s at level l_1 to obtain T'. If $\lambda(T) > 4$, then $\lambda(T)p > 4p$ and, equivalently, $\lambda(T)p - 4p = p(\lambda(T) - 4) > 0$. Therefore, by (4.2), we have that $c(T') < c(T)$. Moreover, note that each application of $descend(s)$ eliminates a leaf in level l_1 and a leaf in level l_k. By successive applications of descend(s) to symbol leaves, it is possible to obtain an optimal uniform HHT such that $\lambda(T) \leq 3$. □

We will proceed in the remaining of this section by providing a lower bound on the cost of uniform HHTs. For this, we need to generalize the concept of Hamming-Huffman trees. A *k-Hamming-Huffman forest*, or *k-HHF*, is a forest F of HHTs such that the symbol leaves are distributed among exactly k distinct levels of F. Note that the trees of F may have a height greater than k since there might be some levels of F with no symbol leaves. The cost of a k-HHF is defined by the sum of the costs of its HHTs.

The strategy to derive the lower bound on the cost of a k-HHF for ℓ symbols and r trees is as follows. Consider h_1 to be the first level in which symbol leaves appear in F. Let ℓ_1 be the number of symbol leaves to be represented in the level h_1. Clearly, the most desirable arrangement for choosing ℓ_1 nodes at level h_1 is one in which the corresponding error leaves are minimized, that is, in which the free nodes are maximized. This is so because the remaining $\ell - \ell_1$ symbol leaves must be allocated as descendants of the resulting free nodes at level h_1, and the more resulting free nodes, the better. At this point, this strategy will deal with all those free nodes as independent trees of a $(k-1)$-HHF. But, some of them may actually be part of the same HHT of the k-HHF and, because of that, the symbol leaves allocated in a tree descending from a free node produce error leaves that may conflict with the allocation of symbol leaves descending from another free node. Since the possibility of conflict will not be dealt with, the resulting k-HHF may not be feasible and that is why this strategy yields a lower bound on the cost of this k-HHF. The lower bound, defined as $c_F(k, r, \ell)$, on the cost of an optimal k-HHF of r disjoint HHTs for ℓ symbols derived from this strategy is evaluated as follows.

First note that if $\ell = 0$, then $c_F(k, r, \ell) = 0$. Also, if $r = 0$ (resp. $k = 0$) and $\ell \geq 1$, it means that there are not enough free nodes to accommodate the remaining ℓ symbols. In other words, this scenario leads to an unfeasible solution and, therefore, $c_F(k, r, \ell) = +\infty$. If $k = 1$ and $\ell, r \geq 1$, the resulting problem is equivalent to the one of distributing ℓ symbols among r 1-HHTs with the same height. In particular, each one of these trees must have at least $h(\lceil \frac{\ell}{r} \rceil)$ leaves to be able to accommodate all the symbols. Thus, using the same reasoning as the one used in Theorem 15 and, as the symbols have an equal probability of occurrence, $c_F(k, r, \ell) = h(\lceil \frac{\ell}{r} \rceil)$. For the general case, the algorithm minimizes the cost over all possible pairs (ℓ_1, h_1). Note that despite the fact that there are only ℓ_1 symbols at level h_1, all the remaining $\ell - \ell_1$ symbols have a prefix with size h_1 in their codifications. Given that, the first part of the cost of the general case is given by $h_1(\ell_1 + (\ell - \ell_1)) = h_1\ell$.

For the remaining part of the cost, the algorithm uses pre-computed values to solve the problem of distributing $\ell - \ell_1$ symbols among a $(k-1)$-HHF in which the roots are the resulting free nodes, in a dynamic programming fashion. Formally, $c_F(k, r, \ell)$ can be expressed as

$$c_{\mathrm{F}}(k,r,\ell) = \begin{cases} 0, & \text{if } \ell = 0 \\ +\infty, & \text{if } \ell \geq 1, \\ & (r = 0 \text{ or } k = 0) \\ h(\lceil \frac{\ell}{r} \rceil), & \text{if } \ell, r \geq 1,\ k = 1 \\ \min_{\ell_1 \in D_1 \text{ and } h_1 \in D_2} \{h_1\ell + c_F(k-1, \rho_F(r,\ell_1,h_1), \ell-\ell_1)\}, & \text{otherwise} \end{cases}$$

where:

- $D_1 = \{1, \ldots, \ell - k + 1\}$,
- $D_2 = \{h(\lceil \frac{\ell_1}{r} \rceil), \ldots, h(\lceil \frac{\ell}{r} \rceil)\}$, and
- $\rho_F(r, \ell, h)$ denotes the maximum number of free nodes when ℓ symbol leaves are allocated at level h of an HHF consisting of r HHTs, and h is the first level having leaves. The computation of ρ_F will be discussed next.

For 2-HHTs, $c_F(2, 1, \ell)$ is exactly the cost of a uniform 2-HHT using the algorithm presented in Section 4.2. Moreover, considering general uniform HHTs, the cost of an optimal uniform HHT with ℓ symbol is at least

$$\min\{c_F(k, 1, \ell) \mid 1 \leq k \leq \ell\} \tag{4.3}$$

Figure 4.3 depicts the strategy being adopted in the computation of c_F.

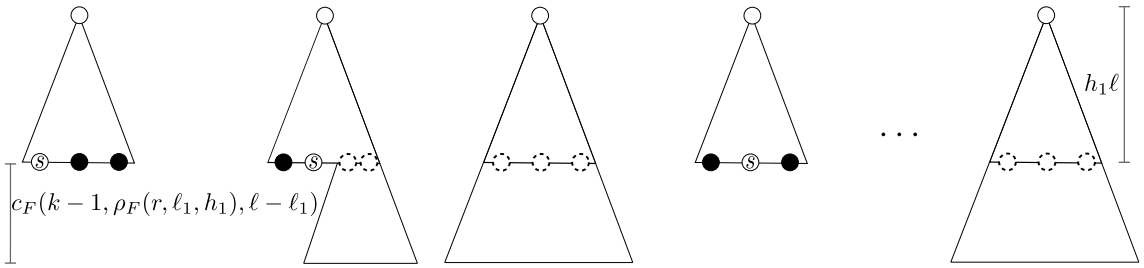


Figure 4.3: Example of the strategy used to evaluate $c_F(k, r, \ell)$.

The computation of $\rho_F(r, \ell, h)$ will also be carried out by a dynamic programming algorithm. First, note that $\rho_F(r, \ell, h)$ equals the maximum number of free nodes when ℓ symbols are distributed among the leaves of r full HHTs with height h. So, the strategy to yield the recurrence is as follows. First, suppose that ℓ_1 symbols are to be allocated into a single HHT. Therefore, the remaining $\ell - \ell_1$ symbols have to be allocated among the leaves of $r - 1$ full HHTs with height h. Both allocations must be done in such a way that the number of free nodes is maximized. The former can be computed with the aid of the formula given in (4.1). The latter can be determined using recursion. Formally, we

have

$$\rho_F(r,\ell,h) = \begin{cases} r2^h, & \text{if } \ell = 0 \\ -\infty, & \text{if } \ell \geq 1, \ (r = 0 \text{ or } \\ & h = 0 \text{ or } \ell > r2^{h-1}) \\ \max_{0 \leq \ell_1 \leq \min\{2^{h-1},\ell\}} \{\rho(\ell_1,h) + \rho_F(r-1,\ell-\ell_1,h)\}, & \text{otherwise.} \end{cases}$$

Precomputing the values of ρ_F requires a matrix whose number of elements is $O(\ell^2 \log \ell)$, as r is limited by ℓ and h is limited by $h(\ell)$. Moreover, as the processing of each cell of such matrix requires $O(\ell)$ steps, precomputing the values of ρ_F takes time $O(\ell^3 \log \ell)$. Assuming that the values of ρ_F are available at constant time, precomputing the values of c_F depends on a matrix whose number of elements is $O(\ell^2)$, as $k \leq 4$ by Theorem 18 and the remaining parameters are limited by ℓ. Furthermore, since processing each cell of such a matrix requires $O(D_1 D_2) = O(\ell \log \ell)$ steps, evaluating c_F takes time $O(\ell^3 \log \ell)$. Therefore, the proposed lower bound can be computed in time $O(\ell^3 \log \ell)$ and space $O(\ell^3)$.

4.4 Experimental Results

In this section, we describe some experimental results which have been performed in the context of the previous sections.

We have conducted three experiments. The first one is related to the algorithm described in Section 4.2. It compares uniform [2]-HHTs with the lower bound described in Section 4.3. The second experiment compares general [2]-HHTs with the Huffman trees aiming to enlighten the tradeoffs of both strategies. In the third experiment we have implemented a backtracking that finds an optimal uniform Hamming-Huffman tree for $1 \leq \ell \leq 38$.

Implementations of such algorithms were executed on a notebook having a CPU Core i7, with 8 GB RAM, running Ubuntu 16.04 OS. The algorithms were implemented in C++. The results are presented next.

All the programs related to this section are available at [30].

4.4.1 Uniform [3]-HHT optimality hypothesis

As the first experiment, we tested the hypothesis that uniform [3]-HHTs are indeed optimal. In this case, for all $1 \leq \ell \leq 4096$, we have compared the costs of the algorithms in Section 4.1 and Section 4.2 with those produced by the algorithm in Section 4.3. Some values of this comparison are presented in Table 4.1. The first column represents the number of symbols. The second shows the cost of the corresponding [2]-HHT. The third represents the cost of the lower bound described in Section 4.3. This column is divided into two parts. The first is the minimum k value that minimized the cost of the resulting tree and the second is the cost of the tree. The last column of the table gives the relative

differences between the costs presented in the last two columns. These costs and their relative differences are depicted in Figure 4.4 and Figure 4.5, respectively. By observing the table and the figures one can note that, for symbols with uniform probability of occurrence, the cost of [2]-HHTs are very close to the ones of the lower bound, for all tested values of ℓ. The difference between these costs was no more than 2.1%. Moreover, all the trees obtained by the lower bound have symbol leaves in at most three different levels.

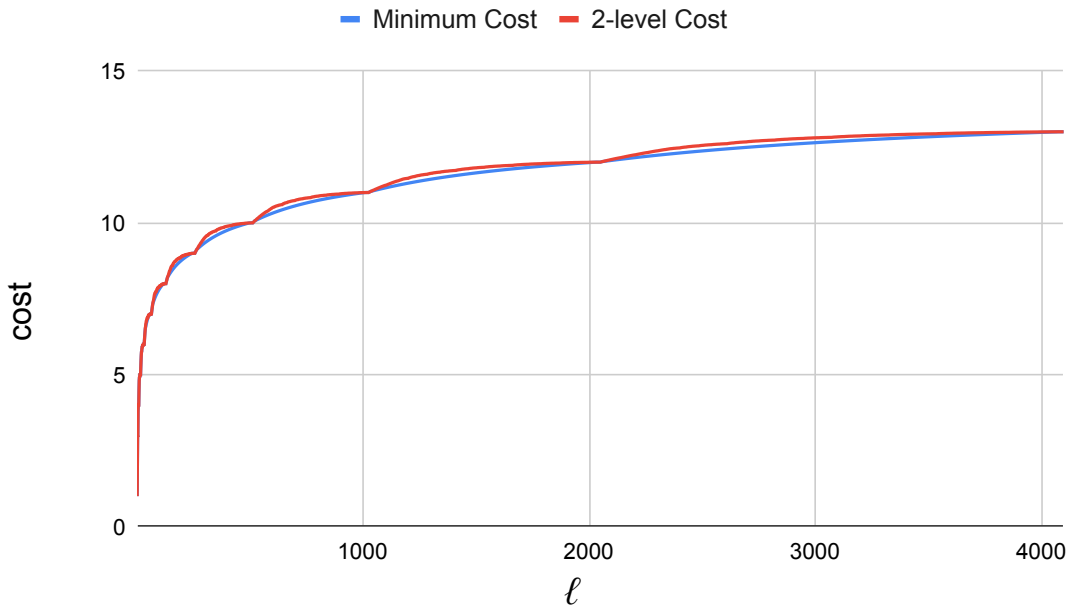


Figure 4.4: Costs of optimal uniform [2]-HHTs and the lower bound of uniform k-HHTs for ℓ symbols.

4.4.2 [2]-HHTs efficiency

In the second experiment, we have compared the costs and the error detection capabilities between [2]-HHTs and Huffman trees. The goal of this experiment is to present the tradeoffs of using [2]-HHTs instead of Huffman trees. In this comparison, we analyze their differences in compression and error detection rates.

Considering the compression, we have performed two tests. First, we compared the costs of uniform [2]-HHTs with the costs of uniform HTs. The second test compares the costs of [2]-HHTs and HTs for the Zipf distribution. The Zipf distribution is well-known for its empirical correspondence with the frequencies of words in natural languages [31]. This relation describes that the i-th most frequent word in an alphabet occurs with frequency $\frac{1}{i}$. We use this distribution to simulate real-world compressions. Both these comparisons were done for $10 \leq \ell \leq 1111110$ and the results are shown in Table 4.2, which is organized similarly to Table 4.1. For both cases, the difference in the costs of the trees was inversely proportional to the number of symbols being encoded. Considering uniform trees, this difference converged to around 5% and, for the Zipf's distribution, this difference converged

ℓ	[2]-HHT cost	lower bound k-HHT		% diff
		k	cost	
3	3.000	1	3.000	0.000
85	7.741	3	7.624	1.543
167	8.725	3	8.545	2.102
249	9.000	1	9.000	0.000
331	9.689	3	9.498	2.004
413	9.908	3	9.797	1.137
495	9.996	2	9.996	0.000
577	10.385	3	10.255	1.268
659	10.663	3	10.472	1.826
741	10.810	3	10.641	1.585
823	10.903	3	10.776	1.173
905	10.962	3	10.887	0.690
987	10.995	3	10.980	0.138
1069	11.153	3	11.102	0.463
1151	11.379	3	11.237	1.260
1233	11.540	3	11.354	1.636
1315	11.651	3	11.457	1.693
1397	11.722	3	11.548	1.506
1479	11.800	3	11.628	1.477
1561	11.853	3	11.700	1.303
1643	11.890	3	11.765	1.066
1725	11.930	3	11.824	0.897
1807	11.957	3	11.877	0.676
1889	11.978	3	11.926	0.439
1971	11.993	3	11.971	0.186
2053	12.010	2	12.010	0.000
2135	12.144	3	12.091	0.438
2217	12.263	3	12.162	0.834
2299	12.371	3	12.227	1.177
2381	12.459	3	12.289	1.388
2463	12.527	3	12.346	1.470
2545	12.580	3	12.399	1.458
2627	12.628	3	12.449	1.437
2709	12.680	3	12.496	1.474
2791	12.719	3	12.540	1.429
2873	12.756	3	12.582	1.383
2955	12.787	3	12.621	1.314
3037	12.811	3	12.658	1.210
3119	12.841	3	12.693	1.164
3201	12.867	3	12.727	1.097
3283	12.889	3	12.759	1.017
3365	12.908	3	12.789	0.934
3447	12.922	3	12.818	0.815
3529	12.934	3	12.845	0.693
3611	12.954	3	12.872	0.639
3693	12.966	3	12.897	0.535
3775	12.976	3	12.921	0.428
3857	12.983	3	12.943	0.302
3939	12.992	3	12.965	0.204
4021	12.997	3	12.987	0.080

ℓ	[2]-HHT cost	lower bound k-HHT		% diff
		k	Cost	
44	6.841	3	6.750	1.347
126	8.000	1	8.000	0.000
208	8.928	3	8.832	1.089
290	9.421	3	9.286	1.448
372	9.831	3	9.664	1.725
454	9.965	3	9.905	0.600
536	10.164	3	10.121	0.424
618	10.560	3	10.371	1.826
700	10.744	3	10.561	1.731
782	10.859	3	10.712	1.373
864	10.936	3	10.834	0.940
946	10.981	3	10.936	0.416
1028	11.016	2	11.016	0.000
1110	11.270	3	11.172	0.879
1192	11.463	3	11.298	1.463
1274	11.594	3	11.407	1.638
1356	11.691	3	11.504	1.628
1438	11.766	3	11.589	1.530
1520	11.825	3	11.665	1.370
1602	11.873	3	11.733	1.186
1684	11.913	3	11.795	0.997
1766	11.944	3	11.851	0.784
1848	11.968	3	11.902	0.550
1930	11.987	3	11.949	0.321
2012	11.998	3	11.992	0.054
2094	12.081	3	12.054	0.226
2176	12.207	3	12.127	0.656
2258	12.317	3	12.195	0.999
2340	12.421	3	12.259	1.321
2422	12.495	3	12.318	1.438
2504	12.555	3	12.373	1.472
2586	12.601	3	12.424	1.419
2668	12.656	3	12.473	1.472
2750	12.699	3	12.518	1.444
2832	12.737	3	12.561	1.403
2914	12.773	3	12.602	1.362
2996	12.801	3	12.640	1.273
3078	12.822	3	12.676	1.153
3160	12.855	3	12.710	1.140
3242	12.879	3	12.743	1.070
3324	12.894	3	12.774	0.940
3406	12.917	3	12.804	0.885
3488	12.930	3	12.832	0.766
3570	12.946	3	12.859	0.677
3652	12.959	3	12.884	0.582
3734	12.970	3	12.909	0.473
3816	12.980	3	12.932	0.369
3898	12.988	3	12.955	0.259
3980	12.994	3	12.976	0.141
4062	12.999	3	12.997	0.019

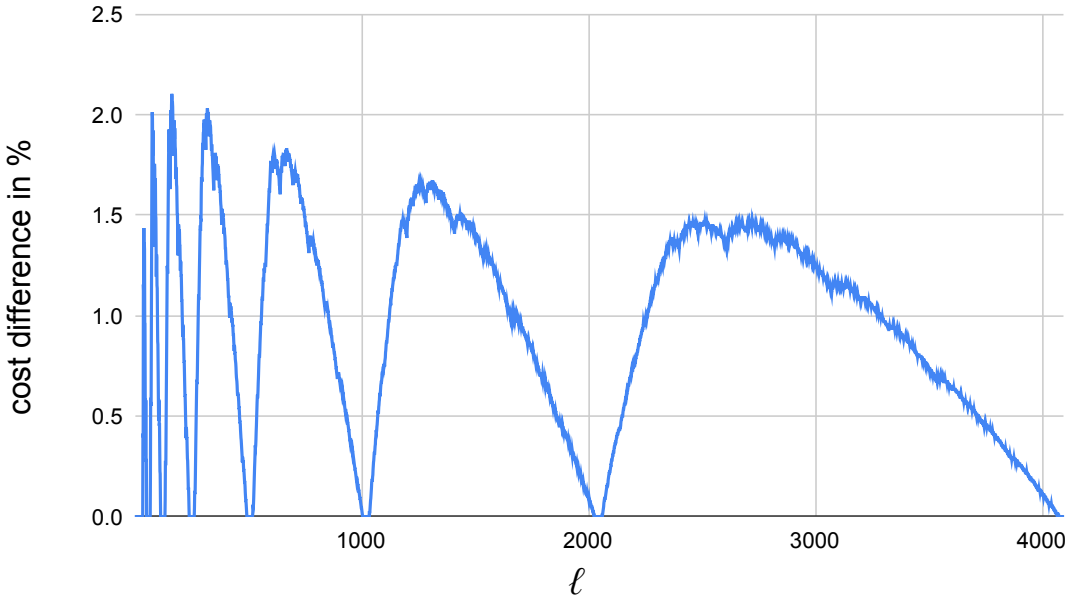


Figure 4.5: Difference in percent between the cost of optimal uniform [2]-HHTs and the cost of the lower bound of uniform k-HHTs for ℓ symbols.

to around 25%.

Concerning error detection, we have compared optimal [2]-HHTs, HTs, and even trees. Notice that the Huffman trees have some sort of error detection capability. This occurs when, at the end of the process of decoding, the last node being processed by the HT is not a leaf. In this case, it means that some bits of the message have been corrupted. For this experiment, we build [2]-HHTs and HTs considering the Zipf distribution. The results reported for even trees are those from [32] in which a similar strategy of testing has been used. We have chosen a value for ℓ, in the range $10 \leq \ell \leq 500000$, in such a way that the related optimal [2]-HHT has the value

$$\frac{\text{number of symbol leaves}}{\text{number of error leaves}}$$

maximized. That is, the resulting tree minimizes the proportion of error leaves in comparison with symbols leaves, meaning that such a tree is the one that has the least capacity of error detection. For the given ℓ, we created an optimal [2]-HHT and an HT for ℓ symbols considering the Zipf distribution. For such trees, we have tested random messages with b symbols, $b \in \{10, 25, 50, 100, 250, 500, 1000, 2500, 5000\}$. For each one of these messages, we introduced i random errors in their bits, for all $1 \leq i \leq \min\{b, 20\}$. For each value of i, we ran the test one million times, counting how many times the tree could detect the error. The percentage of detection of each tree is presented in Table 4.3. In this table, one may observe that the error detection capability of HTs seems to decrease as b increases. Also, comparing even trees with the optimal [2]-HHT, one may note that in both trees the error detection capability seems to be proportional to b. Moreover, the optimal [2]-HHT

Uniform Probabilities			
ℓ	[2]-HHT cost	Huffman cost	% diff
10	4.800	3.400	41.176
20	5.750	4.400	30.682
30	6.000	4.933	21.622
40	6.700	5.400	24.074
50	6.940	5.720	21.329
60	7.000	5.933	17.978
70	7.343	6.171	18.981
80	7.675	6.400	19.922
90	7.822	6.578	18.919
100	7.920	6.720	17.857
110	7.973	6.836	16.622
210	8.933	7.781	14.810
310	9.584	8.348	14.799
410	9.905	8.751	13.183
510	10.000	8.996	11.160
610	10.538	9.321	13.050
710	10.765	9.558	12.629
810	10.894	9.736	11.894
910	10.965	9.875	11.039
1010	10.999	9.986	10.143
1110	11.270	10.155	10.983
2110	12.108	11.059	9.488
3110	12.839	11.683	9.892
4110	13.013	12.007	8.381
5110	13.581	12.397	9.552
6110	13.814	12.659	9.125
7110	13.941	12.848	8.508
8110	13.998	12.990	7.764
9110	14.334	13.202	8.576
10110	14.549	13.379	8.743
11110	14.696	13.525	8.653
21110	15.611	14.448	8.055
31110	15.983	14.947	6.936
41110	16.565	15.406	7.522
51110	16.845	15.718	7.171
61110	16.974	15.928	6.573
71110	17.258	16.157	6.814
81110	17.532	16.384	7.009
91110	17.706	16.561	6.912
101110	17.830	16.704	6.741
111110	17.911	16.820	6.484
211110	18.867	17.758	6.243
311110	19.450	18.315	6.201
411110	19.838	18.725	5.943
511110	19.992	18.974	5.363
611110	20.408	19.284	5.828
711110	20.663	19.525	5.826
811110	20.821	19.707	5.654
911110	20.925	19.849	5.419
1011110	20.987	19.963	5.131
1111110	21.183	20.113	5.321

Zipf Distribution			
ℓ	[2]-HHT cost	Huffman cost	% diff
10	4.327	2.934	47.479
20	5.263	3.669	43.442
30	5.732	4.102	39.757
40	6.016	4.403	36.634
50	6.271	4.636	35.270
60	6.497	4.825	34.645
70	6.659	4.988	33.498
80	6.811	5.126	32.854
90	6.959	5.246	32.652
100	7.045	5.352	31.622
110	7.145	5.447	31.158
210	7.854	6.073	29.331
310	8.277	6.436	28.601
410	8.589	6.695	28.293
510	8.833	6.897	28.066
610	9.015	7.062	27.656
710	9.159	7.201	27.198
810	9.301	7.323	27.014
910	9.446	7.430	27.134
1010	9.538	7.525	26.758
1110	9.658	7.610	26.912
2110	10.372	8.187	26.695
3110	10.731	8.534	25.750
4110	11.052	8.780	25.871
5110	11.300	8.973	25.937
6110	11.451	9.130	25.420
7110	11.652	9.263	25.786
8110	11.789	9.378	25.707
9110	11.944	9.480	25.986
10110	12.035	9.572	25.733
11110	12.129	9.654	25.633
21110	12.817	10.212	25.513
31110	13.230	10.544	25.474
41110	13.542	10.782	25.600
51110	13.737	10.965	25.280
61110	13.942	11.115	25.430
71110	14.149	11.243	25.852
81110	14.264	11.353	25.641
91110	14.371	11.450	25.517
101110	14.466	11.536	25.392
111110	14.582	11.615	25.543
211110	15.269	12.149	25.678
311110	15.722	12.471	26.069
411110	15.985	12.701	25.851
511110	16.225	12.881	25.955
611110	16.466	13.028	26.385
711110	16.596	13.153	26.174
811110	16.706	13.261	25.973
911110	16.837	13.357	26.049
1011110	16.960	13.443	26.163
1111110	17.080	13.520	26.331

Table 4.2: Comparison between the costs of optimal [2]-HHTs and HTs.

b	Huffman Tree (%)	Even Tree (%)	Optimal [2]-HHT (%)
10	66.1303	90.3540	98.9303
25	73.9825	98.7620	99.8219
50	67.0522	99.0953	99.9172
100	42.4929	99.4894	99.9584
250	34.3946	99.8482	99.9835
500	38.6768	99.9150	99.9923
1000	9.4573	99.9595	99.9958
2500	1.7315	99.9831	99.9983
5000	2.3790	99.9922	99.9994

Table 4.3: Comparison between the error detection capabilities of optimal HTs, even trees and [2]-HHTs.

seems to have a significantly greater detection capability. For instance, for a message with 500 symbols, the optimal [2]-HHT achieves an error detection rate that is achieved by the even tree only when the message has 5000 symbols.

4.4.3 Backtracking for optimal uniform HHTs

In the third experiment, we used backtracking to build an optimal uniform HHT for all $1 \leq \ell \leq 38$. We have concluded that, for these values of ℓ, there is always an optimal Hamming-Huffman tree with at most two levels with symbol leaves. Besides that, in some cases, there is also an optimal tree with more than two levels. For instance, for $\ell = 38$, there is also an optimal tree with three levels. Another interesting aspect of this experiment is the fact that, considering optimal uniform Hamming-Huffman trees for 5 symbols, the backtracking obtained the tree depicted in Figure 4.6 which has a different structure from the one depicted in Figure 2.6, presented in the literature.

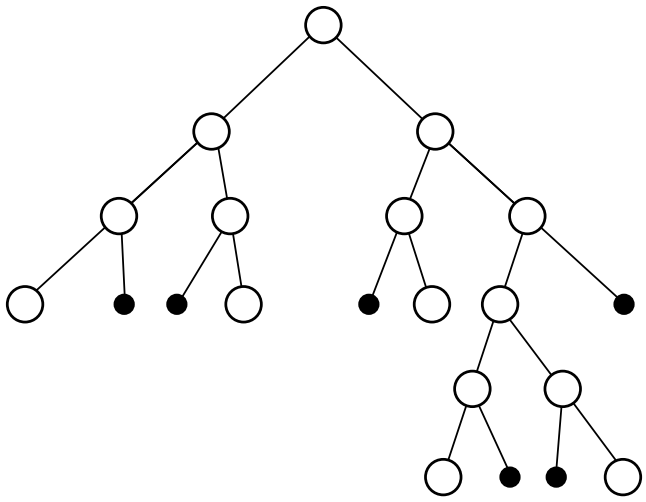


Figure 4.6: An optimal uniform Hamming-Huffman tree for 5 symbols.

Chapter 5

Conclusions

In this thesis, we have studied two topics of computer science, namely, thinness in graphs and Hamming-Huffman trees. The main contributions of this work for both topics are described next.

With respect to thinness in graphs, we have introduced two classes of graphs: precedence k-thin and precedence proper k-thin graphs, subclasses of k-thin and proper k-thin graphs, respectively. Concerning precedence k-thin graphs, we presented a polynomial time algorithm that receives as input a graph G and a k-partition of $V(G)$ and decides whether G is a precedence k-thin graph with respect to the given partition. This result is presented in Section 3.1. Regarding the precedence proper k-thin graphs, for the same input, we proved that if k is a fixed value, then it is possible to decide whether G is a precedence proper k-thin graph with respect to the given partition in polynomial time. For variable k, the related recognition problem is NP-complete. These results are presented in Section 3.2. Also, using threshold graphs, we characterize both precedence k-thin and precedence proper k-thin graphs.

Concerning Hamming-Huffman trees, we have presented a restricted case of the problem of building optimal Hamming-Huffman trees. Namely, we introduced the problem of building k-Hamming-Huffman trees (k-HHTs), which are Hamming-Huffman trees in which the symbol leaves are distributed in exactly k distinct levels. For $k \leq 2$, we presented a polynomial time algorithm to solve the problem. We showed that such a case is reduced to the problem of finding an independent set L with a certain size ℓ of a hypercube Q_n such that the cardinality of the neighborhood of L is minimum, over all such independent sets of size ℓ. The latter is a well-studied problem and it has already been solved. For $k \geq 3$, we presented an algorithm to evaluate a lower bound on the cost of such trees when the symbols have a uniform probability of occurrence. Moreover, we proved that, for uniform frequencies, an optimal HHT is always a [5]-HHT and that there exists an optimal HHT which is a [4]-HHT.

Lastly, we have made some experiments to investigate the optimality of uniform [2]-HHTs and to measure the capabilities of compression and error detection of [2]-HHTs. Considering these experiments, we conjectured that there is an optimal uniform HHT in

which the leaves lie on at most three levels. We formalize this conjecture as follows.

Conjecture 1. *Let* Γ *be a set of symbols having the same frequency. There exists an optimal Hamming-Huffman* T *tree associated with* Γ *such that* T *is a* [3]*-HHT.*

Also, we conclude that 2-HHTs are indeed a viable solution to compress text data in real-world situations. In comparison with HTs, its cost is around 25% higher but it provides an excellent error detection rate. For instance, for block messages of size 5000, our experiment showed that the error detection rate is around 99.9994% for 2-HHTs.

Concerning precedence k-thin and precedence proper k-thin, some open questions are highlighted:

- Given a graph G, what is the complexity of computing *pre-thin*(G) and *pre-pthin*(G)?
- Given a graph G and an integer k, what is the complexity of determining if *pre-thin*(G), or *pre-pthin*(G), is at most k?
- How do *pre-thin*(G) and *pre-pthin*(G) relate to *thin*(G) and *pthin*(G), respectively?
- Is it possible to extend the results of this paper to consider other types of orderings (partial orders) and restrictions?

Considering k-HHTs, some questions remain open:

- Conjecture 1 holds?
- The k-HHT problem can be solved in polynomial time, for $k > 2$?
- Is it possible to design a dynamic programming to build, or evaluate the cost, of k-HHTs in general?

References

[1] MANNINO, C., ORIOLO, G., RICCI, F., et al. "The stable set problem and the thinness of a graph", *Operations Research Letters*, v. 35, pp. 1–9, 2007.

[2] BONOMO, F., DE ESTRADA, D. "On the thinness and proper thinness of a graph", *Discrete Applied Mathematics*, v. 261, pp. 78–92, 2019.

[3] BONOMO, F., MATTIA, S., ORIOLO, G. "Bounded coloring of co-comparability graphs and the pickup and delivery tour combination problem", *Theoretical Computer Science*, v. 412, n. 45, pp. 6261–6268, 2011.

[4] TROTTER, W. T., HARARY, F. "On double and multiple interval graphs", *Journal of Graph Theory*, v. 3, pp. 205–211, 1979.

[5] KUMAR, N., DEO, N. "Multidimensional interval graphs", *Congressus Numerantium*, v. 102, pp. 45–56, 1994.

[6] GYÁRFÁS, A., WEST, D. "Multitrack interval graphs", *Congressus Numerantium*, v. 109, pp. 109–116, 1995.

[7] WEST, D. B., SHMOYS, D. B. "Recognizing graphs with fixed interval number is NP-complete", *Discrete Applied Mathematics*, v. 8, n. 3, pp. 295–305, 1984.

[8] JIANG, M. "Recognizing d-Interval Graphs and d-Track Interval Graphs", *Algorithmica*, v. 66, n. 3, pp. 541–563, 2013.

[9] A. HUFFMAN, D. "A method for the construction of minimum redundancy codes", *Proceedings of the IRE*, v. 40, pp. 1098–1101, 1951.

[10] HAMMING, R. W. *Coding and Information Theory*. Prentice-Hall, 1986.

[11] FARIA, L., OLIVEIRA, F. S., PINTO, P. E. D., et al. "On the minimum neighborhood of independent sets in the n-cube", *Matemática Contemporanêa*, v. 44, pp. 1 – 10, 2016.

[12] KNUTH, D. *The Art of Computer Programming*, v. 1. Reading, MA, Addison–Wesley, 1968.

[13] OLARIU, S. "An optimal greedy heuristic to color interval graphs", *Information Processing Letters*, v. 37, pp. 21–25, 1991.

[14] ROBERTS, F. S. "Indifference graphs". In: Harary, F. (Ed.), *Proof Techniques in Graph Theory*, Academic Press, pp. 139–146, New York, 1969.

[15] BOOTH, K. S., LUEKER, G. S. "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms", *Journal of Computer and System Sciences*, v. 13, n. 3, pp. 335–379, 1976.

[16] BRANDWEIN, E., SANSONE, A. *On the thinness of trees and other graph classes*. Tese de Mestrado, Universidad de Buenos Aires, Buenos Aires, Argentina, 2022.

[17] BONOMO-BRABERMAN, F., GONZALEZ, C. L., OLIVEIRA, F. S., et al. "Thinness of product graphs", *DISCRETE APPLIED MATHEMATICS*, v. 312, pp. 52–71, 2022.

[18] PESSOA, A. "An approximation algorithm for constructing error detecting prefix codes", *Optimization Online*, 2006.

[19] PESSOA, A. "A note on the construction of error detecting/correcting prefix codes", *Information Processing Letters*, v. 107, pp. 34–38, 2008.

[20] PINTO, P. E. D., PROTTI, F., SZWARCFITER, J. L. "A Huffman-based error detecting code", *Proc. of the Experimental and Efficient Algorithms (WEA'2004), Lecture Notes in Computer Science*, v. 3059, n. 6, pp. 446–457, 2004.

[21] PINTO, P. E. D., PROTTI, F., SZWARCFITER, J. L. "Parity Codes", *Rairo - Theoretical Informatics and Applications*, v. 39, pp. 263 – 278, 2005.

[22] PINTO, P. E. D., PROTTI, F., SZWARCFITER, J. L. "Exact and approximation algorithms for error-detecting even codes", *Theoretical Computer Science*, v. 440-441, pp. 60 – 72, 2012.

[23] WENISCH, T., SWASZEK, P. F., UHT, A. K. "Combined error correction and compression codes", *IEEE International Symposium on Information Theorys*, p. 238, 2001.

[24] SCHAEFER, T. J. "The Complexity of Satisfiability Problems". In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pp. 216–226. ACM, 1978.

[25] KRUSKAL, J. B. "The Number of Simplices in a Complex". In: Bellman, R. (Ed.), *Mathematical Optimization Techniques*, University of California Press, pp. 251–278, 1963.

[26] KATONA, G. "The Hamming sphere has minimum boundary", *Studia Scientiarum Mathematicarum Hungarica*, v. 10, pp. 131–140, 1977.

[27] KÖRNER, J., WEI, V. K. "Addendum to "odd and even Hamming spheres also have minimum boundary"", *Discrete Mathematics*, v. 62, pp. 105–106, 1986.

[28] KÖRNER, J., WEI, V. K. "Odd and even Hamming spheres also have minimum boundary", *Discrete Mathematics*, v. 51, pp. 147–165, 1984.

[29] KNUTH, D. E. *Art of Computer Programming, Volume 1.* Addison-Wesley, 2005.

[30] `https://github.com/moysessj/restricted-hamming-huffman-trees.git`. Accessed: 2022-05-17.

[31] ZIPF, G. K. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, 1949.

[32] PINTO, P. E. D., PROTTI, F., SZWARCFITER, J. L. "Exact and Experimental Algorithms for a Huffman-Based Error Detecting Code." *Lecture Notes in Computer Science*, v. 5532, pp. 311 – 324, 2009.