SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

The class of distance-hereditary graphs, the Hamiltonian problems and a linear time algorithm

av

Mattias Timonen

2013 - No 15

The class of distance-hereditary graphs, the Hamiltonian problems and a linear time algorithm

Mattias Timonen

Självständigt arbete i matematik 30 högskolepoäng, Grundnivå

Handledare: Jörgen Backelin

2013

Abstract

This paper deals with graphs and graph algortihms for solving the Hamiltonian Problems. We consider the class of distance-herditary graphs, for which there exist linear time algorithms for determining whether a given distance-hereditary graph is Hamiltonian or not. We present such an algorithm. We also give a new idea on the existence of Hamiltonian cycle in graphs that are not trivially non-hamiltonian. We give an algorithm, based on this new idea, for solving the Hamiltonian Problem on graphs with maximum degree gerater than one. Each algorithm is exemplified.

Sammanfattning

Denna uppsats behandlar grafer och grafalgoritmer som löser hamiltonproblemen. Vi beaktar klassen av distans-hereditära grafer, för vilka det finns linjärtidsalgortimer som avgör om en given distans-hereditär graf är hamiltonsk eller inte. Vi presenterar en sådan algoritm. Vi ger också en ny infallsvinkel på existensen av hamiltoncykler i grafer som inte är trivialt icke-hamiltonska. Vi visar en algoritm som baseras på denna nya infallsvinkel som löser hamiltonproblemen på grafer vars maximala nodgrad överstiger ett. Varje algoritm exemplifieras.

Acknowledgments

I would like to thank my supervisor Jörgen Backelin for his dedication throughout the process of writing this paper.

CONTENTS

A persistent theme in graph theory has been a desire to determine, in some reasonable sense, which graphs have Hamiltonian circuits and which have not, i.e, we want necessary and sufficient conditions for a graph to have a Hamiltonian circuit. Of course, such necessary and sufficient conditions must be of a psychologically satisfactory kind, and we should not, for example, want a theorem which merely said, perhaps in a slightly disguised form, that a graph has a Hamiltonian circuit if and only if it has a Hamiltonian circuit. ... Crispin Nash-Williams (1975)

1. INTRODUCTION

The Hamiltonian problems include the Hamiltonian path, Hamiltonian cycle, the 1HP and 2HP problems. The first two are to determine whether there exist a Hamiltonian path respectively cycle in a given graph, and find it if it exists. The 1HP respectively 2HP is to determine whether there exists a Hamiltonian path starting at a vertex v, respectively starting at a vertex u and ending at a vertex w and find such paths if they exist. These problems are well known to be NP-complete for general graphs. Thus, for general graphs there exists no known deterministic algorithms that run in linear time for solving any of these problems. This fact have inspired mathematicians and computer scientists to study special classes of graphs, and develop algorithms applicable on them (Hung and Chang for example that are referred to in this paper). Some of them do indeed solve the Hamiltonian problem in linear time. The author of this paper was inspired to study Hamiltonian graphs in order to find out what (not trivially) distinguishes them from others. This led to the contents of section 10.

The aim of this paper is to gain some insights as to having a graph problem and design an algorithm that solves that problem. We shall present a linear-time algorithm for the Hamiltonian problems on distance-hereditary graphs and its theoretical foundation. These insights should be of help as we also aim to uniquely characterize Hamiltonian graphs and describe an algorithm based on this characterization, or theoretical foundation if one wants. We show this algorithm's being correct for one instance of the Hamiltonian cycle problem.

Given a distance-hereditary graph and the task to find out whether it has a Hamiltonian path or cycle, there are essentially three stages one has to work through. The first stage is to determine a pruning sequence, which is found by a linear time algorithm. Secondly one needs the so called decomposition tree, and we shall see how it is constructed with the aid of the pruning sequence, also in linear time. The decomposition tree is finally used in a recursive program to find the numbers $\kappa_1(G), \kappa_2(G), f(G_L), f(G_R)$ in linear time for the distance-hereditary graph at hand. That is the last stage, and once those numbers are computed for a distance-hereditary graph one easily (in constant time, actually) can determine whether it is Hamiltonian or not. Each stage in the procedure, and each step, will be explained and performed on a distance-hereditary graph.

Moreover, we are going to see how distance-hereditary graphs are defined, what characterizes them and also how they can be constructed in two somewhat different ways. One, the older definition, is based on adding one new vertex at the time using *one-vertex-extensions*, and in the new recursive definition, one forms a distance-hereditary graph from two pairs. Each such pair is a distance-hereditary graph together with its twin-set. It is of interest to see the connection, or the leap, between the old, original way if one wants, and the new way to construct a distance-hereditary graph. This connection is essentially the so called one-vertex-extension tree, $ET(G)$, and the information it provides about the structure of distance-hereditary graphs.

The structure of this paper is such that in section 2 we state assumptions and recall basic definitions from graph theory, whilst some specialties will be defined in addition to their appearance in the text. The idea is that, when reading, these not so standard concepts will be close at hand. Section 3 deals briefly with concepts from computer science although this paper is a mathematical one. Therefore we shall remind ourselves about some basic concepts from that area such as: ”problem”, ”algorithm” and ”complexity”. In section 4 we deal thoroughly with the class of distance-hereditary graphs. First they will be defined of course, then we present the ”old” way of constructing distance hereditary graphs using *one-vertex-extensions*, followed by a few theorems and proofs. In section 5 we define cographs. Section 6 is devoted to the algorithm that is used to determine whether a graph is distance-hereditary or not. We present a distance-hereditary graph recognition algorithm. In Section 7 we present the “new” definition of distance-hereditary graphs by presenting the concept of *one-vertex-extension ordering*, on which both the above mentioned *pruning sequence* and the *one-vertex-extension tree* relies. The one-vertex-extension tree is the foundation of the concept of

twin-set which is an essential part of the new definition of distance hereditary graphs. We give proofs of a number of lemmas dealing with the one-vertex extension tree. Having done that we are ready for the new recursive definition of distance hereditary graphs. We then present the *decomposition tree*, used in the program presented in next section. Section 8 is devoted to the lemmas and theorems that leads to the program for finding $\kappa_1(G), \kappa_2(G)$ and $f(G)$ in order to determine whether the distance-hereditary graph at hand has a Hamiltonian path or not. Section 9 deals briefly with the extension of the solution of the Hamiltonian path problem to 1HP, 2HP and the Hamiltonian cycle problems. In Section 10 we define a way to partition graphs, and present a necessary and sufficient condition for Hamiltonicity in general graphs based on that partition. It is followed by a few corollaries. We present an algorithm (non-linear though) for solving the Hamiltonian Cycle problem on general graphs based on the N&S-condition.

2. GRAPH THEORETICAL BASICS

2.1. Definitions and Notation

We assume that the reader is familiar with the concept of graphs in terms of vertices (or nodes) and edges. All graphs in this text are non-empty, simple and finite. Below will be stated a few necessary graph theoretical concepts that will be used in this text, or needed to explain others.

$G = (V, E)$: denotes a graph with vertex set V and edge set E. Vertices are denoted by small letters u, v or v_1, v_2 ..... An edge between two, thus adjacent, vertices u and v is denoted by uv. The size or cardinality of a graph,$|G|$, is equal to number of vertices in G.
$E(A, B)$: a set of edges between vertices in A and vertices in B, where A and B are sets of vertices.
Chord: a *chord* is an edge that joins two vertices of a cycle (or a path) but is not an edge of the cycle (or the path) itself.
Clique: a *clique* of a graph G, is its maximal complete subgraph of G.
Closed neighborhood: the *closed neighborhood*, $N[x]$, of a vertex v, is the neigbourhood of v and *v* itself.
Cograph: *complement-reducible graph* See section 5
Complete graph: a graph in which all vertices are pairwise adjacent is called a *complete* graph and the complete graph on n vertices is denoted by K_n.
Connected graph: a graph is said to be *connected* if there is a path between any two vertices in G.
Cycle: if $v_0 = v_k$ in a path $v_0 e_0 v_1 e_1 \dots e_{k-1} v_k$,then the path is a *cycle*. A *chord-free cycle* is an induced cycle in a graph. We often denote a cycle by its vertex sequence and write $C = v_0 v_1 \dots v_{k-1}$
Diameter: the greatest distance between two vertices in G is called *the diameter* of G.
Distance: the *distance* $d(u, v)$ of two vertices in G is the length of a shortest path from u to v in G. The distance between two vertices in different components of a disconnected graph is infinite.
Distance layout: all vertices v_i, in G, such that $d(u, v_i) = d$.
Finite graph: if the number of vertices in a graph G is finite, then we say that the *graph is finite*. Else it is infinite.
Induced subgraph: if $G' \subseteq G$ and G' contains every edge uv of *E* with $u, v \in V'$, then G' is an induced subgraph of G. We let $G[V']$ denote the *induced subgraph*.
Inner face: a bounded region in a plane graph, see Diestel for technicalities.
Join of graphs: the join $G = G' + G''$ of two graphs G' and G'' with disjoint vertex sets V' and V'' respectively and edge sets E' and E'' respectively is the union of the G' and G'' together with all edges joining V' and V''.
Length of a cycle: the *length* of a cycle is the number of edges or the number of vertices as they are the same.
Length of a path: the *length*, $l(u, v)$ of a path $v_0 e_0 v_1 e_1 \dots e_{k-1} v_k$ is k, i.e. the number of edges.
Long cycle: if the length of a cycle is at least 5, then we say that it is a *long cycle*.
Neighborhood: the *neighborhood*, $N(v)$, of a vertex is the set of vertices that are adjacent to the vertex v.
Parent(v): if, in a rooted tree with root v_0 and an edge uv, it holds that $d(v_0, v) = d(v_0, u) + 1$ then we say that u is the *parent* of v in T and we denote it $p(v)$. If $d(v_0, v) \geq d(v_0, u) + 1$ in T we say that u is an *ancestor* of v in T.
Path: a *path* is a non-empty graph $P = (V, E)$ of the form $V = \{v_0, v_1, \dots, v_k\}$, $E = \{v_0 v_1, v_1 v_2, \dots, v_{k-1} v_k\}$. We can let $e_i = v_i v_{i+1}$ and thus write $E = \{e_0, e_1, \dots, e_{k-2}, e_{k-1}\}$. In a path all v_i are distinct, all e_i are distinct and $v_0$and v_k are called end vertices. A path is usually denoted by P. We can write $P = v_0 e_0 v_1 e_1 \dots e_{k-1} v_k$ or $P = v_0 v_1, \dots, v_k$, $P = v_0 - v_1 - \dots - v_k$ whence the edges are implicit in the latter cases. Also, we can denote a path with start vertex $v_0$and end vertex v_k in an intuitive manner by $v_0 P v_k$ or $P_{v_0 v_k}$.

Path cover: a *path cover* of a graph G is a set of disjoint paths such that they together contain all the vertices of G.

P_4: a path on four vertices, i.e. of length 3.

Plane graph: a graph drawn in the plane such that no edges cross. For technicalities, see Diestel.

Root: sometimes it is convenient to consider one vertex of a tree as special; such a vertex is then called the *root* of this tree. A tree with a root is a *rooted tree*. Choosing a root v_0 in a tree imposes a partial ordering on $V(T)$ by letting $r \leq h$, if v_r is on a path $v_0 e_0 v_1 e_1 \ldots e_{h-1} v_h$ of the tree.

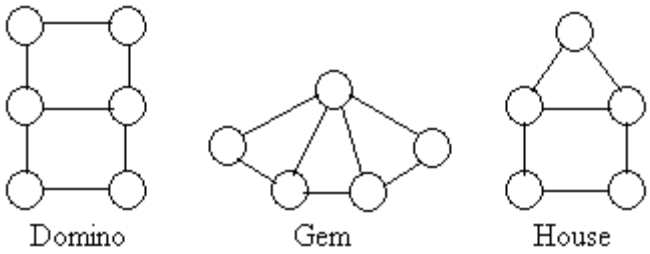
Subgraph: a *subgraph* $G' = (V', E')$, denoted $G' \subseteq G$, is a graph for which $V' \subseteq V$ and $E' \subseteq E$.

Stable Set: a *stable set* of vertices is a set of vertices that are pairwise non-adjacent.

Tree: a connected graph that has no cycles is called a *tree* and is usually, and here, denoted T.

Union of graphs: the union $G = G' \cup G''$ of two graphs G' and G'' with disjoint vertex sets V' and V'' respectively and edge sets E' and E'' respectively is the graph G with vertex set $V' \cup V''$and edge set $E' \cup E''$

Domino, Gem and House: Three forbidden subgraphs in a distance-hereditary graph.



3. PROBLEMS, ALGORITHMS AND COMPLEXITY

3.1. PROBLEMS

3.1.1. Problems, Decision Problem and Solution to the Problem

A decision problem in the sense that is used here has nothing to do with human ambivalence before choosing "between a rock and a hard place", but has quite a precise meaning which we will learn just after we have learned what a problem is, (Golumbic p.23):

> A problem consists of two things: a question to be answered, and a solution to be found.
> If the question has a "yes" or "no" answer then the problem is a *decision problem.*
> The solution can be a requirement to be fulfilled, a best possible situation or a structure to be found.

A decision problem may be: "can the vertices of a graph be ordered $(v_0, \dots, v_n)$ so that v_i and v_{i+1} are adjacent in G for $1 \leq i \leq n-1$?", or it may be this: "can the vertices of a graph be ordered in some way, $(v_0, \dots, v_n)$, so that for $1 \leq i \leq n-1$we have that v_i and v_{i+1} are adjacent, and we also have that v_n and v_0 are adjacent, in G?". Those questions are seeking the yes- or no-answer to whether there is a particular structure, namely a Hamiltonian path respectively a Hamiltonian cycle in a graph G and hence the Hamiltonian Path- and Hamiltonian Cycle-problems are decision problems.

3.1.2. Instance of a Problem

Suppose we have a problem, the Hamiltonian Cycle Problem say, a graph at hand and some knowledge about it, the adjacency matrix for example, and we will be satisfied when we have a correct answer to whether our graph has a Hamiltonian Cycle or not, then we have an *instance of a problem*. In more general and slightly formal terms one can say (Gould p. 15):

> A graph problem can be posed in terms of a number of parameters. When specifying the parameters in addition to the description of the problem and what is meant by a solution, then one have an *instance* of a problem. The description may typically be in the form of the graph structure: adjacency matrix, degree sequence.

In this paper an instance of a problem will for example be a given distance-hereditary graph together with a particular tree representation, namely a decomposition tree $DT(G)$: *given this distance-hereditary graph and $DT(G)$, does it have a Hamiltonian path?*

3.2. ALGORITHMS AND COMPLEXITY

3.2.1. Algorithms

What is an algorithm? In a sufficiently deep sense it can be thought of as an instruction, and in particular an instruction to a computer on how to compute the solution to a problem. There are further concepts associated with algotihms such as *state* and *determinism* (Golumbic pp.23,26): Take into consideration the current values of all variables and the location of the instruction to be executed, that is the *state* of the algorithm. If for each state, execution of the instruction determines at most one new state, then the algorithm is *deterministic*. It explores the alternatives one at the time so to say. *Nondeterministic* algorithms may allow several new states and for each new state it branches off and explores all alternatives simultaneously.

Later on in this paper we will state a number of algorithms and present them in pseudocode, which is a way of expressing the computer logic in almost, or bitwise, ordinary language.

3.2.2. Complexity and Efficiency

The efficiency of algorithms is measured in *space complexity* or *time complexity.* Space complexity is not an issue to the extent that time complexity is, due to the powerful storage capacity of modern computers and computer environments, so we'll leave it at that.

Time complexity however, despite the computational power of computers today, is still a concern. This measure is relative and not measured in time units, and there are a couple of reasons why such an absolute measure would be impractical, (Gould p.16): computational speed of computers differ and programmers vary in skills. This may effect the time in seconds or minutes it takes for the algorithm to be performed. Therefore it is of higher interest to study the relative time it takes to carry out an algorithm, more specificly in terms of the number of elementary computational steps involved in the algorithm. Taking into consideration the worst-case scenario one gets the upper bound for the complexity of the algorithm.

How do we express this relative time measure then? A function f, usually of the size of the problem (the number of vertices for example), expresses the complexity of the algorithm $\mathcal{A}$. One says that the function f runs in time $O(f(n))$, if there exists an implementation of $\mathcal{A}$ such that for some constant $c > 0$, $\mathcal{A}$ terminates after at most $cf(m)$ computational steps for all instances of size m.

3.2.3. The Classes P and NP

A decision problem is in the class P, if there exists a deterministic algorithm that solves the problem in polynomial time. Herbert S Wilf states this (Wilf p.106):

> *We say that a decision problem belongs to the class P if there is an algorithm A and a number c such that for every instance I of the problem the algorithm A will produce a solution in time $O(B^c)$, where B is the number of bits in the input string that represents I.*
>
> *To put it more briefly, P is the set of easy decision problems.*

For each of the problems in that class there is a fast (polynomial time) algorithm that provides a solution.

A decision problem is in the class NP if an answer that is claimed to be correct is easy to check. Also if a nondeterministic algorithm solves a problem in polynomial time, then the problem is in the class NP. If, in each state of a non-deterministic algorithm, the ”correct” choice is made for the next sate, then computing time is polynomial. If instead all possible choices are to be made, then a nondeterministic algorithm becomes deterministic and requires an exponential number of elementary computational steps.

Problems in the class NP can be classified according to reducibility, a concept that can be described by the following (similar to “the teakettle principle”): if one is given a pen, a rubber and a blank sheet of paper and is asked to draw a graph on ten vertices with no edges on the paper, then one makes ten dots on the paper. If one is given a sheet of paper with five dots and is asked to do the same thing one is tempted to draw five dots on the paper. But then one would have solved another problem, so instead one can reduce the problem to the first one by erasing the five given dots and draw ten new ones.

Consider two problems S and T. T is *quickly reducible* to S, if for every instance ”t” of T we can convert it in $p(m)$-time (p is a polynomial) to ”s”, an instance of S in such a way that they have the same answer - ”yes” or ”no”.

A decision problem is *NP-hard* if all problems in the class NP are quickly reducible to it. An NP-hard problem in NP is NP-complete. For NP -complete problems only exponential time algorithms are known. The best known lower bounds are polynomial functions, though. If a polynomial time algorithm should be proved to exist for one of them, then there exists one for each of them.

4. DISTANCE-HEREDITARY GRAPHS

4.1. DEFINITION

A description of distance hereditary graphs is the following (Bandelt and Mulder p.182):

> a *distance-hereditary graph* is a connected graph in which every induced path is isometric. That is, the distance of any two vertices in an induced path equals their distance in the graph. So any connected induced subgraph of a distance hereditary graph inherits its distance function from G, where the distance function gives the length of a shortest path between two vertices.

Rephrasing it even more intuitively one can say that the distance between any two vertices x and y in G, is the same in any induced subgraph that contains x and y.

Distance-hereditary graphs are also known under the name *completely separable graphs.* In this paper we will only refer to this class of graphs as distance-hereditary graphs.

The definition is theoretically appealing in its simplicity. In practice however, it may be difficult and tedious to check the condition using the definition on other than small graphs. For instance checking that a path on a small number of vertices is distance hereditary is easy, but as the number of vertices grows and cycles appears, the checking rapidly becomes tedious. In this paper we will discuss recursive methods to construct distance hereditary graphs and algorithmic methods to determine whether a given graph is distance hereditary or not.

4.2. CONSTRUCTION OF A DISTANCE-HEREDITARY GRAPH

4.2.1. One-vertex extensions

We shall now present the concept of one-vertex extension. It is pretty much what it sound like; extending a graph by adding one vertex at the time, although following some particular rules while doing so. Consider the induced subgraph $G' = (V', E')$ of $G = (V' \cup \{x\}, E)$, and a vertex x' in G'. If G' were extended to G by adding a new vertex x such that:

- $i)$ $N(x) = \{x'\}$, we say that we were attaching a pendant vertex x to x' and denote the extension by $x(P)x'$
- $ii)$ $N(x) = N(x')$, we say that x' and x are false twins, and denote the extension by $x(F)x'$
- $iii)$ $N[x] = N[x']$, we say that x' and x are true twins, and denote the extension by $x(T)x'$

The last two operations are also referred to as splitting vertices (Bandelt and Mulder) among other names, but in this paper they will be referred to as false-twin operation respectively true-twin operation, whilst the first one will be referred to as a pendant operation. Any of the operations above can of course act on any kind of graph, but they are of particular interest in the theory of distance hereditary graphs. It is so because those three one-vertex extensions are such that, using them and only them as we shall see in theorem 1 below, all finite distance-hereditary graphs can be obtained. See also *Appendix 1* for examples of each operation.

4.3 MORE ABOUT DISTANCE-HEREDITARY GRAPHS

Below we present a few theorems and proofs of some, following the authors giving them. The first theorem, theorem 1 given and proved by Bandelt and Mulder, relates a graph's being distance-hereditary to the one-vertex-extensions. We shall briefly present the proof of this theorem. In

theorem 2 given by Chang, Hsieh and Chen, will be stated that there is an order in which the extensions are employed and that this ordering also will generate a distance-hereditary graph. Theorem 3, given by Hammer and Maffray, gives a number of characteristics of distance-hereditary graphs, and states that these properties are equivalent to G's being a distance-hereditary graph. Hammer and Maffray does not use the term "distance-hereditary graph", instead they talk about "completely separable graphs", a notion based on Boolean Functions. Note that statement (d) of theorem 3 is proved in the establishment of the "only if" part of theorem 1 in which is shown that every distance-hereditary graph G contains a pair of twins or *two* pendant vertices (and hence a [one] pendant vertex). This is a connection between the two theorems. This also proves theorem 2. The proof of theorem 3 is partially given here, and otherwise given by Hammer and Maffray. Finally, theorem 5, preceded by theorem 4 and a lemma, all of which are given and proved by Bandelt and Mulder, gives a few conditions that are necessary and sufficient for a graph's being distance-hereditary. This theorem is also fundamental to the linear time algorithm that is used to determine whether a given graph is distance-hereditary or not. The theorem is followed by a presentation of its proof.

4.3.1. Theorems 1, 2 and 3

Theorem 1 (Bandelt and Mulder p.188):

A connected graph $G, |G| \geq 2$, is a distance-hereditary graph if and only if it is obtained from K_2 by a sequence of one-vertex-extensions: pendant, false-twin and true-twin operations, where K_2 is the complete graph on two vertices.

Remark: a graph $G, |G| \geq 2$, is distance-hereditary but not connected if it is obtained from K_1 by a sequence of one-vertex-extensions starting with a false-twin operation. To see this consider $N[v_1]$ and $N[v_2]$ when adding v_2 to $G = \{v_1\}$ by a false-twin operation.

Theorem 2 (Chang et al. p345):

A graph G is a distance-hereditary graph if and only if it has a one-vertex-extension-ordering.

Remark: A one-vertex-extension ordering is a set of nodes together with an ordering that constitutes the order in which the vertices are attached to the graph. See section 7.

Theorem 3. Based on Hammer and Maffray theorem 4.2:

The following five properties are equivalent

(a) G is a distance hereditary graph

(b) G contains none of house, gem, domino, chordless cycle of length ≥ 5 (long cycle) as an induced subgraph

(c) Every long cycle of G has at least two crossing chords

(d) Every induced subgraph of G has a pair of twins or a pendant vertex

(e) Given any two vertices u and v of G, all chordless paths from u to v have the same length

4.3.2. Proof of Theorem 1

In the following, we refer to Bandelt and Mulder p188-191. The proof of the if-part is rather short so we present it below. When it comes to the only-if-part, there are a number of cases and subcases. The proof in its whole is quite long, so we will only briefly present the conclusion in each case and we refer to Bandelt and Mulder for details.

The "if" part of the theorem, that is: if G is obtained from K_2 by a series of one-vertex-extensions, then G is distance hereditary is proved by induction on the number of vertices in the following way (Bandelt and Mulder p.188):

Let G be a distance-hereditary graph with at least two vertices. Any graph obtained from G by

attaching a pendant vertex is, evidently, again a distance hereditary graph. Let G' be obtained from G by adding a twin x' to a vertex x of G. Any induced path in G' containing at most one of x and x' is isometric by induction hypothesis. If both x and x' are in an induced path P of G', then either P has length 1 (if x and x' are adjacent, i.e. if x' is a true twin to x) or P has length 2 (if x' is a false twin to x). In either case P, trivially, is isometric. Therefore, G' is distance hereditary.

The "only if" part, that is, if G is distance hereditary then G is obtained from K_2 by a series of one-vertex-extensions, is proved by starting with the following assertion: let G be a distance hereditary graph with $n \geq 3$ vertices. Assume that every distance hereditary graph with at least two and at most $n-1$ vertices contains either a pair of twins or two pendant vertices (Bandelt and Mulder p.188). To prove that G contains either a pair of twins or two pendant vertices the authors distinguish a number of cases. We present the assertion of each case together with a very brief description of the proof.

$Case$ 1(Bandelt and Mulder p.188):

G contains at least two pendant vertices which is the trivial case and there is nothing to prove.

$Case$ 2 (Bandelt and Mulder p.188):

G contains exactly one pendant vertex z′, which is attached to the vertex z, say.

To prove this they consider the vertex-deleted subgraph $G - z'$, which contains *one* pendant vertex or none. In either case, by the induction hypothesis G contains a pair of twins.

$Case$ 3 (Bandelt and Mulder p.188):

G has no pendant vertices, but for some vertex z of G, the vertex-deleted subgraph $G - z$ has at least two pendant vertices. Let u', v' be two pedant vertices of $G-z$; say, u' is attached to u and v' to v. They distinguish two subcases.

$Case$ 3.1(Bandelt and Mulder p.188):

The vertices u' and v' belong to the same component of $G - z$.

They show that the two pendant vertices must be attached to the same vertex, i.e that $u = v$, in $G-z$, by showing that the converse implies one of the forbidden subgraphs in $G-z$, where $G-z$ is distance-hereditary by induction hypothesis. They conclude that $u = v$ and that this implies that u', v' is a pair of twins in G.

$Case$ 3.2 (Bandelt and Mulder p.189):

All pendant vertices of $G-z$ belong to different components of $G-z$.

To prove that $G-z$ contains at least two pendant vertices, Bandelt andMulder argues that z is a cut-vertex and hence the removal splits G into disconnected subgraphs. They further argue that the component H containing u' has at least two vertices, is a distance-hereditary graph containing exactly one pendant vertex and hence by hypothesis having a pair of twins x, x'. They settle the case by showing a number of outcomes depending on the adjacency relation between z and x, x'.

$Case$ 4 (Bandelt and Mulder pp.189-191):

G has no pendant vertices, and for every vertex z of G, the subgraph $G-z$ contains at most one pendant vertex.

$G-z$ contains by hypothesis a pair of twins and one want to show that this implies that so do G. Bandelt and Mulder argues that the pair of twins in $G-z$ cannot involve a pendant vertex and that the pendant vertex cannot be attached to any of the vertices in the pair of twins. Then by way of contradiction, they assume that G does *not* contain a pair of twins, and argue that z is thus adjacent to exactly one vertex of a pair of twins x, x' in $G-z$.

They let z be a vertex in G and have maximum degree and they let u, u' be a pair of twins in $G-z$ such that z is adjacent to u' but not to u. The following argument of case 4 is subdivided into two

main subcases, one where the pair of twins u, u' in $G-z$ are adjacent and one where they are not. In either way, Bandelt and Mulder show that for each subcase this will lead to a contradiction either by violating the maximality assumption on z, the existence of forbidden subgraphs (see theorem 3.b) or the existence of troublesome pairs of twins. The strategy is to investigate adjacency relations between z and pairs of twins, adjacency relations between couples of pairs of twins and the interrelations between all three entities. For details see Bandelt and Mulder pages 189-191)

In this manner Bandelt and Mulder have proved the existence of either a pair of twins or two pendant vertices in any distance-hereditary graph on at least two vertices. They conclude that (Bandelt Mulder p.191):

> we can decompose G according to the one-vertex-extensions until we finally arrive at the graph K_2

and thus they have settled the proof of theorem 1.

4.3.3. Proof of Theorem 3

The theory behind theorem 4.2 in Hammer Maffray is based on Boolean functions and a property thereof called separablity (Hammer and Maffray p.85). That is outside the scope of this paper, therefore we give proofs in terms of graph theoretical concepts where the authors do not. First we give a proof of that property **(e)** is equivalent to the graph G's being distance hereditary, i.e. that **(a)** is equivalent to **(e)**. Using Theorem 1 we argue that **(a)** is equivalent to **(d)**. Then we give an outline for the proof of the implication **(a)⇒(b)**. Hammer and Maffray show that **(b)⇒(c)** and finally, we give a proof of the implication **(c)⇒(e)**, and we are done.

(a)⇔(e) (Backelin and Timonen): First we prove the implication **(a)⇒(e)**. Assume that G is a distance-hereditary graph, that $P = \{u, \dots, v\}$ is a chord-free path in G and that P has length $d_P(u, v)$. P is an induced subgraph of G. Since G is distance-hereditary we have that $d_P(u, v) = d_G(u, v)$, and in every induced subgraph of G the distance between u and v is the same, and therefore all chord-free paths from u to v has the same length.
To prove that **(e)⇒(a)**, we assume that G is a connected graph, and H is a connected subgraph of G. We also assume that $l_G(u, v)$ is the length of any shortest path from u to v in G. P is chord-free (else there would be a shorter path). Finally we assume that $l_H(u, v)$ is the length of a shortest path from u to v in H. This path is chord-free for the same reason. Thus we have that the distance between any two vertices in G, is the same in any induced connected subgraph H, and hence G is distance-hereditary by definition.

(a)⇔(d): The equivalence follows from Theorem 1 and its proof.

(a)⇒(b): to show that a distance hereditary graph does not contain any of the forbidden subgraphs as an induced subgraph is a fairly straight forward exercise done by considering the subgraph $H - x$, where H is one of the house, domino, gem or long cycle, and where x is a vertex of highest degree. By removing x in the house, gem or domino it is obvious that the paths of length two on three vertices, $P = \{v, x, w\}$ with x not on any endpoint, the remaining paths from v to w in H is not of length two.
If H is the long cycle, we have that if $P = \{v, x, w\}$ as above, the removal of x will result in a path of length $n - 1 \geq 3$. Thus the distance between v and w in $H - x \geq 2$.
Thus the removal of x from H as explained above results in a change of distances between two vertices in the subgraph $H - x$, and hence they cannot exist as induced subgraphs of a distance hereditary graph.

(b)⇒(c), we follow Hammer and Maffray who use induction on k, where k is the length of a long cycle C of G, (Hammer and Maffray p.89):

If $k = 5$, then C must have two crossing chords, otherwise its vertices would induce a pentagon, a house or a gem. If $k = 6$, either the vertices of C induce a hexagon or a domino, which are forbidden, or C has two non-triangular chords and they necessarily cross each other, or it contains a cycle of length 5 and thus two crossing chords by the induction hypothesis. If $k \geq 7$, the cycle C must have a chord since G contains no long chordless cycle. This chord divides C in two shorter cycles, one of which has length at least 5. By the induction hypothesis this subcycle of C has two crossing chords, which in turn are crossing chords of C.

(c)⇒(e) We give a proof for the implication (Backelin and Timonen):
The distance between the vertices u and v in a graph is, by definition, the minimum length of a path from u to v: $d(u,v) = min\{l(u,v)\}$. $d(u,v)$ is also equal to the minimum length of a chord-free path between u and v.
We want to show that for any chord-free path between u and v $l(u,v) = d(u,v)$, and we will do so by induction on $d(u,v)$.
The statement is true for $d(u,v) \leq 1$, that is for graphs on identical or adjacent vertices.
We assume, for some $k \geq 1$, that if $d(u,v) \leq k$, it holds that every chord free path between u and v has length $d(u,v)$. Now we want to show that this holds when $d(u,v) = k+1$.

Let $P = u\text{–}x_1 - \ ... - x_k\text{–}v$ be a shortest path between u and v, with distance $d_P(u,v) = k+1$, and let $Q = u\text{–}y_1 - \ ... - y_n\text{–}v$ be any chord free path between u and v with distance $d_Q(u,v) = n+1$.

We want to show that $n = k$. We get two cases:

$Case\ 1$. There exists a vertex $w = x_i = y_j$. We have that $d_P(u,v) = i \leq k$. By hypothesis $d_Q(u,v) = i$. Also $d_P(w,u) \leq k$ and by hypothesis $d_P(w,u) = d_Q(u,v) \leq k$. Hence $n = k$.

$Case\ 2$. There is no vertex $w = x_i = y_j$, i.e.
$P\{u,v\} \cap Q\{u,v\} = \{x_1 - \ ... - x_k\} \cap \{y_1 - \ ... - y_n\} = \emptyset$.
$2 \leq k+1 \leq n, u\text{–}x_1 - \cdots - x_n\text{–}v\text{–}\ y_k - \cdots - y_1\text{–}u$ is a long cycle which we know has two crossing chords.
Let i be the smallest index such that there is a crossing chord $y_i - x_j$.
Claim 1: For x_j it holds that $j \geq 2$.
Proof: Since $y_i - x_j$ is a crossing chord, there is a chord $y_i' - x_j'$ being crossed. Since i is minimal we have that $i < i'$. In order to preserve the crossing condition this leaves us no other possibility than that $j > j'$, thus $j \geq 2$.
Claim 2: The smallest *i* such that there is a crossing chord $y_i - x_j$ is $i = 1$.
Proof: assume $i > 1$. Since $x_j \in \{x_2, \ldots, x_k\}$, $u\text{–}y_1 - \cdots - y_i\text{–}x_j - \cdots - x_1 - u$ is a long cycle and thus has a crossing chord from $y_r, r < i$ which contradicts our assumption. Hence $i = 1$.

Now assume that $y_1 - x_j$ is the last chord from y_1.
Claim 3: The greatest j such that there is a crossing chord $y_1 - x_j$ is $j = 2$.
Proof: $u\text{–}y_1 - x_j$ is chord-free and $d(u, x_j) = 2$. $[u\text{–}x_1 - \cdots - x_j \Rightarrow d_P(u,x_j) = 2] \Leftrightarrow [j = 2]$, since we have by assumption that all chord-free paths from z to z', have the same length as the shortest path from z to z' when $d(z,z') \leq k$.
When $z = y_1$ and $z' = v$ we have that $y_1\text{–}x_2\text{–} \ldots - x_k\text{–}v$ is the shortest path from y_1 to v, and hence has the same length as the chord free path $y_1\text{–}y_2 - \ldots - y_n\text{–}v$. Hence $k = n$.

4.3.4 Theorem 5 with preliminaries

Theorem 5, stated and proved by Bandelt and Mulder, is the foundation for the linear-time algorithm which is used to determine whether a graph is distance-hereditary or not. We shall present the theorem and its proof. Before we do that, we shall present the definition of a few concepts, theorem 4 and a lemma. The proof of theorem 4 is omitted here but is given by the authors.

Definition 4.3.4.2 (Bandelt and Mulder p.184):

The *interval function,* $I(u,v)$, for a graph G and for any pair u, v of vertices in G, is defined, by $I(u,v) = \{x \mid x \text{ is a vertex of } G \text{ on some shortest } (u,v)-\text{path}\}$.

The following theorem is an extension of theorem 3.

Theorem 4. (Bandelt and Mulder p.194):

Let G be a connected graph with distance function d and interval function I. Then the following conditions are equivalent:

- **(a)** G is distance-hereditary
- **(b)** For any two vertices u and v with $d(u,v) = 2$, there is no induced (u,v)-path of length greater than 2.
- **(c)** The gem, the domino, the house and the long cycles are not induced subgraphs of G,
- **(d)** The gem, the domino, the house and the long cycles are not isometric subgraphs of G,
- **(e)** The gem, the domino, the house and the long cycles are not induced (or isometric) subgraphs of G, and $I(u,v) \cap I(v,w) = \{v\} \Longrightarrow d(u,w) \geq d(u,v) + d(v,w) - 1$,
- **(f)** the gem is not an induced subgraph of G, and for any three vertices u, v, w at least two of the following inclusions hold: $I(u,v) \subseteq I(u,w) \cup I(v,w), I(u,w) \subseteq I(u,v) \cup I(v,w), I(v,w) \subseteq I(u,v) \cup I(u,w)$
- **(g)** for any four vertices u, v, w, x at least two of the following distance sums are equal: $d(u,v) + d(w,x)$, $d(u,w) + d(v,x)$, $d(u,x) + d(v,w)$,
- **(h)** G satisfies condition **(g)**, and if in **(g)** the smaller distance sums are equal, then the largest one exceeds the smaller ones by at most 2.

A graph G can be decomposed into *levels* with respect to the distance from a fixed vertex. A level is the set of vertices on a particular distance, k, from a particular vertex u:

Definition 4.3.4.1 (Bandelt and Mulder p.200):

The *k:th level with respect to u,* is defined by

$$N_k(u) = \{x | x \text{ is a vertex of } G \text{ with } d(u,x) = k\}$$

Each such level in a distance-hereditary graph can be viewed as a building stone of such a graph, provided, as we shall see in theorem 5 below, each level contains no paths of length 3 as induced subgraphs.

Definition 4.3.4.3. (Bandelt and Mulder p.200):

The $k{:}\,th$ internal level of any interval, $I(u,v)$, is the intersection of the interval function and the $k{:}\,th$ level, denoted $N_k(u,v)$.

Lemma 4.3.4.4. (Bandelt and Mulder p.200):

Let G be a distance-hereditary graph. Then for any two vertices u and v, and for any integer i with $0 \leq i < d(u,v)$, every vertex in $N_i(u,v)$ is adjacent to all vertices in $N_{i+1}(u,v)$.

Theorem 5. (Bandelt and Mulder p.201):

Let G be a connected graph, and let u be any vertex of G. Then G is distance-hereditary if and

only if G satisfies the following five conditions, for any integer $k \geq 1$:

(a) if v, w are two vertices in the same component of $N_k(u)$, then
$$N(v) \cap N_{k-1}(u) = N(w) \cap N_{k-1}(u) ,$$

(b) there is no induced path of length 3 (i.e no P_4 as an induced subgraph) in $N_k(u)$.

(c) if a vertex v of $N_k(u)$, has neighbors x and y in two distinct components X and Y of $N_{k-1}(u)$, then v is adjacent to all vertices in X and Y, and
$$N(x) \cap N_{k-2}(u) = N(y) \cap N_{k-2}(u) ,$$

(d) if v, w are vertices in different components of $N_k(u)$, then $N(v) \cap N_{k-1}(u)$, and $N(w) \cap N_{k-1}(u)$, are either disjoint, or one of the two sets is contained in the other.

(e) if a vertex v of $N_k(u)$, is adjacent to two vertices x and y in the same component of $N_{k-1}(u)$,, then the vertices of this component which are not adjacent to v are adjacent to either both x and y, or none.

4.3.5 Proof of Theorem 5

The following proof is given by Bandelt and Mulder. They first prove the only if part(Bandelt and Mulder p. 201-203):

Let G be distance-hereditary. To prove **4(a)** it suffices to show that the assertion holds for any two adjacent vertices v, w in $N_k(u)$. From condition (e) of theorem 4 they infer that there is some neighbor x of v and w in $N_{k-1}(u)$. Suppose that there exists a neighbor y of v in $N_{k-1}(u)$ which is not adjacent to w. Then by Lemma 1, one can find a common neighbor z of x and y in $N_{k-2}(u)$. Then v, w, x, y, z induce one of the forbidden subgraphs - house, gem or domino – which is a contradiction. Hence **(a)** is true.

Assume that P is an induced path of length 3 in $N_k(u)$. Then by **(a),** there exists a vertex x in $N_{k-1}(u)$ adjacent to all vertices in P. Hence P and x induce a gem, which is forbidden. This proves **(b).**

For x and y, given as in **(c)** one can find a vertex z in $N_{k-2}(u)$ adjacent to both x and y, by the Lemma. Then x, v, y, z induce a $4 - circuit$ in G. If w is some vertex in $N_{k-1}(u)$ adjacent to one of x and y, then by **(a),** also w is adjacent to z. Since the house may not occur in G, it follows that w and v are adjacent, proving the first part of **(c)**. Finally, note that, by the Lemma, every neighbor of x in $N_{k-2}(u)$ is also adjacent to y. This settles **(c)**.

Next, given u, w as in **(d)**, let x be a common neighbor of v and w in $N_{k-1}(u)$, and let y and z be vertices in $N_{k-1}(u)$ such that y is adjacent to v but not to w, and z is adjacent to w but not to v. Then, by **(a)** and **(c)**, there exists a vertex t in $N_{k-2}(u)$ adjacent to all three verices x, y and z. Then the subgraph induced by $\{t, v, w, x, y, z\}$ contains one of the forbidden subgraphs as an induced subgraph, whence one is done.

Finally, to prove **(d)** let x and y be neighbors of v within the same component of $N_{k-1}(u)$. From **(a)** one knows that there exists a vertex z in $N_{k-2}(u)$ adjacent to all vertices of this component. Let t be a vertex of the latter adjacent to x but not to v. If t and y re not adjacent, then $\{t, v, x, y, z\}$ would induce one of the forbidden subgraphs.

This concludes the ”only if” part.

Conversely, let G satisfy conditions **(a)** through **(e).** Bandelt and Mulder show by induction on k that the subgraph G_k of G induced by all levels $N_i(u)$ with $i \leq k$ does not contain any C_n, $n \geq 5$, or any of the forbidden subgraphs house, gem or domino as an induced subgraph. For $k \leq 1$ this is trivial by **(b)**. So let $k \geq 2$. By hypothesis, one only have to consider configurations touching $N_k(u)$. First they prove two simple facts, **(A)** and **(B)** below, which are used in the sequel.

(A) If $P = v - w - x - y$ is an induced path in G_k such that the internal vertex x is in $N_k(u)$, then v is also in $N_k(u)$, and w, y are in $N_{k-1}(u)$.
Proof: This is seen as follows. By **(b)** at least one vertex of P belongs to $N_{k-1}(u)$. Then by **(a)** there cannot be any edge of P in $N_k(u)$. Hence w and y are in $N_{k-1}(u)$. If v is in $N_{k-2}(u)$, then either **(a)** or **(c)** is violated, depending on whether w and y are in the same component of $N_{k-1}(u)$ or not. If v is in $N_{k-1}(u)$, then either **(e)** or **(c)** is violated.

(B) If $Q = v - w - x - y - z$ is a path in G_k with no chords except possibly vz such that A has an internal vertex in $N_k(u)$, then v, x, z are in $N_k(u)$ and w, y are in $N_{k-1}(u)$.
Proof: By (A), the vertices of Q are alternatively in $N_{k-1}(u)$ and $N_k(u)$. If v, x, z are in $N_{k-1}(u)$, then either **(a)** or **(d)** is violated, depending on whether w and y are in the same component of $N_k(u)$ or not.

Now, from **(B)** and **(a)** they infer that there is no induced C_n, $n \geq 5$ in G_k. Note that the domino contains two induced paths of length 4. If this graph were induced in G_k, then at least one of the two paths would violate the conclusion of **(B)**
To prove that the graphs house and gem does not occur in G_k, Bandelt and Mulder consider induced paths of length 3 touching $N_k(u)$ such that the end vertices are at distance 2 in G_k. By **(A)** and **(a)** there are only five possibilities for such paths. Figure 4.3.5.1– 4.3.5.5, illustrating the cases (1)-(5). In the figures $v \in N_k(u)$ and $w \in N_{k-1}(u)$ indicating the levels. Let z be a common neighbor of v and y.

Case 1.

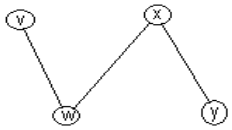


Figure 4.3.5.1 (Bandelt and Mulder p.203)

It follows from **(a)** that z is in $N_{k-1}(u)$. But then either **(e)** or **(c)** is violated.

Case 2. The vertices v and y belong to different components of $N_k(u)$ by **(a).**

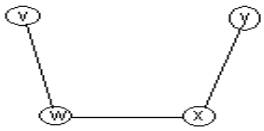


Figure 4.3.5.2 (Bandelt and Mulder p.203)
Hence z is in $N_{k-1}(u)$. This however, together with **(e)** violates **(d)**.

Case 3. It follows from **(a)** that z is in $N_{k-1}(u)$, conflicting with **(e).**

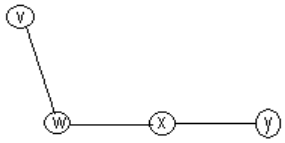


Figure 4.3.5.3 (Bandelt and Mulder p.203)

Case 4. Similarly to *Case* 3, z is in $N_{k-1}(u)$. But this violates either **(e)** or **(c).**

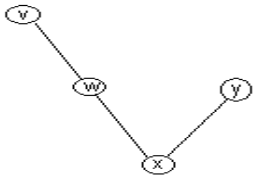


Figure 4.3.5.4 (Bandelt and Mulder p.203)

Case 5. Either **(a)** or **(c)** is not satisfied.

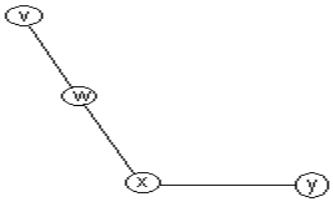


Figure 4.3.5.5 (Bandelt and Mulder p.203)

Now, in view of Theorem 4, the proof is completed. □

5 COGRAPHS
Complement-reducible graphs

5.1 DEFINITION AND FEATURES OF COGRAPHS

In this section we will become familiar with cographs. They have been discovered and studied under various names by different authors; "$D * -graphs$" (Jung), "P_4- restricted grahps" (Corneil, Lerchs and Burlinham), "Hereditary Dacey graphs" (Sumner), and "2-parity graphs" (Burlet and Uhry) are names and authors given on the cograph page on en.Wikipedia.org/cograph.

In this paper we will use the name cograph, as we shall later on become acquaintances to a *cograph* recognition algorithm given by Corneil, Perl and Stewart.

There are a number of characteristics that are equivalent to a graph's being a cograph. The following conditions are found in wolfram.mathworld, Corneil et al., Mandelt & Mulder and Chuang-Chieh Lin. Some conditions are the same but formulated differently, others are unique.

We will be concerned with the cotree later on in the text, thus first and foremost, (Corneil et al. p. 927):

> Cographs have two remarcable properties:
> 1) Cographs are precisely the class of graphs that does not have a path of length three, that is a path containing exactly four vertices, as an induced subgraph.
> 2) They can be uniquely represented by a tree, a so called *cotree.*
> The cotree consists of internal nodes and leaves. The internal nodes are labelled 0 or 1. The root is always labelled 1 thus being a (1) node. The labelling of internal nodes are such that (0) nodes and (1) nodes alternate along every path, through the tree, starting from the root. The leaves in the cotree T, correspond to the vertices of the graph G. Two vertices x and y are adjacent in G if and only if the unique path from x to the root of the tree meets the unique path from y to the root of the tree at a (1) node..

(Chuang-Chieh Lin):

> The internal nodes, (0) -nodes respectively (1) nodes of a cotree corresponds to union respectively join operations.

Moreover, one can state a number of equivalent conditions so that if a graph satisfies one of them it is a cograph and thus indeed satisfies all of them and Wolfram Mathworld presents the following(mathworld.wolfram.com/Cograph):

1. G can be constructed from isolated vertices by disjoint union and graph join operations.
2. G is the disjoint union of distance-hereditary graphs with diameter at most 2.
3. In every induced subgraph H of G, the intersection of any maximal clique and any maximum independent set contains precisely one vertex
4. Every nontrivial subgraph of G has at least one pair of twins
5. The graph complement of every nontrivial connected subgraph of G is disconnected
6. Every connected subgraph of G has diameter at most 2.
7. G does not contain the path graph P_4 as an induced subgraph

Chuang-Chieh Lin states and proves the following: G's being a cograph implies condition 4. (4)⇒(3), (3)⇒(7), (7)⇒(5) and finally that (5) implies that G is a cograph. In this paper we will be concerned mainly with condition (7). Bandelt and Mulder (Bandelt and Mulder p. 193) states the conditions (1), (2) and (7) and also gives the condition

> G can be obtained from the one-vertex graph by a sequence of vertex splittings (i.e. twin operations).

In the appendix we will construct cotrees when applying the cograph recognition algorithm.

6 DETERMINING WHETHER A GRAPH IS DISTANCE-HEREDITARY OR NOT

Recall theorem 1, 2 and 3.d. They really say the same thing but one can take this a bit further by explaining that this property has an actual meaning. Hammer and Maffray gives the explanation: given a distance hereditary graph G, $|G| = n$, there is an indexing of the vertices and a list of "words" on the form v_iPv_j, v_iFv_j and v_iTv_j, with their respective meanings: v_i is a pendant vertex attached to v_j, v_i is a false twin to v_j, v_i is a true twin to v_j, for $1 \leq i < n$ and $j < i$. This means that the graph G induced by vertices $v_0, \ldots, v_i$ is obtained from the subgraph G' induced by vertices $v_0, \ldots, v_{i-1}$ by making v_i either a pendant, a false twin or a true twin to v_j. This list of words is called a *pruning sequence* (Hammer and Maffray p.90-91). Thus we have that a graph G is a distance-hereditary graph if and only if it has a pruning sequence by Theorem 1 and Theorem 3.d.

We will use a linear time distance-hereditary graph recognition algorithm, first introduced by Hammer and Maffray. However, Damiand, Habib and Paul, found that there was an error in the their algorithm such that although the graph contains a domino or a house, it will answer "yes, the graph is distance-hereditary" if one starts with a degree 3 vertex. This is corrected by Damiand et al.
In the algorithm one also employs a cograph recognition algorithm given by Corneil, Perl and Stewart. We present the algorithm in this section and demonstrate it in Appendix 4.

6.1 RECOGNITION ALGORITHMS

In this section we present the linear-time algorithm used to determine whether a given graph is distance-hereditary or not. As mentioned above, a cograph recognition algorithm is employed, which in turn calls the procedure $MARK(x)$ and the function $FIND - LOWEST$.
Clearly, each subalgorithm, procedure and function employed during the quest for the overall purpose of the algorithm must also run in at most linear time. We shall see that this is the case as we present time complexity analysis for each and one of them.

6.1.1. Distance-Hereditary Graph Recognition Algorithm

Now, recall theorem 5 with its five conditions. The distance-hereditary graph recognition algorithm is based on this theorem. This will be clearer when we present the proof of the algorithms correctness.
The linear time algorithm for recognizing a distance-hereditary graph calls a number of algorithms such that there is algorithms, procedures and functions within the algorithm for different purposes. The working order is simplified as follows: Find the *distance layouts, L_i ,(i is the distance from the starting vertex)* of the input graph $G = (V, E)$, then find the connected components of each such level. Check if each connected component in the distance layout are free from induced paths of length 3, i.e. whether they are cographs or not. This is done by employing the cograph recognition algorithm of Corneil et al.'s. Now, since cographs are distance-hereditary they have a pruning sequence. Hence we get a pruning sequence and a "last vertex" of L_i, to which we contract the distance layout at hand – this means the other vertices of each connected component of L_i are removed from the graph during the pruning. For the remaining vertices of L_i, one first determines which are pendant vertices to any vertex in a distance layout, L_{i-1}, that is one distance unit closer to the starting vertex. For the others, those of higher *inner degree* one needs to check that the closer distance level is a cograph too, since, if the remaining vertices are twins or pendants to vertices in the closer level – that must also be a cograph (by theorem 5(b)) in order to still have the possibility of G being a distance-hereditary graph.
We shall now present each algorithm in the order in which they are first employed. First, to get an

overview, only stated in terms of input and output and its main task. Thereafter fully described in pseudo code. We will later, in Appendix 4, give a total and successful application of the algorithm.

Algorithm 1:Prune-dhg(G) (Damiand et al.)
Input: a graph $G = (V, E)$.
Output: a pruning sequence $iff\ G$ is a distance-hereditary graph.
Compute the distance layouts, L_i, of G.

Algorithm 2: Prune-cograph (G) (Damiand et al.)
Input: a graph $G = G(L_i)$
Output: a pruning sequence and its last vertex iff $G(L_i)$ is a cograph.
Compute a cotree of $G(L_i)$ using cograph recognition algorithm

Algorithm 3: Cograph-Recognition (Corneil et al.)
Input: a graph $G(L_i)$;
Output: a cotree iff $G(L_i)$ is a cograph

Procedure: $MARK(x)$
Input: a cograph $G = (V, E)$, a cotree T and vertex x to be added to G
Output: marked and unmarked leaves and internal nodes of the cotree T.

Function: Find Lowest
Input: the cotree with marked and unmarked leaves and internal nodes.
Output: the lowest marked vertex of T if $G + x$ is a cograph.

Algorithm 4: Verification Step (Damiand et al.)
Input: A graph $G = (V, E)$ and a list of words, $S = [vjXvi \ldots vkXvl]$, where X is either P, F or T.
Output: "True" iff S is a pruning sequence.
Below follows the more detailed descriptions of the algorithms in pseudocode.

6.1.1.1. Algorithm 1, Prune-dhg(G)

(Damiand et al. p.108):
Input: a graph $G = (V, E)$.
Output: a pruning sequence iff G is a distance-hereditary graph.

1. Begin
1.1 $j \leftarrow 1$; compute the distance layouts $L_1, .., L_k$ from an arbitrary starting vertex v.
2.0 **For** $i = k$ downto 1 **Do**
2.1 **For each** connected component cc of $G[L_i]$ **Do**
2.1.1 $z \leftarrow$ Prune-cograph$(G[cc], j)$;
contract cc into z;
$j \leftarrow j + |cc| - 1$;
sort the vertices of $G[L_i]$ by increasing inner degree;
For each vertex x of L_i with inner degree 1 **Do**
let y be the only neighbor of x;
$\sigma(j) \leftarrow x$ and $s_j \leftarrow (xPy)$;
$j \leftarrow j + 1$;
If $i \neq 1$ **Then**
For each x in L_i taken in increasing inner degree order **Do**
$y \leftarrow$ Prune-cograph$(G[N_{i-1}(x)], j)$;
contract $N_{i-1}(x)$ into y;
j$\leftarrow j + |N_{i-1}(x)| - 1$;

$\sigma(j) \leftarrow x$ and $s_j \leftarrow (xPy)$;
$j \leftarrow j + 1$;

End

6.1.1.2. Algorithm 2, Prune-cograph(G, j)

(Damiand et al. p.107):
Input: a graph G
Output; A pruning sequence (S, σ) and the last vertex of the pruning sequence if and only if G is a cograph;

Begin
Call cograph-recognition(G) algorithm to compute a cotree T of G.
Let A be the nodes of T having only leaves as descendant;
While $A \neq \emptyset$ **Do**
Pick an arbitrary node N in A;
Pick an arbitrary son x of N;
For Each son $y \neq x$ of N **Do**
If N is a 1-node
Then $\sigma(j') = y$ and $s_j' \leftarrow (yTx)$;
Else $\sigma(j') = y$ and $s_j' \leftarrow (yFx)$;
$j \leftarrow j' + 1$;
Replace N by x in T;
If x is the root of T **Then** Return; x is the last vertex of the pruning sequence;
If father(N) has only leaves as descendant **Then**
add father(N) to A;
End,

6.1.1.3. Timing Analysis of Algorithm 1, Prune-dhg(G) and Algorithm 2, Prune-cograph(G, j)

We shall here present a theorem and its proof given by Damiand et al. The theorem states that the algorithm Prune-dhg(G) actually computes a pruning sequence if G is distance-hereditary, and also the converse, that is, if it computes a pruning sequence then G is distance hereditary. Moreover, the theorem states that the algorithm runs in $O(n + m)$. The latter statement, however, does depend on the use of a proper linear time cograph recognition algorithm. The one we present and use here is the one given by Corneil, Perl and Stewart which later will be shown to run in linear time. First we do need a new definition for the proof of theorem 6. We closely follow Damiand, Habib and Paul through the definition, the theorem and the proof.

Definition 6.1.1.3.1 is based on Damiand et al. definition 4. A set of vertices M of a graph G is a *module* iff for any x and y in M, $N(x) \backslash M = N(y) \backslash M$. A module M is a *prime module* iff any subset S, such that $|S| \neq 1$ and $S \neq M$, of M is not a module. A module M is a *strong module* iff for any module M' either M' is a subset of M or M is a subset of M'.

Theorem 6 (Damiand et al. p.108). Algorithm 1, prune-dhg(G), computes a pruning sequence of G iff G is a distance-hereditary graph. It runs in $O(n + m)$.

Proof (Damiand et al. pp.108-109). If the computed sequence is a pruning sequence, then G is a distance-hereditary graph (Theorem 3(d)). So we just have to prove the converse. During the i: th loop 1, all the vertices of the sets L_j for $i < j \leq k$ have been removed. By theorem 5(a), each connected component of $G[L_i]$ is a module. Since $G[L_i]$ is a cograph (theorem 5(b)), twins in $G[L_i]$ are also twins in G. Thus, in loop 2,

we can contract each connected component cc of $G[L_i]$ and build a pruning sequence of $G[cc]$ with Algorithm 2. At this step L_i is a stable set. Thus (loop 5), the remaining vertices of L_i with inner degree 1 can be removed as pendant vertices.
Now by theorem 5(d), the neighborhood of two distinct vertices of L_i are either disjoint or one these neighborhoods (ordering the vertices with respect to their inner degree produces such a linear extension). Let x be the first vertex in this ordering. Let u and v be two distinct vertices of $N_{i-1}(x)$.. By theorem 5(d), $N_i(u) = N_i(v)$. If u and v are in distinct connected component cc(u) and cc(v) of L_{i-1}, then $N_{i-2}(u) = N_{i-2}(v)$ (Theorem 5(c)). Moreover, $cc(u)$ and $cc(v)$ are included in $N_{i-1}(x)$. Finally, if u and v are in the same connected component cc of L_{i-1}, theorem 5(d) shows that $N_{i-2}(u) = N_{i-2}(v)$ and by theorem 5(e), if w in cc is not adjacent to x then w is adjacent to both u and v or none of them. Therefore, $N_{i-1}(x)$ is a module. Since it is contained in L_{i-1}, $G[N_{i-1}(x)]$ is a cograph. Twins in $G[N_{i-1}(x)]$ are twins in G: $N_{i-1}(x)$ can be contracted into a single vertex and compute a pruning sequence of $G[N_{i-1}(x)]$. Respecting the linear extension of neighborhoods, the previous argument can be applied to all remaining vertices of L_i. That ends the proof of the correctness of the theorem.
They now give some ideas for the complexity issues. Computing the distance layouts can be done in $\mathcal{O}(n+m)$ via a breadth first search. Each connected component of $G[L_i]$ can be contracted into a sinlge vertex in $\mathcal{O}(|G[L_i]|)$. Sorting the vertices of L_i with respect to their inner degree can also be done in linear time using som bucket sort. For each distance layout, the global complexity of contracting $N_{i-1}(x)$ into y is $\mathcal{O}(|\ G[L_i \cup L_{i-1}]|)$. Thus, the whole complexity is $\mathcal{O}(n+m)$.

6.1.2. Algorithm 3, Cograph-Recognition(G)

This algorithm is given by Corneil, Perl and Stweart. We follow the authors and present the main frame of the cograph-recognition algorithm first and then specify procedure $MARK(x)$ that is called in step 2.1 and function $FIND-LOWEST$ that is called in step 2.4. As to $MARK(x)$ there follows a theorem in which concepts derived from $MARK(x)$ is used to state equivalence conditions for an extended cograph to be a cograph too. The iterations, each and every one of them, of the cograph-recognition algorithm essentially consists of an efficient implementation of that theorem.

6.1.2.1 Cograph-Recognition(G)

(Corneil et al. p928-932).

The algorithm is incremental in the sense that the vertices are processed one by one. We begin on the cotree for two vertices in step 1 and incorporate the remaining vertices one by one.
Given a graph $G = (V, E)$ with vertices arbitrarily indexed $v_1, \ldots, v_n$ this algorithm determines whether or not G is a cograph and constructs the cotree T of G, if G is a cograph.

1. Initialize
   Create a new (1) node R
1.1 **If** $(v_1, v_2) \in E(G)$
   Then add v_1, v_2 as children of R
   Else create a new (0) node N;
      add N as a child of R;
      add v_1 and v_2 as children of N

2. (iteratively incorporate $v_1, \dots, v_n$ into T)
For $x \leftarrow v_3, \dots, v_n$ **Do**
2.1 **Call** procedure **MARK(x)**
2.2 **If** all nodes of T were marked and unmarked
Then add x as a child of R
goto endloop
2.3 **If** no nodes of T mere marked
Then **if** $d(R) = 1$
then add x as a child of the only child of R
else create a new (1) node R with one child(and a new (0) node) and two grandchildren: x and the old root;
goto endloop
2.4 $u \leftarrow$ **FIND-LOWEST**
2.5 Let A (B) denote the set of children of u which were (were not) marked
if $label(u) = 0 (= 1)$
then if $|A| = 1$ $(|B| = 1)$
then if w $\in A$ (in B) is a leaf
then add a new (1) node ((0) node) in place of w and make w and x children of this new node
else add x as a new child of w
else remove all elements of A from u and add them as children of a new node y with $label(y) = label(u)$
if u is a (0) node
then add a new (1) node as a child of u;
children of this new (1) node are x and y
else remove u from its parent and add y in its place;
add a new (0) node as a child of y;
children of this new (0) node are x and y

endloop

End COGRAPH-RECOGNITION

6.1.2.2 Procedure MARK(x)

We shall not only present the procedure $MARK(x)$ but also set it into context by presenting a theorem in which output from the procedure i used. This theorem states two conditions, containing concepts derived from the output of $MARK(x)$, that are equivalent to $G + x$ being a cograph, where G is a cograph with cotree T. Following the theorem there are a number of conclusions drawn that will be used in the $FIND - LOWEST$ function.

Some notation that is used in the procedure(Corneil et al. pp. 928-929):
$d(w)$ denotes the number of children of w in T and $md(w)$ is the current number of children of w which have been both ”marked” and ”unmarked”. For all nodes w, the value of $md(w)$ is initially set to 0 and reset to 0 when w is unmarked.

Mark all leaves of T which are adjacent to x
For each marked node u of T with $d(u) = md(u)$ Do
unmark u;
$md(u) \leftarrow 0$;
if $u \neq R$ **then**

mark (w) where w is the parent of u;
$md(w) \leftarrow md(w) + 1$;
insert u at the head of a linked list of marked and unmarked children of w
end

If any vertex remains marked and $d(R) = 1$ **Then** mark R;
End MARK.

6.1.2.3. Theorem 7

Now suppose we have worked through $MARK(x)$. The following notation, theorem and proof are given by (Corneil et al. p. 929):

Let M denote the set of internal nodes of T which remain marked, and let α be a node in M with lowest level in the tree and let β be a node in $M\backslash\{\alpha\}$ with lowest level. We say a marked (1) node γ is properly marked if and only if $md(\gamma) = d(\gamma) - 1$. A legitimate alternating path in a marked cotree is a path of adjacent alternating properly marked (1) nodes and unmarked (0) nodes, the extreme point of which are (1) nodes.

Note that the root has the lowest level, and that a legitimate alternating path is directed from higher to lower levels.

Theorem 7 (Corneil et al. p. 929):
If G is a cograph with cotree T then $G + x$ is a cograph if and only if
(1) M is empty or
(2) **(a)** $M\backslash\{\alpha\}$ consists of exactly the (1) nodes of a (possibly empty) legitimate alternating path which ends at R and
(b) α is either a (0) node whose parent is β, or α is a (1) node whose grandparent, if it exists, is β.

6.1.2.4. Proof of Theorem 7

Proof (Corneil et al. pp. 929-930):
Only if: If the conditions of theorem 7 do not hold, then we have at least one of the following conditions:

(i) $M\backslash\{\alpha\}$ contains a (0) node
(ii) There exists a (1) node in $M\backslash\{\alpha\}$ which is not properly marked
(iii) There exists $\gamma \neq R$ in $M\backslash\{\alpha\}$ such that the grandparent of γ is not in $M\backslash\{\alpha\}$
(iv) The vertices of $M\backslash\{\alpha\}$ do not lie on one path to R
(v) α is a (0) node whose parent is not β
(vi) α is a (1) node which has a grandparent which is not β

By definition, any vertex in M has been marked but not unmarked, and this implies there is at least one descendant leaf adjacent to x and at least one not adjacent to x. Using this fact, it is fairly straightforward to show that any of the above six conditions implies the existence of an induced P_4 in the graph $G + x$. As an example, we demonstrate an induced P_4 in $G + x$ if condition (i) is found to be true. The following notation is used: for any internal node θ of T, $des(\theta)$ denotes the set of descendants of θ, that is, the leaves of the subtree of T rooted at θ. Let γ be a (0) node in $M\backslash\{\alpha\}$ and let δ be the lowest common ancestor of α and γ in T. Note the possibility that $\delta = \gamma$. There are four cases to be considered, depending on the labels (0) or (1) of α and δ.

$Case\ 1$ α and δ are both (0) nodes.

Proof: there is an induced P_4 on vertices b, c, x, d if x and c are adjacent in $G + x$, or on b, c, a, x if x and c are not adjacent, where; $a \in des(\alpha)$ and is adjacent to x; $b \in des(\alpha)$ and is not adjacent to x; $c \in des(parent(\alpha))\backslash des(\alpha)$; $d \in des(\gamma)$ and is adjacent to x. If $\delta = \gamma$, we require that $d \in des(\gamma)\backslash des(\theta)$, where θ is the child of γ on the $\alpha - \gamma$ path. $Cases\ 2, 3$ and 4 follow similarly.

Remark: This fairly straightforward way of showing the existence of an induced P_4 requires that one have in mind that vertices are adjacent in a cograph G iff their unique path towards R in the cotree meets at a (1)-node. We give $case\ 2$ with some intended clarifications:

$Case\ 2$. α is a (1) node and δ is (0) node, otherwise as in case 1 where c is not adjacent to x. c is not adjacent to any of the vertices: α is a (1) node, therefore c is a descendant of a (0) node since the sign of internal nodes in a cotree alternates. b is adjacent only to a since they meet at α, a is adjacent to x by assumption and so is d. Hence d, x, a, b induces a P_4. If c is adjacent to (only) x, then d, x, a, b and c, x, a, b induces P_4.

If: we complete this part of the proof by constructing T', the cotree repsresenting $G + x$.

1. If M is empty, then x can be added as a child of the root if $G + x$ is connected, or as a child of the only child of the root in the case where $G + x$ and G are both disconnected. If $G + x$ is disconnected but G is connected the root of T and x both become children of the only child of a new root.

2. There is a lowest marked node $\alpha \in M$. Let A be the children of α which were marked and subsequently unmarked by procedure $MARK$. Similarly, let B be the children of α which were not marked by $MARK$. The fact that $\alpha \in M$ implies that $|A| \geq 1$ and $|B| \geq 1$. To construct T' there are two cases to consider.

$Case\ 1$. α is a (0) node. In this case, the elements of A and B are either leaves or (1) nodes. If $|A| = 1$ and $\alpha \in A$ is a leaf, then we add a new (1) node in place of α and make α and x children of this node. If $|A| = 1$ and $\alpha \in A$ is a (1) node then we simply add x as a new child of α. If $|A| > 1$ then we remove all elements of A from α and add a new (1) node in their place. Children of this new node are x and a new (0) node with elements of A as children.

$Case\ 2$. α is a (1) node. The proof follows exactly as in $case\ 1$, except that B is examined instead of A, and the roles of (0) nodes and (1) nodes are reversed.

To see that T' is an accurate representation of $G + x$, we observer that the alterations to T correctly reflect adjacencies of x with vertices in the subtree rooted at α, and the fact that we have a legitimate alternating path from α to R guarantees that all other adjacencies of x are correctly represented. Adjacencies among vertices of G remain unchanged as required.

6.1.2.5. Function FIND-LOWEST

(Corneil et al. pp. 931-932):

The following notation is used: u is the lowest marked vertex so far examined; w denotes the lowest marked (1) node examined before u; y is a marked (1) node which is not properly marked or a marked (0) node if either exists in T. Whenever the procedure finds that $G + x$ is not a cograph, an accompanying comment indicates which of the conditions $i - vi$ from the proof of the theorem holds. When this occurs it is assumed that the entire algorithm is terminated.

1. (initialize and check root.)
   $y \leftarrow \Lambda$
   If R is not marked

then $G + x$ is not a cograph /* (iii)
else do
if $md(R) \neq d(R) - 1$ **then** y $\leftarrow R$;
unmark R;
$md(R) \leftarrow 0$;
$u \leftarrow w \leftarrow R$
End

2. Choose an arbitrary marked vertex u and follow the path from u to w, checking for a legitimate alternating path and unmarking vertices along the path.

While there are marked vertices remaining in T **Do**
choose an arbitrary marked vertex u
2.1 **If** $y \neq \Lambda$
Then $G + x$ is not a cograph by $(cond.i)$ or $(cond.ii)$
If label$(u) = 1$
Then do
if m$d(u) \neq d(u) - 1$
then $y \leftarrow u$
if parent(u) is marked
then $G + x$ is not a cograph by $(con.i)$ and $(cond.vi)$
else $t \leftarrow$ parent(parent(u))
end
else do $y \leftarrow u$;
$t \leftarrow$ parent(u)
end
unmark u;
$md(u) \leftarrow 0$
2.2 **while** $t \neq w$ **do**
if $t = R$ **then** $G + x$ is not a cograph $(cond.iv)$
if t is not marked
then $G + x$ is not a cograph by $(cond.iii)$ or $(cond.v)$ or $(cond.vi)$
if $md(t) \neq d(t) - 1$
then $G + x$ is not a cograph by $(cond.ii)$
if parent(t) is marked
then $G + x$ is not a cograph by $(cond.i)$
unmark t;
$md(t) \leftarrow 0$
end
2.3 $w \leftarrow u$
end (step 2)
End FIND-LOWEST

6.1.2.6 . Timing analysis of $MARK(x)$ and $FIND - LOWEST$

In this section follows the timing analysis for the algorithm (Corneil et al. pp. 933-934):

The algorithm relies on a time bound of $\mathcal{O}(deg(x))$ for the iteration adding x to T, where $deg(x)$ is the degree of x in $G + x$.

Since all internal nodes of T, except possibly the root, have at least two children, we know that the $MARK(x)$ procedure will examine only $\mathcal{O}(deg(x))$ nodes. For each of these nodes,

the processing is done in constant time, and thus the time bound for procedure $MARK(x)$ is $\mathcal{O}(deg(x))$. It is clear that $|M|$ is also bounded by $\mathcal{O}(deg(x))$, and since $FIND - LOWEST$ examines each marked node once in constant time, the time for this function is $O(|M|) = \mathcal{O}(deg(x))$.

All but one of the tree alterations can be done in constant time. The only cases which may require more than constant time are those where the lowest marked node is a (0) node ((1) node) which has two or more children which have been marked and unmarked (not been marked). In both cases, we are careful to move the children which were both marked and unmarked, since the cardinality of this set is $\mathcal{O}(deg(x))$whereas the cardinality of the set of children which were not marked is not similarly bounded. In procedure $MARK(x)$ we have maintained a linked list of the children which were marked and subsequently unmarked, and hence, they can be accessed in time bounded by $\mathcal{O}(deg(x))$. Therefore, all of the tree modifications can be done in $\mathcal{O}(deg(x))$ time. Thus, we have the required bound for each iteration, implying an overall time bound of $\mathcal{O}(m + n)$ for the entire cograph-recognition algorithm. This together with the linear time bound for the distance-hereditary graph recognition algorithm (theorem 6), we do indeed get an overall time bound of $\mathcal{O}(m + n)$

7. FORMATION OF A DISTANCE-HEREDITARY GRAPH FROM TWO OF THAT KIND

The authors Chang, Hsieh and Chen give a new recursive definition of distance-hereditary graphs, using the concept of *twin-set* and consider a distance-hereditary graph as being recursively constructed by applying the pendant, false twin and true twin operations on distance-hereditary graphs not restricted to consist of a single vertex.

This is made possible through their gaining some insights about distance-hereditary graphs and the representation of such graphs in the so called *one-vertex-extension tree* (Chang et al. p. 345) which is directly based on the *one-vertex-extension ordering* (Chang et al. p. 345). Before revealing this new recursive definition we will need quite some prerequisites which is given below, following Chang, Hsieh and Chen (see also appendix 2).

7.1. ONE-VERTEX-EXTENSION ORDERING

As mentioned above, a one-vertex-extension ordering is a set of nodes together with an ordering that constitutes the order in which each vertex is added to the graph by twin or pendant operations. Exactly as with the pruning sequence, the *one-vertex-extension ordering* $V_i = \{v_0 < ... < v_i\}$ (Chang et al. p. 345) means that the distance-hereditary graph G, induced by the set of vertices $v_0, \dots, v_i$ is obtained from the subgraph G' of G, induced by the set of vertices $v_0, \dots, v_{i-1}$ making v_i either a pendant or a twin to v_j, $1 \leq j < i$. It is merely a reformulation of the pruning sequence. Neither the pruning sequence nor the one vertex extension ordering is unique, since there may be different sequences of one-vertex extensions leading to the same distance hereditary graph.

7.2. ONE-VERTEX-EXTENSION TREE, $ET(G)$

7.2.1. Construction of $ET(G)$

We learn from (Chang et al. p.345) that the *one-vertex-extension tree*, $ET(G)$ with respect to the one-vertex-extension ordering of G, $V = \{v_0 < ... < v_n\}$ is constructed as follows:

Let v_1 be the *root*, and for $1 \leq j < i \leq n$ follow the one-vertex-extension ordering so that if a vertex v_i is one of iPj, iFj and iTj to vertex v_j then it is a child of v_j. Order the children of a node as they are ordered in the one-vertex-extension ordering. In this way $ET(G)$ becomes a *rooted ordered tree*.

Notice now that when picturing the tree in one's mind or drawing the tree with the root ”at the top” of the picture, downwards along a path through the tree, indices are strictly increasing (this is the imposed partial ordering). We think of the first child of a vertex in $ET(G)$ as the leftmost one, and add the others in increasing order to the right. Notice here that we can have lower indices at the same distance from v_1 to the right in *another branch* of the tree.

7.2.2. Features of the $ET(G)$

We present the features of the one-vertex-extension tree, $ET(G)$, following (Chang et al. pp. 345-346):

> Let v_j be a parent to v_i in $ET(G)$, $j < i$. We denote by $v_j v_i$ an edge in $ET(G)$. We call it a P-edge or a T- respectively an F-edge if v_i is a pendant vertex attached to v_j, or a true respectively a false twin to v_j.
>
> If G is connected, then $v_1 v_2$ is either a P- or T-edge, (recall Remark under theorem 1).
>
> For two vertices v_i and v_j which are siblings of each other in ET(G) and $i < j$, we say that v_i

is to the left of v_j, respectively v_j is to the right of v_i. If j is the maximum (respectively minimum) number such that v_j is a child of v_i, we say that v_j is the *rightmost* (respectively *leftmost*) child of v_i.
$ET[i]$ denotes the subtree of $ET(G)$ rooted at node v_i and induced by v_i and all of its descendants.
Suppose $v_{s_1} < v_{s_2} < \cdots < v_{s_j} < v_{s_{j+1}} < \cdots < v_{s_k}$ are children of v_i and v_{s_j} is not the rightmost child of v_i . Then $ET[i, s_j]$ denotes the subtree of v_i with respect to node v_{s_j}, which is the subtree rooted at v_i and induced by v_i , $v_{s_j}, v_{s_{j+1}}, \ldots, v_{s_k}$ and all decendants of $v_{s_j}, v_{s_{j+1}}, \ldots, v_{s_k}$.
$V(i)$ and $V(i, s_j)$ denote the sets of vertices in $ET[i]$ respectively $ET[i, s_j]$.
$V(i, s_j) - V(s_j)$ is denoted by $V_R(i, s_j)$ and $ET_R[i, s_j]$ denotes the corresponding subtree.
We use $G[i]$ instead of $G[V(i)]$ as a simplification. Similarly we let $G[i, s_j] = G[V(i, s_j)]$ and $G_R[i, s_j] = G_R[V(i, s_j)]$.

7.3. TWIN-SET

We present the concept of twin set closely following (Chang et al. pp. 345-346):

The twin set $TS(i)$ of a subtree, $ET[i]$ rooted at v_i, is the set of vertices that are descendants of v_i and such that they can be reached from v_i through twin edges only. $|TS(i)| \geq 1$ since v_i itself is in the twin-set.

The twin-set of v_i with respect to node v_{s_j}, denoted $TS(i, s_j)$ is the set of nodes that are both in $ET[i, s_j]$ and $TS(i)$, i.e. $TS(i, s_j) = TS(i) \cap ET[i, s_j]$. Similarly $TS_R(i, s_j)$ is the set of vertices that are both in $ET_R[i, s_j]$ and $TS(i)$, that is $TS_R(i, s_j) = TS(i) \cap ET_R[i, s_j]$.

The twin set of G, $TS(G)$, is the set of vertices that can be reached from the root v_1 of $ET(G)$ through twin edges only.

7.4. A FEW LEMMAS WITH PROOF.

In this section we present lemmas $7.4.2 - 7.4.7$ given by Chang, Hsieh and Chen. We give proofs $7.4.3 - 7.4.7$. We also give a lemma 7.4.1, with proof, which is useful when proving lemmas $7.4.3 - 7.4.7$. We use $p(v) = u$ to denote that u is the parent of v in $ET(G)$. By $N(v)$ we mean the neighborhood of v in G.

Lemma 7.4.1: Assume that $v_h, h > 1$, is a vertex in $ET(G)$ of G with one-vertex-extension ordering $V = \{v_1 < \ldots < v_n\}$. Also let $v_r \in V(h)$ and $v_s \in V(1) \backslash V(h)$ and assume that $v_r \in N v_s)$ (or equivalently $v_s \in N(v_r)$)

(a) If $r > s$, then there exists a vertex $v_{r'}$ such that $r' < r$, $v_{r'} \neq v_s$, $v_r{}' v_r$ is a twin edge and $v_s \in N(v_{r'})$.

Proof of (a): We have that $r > s > 1$, so there must be a father to v_r, say $v_r{}'$. Clearly $r' < r$.
Now $\{v_{r'}, v_s\} \subseteq N(v_r)$ when one-vertex-extension ordering $V = \{v_1 < \ldots < v_r\}$. Since $v_r \notin V(h)$ by assumption, v_r is not a descendant to v_h, therefore neither is its ancestor $v_{r'}$, so $v_r{}' \notin V(h)$ and thus $v_{r'} \neq v_s$.
Since $\{v_{r'}, v_s\} \subseteq N(v_r)$ we conclude that v_r is not a pendant vertex attached to $v_{r'}$, hence a twin. $v_r{}' v_r$ is therefore a twin edge in $ET(G)$ and $N[v_r] \backslash \{v_{r'}, v_r\} = N[v_{r'}] \backslash \{v_{r'}, v_r\}$ which implies that $v_s \in N(v_r)$ if $v_s \in N(v_{r'})$.
It is easily seen that those relations between $v_{r'}, v_r$ and v_s will not change when

the one-vertex-extension ordering $V = \{v_1 < ... < v_n\}, n > k$.

(b) If $s > r \geq 1$ and $p(v_h) = v_i \neq v_r$ *or* $v_s' \neq v_i$
Then there exists a vertex v_s' such that $s' < s$, $v_s' \neq v_r$, the edge $v_s'v_s$ is a twin edge and $v_r \in N(v_s')$.

Proof of (b) (Backelin and Timonen): We have that $s > r \geq 1$, so there must be a father to v_s , say $v_{S'}$. $s' < s$ clearly.
Now $\{v_s', v_r\} \in N(v_s)$ when one-vertex-extension ordering $V = \{v_1 < ... < v_s\}$. $v_s \in V(h)$ i.e. v_s is a descendant to v_h and so is any ancestor v_s' to v_s, for which $s' \geq h$. Thus $v_s' \in V(h) U \{v_i\}$. If $v_s' \neq v_i$ then $v_s' \in V(h)$ and since $v_r \notin V(h)$ we have that $v_s' \neq v_r$, otherwise we have by assumption that $v_s' = v_i \neq v_r$
Now, since $\{v_s', v_r\} \in N(v_s)$ we conclude thatv_s is not a pendant vertex attached to v_s', hence a twin. The edge $v_s'v_s$ is therefore a twin edge in $ET(G)$ and $N_G[v_s]\backslash\{v_s', v_s\} = N_G[v_s']\backslash \{v_s', v_s\}$which implies that $v_r \in N(v_s)$ if $v_r \in N(v_s')$□

Next we present lemmas given by Hung and Chang and we give proofs for each and every one.

Lemma 7.4.2 (Chang et al. p.346): The subtrees $ET[i]$ and $ET[i, h]$ of a one-vertex-extension tree of a distance-hereditary graph are also one-vertex-extension trees of the subgraphs $G[i]$ and $G[i, h]$ respectively.

Lemma 7.4.3 (Chang et al. p.346): Suppose v_h is a child of v_i in $ET(G)$ and v_iv_h is a P edge. Then the following two statements hold. (i) Every vertex in $V(h)$ is adjacent to nly vertices in $V(h) \cup TS_R(i, h)$, i.e. $N_G[V(h)] \subseteq V(h) \cup TS_R(i, h)$. (ii) every vertex in $V(h) - TS(h)$ is adjacent to only vertices in $V(h)$, i.e. $N_G[V(h) - TS(h)] \subseteq V(h)$.

Proof (Backelin and Timonen): We consider the following statements.
$A =$ "v_iv_h is a P-edge in $ET[G]$"
$B_1 =$ "$\forall v_s \in V(h)$ and $v_r \in N(v_S), v_r \in V(h) \cup TS_R(i, h)$"
$\neg B_1 =$ "$\exists v_s \in V(h)$ and $v_r \in N(v_s)\backslash(V(h)\ U\ TS_R(i, h))$"
$B_2 =$ "$\forall\ v_s \in V(h)\backslash TS(h)$ and $v_r \in N(v_s),\ v_r \in V(h)$.
$\neg B_2 =$ "$\exists v_s \in V(h)\backslash TS(h)$ and $v_r \in N(v_s)\backslash V(h)$.
To prove that $A \Rightarrow B$ is equivalent to prove that assuming A and $\neg B \Rightarrow \perp$

First we prove that statement (i) hold.
Assume A and $\neg\ B_1$.Define $\Omega = \{(v_s, v_r) \mid v_s \in V(h), v_r \in N(v_s)\backslash(V(h) \cup TS_R(i, h))\}$. Assume Ω is not empty. Then there is $\{v_s, v_r\} \in \Omega$ such that $s + r$ is the smallest sum of indices.
Remark: This sum should be the sum of $s = h$, h is the lowest index in $V(h)$, and some lowest r.

First let $r > s$. By lemma 7.4.1(a) $r' < r, v_r' \neq v_s, v_r'v_r$ is a twin edge and $v_s \in N(v_r')$. Also by assumption $v_r \neq v_i$ since $v_i \in TS_R(i, h)$. Now we have $(v_s, v_r') \in \Omega$. But $s + r' < s + r$ so we have that for any sum $s + r$ of indices in Ω, there is a smaller sum $s + r'$ of indices in Ω. That contradicts the assumption that Ω is nonempty in which case there would be a smallest sum of indices.

Next consider $s > r$. We have by assumption that $v_r \in TS_R(i, h)$ and hence $v_i \neq v_r$
Applying Lemma 7.4.1(b) for $s > r$ and $v_h \neq v_r$ we have that $v_r \in N(v_s') \Rightarrow (v_s', v_r) \in \Omega$.
But $s' + r < s + r = \perp$. This settles the accuracy of statement (i).
Next we prove that condition (ii) holds. As above, assume A and $\neg B_2$
Define $\Omega = \{(v_s, v_r) \mid v_s \in V(h)\backslash TS(h)\ , v_r \in N(v_s)\backslash V(h)\}$

Assume Ω is nonempty. Then there is $(v_s, v_r) \in \Omega$ such that $s + r$ is the smallest sum of indices.
Remark: Such a sum must be the sum of some r and the lowest index in $V(h)\backslash TS(h)$ wich would be held by the first pendant vertex attached to v_h.

First let $r > s$.
By lemma 7.4.1(a) $r' < r, v_r' \neq v_s, v_r'v_r$ is a twin edge and $v_s \in N(v_r') \Rightarrow (v_s, v_r') \in \Omega$. But $s' + r < s + r = \perp$

Now let $s > r$, Since $s > 1$ there is $p(v_s) = v_s'$, $v_s \in V(h)\backslash TS(h)$ by assumption so we know that $s > h$ and therefore $f(v_s) = v_s' \neq v_i$. Now applying 7.4.1(b) for $s > r$ and $p(v_s) = v_s' \neq v_i$ gives at hand that $v_r \in N(v_s') \Rightarrow (v_s', v_r) \in \Omega$. But $s' + r < s + r = \perp$
If $v_s' = v_h$ and $v_r \in N(v_s)$ we would have $v_s v_h$ being a twin edge and therefore $v_s \in TS(h)$ which again is on the contrary to our assumption. This settles the accuracy of statement (ii). □

Lemma 7.4.4 (Chang et al. p.346): Suppose v_h is a child of v_i in $ET(G)$ and the type of edge $v_i v_h$ is T. Then, in graph $G$the twin set of $ET[h]$ and the twin set of $ET_R[i, h]$ form a join (i.e. each vertex in $ET[h]$ is adjacent to each vertex in $ET_R[i, h]$).

Proof (Backelin and Timonen)**:** We consider the statements:
A = "$[v_i, v_h]$ is a T-edge in $ET[G]$"
B = "$\forall v_s \in TS(h)$ and $v_r \in TS_R(i, h) : v_r \in N(v_s)$"
$\neg B$ = "$\exists v_s \in TS(h)$ and $v_r \in TS_R(i, h)$ s.t. $v_r \notin N(v_s)$"
To prove that $A \Rightarrow B$ is equivalent to prove that assuming A and $\neg B \Rightarrow \perp$
We define $\Omega = \{(v_s, v_r) \mid v_s \in TS(h)$ and $v_r \in TS_R(i, h)\ s.t.\ v_r \notin N(v_s)\}$. Assume A and $\neg B$ i.e. A and Ω is not empty. Then there is $(v_s, v_r) \in \Omega$ such that $s + r$ is the smallest sum of indicies. This sum should be $s + r = h + i$ since h is the lowest index in $TS(h)$ and i is the lowest index in $TS_R(i, h)$

To prove this first let $s > h$
We have that $s > h > 1$ since v_h has a father v_i. The parent of v_s is v_s'. Since $v_s \in TS(h)$ so is v_s' and $v_s'v_s$ is a twin edge. This implies that $v_r \notin N(v_s) \Rightarrow v_r \notin N(v_s')$. Hence $(v_s', v_r) \in \Omega$ but $s' < s$ so that $s' + r < s + r$ for all indices of $s > h$ which contradicts our assumption about the sum $s + r$.This holds regardless of r's being greater or smaller than s. If $r < s$, and $v_r \notin N(v_s)$ then it is easily seen that $v_r \notin N(v_s')$. If $r > s$ we will have $v_r \in N(v_r')$ for some $r' < s$ such that $v_r' \notin N(v_s)$ by assumption. And since $v_r \in TS_R(i, h)$, $v_r'v_r =$ twin edge and thus $v_r \notin N(v_s)$.

Next let $r > i$.
We have that $r > i \geq 1$ and thus there is a father v_r' to v_r and also $v_r' \in TS_R(i, h), s + r' < s + r$, $v_r'v_r$ is clearly a twin edge which implies that $v_s \notin N(v_r) \Rightarrow v_s \notin N(v_r')$ for a couple $(v_s, v_r') \in \Omega$. This holds regardless of s's being greater or smaller than r.
If $s < r$ and $v_s \notin N(v_r)$ then it is easily seen that $v_s \notin N(v_r')$. If $s > r$ we will have $v_s \in N(v_s')$ for some $s' < r$ such that $v_s' \notin N(v_r)$ by assumption. And since $v_s \in TS(h), v_s'v_s$ is a twin edge and thus $v_s \notin N(v_r)$. Now we know that $s = h$ and $r = i$ so that $s + r = h + i$ must be the smallest sum of indices in Ω. But since $v_i v_h$ is a T-edge, $v_i \in N(v_h)$ and so $(v_i, v_h) \notin \Omega$.

Corollary 7.4.5 (Chang et al. p.346): Suppose v_h is a child of v_i in $ET(G)$ and $v_i v_h$ is a P edge. Then, in graph $G$the twin set of $ET[h]$ and the twin set of $ET_R[i, h]$ form a join.

Proof (Backelin and Timonen)**:** The very same argumentation holds for $v_i v_h$ is a P-edge, in which case also $v_i \in N(v_h)$. Remark: if $v_i v_h$ is an F-edge, v_i and v_h are not adjacent.

Lemma 7.4.6 (Chang et al. p.347): Suppose v_i is a node in $ET(G)$. Then $N_G[V_i - TS(i)] \subseteq V(i)$, i.e. vertices in $V(i)$ except those in the twin set of v_i are adjacent to only vertices in the subtree of $ET(G)$ rooted at v_i.

Proof (Backelin and Timonen)**:** We consider the statements:
$A =$ "v_i is a node in $ET(G)$.
$B =$ "$\forall v_s \in V(i) \backslash TS(i)$ and $v_r \in N(v_s)$; $v_r \in V(i)$"
$\neg B =$ "$\exists v_s \in V(i) \backslash TS(i)$ and $v_r \in N(v_s)$; $v_r \notin V(i)$"
To prove that $A \Rightarrow B$ is equivalent to prove that assuming A and $\neg B \Rightarrow \perp$

Define $\Omega = \{(v_s, v_r) \mid v_s \in V(i) \backslash TS(i)$ and $v_r \in N(v_s), v_r \notin V(i)\}$
Assume Ω is nonempty. Then there is $(v_s, v_r) \in \Omega$ s.t. $s + r$ is the smallest sum of indices.

First assume $r > s$. By lemma 7.4.1(a) $r' < r, v_{r'} \neq v_s, v_{r'} v_r$ is a twin edge and $v_s \in N(v_{r'}) \Rightarrow$ $(v_s, v_{r'}) \in \Omega$. But $s' + r < s + r = \perp$

Next assume $s > r$. Since $v_s \notin TS(i)$ we know that $s > h$ so we have that $p(v_s) = v_{s'} \neq v_i$. lemma 7.4.1.(a) implies $v_r \in N(v_{s'}) \Rightarrow (v_{s'}, v_r) \in \Omega$. But $s' + r < s + r = \perp$

Lemma 7.4.7 (Chang et al. p.347): Suppose v_h is a child of v_i in $ET(G)$ and $v_i v_h$ is an F edge. Then, $G[i, h]$ is not connected, and in graph G no vertex in $V(h)$ is adjacent to any vertex in $V_R(i, h)$.

Proof (Backelin and Timonen)**:** Let G' be the distance-hereditary subgraph of G, with one-vertex extension $\{v_1, \ldots, v_i, v_h\}$, where v_i and v_h are false twins. Hence in G', $N_{G'}(v_i) = N_{G'}(v_h)$, whereas in $ET(G)$ the edge $v_i v_h$ is an F-edge and there are no right siblings to v_h.

Let $A = N_{G'}(v_i) = N_{G'}(v_h)$. We have that neither v_i nor v_h is contained in A. The set A is a subset of $V(G')$ and contains vertices in $V(G') \backslash V(i, h)$.

In order to prove that no vertex in $V(h)$ is adjacent to any vertex in $V_R[i, h]$, we want to show that any vertex in $V(h)$ has neigbhors only in $V(h) \cup A$ and any vertex in $V_R(i, h)$ has neighbors only in $V_R(i, h) \cup A$.

We will use induction to prove that any vertex in $V(h)$ has neighbors only in $V(h) \cup A$.

Let $V(h) = \{x_1, x_2, \ldots\}$, where x_a is ordered by increasing vertex index, whence $x_1 = v_h$.

Let $a \in \{1, \ldots, |V(h)|\}$ and let $G^{(a)}$ be the induced subgraph on $v_1, \ldots, v_i, v_h, x_1, \ldots, x_a$. Thus $G' = G^{(1)}$. We shall prove that $N_{G^{(a)}}(x_j) \in V(h) \cup A$ for $j \in \{1, \ldots, a\}$.
We have that $N_{G'}(x_1) = N_{G'}(v_h) = A$. Thus the assertion holds for $a = 1$.

Assume that for $a < |V(h)|$ and $x_j \in \{x_1, \ldots, x_a\}$ it holds that $N_{G^{(a)}}(x_j) \in V(h) \cup A$. Note that $G^{(a+1)}$ is the extension of $G^{(a)}$ by attaching the vertex x_{a+1} to a vertex x_b, $1 \leq b < a + 1$, by a one-vertex extension. We have three cases.

Case 1. If x_{a+1} is a pendant vertex to x_b, then $N_{G^{(a+1)}}(x_{a+1}) = x_b \in V(h)$.

Case 2. If x_{a+1} is a true twin to x_b, then $N_{G^{(a+1)}}(x_{a+1}) = N_{G^{(a+1)}}[x_b]$. By induction hypothesis we have that $N_{G^{(a+1)}}[x_b] \in x_b \cup V(h) \cup A$.

Case 3. If x_{a+1} is a false twin to x_b, then $N_{G^{(a+1)}}(x_{a+1}) = N_{G^{(a+1)}}(x_b)$. By induction hypothesis we have that $N_{G^{(a+1)}}(x_b) \in V(h) \cup A$.

Hence any such vertex x_{a+1} is a child of x_b in $ET(G^{(a+1)})$ and will thereby be contained in $V(h)$ although not contained in A. x_{a+1} may have as neighbors the vertex x_b , vertices in A or vertices in $V(h)$, that is $N_{G^{(a+1)}}(x_j) \in N_{G^{(a)}}(x_j) \cup \{x_{a+1}\}$. Thus it holds that any vertex in $V(h)$ has neighbors only in $A \cup V(h)$.

Now, let G' be the distance-hereditary subgraph of G, with one-vertex extension $\{v_1, \dots, v_i, v_h, x_1, \dots, x_a\}$, where v_i and v_h are false twins, and$x_1, \dots, x_a \in A \cup V(h)$.

We will again use induction to prove that any vertex in $V_R(i,h)$ has neighbors in $V_R(i,h) \cup A$ only, and hence no neighbors in $V(h)$.

Let $V_R(i,h) = \{v_i, y_1, y_2, \dots\}$, where y_k is ordered by increasing vertex index. Let $c \in \{1, \dots, |V_R(i,h)|\}$ and let $G^{(c)}$ be the induced subgraph on $v_1, \dots, v_i, v_h, x_1, \dots, x_a, y_1, \dots, y_c$. Thus, with previously used notation, $G' = G^{(1)}$. We shall prove that $N_{G^{(c)}}(y_k) \in V_R(i,h) \cup A$ for $k \in \{1, \dots, c\}$.

We have that $N_{G'}(v_i) = A$ and also that y_1 is a child of v_i in $ET(G^{(1)})$. y_1 is either a pendant vertex, a true twin- or a false twin to v_i. If y_1 is a pendant vertex to v_i then $N_{G'}(y_1) = v_i \in V_R(i,h) \neq V(h)$. In the other two cases, where y_1 is a false twin respectively a true twin to v_i it holds that $N_{G'}(y_1) = N(v_i) = A \neq V(h)$ respectively $N_{G'}(y_1) = N[v_i] \in V_R(i,h) \cup A \neq V(h)$. Thus the assertion holds for $c = 1$.

Assume that for $c < |V_R(i,h)|$ and $y_k \in \{y_1, \dots, y_c\}$ it holds that $N_{G^{(c)}}(y_k) \in V_R(i,h) \cup A$. Again, note that $G^{(c+1)}$ is the extension of $G^{(c)}$ by attaching the vertex y_{c+1} to a vertex y_d, $1 \leq d < c+1$, by a one-vertex extension. We have three cases.

> $Case$ 1. If y_{c+1} is a pendant vertex to y_d, then $N_{G^{(c+1)}}(y_{c+1}) = y_d \in V_R(i,h)$.
>
> $Case$ 2. If y_{c+1} is a true twin to y_d, then $N_{G^{(c+1)}}(y_{c+1}) = N_{G^{(c+1)}}[y_d]$. By induction hypothesis we have that $N_{G^{(a+1)}}[y_d] \in V_R(i,h) \cup A$.
>
> $Case$ 3. If y_{c+1} is a false twin to y_d, then $N_{G^{(c+1)}}(y_{c+1}) = N_{G^{(c+1)}}(y_d)$.. By induction hypothesis we have that $N_{G^{(c+1)}}(y_d) \in V_R(i,h) \cup A$.

Hence any such vertex y_{c+1} is a child of y_d in $ET(G^{(c+1)})$ and will thereby be contained in $V_R(i,h)$ although not contained in A. y_{c+1} may have as neighbors the vertex y_d, vertices in A or vertices in $V_R(i,h)$, that is $N_{G^{(c+1)}}(y_k) \in N_{G^{(c)}}(y_k) \cup \{y_{c+1}\}$. Thus it holds that any vertex in $V_R(i,h)$ has neighbors only in $V_R(i,h) \cup A$ and hence no neighbors in $V(h)$.Since A, the only common neighborhood of vertices in $V(h)$ and vertices in $V_R(i,h)$ is not contained in $G(i,h)$ it follows that $G(i,h)$ is disconnected

7.5. FROM OLD TO NEW DEFINITION OF DISTANCE-HEREDITARY GRAPHS

Closely following Chang, Hsieh and Chen we will here present the very leap from the old to the new definition. Then we present Chang et al.'s new definition of distance-hereditary graphs stated as in Hung and Chang. In the next section we present the decomposition tree. First we recall a few key concepts from above. A distance-hereditary graph has a one-vertex-extension ordering, from which one can obtain a one-vertex-extension tree. This tree reveals information about the distance hereditary graph, and using the lemmas presented above one can make a few observations and conclusions.

With the definitions from the section 7.2.1- 7.2.3 Chang et al. define the twin-set $TS(1)$ of G. They also show that one can partition the vertex set into four disjoint sets (Chang et al. p.347):

$V(2) - TS(2), TS(2),\ V_R(1,2) - TS_R(1,2),$ and $TS_R(1,2)$. We can consider that G is formed from $G[2]$ and $G_R[1,2]$ according to the type of edge v_1v_2 , that is the type of edge between the root and the first child of the root, in $ET(G)$. If v_1v_2 is P or T we have by lemma 7.4.1 and 7.4.2 that G is formed by connecting every vertex in $TS(2)$ to all vertices in $TS_R(1,2)$. If the edge v_1v_2 is of type P, then $TS(G) = TS_R(1,2)$. If the edge $v_1v_2$is of type T, then $TS(G) = TS(2) \cup TS_R(1,2)$. If the edge $v_1v_2$is of type F we have by lemma 7.4.7 that $G[2]$ and $G_R[1,2]$ are not connected, i.e. G is the union of $G[2]$ and $G_R[1,2]$. $TS(G) = TS(2) \cup TS_R(1,2)$

This is the very leap from the one-vertex-extensions to the formation of a distance hereditary graph from two others that may have more than one vertex.

Now that all prerequisites are settled it is time to present the recursive definition of distance-hereditary graphs. This definition is actually a definition of pairs: $(G, TS(G))$, the distance-hereditary graph together with its twin-set. The reader is also referred to Appendix 3.

Definition 7.5.1 (Chang et al. p347, with the notation from Hung and Chang p.414):

The class of distance-hereditary graphs can be defined by the following recursive definition:
1) A graph consisting of a single vertex is a distance-hereditary graph with the twin set $\{v\}$.
2) If G_L and G_R are distance-hereditary graphs then the union G of G_L and $G_R$is a distance-hereditary graph and $TS(G) = TS(G_L) \cup TS(G_R)$. Then G is formed from G_L and G_R by a false-twin operation. This is denoted $G = G_L(F)G_R$.
3) If G_L and G_R are distance-hereditary graphs, then the graph obtained from G_L and G_R by connecting every vertex of $TS(G_L)$ to all vertices of $TS(G_R)$ is a distance-hereditary graph and $TS(G) = TS(G_L) \cup TS(G_R)$. We say that G is formed from G_L and G_R by a true-twin operation. This is denoted $G = G_L(T)G_R$.
4) If G_L and G_R are distance-hereditary graphs, then the graph G, obtained from G_L and G_R by connecting every vertex of $TS(G_L)$ to all vertices of $TS(G_R)$ is a distance-hereditary graph and $TS(G) = TS(G_L)$. In this case we say that G is formed by a pendant operation. This is denoted $G = G_L(P)G_R$.

Note that with the notation from the partitioning of $V(G)$ above we have, perhaps a bit confusing, that $G_L = G_R[1,2]$ and $G_R = G[2]$.
The graphs obtained from a true twin respectively a pendant operation are isomorphic, having different twin sets only.
If $G = G_L(F)G_R$, then G is disconnected for any G_L and G_R in the new definition, whereas with the old definition only extending K_1 by a false twin operation will result in a disconnected distance hereditary graph.
A distance-hereditary graph in "the old sense" has a twin-set, as seen in the derivation of $ET(G)$.

7.6. The decomposition tree, $DT(G)$

Definition 7.6.1. (Chang et al. p347, with the notation from Hung and Chang p.414):

The decomposition tree $DT(G)$ of a distance-hereditary graph G consisting of a single vertex v is a tree of one node labeled by v. If G is formed from G_L and G_R by a false-twin (respectively true-twin, pendant) operation, then the root of the decomposition tree $DT(G)$ is a node labeled by F (respectively T, P) with the roots of $DT(G_L)$ and $DT(G_R)$ being the left and right

children of the root of $DT(G)$, respectively.

Given the one-vertex-extension tree $ET(G)$ of Gone recursively constructs the decomposition tree, $DT(G)$, of G in the following way:
The root is a vertex labeled by the type of edge from the root of $ET(G)$ to its first child, i.e. the type of edge v_1v_2 in $ET(G)$. Next, determine $V_R(1,2)$ and $V(2)$ in $ET(G)$. The conclusions derived above allows us to consider G as being formed from $G_R[1,2]$ and $G[2]$ according to the type of edge v_1v_2 in $ET(G)$. Let $G_L = G_R[1,2]$ and $G_R = G[2]$. By lemma 2, the subtrees $ET(G_L)$ and $ET(G_R)$ are also one-vertex-extension trees of the sub graphs G_L and G_R respectively. Thus there is a decomposition tree $DT(G_L)$ and a decomposition tree $DT(G_R)$. This will be demonstrated in the appendix.

.7.7 A Twin Set Theorem with proof

Here we will state a theorem and give its proof, followed by a corollary and a remark on forbidden subgraphs in twin-sets of distance-hereditary graphs.

Theorem 8. A twin-set of a distance-hereditary graph can not induce a P_4 as a subgraph.

Proof: Assume that G_R is a distance-hereditary graph with twin-set $v_1, .., v_4$ such that this twin-set induces a P_4. Assume that G_L is a distance-hereditary graph on one vertex u_1 (hence $TS(G_L) = u_1$), and that $G = G_L(T)G_R$. Then G is a distance-hereditary graph and the first level with respect to u_1, $N_1(u_1)$ consists of an induced path of length 3, which by condition (ii) of theorem 4 is impossible for a distance-hereditary graph. □

Corollary: Using twin-set operations, it is impossible to create a path of length three, such that all four vertices belong to the twin-set.

Remark 1: the graph constructed above by a true-twin operation is readily seen to be the gem – a forbidden subgraph in any distance-hereditary graph.
Remark 2: in the same way, one can deduce the set of forbidden subgraphs (and thereby impossible to construct with twin- and pendant operations) induced by the twin-set of a distance-hereditary graph emanating from the overall forbidden subgraphs of distance-hereditary graphs, namely the house, the domino and the long cycles.

8. A SOLUTION TO THE HAMILTONIAN PATH-PROBLEM ON DISTANCE-HEREDITARY GRAPHS

Before presenting the recursive program for determining whether a given distance-hereditary has a Hamiltonian path or not we do need some definitions of concepts that are used for doing so. We also need a few lemmas and theorems. They are presented in this section. All lemmas, theorems and proofs thereof presented below are given by Hung and Chang. We shall present the proofs of the main theorems: theorems 9, 10 and 11 as given by Hung and Chang, whence the interested reader is referred to Hung and Chang for proofs of the lemmas.

8.1 PREREQUISITES FOR THE SOLUTION TO HAMILTONIAN PROBLEMS ON DISTANCE-HEREDITARY GRAPHS

In the definitions below it is assumed that $G = (V, E)$ is a distance-hereditary graph and either $G = G_L(P)G_R$, $G = G_L(F)G_R$ or $G = G_L(T)G_R$. V_L and V_R denotes the vertex sets of G_L and G_R respectively.

Definition 8.1.1 (Hung and Chang p.416):

The *first* and the *last* vertices that are visited by the path P, are called the path-start and path-end of P, respectively. Both of them are *end vertices* of P. For a path P of a graph, we allow that the path-start and path-end of P are the same only in the case when P contains exactly one vertex.

Definition 8.1.2 (Hung and Chang p.416):

A *path cover* PC of a graph G is a set of pairwise vertex-disjoint paths of G such that all vertices are visited by exactly one path in PC. A *minimum path cover* of G is a path cover of G of minimum cardinality.

Definition 8.1.3 (Hung and Chang p.416):

For a path cover PC of G, the end vertices not in TS(G)are called free vertices of PC. The free number of PC in G is the number of free vertices, and is denoted $\tau(G, PC)$. For simplicity we shall in this paper denote it τ.

Define$\tau_L(G, PC) = |\{v \mid v \text{ is an end vertex of a path in } PC, v \notin TS(G) \text{ and } v \in V_L\}|$. That is, v is a free vertex in G that origins from the left graph G_L when forming G. $\tau_R(G, PC)$ is defined in the same way, and in this paper $\tau_L$and τ_R will be used in the obvious way.

Definition 8.1.4 (Hung and Chang p.416):

A path of G is called a *twin-set path* or, in this paper a *proper twin set path*, if both its end vertices are in $TS(G)$. A path of G is called a *semi-twin-set path* if exactly one of its end vertices is in $TS(G)$.

A path cover PC is called a *twin-set-path cover* of G if the following holds:

(a) every path in PC is either a semi- or a proper twin-set path.

(b) at most two paths in PC are semi-twin-set paths.

Notice that a path on one vertex is either a proper twin set path or not a twin set path, since the start and the end vertices are the same and thus cannot be in different sets.

Definition 8.1.5 (Hung and Chang p.416):

$f(G)$ denotes the minimum free number of a twin-set path cover of G. If G has no twin-set path cover, then $f(G)$ is undefined. A twin-set path cover PC of G with $\tau = f(G)$ is

called a minimum-free-number twin-set path cover of G if $f(G)$ is defined.

Definition 8.1.6 (Hung and Chang p.416):

Let U be a subset of $V(G)$, and let P be a path in G. A subpath P' of P is called a *U-subpath* of P if P' visits vertices in U only. Such a U-subpath is *U-maximal* if it is not a proper subpath of any U-subpath of P. For PC denote by $U(PC)$ the set of all U-maximal subpaths of all paths in PC.

Proposition 8.1.7 (Hung and Chang p.417):

Assume that PC is a twin-set path cover of a distance-hereditary graph G. Then, $\tau_L(G, PC) = \tau\big(G_L, V_L(PC)\big)$ and $\tau_R(G, PC) \geq \tau\big(G_R, V_R(PC)\big)$.

Lemma 8.1.8 (Hung and Chang p.419):

Assume that G is formed from G_L and G_R by one of a false-twin operation, a true-twin operation, and a pendant operation, and that PC is a twin-set path cover of G. Then, $V_L(PC)$ and $V_R(PC)$ are twin-set path covers of G_L and G_R respectively and $\tau(G, PC) \geq f(G_L) + f(G_R)$

8.1.1 Theorem 9 with proof

Theorem 9 (Hung and Chang p.419):

Assume that $G = G_L(F)G_R$ or $G = G_L(T)G_R$.

Then the following two statements hold:

(a) G has a twin-set path cover if and only if both G_L and G_R have twin-set path covers, and $f(G_L) + f(G_R) \leq 2$.

(b) If G has a twin-set path cover, then $f(G) = f(G_L) + f(G_R)$.

Proof (Hung and Chang p.419):

We first prove statement 9(a). Suppose that PC is a twin-set path cover of G. By definition, $\tau(G, PC) \leq 2$. By Lemma 8.1.8, $V_L(PC)$ and $V_R(PC)$ are twin-set path covers of G_L and G_R respectively, and $\tau(G, PC) \geq f(G_L) + f(G_R)$. Thus $f(G_L) + f(G_R) \leq 2$. This proves the only if part of 9(a).

Suppose that both G_L and G_R have twin-set path covers and $f(G_L) + f(G_R) \leq 2$. Then, G_L and G_R have minimum-free-number twin-set path covers PC_L and PC_R, respectively.

Let $PC = PC_L \cup PC_R$. Then, $\tau(G, PC) \leq 2$ since $\tau(G, PC) = f(G_L) + f(G_R)$ and $f(G_L) + f(G_R) \leq 2$.

Hence, PC is clearly a twin-set path cover of G and hence, G has a twin-set path cover. This proves the if part of statement 1.

Next we prove statement 2. Suppose G has a twin-set path cover. By statement (i) both G_L and G_R have twin-set path covers and hence, G_L and G_R have minimum-free-number twin-set path covers PC_L and PC_R, respectively. By definition, $\tau(G_L, PC_L) = f(G_L)$ and , $\tau(G_R, PC_R) = f(G_R)$.

Thus $\tau(G, PC_L \cup PC_R) = \tau(G_L, PC_L) + \tau(G_R, PC_R) = f(G_L) + f(G_R)$.

By Lemma 8.1.8, $PC_L \cup PC_R$ is a minimum-free-number twin-set path cover of G, and $f(G) = f(G_L) + f(G_R)$. This prove statement 9(b). □

Lemma 8.1.9 (Hung and Chang p.419):

Assume that G is formed from G_L and G_R by either a false-twin operation or a true-twin operation. If PC is a minimum-free-number twin-set path cover of G, then $V_L(PC)$ and $V_R(PC)$ are twin-set path covers of G_L and G_R, respectively.

Lemma 8.1.10 (Hung and Chang p.420):

Assume that G is formed from G_L and G_R by a false-twin operation, PC is a twin-set path cover of G of size p with τ free vertices, and that PC_L and PC_R are minimum-free-number twin-set path covers of G_L and G_R respectively, where $|V_L(PC)| \geq |PC_L| \geq |V_L(PC)| - \tau(G_L, V_L(PC)) + f(G_L)$ and $|V_R(PC)| \geq |PC_R| \geq |V_R(PC)| - \tau(G_R, V_R(PC)) + f(G_R)$. Then, there exists a minimum-free-number twin-set path cover PC_f of G of size p_f such that $p \geq p_f \geq p - \tau + f(G)$.

Lemma 8.1.11 (Hung and Chang p.420):

Assume that G is formed from G_L and G_R by a true-twin operation and that PC is a twin-set path cover of G.

Then, $|V_L(PC)| + |V_R(PC)| \geq |PC| \geq$
$\max\{1, \tau(G, PC), |V_L(PC)| - |V_R(PC)| + \tau_R(G, PC), |V_R(PC)| - |V_L(PC)| + \tau_L(G, PC)\}$.

Lemma 8.1.12 (Hung and Chang p.421):

Assume that G is formed from G_L and G_R by a true-twin operation, PC_L and PC_R are minimum-free-number twin-set path covers of G_L and G_R respectively, and that $\tau = \tau(G_L, V_L(PC)) + \tau(G_R, V_R(PC)) \leq 2$. Then, for any number k, where $|PC_L| + |PC_R| \geq k \geq \max\{1, \tau, |PC_L| - |PC_R| + \tau(G_R, PC_R), |PC_R| - |PC_L| + \tau(G_L, PC_L)\}$, there exists a twin-set path cover PC of G of size k such that $VL(PC) = PC_L, V_R(PC) = PC_R$, and $\tau(G, PC) = \tau$.

Lemma 8.1.13 (Hung and Chang p.422):

Assume that G is formed from G_L and G_R by a true-twin operation, PC is a twin-set path cover of G of size p with τ free vertices, and that PC_L and PC_R are minimum-free-number twin-set path covers of G_L and G_R respectively, $|V_L(PC)| \geq |PC_L| \geq |V_L(PC)| - \tau(G_L, V_L(PC)) + f(G_L)$ and $|V_R(PC)| \geq |PC_R| \geq |V_R(PC)| - \tau(G_R, V_R(PC)) + f(G_R)$. Then, there exists a minimum-free-number twin-set path cover PC_f of G of size p_f such that $p \geq p_f \geq p - \tau + f(G)$.

Lemma 8.1.14 (Hung and Chang p.423):

Assume that G is formed from G_L and G_R by a pendant operation and that PC is a twin-set path cover of G. Then, $|PC| = |V_L(PC)| - |V_R(PC)| + \tau_R(G, PC)$ and $|V_L(PC)| - |V_R(PC)| \geq \tau_L(G, PC) \geq f(GL)$.

Lemma 8.1.15 (Hung and Chang p.423):

Assume that G is formed from G_L and G_R by a pendant operation, PC_L and PC_R are minimum-free-number twin-set path covers of G_L and G_R respectively, $\tau(G_L, PC_L) + \tau(G_R, PC_R) \leq 2$, and that $|PC_L| - |PC_R| \geq \tau(G_L, PC_L)$. Then G has a twin-set path cover PC satisfying the following conditions:

(a) $V_L(PC) = PC_L, V_R(PC) = PC_R$;

(b) exactly one of the following two conditions holds:

2.1 $\tau(G, PC) = \tau(G_L, PC_L) + \tau(G_R, PC_R)$ *and* $|PC| = |PC_L| - |PC_R| + \tau(G_R, PC_R)$;

2.2 $\tau(G_L, PC_L) = \tau(G_R, PC_R) = 0 |PC_L| = |PC_R|, |PC| = 1$, and $\tau(G, PC) = \tau_R(G, PC) = 1$.

Lemma 8.1.16 (Hung and Chang p.424):

Assume that G is formed from G_L and G_R by a pendant operation, PC is a twin-set path cover of G of size p with τ free vertices, and that PC_L and PC_R are minimum-free-number twin-set path covers of G_L and G_R respectively, where
$|V_L(PC)| \geq |PC_L| \geq |V_L(PC)| - \tau(G_L, V_L(PC)) + f(G_L)$ and $|V_R(PC)| \geq |PC_R| \geq |V_R(PC)| - \tau(G_R, V_R(PC)) + f(G_R)$. Then, there exists a twin-set path cover PC_f of G of size

p_f satisfying the following conditions:

1) $V_L(PC_f) = PC_L$, and $V_R(PC_f) = PC_R$,
2) $p \geq p_f \geq p - \tau + \tau(G, PC_f)$,
3) $\tau(G, PC_f) \leq \tau$,
4) exactly one of the following two conditions holds:
   (a) $\tau(G, PC_f) = f(G_L) + f(G_R)$,
   (b) $f(G_L) = f(G_R) = 0,\ p_f = 1$, and $\tau(G, PC_f) = \tau_R(G, PC_f) = 1$.

We shall now state one of the main theorems of this section. It contains a recursive definition of the numbers $\kappa_1(G)$ and $\kappa_2(G)$ which are the maximum, respectively minimum size of twin-set path cover of G.

8.1.2. Theorem 10

Theorem 10 (Hung and Chang p.425):

Assume that either $G = G_L(P)G_R$, $G = G_L(F)G_R$ or $G = G_L(T)G_R$.
Then the following statements hold:

(1) if G has a minimum-free-number twin-set path cover of size p, then G has a minimum-free-number twin-set path cover PC_f of size p such that $V_L(PC_f)$ and $V_R(PC_f)$ are minimum-free-number twin-set path covers of G_L and G_R respectively.

(2) If G has a twin-set path cover of size p with free number τ, then G has a minimum-free-number twin-set path cover of size p_f such that$p \geq p_f \geq p - \tau + f(G)$.

(3) Suppose G has a twin-set path cover. Then, there exists two numbers $\kappa_1(G)$ and $\kappa_2(G)$ such that G has a minimum-free-number twin-set path cover of size k, if and only if $\kappa_1(G) \geq k \geq \kappa_2(G)$. Moreover, the values of $\kappa_1(G)$ and $\kappa_2(G)$ are as determined recursively as follows:

   (a) if $G = G_L(F)G_R$, then $\kappa_1(G) = \kappa_1(G_L) + \kappa_1(G_R)$ and $\kappa_2(G) = \kappa_2(G_L) + \kappa_2(G_R)$.

   (b) if $G = G_L(T)G_R$, then $\kappa_1(G) = \kappa_1(G_L) + \kappa_1(G_R)$ and $\kappa_2(G) = max\{1, f(G), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)\}$.

   (c) if $G = G_L(P)G_R$, then $\kappa_1(G) = max\{1, \kappa_1(G_L) - \kappa_2(G_R) + f(G_R) + f(GR)\}$ and $\kappa_2(G) = max\{1, f(G_L) + f(G_R), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R)\}$.

(4) Suppose $G = G_L(P)G_R$. Then G has a twin-set path cover if and only if the following three conditions are satisfied:
   (a) both G_L and $G_R$have twin-set path covers.
   (b) b) $f(G_L) + f(G_R) \leq 2$.
   (c) $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L)$.

(5) If $G = G_L(P)G_R$ and G has a twin-set path cover, then $f(G) = f(G_L) + f(G_R)$ if $\kappa_1(G_L) - \kappa_2(G_R) \geq 0$; and $f(G) = max(1, f(G_R))$ otherwise.

8.1.3. Proof of theorem 10

Following Hung and Chang we present the proof of theorem 10 (Hung and Chang pp.426-434):

This theorem is proved by using induction on the number of vertices of G. Clearly, statement 2 and 3 (except the recursive definition) hold true if the number of vertices is one.
In this case let $\kappa_1(G) = \kappa_2(G) = 1$ and $f(G) = 0$. by induction hypothesis, all statements hold true for graphs with the number of vertices smaller than G. Hence, all statements hold true for G_L and G_R. We will prove statement (1) by using the induction hypothesis of statement (2). By induction hypotheses of statement (2) and statement (3), we prove statement (2). We will prove statement (4) by using statement (1) and the induction hypothesis of statement (3). By statement (1), statement (4), and the induction hypothesis of statement (3), we prove statement (3). We will prove statement (5) by using statement (4) and the proof of statement (3). These statements in this theorem are closely related and we shall prove the theorem as a whole. In the following, we will prove all statements hold true for G.

Proof of statement (1). By Lemma 8.1.9, the statement is true if G is formed from G_L and $G_R$by either a false twin operation or a true-twin operation. In the following, assume G is formed from G_L and $G_R$by a pendant operation. Consider the following two cases:
Case 1: $f(G) = 0$. Let PC be any minimum-free-number twin-set path-cover of G. Clearly $\tau_L(G, PC) = 0$ and $\tau_R(G, PC) = 0$. By proposition 2.2,$\tau_L(G, PC) = \tau\big(G_L, V_L(PC)\big)$ and $\tau_R(G, PC) \geq \tau(G_R, V_R(PC))$. Therefore, $\tau\big(G_L, V_L(PC)\big) = \tau(G_R, V_R(PC)) = 0$ andhence, $V_L(PC)$ and $V_R(PC)$ are minimum-free-number twin-set path covers of G_L and G_R respectively.
Case 2: $f(G) \geq 1$. Let PC be any minimum-free-number twin-set path cover of G of size p with τ free vertices. By induction hypothesis of statement (2), there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and $G_R$respectively such that $|V_L(PC)| \geq |PC_L| \geq |V_L(PC)| - \tau\big(G_L, V_L(PC)\big) + f(G_L)$ and $|V_R(PC)| \geq |PC_R| \geq |V_R(PC)| - \tau\big(G_R, V_R(PC)\big) + f(G_R)$. By Lemma 8.1.16 there exists a twin-set cover PC_f of G of size p_f such that $V_L\big(PC_f\big) = PC_L$, and $V_R(PC_f) = PC_R$, $p \geq p_f \geq p - \tau + \tau\big(G, PC_f\big)$, $\tau\big(G, PC_f\big) \leq \tau$, and exactly one of the following two conditions holds:
$(c.1.1)\ \tau\big(G, PC_f\big) = f(G_L) + f(G_R)$,
$(c.1.2)\ f(G_L) = f(G_R) = 0, p_f = 1, and\ \tau\big(G, PC_f\big) = \tau_R\big(G, PC_f\big) = 1.$
If condition (c.1.1) holds, then PC_f is a minimum-free-number twin-set path cover of G by Lemma 8.1.8. If condition (c.1.2) holds, then PC_f is a minimum-free-number twin -set path cover of G by Lemma 8.1.8 since $\tau\big(G, PC_f\big) = 1$ and $f(G) \geq 1$. In all the above two subcases PC_f is a minimum-free-number twin-set path cover of G and hence $\tau\big(G, PC_f\big) = \tau$. Since $p \geq p_f \geq p - \tau + \tau\big(G, PC_f\big)$ and $\tau\big(G, PC_f\big) = \tau$, we have $p_f = p$. Therefore PC_f is a minimum-free-number twin-set path cover of G of size p with that $V_L\big(PC_f\big)$ and $V_R(PC_f)$, are minimum-free-number twin-set path covers of G_L and G_R respectively.
By arguments given in the above two cases, this statement holds when G is formed from G_L and G_R by a pendant operation. This completes the proof of statement (1). □

Proof of statement (2). Assume PC is a twin-set path cover of G of size p with τ free vertices. By Lemma 8.1.8, $V_R(PC)$ and $V_R(PC)$ are twin-set path covers of G_L and G_R respectively. By induction hypothesis of this statement, there exist minimum-free-number twin-set path-covers PC_L and PC_R of G_L and G_R respectively, such that
$|V_L(PC)| \geq |PC_L| \geq |V_L(PC)| - \tau\big(G_L, V_L(PC)\big) + f(G_L)$ and $|V_R(PC)| \geq |PC_R| \geq |V_R(PC)| - \tau\big(G_R, V_R(PC)\big) + f(G_R)$. By lemmas 8.10 and 8.13 this statement is true if G is formed from G_L and G_R by either a false-twin operation or a true-twin operation. In the following, assume G is formed from G_L and G_R by a pendant operation. By Lemma 8.1.16, there exists a twin-set path cover PC_f of G of size p_f such that

$V_L(PC_f) = PC_L$, and $V_R(PC_f) = PC_R$, $p \geq p_f \geq p - \tau + \tau(G, PC_f)$, $\tau(G, PC_f) \leq \tau$, and exactly one of the following two conditions holds:
$(c.2.1)$ $\tau(G, PC_f) = f(G_L) + f(G_R)$,
$(c.2.2)$ $f(G_L) = f(G_R) = 0, p_f = 1$, and $\tau(G, PC_f) = \tau_R(G, PC_f) = 1$.
Consider the following four cases:
$Case\ 1$. Condition (c.2.1) holds. In this case, PC_f is a minimum-free-number twin-set path cover of G and $p \geq p_f \geq p - \tau + f(G)$ since $\tau(G, PC_f) = f(G)$ by Lemma 8.1.8. Hence, this statement is true in this case.

$Case\ 2$. Condition (c.2.2) holds and $f(G) = 1$. This statement is clearly true in this case.

$Case\ 3$ Condition (c.2.2) holds, $f(G) = 0$, and there exists a minimum-free-number twin-set path cover PC_{f^*} of G of size 1. Let PC_f be PC_{f^*}. Then, this statement holds since $|PC_{f^*}| = p_f = 1$ and $\tau(G, PC_{f^*}) = f(G) = 0$.

$Case\ 4$ Condition (c.2.2) holds, $f(G) = 0$, and all minimum-free-number twin-set path covers of G are of size greater than 1. Since the only path in PC_f starts from a vertex in the twin-set of G_L and ends at a vertex of in the twin-set of G_R, we have that $|V_L(PC_f)| = |V_R(PC_f)|$ and $V_L(PC_f)$ and $V_R(PC_f)$ are minimum-free-number twin-set path covers of G_L and $G_R$respectively. Let PC_F be a minimum-free-number twin-set path cover of G. By assumption, $|PC_f| > 1$and $\tau(G, PC_f) = f(G) = 0$. By Lemma 8.1.14 $|PC_F| = |V_L(PC_F)| - |V_R(PC_F)| + \tau_R(G, PC_F) = |V_L(PC_F)| - |V_R(PC_F)| > 1$ and hence $|V_L(PC_F)| > |V_R(PC_F)|$. By proposition 8.7 $\tau_L(G, PC_F) = \tau(G_L, V_L(PC_F))$ and $\tau_L(G, PC_F) \geq \tau(G_R, V_R(PC_F))$. Since $f(G) = \tau(G, PC_F) = \tau_L(G, PC_F) + \tau_R(G, PC_F) = 0, 0 \geq \tau(G_L, V_L(PC_F)) + \tau(G_R, V_R(PC_F))$ and hence $\tau(G_L, V_L(PC_F)) = \tau(G_R, V_R(PC_F)) = 0$. Hence, $V_L(PC_F)$ and $V_R(PC_F)$ are minimum-free-number twin-set path covers of G_L and G_R respectively. There are the following two subcases:
$Case\ 4.1$ $|V_R(PC_F)| < |V_R(PC_f)|$. Since $|V_L(PC_f)| - 1 = |V_R(PC_f)| - 1 \geq |V_R(PC_F)|$ and both $V_R(PC_F)$ and $V_R(PC_f)$ are minimum-free-number twin-set path covers of G_R, there exists a minimum-free-number twin-set path cover of G_R of size $|V_L(PC_f)| - 1$ by induction hypothesis of statement (3).
$Case\ 4.2$ $|V_R(PC_F)| \geq |V_R(PC_f)|$. Since $|V_L(PC_F)| > |V_R(PC_F)| \geq |V_R(PC_f)| = |V_L(PC_f)|$, we have $|V_L(PC_F)| \geq |V_R(PC_f)| + 1$. since both $V_L(PC_F)$ and $V_L(PC_f)$ are minimum free-number twin-set path covers of G_L, there exists a minimum-free-number twin-set path cover of G_L of size$|V_R(PC_f)| + 1$ by induction hypothesis of statement (3).
In both the above two subcases, we can obtain a Hamiltonian path of G starting from a vertex in the twin set of G_L and ending at a vertex in the twin set of G_L. Hence, we have a minimum-free-number twin-set path cover of G of size 1, a contradiction.
By arguments given in the above four cases, this statement holds when G is formed from G_L and G_R by a pendant operation. This completes the proof of statement (2). □

Since statement (4) will be used to prove statement (3) in case that G is formed from G_L and G_R by a pendant operation, we prove statement (4) before statement (3) in the following.

Proof of statement (4). We first prove the only if part of this statement. Suppose G has a twin-set path cover. By Lemma 8.1.8 both G_L and $G_R$and have twin-set path covers and $f(G_L) + f(G_R) \leq 2$. Hence, condition (a) and condition (b) of this statement are satisfied. Since G has at twin-set path cover, G has a minimum-free-number twin-set path cover. By statement (1), G

has a minimum-free-number twin-set path cover PC such that $V_L(PC)$ and $V_R(PC)$ are minimum-free-number twin-set path covers of G_L and $G_R$respectively. By Lemma 8.1.14. $|VL\ (PC)| - |VR(PC)| \geq f(GL)$. By induction hypothesis of statement (3), $\kappa_1(G_L) \geq |V_L(PC)| \geq \kappa_2(G_L)$ and $\kappa_1(G_R) \geq |V_R(PC)| \geq \kappa_2(G_R)$. Thus $\kappa_1(G_L) - \kappa_2(G_R) \geq |V_L(PC)| - |V_R(PC)| \geq f(GL)$ and hence condition (c) of this statement is satisfied. This proves the only if part of this statement. Next, we prove the if part of the statement. By induction hypothesis of statement (3), there exist minimum-free-number twin-set path-covers PC_L and PC_R of G_L and $G_R$respectively such that $|PC_L| = \kappa_1(G_L)$ and $|PC_R| = \kappa_2(G_R)$. Since $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L), |PC_L| - |PC_R| \geq f((G_L) = \tau(G_L, PC_L)$. Clearly $\tau(G_L, PC_L) + \tau(G_R, PC_R) = f(G_L) + f(G_R) \leq 2$. By Lemma 8.1.15 G has a twin-set path cover and hence, the if part of this statement holds true. This completes the proof of statement 4. □

Proof of statement (3). By assumption, G has a twin-set path cover. Hence, G has a minimum-free-number twin-set path cover. Suppose PC is a minimum-free-number twin-set path cover of G of size k. We have three cases: G is formed from G_L and $G_R$by a false-twin operation, a true-twin operation and a pendant operation.

Case 1: G is formed from G_L and G_R by a false-twin operation.

We first prove the only if part of this statement in this case. Since all vertices in V_L are not adjacent to any vertex in V_R, $|PC| = k = |V_L(PC)| + |V_R(PC)|$. By Lemma 8.1.9, $V_L(PC)$ and$V_R(PC)$ are minimum-free-number twin set path covers of G_L and G_R respectively. By induction hypothesis of this statement, we have

$\kappa_1(G_L) \geq |V_L(PC)| \geq \kappa_2(G_L)$ (eq. 1)
$\kappa_1(G_R) \geq |V_R(PC)| \geq \kappa_2(G_R)$ (eq. 2)

Combining (eq.1) and (eq.2), we get $\kappa_1(G) = \kappa_1(G_L) + \kappa_1(G_R) \geq \kappa_2(G_L) + \kappa_2(G_R) = \kappa_2(G)$. This proves the only if part of statement (3) of theorem 9 in case that G is formed from G_L and G_R by a false-twin operation.

Next we prove the if part of this statement in this case. Suppose $\kappa_1(G) = \kappa_1(G_L) + \kappa_1(G_R) \geq k \geq \kappa_2(G_L) + \kappa_2(G_R) = \kappa_2(G)$. Then there exist two numbers, $k_L$and k_R such that $\kappa_1(G_L) \geq k_L \geq \kappa_2(G_L), \kappa_1(G_R) \geq k_R \geq \kappa_2(G_R)$, and $k = k_L + k_R$. By induction hypothesis of this statement, there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R, respectively such that $|PC_L| = k_L$ and $|PC_R| = k_R$. Clearly $\tau(G, PC_L \cup PC_R) = \tau(G_L, PC_L) + \tau(G_R, PC_R) = f(G_L) + f(G_R) \leq 2$ and $PC_L \cup PC_R$ is a twin-set path cover of G. By theorem 9, $f(G) = f(G_L) + f(G_R)$. Thus, for $\kappa_1(G) \geq k \geq \kappa_2(G)$, there exist a minimum-free-number twin-set path cover of G of size k. This proves the if part of statement (3) of theorem 9 in case that G is formed from G_L and G_R by a false-twin operation.

Case II: G is formed from G_L and G_R by a true-twin operation.

By Lemma 8.1.9, $V_L(PC)$ and $V_R(PC)$ are minimum-free-number twin-set path covers of G_L and G_R respectively. By induction hypothesis of this statement we have $\kappa_1(G_L) \geq |V_L(PC)| \geq \kappa_2(G_L)$ and $\kappa_1(G_R) \geq |V_R(PC)| \geq \kappa_2(G_R)$. By Lemma 8.1.11, $\kappa_1(G) = \kappa_1(G_L) + \kappa_1(G_R) \geq |V_L(PC)| + |V_R(PC)| \geq |PC| \geq max\{1, \tau(G, PC), |V_L(PC)| - |V_R(PC)| + \tau_R(G, PC), |V_R(PC)| - |V_L(PC)| + \tau_L(G, PC) \geq max\{1, f(G), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)\} = \kappa_2(G)$.. This proves the only if part of statement (3) of theorem 10 in case that G is formed from G_L and G_R by a true-twin operation.

We next prove the if part of this statement in this case by showing that the following statement holds: for any number k, where $\kappa_1(G) = \kappa_1(G_L) + \kappa_1(G_R) \geq k$ and

$k \geq max\{1, f(G), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)\} = \kappa_2(G)$, there exists a minimum-free-number twin-set path cover of G of size k. By theorem 9, $f(G) = f(G_L) + f(G_R)$.

Consider the following cases:

$Case\ 1$. $\kappa_1(G_L) + \kappa_1(G_R) \geq k \geq \kappa_2(G_L) + \kappa_2(G_R)$. There exist two numbers, k_L and k_R such that $\kappa_1(G_L) \geq k_L \geq \kappa_2(G_L), \kappa_1(G_R) \geq k_R \geq \kappa_2(G_R)$, and $k = k_L + k_R$. By induction hypothesis of this statement, there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R respectively such that $|PC_L| = k_L$ and $|PC_R| = k_R$. Clearly $\tau(G, PC_L \cup PC_R) = \tau(G_L, PC_L) + \tau(G_R, PC_R) = f(G_L) + f(G_R) \leq 2$. By theorem 9, $PC_L \cup PC_R$ is a minimum-free-number twin-set path cover of G of size k.

$Case\ 2$.

$\kappa_2(G_L) + \kappa_2(G_R) \geq k \geq max\{1, f(G), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)\} = \kappa_2(G)$. In this case there are three subcases:

$Case\ 2.1$ $\kappa_2(G_L) \geq \kappa_1(G_R)$. In this subcase, $\kappa_1(G_L) \geq \kappa_2(G_L) \geq \kappa_1(G_R) \geq \kappa_2(G_R)$. Thus we have

$\kappa_2(G_R) - \kappa_1(G_L) + f(G_L) \leq f(G_L) + f(G_R) = f(G)$ (eq.3)

$\kappa_1(G_R) - \kappa_2(G_L) + f(GL) \leq f(G_L) + f(G_R) = f(G)$ (eq.4)

By induction hypothesis of this statement, there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R respectively such that $|PC_L| = \kappa_2(G_L)$ and $|PC_R| = \kappa_1(G_R)$. By Lemma 8.1.12, for any number k, where $|PC_L| + |PC_R| \geq k \geq max\{1, f(G), |PC_L| - |PC_R| + f(G_R), |PC_R| - |PC_L| + f(G_L)\}$, there exists a minimum-free-number twin-set path cover of G of size k. That is, for any number k, where

$\kappa_2(G_L) + \kappa_1(G_R) \geq k \geq max\{1, f(G), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_1(G_R) - \kappa_2(G_L) + f(G_L)$

there exists a minimum-free-number twin-set path cover of G of size k. By eqs (eq.3) and (eq.4)$max\{1, f(G), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_1(G_R) - \kappa_2(G_L) + f(G_L) = max\{1, f(G), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L) = \kappa 2(G)$. Hence, for any number k, where $\kappa_2(G_L) + \kappa_1(G_R) \geq \kappa_2(G_L) + \kappa_2(G_R) \geq k \geq \kappa_2(G)$, there exists a minimum-free-number twin-set path cover of G of size k.

$Case\ 2.2$ $\kappa_2(G_R) \geq \kappa_1(G_L)$. By symmetry, we can prove this subcase by arguments similar to those for proving Case 2.1

$Case\ 2.3$ Neither $\kappa_2(G_L) \geq \kappa_1(G_R)$ nor $\kappa_2(G_R) \geq \kappa_1(G_L)$. In this subcase, $\kappa_2(G_L) - \kappa_1(G_R) \leq 0$ and $\kappa_2(G_R) - \kappa_1(G_L) \leq 0$. Hence, we have

$\kappa_2(G_L) - \kappa_1(G_R) + f(G_R) \leq f(G)$ (eq.5)

$\kappa_2(G_R) - \kappa_1(G_L) + f(G_L) \leq f(G)$ (eq.6)

By eqs (eq.5) and (eq.6), $\kappa_2(G) = \max\{1, f(G)\}$. By induction hypothesis of this statement, there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R respectively such that $\kappa_1(G_L) \geq |PC_L| \geq \kappa_2(G_L)$ and $\kappa_1(G_R) \geq |PC_R| \geq \kappa_2(G_R)$. If $\kappa_2(G_L) \geq \kappa_2(G_R)$, then $\kappa_1(G_R) \geq \kappa_2(G_L) \geq \kappa_2(G_R)$ and hence, we can obtain minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R respectively such that $|PC_L| = |PC_R| = \kappa_2(G_L)$. Similarly, we can obtain the minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R respectively such that $|PC_L| = |PC_R| = \kappa_2(G_R)$ if $\kappa_2(G_L) \leq \kappa_2(G_R)$. In any case, we can obtain the minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R respectively such that $|PC_L| = |PC_R|$ and $|PC_L| + |PC_R| \geq \kappa_2(G_L)^+ \kappa_2(G_R)$. By Lemma 8.1.12, for any number k, where $|PC_L| + |PC_R| \geq k \geq \{max\{1, f(G), |PC_L| - |PC_R| + f(G_R), |PC_R| - |PC_L| + f(G_L)\} = max\{1, f(G)\} = \kappa_2(G)$, there exists a minimum-free-

number twin-set path cover of G of size k. Hence, for any number k, where $|PC_L| + |PC_R| \geq \kappa_2(G_L) + \kappa_2(G_R) \geq k \geq \kappa_2(G)$, there exists a minimum-free-number twin-set path cover of G of size k.

By arguments given in the above cases, the if part of statement (3) of theorem 10 holds true in case that G is formed from G_L and G_R by a true-twin operation.

Case 3 *is formed from* G_L and G_R *by a pendant operation.*

By assumption, G has a twin-set path cover and hence, G has a minimum-free-number twin-set path cover. By statement (1), assume PC is a minimum-free-number twin-set path cover of G with VL (PC) and V_R(PC) being minimum-free-number twin-set path covers of G_L and G_R respectively. By statement (4) we have

$$f(G_L) + f(G_R) \leq 2 \quad \text{(eq.7)}$$
$$\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L) \quad \text{(eq.8)}$$

We first prove the following five claims which are used in proving statement (3) in this case.

Claim 1. $|PC| = 1$ or $\tau_R(G, PC) = f(G_R)$.

Proof. By definition $\tau_R(G, PC) \geq f(GR)$ and $|PC| \geq 1$. Assume, by contradiction, that $|\mathrm{PC}| > 1$ and $\tau_R(G, PC) > f(GR)$. Since $V_R(PC)$ is a minimum-free-number twin-set path cover of G_R and $\tau_R(G, PC) > f(G_R) = \tau(G_R, V_R(PC))$, there exists a path P_1 in PC having an end vertex in the twin set of G_L, and the other end vertex in the twin set of G_R. Let P_2 be another path in PC other than P_1. By definition, P_2 has an end vertex in the twin set of G_L. Without loss of generality, assume the path-end of P_1 is in the twin set of G_R and the path-start of P_2 is in the twin set of G_L. Then, $P = P_1P_2$ is a path having at least one end vertex in the twin set of G_L. Clearly, $(PC \backslash \{P_1, P_2\}) \cup \{P\}$ is a twin-set path cover of G with less free vertices than PC, a contradiction. Thus, $|PC| = 1$ or $\tau_R(G, PC) = f(G_R)$ □

Claim 2. If $\tau_R(G, PC) \geq f(G_R)$, then $|PC| = 1, f(G_L) = f(G_R) = 0, \tau_R(G, PC) = 1$ and $\kappa_1(G_L) = \kappa_2(G_R)$.

Proof. By claim 1, $|PC| = 1$ or $\tau_R(G, PC) = f(G_R)$. Since $\tau_R(G, PC) \geq f(G_R)$, we have $|PC| = 1$. Let P be the only path in PC. By definition, P has one end vertex in the twin set of G_L. Without loss of generality, assume that the path-start of P is in the twin set of G_L. Consider the following three cases:

Case 1 The path-end of P is in V_L. Clearly, $\tau_R(G, PC) = f(G_R) = 0$. A contradiction.

Case 2 The path-end of P is in V_R but not in the twin set of G_R. It is easy to see that $\tau_R(G, PC) = f(G_R) = 1$ in this case, a contradiction.

Case 3 The path-end of P is in the twin set of G_R. In this case, $\tau_R(G, PC) = 1$ and $f(G_L) + f(G_R) = 0$. By Lemma 8.1.14, $|PC| = 1 = |V_L(PC)| - |V_R(PC)| + \tau_R(G, PC)$ and hence $= |V_L(PC)| = |V_R(PC)|$. By assumption, $V_L(PC)$ and $V_R(PC)$ are minimum-free-number twin-set path covers of G_L and G_R respectively. By induction hypothesis of statement (3), $\kappa_1(G_L) \geq |V_L(PC)| \geq \kappa_2(G_L)$ and $\kappa_1(G_R) \geq |V_R(PC)| \geq \kappa_2(G_R)$. By eq (eq.8) $\kappa_1(G_L) \geq \kappa_2(G_R)$. Assume, by contradiction, that $\kappa_1(G_L) > \kappa_2(G_R)$. Then, there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R, respectively, such that $|PC_L| - |PC_R| = 1$. we then can construct from PC_L and PC_R a new twin-set path cover of G without any free vertex, a contradiction. Therefore, $\kappa_1(G_L) = \kappa_2(G_R)$□

Claim 3. If $\kappa_1(G_L) = \kappa_2(G_R)$ and $f(G_L) = f(G_R) = 0$, then $|PC| = 1$ and $\tau_R(G, PC) = 1$.

Proof. By induction hypothesis of statement 3), $\kappa_1(G_L) \geq |V_L(PC)| \geq \kappa_2(G_L)$ and $\kappa_1(G_R) \geq |V_R(PC)| \geq \kappa_2(G_R)$. Since $\kappa_1(G_L) = \kappa_2(G_R)$, we have $|V_L(PC)| \leq |V_R(PC)|$. By Lemma 8.1.14, $|V_L(PC)| \geq |V_R(PC)|$. Hence, $|V_L(PC)| = |V_R(PC)|$. By Lemma 8.1.14, $|PC| = |V_L(PC)| - |V_R(PC)| + \tau_R(G, PC)$. Hence, $|PC| = \tau_R(G, PC) = 1$ □

Claim 4. $\kappa_1(G_L) - \kappa_2(G_R) + \tau_R(G, PC) = max\{1, \kappa_1(G_L) - \kappa_2(G_R) + fG_R)\}$.

Proof. By eqn (s.3.8), $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L) \geq 0$. There are the following two cases:
$Case\ 1.\ \kappa_1(G_L) - \kappa_2(G_R) > 0$. In this case, $\tau_R(G, PC) = f(G_R)$ by claim 2. Thus, $\kappa_1(G_L) - \kappa_2(G_R) + \tau_R(G, PC) = \kappa_1(G_L) - \kappa_2(G_R) + f(G_R) > 1$.
$Case\ 2.\ \kappa_1(G_L) - \kappa_2(G_R) = 0$. Since $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L)$ and $\kappa_1(G_L) = \kappa_2(G_R)$, we have $f(G_L) = 0$. Suppose that $f(G_R) \neq 0$. Then $\tau_R(G, PC) = f(G_R)$ by Claim 2 and hence $\kappa_1(G_L) - \kappa_2(G_R) + \tau_R(G, PC) = \kappa_1(G_L) - \kappa_2(G_R) + f(G_R) = f(G_R) \geq 1$. Thus, $\kappa_1(G_L) - \kappa_2(G_R) + \tau_R(G, PC) = f(G_R)$. On the other hand, suppose that $f(G_R) = 0$. By Claim 3, $|PC| = 1$ and $\tau_R(G, PC) = 1$. Hence, $\kappa_1(G_L) - \kappa_2(G_R) + \tau_R(G, PC) = 1$. □

Claim 5. $f(G) = f(G_L) + f(G_R)$ if $\kappa_1(G_L) - \kappa_2(G_R) > 0$; and $f(G) = max\{1, f(G_R)\}$ otherwise.

Proof. By (eq.8), $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L) \geq 0$. There are the following two cases:
$Case\ 1.\ \kappa_1(G_L) - \kappa_2(G_R) > 0$. By Claim 2, $\tau_R(G, PC) = f(G_R)$. By proposition 8.7, $\tau_L(G, PC) = \tau\big(G_L, V_L(PC)\big)$. Since PC is a minimum-free-number twin-set path cover of G and $V_L(PC)$ is a minimum-free-number twin-set path cover of G_L, we have $f(G) = \tau(G, PC) = \tau_L(G, PC) + \tau_R(G, PC) = \tau\big(G_L, V_L(PC)\big) + \tau_R(G, PC) = f(G_L) + f(G_R)$.
$Case\ 2.\ \kappa_1(G_L) - \kappa_2(G_R) = 0$. in this case, $f(G_L) = 0$. By proposition 2.2 $\tau_L(G, PC) = \tau\big(G_L, V_L(PC)\big)$. Since $V_L(PC)$ is a minimum-free-number twin-set path cover of G_L, we have $\tau\big(G_L, V_L(PC)\big) = f(G_L) = 0$. Hence, $f(G) = \tau(G, PC) = \tau_L(G, PC) + \tau_R(G, PC) = \tau_R(G, PC)$. Suppose $f(G_R) \neq 0$. Then, $\tau_R(G, PC) = f(G_R)$ by Claim 2. Hence, $f(G) = f(G_R)$. On the other hand, suppose that $f(G_R) = 0$. By Claim 3, we have $|PC| = 1$ and $\tau_R(G, PC) = 1$. Hence, $f(G) = \tau_R(G, PC) = 1$.
By arguments given in the above two cases, $f(G) = f(G_L) + f(G_R)$ if $\kappa_1(G_L) - \kappa_2(G_R) > 0$; and $f(G) = max\{1, f(G_R)\}$ if $\kappa_1(G_L) - \kappa_2(G_R) = 0$ □

Based upon the above claims, we prove statement (3) of Theorem 10 in case that G is formed from G_L and G_R by a pendant operation in the following.
(*Only if* part of $Case\ 3$) We now prove the only if part of statement (3) in case that G is formed from G_L and G_R by a pendant operation by showing that the following statement holds: if G has a minimum-free-number twin-set path cover of size k, then
$\kappa_1(G) = max\{1, \kappa_1(G_L) - \kappa_2(G_R) + f(G_R)\} \geq k \geq max\{1, f(G_L) + f(G_R), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R)\} = \kappa_2(G)$
By statement (1), assume that PC is a minimum-free-number twin-set path cover of G of size k with $V_L(PC)$ and $V_R(PC)$ are minimum-free-number twin-set path covers of G_L and G_R respectively.By Lemma 8.1.14, we have

$$|PC| = |V_L(PC)| - |V_R(PC)| + \tau_R(G, PC) \quad \text{(eq.9)}$$

By induction hypothesis of statement (3) we have

$$\kappa_1(G_L) \geq |V_L(PC)| \geq \kappa_2(G_L) \quad \text{(eq.10)}$$
$$\kappa_1(G_R) \geq |V_R(PC)| \geq \kappa_2(G_R) \quad \text{(eq.11)}$$

Combining (eq.9) - (eq.11) we get

$$\kappa_1(G_L) - \kappa_2(G_R) + \tau_R(G, PC) \geq |PC| \geq \kappa_2(G_L) - \kappa_1(G_R) + \tau_R(G, PC). \quad \text{(eq.12)}$$

Since $\tau_R(G, PC) \geq f(G_R)$, we have $|PC| \geq \kappa_2(G_L) - \kappa_1(G_R) + f(G_R)$. Clearly, $|PC| \geq 1$ and $|PC| \geq f(G_L) + f(G_R)$. Thus $|PC| \geq max\{1, f(G_L) + f(G_R), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R)\}$. By (eqn.12) and Claim 4, $max\{1, \kappa_1(G_L) - \kappa_2(G_R) + f(G_R)\} \geq |PC|$. This proves the only if part of statement (3) of Theorem 10 in case that G is formed from G_L and G_R by a pendant operation.

(*If* part of $Case\ 3$) We next prove the if part of statement (3) in case that G is formed from G_L and G_R by a pendant operation by showing that the following statement holds: if G has a twin-set path cover and $\kappa_1(G) = max\{1, \kappa_1(G_L) - \kappa_2(G_R) + f(G_R)\} \geq k \geq max\{1, f(G_L) + f(G_R), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R)\} = \kappa_2(G)$, then G has a minimum-free-number twin-set path cover of size k.

In the following, we prove that there exists a minimum-free-number twin-set path cover of G of size k. By statement (4), $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L) \geq 0$. Consider the following two cases:

$Case\ 1$. $\kappa_1(G_L) - \kappa_2(G_R) = 0$ and $f(G_R) = 0$. In this case, $f(G_L) = 0$, since $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L)$. It is easy to see that $\kappa_2(G_L) - \kappa_1(G_R) \leq \kappa_1(G_L) - \kappa_2(G_R) = \kappa_2(G_R) - \kappa_1(G_L) \leq 0$.
Therefore, $\kappa_1(G) = \kappa_2(G) = 1 = k$. By induction hypothesis of statement (3), there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R respectively such that $|PC_L| = \kappa_1(G_L)$ and $|PC_R| = \kappa_2(G_R)$. Since $f(G_L) = 0$, we have $\tau(G_L, PC_L) = 0$. Hence, $\tau(G_L, PC_L) + \tau(G_R, PC_R) \leq 2$ and $|PC_L| - |PC_R| \geq \tau(G_L, PC_L)$. By Lemma 8.1.15, we can construct from PC_L and PC_R a twin set path cover PC of G of size 1 such that $\tau(G, PC) = 1$. By Claim 5, PC is a minimum-free-number twin-set path cover of G.
$Case\ 2$. $\kappa_1(G_L) - \kappa_2(G_R) > 0$or $f(G_R) > 0$. clearly, $\kappa_1(G) = \kappa_1(G_L) - \kappa_2(G_R) + f(G_R)$. Since $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L)$, we have

$$\kappa_1(G_L) - \kappa_2(G_R) + f(G_R) \geq f(G_L) + f(G_R) \quad \text{(eq.13)}$$

Since $\kappa_1(G_L) \geq \kappa_2(G_L)$ and $\kappa_1(G_R) \geq \kappa_2(G_R)$, we have

$$\kappa_1(G_L) - \kappa_2(G_R) + f(G_R) \geq \kappa_2(G_L) - \kappa_1(G_R) + f(G_R) \quad \text{(eq.14)}$$

By (eq.1.3) and (eq.14), and $\kappa_1(G) \geq 1$, we get $\kappa_1(G) \geq \kappa_2(G)$. Thus we have

$$\kappa_1(G_L) - \kappa_2(G_R) + f(G_R) \geq k \geq max\{1, f(G_L) + f(G_R), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R)\} \quad \text{(eq.15)}$$

Since $\kappa_1(G_L) \geq \kappa_2(G_L)$ and $\kappa_1(G_R) \geq \kappa_2(G_R)$, the following statement holds: for any number k, where $\kappa_1(G_L) - \kappa_2(G_R) + f(G_R) \geq k \geq max\{1, f(G_L) + f(G_R), \kappa_2(G_L) - \kappa_1(G_R) + f(G_R)\}$, there exist k_L and k_R such that$\kappa_1(G_L) \geq k_L \geq \kappa_2(G_L), \kappa_1(G_R) \geq k_R \geq \kappa_2(G_R)$, and $k = k_L - k_R + f(G_R)$. By induction hypothesis of statement (3), there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R of size k_L and k_R respectively. Clearly, $k_L - k_R \geq f(G_L)$. Furthermore, $k = k_L - k_R \geq 1$ if $f(G_R) = 0$. Hence, $f(G_R) \neq 0$ or $k_L \neq k_R$. By Lemma 8.1.15, we can construct from PC_L and PC_R a twin-set path cover PC of G of size k

such that $\tau(G, PC) = f(G_L) + f(G_R)$. By Claim 5, PC is a minimum-free-number twin-set path cover of G.
By arguments given in the above two cases, the if part of statement (3) of theorem 10 holds true in case that G is formed from G_L and G_R by a pendant operation. This completes the proof of statement (3). □

Proof of statement (5) By statement (4), both G_L and G_R have twin-set path covers, $f(G_L) + f(G_R) \leq 2$, and $\kappa_1(G_L) - \kappa_2(G_R) \geq f(G_L)$. Following Claim 5 in the proof of statement (3), the statement holds true. □

Now that theorem 10 is proved, we shall proceed with showing how to decide whether a distance-hereditary graph has a Hamiltonian path. To our help we shall present a couple of lemmas and a corollary, and end up in theorem 11 that states an equivalence condition for the existence of a Hamiltonian path in a distance-hereditary graph. This theorem is also fundamental and will be proved.

Lemma 8.17 (Hung and Chang p.434):
Assume G is formed from G_L and G_R by a true-twin operation and $f(G_L) + f(G_R) = 2$. Then, G has a Hamiltonian path if and only if there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R, respectively, such that $|PC_R| = |PC_L| - f(G_L) + 1$.

Corollary 8.18 (Hung and Chang p.435):
Assume G is formed from G_L and G_R by a true-twin operation and $f(G_L) + f(G_R) = 2$. Then, G has a Hamiltonian path if and only if there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R, respectively, such that $|PC_L| = |PC_R| - f(G_R) + 1$.

Lemma 8.19 (Hung and Chang p.435):
Assume G is formed from G_L and G_R by a true-twin operation and G has a Hamiltonian path. Let P be a Hamiltonian path of G with maximum number of end vertices in the twin set of G and $PC = \{P\}$. Then, $V_L(PC)$ and $V_R(PC)$ are minimum-free-number twin-set path covers of G_L and G_R respectively.

8.1.4. Theorem 11

Theorem 11 (Hung and Chang p.436):
Assume G is formed from G_L and G_R by a true-twin operation. Then, G has a Hamiltonian path if and only if (1) $f(G_L) + f(G_R) \leq 1$ and $\kappa_2(G) = 1$ or (2) $f(G_L) + f(G_R) = 2$ and $max\{1, \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)\} = 1$.

8.1.5 Proof of theorem 11

Proof (Hung and Chang pp.436-437):
Only if part: Suppose G has a Hamiltonian path. We consider a Hamiltonian path P of G with maximum number of end vertices in the twin set of G. Let $PC = \{P\}$. By definition, PC is a twin set path cover of G. By lemma 8.19 $V_L(PC)$ and $V_R(PC)$ are minimum-free-number twin-set path covers of G_L and G_R respectively. There are two cases:
$Case\ 1$. P has at least one end vertex in the twin set of G. Then, PC is a twin-set path cover of G. By theorem 9, $f(G) = f(G_L) + f(G_R)$. By statement (3) of theorem 10, $f(G) = f(G_L) + f(G_R) \leq 1$ and $\kappa_2(G) = 1$.
$Case\ 2$. Neither of the two end vertices of P is in the twin set of G. Since $\tau\big(G_L, V_L(PC)\big) = f(G_L))$ and $\tau\big(G_R, V_R(PC)\big) = f(G_R)$, we have $f(G_L) + f(G_R) = 2$. By statement (3) of

theorem 10, we have

$\kappa_1(G_L) \geq |V_L(PC)| \geq \kappa_2(G_L)$ (eq.16)
$\kappa_1(G_R) \geq |V_R(PC)| \geq \kappa_2(G_R)$ (eq.17)

By lemma 8.17 $|PC_R| = |PC_L| - f(G_L) + 1$. By (eq.16) and (eq.17), $1 = |V_R(PC)| - |V_L(PC)| + f(G_L) \geq \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)$. On the other hand, $|PC_L| = |PC_L| - f(G_R) + 1$. By (eq.16) and (eq.17), $1 = |V_L(PC)| - |V_R(PC)| + f(G_R) \geq \kappa_2(G_L) - \kappa_1(G_R) + f(G_R)$. Therefore, $max\{1, \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)\}$=1.

If part:
$Case$ 1. $f(G_L) + f(G_R) \leq 1$ and $\kappa_2(G) = 1$. By definition, there exists a path cover of G of size 1. Hence, G has a Hamiltonian path.
$Case$ 2. $f(G_L) + f(G_R) = 2$ and $max\{1, \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)\}$=1. There are the following two subcases;
$Case$ 2.1. $\kappa_1(G_L) \geq \kappa_1(G_R)$. Since $1 \geq \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_1(G_R) - f(G_R) + 1 \geq \kappa_2(G_L)$. Suppose that $\kappa_1(G_L) \geq \kappa_1(G_R) - f(G_R) + 1$. Let $k_R = \kappa_1(G_R)$ and $k_L = k_R - f(G_R) + 1 \geq \kappa_2(G_L)$. By statement (3) of theorem 10, there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R, respectively, such that $|PC_L| = k_L$ and $|PC_R| = k_R$. By corollary 8.18 G has a Hamiltonian path. On the other hand, suppose that $\kappa_1(G_R) - f(G_R) + 1 \geq \kappa_1(G_L)$. Since $\kappa_1(G_L) \geq \kappa_1(G_R)$, we have $f(G_R) = 0$, $f(G_L) = 2$, and $\kappa_1(G_L) = \kappa_1(G_R)$. Since $1 \geq \kappa_2(G_R) - \kappa_1(G_L) + f(G_L), \kappa_1(G_L) \geq \kappa_2(G_R) + f(G_L) - 1 \geq \kappa_2(G_R) + 1$.Let $k_L = \kappa_1(G_L)$ and $k_R = k_L - 1$. Then, $\kappa_1(G_R) > k_R \geq \kappa_2(G_R)$. By statement (3) of theorem 10, there exist two twin-set path covers PC_L and PC_R of G_L and G_R, respectively, such that $|PC_L| = k_L$ and $|PC_R| = k_R$. By corollary 8.18, G has a Hamiltonian path.
$Case$ 2.1. $\kappa_1(G_L) < \kappa_1(G_R)$. Since, $1 \geq \kappa_2(G_R) - \kappa_1(G_L) + f(G_L), \kappa_1(G_L) - f(G_L) + 1 \geq \kappa_2(G_R)$. Let $k_L = \kappa_1(G_L)$ and $k_R = k_L - f(G_L) - 1$. Then $\kappa_1(G_R) > k_R \geq \kappa_2(G_R)$. By theorem 10, there exist minimum-free-number twin-set path covers PC_L and PC_R of G_L and G_R, respectively, such that $|PC_L| = k_L$ and $|PC_R| = k_R$. By Lemma 8.17, G has a Hamiltonian path. □

By Theorems 9 and 10, we have a recursive program for computing $f(G)$, $\kappa_1(G)$ and $\kappa_2(G)$ in linear time using the decomposition tree $DT(G)$ of a distance-hereditary graph G. By theorem 11, whether G has a Hamiltonian path can be determined in constant time if $f(G_L)$, $f(G_R)$, $\kappa_1(G_L), \kappa_2(G_L), \kappa_1(G_R)$ and $\kappa_2(G_R)$ are given. Hence, we conclude the following theorem:

Theorem 12 (Hung and Chang p.437):
The Hamiltonian path problem on distance-hereditary graphs can be solved in $O(m + n)$ time.

9 THE OTHER HAMILTONIAN PROBLEMS

Hung and Chang show the reduction of the $2HP, 1HP$ and Hamiltonian cycle problems on distance-hereditary graphs to the Hamiltonian path problem on the same class of graphs. We present the reduction techniques for $2HP$ and $1HP$ problems respectively the Hamiltonian cycle problems (Hung and chang p.438):

9.1 The $2HP$ problem and the $1HP$ problem

To find a Hamiltonian path starting at vertex u and ending at vertex v is known as the $2HP$ problem. Given a distance-hereditary graph $G = (V, E)$ where $u, v \in V$, and the wish to solve the $2HP$ problem, the process is as follows: one add two pendant vertices u' and v' to u and v respectively in order to obtain the distance hereditary graph $G' = (V', E') = (V \cup \{u', v'\}, E \cup \{uu', vv'\})$. Now, G has a Hamiltonian path from u to v if and only if G' has a Hamiltonian path. Thus the $2HP$ problem can be solved in linear time with this reduction technique. Similarly one can reduce the $1HP$ problem, which is to find a Hamiltonian path starting at vertex u, by adding a pendant vertex u' to u. G has a Hamiltonian path starting at u' if and only if G' has a Hamiltonian path

9.2 The Hamiltonian cycle problem

In order to solve the Hamiltonian cycle problem on a given distance-hereditary graph $G = (V, E)$ one add a vertex u' as a false twin to a vertex u in G to obtain G', that is $N(u') = N(u)$ in G'. Then G has a Hamiltonian cycle if and only if G' has a Hamiltonian path starting at vertex u and ending at vertex u'.

9.3 Theorem 13

Theorem 13 (Hung and Chang p.438):

The $1HP, 2HP$ and Hamiltonian cycle problem on distance-hereditary graphs can be solved in $O(m + n)$ time.

10 A SOLUTION TO THE HAMILTONIAN CYCLE PROBLEM

In this part we will give, as to knowledge of the author's of this papers, a new idea regarding the existence of Hamiltonian cycles in a graph G. We shall give a necessary and sufficient condition for Hamiltonicity. This theorem is based on the introduced concept of k-partitioning around a vertex in a graph G. Before we prove the theorem we will give a lemma with proof. We also give an algorithm, based on that condition, that give a "yes" or "no" answer to the problem of a graph's being Hamiltonian. Also one can find at least one Hamiltonian cycle if it exists.

We will later on state a theorem on summation of cycles in a k-partition of a graph, followed by a few corollaries and a conjecture.

10.1 k-PARTITION AROUND A VERTEX

Definition 10.1: We say that a connected graph $G = \big(V(G), E(G)\big)$ with $(|V(G)|, |E(G)|) = (n, m)$ is k-partitioned around v_0 and that $G_1, \dots, G_k$ is a k-partition of G,where degree $d(v_0) = d \geq 2$ and $k = d - 1$, if the following conditions are satisfied for some ordered k-tuple of subgraphs $(G_1, \dots, G_k)$ of G and some order $u_1, \dots, u_{k+1}$ of $N(v_0)$:

1. $\bigcup_{i=1}^{k} V(G_i) = V(G)$
2. $G_i \cap G_{i+1} = (\{v_0, u_{i+1}\}, \{v_0 u_{i+1}\})$ for $1 \leq i \leq k-1$ when $k \geq 2$
3. $V(G_i) \cap N[v_0] = \{v_0, u_i, u_{i+1}\}$ for $1 \leq i \leq k$,
4. $G_i \cap G_{i+r} = (\{v_0\}, \emptyset)$ for $1 \leq i \leq k-2$ and some $r > 1$ such that $i + r \leq k$, when $k \geq 3$

Remark: each G_i is not necessarily an induced subgraph of G.

Theorem 14: Assume $G_1, \dots, G_k$ is a k-partition around v_0 in G. Then the following holds:

$n = \sum_{i=1}^{k} |G_i| - k + 1 - \sum_{i=1}^{k} \sum_{j=i+1}^{k} |G_i \cap G_j| + \binom{k}{2}$ (eq.10.1)

Proof: $\sum_{i=1}^{k} |G_i|$ sums the number of vertices in each subgraph. However, this means that v_0 is counted k times, henc e the term $(-k)$, whence to have exactly one v_0 the term 1 is necessary. Moreover, that sum also includes the number of vertices in pairwise intersections of the subgraphs and hence the term $-\sum_{i=1}^{k} \sum_{j=1}^{k} |G_i \cap G_j|$, in which v_0 is counted, and therefore also subtracted, once for every pair. Therefore the number of pairs, $\binom{k}{2}$, has to be added in order count v_0 exactly once.

10.1.1 Lemma 10.1

We shall give a lemma with proof. This lemma consists of two parts depending on the length of a cycle C. In part (a) , $|C| > 3$ and in part (b) , $|C| = 3$, and is then used in the proof of theorem 15.

Lemma 10.1(a): Suppose $C, |C| > 3$, is a chord-free cycle where v, x, y are vertices in C and $N[v] = \{v, x, y\}$, and consider a path P with vertex set $V = \{v, x, y\}$. Then P is an induced subpath of C with ordering $P = y - v - x$, $x \notin N(y)$, and there exists a chord-free path from y to x not containing v.

Proof: Let $P = v - y - x$ (or $P = v - x - y$)be a path in C. Since both x and $y \in N(v)$ there is a chord vx in P and hence in C. But C is chord-free and therefore such P cannot exist in C. On the other hand, if $P = y - v - x$ is a subpath of the chord-free cycle C we have that there is no edge xy, and hence there must be a path P_{xy} from x to y along C. Since C is chord-free, so is P_{xy}.

Lemma 10.1(b): If $C, |\mathrm{C}| = 3$, is a cycle where v, x, y are vertices in C and $N[v] = \{v, x, y\}$, then any order of the vertices v, x and y defines a path in C.

Proof: This is easily seen or verified by trying all cases.

10.2. THEOREM 15, A NECESSARY AND SUFFICIENT CONDITION FOR HAMILTONICITY

The following theorem states a necessary and sufficient condition for a graph to be Hamiltonian.

10.2.1 Theorem 15

Theorem 15: Let $G = \big(V(G), E(G)\big)$ be a graph. Then the following three conditions are equivalent:

1) G is Hamiltonian
2) $\forall\, v_0 \in V(G)$ there exists a k-partition around v_0, where each subgraph is a cycle.
3) $\exists\, v_0 \in V(G)$ such that there exists a k-partition around v_0, where each subgraph is a cycle.

We begin by proving that a Hamiltonian cycle implies a k-partition where each part is a cycle, i.e. that 1) ⇒ 2).

10.2.2 Proof of theorem 15

Proof: Assume there is a Hamiltonian cycle C in G, and that $d(v_0) \geq 2$. Without loss of generality we can let $C = v_0 v_1 \dots v_{n-1} v_0$.

First consider the case when vertex v_0 has degree $d(v_0) = 2$, hence $k = 1$. In this case the Hamiltonian cycle C is trivially the only part in a k-partition.

Next assume that vertex v_0 has degree $d(v_0) = d = k + 1 \geq 3$. Let G' be a subgraph of G where $N_{G'}(v_0) = \{u_1, \dots, u_d\}$, where $u_1 = v_1$, $u_d = u_{k+1} = v_{n-1}$ and $u_a = v_{i_a}$ where $1 < i_a < \dots < n-1$. Consider $G' = (V, E(C) \cup \{c_a\}),\ a = 2, \dots, d-2$ where the edges $c_a = \{v_0 u_a\}$ are those indicent to v_0 in G but not contained in C. Those edges are chords of C in G'.

The two neighbors of v_0 that (except v_0 itself) are the first respectively last vertices along the closed path C are u_1 and u_{k+1} in G', i.e. v_1 and v_{n-1}.

Let P_a be the path between u_a and u_{a+1}, $a = 1, \dots, d-1$. Hence we have $P_1, \dots, P_{d-1}$. We can thus write $C = v_0 u_1 P_1 u_2 P_2 u_3 \dots u_{d-2} P_{d-2} u_{d-1} P_{d-1} u_d v_0$ with chords $v_0 u_i, i = 2, \dots, d-1$. Hence there are subcycles

$C_1 = \{v_0 u_1 P_1 u_2 v_0\}, C_2 = \{v_0 u_2 P_2 u_3 v_0\}, \dots, C_{d-2} = \{v_0 u_{d-3} P_{d-2} u_{d-2} v_0\},\ C_{d-1} = \{v_0 u_{d-2} P_{d-1} u_d v_0\}$ all of which are chord-free in G' (but generally not in G).

We shall here show that $(C_1, \dots, C_k)$ is a k-partition around v_0 in G, by checking the conditions of the definition:

Condition (1): We have that $\bigcup_{i=1}^{k=d-1} V(C_i) = V(G') = V(G)$ and thus condition 1 is satisfied.

Condition (2): Consider the intersection $C_i \cap C_{i+1} = \{v_0 u_i P_i u_{i+1} v_0\} \cap \{v_0 u_{i+1} P_{i+1} u_{i+2} v_0\} = (\{u_{i+1}, v_0\}, \{u_{i+1} v_0\})$. This satisfies condition 2.

Condition (3): Consider the intersection $C_i \cap N[v_0] = \{v_0 u_i P_i u_{i+1} v_0\} \cap \{v_0, u_1, \dots u_{k+1}\} = (v_0, u_1, u_{i+1})$ since $i = 1, \dots, k+1 = d$. This satisfies condition 3.

Condition (4): Consider, for $r > 1$, the intersection

$C_i \cap C_{i+r} = \{v_0 u_i P_i u_{i+1} v_0\} \cap \{v_0 u_{i+r} P_{i+r} u_{i+r+1} v_0\} = (\{v_0\}, \emptyset)$. Condition 4 is thus satisfied and this settles the if-part.

2) ⇒ 3) is trivial.

Next we prove 3) ⇒ 1), that is, if $G_1, \dots, G_k$ is a k-partition of G and each G_i is a cycle, then G is Hamiltonian.

First consider the case when $k = 1$. Then trivially G_1 is a Hamiltonian cycle in G.

Next let $k \geq 2$.

Assume $C_1, C_2, .., C_k$ is a k-partition around v_0 in G where each part is a cycle and $N(v_0) =$

$\{u_1, u_2, \dots, u_{k+1}\}$. Then by definition we have $\bigcup_{i=1}^{k} V(C_i) = V(G)$ by condition 1, $C_i \cap C_{i+1} = (\{v_0, u_{i+1}\}, \{v_0 u_{i+1}\})$ by condition 2. By condition 3 we have $C_i \cap N[v_0] = \{v_0, u_i, u_{i+1}\}$. By condition 4 we have that $C_i \cap C_{i+r} = (\{v_0\}, \emptyset)$ when $r > 1$. Without loss of generality we can consider the subgraph G' where $C_1, C_2, .., C_k$ are chord-free. Each cycle can be written C_i=$v_0 u_i v_1 \dots v_j u_{i+1} v_0, 1 < j < |C_i| - 3$.

By condition 3 and lemma, we can start a walk at v_0 along C_1 first meeting u_1 at distance 1 from v_0, until we arrive at u_2 by following the subpath $P_{u_1 u_2} = u_1 v_1 \dots v_{|C_i|-3} u_2$ of C_1, where $P_{u_1 u_2}$ contains all vertices of C_1 but v_0. From u_2 we could, by condition 2, follow the edge $u_2 v_0$ to v_0. However, by condition 3 and lemma we can instead follow the subpath $P_{u_2 u_3}$ of C_2 and continue from u_2 and reach u_3. Also by condition 4, this is the only choice if we do not walk back along C_1 or return to v_0 through $u_2 v_0$. Similarly when $k \geq 3$: instead of returning to v_0 at each u_i for $1 < i \leq k$, we can continue along every path $P_{u_i u_{i+1}}$until we reach u_{k+1} which is the last neighbor of v_0. We close the path into a cycle by returning to v_0 from the last vertex u_{k+1} in C_k.

Thus we have started at v_0, walked through all vertices of $C_1, \dots, C_k$ (exactly once). Hence the k-partition has a Hamiltonian cycle, and since by assumption, $\bigcup_{i=1}^{k} V(C_i) = V(G)$ there is a Hamiltonian cycle in G.

Moreover, by the arbitrary choice of v_0 we have shown that if there is a k-partition around v_0 with $d(v_0) \geq 2$ such that G has a Hamiltonian cycle, then there is a k-partition around all vertices v with $d(v) \geq 2$ in G such that there is a Hamiltonian cycle. ■

10.2.3. On the number of vertices and lengths of cycles

Theorem 16. Let $C_1, \dots, C_k$ be cycles. If $C_1, \dots, C_k$ is a k-partition around v_0 of a graph G with n vertices, then

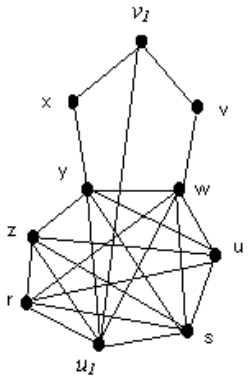
$$\sum_{i=1}^{k} |C_i| - 2(k-1) = n \qquad \text{(eq.10.2)}$$

Proof: Induction on k. ■

Corollary 1. For a cycle C of maximum length in a k-partition of a graph G, the following condition holds: $|C| \leq n - d(v_0) + 2$.

Proof: By letting all cycles but one be of shortest possible length, i.e. $|C_i| = 3$ for $k = 1, \dots, (k-1)$ we can write (eq.10.2) as $3(k-1) + |C_k| - 2(k-1) = n \Leftrightarrow |C_k| = n - (k-1) = n - (d(v_0) - 1 - 1) = n - d(v_0) + 2$. ■

Remark: this is a theoretical upper bound and lacks significance for large n and small $d(v_0)$. However, by a clever choice of v_0 in some graphs the length of a longest cycle in a k-partition can be strictly less than $n - d(v_0) + 2$. The graph in figure 10.1 shows such a case. We have a k-partition around $v_0 = u_1$ with $d(v_0) = 6$, $k = 5$, and $C_1 = u_1, r, z$, $C_2 = u_1, z, y$, $C_3 = u_1, y, x, v_1$, $C_4 = u_1, v_1, v, w$ and $C_5 = u_1, w, u, s$. We see that $n = 10$, $d(u_1) = 6$ and hence $n - d(v_0) + 2 = 6$, but the longest cycle in the k-partition around u_1 in G is of length four.



G fig 10.1

Corollary 2. If a connected plane graph G on n vertices with $d_{min} \geq 2$ satisfies the following two conditions:(i) n is odd,(ii) every induced cycle in G is of even length, then G is not Hamiltonian.

Proof: since every induced cycle in G is of even length, so is any cycle in any k-partition of cycles of G. By theorem 4 the cycles sums up to an even number which is violated by condition 1. Hence condition 1 and condition 2 are mutually exclusive in a plane Hamiltonian graph. ■

Remark: the Herschel graph which is known to be non-hamiltonian has $n = 11$ vertices whence every induced cycle is of length 4.

Corollary 3. If there exist two induced cycles in G that have at least three vertices in common in a plane graph G with $d_{min} \geq 2$, then G is not Hamiltonian.
Proof: If G is Hamiltonian then there exists a k-partition of G where each part is a cycle. Then if there exists two cycles C_i and C_{i+1} such that $V(C_i \cap C_{i+1}) = \{v_0, x_1, \dots, x_j, u_{i+1}\}$ for some vertex $x_k, 1 \leq k \leq j$ which is absurd in view of condition 2 of the definition of k-partition ■

Conjecture 1: Given a set of cycles $C = \{C_1, \dots, C_k\}$. Then C is a k-partition of a graph G with n vertices, and hence G is Hamiltonian if and only if

$$\sum_{i=1}^{k}\sum_{j=i+1}^{k}|C_i \cap C_j| = \frac{(k-1)(k+1)}{2} \qquad \text{(eq.10.3)}$$

Putting (eq.10.1)=(eq.10.2), algebraic manipulations will yield (eq.10.3)

10.3 ALGORITHM CYCLE-k-PARTITION RECOGNITION

This algorithm finds a k-partition around v_0 of cycles in a graph G. In order to do so, we first need to find a set of cycles starting from an arbitrary v_0. By corollary 1 we need not find a cycle of length greater than $n - d(v_0) + 2$, hence what we are looking for is all cycles starting at v_0 and has length at most $n - d(v_0) + 2$. This is done by Algorithm SingleSourceCycleSearch(v_j). Next we need to match the cycles together in a number of combinations based on the definition of k-partition and some results given above. We do that in Algorithm "Match-k-cycles".
This algorithm is probably exponential in the worst case, but experience suggests that on small graphs one can expect to find a solution in reasonable time. Moreover, there may be graphs with particular qualities such that the algorithm runs efficiently. Such qualities could be for example diameter and/or density under certain constraints or any other graph invariant with or without constraints. This is subject to further research.

10.3.1 Algorithm SingleSourceCycleSearch(v_j), SSCS(v_j)

This algorithm finds all paths between the neighbors $\{u_1, \dots, u_{d(v_0)}\}$ of a chosen vertex v_0. In each loop i, the algorithm finds every path from u_i to u_j, $1 \le i < j \le d(v_0)$. When all paths from u_i to u_j are found one do not want to find them again in backwards order. Therefore in the next loop, where we search for paths from u_{i+1} to u_j one can reduce the graph at hand with u_i (thus a fewer number of computational steps is needed as each loop is completed). Any such path is easily seen to be a subpath of a cycle starting and ending at v_0 since u_i and u_j are neighbors of v_0.

$F_1 =$ {forbidden vertices through the i: th-loop, v_0 and u_i},
$F_2 =$ {vertices already on the path},Whenever a vertex is removed from F_2 it is put back to its origin. $A_0 = N(v_0) = \{u_1, \dots, u_{d_0}\}$ for some ordering of $N(v_0)$. $A_1 = V(\backslash(F_1 \cup F_2 \cup A_0)$, this changes dynamically as vertices are moved in and out of F_2. $A = A_1 \cup A_0$. $T =$ { a tree rooted at v_0}, whenever we write $v_{j+1} \mapsto T$ we mean that v_{j+1} is added as a child of v_j. $L_{d_T} =$ {vertices on distance d_T from v_0 in T_i}, $B_\alpha^\beta = \{v_0, v_1, \dots, v_{d_T}\}$ - the first β vertices of the α: th branch of T_i, the superscript index of B_α^β changes when a vertex is added to B_α^β. $\mathcal{C} =$ {the set of cycles found in SSCSearch(v)}.

We use left arrows to assign a value a to a variable x, $x \leftarrow a$. We use right arrows from bar to indicate that a variable x becomes an element in a set S, $x \mapsto S$. Whenever we switch between routines by "call subroutine" or "go to algorithm step x" the current routine, which we leave, is stopped.

Input: $G = (V, E)$ with $(|V|, |E|) = (n, m)$, all vertices are labeled.
Output: all cycles in G starting from a vertex v_0 with degree $d(v_0) = d_0 \ge 3$ with length at most $n - d + 2$.

1. Initialize

$\alpha \leftarrow 1; \beta \leftarrow 1; r \leftarrow 1; d_T \leftarrow 1; B_\alpha^{d_T} \leftarrow \emptyset;\ F_1, F_2 \leftarrow \emptyset; A_0 \leftarrow \emptyset; A_1 = V\backslash(F_1 \cup F_2 \cup A_0);$
$i \leftarrow 1; L_{d_T} \leftarrow \emptyset$ for $d_T = (1, \dots . (n - d + 2)$;

2. Choose any vertex v with degree $d(v) \ge 3$ and $v_0 \leftarrow v$, order $N(v_0)$ in an arbitrary order $\{u_1, \dots, u_{d_0}\}$ and $A_0 \leftarrow \{u_1, \dots, u_{d_0}\}$;

3. $v_0 \mapsto F_1$;

4. $T_i \leftarrow v_0$;

5. For $i = 1, \dots, (d_0 - 1)$ **Do**

Input $=(G\backslash\cup_{\mathbf{0}}^{\boldsymbol{i-1}} u_s);\ A_0 \leftarrow \{u_1, \dots, u_{d_0}\}\backslash(\cup_{\mathbf{0}}^{\boldsymbol{i-1}} u_s)$;

remove x s.t. $d_{G\backslash\cup_{\mathbf{0}}^{\boldsymbol{i-1}} u_s}(x) = 1$ from $G\backslash\cup_{\mathbf{0}}^{\boldsymbol{i-1}} u_s$;

6. $u_i \mapsto T_i$ as a child of v_0;

7. $v_0, u_i \mapsto B_\alpha^\beta$;

8. $u_i \mapsto L_{d_T}$;

9. $u_i \mapsto F_1$;

10. $v_j \leftarrow u_i$;

11. **If** $N(v_j) \cap A \neq \emptyset$ **;**

Choose $w \in N(v_j) |\ w \notin F_2$

12. $v_{j+1} \leftarrow w$;

13. $d_T \leftarrow d_T + 1$;

14. **If** $d_T = n - d + 2$ **then**

Go to Backtrack(T, w);

End If;

15. $v_j \mapsto F_2$;
16. $v_{j+1} \mapsto T_i$;
17. $v_{j+1} \mapsto B_\alpha^\beta$;
18. $B_\alpha^{\beta+1} \leftarrow B_\alpha^\beta$;
19. **If** $d_T \geq 2$ **then**
20. **If** $v_{j+1} \in A_0$ **Then**
21. **For all** $w \in L_{d_T} \cap F_2 | w \notin A_0$
22. Remove w from F_2;
End For;
Else
23. **For all** $w \in L_{d_T} \cap F_2$
24. Remove w from F_2;
End for;
End If;
25. $v_{j+1} \mapsto L_{d_T}$;
EndIf
26. **If** $v_{j+1} \in A_0$
27. $C_r \leftarrow B_\alpha^\beta$;
28. $C_r \mapsto \mathcal{C}$;
29 $r \leftarrow r + 1; v_{j+1} \mapsto F_2$;
30. Backtrack to v_j;
31. $d_T \leftarrow d_T - 1$;
32. **If** $N(v_j) \cap A \neq \emptyset$ **Then**
33. $\alpha = \alpha + 1$;
34. $B_\alpha^\beta \leftarrow B_{\alpha-1}^{\beta+1} \backslash \{ v \in (\cup_{m=d_T+1}^{n-d+1} L_m) \}$;
EndIf
35. **Go to** SSCS (v_j) step 11;
End If
36. **Else If** $v_{j+1} \in A_1$ **then**
37. $j + 1 \leftarrow j$;
38. **Go to** SSCS (v_j) step 11;
End If
39. **Else If** $N(v_j) \cap A = \emptyset$
40. **If** v_j=u_i **Then**
41. $\alpha \leftarrow \alpha + 1$;
42. $B_\alpha^\beta \leftarrow v_0$;
43. $L_{d_T} \leftarrow \emptyset, d_T = (1, \ldots . (n - d + 2)$;
44. $F_2 \leftarrow \emptyset$;
45. $i \leftarrow i + 1$;
46. **Go to** SSCS(v_j) step 4;
End If
47. **Go to** Backtrack(T, v_j);
End If
48. End For
End Algorithm SSCS(v)

10.3.1.1 Procedure Backtrack(T, v_j)

Backtrack(T, v_j);

1. $v_{j-1} \leftarrow p_{T_i}(v_j)$;
2. **If** $v_{j-1} = v_0$ **then**
3. $\alpha \leftarrow \alpha + 1$;
4. $B_\alpha^\beta \leftarrow v_0$;
5. $L_{d_T} \leftarrow \emptyset, d_T = (1, \ldots . (n - d + 2)$;
6. $F_2 \leftarrow \emptyset$;
7. $i \leftarrow i + 1$;
8. **Go to** sscs(v) step 4;
End If;
9. $d_T \leftarrow d_T - 1$;
10. **For all** $w \in \{(L_{d_T+1} \cup L_{d_T+2}) \cap F_2\}$; **Do**
11. **If** $w \notin N_{T_i}(v_{j-1})$ **Then**
12. Remove w from F_2;
End If;
End For;
13. **For all** $w \in \{c_{T_i}(v_{j-1})\}$ **Do**
14. $w \mapsto F_2$; //vertices that were removed from F_2 in step 20 or step 22 of sscs may be available in backtrack
End For;
15. **If** $N(v_{j-1}) \cap A \neq \emptyset$
16. $\alpha = \alpha + 1$;
17. $B_\alpha^\beta \leftarrow B_{\alpha-1}^{x=last\ stored\ index} \backslash (\cup_{m=d_T+1}^{n-d+1} L_m)$;
18. $v_{j-1} \mapsto B_\alpha^\beta$;
19. $j \leftarrow j - 1$;
20. **Go to** SingleSourceCycleSearch(v_j) step 11;
End if
21. **ElseIf** $N(v_{j-1}) \cap A = \emptyset$
22. $j \leftarrow j - 1$;
23. **Go to** Backtrack(T_i, v_j) step 1;
End If;

10.3.2 Algorithm Match-k-cycles

When algorithm SSCS(v_j) is done, there is a set of cycles $\{B_\alpha^\beta\}$ starting and ending at v_0. We want to test them for being parts of a k-partition of G. The worst case is, when there is no further information and, either when there is no Hamiltonian cycle in G, or when there exists exactly one – which we find in the very last set of checked combinations of cycles. We will in those cases have checked all combinations of k cycles among all α cycles and test each combination against the definition of k-partition. However, there are some "pruning" information to be obtained. By theorem 4, there is a limit to which of the cycles we need to test together, namely k cycles, $C_i, i = 1, \ldots, k$, that sum up to $n + 2(k - 1)$ under the constraint $3 \leq C_i \leq n - d + 2$ (by corollary 1). For instance, say that we have $n = 11, k = 3$, then $n + 2(k - 1) = 15$. All partitions of 15 into three parts are the following: (3,3,9), (3,4,8), (3,5,7), (3,6,6,), (4,4,7), (4,5,6), (5,5,5). Therefore, in that case, we would want to test all triplets of cycles $(C_{i_1}, C_{i_2}, C_{i_3})$ whose lengths matches the partitioning of the number 15 into three parts against the definition of a k-partition.
More generally stated in pseudo-pseudo code:
Input: $C = \{the\ set\ of\ cycles\ found\ in\ SSCSearch(v)\}$

Output: A Hamiltonian cycle if and only if there is one
For all k-tuples $(|C_{i_1}|,..,|C_{i_k}|)$ from the output of algorithm SSCS(v_j), corresponding to a partition of the number $(n - d(v_0) + 2)$ into k parts,
Do Check each k-tuples $(C_{i_1},..,C_{i_k})$ against the definition of k-partition

If the k-tuple $(C_{i_1},..,C_{i_k})$ satisfies the conditions of k-partition

Then G is Hamiltonian and has a Hamiltonian cycle

$C_{n_p} = v_0 u_{i_1} P_{i_1} u_{i_2}, \dots, u_{i_k} P_{i_k} u_{i_{k+1}} v_0$ for $1 \leq p \leq N$, N is the number of tested k-tuples $(C_{i_1},..,C_{i_k})$.

11 CONCLUSIONS AND DISCUSSION

We have seen what distance-hereditary graphs are; what characterizes them and how they are constructed. We have also seen how to solve the Hamiltonian problem, on this class of graphs, in linear time using a linear program for computing "the constants of the $DT(G)$". We have learned that given those constants of a graph and its decomposition tree, we can solve the problem in constant time. Moreover have we given a characterization of Hamiltonian graphs and presented an algorithm that is based on that very characterization.

The achievement of finding a linear-time algorithm for the Hamiltonian problem, albeit for a special class of graphs, is both interesting and admirable. Not at least in the context of the vast and rigorous theoretical foundation it relies on. Also, in view of the problem itself belonging to the class of NP-complete problems one must appreciate that there is a solution in linear time at all. Furthermore, in view of the algorithm presented in chapter 10, it is striking that the effort put in to find and describe the theoretical foundation seems to stand in reversed proportion to the efficiency of the algorithm. A lot of theory yields an efficient algorithm, not so much theory yields a much less or even inefficient algorithm. It holds in this case anyway.

As to the theory of distance-hereditary graphs one could, as attempted in theorem 8, study the very twin set. Outstanding questions regarding theory of Hamiltonian graphs as given in chapter 10, are quite a few. Given the knowledge of this substructure, is there a way to give a meaningful recursive definition of this class of graphs similarly to what is done with distance-hereditary graphs and cographs for example. Can one use the summation rule of theorem 16 to define a (Abelian) group in a meaningful way and why would that be of interest? Attempts have been made. By giving a corresponding formula for the number of edges: $\sum_{i=1}^{k}|C_i| - (k-1) = m$ and consider condition 2 of the definition as a group operation denoted $\oplus$. Then for $k = 2$ it holds that for $G_1$with $(|V|, |E|) = (n_1, m_1)$ and $G_2$with $(|V|, |E|) = (n_2, m_2)$ it holds that $G_1 \oplus G_2 = (n_1 + n_2 - 2, m_1 + m_2 - 1)$. Since it is effectively addition of numbers the operation is commutative and associative and there is an identity element (2, 1). So far so good, if one also think of the operation of actually put two graphs together – which also is a graph and hence the closure condition holds. However, this means trouble when we come to the inverse element: it is found to be $(4 - n, 2 - m)$ for any graph with $(|V|, |E|) = (n, m)$. For $n > 4, m > 2$ we have a negative number of vertices and edges. Could one define graphs in a meaningful way with such characteristics? Would group theory be applicable or other fields of algebra. Far fetched,yes, but so must for example the finding of quaternion algebra also have been…

Moreover, outstanding questions on cycle-k-partiton recognition algorithm are quite a few: does it solve the problem for all instances of the problem? What exactly is the order of its time complexity? What ways are there to refine the given one? Are there other search-algorithms of higher efficiency given the knowledge of this substructure in Hamiltonian graphs? On which graphs does it perform at its best? In average? What is the optimal choice of start vertex? Those are questions to be answered.

REFERENCES

Backelin and Timonen; Jörgen Backelin Supervisor, Meetings and discussions (2008-2013)

Bandelt and Mulder: H.J. Bandelt, H.M. Mulder, Distance-hereditary graphs. J. Combin. Theory Ser. B 41 (1986) pp.182-208.

Chang et al.: M.S. Chang, S.Y. Hsieh, G.H. Chen: Dynamic Programming on Distance-Hereditary Graphs, Lecture Notes in Computer Science, Vol. 1350, Springer, Berlin/New York, 1997, pp.344-353

Chuang-Chieh Lin: Joseph Chuang-Chieh Lin: Talk on Computation Theory Laboratory, Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan. November 17, 2009 Supervisor: Professor Maw-Shang Chang. The notes from the talk are available on the following web page: http://idv.sinica.edu.tw/josephcclin/paper/cographs.pdf (2013-05-28)

Corneil et al.: D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, SIAM J. Comput. 14 (4) (1985) 926-934.

Damiand et al.: G. Damiand, M. Habib, C. Paul: A simple paradigm for graph recognition: application to cographs and distance-hereditary graphs, Theoretical Computer Science 263 (2001) pp. 99-111

Golumbic: Martin C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, First edition, Academic Press, New York, 1980

Gould: Ronald J. Gould, Graph Theory, Benjamin/Cummings Publishing Co., Menlo Park, CA, 1988

Hammer and Maffray: P.L. Hammer, J. Maffray, Completely Separable graphs, Discrete Appl. Math. 27 (1990) pp.85-99

Wolfram Mathworld: http://mathworld.wolfram.com/about/author.html, http://mathworld.wolfram.com/Cograph.html (2013-05-28)

with subreferences:

- Brandstadt, A.; Le, V. B.; Spinrad, J. P. Graph Classes: A Survey. Philadelphia, PA: SIAM, 1999.
- Brouwer, A. E.; Cohen, A. M.; and Neumaier, A. New York: Springer-Verlag, p. 435, 1989.
- Corneil, D. H.; Lerchs, H.; and Stewart Burlingham, L. "Complement Reducible Graphs." Discr. Appl. Math. 3, 163-174, 1981.
- Sloane, N. J. A. Sequence A000084/M1207 in "The On-Line Encyclopedia of Integer Sequences."
- Weisstein, E. W. "Re: Cographs." Oct. 9, 2003a. http://listserv.nodak.edu/scripts/wa.exe?A2=ind0310&L=graphnet&P=R743.
- Weisstein, E. W. "Cographs <=> Series-Parallel Networks." Oct. 23, 2003b. http://listserv.nodak.edu/scripts/wa.exe?A2=ind0310&L=graphnet&P=R1929

Wikipedia: Cographs http://en.wikipedia.org/w/index.php?title=Cograph&oldid=552342134

with subreferences:

- Jung, H. A. (1978), "On a class of posets and the corresponding comparability graphs", *Journal of Combinatorial Theory, Series B* **24** (2): 125–133
- Corneil, D. G.; Lerchs, H.; Burlingham, L. Stewart (1981), "Complement reducible graphs", *Discrete Applied Mathematics* **3** (3): 163–174,
- Sumner, D. P. (1974), "Dacey graphs", *J. Austral. Math. Soc.* **18** (04): 492–502
- Burlet, M.; Uhry, J. P. (1984), "Parity Graphs", *Topics on Perfect Graphs*, Annals of Discrete Mathematics **21**, pp. 253–277.

Wilf: Herbert S. Wilf, “Algorithms and Complexity”, internet edition 1994
http://www.math.upenn.edu/~wilf/AlgComp3.html (2013-05-28)

Appendix 1

A1.1 One-vertex-extensions

We shall use the one-vertex-extension operations in order to construct a distance hereditary graph, but first let us recall the set of rules for the operations:

Consider the induced subgraph $G' = (V', E')$ of $G = (V' U\{x\}, E)$, and a vertex x' in G'. If G' were extended to G by adding a new vertex x such that:

- $i)$ $N(x) = \{x'\}$, we say that we were attaching a pendant vertex x to x' and denote the extension by $x(P)x'$
- $ii)$ $N(x) = N(x')$, we say that x' and x are false twins, and denote the extension by $x(F)x'$
- $iii)$ $N[x] = N[x']$, we say that x' and x are true twins, and denote the extension by $x(T)x'$

We start with K_2(which by theorem 3.iv contains a pair of twins or a pendant vertex and hence either $v_2(P)v_1$or $v_2(T)v_1$, the complete graph on two vertices v_1 and v_2 and we shall employ the one-vertex-extensions in the following (random) order: we begin with a false twin operation, followed by a pendant operation and ending with another false twin operation.

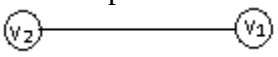


fig 1.1

To this graph we add the vertex v_3 as a false twin to v_1 (i.e. they have the common open neighborhood)...

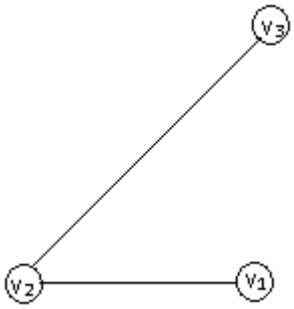


fig 1.2

…followed by the addition of another pendant vertex v_4 to v_2, thus v_2 is its only neighbor…

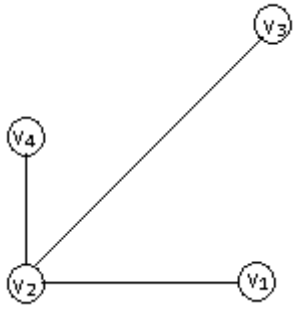
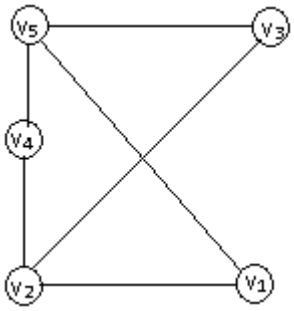


fig 1.3

… and we finish the construction of G_1 by adding, v_5 as a false twin to v_2:



G_1 fig 1.4

G_1 is a distance hereditary graph.

Next an example with the following order of operations, starting from K_2:
True twin, pendant, false twin and finally another true twin operation.

We begin with K_2on vertices u_1 and u_2…

fig 1.5

…and add $u_3$being a true twin to u_1…

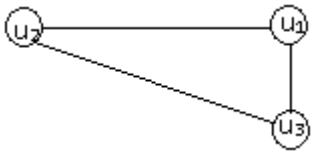


fig 1.6

…then make u_4 a pendant vertex attached to u_1…

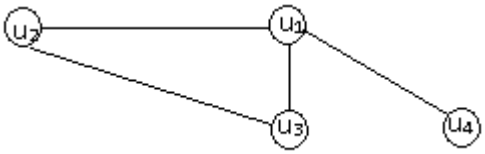


fig 1.7

…followed by u_5 being a false twin to u_1, having the same open neighborhood as u_1…

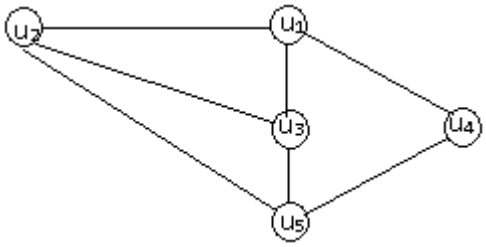
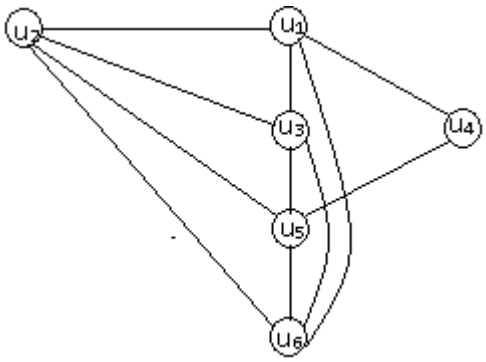


fig 1.8

…and we finish by attaching u_6 by at true twin to u_3…



G_2 fig 1.9

..giving us G_2, a distance hereditary graph.

Appendix 2

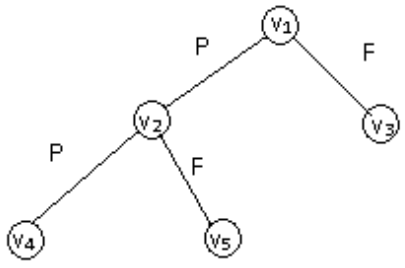
A2.1 One-vertex-extension tree

We shall here use the definition and instruction given in section 7.2 to build a one-vertex-extension tree for each of the graphs in fig 1.4 and fig 1.9 respectively.

Let v_1 be the root, and for $1 \leq j < i \leq n$ follow the one-vertex-extension ordering so that if a vertex v_i is one of iPj, iFj and iTj to vertex v_j then it is a son to v_j. Order the sons of a node as they are ordered in the one-vertex-extension ordering. Let v_j be a parent to v_i in $ET(G)$, j<i. We denote by $v_j v_i$ an edge in $ET(G)$. We call it a P-edge, or a T- respectively an F-edge if v_i is a pendant vertex attached to v_j, or a true respectively a false twin to v_j.
If G is connected, then $v_1 v_2$ is either a P- or T-edge.

From the graphs above we have the respective one-vertex-extension orderings:
$V(G_1) = \{v_1, .., v_5\}$ and $V(G_2) = \{u_1, .., u_6\}$. We begun with K_2 in both graphs and for G_1 we now assume that $v_2(P)v_1$, and in G_2 that $u_2(T)u_1$, giving us the respective lists of "words", or pruning sequences in reverse: $v_2(P)v_1$, $v_3(F)v_1$, $v_4(P)v_2$and $v_5(F)v_2$ respectively $u_2(T)u_1$, $u_3(T)u_1$, $u_4(P)u_1$, $u_5(F)u_1$ and $u_6(T)u_3$. Thus we have the following one-vertex-extension trees, T_1 respectively T_2, associated with the graphs G_1 and G_2 respectively. We mark the edges with P, T or F depending on what relation the son has to it's parent.

T_1 T_2



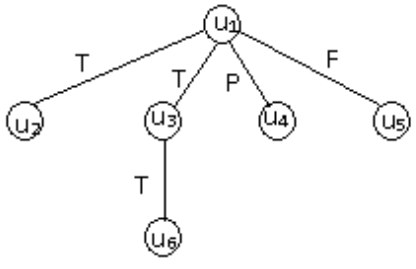


fig 2.1 fig 2.2

A2.2 Twin Set

We see that in T_1 (fig 2.1) only v_3 can be reach from the root v_1 (which is always in the twin set) through a (false) twin edge and thus the twin set of G_1, $TS(G_1) = \{v_1, v_3\}$. In T_2 (fig 2.2) we have

that all vertices but u_4 can be reached from u1 through twin edges and thus $TS(\mathrm{G}_2) = \{u_1, u_2, u_3, u_5, u_6\}$.

Appendix 3

In this appendix we shall use the new recursive definition of distance-hereditary graphs in order to construct one from two other ones – namely those given in fig 1.4 and fig 1.9 of appendix 1.

A3.1 Construction of a distance-hereditary graph from two other ones

Consider the graphs given in appendix 1, G_1 and G_2. Recall the set of operations used to form a distance-hereditary graph from two other ones given in section 7.5.1:

1) A graph consisting of a single vertex is a distance-hereditary graph with the twin set {v}.

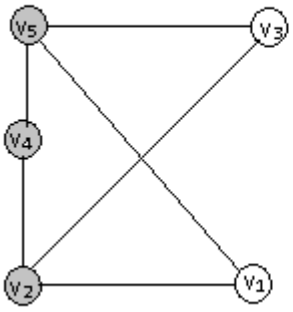
2) If , G_L and G_R are distance-hereditary graphs then the union G of G_L and G_R is a distance-hereditary graph and $TS(G) = TS(G_L) \cup TS(G_R)$. Then G is formed from G_L and G_R by a false-twin operation. This is denoted $G = G_L(F)G_R$.

3) If G_L and G_R are distance-hereditary graphs, then the graph G obtained from G_L and $G_R$by connecting every vertex of $TS(G_L)$ to all vertices of $TS(G_R)$ is a distance-hereditary graph and $TS(G) = TS(G_L) \cup TS(G_R)$. We say that G is formed from G_L and $G_R$by a true-twin operation. This is denoted $G = G_L(T)G_R$.

4) If G_L and $G_R$are distance-hereditary graphs, then the graph G, obtained from G_L and G_R by connecting every vertex of $TS(G_L)$ to all vertices of $TS(G_R)$ is a distance-hereditary graph and $TS(G) = TS(G_R)$. In this case we say that G is formed by a pendant operation. This is denoted $G = G_L(P)G_R$.

Now, let G_1 and G_2 be G_L and G_R. We make the nodes in each graph who are not in their respective twin sets slightly darker.

$G_1 = G_L$ $G_2 = G_R$



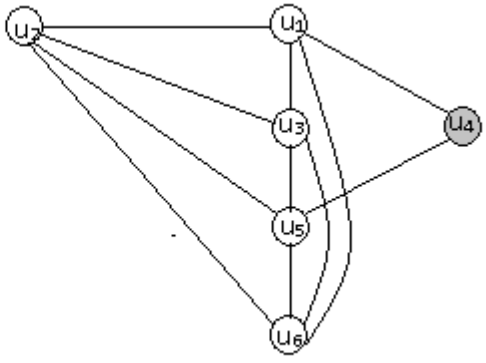
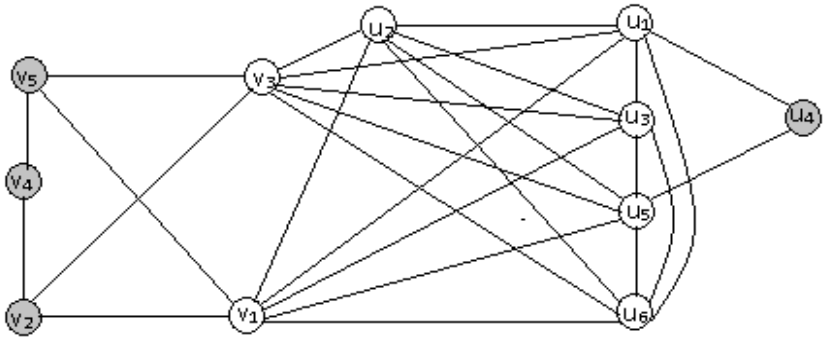


fig 3.1 fig 3.2

We shall employ the twin set operation on G_1 and G_2 to form G, a distance hereditary graph. Since $G = G_L(T)G_R$ we attach every vertex in $TS(G_L)$ to every vertex in $TS(G_R)$ and $TS(G) = TS(G_L) \cup TS(G_R)$, seen in the figures as the vertices not shadowed.



$G = G_L(T)G_R$ fig 3.3

Appendix 4

A4.1 Determining whether G is distance-hereditary or not

Given G above, we "know" that it is a distance hereditary graph since it is constructed from two other ones. Despite our knowledge about the graph, we shall work through all the algorithms from determining wether it is distance-hereditary or not, to the one where we solve the Hamiltonian problem on that graph.

First we employ the algorithm "Prune dhg(G). Our input is the graph $G = (V, E)$ with adjacency matrix M

	v_1	v_2	v_3	v_4	v_5	u_1	u_2	u_3	u_4	u_5	u_6
v_1		1			1	1	1	1		1	1
v_2	1		1	1							
v_3		1			1	1	1	1		1	1
v_4		1			1						
v_5	1		1	1							
u_1	1		1			1	1		1		1
u_2	1		1			1		1		1	1
u_3	1		1			1	1			1	1
u_4						1				1	
u_5	1		1				1	1	1		1
u_6	1		1			1	1	1		1	

fig 4.1

We begin by computing the distance layouts from starting vertex v_3. Thus L_1 is the neighborhood of v_3: $L_1 = \{v_2, v_5, u_1, u_2, u_5, u_6\}$, $L_2 = \{v_1, v_4, u_4\}$.
Set $j \leftarrow 1, i \leftarrow 2$
The connected components of $G(L_2)$ are the three graphs on one vertex, namely $v_1, v_4$and u_4.
There will be no cograph-pruning on one vertex graphs (they are cographs thouhg), contracted they are already so we sort the vertices of $G(L_2)$ by increasing inner degree, $id(v)$: $id(v_1) = 7, id(v_4) = 2, id(u_4) = 2$.
None have inner degree 1 so we go to next if-statement, checking that $i \neq 1$ $(i = 2)$ so that for each x inL_2 taken in increasing inner degree we employ the prune-cograph algorithm

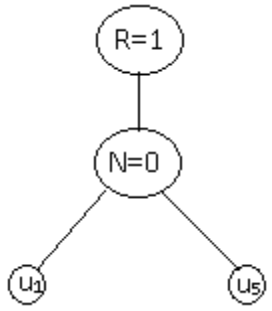
We set $x = u_4$. Checking the graph on u_4's neighbors in L_1: $G[N_1(x)] = G[N_1(u_4)]$ is the disconnected graph on two vertices, namely u_5 and u_1.

$G[N_1(u_4)]$ 

fig 4.2

Prune-cograph algorithm on $G = (\{u_5, u_1\}, \{\emptyset\})$ yields:
Compute cotree T of G. We need algorithm cograph-recognition(G):
First we create a new (1) $node, R$.
Since$[u_5, u_1]$ is not in $E(G)$, create a new (0) $node\ N$, and add N as a child of R and we add u_1 and u_5 as children of N. Then there are no vertices to iteartively incorporate into T.

The cotree T of the cograph $G[N_1(u_4)]$:



cotree T fig 4.3

We now let A be the set of nodes of T having only leaves as descendants, A $= \{N\}$
Now A $\neq \emptyset$ so we pick an arbitrary node N in A, that is the only element N, of A. We also pick an arbitrary son x of N, u_1. For each son $y \neq x$ one wants to find their relation, false or true twins. N is a (0) $node$, so we set $\rho(1) \leftarrow u_5$, $s_1 = u_5 F u_1$.
$j \leftarrow j + 1 = 1 + 1 = 2$.
Replacing N by x and loop once more, we replace R by $x = u_1$ and return u_1 which is the last vertex of the pruning sequence. We contract $N_1(u_4)$ into u_1 and we set $\rho(2) \leftarrow u4,\ s_2 = u_4 P u_1$.
Now the distance layouts are $L_1 = \{v_2, v_5, u_1, u_2, u_3, u_5, u_6\},\ L_2 = \{v_1, v_4, u_4\}$.

Next we look at $x = v_4$. $G[N_1(x)] = G[N_1(v_4)]$ is the disconnected graph on two vertices, v_2 and v_5. In the same way as above we have that $\rho(3) \leftarrow v_5,\ s_3 = v_5(F)v_2,\ \rho(4) \leftarrow v_4,\ s_4 = v_4(P)v_2$.
Now the distance layouts are $L_1 = \{v_2, v_5, u_1, u_2, u_3, u_5, u_6\},\ L_2 = \{v_1, v_4, u_4\}$.

Now the only remaining vertex inL_2 is $x = v_1$. Note that u_5 and v_5 were in $N_1(v_1)$, but they were deleted from the graph when contracting $N_1(u_4)$ and $N_1(v_4)$ above, and hence now $id(v_1) = |N_1(v_1)| = 5$.

Thus $x = v_1$.
The graph $G^* = G[(N_1(v_1)]$ is the disconnected graph on five vertices:

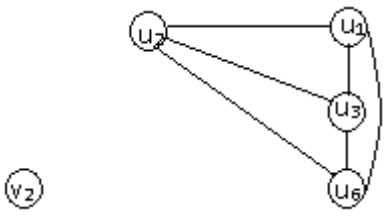
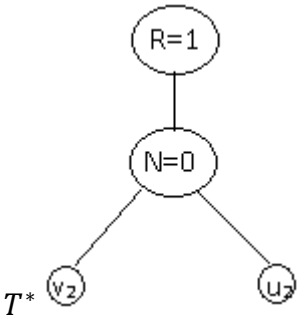


fig 4.4

We employ prune-cograph($G^*, j = 5$) and start with cograph-recognition (G^*) to compute a cotree T of G^*:
Initialization with v_2 and u_2 gives us the cotree T^*



T^* fig 4.5

1) We add u_1 into T^*
   2.1We call procedure $Mark(x)$ where $x = u_1$.

   Mark(u_1)
   Begin by marking all leaves adjacent to $x = u_1$: we mark u_2.
Round 1 Step 1, we unmark u_2 since $d(u_2) = md(u_2) = 0$. $md(u_2)$ is already 0
   Step 2, $u_2 \neq R$ so we mark $w = N$ which is the parent of u_1.
   Step 3, we set $md(N) = md(N) + 1 = 1$
Round 2 step 2 $\nexists v$: $md(v) = d(v)$
   End
   There is a marked vertex and $d(R) = 1$ so we mark R
End Mark.

The following table and tree is obtained, the stars in the tree indicates that the vertex has been marked, and brackets around stars indicates that the veretx has been unmarked. See fig4.6 below.

Table 1, markings

	v_2	u_2	N	R
$d(v)$	0	0	2	1
mark		|	|	
unmark		|		
$md(v)$			|	

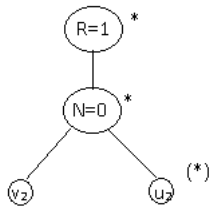


Fig 4.6

Now, we are back in cograph-recognition(G^*):
Neither all nodes, nor no nodes were marked and unmarked so we jump to step 2.4 and call FIND-LOWEST, F-L.
We make a few notes: we have that $md(R) = d(R) - 1$, hence R is properly marked. y is a marked (0)-node N. Also, we set up a table to keep track of identities of vertices u, y, R, w and t. The information is used in step 2 of F-L.

1) Initialization: $y \leftarrow \Lambda$ (nil value). We note that R is marked, thus we might have a cograph. However, $md(R) = d(R) - 1$ so we unmark R, set md(R) to 0 (if needed) and set $u = w = R$. End.

Table 2, identities after initialization of F-L.

	u	y	R	w	t
1		Λ			
1.1	$w = R$	Λ		R	

We choose N since this is the only marked vertex left after initialization of F-L where we unmarked R. We note that $y = \Lambda$ and hence G might be a cograph. Label N is 0 so we do $y \leftarrow N, t \leftarrow$parent$(N) = R$. We unmark N and set md(N)=0. The table of identities now looks as follows:

Table 3, identities at end of each step

	u	y	R	w	t
2	N	Λ			
2.1	$w = R$	Λ		R	
2.3	$w = R$	N		R	R

2.2) is skipped since $t = w = R$.
2.3) set $w \leftarrow u, R \leftarrow N$
End step 2
END FIND-LOWEST.

Again, back in cograph-recognition(G^*) we perform step 2.5. $A = \{u_2\}, B = \{v_1\}$, label N is 0. u_2 in A is a leaf so we add a new 1-node in place of u_2 and make u_2 and u_1 (the veretx we were about to add to T when we begun the cograph-recognition algorithm) children of this node. We thus end up with the following cotree:

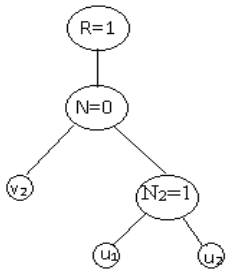


Fig 4.7

Iteratively incorporating u_3 and u_6 as above we will end up with the cotree having v_2, u_1, u_2, u_3 and u_6 as leaves, where v_2 is the child of N, all others are children of N_2 as follows:

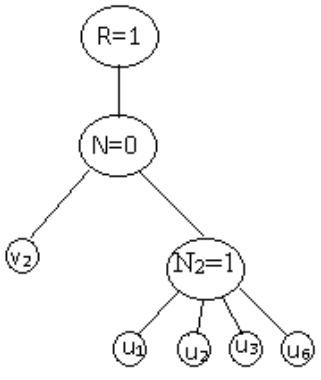


fig 4.8

Now we have found out that the graph $G^* = G[(N_1(x = v_1)]$ is a cograph. We started with prune-dhg(G), who calls prune-cg(G) which in turn calls cograph-recognition(G) with subroutines

$Mark(x)$ and Find-Lowest. Cograph recognition returns a cotree iff G is a cograph. The cograph is also a distance-hereditary graph. Prune-cograph yields a pruning-sequence, which we are about to find now:

$A = \{N_2\}$, we choose $x = u_1$.
Now looping through choosing each son $y \neq u_1$ of N_2 we get the following.
N_2 is a 1-node.
$y = u_6, \rho(5) = u_6,\ s_5 = u_6 T u_1$.
$y = u_3, \rho(6) = u_3,\ s_6 = u_3 T u_1$.
$y = u_2, \rho(7) = u_2,\ s_7 = u_2 T u_1$.

We replace N_2 by $x = u_1$ which gives us the cotree with R as a root, N (a 0-node) as an internal node with v_2 and u1 as its only children. We thus add N to A.

We choose N, and we choose $x = v_2$.
The only son $y \neq v_2$ of N is u_1 and since N is a 0-node we have $\rho(8) = u_1, s_8 = u_1(F)v_2$.
Replacing N by v_2 yields a cotree with R as the root and v_2 as its only child. There is no son $y \neq v_2$ and thus no more words to the pruning sequence is produced in the loop. We replace R by v_2 in T. Now v_2 is the root so we return v_2 as the last vertex of the pruning sequence.

End cograph-recognition.

We are now back in Prune-dhg(G) in step 2.1.1 and we have from above that $y \leftarrow v_4$. We contract G^* into v_4. Now the distance layouts are $L_1 = \{v_2, \sout{v_5, u_1, u_2, u_3, u_5, u_6}\},\ L_2 = \{v_1, \sout{v_4, u_4}\}$. The only connected component of L_2 is the vertex v_1. It has inner degree 1 so we let y$\leftarrow v_2$ the only neigbor of v_1. $\rho(9) = v_1,\ s_9 = v_1(P)v_2$.
Now also v_1 is deleted from the distance layouts, leaving only v_2 in $L_1$and v_3, the starting vertex, as the remaining vertices. v_2 has inner degree 1.
We let $x = v_2$ and $y = v_3$, its only neighbor and $\rho(10) = v_2,\ s_{10} = v_2(P)v_3$
We're done!

The pruning sequence, i.e. the order in which the vertices are added using one of pendant-, true twin- or false twin-operation, is the following (note that the algorithm yields the sequence in backwards so that the word labelled s_{10} is the starting operation, followed by s_9 and so on):
$v_2 P v_3, v_1 P v_2, u_1 F v_2, u_2 T u_1, u_3 T u_1, u_6 T u_1, v_4 P v_2, v_5 F v_2, u_4 P u_1, u_5 F u_1$.

Let us try to recreate the graph we started with:

$v_2 P v_3$ $v_1 P v_2$ $u_1 F v_2$

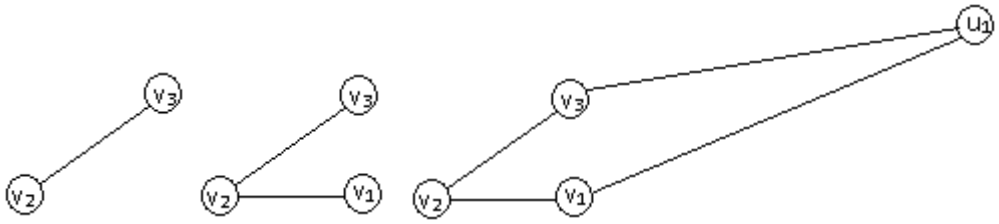


Fig4.9 fig 4.10 fig 4.11

u_2Tu_1

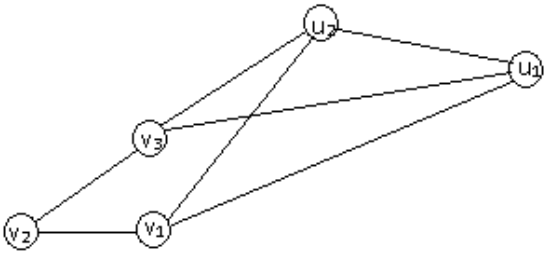


fig 4.12

u_3Tu_1

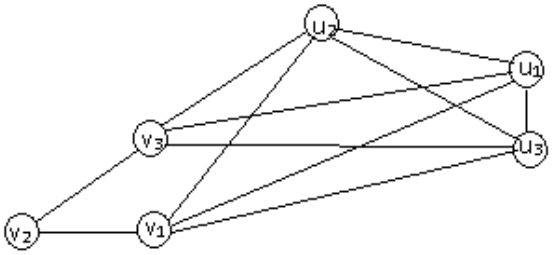


fig 4.13

u_6Tu_1

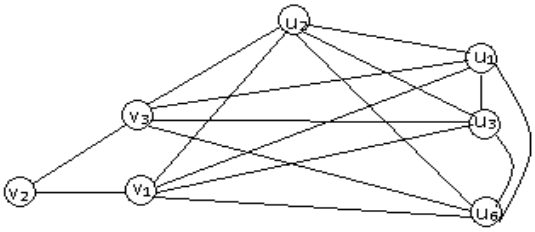


fig 4.14

v_4Pv_2

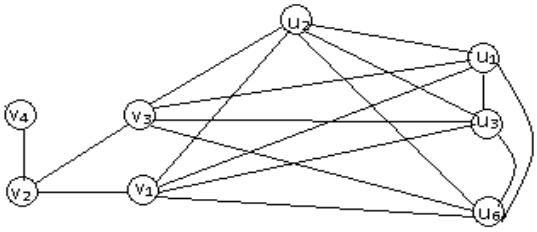
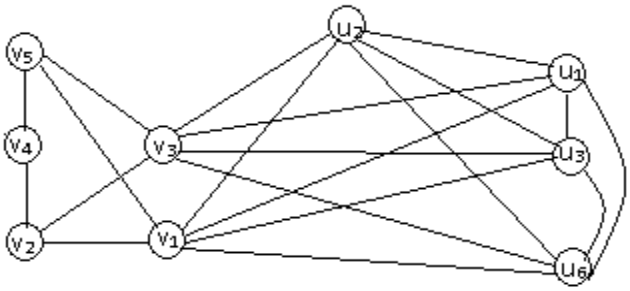
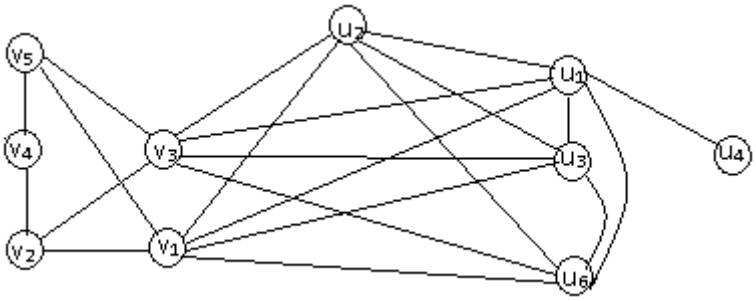


fig 4.15



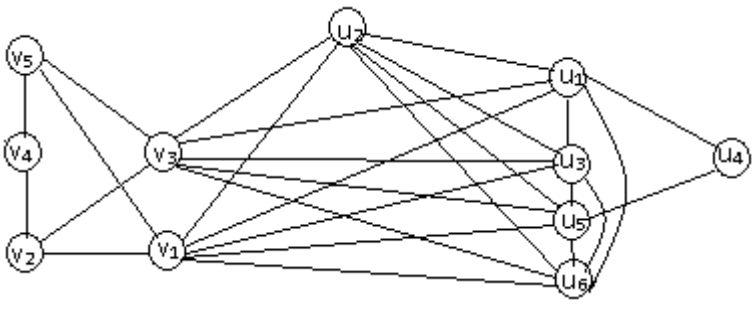
v_5Fv_2

fig 4.16



u_4Pu_1

fig 4.17



u_5Fu_1

fig 4.18

The ”new” graph, G^*, emanating from our pruning-sequence, see fig 4.19 below;

G^*

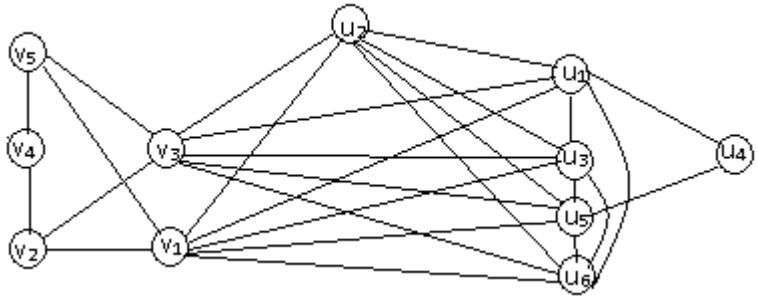


fig 4.19

The original graph G (fig 3.3) on which we employed thealgorithm prune-dhg(G):

G

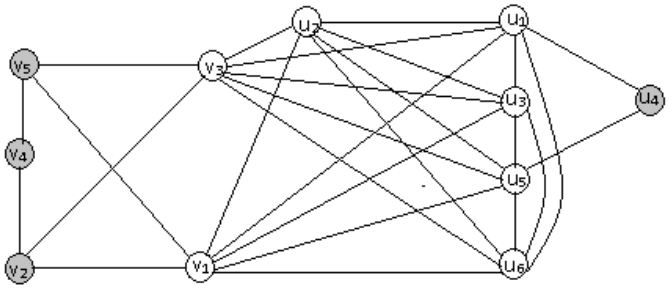


fig 4.20

It is easlisy seen that hey are isomorphic by comparing their respective adjacency matrixes.

The one-vertex-extention tree $ET(G^*)$ of the graph G^* (fig 4.21) is given by the prining sequence. We see that the twin set of G^* consists of the vertex v_3 only.

$ET(G^*)$

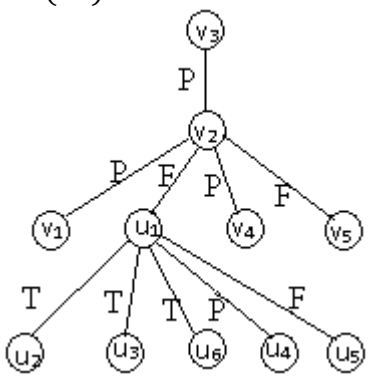


fig 4.21

A4.2 The Decomposition Tree

In the following, we let $G^* = G$. Given the one-vertex-extension tree we construct the decomposition tree, $DT(G)$, of G by letting the root be a vertex labeled by the type of edge from the

root of $ET(G)$ to its first child, here the edge v_2v_3 is a P-edge. We find that $V_R(1,2) = v_3$ and $V(2) = V[v_2]$ i.e. $v_2$and all of its descendants. Thus $G = G_L(P)G_R$=$G_R[1,2](P)G[2] = v_3(P)G[v_2]$. The twin set of G is the twin set of G_L; $TS(G) = TS(G_L) = TS(G_R[1,2]) = v_3$. The Decomposition tree $DT(G)$ of G now consists of a root labeled P and its two children, $DT(G_L)$ and $DT(G_R)$ being v_3 and $G[v_2]$ respectively.

$DT(G) = G_L(P)G_R$

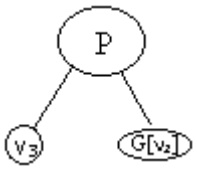


fig 4.22

Next, consider $ET(G[2]) = ET(G^{(1)})$:

$ET(G[2]) = ET(G^{(1)})$

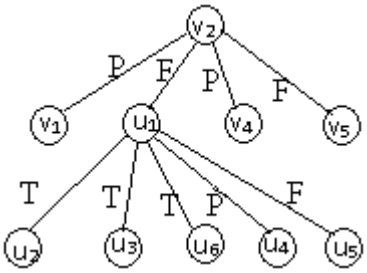
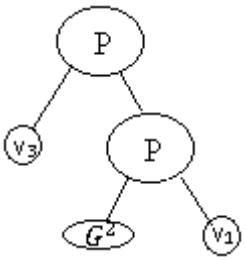


fig 4.23

We have that $V^{(1)}(2) = v_1$, $V^{(1)}{}_R[1,2] = V^{(1)}\backslash\{v_1\}$ and the type of edge from the root to its first child, i.e. the type of edge v_2v_1, is P. We thus conclude that the subgraph $G^{(1)}$of G is formed from $G^{(1)}{}_L$ and $G^{(1)}{}_R$ by a pendant operation, where $G^{(1)}{}_L = G^{(1)}{}_R[1,2] = G^{(1)}\backslash\{v_1\} = G^{(2)}$ and $G^{(1)}{}_R = G^{(1)}[2] = v_1$. The decomposition tree, $DT(G)$, now consists of a root labeled P, with two children, v_3 and a node labeled P. The latter have two children, v_1 and $G^{(1)}[2]$. $TS\left(G^{(1)}\right) = TS\left(G^{(1)}{}_L\right) = TS(G^{(1)}{}_R[1,2]) = TS(G^{(2)})$.



$DT(G)$ fig 4.24

We proceed with $ET(G^2)$:

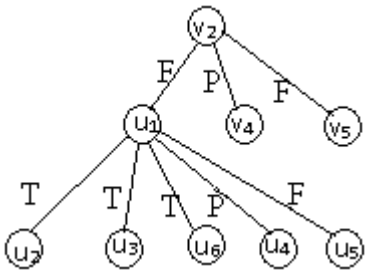


fig 4.25

We have that $V^{(2)}(2) = V[u_1] = \{u_1$ and all of its desecendants$\}$, ${V^{(2)}}_R[1,2] = \{v_2, v_4, v_5\}$ and the type of edge from the root to its first child, i.e. the type of edge v_2u_1, is F. We thus conclude that the subgraph $G^{(2)}$of G is formed from ${G^{(2)}}_L$ and ${G^{(2)}}_R$ by a false twin operation, where ${G^{(2)}}_L = {G^{(2)}}_R[1,2]$ (i.e. the graph with vertex-set $\{v_2, v_4, v_5\}$) and ${G^{(2)}}_R = G^{(2)}[2] = G[u_1]$ (i.e. the graph on the set of vertices consisting of u_1 and all of its descendants). The decomposition tree, $DT(G)$, now consists of a root labeled P, with two children, v_3 and a node labeled P. The latter have two children, a node labeled F and v_1. The node labeled F have two children: ${G^{(2)}}_L = {G^{(2)}}_R[1,2]$ and and ${G^{(2)}}_R = G^{(2)}[2] = G[u_1]$. We have that $TS\left(G^{(2)}\right) = TS\left({G^{(2)}}_L\right) \cup TS({G^{(2)}}_R[1,2])$.

$DT(G)$

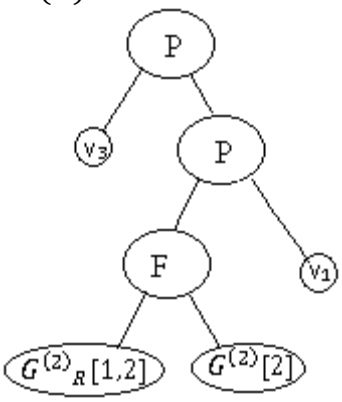


fig 4.26

Working our way through the subgraphs ${G^{(2)}}_R[1,2]$ and $G^{(2)}[2]$ in this manner we eventually end up with the following decomposition tree $DT(G)$ of the distance hereditary graph G, see fig 4.27 below:

$DT(G)$

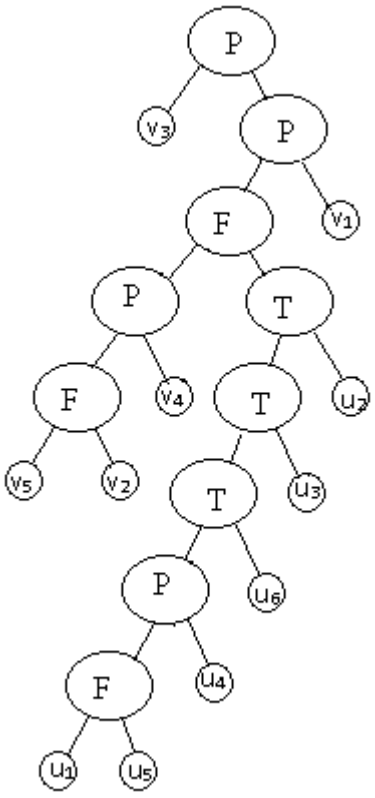


fig 4.27

A4.2.1 The Constants of the Decomposition Tree

Theorem 9 and theorem 10 yields a recursive program for computing the maximum and the minimum cardinalities, $\kappa_1(G)$ and $\kappa_2(G)$ respectively, of the minimum-free-number twin-set path covers (mfn-tspc) of G and the free number $f(G)$ in linear time using the decomposition tree of our distance-hereditary graph, G, if G has a twin-set path cover. These constants are used in theorem 11 to state necessary and sufficient conditions for a distance-hereditary graph G to have a Hamiltonian Path.

We shall consider the label of each internal node of $DT(G)$ as stating the type of operation used to form a distance hereditary graph from two other ones, namely the children of each internal node, where a child being an internal node represents a subgraph on at least two vertices, and a child being a leaf in $DT(G)$ is a subgraph on one node.
Note that in this section, vertices in the twin set of a graph are shadowed (as opposed to appendix 3).

The following table is used to keep track of events: table 4.1

i – the level of internal nodes in DT(G) where level 1 is max distance from the root.	G The distance-hereditary graph at hand (in the rows)	$\exists\ tspc$:"there exists a twin-set path cover"	$\kappa_1(G)$ maximum cardinality of mfn-tspc of G	$\kappa_2(G)$ minimum cardinality of mfn-tspc of G	$f(G)$ Free number of tspc of G	$TS(G)$ The twin set of G
G_L	The left graph	"there exists a twin-set path cover" is True or False for G_L				
G_R	The right graph	"there exists a twin-set path cover" is True or False for G_R				
$G^{(i)} = G_L(\cdot)G_R$	Type of operation, $(\cdot) =$ P, T or F, used to form $G^{(i)}$ from G_L and G_R.	"there exists a twin-set path cover" is True or False for $G^{(i)}$				

We begin at level 1 of $DT(G)$ and consider u_1 and u_5 as being two graphs used to form a distance-hereditary graph $G^{(1)}$ using false-twin operation.

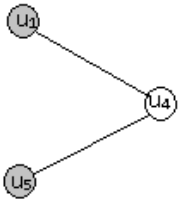
table 4.2

1	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	u_1	True ($u_1$itself)	0 by definition	1 by definition	1 by definition	u_1
G_R	u_5	True (u_5 itself)	0 by definition	1 by definition	1 by definition	u_5
$G^{(1)}$	$G_L(F)G_R$	True by thm 9	0 by thm 9.2	2 by thm 10.1	2 by thm 10.3.a	$u_1 \cup u_5$

$G^{(1)}$ fig 4.28

table 4.3

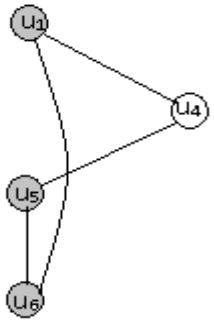
2	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	$G^{(1)}$	True	0	2	2	$TS(G^{(1)})$
G_R	u_4	True	0	1	1	u_4
$G^{(2)}$	$G_L(P)G_R$	True by thm 10.4; a)$\exists\ tspc$ of G_L and G_R b)$f(G_L) + f(G_R) \leq 2$ c)$\kappa_2(G_L) - \kappa_1(G_R) \geq f(G)$	0 by thm 10.5; since $\kappa_2(G_L) - \kappa_1(G_R) > 0$ we have $f(G) = f(G_L) + f(G_R)$	1 by thm 10.3.c	1 by thm 10.3.c	$TS(G^{(1)})$



$G^{(2)}$ fig 4.29

Table 4.4

3	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	$G^{(2)}$	True	0	1	1	$TS(G^{(2)})$
G_R	u_6	True	0	1	1	u_6
$G^{(3)}$	$G_L(T)G_R$	True by thm 9.1	0 by thm 10.2	2 by thm 10.3.b	1 by thm10.3.b	$TS(G^{(2)}) \cup u_6$



$G^{(3)}$ fig 4.30

Table 4.5 a

4a		$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	v_5	True	0	1	1	v_5
G_R	v_2	True	0	1	1	v_2
$G^{(4a)}$	$G_L(F)G_R$	True	0	2	2	$v_2 \cup v_5$

Table 4.5 b

4 b	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	$G^{(3)}$	True	0	2	1	$TS(G^{(3)})$
G_R	u_3	True	0	1	1	u_3
$G^{(4b)}$	$G_L(T)G_R$	True	0	3	1	$TS(G^{(3)}) \cup u_3$

$G^{(4a)}$ $G^{(4b)}$

fig 4.31

Table 4.6 a

5a	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	$G^{(4a)}$	True	0	2	2	$TS(G^{(4a)})$
G_R	v_4	True	0	1	1	v_4
$G^{(5a)}$	$G_L(P)G_R$	True	0	1	1	$TS(G^{(4a)})$

Table 4.6b

5b	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	$G^{(4b)}$	True	0	3	1	$TS(G^{(4b)})$
G_R	u_2	True	0	1	1	u_2
$G^{(5b)}$	$G_L(T)G_R$	True	0	4	1	$TS(G^{(4a)}) \cup u_2$

$G^{(5a)}$ $G^{(5b)}$

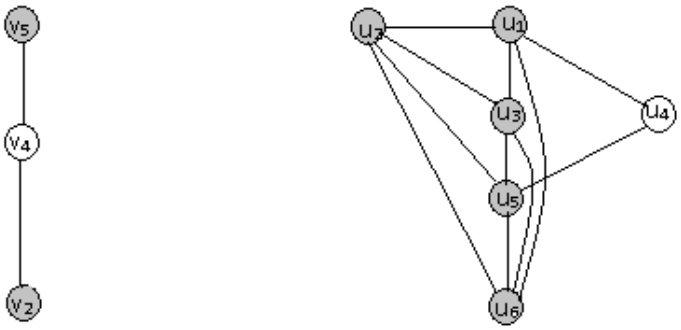


fig 4.32

Table 4.7

6	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	$G^{(5a)}$	True	0	1	1	$TS(G^{(5a)})$
G_R	$G^{(5b)}$	True	0	4	1	$TS(G^{(5b)})$
$G^{(6)}$	$G_L(F)G_R$	True	0	5	2	$TS(G^{(5a)}) \cup TS(G^{(5b)})$

$G^{(6)}$

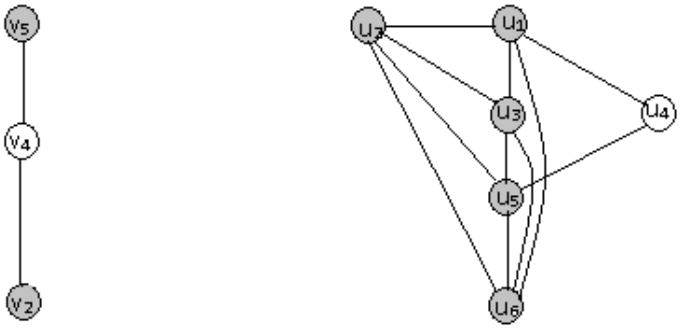


fig 4.33

Table 4.8

7	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	$G^{(6)}$	True	0	5	2	$TS(G^{(6)})$
G_R	v_1	True	0	1	1	v_1
$G^{(7)}$	$G_L(P)G_R$	True	0	4	1	$TS(G^{(6)})$

$G^{(7)}$

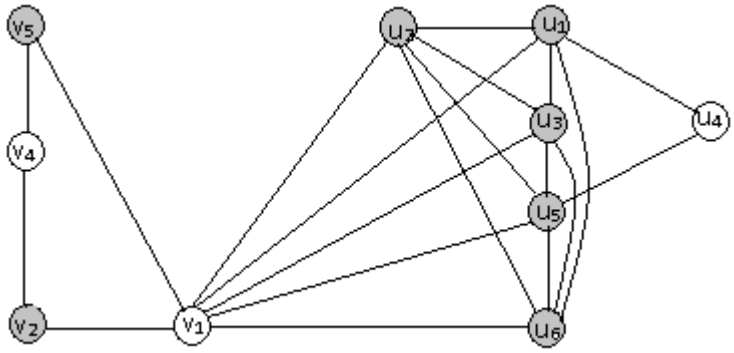


fig 4.34

Table 4.9

8	G	$\exists\ tspc$	$f(G)$	$\kappa_1(G)$	$\kappa_2(G)$	$TS(G)$
G_L	v_3	True	0	1	1	v_3
G_R	$G^{(7)}$	True	0	4	1	$TS(G^{(7)})$
$G^{(8)}$	$G_L(P)G_R$	True	1	1	1	v_3

$G^{(8)}$

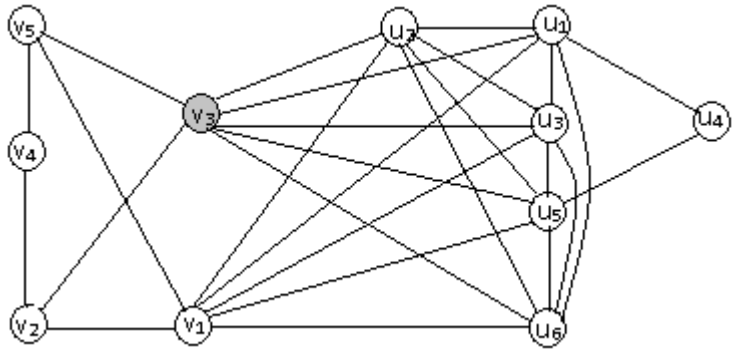


fig 4.35.

Now by theorem 9 and theorem 10 we have recursively computed $\kappa_1(G) = 1, \kappa_2(G) = 1, f(G_L) = 0$ and $f(G_R) = 0$. By theorem 11 we have that $G^{(8)}$ has a Hamiltonian path if and only if:
either
(1) $f(G_L) + f(G_R) \leq 1$ and $\kappa_2(G) = 1$, or

(2) $f(G_L) + f(G_R) = 2$ and $\max\{1, \kappa_2(G_L) - \kappa_1(G_R) + f(G_R), \kappa_2(G_R) - \kappa_1(G_L) + f(G_L)\} = 1$.

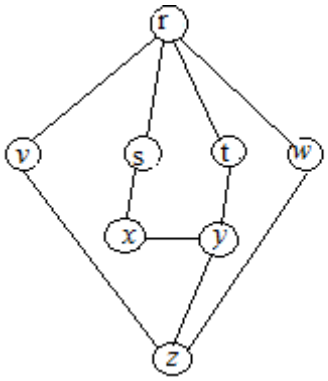
We have that (1) is true, and hence the distance-hereditary graph $G^{(8)}$ has a Hamiltonian path (which is easily confirmed by visual inspection!).

Appendix 5

In this section we shall give an example of an instance of the cycle-k-partition recognition. We shall also give examples on the outcome from cycle-k-partition recognition on other instances.

In this first example, we shall work with the graph G of figure 5.1. We keep track of events in table 5.1 and in table 5.2. In table 5.1 columns give the status of each parameter after each time we process algorithms step 11. We get there either after a new i-loop starts or when we are instructed to go there. Lined throuhg signs, like ~~y,~~ in the table is supposed to describe the dynamics, a vertex is lined through when moved from a set.

We will also grow trees accordingly to the proceedings and the paths found in the algorithm SSCS(v_j) as we also keep track of the paths we discover in sets B_α^β. This is for visualization purpose.



G fig. 5.1

We initialize and in step 1 we set a value to parameters, and sets are assigned the empty set as its only element. In initialization step 2 we choose the vertex r to be our v_0 and $N(v_0) = \{v, s, t, w\}$ is set to $\{u_1, u_2, u_3, u_4\}$ and we set $A_0 = \{u_1, u_2, u_3, u_4\}$. Also, in step 4, we root the first tree T_1 at v_0, and add the vertex u_1 as a child of v_0. We have thus a new labeling of the graph G, and a tree rooted at v_0 (see figure 5.2).

G: T_1:

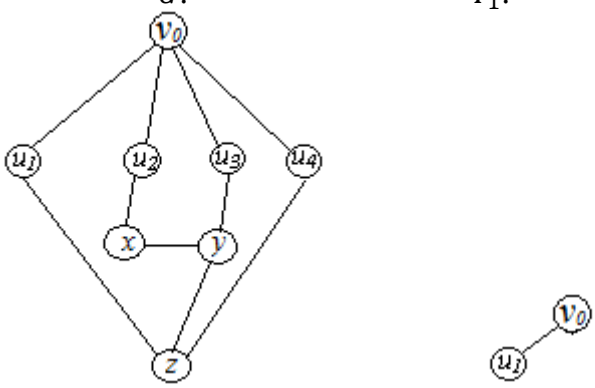


fig 5.2.

Table 5.1

Loop i	initialization	$i=1$	$i=1$	$i=1$	$i=1$	$i=1$	$i=1$	$i=$	$i=1$
Number of re-startsof (step 11)		$rs=1$	$rs=2$	$rs=3$	$rs=4$	$rs=5$	$rs=6$	$rs=$	7 backtrack to v_0
α	1	1	2	2	~~2,~~ 3	3	4		
r	1	1	2	2	3	3	4		
d_T	1	2	~~3,~~ 2,	3	~~4,~~3	4	~~5~~,4		~~4,3,2,~~1
β	1	2	2	3	~~4,~~3	4	5		
F_1	v_0, u_1	v_0, u_1	v_0, u_1	v_0, u_1	v_0, u_1	v_0, u_1	v_0, u_1	v_0 , u_1	v_0, u_1
F_2			z, u_4	z, ~~u_4~~	z, y, u_3	z, y, ~~u_3~~	x, y, z, u		~~$x, y, z, u_2,$~~ ~~u_2, u_3, u_4~~
A_0	~~u_1~~, u_2, u_3, u_4	u_2, u_3, u_4	u_2, u_3, ~~u_4~~	u_2, u_3, u_4	u_2, ~~u_3~~, u_4	u_2, u_3, u_4	~~u_2~~, u_3, u_4		~~u_2, u_3, u_4~~, u_2, u_3, u_4
A_1	x, y, z	x, y, z	$x, y,$ ~~z~~	$x, y,$ ~~z~~	$x,$ ~~y, z~~	$x,$ ~~y, z~~	~~x, y, z~~		~~$x, y, z,$~~ x, y
B_α^β	$B_1^1 = \{v_0, u_1\}$	$B_1^2 = \{v_0, u_1, z\}$	$B_1^3 = \{v_0, u_1, z, u_4\}$ $B_2^2 = \{v_0, u_1, z\}$	$B_2^3 = \{v_0, u_1, z, y\}$	$B_2^4 = \{v_0, u_1, z, y, u_3\}$ $B_3^3 = \{v_0, u_1, z, y\}$	$B_3^4 = \{v_0, u_1, z, y, x\}$	$B_3^5 = \{v_0, u_1, z, y, x, u_2\}$ $B_4^4 = \{v_0, u_1, z, y, x\}$		
$\mathcal{C}$			$C_1 = B_1^3$		$C_2 = B_2^4$		$C_3 = B_3^5$		
v_{j-1} in/out									
v_j in/out	u_1	u_1 / z	z/z	z / y	y / y	y / x	x/x		
v_{j+1}		z	u_4	y	u_3	x	u_2		

Table 5.2 Vertices at distance d_T from v_0 in T_1

	$d_T=1$	$d_T=2$	$d_T=3$	$d_T=4$	$d_T=5$				
L_{d_T}	u_1	z	u_4,y	u_3 ,x	u_2				

Note that when we say "move x to S" we actually mean "create a copy of x and insert in S" When all values in the initialization steps 1-10 is done, we begin with the subroutine first iteration at step 11. We have $v_j = u_1$ and $N(v_j) \cap A = z \neq \emptyset$, so we have but one choice of w. Step 12 $v_{j+1} \leftarrow w = z$. We set $d_T \leftarrow d_T + 1 = 2$ in step 13. A check in step 14: if we have come as far away from v_0 as the length of a longest cycle in a k-partition we start backtracking since we need not cycles of greater length, this is not the case now since $n - d + 2 = 8 - 4 + 2 = 6$. Step 15-17, we move u_1 to F_2 Set "flexible" set of forbidden vertices, expand the tree by adding the vertex z as a child of u_1 in T_1 and at the end of B_1^1. We are still in the same branch of T_1 so we change only the superscript of B_α^β; $B_{\alpha=1}^{\beta=2} \leftarrow B_{\alpha=1}^{\beta=1}$. We have gone far enough by the if-statement in step19 and z does not belong to A_0, but $L_{d_T} = L_2$ is empty so there is nothing to remove in step 22 or 24. Thus we insert z into L_2 at step 25. Again, z does not belong to A_0 so we skip the steps 27-35. We note that $z \in A_1$ and switch index from $j+1$ to j, thus having $v_j = z$ instead of $v_{j+1} = z$ and $v_j = u_1$ after

step 37 (hence "outgoing" $v_j = z$). Next, step 38 throws us back to step 11 with ingoing $v_j = z$. The tree T_1 now looks as follows in fig 5.3.

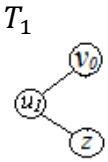


fig 5.3

This is the second time we are in step 11. We move on from step 11 with $v_j = z$. As it happens we choose $w = u_4$ between u_4 and y, $v_{j+1} \leftarrow w = u_4$; $d_T \leftarrow 3$, (skip step 14), move z to F_2, add u_4 to T_1 as a child of z and to B_1^2 which we index to B_1^2 and we're done to step 18. Now, in step 20-22, $u_4 \in A_0$so we remove from $L_{d_T} = L_3$ all vertices that are in both L_3 and F_2 but are not in A_0 (This is of importance when backtracking). But L_3 is empty so nothing happens. We insert u_4 in L_3 at step 25.

Now in step 26-29, again since $u_4 \in A_0$, we set $C_r \leftarrow B_\alpha^\beta$ i.e. $C_1 = \{v_0, u_1, z, u_4\}$ and store C_1 in $\mathcal{C}$. We increase cycle index $r \leftarrow r + 1 = 2$ and move u_4 to F_2. Step 30-31 says we backtrack to $v_j = z$ and decrease $d_T \leftarrow d_T - 1$=3-1=2. Now in step 32, we check if there are any available vertices to explore in the neighborhood of z: there is the vertex y so we do create a new branch in T_1 by increasing the branch index $\alpha = \alpha + 1 = 2$ in step 33 and in step 34 we remove all vertices on further distance from v_0 in the tree from the current branch. This is preparations for extending the branch. Now, we go back to step 11 again. See column $sr = 2$ for the current sate of the algorithm before we start at step 11 a third time. Note that $v_j = z$ and the vertex y has not been explored. The tree T_1 now looks as follows in fig 5.4.

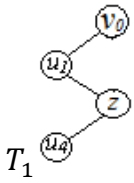


fig 5.4

Third time at SSCS(z) step 11. Step 11-13, y is the only available vertex, thus $v_{j+1} \leftarrow y$ and $d_T \leftarrow d_T + 1$=3. Skip 14. Step 15-18:We move z to F_2 (is already there) and add $v_{j+1} = y$ as a child of z in T_1 and insert y in B_α^β whose superindex increases. Step 19,23 and 24 yields the removal of u_4 from F_2 (in case there is a path thereto from y). We insert y into L_3 by step 25. The steps 26-35 are skipped, instead we set $v_{j+1} = y$ to $v_j = y$ and again, we go back to step 11. The column $rs = 3$ gives the state of the algorithm before processing from step 11 a fourth time. T_1 now looks as in figure 5.5.

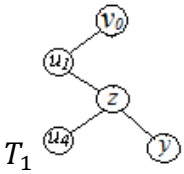
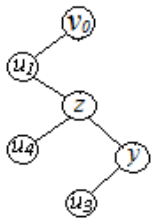


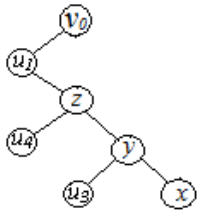
fig 5.5

Fourth time att sscs(v_j = y) step 11-13; $N(y) = \{x, u_3\}$. We choose u_3 and set $v_{j+1} = u_3$, $d_T = 4$. Skip step 14. y is inserted in F_2 at step 15, u_3 is added as a child of y in T_1 and inserted in B_α^β for which we change indices. Hence we have come to step 19: $d_T = 4 \geq 2$, perform step 20 since $u_3 \in A_0$. L_4 is empty so there will be no removal from F_2. Next step 25: u_3 is inserted into L_4. Steps 26-34: store the cycle just found and increase cycle index, insert u_3 into F_2 backtrack to y and decrease d_T by 1 to $d_T = 3$ and remove vertices of greater distance then 3 from the root in T_1 from B_α^β, and index B_α^β as a new branch. Step 35: go to step 11.



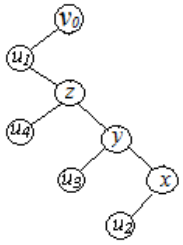
T_1 fig 5.6

Step 11 a fifth time! $v_j = y$, $v_{j+1} = x$, $d_T = 4$, we add y to F_2, we add x as a child of y in T_1 and insert x into B_3^4. Step 19, 23 and 24; we remove u_3 from F_2 . insert x into L_4. Skip 26-35.Set new index for $v_{j+1} = x$ to $v_j = x$, go to step 11.



T_1 fig 5.7

Step 11 once again: $v_j = x$, $v_{j+1} = u_2$, $d_T = 5$, we add x to F_2, we add u_2 as a child of x in T_1 and insert u_2 into B_3^4 which is set to B_3^5. Step 19-22: nothing since L_5 is empty. Insert u_2 to L_5. Steps 26-31: $u_2 \in A_0$, we store B_α^β as a cycle C_3, increase r to $r = 4$, insert : u_2 into F_2. Backtrack to $v_j = x$, decrease d_T to $d_T = 4$. Step 32: $N(x) \cap A \neq \emptyset$ is false so we go to step 11. first we have a glance at T_1 in fig 5.8.



T_1 fig 5.8

Step 11, seventh time, $v_j = x$. $N(x) \cap A = \emptyset$. Therefore we do not enter the if-statement at step 12, but instead go to step 39. Now, this time, $v_j \neq u_j$ so we go straight to step 47 wher we are directed to Backtrack(T, v_j).

Backtrack (T_1, x).
Step 1, the parent of x in T_1 is set to v_{j-1}, thus $v_{j-1} = y$. We jump to step 9 where $d_T \leftarrow d_T - 1 = 3$. Now, step 10 -12, check if there are vertices in F_2 that also are on the same level in the tree as the children and grandchildren of y but not themselves being neither children nor parents of y. If there are such vertices, remove them from F_2: we are backtracking through the tree now, and there may be vertices left in the forbidden set from the process of growing the tree. Thus, we remove u_2 from F_2.
Next move, step 13-14: all children of y are made unavailable so that we do not branch off to them again. We thus insert u_3 into F_2 again. We check if there are any possibilities of growing a new branch from this state in step 15: there is not, so we jump to step 21 and set $v_j = y$ and go to step 1 of Backtrack(T_i, v_j) with parameters $T_i = T_1$ and $v_j = y$.

Through the backtracking procedure we want to grow a new branch if and only if there is a uniquely new branch to grow – therefore, at each state where we can branch off, we make sure that all vertices in the neighborhood of v_{j-1} in T_i are in F_2. Otherwise we would get stuck in an eternal loop. We also make sure that vertices that are not explored in the current branch are made available – step 11 of Backtrack(T_i, v_j).

We continue the procedure: Backtrack(T_1, y) step 1: the parent of y in T_1 is z, so we set $v_{j-1} = z$. We jump to step 9 and set $d_T = 2$. Step 10-12: All vertices in L_3 and L_4 that are also in F_2 but are not neighbors of z in T_1, that is u_3 and x, are removed from F_2. Step 13-14: make sure that both y and u_4 is in F_2 – we thus insert u_4 in F_2. There are no available neighbors of z in G so we set $v_j = z$, and go to

Backtrack(T_1, z) step 1: $v_{j-1} = u_1$, step 9 $d_T = 1$. Step 10-12: we remove u_4 and y from F_2. All children of u_1 in T_1 are inserted into F_2 in step 13-14. Step 15-20 are skipped. We set $v_j = u_1$ and go to step 1 of Backtrack(T_i, v_j).

Now, step 1 of Backtrack(T_1, u_1): $v_{j-1} = v_0$. Step 2-8: we prepare for next i-loop by increasing α since next branch will necessarily be unique starting with v_0u_2 in a whole new tree. We start building branches from the beginning $B_\alpha^\beta \leftarrow v_0$, we empty all L_{d_T}'s since they will have different elements when building the next tree. We empty F_2 because we want to have all vertices available from the start (except v_0), and finally $i \leftarrow i + 1 = 2$. Then we go to step 4 of SSCS(v_j). In table 5.1 is now the state of the algorithm just before preforming step 2-8.

Now that we are in step 4 of SSCS(v_j) the following happens: We root a new tree $T_i = T_2$ since $i = 2$. Step 5: we remove all previously used neighbors of v_0 which in this instance and state of the algorithm is equivalent to u_1. Also, u_1 is removed from A_0. As it may happen, when we delete a neighbor u_i of v_0, a vertex $x \in N_G(u_i)$ can end up with $d_{G\setminus\{u_i\}}(x) = 1$. The point is that we do not want unnecessary information as input into the algorithm. Now, for the second loop, we thus have the following input graph $G^{(2)} = G\setminus\{u_i\}$

$G^{(2)}$

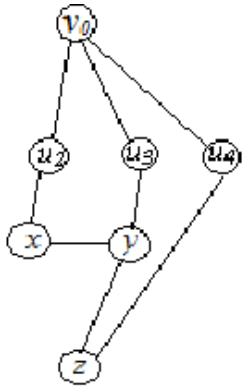


fig 5.9

Moreover, T_2 will grow as follows in fig 5.10 below.

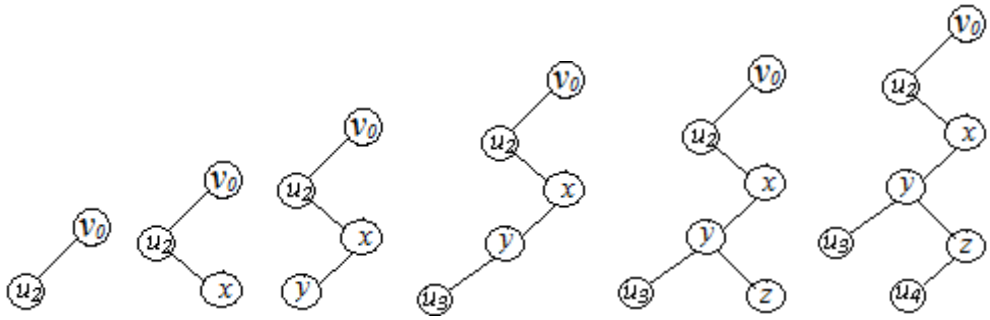
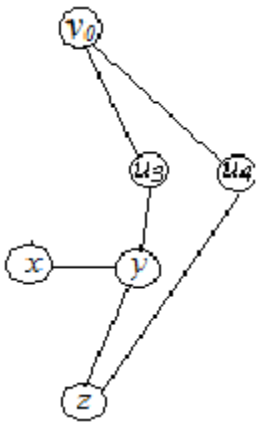
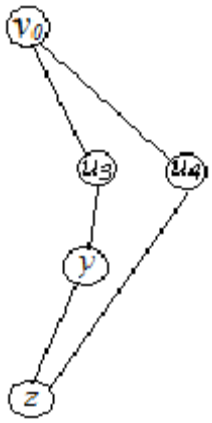


fig 5.10

In the third, and final loop, we will have the following input graph $G^{(3)} = G\setminus\{u_1, u_2\}$ in which x will have degree one and is thus removed. We thus end up with $G^{(3')} = G\setminus\{u_1, u_2, x\}$ which will be the actual input graph in the third loop,see figure 5.11.



$G^{(3)} = G\backslash\{u_1,\ u_2\}$



$G^{(3')} = G\backslash\{u_1, u_2, x\}$ fig 5.11

The tree T_3 will grow as follows:

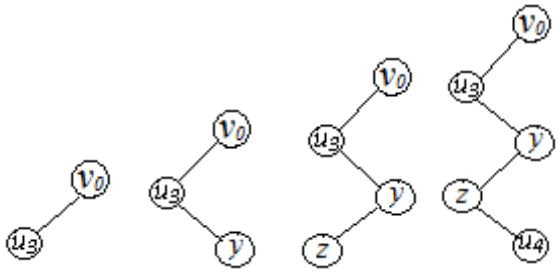


fig 5.12

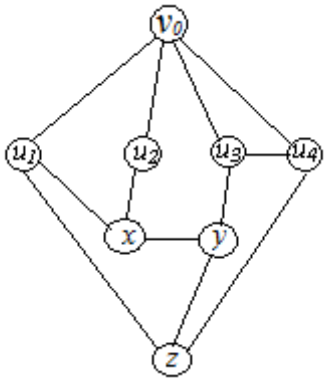
The set of cycles found in this particular instance of the Hamiltonian cycle problem is the following: $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5, C_6\}$ where $C_1 = v_0u_1zu_4$, $C_2 = v_0u_1zyu_3$, and $C_3 = v_0u_1zyxu_2$ were found in the first loop, $C_4 = v_0u_2xyu_3$ and $C_5 = v_0u_2xyzu_4$ is to be found in loop two and $C_6 = v_0u_3yzu_4$ will be found in the third and last loop. The absolute values of each cycle is: $|C_1| = 4$, $|C_2| = 5$, $|C_3| = 6$, $|C_4| = 5$, $|C_5| = 6$, $|C_6| = 4$

We want to match the cycles in the proper combination to find out whether G is Hamiltonian or not, i.e. which, if there are any, combinations of cycles in $\mathcal{C}$ satisfies the conditions of being a k-partition.

Note here that the enumeration of cycles in $\mathcal{C}$ is not the enumeration of the cycles in the k-partition. We have by theorem 16 that $\sum_{i=1}^{k}|C_i| = n + 2(k-1)$. Here we want $|C_{i_1}| + |C_{i_2}| + |C_{i_3}| = 8 + 2(3-1) = 12$. Partitioning of 12 into 3 parts where the smallest is of size 3 yields the following triples: (3,3,6), (3,4,5), (4,4,6). With our set of cycles only the last partitioning is possible to match. This is done with the combinations C_1, C_3, C_6 or C_1, C_5, C_6. The vertex set of the intersection $V(C_1 \cap C_3) = \{v_0, u_1, z\}$ violates both condition 2 and 3 of the definition since it should be either only $\{v_0\}$ or $\{v_0, u_i\}$ for some $u_i \in N(v_0)$. Likewise does $V(C_1 \cap C_5) = \{v_0, u_4, z\}$. Hence our graph is not Hamiltonian.

Consider the sum $\sum_{i=1}^{k}\sum_{j\neq i, j>i}^{k}|C_i \cap C_j|$. We have $|C_1 \cap C_3|+|C_1 \cap C_6| + |C_3 \cap C_6| = 3 + 3 + 3 = 9$. We also have $|C_1 \cap C_5|+|C_1 \cap C_6| + |C_5 \cap C_6| = 3 + 3 + 4 = 10$. We have $k = 3$ and thus $\frac{(k-1)(k+1)}{2} = \frac{2*5}{2} = 5$. None of the combinations of cycles satisfy the summation condition of conjecture 1, and hence none is a k-partition and therefore G is not Hamiltonian.

Now, consider the graph G', which is the graph $G$with two new edges: u_1x and u_3u_4 (the number of vertices is the same as in G, as well as the neighborhood of v_0).

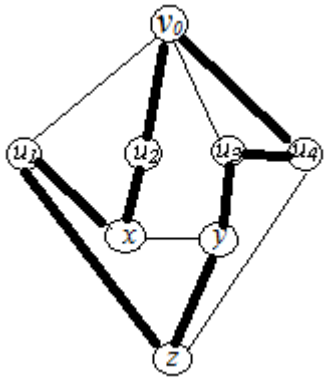


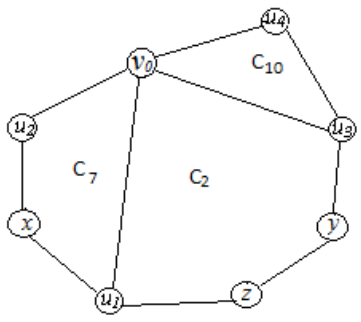
G' fig 5.13

We claim without proof, that the algorithm yields three additional (to the ones found in G) cycles in the first loop, and one additional in the third loop. In the second loop there are no additional cycles since there are no new neighbors of u_2 or any of its neighbors in $G - u_1$, also no cycle starting with v_0u_2 will contain the edge between u_3 and u_4 since the algorithm backtracks when hitting any u_i.

The additional cycles from loop 1 are enumerated starting from 7 here in the example, which is incorrect in the sense that the algorithm enumerates the cycles in the order it finds them. Those found in loop 1 would have indices from 1 to 6. The new cycles coming from loop $i = 1$ are the following: $C_7 = v_0u_1xu_2$, $C_8 = v_0u_1xyu_3$, $C_9 = v_0u_1xyzu_4$ and the additional one from the third loop is $C_{10} = v_0u_3u_4$.

Now we have $|C_1| = 4$, $|C_2| = 5$, $|C_3| = 6$, $|C_4| = 5$, $|C_5| = 6$, $|C_6| = 4$, $|C_7| = 4$, $|C_8| = 5$, $|C_9| = 6$ and , $|C_{10}| = 3$. There is a possibility to match (3,4,5) and (4,4,6). We check the combination $C_2 = v_0u_1zyu_3$, $C_7 = v_0u_1xu_2$ and $C_{10} = v_0u_3u_4$ which in terms of absolute values matches the partition (3,4,5) of the number 12. $C_2 \cap C_7 = (\{v_0, u_1\}, \{v_0u_1\})$, $C_2 \cap C_{10} = (\{v_0, u_3\}, \{v_0u_3\})$ which is fine so far. This requires that C_7 and C_{10} has only the vertex v_0 in common – which is easily verified. Thus conditions 2 and 4 are satisfied. Next we check $V(C_2) \cap N[v_0] = \{v_0, u_1, u_3\}$, $V(C_7) \cap N[v_0] = \{v_0, u_1, u_2\}$, $V(C_{10}) \cap N[v_0] = \{v_0, u_3, u_4\}$. Condition 3 is satisfied. Now, finally we look at the union of the vertex sets (condition 1): $\{v_0, u_1, z, y, u_3\} \cup \{v_0, u_1, x, u_2\} \cup \{v_0, u_3, u_4\} = \{v_0, u_1, u_2, u_3, u_4, x, y, z\} = V(G)$. Thus all conditions of the definition of k-partition are satisfied, and hence G' is Hamiltonian. We can also determine a Hamiltonian cycle that is constituted by this partition. Since C_2 has an edge in common with both C_7 and C_{10}, we cannot start our walk along C_2. We start at v_0 and walk along C_7 until we meet the common vertex of C_7 and C_2 which is u_1. Next we continue our walk along C_2 until we meet the common vertex of C_2 and C_{10} which is u_3. We complete our walk along C_{10}, first to u_4 and we close the walk into a cycle by a final step from u_4 to v_0. Thus, a Hamiltonian path C in G has the following sequence of vertices: $C = v_0u_2xu_1zyu_3u_4v_0$, see fig 5.14 where we also give the 3-partition around v_0 in G', $C' = (C'_1, C'_2, C'_3)$ where $C'_1 = C_7$, $'C_2 = C_2$ and $C'_3 = C_{10}$





G $C' = (C'_1, C'_2, C'_3)$ fig 5.14

Also we have $|C_7 \cap C_2|+|C_7 \cap C_{10}| + |C_2 \cap C_{10}| = 2 + 1 + 2 = 5$ and $\frac{(k-1)(k+1)}{2} = \frac{2*5}{2} = 5$. By conjecture 1 this implies that C_2, C_7 and C_{10} constitutes a k-partition and hence G' is Hamiltonian.