GENERALIZED MAXIMAL INDEPENDENCE PARAMETERS FOR TREES

JOHN MERCER

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Generalized Maximal Independence Parameters for Trees

by

© John Mercer

B.Sc. (Hons.) (Memorial University, St. John's, Canada) 1999

A thesis submitted to the
School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Science

Department of Computer Science
Memorial University of Newfoundland

(June, 2001)

Abstract

An independent set $I \subseteq V$ of a graph $G = (V, E)$ is said to be k-maximal if there does not exist two sets X and Y such that $X \subseteq I$, $Y \subseteq (V - I)$, $|X| < k$ and $(I - X) \cup Y$ is an independent set of cardinality larger than $|I|$. This definition generalizes the traditional concept of maximality of independent sets.

The problem of finding the smallest k-maximal set for a given graph is NP-hard for many simple classes of graphs, such as the class of bipartite graphs. Here we investigate the MINIMUM k-MAXIMAL INDEPENDENT SET (MIN k-MIS) problem for trees. We characterize the k-maximal independent sets for trees and we present an $O(n^3)$ time algorithm for the MIN k-MIS problem under this restriction. We will also show that we can test an independent set for k-maximality in $O(n)$ time for trees.

Contents

List of Figures

Table of symbols

Important classes		
Symbol	Meaning	Page
D^P	Complexity class	4
$\mathcal{C}_k$	Set of configurations of size k	26
$\mathcal{P}_k$	Set of partial configurations of size k	38
$\mathcal{F}_k$	Set of free trees of size k	26
$C((T, I))$	Set of configurations in (T, I)	25
$\mathcal{I}_P(T, r, i, k)$	Set of suboptimal independent sets	52
$\mathcal{I}_C(T, r, i, k)$	Set of suboptimal independent sets	53

Important symbols and conventions		
Symbol	Meaning	page
$G = (V, E)$	An undirected, finite graph	3
$T = (V, E)$	A tree graph	11
$T_1, \ldots, T_m$	Subtrees of a rooted tree T	33
Γ_I	Tree labeling function	13
r	The root vertex of a tree T	34
v	A vertex of a graph or tree	1
$V(G)$	The vertex set of a graph	3
I	A set of independent vertices in a graph	1
X, Y	Two vertex sets of a graph	3
(T, I)	Tree T and independent set $I \subseteq V(T)$	25
(T', X, Y)	Configuration of X and Y in T	25
(T', X', Y')	Partial configuration of X and Y in T	38

1 Introduction

A set I of independent vertices of a graph G is said to be *maximal* if there is no vertex $v \notin I$ such that $I \cup \{v\}$ is also an independent set of G. The MINIMUM MAXIMAL INDEPENDENT SET problem - the problem of of finding the smallest possible maximal independent set for a graph G - is a classic problem in complexity theory. The decision problem corresponding to MINIMUM MAXIMAL INDEPENDENT SET is an example of a NP-complete problem [7].

The theory of maximal independent sets has received some attention. Cook [6] showed that the LEXICOGRAPHICALLY FIRST MAXIMAL INDEPENDENT SET problem is P-complete. Karp and Widgerson [9] proved that the MAXIMAL INDEPENDENT SET problem is complete for NC. Berron and Mata - Montero [1] showed that the MINIMUM MAXIMAL INDEPENDENT SET problem is linear time solvable when the input is restricted to graphs of fixed treewidth.

In this paper, we present an investigation into the complexity of the MINIMUM k-MAXIMAL INDEPENDENT SET (MIN k-MIS) problem when the input is restricted to tree graphs. This problem, first suggested by Cockayne et al. [5], is a generalization of the NP-complete problem described above. The generalized problem has been found to be NP-hard even for some very simple classes of input graphs, such as bipartite graphs [11]. This problem has not been solved for any graph classes other than path or cycle graphs. In this paper we will show that the problem has time complexity $O(n^3)$ when the input is restricted to tree graphs.

The type of generalization that is used in the MIN k-MIS problem was first suggested by Bollobás et al. [2] in reference to the MAXIMUM MINIMAL DOMINATING SET PROBLEM. A subset $S \subseteq V$ of vertices in a graph $G =$

(V, E) is said to be *dominating* if each vertex $v \in (V - S)$ is adjacent to at least one vertex in S. The paper by Bollobás was followed up by Cockayne's paper on the generalized independent set problem. Both papers give upper and lower bound results for simple graph types, such as the classes of path and cycle graphs.

There has been some interest in k-maximality from the algorithmic point of view. McRae [11] proved that the MIN k-MIS problem is NP-complete with fixed $k = 2$ for bipartite graphs and line graphs of bipartite graphs. Manlove [10] proved that the MIN k-MIS problem is NP-hard for the planar graphs with maximum degree 3. Manlove also gave linear time algorithm for the MIN k-MIS problem for the case where $k = 2$ and the input is restricted to trees.

In Section 2 of this paper, we will give a thorough description of the MIN k-MIS problem. We will give a formal description of the problem and demonstrate some basic properties of k-maximal independent sets. We will show that the k-MAXIMAL INDEPENDENT SET problem of testing an vertex set for k-maximal independence is $coNP$-complete, and we will show that the MIN k-MIS problem is D^P-hard.

In Section 3, we will give a useful characterization of k-maximality for those independent sets that correspond to tree graphs. In particular we will show that, for any fixed k, we can test a set I for the property of k-maximality by searching for a fixed number of discrete patterns in the set I and its corresponding tree T. We will show that there is a one-to-one relationship between the set of patterns and the set of unrooted trees.

In Sections 4 and 5, we will use the characterization from Section 3 to prove some algorithmic results. In Section 4, we will present a linear

time algorithm for testing an independent set I that corresponds to a tree T for the property of k-maximality. In Section 5 we will show a $O(n^3)$ dynamic programming algorithm for the MIN k-MIS problem when the input is restricted to tree graphs. We will conclude in Section 6 with a summary and a discussion of open problems.

2 Preliminaries

We define an k-maximal independent set as follows:

Definition 2.1 *Let G be an undirected, finite graph. For any integer k, the set $I \subseteq V(G)$ is k-maximal if:*

- *I is an independent set, and*
- *there does not exist X and Y, where $X \subseteq I$ and $Y \subseteq (V(G) - I)$ such that:*
  - *$|X| = \ell$, $\ell < k$,*
  - *$|Y| = |X| + 1$, and*
  - *$(I - X) \cup Y$ is an independent set.*

In other words, an independent set I is k-maximal if a larger independent set cannot be constructed by first *removing* $\ell < k$ vertices from I and then *adding* $\ell + 1$ vertices (or more).

The parameter k represents freedom in adding new vertices to the independent set. Thus, a k-maximal independent set may not be (k+1)-maximal. Let $I(k, G)$ represent the set of all k-maximal independent sets of a graph G. Then the following is true:

$$I(1, G) \supseteq I(2, G) \supseteq \cdots \supseteq I(|V(G)|, G)$$

The set $I(1, G)$ is precisely the set of independent sets that are *maximal* in the traditional sense. The set $I(|V(G)|, G)$ is the set of all independent sets that are *maximum*.

We are now ready to give a definition for the MINIMUM k-MAXIMAL INDEPENDENT SET problem, formulated as a decision problem.

> MINIMUM k-MAXIMAL INDEPENDENT SET (decision). Given a graph G and two positive integers k and k', is there a k-maximal independent set for G of size less than or equal to k'?

It is easy to show a many-one reduction from the MINIMUM MAXIMAL INDEPENDENT SET problem to the MIN k-MIS problem, or from the compliment of the MAXIMUM INDEPENDENT SET problem to the MIN k-MIS problem. Both the MINIMUM MAXIMAL INDEPENDENT SET and the MAXIMUM INDEPENDENT SET problems are examples of classic NP-complete problems [7]. Thus, the MIN k-MIS problem is hard for the classes NP and $coNP$.

The MIN k-MIS problem is an example of a problem that is *more difficult* than any of the problems in the class NP (under the assumption that $P \neq NP$). In fact, this problem is at least as hard as any of the problems in the class D^P. The class D^P is defined as $\{L_1 \cap L_2 : L_1 \in NP, L_2 \in coNP\}$. The class D^P is a superclass of NP and $coNP$.

Theorem 2.1 *The* MIN k-MIS *problem is hard for the class* D^P.

Proof. We will a give a many-one reduction from the EXACT MAXIMUM INDEPENDENT SET (E-MIS) problem to the MIN k-MIS problem. First, we

need to show that the E-MIS problem is D^P-complete. We define the E-MIS problem below.

> EXACT MAXIMUM INDEPENDENT SET (E-MIS). Given a graph G and a number k, is it true that the largest independent set of G has size exactly k?

The EXACT CLIQUE problem was proven to be complete for D^P [15]. The definition for this problem is given below.

> EXACT CLIQUE (EC). Given a graph G and an integer k, is it true that the largest clique of G has size exactly k?

A *clique* of size k in a graph G is a set of k vertices such that every pair of vertices in the set is connected. The existence of a maximum clique of size k in the graph G implies that there is a maximum independent set of size k in the complement graph of G and vice versa. So, the EC problem is equivalent of the E-MIS problem.

We are now ready to give the reduction from the E-MIS to the MIN k-MIS. Let (G, k) be any instance of the EXACT MAXIMUM INDEPENDENT SET problem. To the graph G, we add

- a set X of $k+2$ additional vertices.
- $(|V(G)| \cdot (k+2))$ additional edges connecting each $x \in X$ to each $v \in V(G)$.
- a vertex y, and
- $k+2$ additional edges connecting y to each $x \in X$.

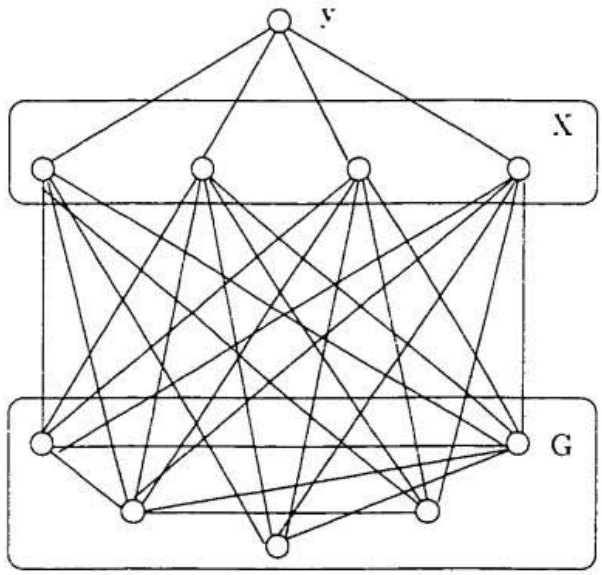

Figure 1: Construction of G' (where $k = 2$)

Call this new graph G'. A diagram of this construction for $k = 2$ is given in Figure 1. This construction adds $O(n)$ extra vertices and $O(n^2)$ extra edges where n is the number of vertices in the graph G. Furthermore, this construction can be performed easily in $O(n^2)$ steps.

We will now show that the graph G contains a maximum independent set of size k if and only if the graph G' contains a $(k+1)$-maximal independent set of size less than or equal to $k+1$.

Assume that I' is a $(k+1)$-maximal independent set of size less than or equal to $k+1$ in G'. If I' contains a vertex in X, then by construction I' does not contain the vertex y or any vertex $v \in V(G)$. Thus, either $I' \subseteq X$ or $I' \subseteq (V(G) \cup \{y\})$. If $I' \subseteq X$ then I' cannot be maximal, since the vertices of X are disjoint and $|I'| < |X|$. So, we can assume that $I' \subseteq (V(G) \cup \{y\})$. In this case, we know that $y \in I'$ since I' is maximal and y is not connected to any vertex in $V(G)$. All that remains is to choose up to k vertices from $V(G)$.

We will now attempt to construct a $(k+1)$-maximal independent set I' of size less than or equal to $k+1$ under the condition that (1) G has a maximum independent set of size less than k, (2) G has a maximum independent set of size greater than k, and (3) G has a maximum independent set of size equal to k. We will find that I' can be constructed if and only if condition (3) holds. From the previous discussion, we can assume that $I' \subseteq (V(G) \cup \{y\})$ and $y \in I$.

(1) If G has a maximum independent set I of size less than k, then there are at most k vertices in the set I' (including the vertex y). In this case, I' would not be $(k+1)$-maximal since $(I' - I') \cup X$ is an independent set of cardinality larger than I'.

(2) Assume that G has a maximum independent set I of size greater than k. Let I' be any independent set of G' such that $I' \subseteq (V(G) \cup \{y\})$ and $|I'| \leq k+1$. Then then set $I'_G = I' \cap V(G)$ has cardinality at most k and the set $(I' - I_G) \cup I$ is an independent set of cardinality larger than I'. Thus, I' cannot be $(k+1)$-maximal.

(3) Let I be a maximum independent set for G. The set $I' = I \cup \{y\}$ is a $(k+1)$-maximal independent set of size $k+1$. We cannot remove the subset I and add $k+1$ vertices, since I is a maximum independent set of G. We cannot add any of the vertices from the set X since each vertex in X is connected to $k+1$ vertices from I, and we can only remove at most k vertices at a time.

To summarize, if G has an independent set of size k then there is an $(k+1)$-maximal independent set of size $(k+1)$ in G', and if G does not have an independent set of size k then there is no $(k+1)$-maximal independent set of size $(k+1)$. So, EXACT MAXIMUM INDEPENDENT SET can be polynomial time many-one reduced to MINIMUM k-MAXIMAL INDEPENDENT SET, and therefore MIN k-MIS is D^P-hard. □

Let us now consider the slightly easier problem of testing an independent set for k-maximality. The definition is given below.

> k-MAXIMAL INDEPENDENT SET (k-MIS). Given a graph $G = (V, E)$, a subset I of the vertices of G, and a positive integer k: is I a k-maximal independent set in the graph G?

We can test an independent set for maximality (under the traditional definition) in $O(n)$ time. This does not hold true for k-maximality. It turns out that the k-MIS problem is *co*NP-complete.

Theorem 2.2 *the* k-MAXIMAL INDEPENDENT SET (k-MIS) *problem is complete for* $coNP$.

Proof: First we will show that the k-MIS problem is a member of $coNP$. Let k-MIS^C denote the complement of the k-MIS problem. If an independent set I is not k-maximal with respect to the graph G, then by definition there exists some $X \subseteq I$ and $Y \subseteq (V(G) - I)$ with $|X| < k$ such that $(I - X) \cup Y$ is an independent set that is larger than I. Thus, we can nondeterministically validate a K-MIS^C instance using the algorithm given below.

```
function Compliment-k-MIS (G,I,k) return boolean is
begin
  check to see if I is an independent set
  if I is not an independent set then
    loop forever
  end if
  guess a number i, where 1 ≤ i ≤ k
  guess a set X ⊆ I such that |X| = i
  guess a set Y ⊆ (V(G) − I) such that |Y| > i
  if such X and Y exist and (I − X) ∪ Y is an independent set then
    return true
  end if
end Compliment-k-MIS
```

We can check to see if the set I is independent by testing each edge of G to see if the edge connects two vertices of I. This be done in $O(n)$ steps, where n is the number of edges in the input graph. We can also guess the subsets $|X|$ and $|Y|$ in $O(n)$ steps. Therefore, the algorithm above has time complexity $O(n)$.

We will now show a many-one reduction from the NP-complete problem MAXIMUM INDEPENDENT SET to the k-MIS^C problem. The definition for the MAXIMUM INDEPENDENT SET problem is given below.

MAXIMUM INDEPENDENT SET. Given a graph G and an integer k, is it true that G has an independent set of size greater than or equal to k?

Let (G, k) be an instance of the MAXIMUM INDEPENDENT SET problem. We will reduce this instance to an instance (G', I', k') of the k-MIS^C problem. To the graph G we add

- A set X of $k-1$ additional vertices, and
- $|V(G)| \cdot (k-1)$ additional edges connecting each vertex $x \in X$ to each vertex $v \in V(G)$.

Call this new graph G'. To complete the reduction we set $I' = X$ and $k' = k$. If we assume that $k \leq |V|$, the construction can be performed in $O(n^2)$ time. If $k > |V|$ then the MAXIMUM INDEPENDENT SET instance is trivially false, so we map these instances to a false instance of k-MIS^C.

If G has a maximal independent set I of size k or greater, then $(I' - X) \cup I$ is an independent set of cardinality larger than k and so I' would not be k-maximal. If G does not have an independent set I of size larger than k then by construction we cannot remove $\ell < k$ vertices from I' and add $\ell + 1$ vertices. So, the set I' is k-maximal with respect to G' if and only if there is a set of k independent vertices in the graph G. The MAXIMUM INDEPENDENT SET can be reduced to the k-MIC^C problem, therefore k-MIS^C is NP-complete and k-MIS is coNP-complete. □

3 Characterization of k-maximality

In this section, we will focus on the structural and algorithmic properties of k-maximal independent sets for tree graphs.

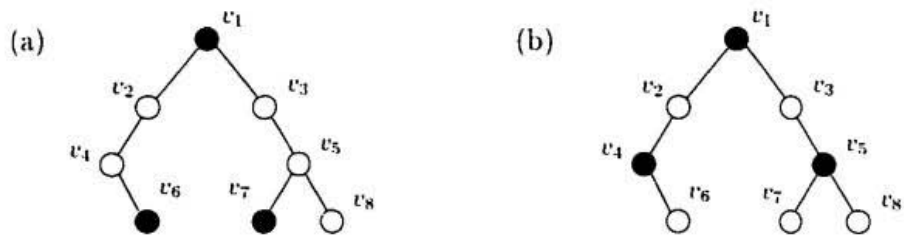

Figure 2: Two independent sets. The independent set (b) is 1-maximal, but the independent set (a) is not.

3.1 Selected cases

We will now observe the structural properties of k-maximal independent sets for certain values of k. We do this so that we can understand the way in which the introduction of the parameter k affects the difficulty of the problem. Later on, we will use the ideas developed in this section to create more general description of k-maximal independent sets.

First, let us consider $k = 1$. This is equivalent to the standard definition of maximality. That is, an independent set I is a 1-maximal independent set for a tree T if there is no vertex $y \notin I$ such that $(I - \{\}) \cup \{y\}$ is independent.

Consider the independent sets in Figure 2. In this figure, we we see two pictorial representations of a tree graph with black nodes representing two independent sets. Let us call these sets I_a and I_b. The set I_a is not 1-maximal, since $I_a \cup \{v_8\}$ is an independent set. The set I_b however, is not 1-maximal. We cannot add any more vertices to I_b, since any new vertex will be adjacent to at least one of the vertices that are already in I_b. Clearly, the following is true:

Observation 3.1 *For tree T and independent set $I \subseteq V(T)$, we can add a*

vertex v to the independent set if and only if v is not adjacent to any vertices in the independent set.

Corollary 1 *An independent set I of T is 1-maximal if and only if there is no vertex $v \in (V(T) - I)$ that is* not *adjacent to any vertex in I.*

The corollary suggests a very simple strategy for detecting 1-maximality. This strategy is outlined in the Test-1-Maximality algorithm given below.

```
    function Test-1-Maximality (T.I) return boolean is
    begin
  for each vertex v ∈ (V(T) − I) loop
  |   for each neighbor vertex w of v loop
  |   |   check to see if w ∈ I
  |   end loop
  |   if v is not adjacent to a vertex in I then return true
  end loop
  return false
end Test-1-Maximality
```

The main loop will be repeated at most $O(n)$ times, where n is the number of vertices in the tree. The neighbor check will be executed exactly twice for each edge. Since there are $O(n)$ edges in the tree, it follows that the total time required to check all of the neighbor vertices will be $O(n)$. Therefore, the algorithm has linear time complexity.

2-maximality

We will now consider the problem of testing a set of independent vertices of a tree for the property of 2-maximality.

Recall that any 1-maximal set is also 2-maximal. So, by the definition of k-maximality we can say that an independent set I is 2-maximal if and only if:

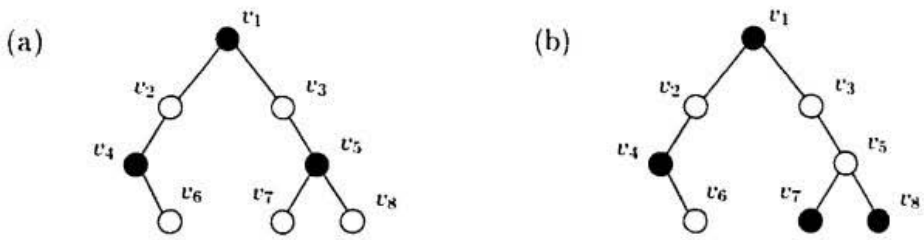

Figure 3: Two independent sets. The independent set (b) is 2-maximal, but the independent set (a) is not.

- I is 1-maximal, or
- there exists subsets $X \subseteq I$ and $Y \subset (V(T) - I)$ such that $|X| = 1$, $|Y| = 2$, and $(I - X) \cup Y$ is independent.

Consider the independent sets of Figure 3. Again, let us call these sets I_a and I_b. The independent set I_a is the 1-maximal set from before. This independent set is not 2-maximal, since $(I_a - \{v_5\}) \cup \{v_7, v_8\}$ is independent. The independent set I_b, however, is a 2-maximal independent set, as it can be easily verified.

The simple strategy adopted in Test-1-Maximality will not help us to detect 2-maximality. In order to detect 2-maximality, we need to consider the effect of removing any one of the vertices from I. We will employ a labeling scheme to help us.

Definition 3.1 *Let $G = (V, E)$ be a graph and let and $I \subseteq V$ be an independent set. The labeling function Γ_I corresponding to the independent set I is a function $\Gamma_I : V \to \{\times, 0, 1, \ldots, |V|\}$.*

For all v in V, we define $\Gamma_I(v)$ as follows:

- *If v is in I, then $\Gamma_I(v) = \times$.*

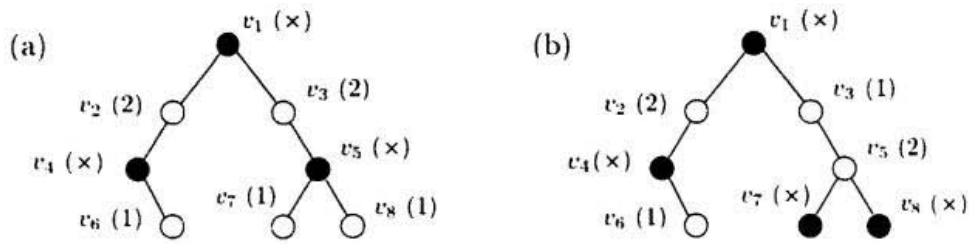

Figure 4: Labeled trees. The independent set (b) is 2-maximal, but the independent set (a) is not.

- *Otherwise.* $\Gamma_I(v) = l$, *where* l *is the number of elements in* I *that are adjacent to* v.

For any given tree T, and independent set I, we can compute all of the labels for all of the vertices of T in linear time by a slight modification of the Test-1-Maximality algorithm.

Once we can have labeled a tree T according to Γ_I, we can prove or disprove the 2-maximality of I. Whenever a 1-maximal independent set is not 2-maximal, certain patterns will appear in the labeling of T. An example labeling is given in Figure 4. These are the sets I_a and I_b from before. Consider the vertices v_5, v_7, and v_8 in I_a. Recall that this independent set is not 2-maximal. We can remove the vertex v_5 from the independent set and then add the vertices v_7 and v_8 to the independent set. Notice that the three vertices involved this transformation are connected in the pattern $1 - \times - 1$. It turns out that any 1-maximal independent set that is not 2-maximal will contain this pattern, so we can use this pattern as a test for 2-maximality.

Lemma 3.2 *Let* I *be a set of independent vertices of a tree* T *such that* I *is* 1*-maximal. Then the set* I *is* 2*-maximal if and only if the labeling of* T

according to Γ_I does not contain a path of three vertices showing the pattern $1 - \times - 1$.

Proof: ($\Longrightarrow$) Let I be any 2-maximal independent set. If the labeling of T according to Γ_I contains a path of three vertices $1 - \times - 1$, then we can increase the cardinality of I by removing the middle vertex and then adding the two end vertices. So I cannot contain the pattern $1 - \times - 1$, as required.

($\Longleftarrow$) Assume that the labeling does not contain a path with the pattern $1 - \times - 1$. If I is not 2-maximal, then there are three vertices x, y_1, and y_2, such that:

- $x \in I$,
- $\{y_1, y_2\} \subseteq (V(T) - I)$, and
- $(I - \{x\}) \cup \{y_1, y_2\}$ is independent.

We will show that in this case the labeling of T must contain the connected path $1 - \times - 1$ in this case.

Now the set I is 1-maximal, so the vertices y_1 and y_2 cannot be added to I unless the vertex x is removed from I. Therefore, x must be adjacent to both y_1 and y_2. Also notice that y_1 and y_2 can only be adjacent to one element of I (namely, the vertex x). If this were not true, then we would not be able to add y_1 and y_2 simply by removing x. Under these conditions, it follows that the path $1 - \times - 1$ must be present in the graph T. This is not possible, therefore I must 2-maximal. □

Now that we know how to detect 2-maximality in a 1-maximal graph, we can detect 2-maximality in general.

Theorem 3.1 *An independent set $I \subseteq V$ in a tree $T = (V, E)$ is 2-maximal if and only if it does not contain a vertex with the label 0 or path of vertices with the pattern $1 - \times - 1$.*

Proof: At the start of Section 3.1, we stated that an independent set is 2-maximal if and only if one of two conditions are satisfied. We know that the first condition is satisfied if and only if $(V(T) - I)$ does not contain a vertex that is not adjacent to any vertex in I. This is equivalent to saying that the labeling Γ_I of T does not contain a vertex with the label 0. By Lemma 3.2, the second condition will hold if and only if the labeling contains a path of vertices in pattern $1 - \times - 1$. □

3-maximality

We will now consider the problem of testing a set of independent vertices of a tree T for the property of 3-maximality.

Recall that any 2-maximal set is also 3-maximal. So, by the definition of k-maximality we can say that an independent set I is 3-maximal if and only if:

- I is 2-maximal, or
- there exists subsets $X \subseteq I$ and $Y \subset (V(T) - I)$ such that $|X| = 2$, $|Y| = 3$, and $(I - X) \cup Y$ is independent.

Consider the independent sets shown in Figure 5. Call these sets I_a and I_b. The independent set I_a is the 2-maximal set from Figure 3. This set is not 3-maximal, for if we let $X = \{v_4, v_1\}$ and let $Y = \{v_2, v_3, v_6\}$, then $(I_a - X) \cup Y$ is independent. However, I_b is a 3-maximal independent set, as it can be easily verified.

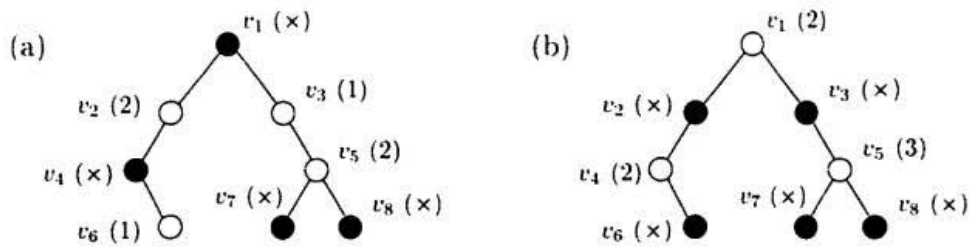

Figure 5: Two independent sets. The independent set (b) is 3-maximal, but the independent set (a) is not.

Notice that, in the independent I_a, the five vertices involved in the replacement are connected in the pattern $1 - \times - 2 - \times - 1$. It turns out that any 2-maximal independent set that is not 3-maximal will contain this pattern, so we can use this pattern as a test for 3-maximality.

Lemma 3.3 *Let I be a 2-maximal independent set with respect to the tree graph T. The set I is 3-maximal if and only if the labeling of T does not contain a path containing five vertices with the labels* $1 - \times - 2 - \times - 1$.

Proof: The proof is similar to that of Lemma 3.2. ($\Longrightarrow$) Assume that I is 3-maximal. If the labeling of T contains a path showing the pattern $1 - \times - 2 - \times - 1$, then we can increase the size of I by removing the two $\times$ vertices from I and then adding the three numbered vertices to I. This contradicts the 3-maximality of I, so the labeling must not contain a path with this pattern.

($\Longleftarrow$) Assume that there are is no path with the labels $1 - \times - 2 - \times - 1$, in order. If I is not 3-maximal, Then there are five vertices, x_1, x_2, y_1, y_2, and y_3, such that:

- $\{x_1, x_2\} \subseteq I$,

- $\{y_1, y_2, y_3\} \subseteq (V(T) - I)$, and
- $(I - \{x_1, x_2\}) \cup \{y_1, y_2, y_3\}$ is independent.

We will show that the labeling of T must contain a path of the form $1 - \times - 2 - \times - 1$.

We can deduce three facts. Firstly, each y_i vertex must be adjacent to either x_1 or x_2. Otherwise, by definition we would be able to add y_i without removing any vertices from the independent set, thus contradicting the 2-maximality of I. Secondly, the vertex x_1 must be adjacent to more than one y vertex. Otherwise, we could add the remaining two y vertices by removing just one vertex (x_2), again contradicting the 2-maximality of I. A similar argument can be applied to show that x_2 is adjacent at least two y vertices. Finally, the vertices x_1 and x_2 must not be adjacent, since they are part of the original independent set.

Under these conditions, it should be clear that the path $1 - \times - 2 - \times - 1$ must be present in the graph T. This is impossible, so I must be 3-maximal. □

Now that we know how to detect 3-maximality in a 2-maximal graph, we can detect 3-maximality in general.

Theorem 3.2 *Let $T = (V, E)$ be a tree and let I be an independent set. The set I is 3-maximal if and only if it does not contain a vertex with the label 0 or sequence of vertices in the pattern $1 - \times - 1$ or a sequence of vertices in the pattern $1 - \times - 2 - \times - 1$.*

Proof: At the start of Section 3.1, we stated that an independent set is 3-maximal if and only if one of two conditions are satisfied. We know that

the first condition is satisfied if and only if the labeling of T does not contain the label 0 or a sequence of vertices in the pattern $1 - \times - 1$. By Lemma 3.2, the second condition will hold if and only if the labeling contains a path of vertices with the pattern $1 - \times - 2 - \times - 1$. □

3.2 Characterization for general k

In the previous section, we have shown how we can test a set of independent vertices of a tree graph for the property of 1-maximality, 2-maximality, or 3-maximality by first labeling the vertices of the tree according to a particular scheme (Γ_I) and then searching for patterns in the labeled tree.

It turns out that this strategy will generalize to any value of k. In this section, we will derive these patterns and show that the patterns correspond one-to-one with the set of *free trees*. A free tree is a tree with no established root.

Recall that an independent set is k-maximal if there does not exist two sets X and Y such that $X \subset I$, $Y \cap I = \emptyset$, $|X| < k$ and $(I - X) \cup Y$ is an independent set that is larger than I. If we compare the definition of k-maximality to $(k+1)$-maximality we observe:

Observation 3.4 *Let $T = (V, E)$ be a tree, and let I be an set of independent vertices of T such that I is k-maximal but not $(k+1)$-maximal for some $k \geq 1$. Then there are subsets X and Y of V such that:*

- $|X| = k$, $X \subseteq I$,
- $|Y| = (k+1)$, $Y \subseteq (V - I)$, *and*
- $((I - X) \cup Y)$ *is a independent set.*

If an independent set I in T is k-maximal and there are no sets X and Y that satisfy the above criteria, then I must also be $(k+1)$-maximal. We can use this fact to prove k-maximality by first testing for 1-maximality, then 2-maximality, and so on up to k. Note that this strategy may not lead to a polynomial time algorithm for detecting k-maximality, since the number of possible configurations of X and Y, as we shall see, will grow exponentially in k. However, the ideas presented in this section will form a basis for a $O(n)$ time algorithm that we will describe later.

For next part of this discussion, we will be considering an independent set I in a tree $T = (V, E)$ that is k-maximal but not $(k+1)$-maximal. By definition of I, there must be two subsets X and Y of V that satisfy the above criteria.

Properties of X and Y

First of all, we show that the vertices of X and Y will form a connected subgraph of T.

Theorem 3.3 *The subgraph of T induced by the vertices $X \cup Y$ is a connected graph.*

Proof: Assume for the sake of contradiction that $X \cup Y$ contains m connected components $G_1, \ldots, G_m$, with $m > 1$. Let $X_i = X \cap V(G_i)$ and $Y_i = Y \cap V(G_i)$, for $1 \leq i \leq m$. By definition, none of the vertices in Y_i are connected to any vertices in X_j for all $j \neq i$. Therefore, $(I - X_i) \cup Y_i$ is a independent set for all i.

By definition, we know that either X_i or Y_i is nonempty for all i. We can deduce that Y_i is nonempty for all i, for otherwise $(I - (X - X_i)) \cup Y$ would be a larger independent set, contradicting the k-maximality of I. We can

also deduce that each X_i will be nonempty, for otherwise $I \cup Y_i$ would be an independent set, contradicting the 1-maximality of T. Hence $X_1, \ldots, X_m$ is a partition of X and $Y_1, \ldots, Y_m$ is a partition of Y.

By the pigeonhole principle, there must be at least one (X_i, Y_i) pair such that $|X_i| < |Y_i|$. The set X_i is smaller than X, since $m > 1$. So $(I - X_i) \cup Y_i$ is an independent set that is larger than I. This contradicts the k-maximality of I, so your assumption must be false. □

As a consequence of Theorem 3.3, we know that the subgraph induced by $X \cup Y$ is a tree. We will call this tree T'. Note that T' may occur anywhere within the tree T.

We can deduce even more information about the way in which the vertices of X and Y are connected.

Observation 3.5 *No two vertices in X are adjacent in the tree T. Likewise, no two vertices in Y are adjacent in the tree T.*

The vertices of X in the graph are disconnected because $X \subseteq I$ and I is an independent set. The vertices of Y are disconnected because the set $(I - X) \cup Y$ is an independent set by definition. Thus the vertices in the tree T' alternate between vertices in X and vertices in Y.

We can also set a minimum on the number of Y vertices that are adjacent to any vertex in X.

Theorem 3.4 *Each vertex $x \in X$ is adjacent to at least two vertices in Y.*

Proof: Clearly, each vertex $x \in X$ must be adjacent to at least one vertex in Y. If this was not true, then we could add the $k + 1$ vertices in Y

by removing the $k-1$ vertices from $X-\{x\}$, contradicting the k-maximality of I.

Now assume for the sake of contradiction that x is adjacent to exactly one vertex $y \in Y$. Let $X' = X - \{x\}$ and $Y' = Y - \{y\}$. None of the vertices in Y' are adjacent to any of the vertices in $I - X'$, thus we can increase the size of the independent set by removing the X' vertices and adding the Y' vertices. But $|X'| < k-1$, which contradicts the k-maximality of I. Thus, our assumption must be false.□

Next we have a rather surprising result about the way in which the X vertices are connected to the Y vertices.

Theorem 3.5 *Each vertex $x \in X$ is adjacent to exactly two vertices in Y.*

In Theorem 3.4 we have shown that each x vertex is adjacent to at least two vertices in Y, and clearly no x vertex can be adjacent to another vertex in X. So, it is sufficient to show that x cannot be adjacent to more than two vertices in the tree induced by $X \cup Y$.

Let T' be the tree induced by $X \cup Y$. Let us say for the sake of contradiction that x is adjacent to m vertices in the tree T', with $m > 2$. Fix x as the root. Now the tree induced by $X \cup Y$ will have m subtrees $T'_1, T'_2, \ldots, T'_m$. Let $X_i = V(T_i) \cap X$ and $Y_i = V(T_i) \cap Y$ for all $1 \leq i \leq m$. Each $y_i \in Y_i$ can only be adjacent to x or a vertex $x_i \in X_i$.

Now we will establish some properties for the sets X_i and Y_i. First of all, the root of each subtree of T' will be a vertex from the set Y. There are at least two subtrees of T', so it follows that $|Y_i| < |Y|$ for all i.

Consider any subtree T'_i of T'. It follows from the k-maximality of T and the cardinality of the Y_i sets that $|X_i| \leq |Y_i| - 1$, for otherwise $((I -$

$(X - X_i)) \cup (Y - Y_i)$ would be an independent set that is larger than I and thus I would not be k-maximal. Now consider any two subtrees T_i' and T_j' of T'. We know that

$$\begin{aligned} & |X_i| \leq |Y_i| - 1. \quad |X_j| \leq |Y_j| - 1 \\ \Longrightarrow & |X_i| + |X_j| \leq |Y_i| + |Y_j| - 2 \\ \Longrightarrow & |X_i \cup X_j \cup \{x\}| < |Y_i \cup Y_j|. \end{aligned}$$

The tree T' contains at least three subtrees, and each of these subtrees contains at least one vertex from Y. It follows that $(Y_i \cup Y_j) \subset Y$. So.

$$\begin{aligned} & |X_i \cup X_j \cup \{x\}| < |Y_i \cup Y_j| < |Y| \\ \Longrightarrow & |X_i \cup X_j \cup \{x\}| < |Y_i \cup Y_j| < k + 1 \\ \Longrightarrow & |X_i \cup X_j \cup \{x\}| < k. \end{aligned}$$

The vertices of $Y_i \cup Y_j$ can only be adjacent to x or vertices in $X_i \cup X_j$. Thus, if we let $X' = X_i \cup X_j \cup \{x\}$ and $Y' = Y_i \cup Y_j$, then $(I - X') \cup Y'$ is a valid independent set that is larger than I. But $|X'| < k$, which is a contradiction since I is k-maximal. □

In contrast, we can connect a vertex in Y to any number of vertices in X.

Observation 3.6 *For all k, there exists a configuration (T', X, Y) such that a vertex $y \in Y$ is adjacent to k vertices in X.*

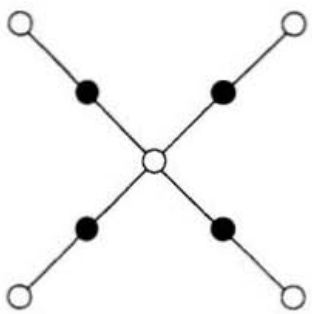

Figure 6: Tree with an independent set shown in black.

Let $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_{k-1}\}$. Let us consider the case where $X \cup Y = V$ and the tree is connected as follows:

- x_i is connected to y_i for all $1 \leq i \leq k$, and
- x_i is connected to y_{k+1} for all $1 \leq i \leq k$.

An example of this graph for $k = 4$ is presented in Figure 6. In the example, it is clear that we can remove four vertices from I and add five, but we cannot remove three vertices and add four. It is easy to show that, in general, the graph described above will be k-maximal but not $(k+1)$-maximal. Thus, a vertex in Y may be connected to any number of vertices in X.

Forbidden Configurations

In this section we will show a one-to-one correspondence between the set of all possible pairs of vertices (X, Y) and the set of free trees.

Let T' be the graph induced by some X and Y. Now consider the graph

F defined in this way:

$$V(F) = \{v_1, \ldots, v_k\}$$
$$E(F) = \{\{v_i, v_j\} | (\exists x \in X)\{\{y_i, x\} \in E(T), \{y_j, x\} \in E(T)\}\}$$

In other words, F graphically expresses the property that two Y vertices y_i and y_j are connected by a third vertex $x \in X$. Using the properties of X and Y outlined in the previous section, we will show that (1) F is a tree and (2) X and Y are uniquely defined by F.

But first, we will formalize our meaning of a configuration.

Definition 3.2 *Define a* configuration *to be a triple* (X, Y) *such that, for some tree* T *and independent set* $I \subseteq V(T)$, *such that* $X \subseteq I$ *and* $Y \subseteq (V(T) - I)$.

- $|X| = |Y| - 1$.
- $(I - X) \cup Y$ *is an independent set of* T.
- *there do not exist two sets* $X' \subset X$ *and* $Y' \subset Y$ *such that* $|X'| = |Y'| - 1$ *and* $(I - X') \cup Y'$ *is an independent set of* T, *and*
- T' *is the tree induced by the subset* $X \cup Y$ *in* T.

The configuration (T', X, Y) contains all the information regarding the arrangement of X and Y within some tree T. Each (T, I) pair may contain many (T', X, Y) triples that satisfy the conditions described above.

Definition 3.3 *Define* $C((T, I))$ *to be the set of all configurations* (T', X, Y) *that can be found in the tree* T *with independent set* $I \subseteq T$.

Notice that the same configuration may be shared by more than one (T, I) pair. For example, if $(T', X, Y) \in C((T, I))$, then $(T', X, Y) \in C((T', X))$.

We will group the configurations according to their size.

Definition 3.4 *Let $\mathcal{C}_k$ be the set of all possible configurations (T', X, Y) such that $|X| = k$ and $(T', X, Y) \in C((T, I))$ for some (T, I).*

We can now restate the definition of k-maximality in terms of configurations. An independent set I is k-maximal with respect to the tree T iff there is no configuration (T', X, Y) such that $(T', X, Y) \in C((T, I))$ and $(T', X, Y) \in \bigcup_{i=0}^{k-1} \mathcal{C}_i$.

In Section 3.1, we solved the k-MIS problem for trees and where $k = 1, \ldots, 3$ by searching for the patterns 0, $1 - \times - 1$, or $1 - \times - 2 - \times - 1$ in the labeling Γ_I of the vertices of the input tree T. In effect, we were searching for the configurations from the sets $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$.

The $\times$ vertices in the patterns correspond to the X vertices of a configuration. The numbered vertices correspond to the Y vertices. Notice that in all three cases, we can determine from the numbers in the labeling that the Y vertices are not adjacent to any of the vertices in $(I - X)$, thus the set $(I - X) \cup Y$ will be an an independent set as required. This idea holds true for the general case, thus we can search for configurations by labeling the input graph according to Γ_I and searching for patterns in this graph.

We are now ready to show the one-to-one correspondence between the $\mathcal{C}_k$ sets and the set of free trees.

Theorem 3.6 *Let $\mathcal{F}_k$ be the set of free trees F such that $|V(F)| = k$. For each fixed k, there is a one-to-one and onto function $g_k : \mathcal{C}_k \rightarrow \mathcal{F}_{k+1}$.*

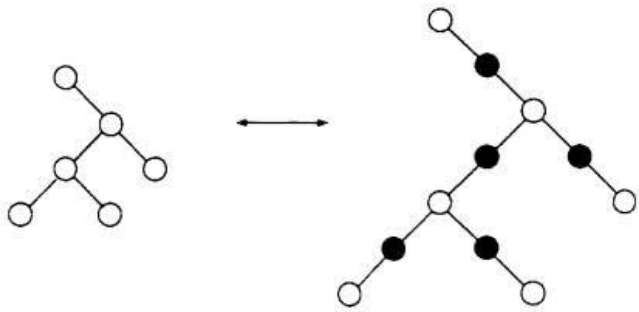

Figure 7: A free tree $F_6 \in \mathcal{F}_6$ and its corresponding configuration

Proof: We construct the function g_k as follows. The vertices $f_1, \ldots, f_{k+1}$ of F_{k+1} will correspond to the vertices $y_1, \ldots, y_{k+1}$ of Y. For each pair $\{f_i, f_j\}$ of vertices in F we join $\{f_i, f_j\}$ with an edge if an only if the vertices y_i and y_j are adjacent to a vertex $x \in X$. An example of a configuration and its corresponding free tree $F_6 \in \mathcal{F}_6$ is given in Figure 7.

We will first show that the constructed graph F is indeed a free tree. Assume for sake of contradiction that there is no path between the vertices f_i and f_j. Then there is no path between the vertices y_i and y_j of T', contradicting Theorem 3.3. Now assume for the sake of contradiction that F contains a cycle. In this eventuality, there would be a cycle in T'. This is impossible, since T' is a subgraph of the tree T. F is a connected graph and F does not contain any cycles, therefore F is a tree as required.

Now we will show that g is one to one. Notice that we can uniquely reconstruct the configuration (T', X, Y) from the tree F. If F has $k+1$ vertices, then there will be k vertices in X, $k+1$ vertices in Y, and thus $2k+1$ vertices in T'. Let $Y = \{y_1, \ldots, y_{k+1}\}$ and let $\{f_1, \ldots, f_{k+1}\}$ be the set of vertices in f. For each edge $\{f_i, f_j\}$ in F, we connect y_i and y_j in T'

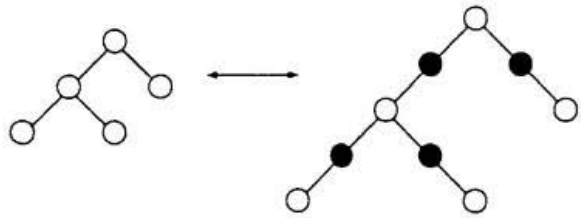

Figure 8: A free tree $F_5 \in \mathcal{F}_5$ and its corresponding configuration

to one of the heretofore unconnected vertices in X. Clearly this will be the original configuration (T', X, Y). There is only way to follow this procedure, so the configuration (T', X, Y) is unique.

It remains to show that g is an onto function. We will prove this by induction. The basis for the induction is F_2. There is only one free tree with two vertices, and that is the graph containing two points with and edge in between. If we use the strategy described above to reconstruct the configuration, we will get the configuration described in the proof of Theorem 3.1. So the function g_k is onto for $k = 1$. For the inductive step, assume that The function $g_{k-1} : \mathcal{C}_{k-1} \rightarrow \mathcal{F}_k$ is onto. We will show that the function g_k is onto. let F_{k+1} be any instance of $\mathcal{F}_{k+1}$. If we remove one leaf node (and the corresponding edge) from F_{k+1} we will get an instance F_k of $\mathcal{F}_k$.

Consider the configuration $g^{-1}(F_{k+1})$. We compute the inverse of g in the manner described previously. The configuration of $g^{-1}(F_k)$ will appear as a substructure of the configuration $g^{-1}(F_{k+1})$. The $g^{-1}(F_{k+1})$ configuration will have two additional vertices: one X vertex connected to the substructure $g^{-1}(F_k)$ and a Y vertex connected to the new X vertex. An example of

this can be seen in Figures 7 and 8. If you compare the two figures, you can see that the second configuration appears as a substructure of the first configuration.

By the inductive assumption we know that $g^{-1}(F_k)$ is in $\mathcal{C}_{k-1}$. Thus if two subsets X and Y are configured according to $g^{-1}(F_k)$ in a tree T, then T is not $(k-1)$-maximal. Let $g^{-1}(F_{k+1}) = (T_{new}, X_{new}, Y_{new})$. If we compare T_{new} to T, we see that T_{new} has two extra vertices. Call these vertices x' and y'. The vertex x' will be connected to y' and some vertex $y \in Y_{new}$ that is part of the substructure defined by $g^{-1}(F_k)$. We show the tree T_{new} is not k-maximal, then we show that $(T_{new}, X_{new}, Y_{new}) \in C((T, I))$ for some tree T and independent set $I \subseteq V(T)$. Hence, $(T(T_{new}, X_{new}, Y_{new})new, X_{new}, Y_{new})$ is a configuration.

We will first show that T_{new} is not k-maximal. There are three cases. (1) If we remove the $k-1$ vertices in X, we will only be able to add $k-1$ vertices (the set $Y - \{y\}$). This is because the vertex x' prevents us from adding the vertex y to the independent set.

(2) If we remove some proper subset S of X, then we can only add the set of vertices in Y_{new} that are neither adjacent to x' nor adjacent to any of the vertices in S (for a total of at most $|S|$ vertices, which is not enough).

(3) If we remove y and some proper subset of $S \subset X$, we can only add y' plus the set of vertices in Y that are only adjacent to vertices in S or x', for at total of $1 + |S|$ vertices. So T_{new} is not k-maximal, as required.

Now if $T = T_{new}$ and $I = X_{new}$ then $(T_{new}, X_{new}, Y_{new}) \in C((T, I))$. Therefore $g^{-1}(F_{k+1})$ is a configuration for all $F_{k+1} \in \mathcal{F}_{k+1}$. Therefore, g_k is onto for all k, which is what we wanted to show. □

With this result, we have developed a characterization of the true in-

stances of the k-MIS problem when the input is restricted to trees. We know that any k-MIS instance (T, I) is a true instance if and only if I is an independent set that does not contain any of the configurations from the sets $\mathcal{C}_0, \ldots, \mathcal{C}_{k-1}$. We know that the $\mathcal{C}_i$ sets are finite because they correspond one-to-one with the set of free trees of a fixed size. We also know that we can generate any set $\mathcal{C}_i$ very easily by starting with the set of free trees of size $i+1$ and using the computable function g_i^{-1}.

However, it is not practical to generate and test for all of the possible configurations. The number of possible free trees will grow exponentially with the size of the input value k. The theory for enumeration of graphs with specific properties was developed by Cayley [4] and continued by Polya [16]. Polya gave a very elegant technique for computing the number of rooted trees of a fixed size. Let trees(x) be the number of rooted trees with x vertices. Then we have:

$$\text{trees}(x) = x \exp\left\{\Sigma_{k=1}^{\infty} \text{trees}(x^k)/k\right\}.$$

The techniques developed by Cayley and Polya were used to produce a counting series for the number of free trees of a fixed size in terms of trees(x) [14]. Let freetrees(x) be the number of free trees with x vertices. Then we have:

$$\text{freetrees}(x) = \text{trees}(x) - \frac{1}{2}(\text{trees}^2(x) - \text{trees}(x^2))$$

The proofs for the two equations above are summarized nicely by Harary and Palmer [8].

So, as we can see, the cardinality of $\mathcal{F}_k$ will grow exponentially as the

value of k increases. The chart below [8] gives the value of $t(k)$ (and thus the cardinality of $\mathcal{F}_k$) for small values of k.

t(k) for $k <= 24$							
k	$t(k)$	k	$t(k)$	k	$t(k)$	k	$t(k)$
1	1	7	48	13	3159	19	317955
2	1	8	115	14	4766	20	823065
3	2	9	286	15	7741	21	2144505
4	4	10	719	16	19320	22	5623756
5	9	11	1301	17	48629	23	14828074
6	20	12	1842	18	123867	24	39299897

4 The k-MIS problem

In this section we will show that the k-MIS problem can be solved in $O(n)$ time for trees. This section is broken down into three parts. In part one we present the theoretical results that are necessary for proving the correctness of our approach. In part two we present a simple $O(n^2)$ algorithm for the k-MIS problem, and in part three we give a linear time algorithm for the problem.

4.1 Detecting C_i configurations

At the end of the last chapter, we established that it is infeasible to solve the k-MAXIMAL INDEPENDENT SET problem by the brute-force approach of searching for all possible configurations of size at most $k-1$. This is because the number of configurations is exponential on the size of the input graph. Further complicating the problem is the fact that some configurations may be very large (on the order of the size of the input graph), thus making it difficult to design a recursive solution to the problem.

Ideally, we would like to have some way of looking at a single vertex v and determining whether or not it is a part of a configuration of a certain size. This is not possible, because this information would depend upon the vertices that are in the vicinity of v. However, it is possible to make a determination about v using a small amount of information about the subtrees that are induced by fixing v as the root of the input tree. This is due to the recursive structure of the $\mathcal{C}_k$ sets: each configuration can be broken up into a number of smaller configurations, and we can join any two configurations into one larger configuration. Thus, it is sufficient to keep a list of the configuration sizes that occur in each of the subtrees of v.

This section is broken up into two parts: one dealing with the case that $v \in I$, and the other dealing with the case that $v \notin I$.

Independent set vertices

We will start with a lemma about the recursive structure of configurations.

Lemma 4.1 *Let $(T, X, Y) \in \mathcal{C}_k$ be a configuration with $k > 0$, and fix any one vertex $x \in X$ as the root (thus $x \in I$). Then the tree T has two subtrees, which we will call T_1 and T_2. If we let $X_i' = X \cap V(T_i)$ and $Y_i = Y \cap V(T_i)$ for $i \in \{1, 2\}$, then (T_1, X_1, Y_1) and (T_2, X_2, Y_2) will be configurations. Furthermore, if $(T_1, X_1, Y_1) \in \mathcal{C}_p$ and $(T_2, X_2, Y_2) \in \mathcal{C}_q$, then $p + q = k - 1$.*

Proof: This is a consequence of the one-to-one correspondence between valid configurations and free trees. Using the function g_k described in Theorem 3.6 we can get the free tree $F_{k+1} \in \mathcal{F}_{k+1}$ that corresponds to the configuration (T, X, Y). Now if we remove the edge in F_{k+1} corresponding to the vertex x in (T, X, Y), then we get two free trees $F_{p+1} \in \mathcal{F}_{p+1}$ and

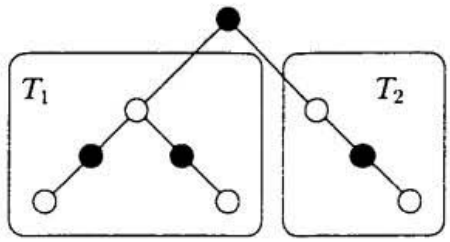

Figure 9: The configuration shown in this figure can be broken down into two smaller configurations.

$F_{q+1} \in \mathcal{F}_{q+1}$ such that F_{p+1} is the free tree that corresponds to the $\mathcal{C}_p$ instance (T_1, X_1, Y_1). F_{q+1} corresponds to the $\mathcal{C}_q$ instance (T_2, Y_2, Y_2), and $(p+1)+(q+1) = (k+1)$. □

An example of this is illustrated in Figure 9. As you can see, the subtree T_1 is part of a configuration from $\mathcal{C}_2$ and the subtree T_2 is part of a configuration from the set $\mathcal{C}_1$. We can imagine that any configuration $\mathcal{C}_k$ with large k will be built out of two smaller configurations, and these will in turn be built out of smaller configurations, and so on.

The inverse of Lemma 4.1 is also true:

Lemma 4.2 *Let (T_1, X_1, Y_1) and (T_2, X_2, Y_2) be any two configurations, where T_1 and T_2 are arbitrarily rooted at vertices $y_1 \in Y_1$ and $y_2 \in Y_2$ respectively. Let T be a tree with a root vertex x and two subtrees T_1 and T_2. If we let $X = X_1 \cup X_2 \cup \{x\}$ and $Y = Y_1 \cup Y_2$, then (T, X, Y) is a configuration. Furthermore, if $(T_1, X_1, Y_1) \in \mathcal{C}_p$ and $(T_2, X_2, Y_2) \in \mathcal{C}_q$, then $(T, X, Y) \in \mathcal{C}_{p+q+1}$.*

Proof: Slight modification of the proof of Lemma 4.1. □

Notice that in Lemma 4.2 the particular form of (T_1, X_1, Y_1) and (T_2, X_2, Y_2) does not affect the construction of the $\mathcal{C}_{p+q+1}$ instance.

In the next theorem, we show how Lemmas 4.1 and 4.2 can be applied to help determine whether or not an independent set vertex r is part of a configuration of a certain size.

Theorem 4.1 *Let (T, I) be such that T is a tree and $I \subset V(T)$ is an independent set. Fix a vertex $r \in I$ as the root. Thus T will have m subtrees $T_1 \ldots T_m$. Now, there exists a configuration $(T', X, Y) \in \mathcal{C}_k$ such that $r \in X$ if and only if there are two distinct subtrees T_i and T_j of T such that:*

- *T_i contains a configuration $(T_i', X_i, Y_i) \in \mathcal{C}_p$ such that the root of T_i is in Y_i;*
- *T_j contains a configuration $(T_j', X_j, Y_j) \in \mathcal{C}_q$ such that the root of T_j is in Y_j; and*
- *$p + q = k - 1$.*

Proof: According to Lemma 4.1, if r is part of a $\mathcal{C}_k$ configuration then we can split this configuration into two smaller configurations by removing the vertex r. These are the two configurations (T_i', X_i, Y_i) and (T_j', X_j, Y_j). According to Lemma 4.2, the existence of the (T_i', X_i, Y_i) and (T_j', X_j, Y_j) configurations imply that r is part of a configuration from the set $\mathcal{C}_k$. Thus the conditions described above are both necessary and sufficient. □

With this result, we see that we only need a small amount of information about each subtree to determine if r is a part of a configuration of size k. For each subtree T_i we keep track of k boolean values indicating whether of not the root of the subtree T_i is part of a configuration in $\mathcal{C}_j$ for $0 \leq j \leq (k-1)$.

Notice that we do not need to know the form of the configuration, just the size. We will store this information in a two dimensional array a of boolean values, where $a[i, j]$ is true iff the root of the ith subtree is part a configuration from the set $\mathcal{C}_j$.

Once we have this information about the subtrees of r, then we can determine if r is part of a $\mathcal{C}_k$ by the algorithm below.

```
function Test-indset-vertex-for-Ck (k,a) return boolean is
begin
  for each pair i,j of unique numbers from {1,...,m} loop
    for each pair of nonnegative numbers p,q such that p+q = k-1 loop
      if a[i,p] and a[j,q] then return true
    end loop
  end loop
  return false
end Test-indset-vertex-for-Ck
```

The outer loop will be executed $O(m^2)$ time where m is the number of subtrees, and the inner loop will be executed $O(k)$ times. The answer is trivially false if $k > |V(T)|$, so we can assume that $k \leq n$. Thus, this algorithm will run in $O(n^3)$ time.

We can simplify the problem considerably by taking note of the fact that we will not be looking for configurations of some *exact* size when we solve the k-MIS or MIN k-MIS problems. Rather, we would only need to prove that r was part of some configuration of size *less than* k. In this case, it is sufficient to determine the size of the smallest possible configuration that includes r, and compare this configuration to the value k.

Naturally, the smallest configuration would be built out of the two smallest sub-configurations. So, for each tree we only need to keep track of one value, which is the size of the smallest possible configuration that includes

the root of that subtree. We will store these values in an m-length integer array a. These values will be computed by the function **min-sizeof-Ck-white-r** which we will give later. In the case where the subtree does not contain any configurations, we use some sentinel value instead (say -1).

The algorithm is given below.

```
function min-sizeof-Ck-black-r (T.r, I) return integer is
begin
  if T is a leaf then return -1
  ' construct the array a
  idx ← 1
  for each subtree T_i of T loop
    if root(T_i) ∉ I then
      a[idx] ← min-sizeof-Ck-white-r(T_i, root(T_i))
      idx ← idx + 1
    end if
  end loop
  ' compute minimum C_k
  if there are less than two nonnegative array elements then
    return -1
  end if
  find the two smallest nonnegative elements a[i] and a[j]
  return a[i] + a[j] + 1
end min-sizeof-Ck-black-r
```

We can determine the number of nonnegative elements in the array in $O(m)$ time, and we can find the two smallest nonnegative elements in $O(m)$ time. Thus, the algorithm has a running time of $O(m)$ excluding the time taken to execute **min-sizeof-Ck-white-r**.

Non-independent set vertices

In this section, we will show how we can check to see if the non-independent set vertex v is part of a configuration of a certain size, using information

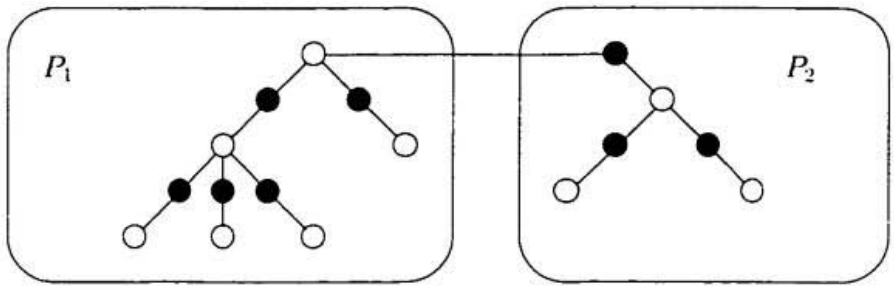

Figure 10: Two partial configurations P_1 and P_2 can be formed by removing an edge from (T, X, Y)

obtained from the subtrees of v. But before we start, we need to introduce the notion of a partial configuration.

Definition 4.1 *Define a* partial configuration *to be a three tuple* (T', X', Y') *such that there exists some configuration* (T, X, Y) *where*

- T' *is one of the connected components that is created by deleting an edge in the tree* T,
- $X' = V(T') \cap X$, *and*
- $Y' = V(T') \cap Y$.

An example of a partial configuration is given in Figure 10. If we remove or 'cut' the central edge, we form two partial configurations P_1 and P_2.

Partial configurations may have one of two forms. As we showed before, the subtree of a configuration may also be a configuration. This situation is illustrated by the tree P_1 in Figure 10 if we fix the topmost vertex of P_2 as the root. The other possibility is that a partial configuration may be a configuration with an extra X vertex attached to one of the Y vertices. This

is illustrated by the tree P_2. It is the latter case that we are interested in, so we will give this case a separate definition.

Definition 4.2 *A strictly partial configuration is partial configuration* (T', X', Y') *such that* $(T', X', Y') \notin C_i$ *for any* i. *Furthermore, for all* $k > 0$, *define the set* $\mathcal{P}_k$ *to be the set of strictly partial configurations such that* $|X'| = k$.

We will now formalize the properties of the strictly partial configurations.

Let (T, X, Y) be any configuration, and let (T', X', Y') be a strictly partial configuration that is formed by taking a subtree of T. Then there is only one vertex of degree 1 in T' that in the set X. Recall that (T', X', Y') is formed by choosing and edge of (T, X, Y) and then considering all of the vertices and edges that occur to one side of that edge. Now, any edge that is part of a configuration will be between two vertices: one $x \in X$, and one $y \in Y$. If we consider the vertices and edges on the Y side of the cut, then we get a partial configuration that is also a configuration. This is a consequence of Lemma 4.1. Thus, a strictly partial configuration can only be formed by considering the X side of the cut. It follows that T' will have exactly one vertex of degree 1 from the set X, and that will be the vertex that is incident with the cut edge.

Let x be the X vertex from (T', X', Y') that is incident with the cut edge. If T'' is the tree induced by $(X' - \{x\}) \cup Y'$, then $(T'', X' - \{x\}, Y')$ is a configuration. This is also a consequence of Lemma 4.1.

We are now ready to present a lemma that is analogous to Lemma 4.1.

Lemma 4.3 *Let* $(T', X, Y) \in C_k$ *be a configuration with* $k > 0$, *and fix any one vertex* $y \in X$ *as the root. Then the tree* T' *has* m *subtrees, which we will call* $T'_1, \ldots T'_m$. *If we let* $X'_i = X \cap V(T'_i)$ *and* $Y_i = X \cap V(T'_i)$ *for*

$1 \leq i \leq m$. *then* (T_i', X_i, Y_i) *will be a strictly partial configuration for all* i. *Furthermore, if* π *is an integer function such that* $\pi(i) = |X_i|$ *for all* $1 \leq i \leq m$, *then* $\Sigma_{i=1}^{m} \pi(i) = k$.

Proof: We know that each of the (T_i', X_i, Y_i) tuples are partial configurations by definition. But they are also strictly partial configurations, since they each have a degree 1 vertex from the set X, (namely, the root of each T_i'), and we know that every degree 1 vertex in a configuration is from the set Y. Furthermore the sets $X_1, \ldots, X_m$ form a partition of the set X, and so $\Sigma_{i=1}^{m} \pi(i) = k$ as required. □

The inverse of Lemma 4.3 is also true:

Lemma 4.4 *Let* $\{(T_1', X_1, Y_1), \ldots, (T_m', X_m, Y_m)\}$ *be any nonempty set of* m *strictly partial configurations. Also for each* $1 \leq i \leq m$, *let* $x_i \in X_i$ *be the unique vertex in* X_i *such that* x_i *is a degree 1 vertex in* T_i'. *Fix* x_i *as the root of* T_i' *for all* i. *Then there is a unique tree* T' *with root vertex* y *and* m *subtrees* $T_1', \ldots, T_m'$.

If $X = \bigcup_{i=1}^{m} X_i$, $Y = (\bigcup_{i=1}^{m} Y_i) \cup \{y\}$, *and* T' *is as described above, then* $(T', X, Y) \in \mathcal{C}_k$, *where* π *is an integer function such that* $\pi(i) = |X_i|$ *for* $1 \leq i \leq m$ *and* $\Sigma_{i=1}^{m} \pi(i) = k$.

Proof: We will prove this by induction. Our basis will be the case where $m = 1$. If y was not connected to any vertex, then y by itself would form a configuration from the set $\mathcal{C}_0$. But y is connected to the vertex $x_1 \in X_1$ from the strictly partial configuration (T_1', X_1, Y_1). Now we know from our discussion of strictly partial configurations that if we 'remove' the vertex x_1 from (T_1', X_1, Y_1) we will get a configuration. Thus, x_1 joins

two configurations in the manner of Lemma 4.2. Therefore, (T', X, Y) is a configuration as required.

For the inductive step, assume that we can connect $(m-1)$ strictly partial configurations to the vertex y to form a configuration from the set $\mathcal{C}_p$ for some p. Similar to the basis case, x_m connects the $\mathcal{C}_p$ instance to a $\mathcal{C}_q$ instance that is part of the (T'_m, X_m, Y_m), in the manner of Lemma 4.2. Thus the first part of this lemma is true by induction. Furthermore the sets $X_1, \ldots, X_m$ form a partition of the set X, and so $\Sigma_{i=1}^{m} \pi(i) = k$ as required.

Combining Lemmas 4.3 and 4.4, we get the following theorem:

Theorem 4.2 *Let the pair (T, I) be such that T is a tree and $I \subseteq V(T)$ is a nonempty independent set I. Fix a vertex $r \notin I$ as the root. Let r be adjacent to exactly m vertices in I, and without loss of generality let $T_1, \ldots, T_l$ with $l \leq m$ be the subtrees of r that have a vertex from I as the root. There exists a configuration $(T, X, Y) \in \mathcal{C}_k$ with $r \in Y$ if and only if there are l strictly partial configurations $(T'_1, X'_1, Y'_1), \ldots, (T'_l, X'_l, Y'_l)$ corresponding to the subtrees $T_1, \ldots, T_l$ such that:*

- *for all $1 \leq i \leq l$, (T'_i, X_i, Y_i) is in the subtree T_i;*
- *for all $1 \leq i \leq l$, the root of T_i is the unique X vertex that has degree 1 in T'_i; and*
- *if π is an integer function such that $\pi(i) = |X_i|$ for all $1 \leq i \leq l$, then $\Sigma_{i=1}^{l} \pi(i) = k$.*

Proof: ($\Longrightarrow$) Assume that (T, I) and r are as described above, and r is part of a configuration from the set $\mathcal{C}_k$. Call this configuration (T, X, Y).

Then according to Lemma 4.3, r will be connected to a number of strictly partial configurations.

Now r is adjacent to l vertices in I, so r cannot be connected to more than l strictly partial configurations. Also note that all of the vertices that are adjacent to r and members of I must be part of the configuration (T, X, Y). Thus, r is connected to exactly l strictly partial configurations $(T'_1, X'_1, Y'_1), \ldots, (T'_l, X'_l, Y'_l)$. The remainder of the proof follows immediately from Lemma 4.3.

($\Longleftarrow$) Assume that $(T'_1, X'_1, Y'_1), \ldots, (T'_l, X'_l, Y'_l)$ are as described above. Then by Lemma 4.4, we can form a new configuration by joining the strictly partial configurations to the vertex r, provided that r is not adjacent to any other vertices in I. thus r will be adjacent to exactly l vertices in I as required. Furthermore, the $\Sigma_{i=1}^{l} \pi(i) = k$ property will hold because $X_1, \ldots, X_l$ forms a partition of the set X, and the vertices of X will be distributed across the partition. □

As was in the case when $r \in I$, we see that we only need a small amount of information about each subtree to determine if r is a part of a configuration of size k. In fact, we only need to look at the l subtrees $T_1, \ldots, T_l$ that have a independent set vertex as their root.

Let $a[i, j]$ be an $l \times k$ matrix of boolean values such that $a[i, j]$ is set to true iff the root of the T_i contains a configuration from the set $\mathcal{P}_j$.

We will use dynamic programming. Let D be an $l \times k$ matrix of boolean values such that $D[i, j]$ is true if and only if there is a configuration $(T, X, Y) \in \mathcal{C}_j$ in the tree induced by r and the subtrees $T_1, \ldots, T_i$. We can can define $D[i, j]$ recursively in this way:

$$D[1,j] = a[1,j] \quad \text{for all } j$$

$$D[i,j] = \begin{cases} true & \text{if there exist } p \text{ and } q \text{ such that } p+q=j \\ & \text{and } D[i-1,a] \text{ and } a[i,b] \\ false & \text{otherwise} \end{cases}$$

Using memoization, we can compute D efficiently. Then we simply return the value of $D[l,k]$. An algorithm is given below:

```
    function test-nonindset-vertex-Ck (k.a) return boolean is
    begin
  set D[1,j] ← a[1,j] for all 1 ≤ j ≤ k
  for i = 2 to l loop
  |  set D[i,j] ← false for all 1 ≤ j ≤ l
  |  for each p,q such that D[i-1,p] and a[i,q] loop
  |  |  if p+q ≤ k then D[i,p+q] ← true
  |  end loop
  end loop
  if D[l,k] then return true
  return false
end test-nonindset-vertex-Ck
```

The inner 'for' loop will be repeated $O(k^2)$ times, and the outer loop will be repeated $O(m)$ times, for a total running time of $O(n^3)$ at most.

Again, It is much easier to prove that a particular vertex v is part of some configuration of size *less than* k. In this case, it is sufficient to determine the size of the smallest possible configuration that includes r, and compare this configuration to the value k. The smallest configuration must be built from the smallest possible partial configurations, so we only need to keep track of the smallest partial configuration for each of the subtrees $T_1, \ldots, T_l$. We will

store these values in an integer vector a in such a way that $a[i]$ contains the size of the smallest partial configuration for T_i. In the case where the subtree T_i does not contain any strictly partial configurations, we set $a[i]$ to -1. The values for a can be computed by the function **min-sizeof-Ck-black-r** which we gave in a previous section.

The algorithm is given below:

```
function min-sizeof-Ck-white-r (T.r, I) return integer is
begin
  if T is a leaf then return 0
  'construct the array a
  idx ← 1
  for each subtree T_i of T loop
    if root(T_i) in I then
      a[idx] ← min-sizeof-Ck-black-r(T_i, root(T_i))
      idx ← idx + 1
    end if
  end loop
  ' compute minimum C_k
  sum ← 0
  for i = 1 to idx loop
    if a[i] = -1 then return -1
    sum ← sum + a[i]
  end loop
  return sum
end min-sizeof-Ck-white-r
```

This algorithm will run in linear time on the number of subtrees of r, excluding the time taken to execute **min-sizeof-Ck-black-r**.

Now that the **min-sizeof-Ck-black-r** and **min-sizeof-Ck-white-r** algorithms have been defined, we see that these algorithms will call each other recursively on the subtrees of T and the results will be gathered at the root. The values at each node can be computed in linear time on the number of

subtrees, and so the total time for this recursive procedure will be $O(|V(E)|)$. In the next section, we will use this procedure to solve the k-MAXIMAL INDEPENDENT SET problem.

4.2 n-pass algorithm for detecting k-maximality

In the last section we gave two algorithms, **min-sizeof-Ck-white-r** and **min-sizeof-Ck-black-r** that can be use to compute the size of the smallest configuration that includes the root vertex r of T. In this section, we will combine these functions together to produce an $O(n^2)$ time algorithm for the k-MAXIMAL INDEPENDENT SET problem.

Let (T, I, k) be an input to the k-MIS problem. If I is an independent set that is not k-maximal, then there exists some vertex $v \in V(T)$ and some configuration (T', X, Y) such that $|X| < k$ and either $v \in X$ or $v \in Y$. Thus it is sufficient to compute, for each v, the size of the smallest configuration (T', X, Y) such that $v \in X$ (in the case that $v \in I$) or $v \in Y$ (in the case that $v \notin Y$). To do this, we will iteratively apply **min-sizeof-Ck-white-r** and **min-sizeof-Ck-black-r** to each vertex. An algorithm is given on the following page.

```
function npass-test-k-maximality (T, I, k) return boolean is
begin
  if I is not an independent set then return false
  for each vertex v ∈ V(T) loop
    size ← 0
    fix v as the root
    if v ∈ I then
      size ← min-sizeof-Ck-black-r(T,v,I)
    else
      size ← min-sizeof-Ck-white-r(T,v,I)
    end if
    if size ≥ k then return false
  end loop
  return true
end npass-test-k-maximality
```

The independent set test can be completed in $O(n)$ time. The two recursive calls can be completed in $O(|V(T)|)$ time. The main loop will be executed $|V(T)|$ times, for a total running time of $O(n^2)$.

4.3 1-pass algorithm for detecting k-maximality

In this section, we present a $O(n)$ time algorithm for the k-MIS. This algorithm will solve the k-MIS problem for any (T, I, k) instance in a single pass through the tree.

Our strategy as we pass up the tree will be to look at every vertex $v \in V(T)$ and determine whether or not v is part of a configuration of size less than k. Simultaneously, we will check to ensure that I is an independent set. If I is an independent set and no v is found such that v is part of a configuration of size less then k, then (T, I, k) is a true instance. Notice that the choice of the root does not matter in this strategy, so at the start of the algorithm we will choose one vertex from $V(T)$ and fix it as the root.

There is some amount of overlap in this strategy. For each configuration (T', X, Y) in (T, I), there will be several vertices that are part of the (T', X, Y) configuration. To reduce the overlap we note that, for any configuration (T', X, Y), there will be one vertex $v \in X \cup Y$ that is *closest* to the root. Any vertex in the set $X \cup Y$ will either be a descendent of the vertex v or will be equal to the vertex v itself.

Given the above result, we see that we can prove k-maximality if we can prove that there is no vertex $v \in T(V)$ such that v is part of a configuration (T', X, Y) where none of the X or Y vertices occur above v in the tree.

For the next part of the discussion, we will need a new definition.

Definition 4.3 *For any tree T with root $r \in V(T)$ and for any vertex $v \in V(T)$, define T_v to be the tree induced by v and all of the descendents of v.*

Consider a vertex v. If $(T', X, Y) \in C((T, I))$ and none of the X, Y vertices occur above v, then $(T', X, Y) \in C((T_v, I \cap V(T_v)))$. On the other hand, $(T', X, Y) \in C((T_v, I \cap V(T_v)))$ does not always imply $(T', X, Y) \in C((T, I))$.

So, we will need some way to test to see whether or not a configuration (T', X, Y) of $(T_v, I \cap V(T_v))$ is in $C((T, I))$. It turns out that this is simple to do.

Let $p(v)$ be the parent of v. First we will consider the case where $v \in X$. If $p(v) \notin I$ it is easy to see that (T', X, Y) must be a configuration in $C((T, I))$. If $p(v) \in I$ then I is not an independent set so we can trivially reject this k-MIS instance.

Now consider the case where $v \in Y$. If $p(v) \notin I$ then again (T', X, Y) will be in $C((T, I))$. But, if $p(v) \in I$ then (T', X, Y) will not be a configuration

in $C((T, I))$ (since we must remove $X \cup \{p(v)\}$ from I before we can add the Y vertices).

So just by looking at the vertex v and the vertex $p(v)$, we can test to see if the potential configuration (T', X, Y) in $C((T_v, I \cap V(T)))$ is a configuration in $C((T, I))$. Now if we wish to prove k-maximality, we must check for all $v \in V(T)$ every configuration of T_v of size less than k to make sure that is it is not a configuration in $C((T, I))$. If a configuration is of size k or larger we can ignore it, since it will not affect the k-maximality of (T, I).

We will now give an outline of our 1-pass strategy. We will compute four values for each vertex $v \in V$. The intuition is, firstly, that we can solve the k-MIS problem if we have these values for the root, and secondly, we can define the values for each vertex v in terms of the children of v. Therefore, we can solve the problem by computing these values for the root recursively. Our four values are described below:

1. A boolean value $isinI$ which is true if v is in I and false otherwise.

2. A boolean value *invalid* which is true if and only if there is some configuration (T', X, Y) of size smaller than k whose 'highest' node is an descendent of v.

3. An integer $config$, which is the size of the smallest configuration (T', X, Y) in $C((T_v, I \cap V(T_v)))$ such that $v \in X \cup Y$. We set $config$ to -1 if there are no configurations of size less than or equal to k.

4. An integer $spconfig$ which is the size of the smallest strictly partial configuration (T', X', Y') in $C((T_v, I \cap V(T_v)))$ such that $v \in X$ and v

has degree 1 in T'. We set $spconfig$ to -1 if there are no such strictly partial configurations of size less than or equal to k.

So if we have these four values for the root vertex r, we can say that the instance (T, I, k) is true if $invalid = false$ and $config = -1$.

In the case that a vertex v is a leaf node, then this information is trivial to obtain. For $v \in I$ we will have $isinI = true$, $invalid = false$, $config = -1$, and $spconfig = -1$. For $v \notin I$ we will have $isinI = false$, $invalid = false$, $config = 0$, and $spconfig = -1$.

Now for each interior vertex we can compute $config$ and $spconfig$ in the same manner as **min-sizeof-Ck-black-r** and **min-sizeof-Ck-white-r**. The value $isin$ can be computed for v by testing for membership in the set I. Finally, $invalid$ must be $true$ if it is true for some descendant of v or if $v \notin I$ and there is some child of v for which $config \neq -1$. Otherwise $invalid$ must be $false$.

We will now give an algorithm for our 1-pass strategy. The function **onepass-Test-k-maximality** solves the k-MIS problem using the functions **get-params-black-r** and **get-params-white-r** to compute the values of $isinI$, $invalid$, $config$, and $spconfig$ for the interior vertices.

```
procedure get-params-black-r (T, r, var invalid,
                              var config, var spconfig) is
begin
  for i = 1 to m loop
    r_i ←root(T_i)
    if r_i ∈ I then
      get-params-black-r(T_i, r_i, invalid[i], config[i], spconfig[i])
    else
      get-params-white-r(T_i, r_i, invalid[i], config[i], spconfig[i])
    end if
  end loop

  invalid ← false
  for i = 1 to m loop
    if invalid[i] = true or root(T_i) ∈ I then invalid ← true
  end loop

  if config[] contains less than 2 nonnegative array elements then
    config ← −1
  else
    find the smallest nonnegative elements config[i], config[j]
    config ← config[i] + config[j]
  end if

  minval ← ∞
  for i = 1 to m loop
    if (root(T_i) ∉ I) and config[i] ≠ −1 then
      minval ←min(minval.config[i])
    end if
  end loop
  if minval = ∞ then spconfig ← (−1) else spconfig ← minval + 1
end get-params-black-r
```

```
procedure get-params-white-r (T. r. var invalid.
                              var config. var spconfig) is
begin
  for i = 1 to m loop
    r_i ←root(T_i)
    if r_i ∈ I then
      get-params-black-r(T_i, r_i, invalid[i]. config[i]. spconfig[i])
    else
      get-params-white-r(T_i. r_i. invalid[i]. config[i]. spconfig[i])
    end if
  end loop

  invalid ← false
  for i = 1 to m loop
    if invalid[i] = true or (config[i] ≠ −1 and (root(T_i) ∉ I)) then
      invalid ← true
    end if
  end loop

  sum ← 0
  for i = 1 to m loop
    if root(T_i) ∈ I then
      if spconfig[i] = −1 then
        config ← −1
        return
      else
        sum ← sum + spconfig[i]
      end if
    end if
  end loop
  config ← sum

  spconfig ← −1
end get-params-white-r
```

```
    function onepass-test-k-maximality (T, I, k) return boolean is
    begin
  | fix a vertex r ∈ V(T) as root
  | if r ∈ I then
  |   get-params-black-r(T, r, invalid, config, spconfig)
  | else
  |   get-params-white-r(T, r, invalid, config, spconfig)
  | end if
  |
  | if invalid = true then
  |   return true
  | else if config = -1 then
  |   return true
  | else
  |   return false
end onepass-test-k-maximality
```

The functions **get-params-black-r get-params-white-r** will take $O(m)$ time, where m is the number of children of the current vertex. Thus the total time taken for **onepass-test-k-maximality** will be $O(|E|)$.

5 Dynamic programming algorithm for min k-mis problem

In this section, we will present a dynamic programming algorithm for the MINIMUM k-MAXIMAL INDEPENDENT SET (DECISION) problem for trees. Recall the MIN k-MIS problem takes as input a graph G and two integers k and k', and returns true if there exists a k-maximal independent set I for G of size less than or equal to k.

The input to this problem will be a tree-graph T and an integer $k < |V(T)|$. The first step will be to fix one vertex $r \in V(T)$ as the root of T. Then for each vertex $v \in V(T)$, we will compute the size of the smallest

independent set for the tree T_v under a fixed number of conditions. We will need $2k+1$ conditions. The conditions are such that for each vertex v, we can compute size of the optimal independent sets for T_v given the size of the of the optimal independent sets for the children of v.

The first k conditions will be cases where v is part of the computed optimal independent set. The second set of $k+1$ conditions will be cases where v is not part of the independent set. In the next part of this section, we will describe these $2k+1$ conditions in detail. After that, we will show how we can compute the size of the optimal independent sets for T_v given the sizes of the optimal independent sets for the subtrees of T_v.

5.1 The $2k+1$ conditions

Recall the 1-pass algorithm for k-maximality given in Section 4.3. In that section, we described how we could determine the size (measured by $|X|$) of the smallest configuration that included the root vertex r of T using information about the smallest configurations and the smallest strictly partial configurations in the subtrees of T.

We will use a similar idea here. For each subtree, we will optimize the subtree according to a number of conditions. Each condition will place different restrictions on the number of configurations and strictly partial configurations that can occur in the subtree.

Independent set vertices

Definition 5.1 *Let T be a tree with root r. We define the set $\mathcal{I}_P(T, r, i, k)$ to be the set of all independent sets I of T that satisfy the following criteria:*

- *all the configurations of (T, I) are of size k or greater, and*

- *all the strictly partial configurations* (T', X, Y) *such that* $r \in X$ *and* r *has degree 1 in* T' *are of size* i *or greater.*

Notice that the set $\mathcal{I}_P(T, r, 1, k)$ is precisely the set of k-maximal independent sets in the rooted tree T under the assumption that $r \in I$. Also, if we start with $\mathcal{I}_P(T, r, 1, k)$ and we incrementally increase the value of the third parameter, we incrementally reduce the number of independent sets that are included in $\mathcal{I}_P(T, r, i, k)$.

By definition the following inclusion holds:

$$\mathcal{I}_P(T, r, 1, k) \supseteq \mathcal{I}_P(T, r, 2, k) \supseteq \cdots \supseteq \mathcal{I}_P(T, r, k-1, k) \supseteq \mathcal{I}_P(T, r, k, k)$$

Now we are ready to describe the first set of k conditions. For each i, $1 \leq i \leq k$, we will find the size of the smallest independent set in $\mathcal{I}_P(T, r, i, k)$. We will store the sizes of these independent sets in k variables labeled $p_k, p_{k-1}, \ldots p_1$, with each p_i corresponding to the set $\mathcal{I}_P(T, r, i, k)$. If the set $\mathcal{I}_P(T, r, i, k)$ is empty, then we will set the value of p_i to infinity. The following inequality will hold:

$$p_1 \leq p_1 \leq \cdots \leq p_{k-1} \leq p_k$$

Non-independent set vertices

Definition 5.2 *Let* T *be a tree with root* r. *We define the set* $\mathcal{I}_C(T, r, i, k)$ *to be the set of all independent sets* I *of* T *that satisfy the following criteria:*

- *all the configurations of* (T, I) *that do not include the root* r *are of size* k *or greater, and*
- *all the configurations that include the root* r *are of size* i *or greater.*

Notice that the set $\mathcal{I}_C(T, r, k, k)$ is precisely the set of k-maximal independent sets for the tree T with root r under the assumption that $r \notin I$. As we reduce the value of the third parameter, we gradually increase the number of independent sets that are included in $\mathcal{I}_C(T, r, i, k)$.

By definition the following inclusion holds:

$$\mathcal{I}_C(T, r, k, k) \subseteq \mathcal{I}_C(T, r, k-1, k) \subseteq \cdots \subseteq \mathcal{I}_C(T, r, 1, k) \subseteq \mathcal{I}_C(T, r, 0, k)$$

Now we are ready to describe the second set of $(k+1)$ conditions. For each i, where $0 \leq i \leq k$, we will find the smallest independent set from $\mathcal{I}_C(T, r, i, k)$. We will store the sizes of these independent sets in $(k+1)$ variables named $c_k, c_{k-1}, \ldots, c_0$, with each c_i corresponding to the set $\mathcal{I}_C(T, r, i, k)$. If the $\mathcal{I}_C(T, r, i, k)$ set is empty, then we will set the value of c_i to infinity. Now the following inequality will hold:

$$c_k \geq c_{k-1} \geq \cdots \geq c_1 \geq c_0$$

5.2 Top-level algorithm

Consider the c_i and p_i values discussed in the previous section. If we can to compute these values for the root vertex r of T, then it is an easy matter to find the size of the minimum k-maximal independent set.

From our previous discussion $\mathcal{I}_P(T, r, 1, k)$ is the set of all k-maximal independent sets such that $r \in I$, and $\mathcal{I}_C(T, r, k, k)$ is the set of all k-maximal independent sets such that $r \notin I$. The p_1 and c_k values are the sizes of the smallest independent sets in $\mathcal{I}_P(T, r, 1, k)$ and $\mathcal{I}_C(T, r, k, k)$ respectively. Therefore, the size of the smallest k-maximal independent set is the smallest

of p_1 and c_k.

5.3 Minimizing the subtrees

In this section, we will describe how one can compute the size of the p_i and c_i values for a tree T_r given the same information for the subtrees of T_r.

This task is naturally split into two steps: first we assume that $r \in I$ and we compute the p_i values, then we assume $r \notin I$ and compute the c_i values.

Suppose r is a leaf node. If $r \in I$, then there would be no strictly partial configurations in $(T_r, I \cap V(T_r))$ and the cardinality of $I \cap V(T_r)$ is 1. Thus $p_1 = \cdots = p_{k+1} = 1$. If $r \notin I$, then $(T_r, I \cap V(T_r))$ contains one configuration from $\mathcal{C}_0$ and the cardinality of $I \cap V(T_r)$ is 0. Thus $c_0 = 0$ and $c_1 = \cdots = c_k = \infty$.

For the remainder of this section will assume that the c_i and p_i values for the children of v are stored in two dimensional arrays c and p. Let $c[i, j]$ be the value for c_j for the ith subtree and let $p[i, j]$ be the value for p_j for the ith subtree.

Computing $p_1, \ldots, p_k$

In this section we will show how the p_i values for each vertex can be computed recursively. We begin by establishing some properties of the $\mathcal{I}_P(T, r, i, k)$ sets. Then we will show how the smallest member of $\mathcal{I}_P(T, r, i, k)$ can be built from suboptimal components. For the remainder of this section, let T be a rooted tree with root r and m subtrees $T_1, \ldots, T_m$ corresponding to m vertices $v_1, \ldots, v_m$.

The first two lemmas outline some necessary conditions for membership in $\mathcal{I}_P(T, r, i, k)$.

Lemma 5.1 *for all* $1 \leq i \leq k$ *and all* $1 \leq j \leq m$, *if* $I \in \mathcal{I}_P(T, r, i, k)$, *then the set* $I \cap V(T_j)$ *is in* $\mathcal{I}_C(T_j, v_j, i-1, k)$

Proof: Suppose $I \cap V(T_j)$ is not in the class $\mathcal{I}_C(T_j, v_j, i-1, k)$ for some j. Then $(T_j, I \cap V(T_j))$ contains either a configuration of size less than k that does not include the vertex v_j, or a configuration of size less than $i-1$ that includes v_j. In the first case, the set I will not be k-maximal, a contradiction. In the second case, the pair (T, I) would contain a strictly partial configuration (T', X', Y') of size less than i such that $r \in X'$ and r has degree 1 in T', a contradiction. □

Lemma 5.2 *If* $I \in \mathcal{I}_P(T, r, i, k)$, *then there are no two vertices* v_a *and* v_b, *such that:*

- $I \cap V(T_a)$ *is in* $\mathcal{I}_C(T, r, p, k)$ *but is not in* $\mathcal{I}_C(T, r, p+1, k)$;
- $I \cap V(T_b)$ *is in* $\mathcal{I}_C(T, r, q, k)$ *but is not in* $\mathcal{I}_C(T, r, q+1, k)$; *and*
- $p + q + 1 < k$.

Proof: Suppose v_a and v_b exist with the defined properties. Then $(T_a, I \cap V(T_a))$ contains a configuration from $\mathcal{C}_p$ and $(T_b, I \cap V(T_a))$ contains a configuration from $\mathcal{C}_q$. It follows that (T, I) contains a configuration of size $p + q + 1$ where $p + q + 1 < k$, a contradiction since $I \in \mathcal{I}_P(T, r, i, k)$. □

The necessary conditions for membership in $\mathcal{I}_P(T, r, i, k)$ described in Lemmas 5.1 and 5.2 are also sufficient.

Theorem 5.1 *I is in $\mathcal{I}_P(T, r, i, k)$ if and only if*

- *For all $1 \leq j \leq m$, $I \cap V(T_j)$ is in $\mathcal{I}_C(T_j, v_j, i-1, k)$*
- *there are no two subtrees T_a and T_b such that the root of T_a is part of a configuration from the set C_p, the root of T_b is part of a configuration from the set C_q, and $p+q+1 < k$.*

Proof: Lemmas 5.1 and 5.2 show that the two conditions are necessary. It remains to show that they are sufficient. The first condition ensures that I does not contain any strictly partial configurations of size smaller than i. The second condition ensures that neither the root of I nor any other vertex of T is part of a configuration of size smaller than k. Thus I is a member of the class $\mathcal{I}_P(T, r, i, k)$. □

Let us demonstrate the use of this theorem with an example. Consider the case where $k = 5$ and the tree T has three children T_1, T_2, and T_3. Assume that we have computed the arrays c and p in for the children of r. We compute $p_1, p_2, \ldots, p_5$ for the vertex r.

Let us start with the value p_5. By Theorem 5.1, $I \in \mathcal{I}_P(T, r, 5, 5)$ implies that $(T_i, I \cap V(T_i))$ is in $\mathcal{I}_C(T_i, v_i, 4, 5)$ for $i = 1, 2, 3$. By the same theorem, we see that this condition is also sufficient. So all that we have to do is find size of the smallest set in each of the three classes $\mathcal{I}_C(T_1, v_1, 4, 5)$, $\mathcal{I}_C(T_2, v_2, 4, 5)$, and $\mathcal{I}_C(T_3, v_3, 4, 5)$. This has already been computed as $c[1, 4]$, $c[2, 4]$, and $c[3, 4]$. Thus we set $p_5 \leftarrow c[1, 4] + c[2, 4] + c[3, 4] + 1$. The values p_4 and p_3 can be computed in a similar way.

The calculation for p_2 is a bit more elaborate. Our strategy will be to optimize I under two disjoint cases, and then take the optimum value of

the two. If $I \in \mathcal{I}_P(T, r, 2, 6)$, then exactly one of the following statements is true:

1. $I \in \mathcal{I}_P(T, r, 3, 6)$, or
2. $I \notin \mathcal{I}_P(T, r, 3, 6)$, so there is a subtree T_i such that $I \cap V(T_i)$ is a member of the class $\mathcal{I}_C(T_i, v_i, 1, 6)$ but is not a member of the class $\mathcal{I}_C(T_i, v_i, 2, 6)$.

We have already optimized I under the first case, so it remains to optimize under the second case.

Assume that the second statement is true. Then the root vertex v_i of T_i is part of a configuration in C_1 for some i. Now it follows from Lemma 5.2 that the set $I \cap V(T_j)$ must be from the class $\mathcal{I}_C(T, r, 3, 6)$ for $i \neq j$. According to Theorem 5.1, this necessary condition is also sufficient. So all we have to do is optimize the independent sets of the subtrees under these conditions. The size of the optimal subtrees are readily available in the $c[i, j]$ array. Of course, we do not know the optimum value for i, so we will try all three possibilities. This can be done in time linear in the number of subtrees.

The value p_1 can be computed in the same way that p_2 was computed.

The algorithm **compute-p-array** on page 59 computes $p_1, \ldots, p_k$ for a vertex r. This algorithm contains two nested loops that are executed k and m times respectively. Thus, we can compute the p_i values for a particular node in $O(k \cdot m)$ time.

```
function compute-p-array (T, r, k) return array of integer is
begin
  for i = 1 to m loop
    c[i, 1..k] ← compute-c-array(T_i.root(T_i).k)
  end loop

  for j = k to 1 by −1 loop
    -compute p_j
    if 2 + j + 1 ≥ k then
      sum ← Σ_{i=1}^{m} c[i, j]
      preturn[j] ← sum
    else
      -find the maximum difference
      hi ← j,    lo ← k − j − 1    idx ← 1
      maxdiff ← (c[1, hi] − c[1, lo])
      for i = 2 to m loop
        if (c[i, hi] − c[i, lo]) > maxdiff then
          maxdiff ← (c[i, hi] − c[i, lo])
          idx ← i
        end if
      end loop

      sum ← 0
      for i = 1 to m loop
        if i = idx then sum ← sum + c[i, lo] else sum ← sum + c[i, hi]
      end loop

      -compare sum to preturn[j + 1]
      if sum < preturn[j + 1] then
        preturn[j] ← sum
      else
        preturn[j] ← preturn[j + 1]
      end if
    end if
  end loop
  return preturn
end compute-p-array
```

Computing $c_0, \ldots, c_k$

In this section we will show how the c_i values for each vertex can be computed recursively. Similarly to the previous section, we will establish some properties of the $\mathcal{I}_C(T, r, i, k)$ classes, and then use these properties to show that the smallest $\mathcal{I}_C(T, r, i, k)$ set can be built from suboptimal components. Let T, T_i, v_i, and r be as described in the previous section.

The first two lemmas outline some necessary conditions for membership in $\mathcal{I}_C(T, r, i, k)$.

Lemma 5.3 *If $I \in \mathcal{I}_C(T, r, i, k)$ then for all $1 \leq j \leq m$, $I \cap V(T_j)$ is either in $\mathcal{I}_C(T_j, v_j, k, k)$ or $\mathcal{I}_P(T_j, v_j, 1, k)$.*

Proof: By the definition of $\mathcal{I}_C(T, r, i, k)$, all of the configurations of (T, I) that do not include the root r must be of size k or greater. Thus, the set $I \cap V(T_j)$ must be a member either of $\mathcal{I}_C(T_j, v_j, k, k)$ in the case that $v_j \notin I$, or $\mathcal{I}_C(T_j, v_j, 1, k)$ in the case that $v_j \in I$. □

Notice that, by the inclusion properties of the $\mathcal{I}_P$ classes, any independent set I in $\mathcal{I}_P(T_j, v_j, 1, k)$ may also in $\mathcal{I}_P(T_j, v_j, q, k)$ for some $q \geq 1$.

Lemma 5.4 *Let us say that the vertex r of T is adjacent to l vertices in I. Without loss of generality, assume that $v_j \in I$ for all $1 \leq j \leq l$ and $v_j \notin I$ for all $(l+1) \leq j \leq m$. If I is in the set $\mathcal{I}_C(T, r, i, k)$, then there is an integer function π such that:*

- *$I \cap V(T_j)$ is in $\mathcal{I}_P(T_j, v_j, \pi(j), k)$ for all $1 \leq j \leq l$; and*
- *$\Sigma_{j=1}^{l} \pi(j) \geq i$.*

Proof: Suppose that that there is no integer function π that satisfies these requirements. Define the function π to be an integer function such that $\pi(j)$ is the size of the smallest strictly partial configuration of T_j that includes the root vertex v_j of T_j. By the assumption, we know that $\Sigma_{j=1}^{l}\pi(j) < i$. Then by theorem 4.2. The root vertex r is part of a configuration of (T, I) of size less than i. This contradicts the membership of I in $\mathcal{I}_C(T, r, i, k)$. □

The two necessary conditions described above are also sufficient.

Theorem 5.2 *Let us say that r is adjacent to l independent set vertices. Without loss of generality, assume that $v_j \in I$ for all $1 \leq j \leq l$ and $v_j \notin I$ for all $(l+1) \leq j \leq m$. Then I is in the set $\mathcal{I}_C(T, r, i, k)$ if and only if*

- *for all $(l+1) \leq j \leq m$, $I \cap V(T_j)$ is in $\mathcal{I}_C(T_j, v_j, k, k)$; and*
- *there is an integer function π such that $\Sigma_{j=1}^{l}\pi(j) \geq i$ and $I \cap V(T_j)$ is in $\mathcal{I}_P(T_j, v_j, \pi(j), k)$ for all $1 \leq j \leq l$.*

Proof: Lemmas 5.3 and 5.4 prove that the conditions described above are necessary. It remains to show that they are sufficient. The second condition ensures that the size of the smallest configuration that includes the root vertex r is of size greater than or equal to i. Both conditions serve to ensure that (T, I) does not contain any strictly partial configurations of size less than k that do not include the root vertex r. Thus, I is a member of $\mathcal{I}_C(T, r, i, k)$. □

Now consider the case where I is the smallest member of the class $\mathcal{I}_C(T, r, i, k)$. By the theorem above, the subtrees of $I \cap V(T_j)$ must each be a member of one of one of the sets $\mathcal{I}_P(T_j, v_j, 1, k), \ldots, \mathcal{I}_P(T_j, v_j, k, k)$.

or $\mathcal{I}_C(T_j, v_j, k, k)$. Since I is as small as possible, it follows that all of the $I \cap V(T_j)$ sets must be as small as possible for their respective classes, otherwise we could replace one of the $I \cap V(T_j)$ sets with a smaller set, contradicting the minimality of I. Therefore, the set I must be built out of suboptimal components.

We will now show how to compute in the value $c_0, \ldots, c_k$ for the root vertex r using the $k+1$ values $p_1, \ldots, p_k$, and c_k corresponding to $v_1, \ldots, v_m$. As before, we will store these values in the two dimensional arrays c and p. Our basic strategy for computing c_i will be to choose the optimal subtrees in such a way that:

- I is in $\mathcal{I}_C(T, r, i, k)$; and
- I is minimal.

For each subtree T_j, we will choose exactly one value from the set

$$\{p[j, 1], \ldots, p[j, k+1], c[j, k]\}.$$

The first step in our strategy is would be to decide which subtrees would benefit from the choice $c[j, k]$. Recall from section 5.1 that $p_1 \leq \ldots \leq p_{k+1}$. So if $c[j, k] \geq p[j, 1]$, there would be no reason to choose the value $c[j, k]$ for the tree T_j; it would always be more beneficial to choose the value $p[j, 1]$.

Let us say that there are ℓ subtrees T_j of T for which $c[j, k] \geq p[j, 1]$. Consider the independent set that is formed by choosing $p[j, 1]$ in the case that $c[j, k] \geq p[j, 1]$ and by choosing $c[j, k]$ otherwise: In other words, we choose $\min(\{p[j, 1], \ldots, p[j, k+1], c[j, k]\})$ for each j. Call this set I^*. The set I^* is the smallest possible independent set that can be formed from the optimal subcomponents.

Without loss of generality, assume that $\{v_1, \ldots, v_\ell\} \subseteq I^*$. Now there is an integer function π such that $I \cap V(T_j)$ is in $\mathcal{I}_P(T_j, v_j, \pi(j), k)$ for $1 \leq i \leq \ell$, and $\Sigma_{j=1}^{\ell} \pi(j) = \ell$. Thus, I^* meets all of the conditions for membership for $\mathcal{I}_C(T, r, 0, k), \ldots, \mathcal{I}_C(T, r, \ell, k)$. I^* is clearly the smallest possible independent set for these classes, so we will set $c_0 = c_1 = \cdots = c_\ell = |I^*|$. Now all that remains is to determine the values for $c_{\ell+1}, \ldots, c_k$.

Our strategy for computing c_i in the case that $i \geq \ell + 1$ will be to start with the independent set I^* and then gradually replace the subcomponents of I^* with more restrictive (and possibly larger) subcomponents, until we can form a function π such that $\mathcal{I}_P(T_j, v_j, \pi(j), k)$ for $1 \leq i \leq \ell$ and $\Sigma_{j=1}^{\ell} \pi(j) = i$. We will design our replacement strategy in such a way that we ensure the minimality of the new independent set.

Say, for example, that the value $c[j, k]$ was chosen for the tree T_j. We may wish to replace the $c[j, k]$ value with the $p[j, 1]$ value. We know by the definition of I^* that $c[j, k] < p[j, 1]$, so the replacement would increase the size of the independent set by some nonnegative integer value. Likewise, we could replace the value $p[j, 1]$ with $p[j, 2]$, the value $p[j, 2]$ with $p[j, 3]$, and so on. Each time we make a replacement, we would also be increasing the maximum size of $\Sigma_{j=1}^{\ell} \pi(j)$ by 1. We will make replacements in this manner until $\Sigma_{j=1}^{\ell} \pi(j) = i$.

We need to minimize the cost of making $i - \ell$ replacements in the manner described above. These replacements will be distributed among the m subtrees. For the nth replacement in the subtree T_j, we associate a positive integer cost. The cost is related to the relative increase in the size of the independent set. It is easy to see that this problem reduces to the problem described below:

OPTIMAL DISTRIBUTION PROBLEM. Given a collection of n balls, m boxes, and an $m \times n$ dimensional array of integers called *cost*, minimize the cost of distributing the $1, 2, \ldots, n-1$, or n balls among m boxes, where $cost[i, j]$ is the cost of placing j balls in the ith box.

We can solve this problem using a divide-and-conquer approach. The algorithm for this problem is given on page 65. This is a recursive function that returns a one-dimensional array called *totalcost*, where $totalcost[i]$ is the minimum cost of placing i balls into the boxes.

The basis step (when $m = 1$) will take $O(n)$ time and will be executed exactly one time per box for a total contribution of $O(n \cdot m)$. The recursive step will take $O(n^2)$ and will be executed $O(m)$ times for a total contribution of $O(n^2 \cdot m)$.

```
function optimal-distribution (n, m, cost) return array of integer is
begin
  if m = 1 then
    -- basis case
    totalcost[0] ← 0
    for i = 1 to n loop
      totalcost[i] = c[1, i]
    end loop
  else
    -- inductive step
    mid = ⌊m/2⌋
    lcost ← optimal-distribution(n, mid, cost[1..mid, 1..n])
    rcost ← optimal-distribution(n, m − mid, cost[(mid + 1)..m, 1..n])
    for i = 1 to n loop
      bestsofar ← ∞
      for j = 0 to i loop
        if lcost[j] + rcost[i − j] < bestsofar then
          bestsofar ← lcost[j] + rcost[i − j]
        end if
      end loop
      totalcost[i] ← bestsofar
    end loop
  end if
  return totalcost
end optimal-distribution
```

The algorithm **compute-c-array** on page 66 computes $c_0, \ldots, c_k$ for a vertex r. This algorithm will be $O(k^2 \cdot m)$ time for each vertex in $V(T)$.

```
function compute-c-array (T, r, k) return array of integer is
begin
  for i = 1 to m loop
    c[i, 0..k] ← compute-c-array(T_i.root(T_i).k)
    p[i, 1..k] ← compute-p-array(T_i.root(T_i).k)
  end loop

  'compute |I*|, ℓ, and cost.
  size ← 0, ℓ ← 0
  for i = 1 to m loop
    if c[i, k] < p[i, 1] then
      size ← size + c[i, k]
      for j = 1 to k loop cost[i, j] ← p[i, j] − p[i, j − 1]
    else
      size ← size + p[i, 1]
      ℓ = ℓ + 1
      cost[i, 1] ← p[i, 1] − c[i, k]
      for j = 2 to k − 1 loop cost[i, j] ← p[i, j + 1] − p[i, j]
    end if
  end loop

  for i = 1 to min({ℓ, k}) loop cret[i] ← size   if ℓ < k then
    totalcost ← optimal-distribution(k − ℓ + 1, m, cost)
    for j = ℓ + 1 to k loop
      creturn[j] ← size + totalcost[j − ℓ]
    end loop
  end if
end compute-c-array
```

5.4 Summary

Now that we have all of the components, we will put them together to form our algorithm for the MIN k-MIS problem. The first step of the algorithm will be to input T and k. Then we fix any $r \in V(T)$ as the root. Recursively, we compute the $2 \cdot k$ conditions for the root vertex, and then we use this information to compute the size of the smallest k-maximal set as in

Section 5.2. The algorithm is given below.

```
    function compute-c-array (T. k. k') return boolean is
    begin
  | fix some r ∈ V(T) as root
  | c[0..k] ← compute-c-array(T.root(r).k)
  | p[1..k] ← compute-p-array(T.root(r).k)
  | if c[k] ≤ k or p[1] ≤ k' then return true
  | elsereturn false
end compute-c-array
```

Let us compute the running time for this algorithm. The basis step will take $O(n)$ time. Then for each interior vertex, we compute the values for the $2k+1$ conditions. It will take $O(k \cdot m)$ time to compute $c_0, \ldots, c_k$ and it will take $O(k^2 \cdot m)$ time to compute $p_1, \ldots, p_k$, where m is the number of children for the vertex in question. The $O(k \cdot m)$ term is negligible so we can say that we take $O(k^2 \cdot m)$ for each interior node. We can tighten the bound further by taking note that the sum of all of the m terms will be $(|V(T)|-1)$. Thus we can say that the total time taken to compute all of the interior vertices is $O(k^2 \cdot n)$, or $O(n^3)$ if we assume that $k \leq n$. Finally the top level calculation of the can be computed in $O(1)$ time. So total time for the algorithm is the sum of the time taken by the basis, top-level, and recursive steps, or $O(n^3)$.

The algorithm that we used to solve the decision version of the MIN k-MIS problem implicitly constructs a k-maximal independent set of smallest cardinality. So, we could easily convert this algorithm so that it returns a k-maximal independent set of smallest cardinality, and thus we could solve the non-decision version of the MIN k-MIS problem. We would do this using traceback pointers. Whenever we compute a c_i or p_i value, we keep track of how the value was generated in terms of the c_i or p_i values at the child

nodes. From these pointers, we can reconstruct the smallest k-maximal independent set.

6 Summary

The k-maximal independent sets have a rich and interesting structure. In this thesis, we have characterized the structure of the k-maximal independent sets for trees. We have used this characterization to solve two problems related to k-maximal independent sets. We first gave a linear time solution to the k-MAXIMAL INDEPENDENT SET problem for trees, and then we gave an $O(n^3)$ time algorithm for the MINIMUM k-MAXIMAL INDEPENDENT SET independent set problem also for trees.

There still are many interesting open problems related to the k-maximal independent sets. Several problems were suggested by Cockayne et al. [5], and many of these problems remain unsolved to this day.

Our success with the MIN k-MIS problem for trees would suggest some other interesting problems. For example, one could investigate the parallel complexity of the MIN k-MIS problem for trees. It seems that this problem may be solved in polynomial time using the technique of parallel tree contraction [12].

Another interesting avenue of research would be to investigate the complexity of the MIN k-MIS problem for tree-width bounded graphs. Many graph problems that are polynomial-time solvable for trees are also polynomial time solvable for graphs with fixed tree-width. A particularly relevant example would be the MINIMUM MAXIMAL INDEPENDENT SET problem [1].

References

[1] V. Beron. and M. Mata-Montero A Linear Time Algorithm for the Minimum α-Maximal Independent Set on Treewidth Bounded Graphs. *TEMAS*. **5, Vol. 2**. pp. 3-15. May-August 1998.

[2] B. Bollobás. E. Cockayne. and C. M. Mynhardt. Generalized Minimal Domination Parameters. *Disc. Math*. **86**. pp. 89-97. 1990

[3] M. Borowiecki. D. Michalak. and E. Sidorowicz. Generalized domination. independence and irredundance in graphs. *Discussiones Matematicae Graph Theory* **17(1)**. pp. 147-153. 1997.

[4] A. Cayley. A theorem on trees. *Quart. J. Math. Oxford Ser.* **23** pp. 376-378. 1889.

[5] E. J. Cockayne. G. MacGillivray and C. M. Mynhardt. Generalised maximal independence parameters for paths and cycles. *Quaestions Mathematicae*. **13(2)**. 1990.

[6] S. Cook. An observation on time-storage trade-off. *Journal of Computer and Systems Science*. **9(3)**. pp. 308-316. 1974.

[7] M. Garey and D. Johnson Computers and Intractability: A Guide to the Theory of NP-Completeness. June 1979.

[8] F. Harary and E. Palmer. Graphical Enumeration. Academic Press New York and London. 1973.

[9] R. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *J. of the ACM*. pp. 762-773. 1985.

[10] D. Manlove. Minimaximal and maximinimal optimisation problems: a partial order-based approach. PhD thesis. University of Glasgow. Department of Computing Science. June 1998.

[11] A. A. McRae. Generalizing NP-completeness proofs for bipartite and chordal graphs. PhD thesis. Clemson University. Department of Computer Science. South Carolina. 1994.

[12] M. Reid-Miller. G. L. Miller. and F. Modugno. List Ranking and Parallel Tree Contraction. In J. H. Rief. editor. *Synthesis of Parallel Algorithms*. Morgan Kayfmann Publishers. Inc.. 1993.

[13] C.M. Mynhardt. Generalised maximal independence and clique numbers of graphs. *Quaestiones Mathematicae*. **11** pp. 383-398 1988.

[14] R. Otter. The number of trees. *Ann. of Math.* **49** pp. 583-599. (1948).

[15] C. H. Papadimitriou and M. Yannakakis. The Complexity of Facets (and Some Facets of Complexity) *J. Comp. Sys. Sci.*. **28**. pp. 244-259. 1984.

[16] G. Pólya and R. C. Read. Combinatorial Enumeration of Groups. Graphs. and Chemical Compounds. Springer-Veralag New York. Inc. 1987.