Thesis

A Parametric Classification of Directed Acyclic Graphs

Submitted by

Mmanu Chaturvedi

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2017

Abstract

A Parametric Classification of Directed Acyclic Graphs

We consider four NP-hard optimization problems on directed acyclic graphs (DAGs), namely, max clique, min coloring, max independent set and min clique cover. It is well-known that these four problems can be solved in polynomial time on transitive DAGs. It is also known that there can be no polynomial $O(n^{1-\epsilon})$-approximation algorithms for these problems on the general class of DAGs unless $P = NP$. We propose a new parameter, β, as a measure of departure from transitivity for DAGs. We define β to be the number of vertices in a longest path in a DAG such that there is no edge from the first to the last vertex of the path, and 2 if the graph is transitive. Different values of β define a hierarchy of classes of DAGs, starting with the class of transitive DAGs. We give a polynomial time algorithm for finding a max clique when β is bounded by a fixed constant. The algorithm is exponential in β, but we also give a polynomial β-approximation algorithm. We prove that the other three decision problems are NP-hard even for $\beta \geq 4$ and give polynomial algorithms with approximation bounds of β or better in each case. Furthermore, generalizing the definition of quasi-transitivity introduced by Ghouilà-Houri, we define β-quasi-transitivity and prove a more generalized version their theorem relating quasi-transitive orientation and transitive orientation.

Acknowledgements

I would like to thank my advisor, Ross, for being so generous with his time, for his patience while listening to my incorrect claims and for sharing insights on a wide variety of topics, ranging from algorithms to The Onion. His enthusiasm for life is contagious and I hope some of it rubbed off on me. I would like express my gratitude to Dr. Michael Kirby and Dr. Sanjay Rajopadhye for their help, advice and support through my graduate studies. I would like to thank Dr. Iuliana Oprea for being a part of my committee. Thanks to Ms. Sharon van Gorder and Ms. Kim Judith for their help and patience in setting up the defense.

A token of thanks to my friends Rohit, Akshat, Louis, Tomojit and Xiaofeng Ma for their wonderful company. Thanks to Kumar Vijay Mishra for all his help during my stay here. I'll be in debt for the rest of my life to my college senior Mohan K.V for his invaluable advice on graduate school in particular and life in general. Lastly, it's superfluous to thank my parents and sister for their unwavering support in whatever I decide to do with my life, there would be little worth speaking about without it.

This thesis is typset in LaTeX using a document class designed by Leif Anderson.

Table of Contents

List of Figures

CHAPTER 1

Introduction

A *directed graph (digraph)* $D = (V, E)$ consists of a finite set $V(D)$, which is the set of *vertices* and $E(D)$, a finite set of ordered pairs of distinct vertices called *edges*. Similarly, an *undirected graph* $G(V, E)$, or simply a *graph* is defined as a special case of symmetric directed graph where each edge $(u, v) \in E(G)$ denotes two directed edges (u, v) and (v, u). In this work, we refer to an undirected graph by G and its variations. To a directed graph we refer to by the letter D and its variations. A *connected digraph* $D = (V, E)$ is a digraph, the vertices of which cannot be partitioned into two subsets, $A, B \subseteq V$, such that there are no edges from any vertex in A to any vertex in B and vice versa. Similarly, the vertices of a *connected undirected graph* $G = (V, E)$, can't be partitioned into two sets $A, B \subseteq V$ such that there are no edges between any vertex in A to any vertex in B. Unless otherwise stated all the digraphs and undirected graphs in this work are connected. For a directed graph, $D = (V, E)$, we represent the cardinality of V by $n(D)$ and that of E by $m(D)$, similarly for an undirected graph, $n(G) = |V|$ and $m(G) = |E|$. When the graph being referred to is clear from the context, we simply use n and m instead of $n(G)$ and $m(G)$. An *orientation* of an undirected graph is giving directions to each of its edges so that the resultant graph is directed. An *acyclic orientation* is an orientation of a undirected graph such that there are no directed cycles in the resulting directed graph.

For an undirected graph or digraph, a *clique* is the set of vertices which which are pairwise adjacent (i.e. have an edge between them), an *independent set* is a set of vertices which are pairwise non-adjacent. A *clique cover* is a partition of the vertices such that each class forms a clique, and *coloring* a graph is to partition its vertices into independent sets, and a set of vertices with the same color is called a *color class*.

An *optimization problem* is a problem with an objective function which needs to be either maximized or minimized subject to a set of constraints, for eg. given an undirected graph, $G = (V, E)$, find a largest set, $S \subseteq V$, such that the vertices in the set S form a clique. A *decision problem* is a problem for which the output is either 0 or 1 for a given set of constraints, for eg. given a constant k, and an undirected graph, G, does there exist a clique of size at least k. Every optimization problem can be converted to a corresponding decision problem [1].

A problem is said to be polynomial in the input, if the complexity of the algorithm is $O(n^k)$, where n is the number of bits needed to represent the input and k is a fixed real number, the corresponding algorithm is called a *polynomial time algorithm.* The class of all problems for which there are polynomial time algorithms is represented by P. The class of problems for which there exists a certificate which can be checked in polynomial time is called NP, thus $P \subseteq NP$. It remains unknown whether $P = NP$, so a decision problem is said to be NP*-Complete* if every problem in NP can be reduced to it in polynomial time.

Directed acyclic graphs (DAGs) are defined as digraphs that have no directed cycles. The structural properties of DAGs have been studied extensively, and due to properties like topological sort a number of problems which are NP-Complete to solve on the general case of digraphs are solvable in polynomial time on DAGs, for example: the decision problems corresponding to the *Hamiltonian path* problem, longest path problem, minimum path cover etc. [1].

Many graph problems of practical importance reduce to finding a max clique, min coloring, max independent set and the min clique cover. We refer to these optimization problems as *the four optimization problems* henceforth. Furthermore, the corresponding decision problems are NP-Complete to solve on the class of undirected graphs. Since an acyclic orientation of an undirected graph can be found in linear time, the four problems can be reduced from undirected graphs to DAGs, and are as a result NP-Complete even on DAGs.

A *transitive* digraph $D = (V, E)$ is a DAG in which if $(u, v) \in E$ and $(v, w) \in E$ then $(u, w) \in E$. The underlying undirected graph for a transitive digraph is called *comparability graph* which belongs to a more general class of graphs called *perfect graphs* [2]. A great deal of research has been done on the structural properties of *perfect graphs* because many problems which are NP-Complete to solve on the general class of digraphs, like the four optimization problems mentioned before, are polynomial time solvable on the class of *perfect graphs*.

It has been shown by Zuckerman [3] that the four optimization problems cannot have an $O(n^{1-\epsilon})$ approximation algorithm on the class of undirected graphs (therefore on DAGs) unless $P = NP$. We thus attempt to partition the set of DAGs parametrically and give approximation algorithms based on this parameter. We define a parameter, β, as the *number of vertices in the longest path in the DAG such the ends of the path are not adjacent* and call the corresponding DAG *β-transitive*. This definition is a natural extension of transitivity. Clearly, $\beta \in \mathbf{N}$. For transitive graphs we define the value of β to be 2. Thus, $\beta \geq 2$ and it measures how much a DAG departs from transitivity.

Hamburger et.al [4] while using DAGs to model preferences introduced a parameter, λ which gave a parameteric characterization of DAGs. They gave an example of a transitive

DAG for which λ could grow unboundedly. Since λ could grow unboundedly for 'simple' DAGs like transitive DAGs, it begs the question if we're capturing a DAG's complexity sufficiently well with λ.

In this work we give an algorithm for computing β for DAGs, give approximation algorithms for the four optimization problems based on β. Additionally, we close the four optimization problems on β-transitive DAGs with a given β by either proving the corresponding decision problems to be NP-Complete or making note of a polynomial time algorithm to solve them. We then extend a theorem given by Ghouilà-Houri [5] on quasi-transitivity.

This thesis is divided into seven chapters. The first chapter introduces the problem, the second chapter gives some results on related graph classes, the third chapter states some of the properties of the parameter β, the fourth chapter gives the approximation algorithms based on β, the fifth chapter gives the NP-Completeness results, sixth chapter gives a result on quasi-transitivity and orienting an undirected graph for minimum β. The last chapter is on the additional work done by the author.

CHAPTER 2

Related Graph Classes

Due to the intractability of important optimization problems like the four optimization problems on the general class of graphs, the problems are studied on the smaller subset of graphs with structural properties which can be exploited to make those problems tractable.

2.1. Comparability Graphs

Comparability graphs are undirected graphs such that the edges of the graphs can be oriented in such a way the resulting digraph is acyclic and transitive. Although the four optimization problems are intractable on the general class of graphs, they are of polynomial time complexity on comparability graphs.

Various results have been published on the recognition of comparability graphs and transitively orienting them, Golumbic [6], Hell and Huang [7] gave a $O(m\Delta)$ algorithm (where Δ is the maximum degree of any vertex in the graph), Moravan et al. [8] gave a parallel algorithm. McConnell and Spinrad [9] gave a linear time algorithm for finding a transitive orientation for a comparability graph.

Once a comparability graph has been oriented so that the resulting digraph is a transitive DAG, the transitivity can be used to give algorithms for the four optimization problems. We first enumerate a few properties of transitive DAGs and then give the algorithms for solving the four optimization problems.

Property 2.1.1. A path of length L in a transitive DAG, $D = (V, E)$, is a clique of size $L + 1$.

Proof. Let any path of length $L(>= 2)$ be $(v_1, v_2, \ldots, v_{L+1})$. If it not a clique in D, then let (i, j) be a two-tuple such that is no edge between v_i and v_j, and j be the smallest integer greater than i with this property. Clearly, $(v_i, v_{j-1}) \in E$ and $(v_{j-1}, v_j) \in E$ but $(v_i, v_j) \notin E$ which contradicts that D was transitive. □

A *source* in a DAG is a vertex which doesn't have any incoming edges. A DAG always has a source. A *sink* in a DAG is a vertex without any outgoing edges. A DAG always has at least one source and at least one sink [1].

Property 2.1.2. Given a DAG $D = (V, E)$, labeling each vertex of the DAG with the length of a longest path from any source to that vertex takes $O(m + n)$ time.

Proof. The sources of any DAG can be found in $O(m + n)$ time by checking each vertex's degree, let the set of all the sources be S. Then we can create a DAG, $D' = (V \cup s, E \cup \{(s, p_i) | p_i \in S\})$. Now we label each edge of D' with an edge weight of -1 and run single-source shortest path (SSSP) algorithm from s. Thus we have the shortest weighted distance from s to all the vertices of D'. Now, the longest path from any source of the original DAG, D, to every vertex is one less than the absolute value of the labels returned by the SSSP algorithm. The longest path can be found by following the back-pointers created while running the SSSP algorithm. Since SSSP for DAGs takes linear time [1], we have an $O(m + n)$ algorithm. □

A *tournament* is a directed graph, $D = (V, E)$, such that for any two vertices, u and v, either $(u, v) \in E$ or $(v, u) \in E$. A *Hamiltonian path* in a digraph, $D = (V, E)$, is a path of length $|V| - 1$.

Property 2.1.3. Every tournament has a Hamiltonian Path.

Proof. Let $D' = (V, E)$ be a minimal counter-example to the claim, with $|V| = n$. Clearly, $n > 3$. Let us remove one vertex, v from D'. Now, by minimality, the remaining graph has a Hamiltonian path, let it be $(v_1, v_2, \ldots, v_{n-1})$. Now let $p \in [1, n-1]$ be the largest integer such that $(v_p, v) \in E$. If no such p exists, then we have, $(v, v_1, \ldots, v_{n-1})$ as a Hamiltonian path in D'. If such a p exists, then we have $(v_1, v_2, \ldots, v_p, v, v_{p+1}, \ldots, v_{n-1})$ as a Hamiltonian path in D'. Thus there exists no such D'. □

2.1.1. Max Clique.

Theorem 2.1.1. *A maximum clique can be found in $O(n+m)$ in a transitive DAG.*

Proof. The longest path found in the transitive DAG using property 2.1.2, would correspond to a clique by property 2.1.1. Thus an L length path would correspond to an $L+1$ size clique. Since by property 2.1.3 we know that every clique has an Hamiltonian path, we can say that the longest path is the largest clique in the transitive DAG. □

2.1.2. Max Independent Set.

To find the max indpendent set in a transitive DAG, we first need to find a minimum *path cover*. A *path cover* for a digraph $D = (V, E)$ partitions the set of vertices V into subsets such that each subset is a path. On the class of digraphs, this problem is NP-Complete [10] but on the class of DAGs it's solvable in polynomial time.

A *matching*, M, in an undirected graph, $G = (V, E)$ is defined as a subset of edges E, such that no two edges in M share a vertex. A max matching is a largest such set possible. An *undirected bipartite graph*, $G = (V = X \cup Y, E)$ is an undirected graph, such that the vertex set, V can be partitioned into two sets, X and Y, such that all edges, E, are between the vertices in X and Y. A *cut*, $(S, V - S)$, of an undirected graph $G = (V, E)$ is a partition of V. A *cut-set* is the set of all the edges which have one end point in S and the other in

$V - S$. The *max network flow* for an directed graph network, $N = (s \cup t \cup V, E, C)$ given the edge capacities $C : E \longrightarrow \mathbb{N}$, is the problem of finding the maximum flow which can go from the *source* s to *sink* t, such that the flow from each edge is less than or equal to its capacity, this problem has been extensively studied [11]. An $s - t$ *cut* is a partition of V into two sets such that one set contains s and the other t. The *weight of an* $s - t$ *cut* for network flow, is the sum of the capacities of the edges going out of S to $V - S$, i.e. $\{(u, v) | u \in S \text{ and } v \in V - S\}$. The max-flow min-cut theorem states that the max-flow and the minimum weight of any $s - t$ cut are equal [1].

Fulkerson [12] proved that the problem of finding a min path cover in a transitive DAG can be reduced to the problem of finding a max matching in a bipartite graph, it can be as it is extended to the general class of DAGs. Hopcroft et al. [13] gave an algorithm for finding max-bipartite matching in $O(\sqrt{n}m)$ time. Dinic's algorithm for networks in which the max flow from any edge can be 1 can be proved to run in $O(\sqrt{n}m)$ as well [11]. We explain the reduction of max-bipartite matching to network flow as it will be useful in finding a max independent set in transitive DAG.

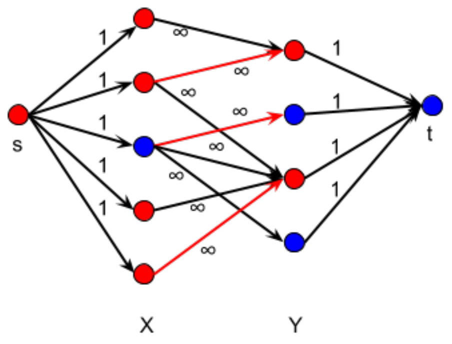

FIGURE 2.1. REDUCTION OF MAX-MATCHING IN BIPARTITE GRAPH TO NETWORK FLOW. THE RED - BLUE VERTICES DENOTE THE CUT.

LEMMA 2.1.2. *It takes $O(\sqrt{n}m)$ to find a min path cover of a DAG, $D = (V, E)$.*

PROOF. Given a DAG, $D = (V, E)$, with $V = \{v_1, v_2, \ldots, v_n\}$, we make an auxiliary undirected bipartite graph, $G' = (V' = X \cup Y, E')$, such that for each v_i we make two vertices, $x_i \in X$ and $y_i \in Y$, and each edge $e \in E$, we make an undirected edge, (x_i, y_i) in G'.

Then to reduce the max bipartite matching in G' to the max flow problem, we introduce a source s, such that s is connected to all the vertices in X and t is connected to all the vertices in Y. We make the capacities of all the edges adjacent to s and t as 1 and the capacities of all edges between X and Y to be ∞, an example is shown in figure 2.1. We call the network N.

According to the *integrality theorem* of flows in networks, if the capacities of all the edges in a network are integers then there always exists a max flow such that flow value in all the edges are integers, and Dinic's algorithm finds a max integral flow. Now, it's easy to see that the max flow in the bipartite graph gives us max matching as well, because of how we have connected s and t, the integrality constraint and how we have given the weights. Let a set that gives max matching be M. Clearly, $|M|$ is the value of the maximum flow. Now, we have the following lemmas due to Fulkerson [12].

LEMMA 2.1.3. *A matching of size $|M|$ in auxiliary graph G' corresponds to a path cover of size $n - |M|$ in D, where n is the number of vertices in D.*

PROOF. Let $\{(x_{i_1}, y_{i_2}), (x_{i_3}, y_{i_4}), \ldots, (x_{i_{2|M|-1}}, y_{i_{2|M|}})\}$ be a max matching in G', such that $i_1 < i_3 < i_5 < i_{2|M|-1}$. Now, these edges can be joined to create a path whenever $x_{i_j} = y_{i_k}$. This corresponds to paths in the original DAG D, since the D is acyclic. Clearly, after joining the edges of the matching and creating single vertex paths for each of the remaining vertices, we have a path cover of size $n - |M|$, as the number of edges in the paths of the created path cover is the same as $|M|$. □

□

A *vertex cover* in an undirected graph or a digraph is a subset, S, of its vertices such that all the edges have at least one vertex from S.

LEMMA 2.1.4. *There is an independent set of size at least equal to the size of a minimum path cover of D. The independent set can be found using maximum matching in G'.*

PROOF. By the max-flow min-cut theorem, we have an $s - t$ cut which determines the max-flow in N, and it can be found while running max-flow algorithm by running a DFS from the source on the residual graph, let $A \subseteq X$ and $B \subseteq Y$ such that A, B are reachable from s in the residual graph. Let $S = \{s\} \cup A \cup B$ and the $s - t$ min-cut be $(S, (X - A) \cup (Y - B) \cup t)$, where $A \subseteq X$ and $B \subseteq Y$. Clearly, $|S| \leq n$, where n is the number of vertices in D, and the equality holds when we have a perfect matching in G'. Now, there can't be any edge from A to $Y - B$ in N, because then the cut will be of ∞ weight. So, we have a vertex cover, $(X - A) \cup B$. The size of $(X - A) \cup B$ is $|M|$ because of how we constructed the network flow graph. The set of vertices in the original DAG, D, corresponding to the vertex cover would be of size at most $|M|$ because each vertex in D corresponds to two vertices in G'. Now when we have a vertex cover of size at most $|M|$ in D, we can get an independent set of size at least $n - |M|$ on D because the complement of a vertex cover is an independent set. □

LEMMA 2.1.5 (Dilworth's Theorem). *If D is transitive, there is an independent set of size equal to the minimum path cover of D.*

Proof. If not, then at least one path in the path cover contains more than one vertex of the independent set. Since the paths correspond to cliques (Property 2.1.1), two vertices from one path can't be a part of an independent set, contradiction. □

Theorem 2.1.6. *In a transitive DAG, a maximum independent set can be found in $O(\sqrt{n}m)$ time.*

Proof. Lemma 2.1.4 explains the way to find an independent set of size at least equal to the size of the minimum path cover. Lemma 2.1.5 proves that the maximum independent set is of size equal to the size of the minimum path cover. So, the maximum set can be found in $O(\sqrt{n}m)$ time. □

2.1.3. Minimum Clique Cover. We first find the relation between minimum path cover and minimum clique cover.

Lemma 2.1.7. *The size of the minimum path cover is at most equal to the size of minimum clique cover.*

Proof. Since every clique has a Hamiltonian path, every clique cover is a path cover, therefore the size of minimum path cover is less than or equal to the size of the minimum clique cover. □

Theorem 2.1.8. *In a transitive DAG, a minimum clique cover can be found in $O(\sqrt{n}m)$ time.*

Proof. A path cover is a clique cover due to property 2.1.1. Due to Lemma 2.1.7, the clique cover found is a minimum one. □

2.1.4. Minimum Coloring.

Theorem 2.1.9. *A minimum coloring can be found in $O(n+m)$ time.*

Proof. We first note that when the vertices are labeled with the longest path ending at them (property 2.1.2), we can use the labels as a coloring. The number of colors used will be equal to $1 + L_{max}$ where L_{max} is the length of a longest path in the transitive DAG. Since, by property 2.1.1 the paths correspond to cliques and the minimum number of colors needed to color a graph is at least equal to the size of the maximum clique, this labeling gives a minimum coloring. $\square$

2.2. Perfect Graphs

For a digraph, $D = (V, E)$, an induced subgraph, $H = (S, E')$, where $S \subseteq V$ and $E' \subseteq E$, is such that all edges whose both end points are in S are contained in E'. Induced subgraphs for undirected graphs are defined similarly. For $G = (V, E)$, let G_A denote the subgraph induced by A, where $A \subseteq V$.

Let $\chi(G)$ we denote the minimum number of colors needed to color G, $\omega(G)$ denote the size of a maximum clique in G, $\alpha(G)$ denote the size of a maximum independent set in G and $k(G)$ denote the minimum number of cliques needed to cover G. *Complement of a graph*, $G = (V, E)$ is defined as $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(u, v) | (u, v) \notin E\}$. An edge (v_i, v_j) between the vertices of a cycle is called a *chord*, and an *odd hole* is an induced cycle of odd length.

Theorem 2.2.1. *[14] [15] The following are equivalent:*

(1) *G = (V,E) is perfect.*

(2) $\omega(G_A) = \chi(G_A)$ *for all* $A \subseteq V$

(3) $\alpha(G_A) = k(G_A)$ *for all* $A \subseteq V$

(4) *$\bar{G}$ is perfect.*

(5) $\omega(G_A)\alpha(G_A) \geq |A|$ *for all* $A \subseteq V$

(6) *In G and $\bar{G}$, every odd length cycle of length greater than 4 has a chord.*

(7) *In G and $\bar{G}$ there is no odd hole of size greater than 3.*

The class of perfect graphs is one of the most extensively studied classes in graph theory. Grötschel and Lovász [16] gave a polynomial time algorithm to find a minimum coloring and maximum independent set on perfect graphs. Since the complement of a perfect graph is perfect, finding minimum clique cover and maximum clique on the original graph, G, is the same as finding minimum coloring and maximum independent set on $\bar{G}$. Thus all four optimization problems can be solved in polynomial time on the set of perfect graphs.

A class of graphs is called *hereditary* if whenever G is a member of the class so is every induced subgraph of G. Clearly, comparability graphs are hereditary. Since $\alpha(G_A) = \omega(G_A)$ for all $A \in V$ for a comparability graph, $G = (V, E)$, comparability graphs are perfect.

2.3. k-extendible Graphs

Spinrad [17] introduced the idea of *k-extendible orderings* and a dynamic programming algorithm on how to calculate the max clique in $O(kn^k)$ time using the property of *k-extendible ordering.* The *k-extendible graphs* are graphs which have a vertex ordering which is *k-extendible.*

Two sets A and B *overlap* if they intersect and neither of them is contained in the other. For an undirected graph, $G = (V, E)$, and an ordering, $\sigma = (v_1, v_2, \ldots, v_n)$, and set of vertices, U, V, W such that $U \subseteq V \subseteq W$, we say V *ends with* U if, no element of $V - U$ is after any element of U. Similarly, we say W begins with V if all the elements of V are before any element of $W - V$.

Definition *k-clique extendible ordering* [4]: An ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of the vertices of $G = (V, E)$ is called a k-extendible ordering, if for any two overlapping cliques, X and Y of size k, if $|X \cap Y| = k - 1$ and $X \cup Y$ begins with $X - Y = \{a\}$ and ends with $Y - X = \{b\}$ in σ, then a and b are adjacent.

CHAPTER 3

Properties of β

In this chapter we discuss a few properties of β, some of which will be used in later chapters when we design approximation algorithms.

Since it's possible to orient an undirected graph in many ways such that the resulting orientation is acyclic, we define the β of an undirected graph to be the minimum β amongst all its possible acyclic orientations. More formally:

Definition *β of an undirected graph*: The minimum possible β amongst all acyclic orientations of an undirected graph.

For a DAG, D we refer to the corresponding β as $\beta(D)$, when the DAG, D is clear from the context, we simply refer to it as β. Similarly, for $\beta(G)$ when the undirected graph is clear from the context, we simply write β.

Unless stated otherwise, we assume that for all the properties enumerated below, the given DAG is β-transitive.

Property 3.0.1. $2 \leq \beta \leq n$

Property 3.0.2. $\beta \leq \lambda + 1$

Proof. Given a DAG $D = (V, E)$, Hamburger et al. [4] create an auxiliary graph, $\tilde{D}$, from D by connecting the unconnected vertices in D with undirected edges. They call the original directed edges in $\tilde{D}$, *edges* and the added undirected edges as *hops*. For a cycle in $\tilde{D}$, we can calculate ratio of edges to hops. λ can be defined as the maximum value of this ratio over all the cycles in $\tilde{D}$. The max path in G with ends not connected will make a cycle in $\tilde{D}$ where the edge to hops ratio would be $\beta - 1$, thus, $\beta \leq \lambda + 1$. $\square$

Property 3.0.3. Given a β-transitive DAG, D, any path of length greater than or equal to β has an edge from the first to the last vertex.

Proof. Follows from definition. $\square$

Property 3.0.4. For a DAG, D, any induced subgraph with a path of length $L-1$ has a clique of size $\lceil L/\beta \rceil$.

Proof. Let the path of length $L-1$ be $(v_1, v_2, \ldots, v_L)$. Due to property 3.0.3, we know that $v_1, v_{\beta+1}, ..., v_{\lceil L/\beta \rceil}$ form a clique. $\square$

Property 3.0.5. For a DAG, D, the subgraph induced by a path can have an independent set of size at most $\lceil \beta/2 \rceil$.

Proof. Consider a path, $P = (v_1, v_2, \ldots, v_L)$ which is a minimal counterexample to the claim. By minimality, the first and the last vertices are disconnected. By the definition of β, $L \leq \beta$, because the first and last vertices are disconnected. Since the largest independent set on a path of length $L-1$ is of $\lceil (L/2) \rceil$ and $\beta \geq L$, the largest independent set on the path P is at most $\lceil \beta/2 \rceil$, contradicting the claim that P is a counterexample. $\square$

Property 3.0.6. For an undirected graph, $\beta \leq \chi$

Proof. Given a color scheme which has the smallest chromatic number, say with partition of the vertices as $A_1, A_2, A_3, \ldots, A_\chi$, if we orient edges from A_i to A_j when $i < j$, then we have the β of the resultant DAG at most χ. $\square$

A *topological sort* of DAG, $D = (V, E)$ is an ordering of the vertices, $\sigma = (v_1, v_2, \ldots, v_n)$ such that if $(v_i, v_j) \in E$, then $i < j$. We prove that the toplogical sort of a β-transitive DAG is in fact a k-extendible ordering for the underlying undirected graph with $k = \beta$.

PROPERTY 3.0.7. A topological sort of a β-transitive DAG, D, is a k-extendible ordering of the underlying undirected graph with $k = \beta$.

PROOF. Given a topological sort of D, let X and Y be two overlapping cliques of size $\beta - 1$, and let $X - Y = \{a\}$ and $Y - X = \{b\}$, such that $X \cup Y$ begins with a and ends with b. In the topological sort clearly there is a path of length β from a to b. Since, by definition of β-transitivity there cannot be a path with vertices more than β with its ends not connected, there has to be an edge between a and b. $\square$

PROPERTY 3.0.8. The subgraph induced by a path of length $L - 1$, i.e. with L vertices in a β-transitive DAG can be divided into β cliques, each of size less than or equal to $\lceil L/\beta \rceil$.

PROOF. Let $(v_1, v_2, \ldots, v_L)$ be an induced path of length $L - 1$, in a β-transitive DAG. Clearly $(v_i, v_{i+\beta}) \in E,\ \forall i \in [1, L - \beta]$. Thus the path can be divided into β cliques each of size at most $\lceil L/\beta \rceil$. $\square$

PROPERTY 3.0.9. The subgraph induced by a path of length $L - 1$ vertices in a β-transitive DAG can be colored with at least $\lceil L/\lceil \beta/2 \rceil \rceil$ colors.

PROOF. Since the maximum size of an independent set on a path in a β-transitive DAG, $D = (V, E)$ is $\lceil \beta/2 \rceil$ (Property 3.0.5), the minimum number of partition classes in a partition of V into independent sets is $\lceil L/\lceil \beta/2 \rceil \rceil$. Since all the vertices in an independent set constituting the partition can be colored with one color, we can have a coloring scheme with $\lceil L/\lceil \beta/2 \rceil \rceil$ colors. $\square$

CHAPTER 4

Computing β and Approximation Algorithms

Approximation algorithms for optimization problems compute not necessarily the exact solution but a solution which is in a certain bound of the exact solution. More formally, say, we have a maximization problem, the exact solution to which is C^*, a ρ-approximation algorithm gives a result, C, such that:

$$\frac{C^*}{C} \leq \rho$$

Similary, for a minimization problem, the exact solution to which is C^*, a ρ-approximation algorithm gives, C, such that:

$$\frac{C}{C^*} \leq \rho$$

In this section, we first give the algorithm for computing β for a DAG. Then we give approximation algorithms based on β, for the four optimization problems, max clique, max independent set, min coloring and min clique cover.

4.1. Computing β

Given a DAG, $D = (V, E)$, we give an $O((m + n)n)$ algorithm for computing β for it. For a DAG with weighted edges, all pair shortest paths, i.e. the shortest paths between all the vertices can be calculated in $O((m + n)n)$ time, by just topologically sorting the DAG and running single source shortest path from each vertex. So, first we make each edge of D a weight of -1 and run all pair shortest path. Then we calculate the longest path between any two non-adjacent vertices of D and take the maximum of the longest path, which takes

$O(n^2)$ making a total of $O(n^2+m(n+m))$ which is the same as $O(nm)$ since D is a connected digraph.

Algorithm 1: *calcBeta*(D)

```
Data: $D = (V, E)$
Result: $\beta$
begin
    Set all edge weights of $G$ to -1
    $M = APSP(D)$ // Calculates all-pair-shortest-paths and stores them in M
    $\beta = 2$
    for all $(u, v) \notin E$ and $v$ is reachable from $u$ do
        $d = M[u][v]$ // Shortest Distance from u to v
        $\beta = max(\beta, |d| + 1)$
        /* absolute value of $d$ because the edges weigh -1. We add 1 to $|d|$ because we
          want the number of vertices in the path, not the edges. */
    return $\beta$
```

4.2. Approximation to max clique

For a β-transitive DAG, $D = (V, E)$, we can calculate β-approximation algorithm for the max clique problem which runs in $O(m + n)$ time.

Lemma 4.2.1. *For a DAG, $D = (V, E)$ the longest path can be calculated in $O(m + n)$ time.*

Proof. We first topologically sort the vertices of D, let $(v_1, v_2, \ldots, v_n)$ be a topological sort. Now, we first label every vertex with zero, i.e. $d(v_i) = 0 \ \forall i \in [1, n]$. Then for each i from 2 to n we calculate $d_i = max(1 + d_j, d_i) \ \forall \{j | (v_j, v_i) \in E\}$ and mark a back-edge from d_i, to d_j if $d_i \neq 0$ and $1 + d_j > d_i$. After this is done for all $i \in [2, n]$, the maximum value of d_i for $i \in [1, n]$ gives the maximum length path, and this path can be found by following the back-edges. □

Theorem 4.2.2. *β-approximate max clique for a β-transitive DAG can be calculated in $O(m+n)$ time.*

Proof. Let a maximum clique in the DAG be of size C^*. Since every clique in a digraph has a Hamiltonian path (property 2.1.3), a longest path in the DAG has $L_{max}(\geq C^*)$ vertices. After calculating the longest path in the DAG in using Lemma 4.2.1, we can construct a clique by using property 3.0.4 of size $C = \lceil L_{max}/\beta \rceil$. Since, $L_{max} \geq C^*$, we have $C^*/C \leq \beta$. □

4.3. Approximation to max independent set

For the max independent set approximation algorithm for a β-transitive DAG, we first need to compute a minimum path cover for the DAG. By Lemma 2.1.2, a minimum path cover for a DAG can be calculated in $O(\sqrt{n}m)$ time and by Lemma 2.1.4 we can calculate an independent set of size at least equal to the size of the minimum path cover. Therefore, we have the following theorem.

Theorem 4.3.1. *[18] For a β-transitive DAG, D, there is a $\lceil \beta/2 \rceil$-approximation algorithm for calculating the maximum independent set in $O(\sqrt{n}m)$ time.*

Proof. For a DAG, as shown in previous section, an independent set of size at least equal to the min path cover can be found in $O(\sqrt{n}m)$ time. Since by property 3.0.5 the max independent set on a path is of size $\lceil \beta/2 \rceil$, the cardinality of max independent set for D is at most $p\lceil \beta/2 \rceil$ in size, if p is the path cover number. As we have found a maximum independent set of cardinality p, we have a $\lceil \beta/2 \rceil$-approximation algorithm for max independent set. □

4.4. Approximation to min coloring

Theorem 4.4.1. *For a β-transitive DAG, D, we have a $\lceil \beta/2 \rceil$-approximation algorithm for min coloring which runs in $O(n+m)$ time.*

Proof. We use the labeling explained in property 2.1.2, and color all the vertices with the same label with the one color. Thus there will be L_{max} number of color classes, where $L_{max} - 1$ is the maximum length of a path in the DAG. From property 3.0.9, we know that a path with L_{max} vertices can be colored with at least $\lceil L_{max}/\lceil \beta/2 \rceil \rceil$ colors. So the least number of colors needed to color the DAG, D, is $\lceil L_{max}/\lceil \beta/2 \rceil \rceil$. Since our algorithm uses L_{max} colors we have a $\lceil \beta/2 \rceil$-approximate algorithm for min coloring. □

4.5. Approximation to min clique cover

Theorem 4.5.1. *For a β-transitive DAG, D, we have a β-approximate algorithm for min clique cover which runs in $O(\sqrt{n}m)$ time.*

Proof. We first find a minimum path cover of D in $O(\sqrt{n}m)$ time. Using property 3.0.8, we can divide a path into β cliques. Thus if we have the size of minimum path cover as p, we have $p\beta$ cliques. Since we know that the minimum clique cover is at least equal to the minimum path cover which is of size p, we have a β-approximate algorithm. □

CHAPTER 5

The Four Optimization Problems On DAGs with a Fixed β

For three of the four optimization problems we consider, namely, max independent set, min coloring and min clique cover, we prove that the corresponding decision problems are NP-Complete for a β-transitive DAG with a fixed β. For the remaining problem of finding a max-clique in β-transitive DAG we give a polynomial time algorithm by invoking a result in [4].

Richard Karp in his seminal paper [19] proved that the corresponding decision problems for the four optimization problems are NP-Complete on the class of undirected graphs. Since every undirected graph can be made into a DAG, the problems remain NP-Complete even on the class of DAGs.

By our definition of β-transitive DAGs, we know that the $\beta = 2$ for transitive DAGs, and there are polynomial time algorithms for all the four optimization problems for transitive DAGs as outlined before. We define the decision problems for the four optimization problems as follows:

(1) MAX CLIQUE: Given a directed graph, G and an integer k, decide if there exists a clique of at size at least k.
(2) MAX INDEPENDENT SET: Given a directed graph, D, and a an integer k, decide if there exists an independent set of size at least k.
(3) MIN COLORING: Given a directed graph, D, and an integer k, decide if the graph can be colored with at most k colors.

(4) MIN CLIQUE COVER: Given a directed graph, D, and an integer k, decide if the graph can be partitioned into at most k sets such that each set is a clique.

5.1. MAX CLIQUE

Using the algorithm outlined using λ introduced in the paper by Hamburger et al. [4], a maximum clique can be found in $O(\lambda m^{\lfloor \lambda+1 \rfloor/2})$. Since in property 3.0.2 we have proved that β at most $\lambda + 1$, the algorithm runs in $O(\beta m^{\beta/2})$ time.

5.2. MAX INDEPENDENT SET

We prove that the decision problem, MAX INDEPENDENT SET is NP-Complete on the set of β-transitive DAGs with a constant, β, such that $\beta \geq 3$.

A tripartite graph, $G = (V = A \cup B \cup C, E)$ is an undirected graph such that the vertices can be partition into three independent sets, A, B and C. The decision problem MAX INDEPENDENT SET on tripartite graph is NP-Complete [10].

Theorem 5.2.1. *MAX INDEPENDENT SET is NP-Complete on β-transitive DAGs with a given β, where $\beta \geq 3$. [20]*

Proof. By theorem 2.1.6, the decision problem, MAX INDEPENDENT SET, is answerable in polynomial time on tripartite graphs which are comparability, however, the edges in tripartite graphs which are not comparability can be oriented such that all edges go from A to B, or A to C or B to C, since the underlying undirected graph is not comparability, $\beta = 3$ for the resulting DAG. Thus we can reduce the problem MAX INDEPENDENT set on tripartite graphs to MAX INDEPENDENT SET on β-transitive DAGs with $\beta = 3$. □

5.3. MIN COLORING

We prove the decision problem, MIN COLORING is NP-Complete on the class of β-transitive DAGs with a constant β, such that $\beta \geq 4$. It remains open for the class of DAGs with $\beta = 3$. To prove this, we use the decision problem, Monotone-Not-All-Equal-3SAT (MONOTONE NAE-3SAT), which is be defined as the following:

MONOTONE NAE-3SAT: Given a set of boolean variables, $U = \{x_1, x_2, \ldots, x_n\}$ and a set of clauses, C, over U such that for each clause $c \in C$, $|c| = 3$, with all the variables in c occurring without negation, i.e. monotonically, does there exist a satisfying truth assignment such that not all the variables in any clause are true.

The NP-Completeness of MONOTONE NAE-3SAT follows from Schafer's paper [21]. We reduce MONOTONE NAE-3SAT to MIN COLORING on β-transitive DAGs with a given β.

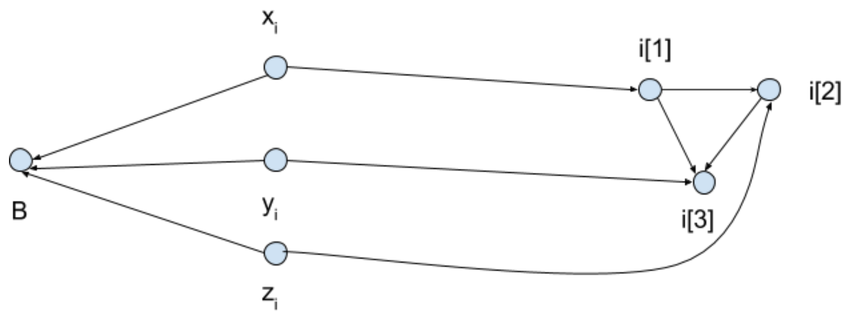

FIGURE 5.1. GADGET FOR MONOTONE NAE-3SAT CLAUSE, $c_i = \{x_i, y_i, z_i\}$

THEOREM 5.3.1. *MONOTONE NAE-3SAT $\leq_P$ MIN COLORING on β-transitive DAGs with constant β, such that $\beta \geq 4$.*

PROOF. Given the expression $E = c_1, c_2, \ldots, c_K$, a clause $c_i = \{x_i, y_i, z_i\}$ can be replaced a gadget shown in Figure 5.1. All the variables are connected to the vertex B. For each variable there is a vertex and for each clause there is a three clique, connected to the corresponding vertices in the clause. Clearly, the corresponding graph for E is a β-transitive DAG with $\beta = 4$ and the number of nodes are $1 + 3|C| + |U|$, where U is the set of variables.

LEMMA 5.3.2. *The β-transitive DAG with $\beta = 4$ corresponding to the monotone NAE-3SAT is 3 colorable if and only if there exists a truth assignment to the monotone NAE-3SAT.*

PROOF. If the β-transitive DAG is three colorable, then let the three colors be 0, 1 and 2. Without loss of generality, let node B be colored with color 2, then all the variables are either colored with 0 or 1, and due to the three clique no two nodes which represent the variables in a clause have the same color. Thus, if color 0 represents false and 1 represents true, we have a truth assignment for the MONOTONE NAE-3SAT.

Conversely, if the MONOTONE NAE-3SAT has a truth assignment, then if we color the node B with color 2 and color the nodes corresponding to the variables with 0 if the variable is false and 1 if variable is true, then there is a color scheme for the three clique because not all variables in a clause can be equal. □

Thus, MONOTONE NAE-3SAT can be reduced to the problem of finding 3 colorability of a β-transitive DAG with $\beta = 4$. □

5.4. MIN CLIQUE COVER

We first prove that MIN CLIQUE COVER is NP-Complete on tripartite graphs by reducing 3D-MATCHING to it.

3D-MATCHING [10]: Given a set $M \subseteq X \times Y \times Z$, where, $|X| = |Y| = |Z| = q$ and X, Y,

Z are disjoint. Does M contain a matching M', i.e., $M' \subseteq M$ such that $|M'| = q$ and no two elements of M' have any coordinate same?

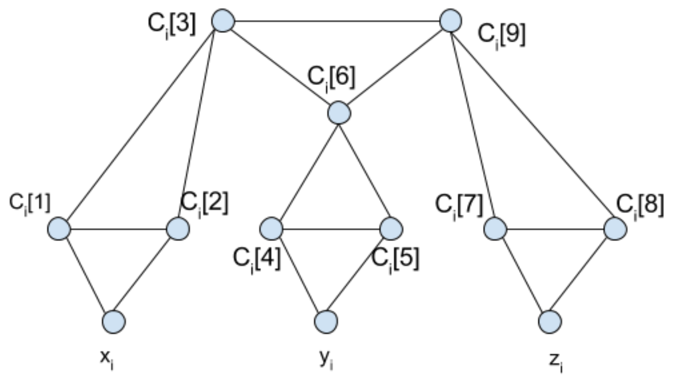

FIGURE 5.2. GADGET CORRESPONDING TO THE MATCHING CLAUSE, $c_i = (x_i, y_i, z_i)$.

THEOREM 5.4.1. *MIN CLIQUE COVER is NP-Complete on tripartite graphs.*

PROOF. Consider an instance of 3D-MATCHING, for a clause $c_i = (x_i, y_i, z_i)$, we use the gadget proposed by Schafer [10] as indicated in Figure 5.2. Now, the graph, G, constructed using this gadget has a total of $3q + 9|C|$ vertices, if $q = |X| = |Y| = |Z|$ and C is the set of clauses. It's easy to see that this graph is tripartite, and then we have the following lemma.

LEMMA 5.4.2. *G has a clique cover of size $q+3|C|$ if and only if 3D-MATCHING returns true.*

PROOF. It's not difficult to see that a 3D matching of size q divides G into $q + 3|C|$ triangles, and thus gives a clique cover of size $q + 3|C|$. Clearly the greatest size clique in G is of size three, and a clique cover of size $q + 3|C|$ is smallest possible clique cover.

Conversely, if there is a clique cover of size $q + 3|C|$, then there are q triangles in the clique cover with the vertices of the form $\{C_i[3],\ C_i[6],\ C_i[9]\}$ for different i's. These q

disjoint triangles correspond to a subset of $X \times Y \times Z$, which gives a matching for 3D-MATCHING. $\square$

$\square$

THEOREM 5.4.3. *MIN CLIQUE COVER is NP-Complete on β-transitive graphs with a given $\beta \geq 3$.*

PROOF. We proved that MIN CLIQUE COVER is NP-Complete on tripartite graphs, we reduce it to MIN CLIQUE COVER on β-transitive DAGs with a constant β. So given a tripartite graph, we can find its min clique cover in polynomial time if it's a comparability graph (theorem 2.1.8). If the tripartite graph, $G = (V = A \cup B \cup C, E)$ is not comparability then it can be converted to a β-transitive DAG with $\beta = 3$ by orienting the edges such that all edges go from A to B or B to C, or A to C. $\square$

CHAPTER 6

Quasi-transitivity and Orienting for Minimum β

In this chapter we prove a more general version of a theorem proposed by Ghouila-Houri [5]. A directed graph is called *quasi-transitive* digraph, if for any pair of edges (x, y) and (y, z) there is at least one edge between x and z [22]. A *quasi-transitive orientation* is an orientation of an undirected graph such that the resulting digraph is *quasi-transitive.* A *transitive orientation* of an undirected graph is defined similarly. Ghouila-Houri [5] proved the following:

Theorem 6.0.1. *A graph G has a quasi-transitive orientation iff it has a transitive orientation.*

We propose more general definitions using β:

Definition *β-transitive orientation*: An *acyclic* orientation of an undirected graph such that resulting DAG is β-transitive.

Definition *Quasi β-transitive digraph*: A *digraph* (not necessarily DAG) which doesn't have a path of length greater than $\beta - 1$ whose end points are not adjacent.

Definition *Quasi β-transitive orientation*: An orientation of an undirected graph such that resulting digraph is quasi β-transitive.

Now we prove a more general version of theorem 6.0.1, using the above definitions.

Theorem 6.0.2. *An undirected graph has a quasi-β transitive orientation, iff it has a β-transitive orientation.*

Proof. Clearly, every β-transitive orientation is a quasi β-transitive orientation. Now, consider a quasi β-transitive digraph, G. To prove that every β-quasi transitive orientation can be made β transitive, we use the depth-first tree of G. Depth first search on an undirected graph partitions the edges of the graph into two sets, back-edges and trees edges. We claim that if the orientation of every back edge of the depth-first tree is flipped, we will get a β transitive orientation. Let the new graph after flipping the back edges be $\tilde{G}$. We call, the flipped edges of G in $\tilde{G}$ as *flipped back edges* and the tree edges in G which were kept unchanged in $\tilde{G}$ as *unflipped edges*.

If $\tilde{G}$ is not a β-transitive digraph, it has a *contradicting path*, a path of length greater than $\beta - 1$ whose ends are not adjacent.

Lemma 6.0.3. *$\tilde{G}$ cannot have a contradicting path with only unflipped edges.*

Proof. If it does, than the path was contradicting in G has well, so G wasn't quasi β transitive, contradiction. □

Lemma 6.0.4. *$\tilde{G}$ cannot have a contradicting path with flipped back edges.*

Proof. Let the longest contradicting path, P, be $(v_1, v_2, \ldots, v_p)$, $(p \geq 3)$. Let (v_i, v_j), such that $(1 \leq i < j \leq p)$ be a flipped back edge. Then the flipped back edge can be replaced by a path with only tree edges. Now, we do this for all the back edges in the graph. Say the new path, P', is $(u_1, u_2, \ldots, u_q)$ $(q > p)$. Now, P' has all distinct vertices, because there are no cycles in $\tilde{G}$. Since the length of the simple path $\tilde{P}$ is greater than P, P wasn't the longest path, contradiction. □

Since a contradicting path in $\tilde{G}$ can have only flipped back edges or unflipped edges, there can't be a contradicting path in $\tilde{G}$. □

6.1. Orienting for minimum β

Given an undirected graph $G = (V, E)$, we study the problem of orienting the edges of the graph to find the minimum β. We prove that the problem is NP-Complete. We first define the corresponding decision problem.

β-ORIENT: Given an undirected graph and a k, does there exist an acyclic orientation of the edges such that the DAG has a β at most k?

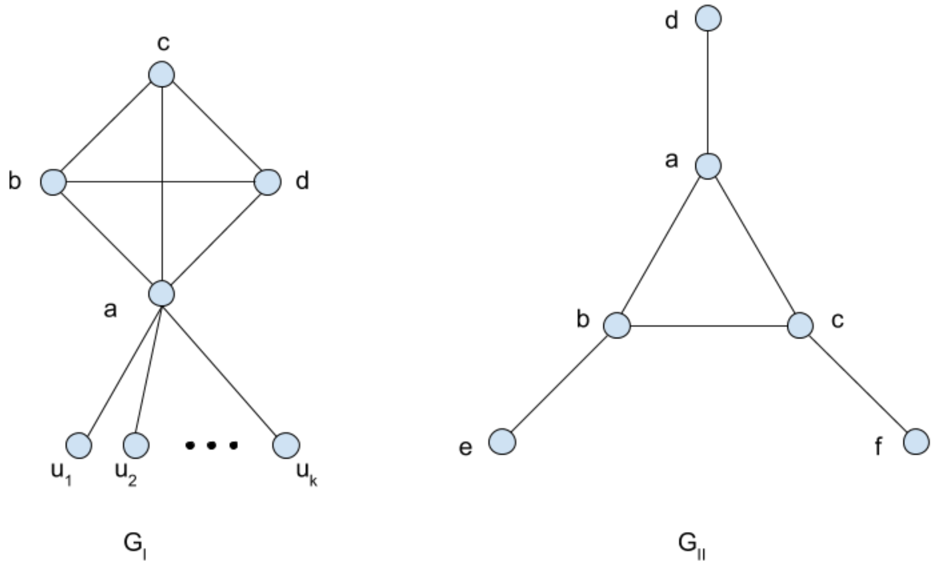

Figure 6.1. Gadgets for proving β-ORIENT is NP-Complete

Theorem 6.1.1. *β-ORIENT is NP-Complete [20].*

Proof. We reduce MONOTONE NAE-3SAT to it. First we consider two lemmas.

Lemma 6.1.2. *G_I in figure 6.1 can have $\beta = 3$ iff all the undirected edges $(a, u_1), (a, u_2), \ldots, (a, u_k)$ are oriented in one direction, i.e. either towards a or in the opposite direction.*

Proof. In the graph induced by $\{a, b, c, d\}$ in G_I, it's not tough to see that in any acyclic orientation, there is always a path of length 2 starting from or ending at a. Let, without

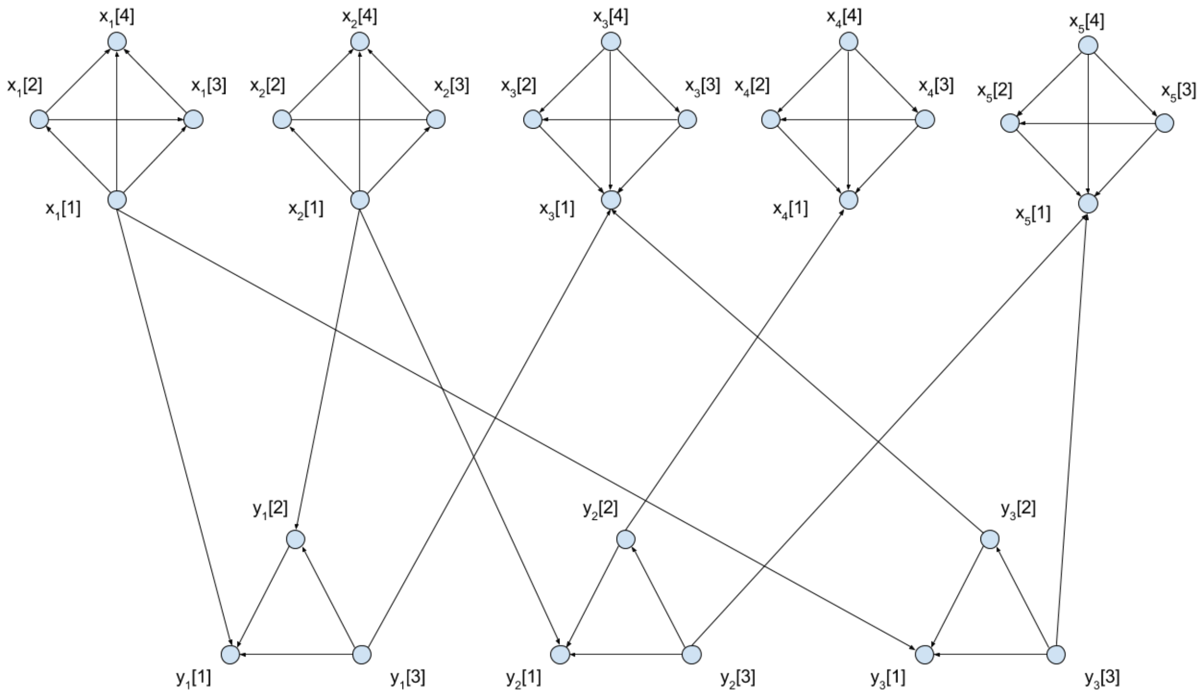

FIGURE 6.2. GRAPH FOR THE EXPRESSION $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5)$

loss of generality, (a, u_i) be oriented towards a and (a, u_j) oriented in the opposite direction, then there is always a path of length 3 either starting from u_i or ending at u_j, so they should be both oriented either towards or away from a.

Conversely, if all the $(a, u_1), (a, u_2), \ldots, (a, u_k)$ are oriented in one direction, it's easy to see that we can have a β-transitive orientation of G_I with $\beta = 3$. □

LEMMA 6.1.3. *In graph G_{II}, $\beta = 3$ iff not all of (b, e), (a, d) and (c, f) can be oriented towards or away from d, e, and f respectively.*

PROOF. For any acyclic orientation of the graph induced by $\{a, b, c\}$ in G_{II}, there is always a path of length 2 (property 2.1.3). Now, without loss of generality, let the path be from a to c. Then if the edge (a, d) is oriented towards a, we have $\beta = 4$ for G_{II} because of the path, (d, a, b, c). If (a, d) is oriented towards d, and (c, f) is oriented towards f, then $\beta = 4$ because of the path a, b, c, f.

Conversely, if not all the (b, e), (a, d) and (c, f) can be oriented towards or away from d, e, and f respectively, it's easy to see that we can have an β-transitive orientation of G_{II} with $\beta = 3$. $\square$

Given an instance of MONOTONE NAE-3SAT represented by the expression $E = c_1, c_2, \ldots, c_K$ over the set of variables $U = x_1, x_2, \ldots, x_n$, for each variable x_i we create a G_{II}, and for each clause we create a G_I. For example, for the expression $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5)$ we have the graph in figure 6.2. Now since the complement of a solution to NAE-3SAT is also a solution, we say x_i is true iff the edges joining node $x_i[1]$ with G_{II} are oriented towards $x_i[1]$ and false if oriented in the opposite direction, figure 6.2.

Essentially, gadget G_I with Lemma 6.1.2 forces the edges representing the variables occurring in all the clauses to be oriented in the same direction. Gadget G_{II} with Lemma 6.1.3 forces the not-all-equal part, i.e., makes sure not all the variables in a clause are equal. Thus, for every satisfying assignment of the NAE-3SAT expression, there exists an acyclic orientation of the edges such that β of the resulting DAG is 3. And for an orientation which has $\beta = 3$, there corresponds a satisfying assignment of the NAE-3SAT. $\square$

CHAPTER 7

Additional Work

Given a digraph, $G(V, E)$ and a weight function, $f : E \to \mathbf{R}$, let the *weight* of any "edge progression" (walk) $\sigma = (e_1, e_2, \ldots, e_p)$ be defined by $w(\sigma) = \sum_{i=1}^{p} f(e_i)$. Let $n = |V|$ and $m = |E|$. Let the *length* of the edge progression be p. Karp [23] gave a characterization of the minimum cycle mean, λ^*, over all the directed cycles in G.

A *strongly-connected component* in a digraph is a subgraph such that any two vertices u, v in the subgraph have a path from u to v and from v to u. Strongly connected components of a digraph can be found in linear time [1].

Since any directed cycle is confined to a single strongly-connected component, Karp's algorithm can be applied separately to the subgraph induced by each component and returning the minimum cycle mean over all the components. Henceforth, we assume that the graph is strongly connected.

Let s be an arbitrary *start vertex* of G. For each vertex $v \in V$, let $F_k(v)$ be the minimum weight of any edge progression of length exactly k from s to v, of ∞ if no such edge progression exists. Karp proves the following:

Theorem 7.0.1.

$$\lambda^* = \min_{v \in V} \max_{0 \le k \le n-1} \left[\frac{F_n(v) - F_k(v)}{n-k} \right]$$

$F_k(v)$ can be found in $O(nm)$ time for all $v \in G$ and all $k \in \{0, 1, \ldots, n\}$ using a dynamic programming algorithm that assigns $F_k(v)$ to an entry of the table indexed by (k, v). This gives an $O(nm)$ algorithm for finding the value of the minimum cycle mean, by Theorem 7.0.1.

For each $v \in V$ and $\{k|0 < k \leq n\}$, the dynamic programming algorithm can assign a backpointer from (k, v) to an entry $(k-1, w)$, giving the predecessor w on an edge progression of length k and weight $F_k(v)$ from s to v. This allows a minimum weight edge progression of length k from s to v to be reconstructed, by following backpointers.

7.1. Finding a cycle of minimum mean

Let a *minimizer* be a vertex v such that $max_{0 \leq k \leq n-1}(F_n(v) - F_k(v))/(n-k) = \lambda^*$, and let a *minimizing pair* be a minimizer v and integer k such that $0 \leq k \leq n-1$ and k maximizes $(F_n(v) - F_k(v))/(n-k)$. Karp suggests the following for finding a cycle of minimum mean weight.

> If the actual cycle yielding the minimum cycle mean is desired, it can be computed by selecting the minimizing [pair] v and k, finding a minimum-weight edge progression of length n from s to v, and extracting a cycle of length $n-k$ occurring within that edge progression.

He does not supply a proof that this procedure is correct, and Figure 7.1 gives a counterexample. In the figure, the minimum cycle mean is 1, which is the mean weight of the cycles (s, a, b) and (a, c, d, e, f). The value of $F_n(g) = F_8(g)$ is 9, given by the edge progression (walk) $(s, a, c, d, e, f, a, c, g)$, and the value of $F_6(g)$ is 7, given by (s, a, b, s, a, c, g). Since $[F_8(g) - F_6(g)]/(8-6) = 1$, which is the minimum cycle mean, g and 6 are a minimizing pair. There is supposed to be a cycle of length $8 - 6 = 2$ on the walk $(s, a, c, d, e, f, a, c, g)$, but there is no cycle of length 2 in the graph.

Karp's proof of Theorem 7.0.1 shows that for some minimizing pair v and k, and some walk W of weight $F_n(v)$ from s to v, the last k edges of W are a cycle of minimum mean

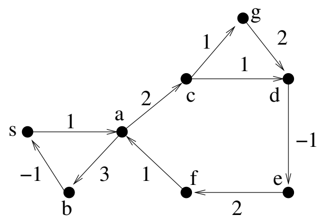

FIGURE 7.1. COUNTEREXAMPLE TO KARP'S ALGORITHM FOR CONSTRUCTING A MINIMUM MEAN WEIGHT CYCLE.

weight. In Figure 7.1, d and 3 are such a pair; the last $n-3=5$ edges of (s,a,c,d,e,f,a,c,d) are a cycle of minimum mean weight. For v, the proof uses a vertex that lies on a cycle of minimum mean weight, and shows that there exists a walk from s to v of length n and weight $F_n(v)$ such that the last k edges of the walk are a cycle of minimum mean weight. However, these conditions do not apply for all minimizing pairs. The example of g in the example of Figure 7.1 shows that a minimizer need not lie on a cycle of minimum cycle mean. Even for a vertex v such that the assumption applies, it is not true for every minimizing pair that v is a part of: in the figure, $(d,3)$ is a minimizing pair for which the assumption applies, but $(d,2)$, given by (s,a,c,d,e,f,a,c,d) and (s,a,b,s,a,c,d), is one where it is not.

One fix would be to apply his suggested algorithm to each minimizing pair, since the assumptions apply to at least one of them. However, even when the assumptions apply to a given minimizing pair (v,k), the existence of more than one cycle of minimum mean weight can mean that there is more than one minimum-weight edge progression of length n from s to v, it may be the case that not all of them satisfy the required conditions, and the dynamic programming algorithm might find one that does not. There is a way around this, but there are $\Theta(n^2)$ minimizing pairs in the worst case, and some care is needed to keep the time bound of this approach to $O(nm)$. The insights developed in the proof of Theorem 7.0.1 suggest better alternatives, and the following is particularly simple.

LEMMA 7.1.1. *Let v be a vertex such that there exists k, where v and k are a minimizing pair. Every cycle on the length n edge progression from s to v of weight $F_n(v)$ is a cycle of minimum mean weight.*

PROOF. Let W be a length n edge progression from s to v of weight $F_n(v)$. Subtracting λ^* from the weight of every edge of G reduces the mean weight of every cycle and edge progression by λ^* in the resulting graph G'. The cycles of minimum mean weight in G are those that have weight 0 in G', if v and k are a minimizing pair, they remain one in G', and W remains a minimum-weight edge progression of length n from s to v in G'. It suffices to show that every cycle on W has weight 0 in G'.

Suppose there's a cycle of positive weight on W. Omission of the cycles on W results in a path P from s to v of weight $w < F_n(v)$ in G'. If $k' = |P|$, then in G', $F_{k'}(v) < F_n(v)$, and $[F_n(v) - F_{k'}(v)]/(n - k') > 0$, which is the minimum cycle mean of G', contradicting that v must be a minimizer in G' by Theorem 7.0.1. □

If v is a minimizer, then following backpointers from (n, v) of the table gives a walk of weight $F_n(v)$ from s to v. Since it is longer than $n - 1$, it has a cycle, and by the lemma, every cycle on it is a cycle of minimum mean weight. By traversing backpointers marking vertices visited by the walk until a previously marked vertex w is encountered, a cycle of minimum mean weight can be identified in $O(n)$ time.

REFERENCES

[1] T. H. Cormen, *Introduction to algorithms.* MIT press, 2009.

[2] M. C. Golumbic, *Algorithmic graph theory and perfect graphs*, vol. 57. Elsevier, 2004.

[3] D. Zuckerman, "Linear degree extractors and the inapproximability of max clique and chromatic number," in *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pp. 681–690, ACM, 2006.

[4] P. Hamburger, R. M. McConnell, and J. P. Spinrad, "Double threshold digraphs,"

[5] A. Ghouilà-Houri, "Characterization of nonoriented graphs whose edges can be oriented in such a way as to obtain the graph of an order relation," *CR Acad. Sci. Paris*, vol. 254, pp. 1370–1371, 1962.

[6] M. C. Golumbic, "The complexity of comparability graph recognition and coloring," *Computing*, vol. 18, no. 3, pp. 199–208, 1977.

[7] P. Hell and J. Huang, "Lexicographic orientation and representation algorithms for comparability graphs, proper circular arc graphs, and proper interval graphs," *Journal of Graph Theory*, vol. 20, no. 3, pp. 361–374, 1995.

[8] M. Morvan and L. Viennot, "Parallel comparability graph recognition and modular decomposition," in *Annual Symposium on Theoretical Aspects of Computer Science*, pp. 169–180, Springer, 1996.

[9] R. M. McConnell and J. P. Spinrad, "Linear-time transitive orientation.," in *SODA*, vol. 97, pp. 19–25, 1997.

[10] M. R. Garey and D. S. Johnson, *Computers and intractability*, vol. 29. wh freeman New York, 2002.

[11] R. E. Tarjan, *Data structures and network algorithms.* SIAM, 1983.

[12] D. R. Fulkerson, "Note on dilworths decomposition theorem for partially ordered sets," in *Proc. Amer. Math. Soc*, vol. 7, pp. 701–702, 1956.

[13] J. E. Hopcroft and R. M. Karp, "An nˆ5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[14] L. Lovász, "A characterization of perfect graphs," *Journal of Combinatorial Theory, Series B*, vol. 13, no. 2, pp. 95–98, 1972.

[15] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas, "The strong perfect graph theorem," *Annals of mathematics*, pp. 51–229, 2006.

[16] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.

[17] J. P. Spinrad, *Efficient graph representations.* American mathematical society, 2003.

[18] S. Manchanda, *On approximating transitivity and tractability of graphs.* PhD thesis, Colorado State University. Libraries, 2016.

[19] R. M. Karp, *Reducibility among combinatorial problems.* Springer, 1972.

[20] Z. Xu. Private Communication.

[21] T. J. Schaefer, "The complexity of satisfiability problems," in *Proceedings of the tenth annual ACM symposium on Theory of computing*, pp. 216–226, ACM, 1978.

[22] J. Bang-Jensen and J. Huang, "Quasi-transitive digraphs," *Journal of Graph Theory*, vol. 20, no. 2, pp. 141–161, 1995.

[23] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete mathematics*, vol. 23, no. 3, pp. 309–311, 1978.