Thesis

On Approximating Transitivity and Tractability of Graphs

Submitted by

Saksham Manchanda

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2016

Abstract

On Approximating Transitivity and Tractability of Graphs

In the general case, in a simple, undirected graph, the problems of finding the largest clique, minimum colouring, maximum independent set and minimum vertex cover are NP-hard. But, there exists some families of graphs, called perfect graphs, where these problems become tractable. One particular class of perfect graphs are the the underlying undirected graphs of transitive digraphs- called comparability graphs. We define a new parameter β to approximate the intransitivity of a given graph. We also use β to give a measure of complexity of finding the largest clique. As β gets worse, the complexity of finding the largest clique quickly grows to exponential times. We also give approximation algorithms that scale with β for all our NP-hard problems. The β measure of a graph can be computed in $O(mn)$, therefore, β can be considered a measure of how tractable these problems are in a graph.

Table of Contents

List of Figures

CHAPTER 1

Introduction

Graph theory is said to have its origins with Leonhard Euler in the early 1700s with the "The Seven Bridges of Konigsberg" problem. Since then, graph theory has developed into a popular and highly useful branch of mathematics and computer science. Many seemingly unrelated problems that come up in a variety of fields can be reduced to graphs and solved using graph algorithms.

However, many problems of practical importance like k-colour were shown to be NP-Complete, which is widely believed to be the same as proving computational intractability. Many of these problems become polynomial-time solvable when restricted to certain graph classes. Interval graphs and comparability graphs are two such graph classes and have a wide variety of applications in the design of fast algorithms for problems that frequently come up in many real-world applications like analysis of genetic structure, synchronization problems, certain scheduling problems.

In 1960, Claude Berge defined the notion of the perfect graph and posited that there was a deep underlying reason for the tractability of optimization problems on perfect graphs, and not mere coincidence. These conjectures were later shown to be true and became theorems. Since then many more graph classes have been shown to be perfect and tractable.

However, when a graph vary even slightly from perfection, it immediately become untractable. A graph can be put into one of two broad buckets of perfect and non-perfect, making optimization problems polynomial time solvable or NP-Complete respectively. This behaviour is clearly undesirable since the graph may still have a certain exploitable structure to help compute the optimization problems. The main thrust of this work is to address this sharp drop in tractability.

We choose comparability graphs (transitive DAGs) and incrementally relax the transitivity constraint in a way such that the DAG falls in one of an infinite hierarchy of DAGs ranging from transitive DAGs to an arbitrary DAG. We define a new parameter β to approximate the transitivity of a DAG and show that β is closely related to the complexity of solving the maximum clique, minimum colouring, maximum independent set and minimum clique cover problems.

1.1. Preliminaries

A graph $G = (V, E)$ is an ordered pair, where $V(G)$ is a non-empty finite set of *vertices* and $E(G)$ is a (possibly empty) set of *directed edges*, which are ordered pairs of vertices. An edge $e \in E(G)$ where $e = \{(u, v)|u, v \in V(G)\}$. The ordered pair (u, v) is a *directed edge.* When this edge exists, we say that v is adjacent to u. The directed edge (u, v) is said to be *outgoing edge* on u and *incoming edge* on v. Also, the shorthand V and E are used $V(G)$ and $E(G)$ respectively, when G is understood. Both E and V are finite sets. We let n denote $|V|$, and m denote $|E|$.

When (u, v) and (v, u) are both edges, we let $uv = \{(u, v), (v, u)\}$ denote an *undirected edge.* If, whenever (u, v) is an edge, (v, u) is an edge, the graph is an *undirected graph.*

For a graph G, all vertices adjacent to a vertex x in G is denoted by $N(x)$, called the *open neighbourhood* of x. The vertices adjacent to x are called the *neighbours* of x. The vertex x is not included in this set. We can also define $N[x]$, called the *closed neighbourhood* of x, by including x along with all its neighbours.

An undirected graph G is *complete* if every pair of distinct vertices shares an edge. The *underlying undirected graph* of a directed graph G is obtained by adding an edge (v, u) for every directed edge (u, v), if it is not already present.

A *walk* is a sequence $((v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k))$ of edges, and we can denote it $(v_1, v_2, \ldots, v_k)$. A *path* is a walk in which no vertex appears twice, and a *cycle* is a path $(v_1, v_2, \ldots, v_k)$ followed by an edge from v_k to v_1. The *length* of a cycle or walk is the number of edges in it, $k - 1$ for a path, and k for a cycle. In an undirected graph, a cycle is assumed to have at least three vertices, to exclude the cycle formed by the directed edges of an undirected edge.

A *directed acyclic graph (DAG)* is a directed graph that has no cycles. There is a way to order the vertices of the DAG such that for every direct edge (u, v), vertex u comes before vertex v in the ordering. Such an ordering is called a *topological ordering* of the DAG. In general, this is not a unique ordering. We can find a topological ordering in $O(|V| + |E|)$ time [1].

The *complement of a graph* $G = (V, E)$ is a graph $\overline{G} = (V, \overline{E})$, where

$\overline{E} = \{(u, v) \in V \ X \ V | u \neq v \ and \ (u, v) \ \notin \ E\}$

An *induced subgraph* H, of a graph G is a graph obtained by using a subset S of the vertices of $V(G)$ and only choosing the edges from $E(G)$ both of whose endpoints appear in S.

A *hereditary property* of a graph G is a property which also holds for all its induced subgraphs.

DEFINITION 1.1. *A clique of an undirected graph $G = (V, E)$ is a set of vertices that induces a complete subgraph. The clique number of G denoted $\omega(G)$, is the size of a clique of maximum size.*

DEFINITION 1.2. *The chromatic number of a graph G is the smallest number of colours needed to assign a colour to each vertex, such that no two adjacent vertices share the same colour. The smallest possible such number is denoted by $\chi(G)$.*

For any graph G, $\omega(G) \leq \chi(G)$. This is because no two vertices of a clique can be assigned the same colour. The equality is not always strict: For example, if G is an odd cycle of size greater than three has a $\omega(G) = 2$ and $\chi(G) = 3$.

DEFINITION 1.3. *An independent set of a graph G is a subset of the vertices such that no two of them are adjacent. The size of a maximum independent set of G is called the stability number of the graph G and is denoted by $\alpha(G)$.*

DEFINITION 1.4. *The clique cover number of G, denoted $k(G)$, is the fewest number of cliques the vertices of the graph G can be partitioned into. The clique cover number of a graph is denoted by $k(G)$.*

Note that $\omega(G) = \alpha(\overline{G})$ and $\chi(G) = k(\overline{G})$, since an independent set in G is a complete graph in $\overline{G}$ and vice versa. The intersection of a maximum independent set with a clique has at most one vertex. Therefore, $\alpha(G) \leq k(G)$. Similarly, $\omega(G) \leq \chi(G)$.

By a clique, independent set, colouring or clique cover of a directed graph G, we mean a clique, independent set, colouring or clique cover respectively of G's underlying undirected graph.

1.2. NP-hard graph problems

A problem can be solved in *polynomial time* if there exists an $O(n^k)$ algorithm for it, for some fixed k. Algorithms that do not run in polynomial time are impractical for all but the smallest instances of a problem. While many problems in the fields of chemistry, social sciences, computer science can be solved by graph-theoretic algorithms, there are some problems for which no polynomial-time algorithm is known.

A *decision problem* is one where the solution is either "yes" or "no." The class P of decision problems are those that can be solved in polynomial time. The class NP of decision problems are ones where a "yes" answer can be verified in polynomial time, possibly given some supplementary information, known as a *certificate*, that may be difficult to find in polynomial time. Supplementary information may not be necessary, so $P \subseteq NP$. An example of a problem that is in NP but is not known to be in P is *INDEPENDENT-SET*, which is the problem of determining whether a given graph G has an independent set of a given size k. It is not known how to find such an independent set in polynomial time, but when the answer is "yes," a list of the vertices of the independent set of size k serves as a certificate. No certificate is known for a "no" answer.

It remains unproven but is widely believed that P is a proper subset of NP. A decision problem is *NP-complete* if every problem in NP reduces to it in polynomial time. Thus, a polynomial-time algorithm for an NP-complete problem would mean that $P = NP$, which is evidence that no polynomial-time algorithm exists for it, though the possibility that $P = NP$ has not been ruled out. INDEPENDENT-SET is an example of an NP-complete problem.

An *optimization problem* is the problem of finding a solution that maximizes (or minimizes) an objective function from a set of all possible solutions. A problem is *NP-hard* if a polynomial-time solution would imply that $P = NP$. An NP-hard problem need not be NP-complete, since it is not required to be a decision problem. An example of an NP-hard problem is the optimization problem of finding a maximum independent set in a given graph; if a polynomial algorithm existed for it, it could be used to solve INDEPENDENT-SET, which would imply $P = NP$. Problems in P are called tractable or easy problems, and NP-hard problems are considered *intractable*.

An *approximation algorithm* for an optimization problem is one that finds a solution such that the objective function is within a certain ratio of the optimum. Many intractable problems have polynomial-time approximation algorithms. For instance, finding a minimum-length tour of a set of points in a plane is NP-hard, but it is possible to get within a factor of 1.5 of the optimum length in polynomial time.

It is NP-hard to compute $\alpha(G)$, $\omega(G)$, $\chi(G)$, or $k(G)$ for an arbitrary graph G. These problems were among 21 problems shown to be NP-hard by Richard Karp in his famous 1972 paper: "Reducibility Among Combinatorial Problems"[2]. Since then, thousands of optimization problems that have great industrial interest have been shown to be NP-hard.

In addition, these problems are not known to have good approximation bounds[3]. The best known approximation bound for $\omega(G)$ is $O(n(loglog\ n)^2/log^3 n)$[4], $\alpha(G)$ is $(\Delta + 2)/3$ where Δ is the maximum number of edges incident to any vertex[5] and $\chi(G)$ is $O(n(logn)^3(loglog\ n)^2)$[6].

In this work, we study subclasses of graphs on which theses problems are tractable, and ones for which the problems have good approximation bounds.

CHAPTER 2

Graphs Classes

2.1. Interval Graphs

An interval graph G is obtained to model intersections among a set of of intervals. This is done by creating a node for each interval and creating an edge between two nodes if and only if their corresponding intervals intersect.

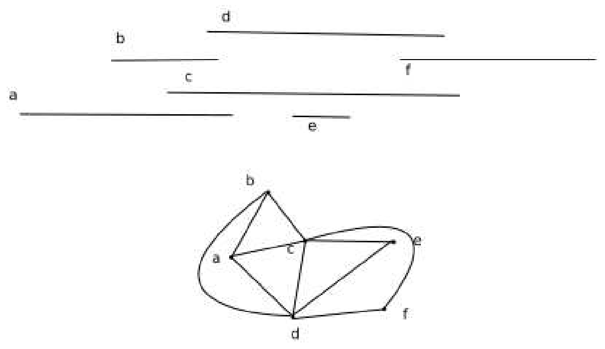

Figure 2.1. An interval graph and its corresponding interval model

Interval graphs are a hereditary class of graphs, which is seen as follows. Let G be an interval graph and G' be an induced subgraph of G. Since G is an interval graph, there exists an interval model for it; removing intervals corresponding to vertices that are not in G' gives an interval model of G', so G' is also an interval graph.

A *chord* on a cycle C is an edge not on C but whose endpoints are both on C. A *chordless cycle* is a cycle of length greater than three that has no chord. In other words, it is an induced cycle.

theorem 2.1. *An interval graph cannot have a chordless cycle.*

Proof. Let G be an interval graph, and suppose it has a chordless cycle $C = (v_1, v_2, \ldots, v_k)$. Let R be an interval model of G, and suppose without loss of generality that v_1 is the vertex

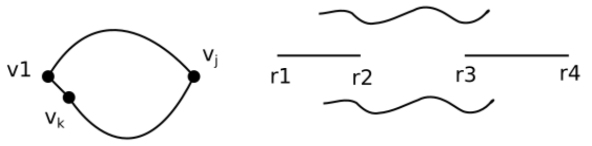

FIGURE 2.2. Interval graphs cannot contain $C_k \; for \; k > 3$

of C representing interval $[r_1, r_2]$, whose right endpoint r_2 is leftmost in R. Since C has non-adjacent pairs, the vertex v_j with interval $[r_3, r_4]$ whose left endpoint is rightmost. Since C is not a clique, $r_2 < r_3$. The path $(v_1, v_2, \ldots, v_j)$ corresponds to a set of intervals whose union contains $[r_2, r_3]$. Similarly for the path $(v_1, v_k, v_{k-1}, \ldots, v_j)$. Let x be a point in (r_2, r_3). An interval from the first path intersects an interval from the second path at x, implying that C has a chord, a contradiction.

Now, by Lemma 2.1, any induced subgraph of an interval graph is also an interval graph. Therefore, if we select vertices which induce a cycle of length greater than 3, that graph cannot be an interval graph. □

2.2. Chordal Graphs

DEFINITION 2.1. *A graph G is* chordal *if and only if every induced cycle of length greater than 3 has a chord edge.*

Every induced subgraph of a chordal graph is also chordal, since a graph on a subset of the vertices of G cannot have a chord. Therefore, graph chordality is a hereditary property.

By Theorem 2.1, all interval graphs are chordal. However, the converse is not true, i.e., not all chordal graphs are interval graphs. This can be seen from figure 2.3. Suppose, without loss of generality, interval a is placed to the left of interval e since they do not

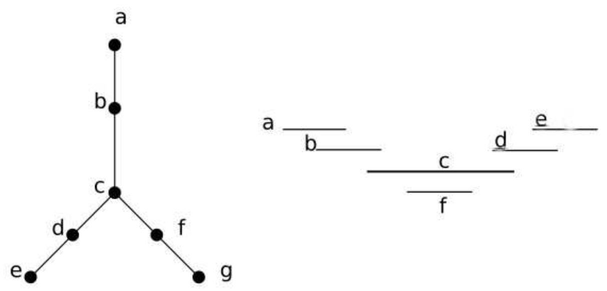

FIGURE 2.3. This tree is not an interval graph despite being chordal

intersect. Now, there is only one way to place the intervals b, c and d obeying the interval graph. Now, interval f must be placed such that it intersects c. There is no way for interval g to be placed such that it intersects f but not c. The graph is chordal, but it is not an interval graph.

DEFINITION 2.2. *A Simplicial Vertex of a graph G is a vertex v, such that $N[v]$ induces a clique.*

DEFINITION 2.3. *A perfect elimination ordering of a graph G is an ordering of its vertices $\sigma = (v_1, v_2, ..., v_n)$, such that the each v_i is simplicial in a subgraph induced by $v_i, v_{i+1}, ..., v_n$.*

In the rest of this section we prove the following theorem:

THEOREM 2.2. *A graph G is chordal if and only if it has a perfect elimination ordering.*

LEMMA 2.1. *If a graph has a perfect elimination ordering, then it is chordal.*

Proof. Assume a graph G has a chordless cycle $C = (v_1, v_2, \ldots, v_k)$ and a perfect elimination ordering. Without loss of generality, v_1 is first in the perfect elimination ordering. Then v_2 and v_k are neighbours of v_1 to its right, so they are adjacent to each other. The edge between them is a chord of C, a contradiction. $\square$

To finish the proof of Theorem 2.2, it remains to show the converse of this, which is that if G is chordal, it has a perfect elimination ordering.

DEFINITION 2.4. *A (a, b)-separator S of a graph G is a subset of the vertices of G, such that removal of S separates a and b into two separate connected components.*

Clearly, such a separator always exists for any two non-adjacent vertices a, b. For example, set V - $\{a, b\}$ is such a separator. A minimal a-b separator S is a separator such that no subset of S is also an a-b separator.

LEMMA 2.2. *In a chordal graph, every $a - b$ separator is a clique.*

Proof. Let S be a minimal separator in a chordal graph, and let $x, y \in S$ be arbitrary distinct vertices of S. Since S is a minimum separator, both x and y must have an edge to a vertex in A and another vertex in B. Therefore, there exists a shortest path P_A x to y whose internal vertices are in A. Similarly, let P_B be a shortest path whose internal vertices are in B. Together, A and B define a cycle of size at least 4. Let cd be a chord, which must exist, since G is chordal. It cannot be the case that $c \in A$ and $d \in B$, since S is a separator. It cannot be the case that one or both of c and d are internal to P_A or P_B, because they would result in shorter paths meeting their definitions. Thus, $cd = xy$, hence x and y are adjacent. Since x and y are arbitrary elements of S, every pair of elements of S is adjacent. □

LEMMA 2.3. *Any chordal graph G has a simplicial vertex. Moreover, if G is not a complete graph, it has two non-adjacent simplicial vertices.*

Proof. Because a subgraph induced from a chordal graph is also chordal, this result can be proved using recursion. Begin by showing the base cases.

If G is a clique, then any vertex is simplicial and we are done.

So assume G is not complete. Therefore, there must be two vertices, say a, b, that are non-adjacent.

Let S be the minimal (a, b)-separator. Now, by definition, graph G has at least two connected components: one containing a(denoted A), the other containing b(denoted B).

It is sufficient to show that both A and B contains a simplicial vertex to prove the theorem. Because A and B are symmetric, we can focus on only one case and the other is proved by symmetry. For A, we have two cases:

1) If A∪S induced a complete graph, a is clearly simplicial.

2) If A∪S is not complete, we recurse on A∪S. This is a strictly smaller vertex set than the original because b cannot be in it, so the recursion ends. Let this yield two non-adjacent simplicial vertices (x, y)

Now, if either of x(or y) is in A, then the neighbourhood of x(or y) is in the graph induced by $A \cup S$, therefore x is also simplicial in G.

By Lemma 2.2, S has to be a clique. Therefore, both x and y cannot be in S and we have found a simplicial vertex. □

LEMMA 2.4. *If a graph is chordal, it has a perfect elimination ordering.*

Let G be a chordal graph. Then by Lemma 2.3, we can select a simplicial vertex v_i which goes first in a possible perfect elimination ordering. Since the graph obtained by deleting vertex v_i is also chordal, we invoke lemma 2.3 again, to keep finding simplicial vertices which results in a perfect elimination ordering.

Theorem 2.2 now follows from Lemma 2.1 and Lemma 2.4

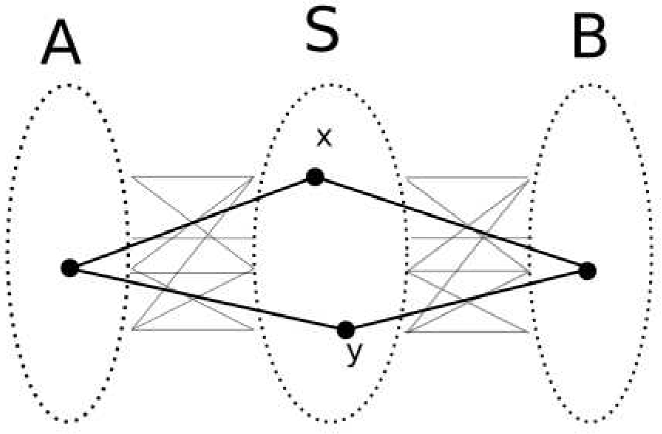

FIGURE 2.4. Chordal graphs have simplicial vertices

2.3. Algorithms on chordal graphs

In 1976, Rose, Lueker and Tarjan[7] gave a linear-time algorithm to find the perfect elimination ordering of a graph G if it is chordal. However, if G is not chordal, the algorithm returns an arbitrary ordering of the vertices. Given an alleged perfect elimination ordering σ for G, it remains to check if it is indeed a perfect elimination ordering.

Given an ordering of the vertices σ, let $N_\sigma(v)$ be the set of all vertices adjacent to v that are to the right of v in σ Also, let $N_\sigma[v]$ be the set obtained by adding v to $N_\sigma(v)$.

To test if σ is a perfect elimination ordering, consider each vertex in the presented order: $\sigma = (u_1, u_2, ...u_n)$. For each vertex u_i, verify that $N_\sigma[u_i]$ is a clique in $O(n^2)$ time. This takes $O(n^3)$ time overall.

To get a linear bound, process each vertex in left to right order of σ. When u_i is reached, let u_j be the earliest vertex in $N_\sigma(u_i)$. Check only that $N_\sigma(u_i) \subseteq N_\sigma[u_j]$. If this is not the case, there must exist some vertex u_k in $N_\sigma(u_i)$, but not in $N_\sigma[u_j]$. So, $N_\sigma[u_i]$ is not a clique and the algorithm returns "false". The algorithm returns "true" if this condition holds for all u_i in σ, which is seen as follows.

Suppose σ is not a perfect elimination ordering and the algorithm returns “true”. Without loss of generality, let u_i be the rightmost vertex in σ for which $N_\sigma[u_i]$ is not complete. Now, let u_j be the first vertex in $N_\sigma(u_i)$. Since, the algorithm did not return “false” at this iteration, u_j has an edge to all vertices in $N_\sigma[u_i]$. Also, since u_i is the rightmost vertex in σ for which $N_\sigma[u]$ is not complete, $N_\sigma[u_j]$ must be complete. These two condition force $N_\sigma[u_i]$ to be complete. This contradicts the assumption that $N_\sigma[u_i]$ is not complete. Therefore σ must be a perfect elimination ordering.

This algorithm looks at all the edges incident on each vertex v once and thus runs in $O(n + m)$ time.

Using these results we can solve our optimization problems on chordal graphs.

THEOREM 2.3. *For a chordal graph G, a minimum colouring and a maximum clique can be computed in linear time. Also, $\omega(G) = \chi(G)$.*

Proof. We use non-negative integers to colour the vertices. Given a perfect elimination ordering $\sigma = (u_1, u_2, \ldots\ u_n)$ for a graph G, process each vertex in the right-to-left order of σ. We colour each vertex u_i with the lowest colour not used in $N_\sigma(u_i)$.

This is a proper colouring because no vertex u_i gets the same colour as a neighbour to its right in σ. Suppose the algorithm uses k colours $(1, 2, \ldots k)$ to colour the graph G, and let u_i be the vertex that gets the colour k. The members of $N_\sigma(u_i)$ must be coloured with $1, 2, \ldots (k - 1)$. This gives us a colouring of size k. Since this is a proper colouring, any minimum colouring must use at most k colours. Therefore, $\chi(G) \leq k$ Also, $N_\sigma[u_i]$ is a clique of size k, so a maximum clique must have a size of at least k. Therefore, $\omega(G) \geq k$ But, we know that $\omega(G) \leq \chi(G)$ for any graph G. Therefore, $\omega(G) = \chi(G) = k$, and $N_\sigma[u_i]$ is a maximum clique. □

THEOREM 2.4. *For a chordal graphs G, a maximum independent set and a minimum clique cover can be computed in linear time. Also, $\alpha(G) = k(G)$*

Proof. Given a perfect elimination ordering $\sigma = (u_1, ..u_n)$ of a graph G, process each vertex in order. When vertex u_i is reached add it to the set S, create a new set K_i and add $N_\sigma(u_i)$ to K_i. Delete the vertices in K_i from G and recurse on the resultant induced subgraph. Let $(K_1, K_2, ...K_h)$ be the cliques returned by the algorithm.

Clearly, S is an independent set, because no vertex is selected for S if it is a neighbour of a vertex that is selected earlier. We can see that it is a maximum one as follows: Let $|S| = h$. Now, any maximum independent set must have at least h vertices. Therefore, $\alpha(G) \geq h$. Also, all the cliques $K_1 \cup ..K_h$ form a clique cover. Since this is a valid clique cover, any minimum clique cover must contain at most h clique. Therefore, $k(G) \leq h$

But for any arbitrary graph, $\alpha(G) \leq k(G)$.

Therefore, $\alpha(G) = k(G) = h$ □

2.4. Comparability Graphs

DEFINITION 2.5. *A strict partial order $P = (S, \prec)$ is a binary relation "$\prec$" over a set S, such that all $a, b, c \in S$, satisfy:*

1) $a \not\prec a$ for all $a \in S$

2) if $a \prec b$ then $b \not\prec a$ (anti-symmetry)

3) if $a \prec b$ and $b \prec c$, then $a \prec c$ (transitivity)

We can model a partial order with a graph G by representing each element in S as a vertex in G and creating a directed edge from a to b if $a \prec b$.

Due to the anti-symmetry and transitivity properties, the graph is acyclic, and due to the transitivity property, the graph is transitive, that is, (a, c) is an edge whenever (a, b) and (b, c) are edges in G. It is therefore a DAG.

DEFINITION 2.6. *We say that u and v are* comparable *in a partial order P if $u \prec v$ or $v \prec u$. A graph $G = (V, E)$ is a* comparability graph *if there exists a partial order $P' = (V, \prec)$ such that $uv \in E$ if and only if u and v are comparable in P'.*

A *transitive orientation* of an undirected graph G is an assignment of directions to the edges of a graph such that the resulting graph is transitive. Clearly, a graph G is a comparability graph if and only if there exists a transitive orientation of G.

A graph representing a partial order can be converted to a comparability graph by ignoring the edge direction, and a comparability graph can be converted to a graph representing a partial order by assigning it a transitive orientation.

LEMMA 2.5. *The complement of an interval graph is a comparability graph.*

Proof. Consider an interval graph G and its corresponding interval model R. Let $\overline{G}$ be G's complement. For each pair $ab \in E(\overline{G})$, let (a, b) be the orientation if a's interval precedes b's, and (b, a) be the orientation of b's interval precedes a's. Since ab is an edge of $\overline{G}$, their intervals do not intersect, so this orientation is uniquely defined by R. It is a transitive orientation because if (a, b) and (b, c) are edges of the orientation, then a's interval precedes b's, which precedes c, which means that a's interval does not intersect c's and that it precedes it. □

The complement of an interval graph is a comparability graph, but the complement of a comparability graph is not always an interval graph. As an example, a cycle with size four is a comparability graph but not an interval graph.

2.5. Algorithms on Partial Orders

The *underlying undirected graph* $G' = (V, E')$ of a directed graph $G = (V, E)$ is the graph obtained by replacing each directed edge in E by a corresponding undirected edge in E'. In other words, we ignore the edge directions. When refering to clique, independent set or colouring of a digraph, we mean the the clique, independent set or colouring respectively of the underlying undirected graph of the digraph.

We will use the following lemmas to solve our optimization problems on partial orders:

LEMMA 2.6. *The subgraph of a DAG G induced by a clique of size k has a directed path of length $(k-1)$.*

Proof. Let $(v_1, v_2, ...v_k)$ be the vertices of a clique of size k. Now, direct all edges from v_i to v_j where $i < j$. Then there is a path of length $(k-1)$ from v_1 to v_k which is built up by taking the edge from v_1 to v_2, v_2 to v_3 and so on upto v_k. □

The converse of Lemma 2.6 applies if G is transitive:

LEMMA 2.7. *If G is transitive, a directed path induces a clique.*

Proof. Let P be a directed path in G. The last vertex in P must have incoming trasitive edges from all vertices that precede it in P. Iterating this argument, the directed path must induce a clique. □

LEMMA 2.8. *In a DAG G, it takes linear time to find the longest directed path in G.*

Proof. We begin by dividing the graph into layers using algorithm 1. This is done by assigning a weight of negative one to each edge and *relaxing* all edges as described in step six of algorithm 1. Vertices returned with weight i are assigned to layer i. This algorithm

Algorithm 1 Computing graph layers

```
1: procedure GRAPH –LAYERS
2:     Assign a weight of -1 to all edges.
3:     Assign a weight of 0 to each vertex.
4:     for Each vertex $u$ in topological order of G do
5:         for Each neighbour $v$ of $u$ do
6:             $wt(v) < min(wt(v), wt(u) - 1)$
7:         end for
8:     end for
9: end procedure
```

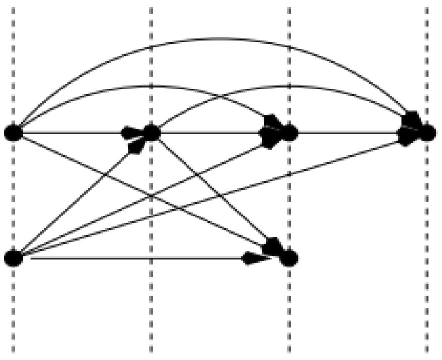

FIGURE 2.5. Stretching out the graph divides it into layers

also gives us a set of directed paths beginning at vertices without any incoming edges (called *sources*), and ending at vertices without any outgoing edges(called *sinks*). Note that the number of layers is one more than the longest path obtained in the graph.

Since we relax in topological order, this algorithm runs in $O(n + m)$ time.

THEOREM 2.5. *For a transitive DAG G, a maximum clique and a minimum colouring can be found in linear time. Also,* $\omega(G) = \chi(G)$

Proof. It follows from Lemma 2.8 that it takes linear time to compute the longest path in G. Let k be the length of the longest path of G and $(k + 1)$ the number of layers.

We assign the colour i to all vertices which belong to the layer i. This is a valid colouring because no edges can go between two vertices on the same layer. Since this is a valid colouring of size $k+1$, any minimum colouring must be of size at most $k+1$. Therefore, $\chi(G) \leq k+1$.

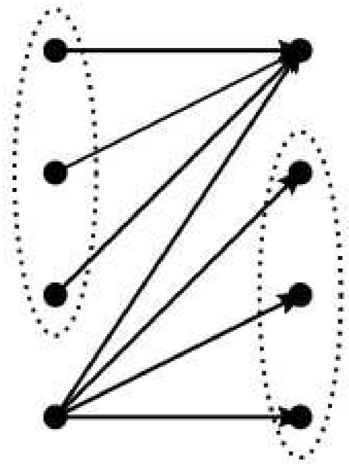

FIGURE 2.6. A maximum indepedent set of six is marked but the layers only have four vertices.

From Lemma 2.7, the path of length k induces a clique of size $k+1$. Therefore, the maximum clique must be of size at least $k+1$, that is $\omega(G) \geq k+1$ But we know that $\omega(G) \leq \chi(G)$ for any graph G. Therefore, $\omega(G) = \chi(G) = k+1$. $\square$

Unfortunately, a similar layering approach does not suffice to produce a maximum independent set. This can be seen with figure 2.6, which has a maximum independent set of six, but the layers have four nodes each.

Figure 2.6 is also an example of a *bipartite graph*. A *bipartite graph* $G = (V, E)$ is a graph whose vertices can be partitioned into two disjoint subsets $V = V_1 \cup V_2$ such that every edge $e \in E$ is of the form $e = (a, b)$, where $a \in V_1$ and $b \in V_2$. All bipartite graphs are comparability graphs because a transitive orientation can be trivially found by orienting all edges from the first to the second partition class.

A *matching* M in a graph G is a set of edges such that no two edges share a common vertex. It takes $O(mn)$ time to find a maximum matching in a bipartite graph[1].

A *vertex cover* of a graph G is a set of vertices S such that all edges of G are incident to at least one vertex in S. Note that it is NP-Complete to find a minimum vertex cover of a graph in general, but it takes $O(n^3)$ time to find one in a bipartite graph using a max flow algorithm [1].

Also, Konig[8] showed that in a bipartite graph, the size of a maximum matching is equal to the size of a minimum vertex cover.

We leverage these properties of bipartite graphs and the following lemmas to arrive at a polynomial time algorithm for maximum independent set for a comparability graph.

An *antichain* of a partial order P is a set of mutually incomparable elements; while a *chain* is a set of mutually comparable ones.

LEMMA 2.9. *In a finite partial order P', the size of a maximum antichain is equal to the minimum number of chains needed to cover all elements of the partial order[9].*

A chain in a partial order is P is a clique in its underlying comparability graph G. Therfore, partitioning a partial order into chains is equivalent to finding a clique cover. This is called the *chain partition* of the partial order. Similarly, an antichain in P is an independent set in G.

Now, lemma 2.9 can be restated as $\alpha(G) = k(G)$ for a comparability graph G.

LEMMA 2.10. *It takes polynomial time to find a minimum chain partition of a partial order $P = (S, \prec)$.*

Proof. We begin by converting P into a bipartite graph $G = (V, E)$ as follows. For each $s \in S$, create two vertices $s', s'' \in V$. Now, create an edge $(u', v'') \in E$ whenever $u \prec v$ in P.

A matching M in G corresponds to a chain partition of P into $|S| - |M|$ chains. This is seen as follows: begin by creating $|S|$ one element chains. For each edge $(u', v'') \in M$, combine the chains ending with u and beginning with v, thereby reducing the number of chains to $|S| - |M|$ by the time the algorithm is done.

Every vertex in G is matched to a unique predecessor in a chain, except the first vertex of the chain. Therefore, the number of chains is equal to the number of unmatched vertices. Since M is a maximum matching, $|S| - |M|$ must be a minimum chain partition.

Finding the maximum matching takes $O(mn)$ time and the subsequent contractions take at most $O(m)$ time. $\square$

THEOREM 2.6. *For a transitive DAG G, a maximum independent set can be found in polynomial time. Also, $\alpha(G) = k(G)$.*

Proof. This follows from lemma 2.9 and 2.10 as follows:

Let $P = (S, \prec)$ be a partial order and G be its underlying comparability graph. We begin by constructing a bipartite graph G' from P and computing the chain partition of P as described in lemma 2.10.

Now, all that is left is to pick one element from each chain in the chain partition to find a maximum independent set.

Let U be the minimum vertex cover of the bipartite graph G'. Let $A_P = \{s \in S : s', s'' \notin U\}$. We claim that A_p is then a maximum independent set. This is because there cannot be an edge between any two vertices in A_P otherwise we have found an edge not covered by U, but since U is a vertex cover this results in a contradiction.

Due the transitivity relation in P, at most one vertex from each chain can be in A_p. Also, no two vertices from the same chain can be missing from U, otherwise the transitive edge between the two vertices will not be covered by U, a contradiction. Therefore, $|A_p| = |S| - |U|$. Since the vertex cover is a minimum one, a larger independent set is not possible.

Also, from Lemma 2.10, $k(G) = |S| - |M|$, where M is the maximum matching. But we know that for a bipartite graph $|M| = |U|$. Therefore, $\alpha(G) = k(G) = |S| - |M|$.

Clearly, it takes $O(n^3)$ to find the maximum independent set, and $O(mn)$ time to find the minimum clique cover for a partial order. □

2.6. Perfect Graphs

Perfect graphs serve to unify these graph classes and provide polynomial time algorithms for our NP-Complete problems for these graphs. For the graph classes we have examined, $\omega(G) = \chi(G)$ for any graph G in the class, and since the classes are heredity, $\omega(H) = \chi(H)$ for every induced subgraph H of G.

DEFINITION 2.7. *A graph G is* perfect *if for every induced subgraph, H,* $\omega(H) = \chi(H)$.

By Theorems 2.3 and 2.5 and the fact that interval, chordal and comparability graphs are hereditary, they are perfect. By Theorems 2.4 and 2.6 the complements of these graphs are also perfect. In fact, the *perfect graph conjecture* was that the complement of every perfect graph was perfect; no counterexample was known, but no reason why it should always be true had been discovered for some time. In 1972 Fulkerson had been working intensively on trying to prove it for months when he received word that an undergraduate, Laslo Lovasz, had proved it. Spurred on by this knowledge, he found his own proof the next day[10]. The perfect graph conjecture had become the *perfect graph theorem*:

THEOREM 2.7. *An undirected graph is perfect if and only if its complement is perfect.*

Every hereditary class of graphs is known to have a characterization in terms of the set of those graphs that are not in the class, but whose proper induced subgraphs are all in the class. These are known as *minimal obstructions* for the class, since a graph is in the class if and only if it does not have one of the them as an induced subgraph. For example, the set of minimal obstructions for the chordal graphs are the cycles of length greater than or equal to four. The minimal obstructions for interval graphs and comparability graphs have also been characterized.

An obvious question was, therefore, characterizing the perfect graphs in terms of their minimal obstructions. Berge conjectured in 1961 that they are the odd-length cycles of length greater than or equal to five, and their complements. [11]. This became known as the *strong perfect graph conjecture.*

The truth of the strong perfect graph conjecture remained one of the most important open problems in graph theory until it was finally proven in 2002 by Chudnowsky et. al.:

THEOREM 2.8. *[12] A graph G is a perfect graph if and only if neither the graph nor its complement has an induced chordless cycle of odd length.*

This is now known as the *strong perfect graph theorem.* Many results about subclasses of perfect graphs that had been proved in the intervening decades are trivial consequences of it. For example, the fact that chordal graphs are perfect is immediate from their definition as having no induced cycle of size greater than or equal to four, since the presence of one of the obstructions for perfect graphs implies such a cycle.

2.7. k-EXTENDIBLE ORDERINGS

For an ordering $\pi = (v_1, v_2, \ldots, v_n)$ of vertices of a graph, then for two vertices $u = v_i$ and $w = v_j$, we will let $u <_\pi w$ denote that $i < j$, that is, that u comes before w in the ordering. When π is understood, we may write this as $u < w$.

In [13], k-clique extendible orderings was introduced as follows:

DEFINITION 2.8. *[13] An ordering $\pi = (v_1, v_2, \ldots, v_n)$ of the vertices of an undirected graph is* k-clique extendible *if, for every k-clique $\{x_1, x_2, \ldots, x_k\}$, where $x_1 <_\pi x_2 <_\pi \ldots, x_{k-1} <_\pi x_k$ and $u <_\pi x_1$ such that $\{x_1, x_2, \ldots, x_{k-1}\}$ are all neighbors of u, then x_k is also a neighbor of u.*

The authors of [13] applied k-clique extendibility to orderings of certain classes of undirected graphs such as visibility graphs and used a dynamic programming algorithm to find the maximum clique in a visibility graph in $O(n^3)$ time. Generalizing to k we get the following theorem:

THEOREM 2.9. *Given a k-clique extendible ordering of an undirected graph, a maximum clique can be found in $O(kn^k)$ time.*

Proof. Let $\pi = (v_1, v_2, \ldots, v_k)$ be a k-clique extendible ordering of a graph G. We begin by computing all cliques of size $(k-1)$ in G. We then reverse the sequences of each clique by arranging them in right-to-left order of π, and make a list L of all the $(k-1)$-cliques, sorted in lexicographic order of their reverse sequences. For a $(k-1)$-clique K, let $M(K)$ denote the size of a maximum clique whose rightmost $(k-1)$ members are K. Next we create a table and insert each $(k-1)$-clique K and $M(K)$ in it in the order L.

We now process each vertex in the left-to-right order of π. Assume by induction on i, when v_i is reached, $M(K')$ is known for each $(k-1)$-clique K' whose rightmost member precedes v_i. The algorithm must find $M(K)$ for each $(k-1)$-clique K whose rightmost member is v_i to make the induction go through.

Let $(v_i, w_{k-1}, w_{k-2} \ldots, w_2)$ be the vertices of K, and let $(w_{k-1}, w_{k-2}, \ldots, w_2, w_1)$ be a clique K' such that v_i is adjacent to w_1. Then the members of $M(K')$ and v_i is a clique by definition of k-extendibility. Further, $M(K') + 1$ is a lower bound on $M(K)$. The table entry for K is then updated with largest of these $M(K') + 1$, and we install a back pointer from K to K' in the table. If there is no such clique K', $M(K)$ is trivially $(k-1)$, and the back pointer in null.

After all vertices have been processed, we scan through the table to find the largest value of $M(K)$. This is the size of the maximum clique in G. Let this be maximum for some clique K. We recover the vertices in the maximum clique as follows:

If the back pointer associated with K is null, the maximum clique is just K. Otherwise, the vertices of K are the $(k-1)$ rightmost elements of the maximum clique, and we follow the backpointer to some clique K'. K/K' and removing the leftmost vertex from K gives us a single vertex which goes to the left of the elements of the maximum clique recovered so far. Now, we recurse on K' to recover all the elements of the maximum clique.

This algorithm runs in polynomial time as follows. It takes $O(k^2 n^{k-1})$ time to find all the $(k-1)$-cliques of G. For the induction step at v_i, all the cliques whose rightmost vertices are v_i are consecutive in the table. Also, removing v_i from this clique gives a clique of size $(k-2)$, say k'. All the $(k-1)$-cliques with k' as its rightmost vertices are consecutive in the table. Let $(k_1, k_2, \ldots k_j)$ be all these $(k-2)$ cliques. All this can be done in by scanning the table in $O(kn^{k-1})$ time.

It takes $O(n)$ time to find a subset of the $(k-2)$-cliques whose leftmost vertices are adjacent to v_i. Therefore, over all $O(n^{k-1})$ cliques, it takes $O(n^k)$ time to complete the inductive step.

Since, the maximum size of the table is n^{k-1} it takes $O(n^{k-1})$ time to find the clique K that maximizes $M(K)$. It takes $O(n)$ to list the elements of the maximum clique.

Therefore, overall it takes $O(k^2 n^{k-1}) = O(kn^k)$ time to find a maximum clique. $\square$

The question of finding the minimum k such that a ordering is k-clique extendible in polynomial time was left open by the authors of [13].

CHAPTER 3

Approximating transitivity in DAGs

As we have seen, the otherwise NP-hard problems of finding a maximum clique, minimum coloring, maximum independent set and minimum clique cover are polynomial for the underlying graph of a transitive DAG. They are NP-hard for the underlying graphs of arbitrary DAGs, which is seen as follows. Given an arbitrary graph G, we can find a DAG D that has G as its underlying graph by assigning an arbitrary ordering $(v_1, v_2, \ldots, v_n)$ of the vertices of G and orienting the edges from left to right in the ordering, that is, ordering each edge v_iv_j as (v_i, v_j) if $i < j$. A polynomial-time algorithm for solving any of the optimization problems on a DAG would give one for an arbitrary graph G, implying P = NP.

In this work, we derive a measure β of the *intransitivity* of a DAG that measures the degree to which it departs from a transitive DAG, and show that this measure is closely tied to the complexity of solving the four optimization problems on its underlying undirected graph, using the DAG. The value of β is at least 1 and at most $n - 1$. It is equal to 1 if the DAG is transitive. Finding a maximum clique takes $O(\beta^2 n^{\beta+1})$, time, which is polynomial for fixed β. For the other optimization problems, we give polynomial-time approximation algorithms with approximation bounds of $\lceil \beta + 1 \rceil$ or $\lceil (\beta + 1)/2 \rceil$.

Thus, the new measure imposes a nested hierarchy of DAGs, each more computationally complex with respect to the optimization problems than its predecessor, spanning the extreme cases of a transitive DAG and an arbitrary DAG.

This work was done jointly with Mmanu Chaturvedi and Xu Zhisheng.

3.1. A measure of the degree of departure of a DAG from transitivity

A graph is transitive if and only if for every directed path P, there is a directed edge from the first to the last vertex of P. Reframing transitivity in this way gives an approach to incrementally relaxing the constraint as follows, to obtain a measure of the degree to which an arbitrary DAG departs from transitivity.

DEFINITION 3.1. *For a DAG $G = (V, E)$, let $\beta(G)$ be the length of the longest directed path P of G such that there is no directed edge from the first to the last vertex of P. If there is no such path, that is, if G is transitive, let $\beta(G) = 1$.*

If G is a DAG that is not transitive, then $\beta(G) \geq 2$, so a DAG is transitive if and only if $\beta(G) = 1$.

THEOREM 3.1. *It takes $O(nm)$ time to compute $\beta(G)$ for a DAG G.*

Proof. We can compute $\beta(G)$ for a DAG G by exploiting the topological sort and running a series of *breadth first searches* (BFS)[1] on the vertices of G. Running a BFS from a vertex u gives us the shortest path to all the other vertices reachable from u.

Let $(u_1, u_2, \ldots, u_n)$ be a topological sort of G. We proceed to run a BFS from each vertex in topological order. Running a BFS from u_i gives us the furthest distance vertex reachable from u_i, $\beta_i(G)$. The vertex that maximizes this value over all u_i gives us the desired value for $\beta(G)$.

It takes $O(n + m)$ time to topologically sort the vertices of G. BFS and subsequent calculations take $O(n+m)$ time for each vertex and therefore $O(nm)$ total over n vertices. □

Note that $\beta(G)$ is a hereditary property for a DAG G. This defines a hierarchy of graph classes with transitive DAGs at the base, and increasingly complex graph classes above it. We now show that $\beta(G)$ is a measure of intractability of the optimization problems on G.

In [13] k-extendible orderings are defined for undirected graphs. Borrowing from this idea, we note that a DAG is transitive if and only if any given topological sort is two-clique extendible.

THEOREM 3.2. *For a DAG G and given k, either all topological sorts of G are k-clique extendible or none are.*

Proof. Suppose some topological sort $\pi = (v_1, v_2, \dots, v_k)$ of G is k-clique extendible. Then for each k-clique $\{x_1, x_2, \dots, x_k\}$, where $x_1 <_\pi x_2 <_\pi \dots, x_{k-1} <_\pi x_k$ and each $u <_\pi x_1$ such that x_1 is a neighbor of u, then for any topological sort τ, $x_1 <_\tau x_2 <_\tau \dots, x_{k-1} <_\tau x_k$ and $u <_\pi x_1$. The conditions that must be met for k-extendibility of π and τ are the same. □

Thus, k-clique extendibility of a topological sort is a property of a DAG, independently of the choice of any particular topological sort. The question of whether a DAG is k-clique extendible does not need to reference a topological sort, and we define a DAG to be *k-clique extendible* if its topological sorts are such.

Now, since k-clique extendibility is a property of a DAG and not a particular topological sort we claim the following theorem:

THEOREM 3.3. *A graph with $\beta(G) = k$ is $(k+1)$-clique extendible.*

Proof. Let a DAG G have a $\beta(G) = k$. Let vertex v_i precede a clique K of size k and v_j be adjacent to all members of K and occur after K.

From lemma 2.6, the longest path from v_i to v_j is of length $(k+1)$ as follow: v_i and clique K form a clique of size $k+1$ and has a longest path of size k from v_i to the rightmost vertex of K. Again, K forms a clique with v_j, therefore there must be an edge from the rightmost

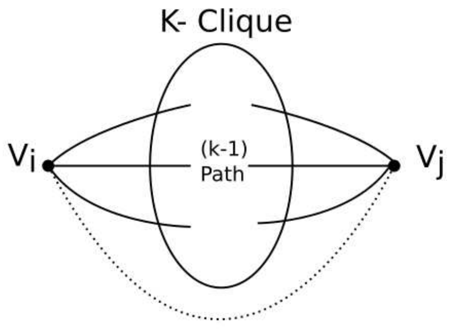

FIGURE 3.1. A graph with $\beta(G) = k$ is also (k+1)-clique extendible.

vertex in K to v_j. Therefore, there is a path of length $k+1$ from v_i to v_j. But by definition of $\beta(G)$, there must exist an edge from v_i to v_j. Therefore, the graph is $(k+1)$-clique extendible. □

However, a graph G, with value $\beta(G) = k$ is at least $(k+1)$-clique extendible, but this is not a tight bound. This is an approximation of the k for which the graph is $(k+1)$ clique extendible. This can be seen with a simple directed path, say P. P is vacuously 3-clique extendible, but its β value varies with the length of the path.

3.2. Algorithms with β measure

Now that we have a way to quickly approximate the departure from transitivity of a graph G, we focus on solving our optimization problems on G. By a *clique*, *independent set*, *coloring*, and *clique cover* of a DAG, we mean a clique, independent set, coloring, and clique cover of the underlying undirected graph. The problem of finding the maximum clique is solvable in time polynomial in $\beta(G)$, but the other problems prove more difficult and remain NP-Complete. We give fast approximation algorithms for these NP-Complete problems.

3.2.1. Maximum Clique.

Corollary 1. *It take $O((k+1)n^{k+1})$ time to find the maximum clique of a DAG G with $\beta(G) = k$.*

Proof. Immediate from Theorem 3.3 and Theorem 2.9. $\square$

theorem 3.4. *There is a linear-time $(\beta(G)+1)$-approximation algorithm for maximum clique of a DAG.*

Proof. By Lemma 2.8 it takes linear time to find a longest directed path in a DAG. Let l be the number of vertices in a longest directed path P. By Lemma 2.6, l is an upper bound on the size of a maximum clique. Let X be the set of vertices obtained by taking every $(k+1)^{th}$ vertex of P, starting with the first. Since $\beta(G) = k$, X is a clique, and it has $\lceil l/(k+1) \rceil$ vertices. $\square$

3.2.2. Maximum Independent Set.

theorem 3.5. *It is NP-complete to decide if a DAG G has an independent set of size k even for a graph with $\beta(G) = 2$.*

We know that it is NP-complete to decide if there exists an independent set of size k in a tripartite graph(3-Colourable Graph with a 3-colouing given). Let a tripartite graph G' be coloured with (C_1, C_2, C_3). Now, direct all edges from a lower colour to a higher colour. This gives a DAG, with the longest path length of two. Therefore, $\beta(G')$ of this graph is no worse than two. Now, given an algorithm to decide if there exists an independent set of size k in a DAG G with $\beta(G) = 2$, we can decide if there is an independent set of size k in G'.

Also, given a possible independent set of size k, we can verify in polynomial time if this is a valid solution. Therefore, the problem is NP-Complete. $\square$

However, it is possible to get a polynomial time approximation algorithm for independent set. We first define the following:

The *path cover* of a graph $G = (V, E)$ is a set of paths such that every vertex $v \in V$ belongs to exactly one path. In general, it is NP-Complete to find a *minimum path cover* for G. However, it takes polynomial time to compute the minimum path cover if G is a DAG using the max flow algorithm[1].

The *transitive closure* of a DAG $G = (V, E)$ is another DAG $G' = (V, E')$ such that $(i, j) \in E'$ if and only if there is a path from i to j in G. The transitive closure essentially fills in all the missing transitive edges of G. Therefore, G' is a partial order.

THEOREM 3.6. *For a DAG G with $\beta(G) = k$, there is a polynomial time $\lceil (k+1)/2 \rceil$-approximation algorithm for maximum independent set.*

Proof. We proceed by first finding the transitive closure G' of the DAG G. We can then find an independent set for G' using the algorithm for finding the maximum independent set for partial order graphs. This takes polynomial time.

If X is a maximum independent set of the DAG G, and Y is the independent set for G' returned by our algorithm, we claim that the number of elements in X is at most $\lceil (k+1)/2 \rceil$ times the number of elements of Y.

To see this, we first divide our DAG into a minimum path cover $(P_1, P_2, ..P_k)$.

Now, consider each of these path individually. Because the $\beta(G)$ of the entire graph is equal to k, the β of the subgraphs induced by the vertices of each of these paths must be less than or equal to k. The number of vertices that can be added to X from each path P_i is at most $\lceil (k+1)/2 \rceil$, when every other vertex can be added to the independent set. If we choose only one vertex from P_i to add to Y, $|Y|$ is no worse than $\lceil (k+1)/2 \rceil$ times $|X|$.

Similarly, we pick one vertex from each of these paths and arrive at a $\lceil (k+1)/2 \rceil$ approximation ratio for the independent set. $\square$

3.2.3. Minimum Clique Cover.

THEOREM 3.7. *For a DAG G with $\beta(G) = k$, there is a linear time $(k+1)$-approximation algorithm for minimum clique covering.*

Proof. Minimum Clique cover follows from the partitioning of the graph into its minimum path cover and the approximation algorithm for maximum clique. We begin by dividing the G into the minimum path cover $(P_1, P_2, ..P_k)$. On each of these paths P_i, it follows from Theorem 3.4 $(v_0, v_{k+1}, v_{2k+2}...)$ vertices form a clique. Therefore, on each path $(v_0, v_{k+1}, v_{2k+2}...)$, $(v_1, v_{k+2}, v_{2k+3}...)$, $(v_2, v_{k+3}, v_{2k+4}...)$ .. are all cliques, and over all paths form a clique cover of the DAG G.

Let X be the the set of minimum clique cover for the DAG G, and Y be the clique cover returned by our algorithm. In the worst case, the entire path is a clique and the set of paths is a minimum clique cover. The set Y has a maximum of $(k+1)$ cliques for each of the paths in X. Therefore, over all paths and in the worst case, set Y is no bigger than $(k+1)$ times the size of X. $\square$

3.2.4. Minimum Colouring.

THEOREM 3.8. *For a DAG G with $\beta(G) = k$, there is a linear time $(k+1)$-approximation algorithm for minimum colouring.*

Proof. For minimum colouring, we first create a source vertex s' and create outgoing edges from s' to all the sources of the DAG. Assign a weight of -1 to all edges of the DAG, then relax all the edges in topological order, finding the maximum distance to all the vertices from the source s'. All vertices at a distance i from the s' form a level L_i. We can colour all vertices in each level L_i with the colour i.

We claim this is a valid colouring. Let us assume for contradiction that this is not a valid colouring, but all vertices are at a maximum distance from s'. Since edges between different labels have different colours at its end points, the only edges spoiling the colouring must have the end points on the same level.

Assume, without loss of generality, there exists an edge directed from x to y in some level L_i. Now, we can take the path to vertex x, and take the edge from (x, y) which has a weight of -1. This gives y a higher distance than x, so it is not a maximum distance from source s', which is a contradiction.

If X is the minimum colouring of DAG G, and Y is the colouring returned by our algorithm, we claim that the set Y is at most $(k+1)$ times the size of X.

It is sufficient to show this holds for the maximum length path, because all shorter paths can be coloured in less than the number of colours needed for the longest path. On the maximum length path, consider vertex v_i. Even in an optimal colouring, all vertices at a distance of $(k+1)$ or more from v_i must have a separate colour. Therefore, the optimal colouring is confined in a distance of $(k+1)$ on the path.

Our algorithm returns $(k+1)$ colours for this section, while an optimal colouring can be possibly done in just one colour. Therefore, set Y is no bigger than $(k+1)$-times the size of set X. $\square$

Bibliography

[1] C. Leiserson, T. Cormen, C. Stein, and R. Rivest, *Introduction to Algorithms.* MIT Press, 3 ed., 1990.

[2] R. M. Karp, “Reducibility among combinatorial problems,” *Complexity of Computer Computations*, p. 85103, 1972.

[3] J. Hastad, “Clique is hard to approximate,” *Acta Mathematica*, pp. 105–142, 1999.

[4] U. Feige, “Approximating maximum clique by removing subgraphs,” *SIAM Journal on Discrete Mathematics*, p. 219225, 2004.

[5] Halldorsson and J. Radhakrishnan, “Greed is good: Approximating independent sets in sparse and bounded-degree graphs,” *Algorithmica*, p. 145163.

[6] Halldorsson, “A still better performance guarantee for approximate graph coloring,” *Information Processing Letters*, p. 1923, 1993.

[7] D. Rose, G. Lueker, and R. E. Tarjan, “Algorithmic aspects of vertex elimination on graphs,” *SIAM Journal on Computing*, pp. 266–283, 1976.

[8] D. Konig, “Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére,” *Matematikai s Természettudományi Értesitö*, p. 104119, 1916.

[9] R. P. Dilworth, “A decomposition theorem for partially ordered sets,” *Annals of Mathematics*, pp. 161–166, 1950.

[10] M. C. Golumbic, *Algorithmic graph theory and perfect graphs.* MIT Press, 2 ed., 1980.

[11] C. Berge, “Farbung von graphen, deren samtliche bzw. deren ungerade kreise starr sind,” *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, p. 104, 1961.

[12] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas, “The strong perfect graph theorem,” *Annals of Mathematics*, pp. 51–229, 2006.

[13] J. P. Spinrad, *Efficient Graph Representations, Fields Institute Monographs.* American Mathematical Society, 19 ed., 1991.