Article

Layered Graphs: A Class that Admits Polynomial Time Solutions for Some Hard Problems

Bhadrachalam Chitturi [1,2], **Srijith Balachander** [1], **Sandeep Satheesh** [1], **Krithic Puthiyoppil** [1]

[1] Department of Computer Science, Amrita Vishwa Vidyapeetham, Amritapuri, India
[2] Department of Computer Science, University of Texas at Dallas, Richardson, Texas 75083, USA

Version April 27, 2018 submitted to Algorithms

Abstract: The independent set, IS, on a graph $G = (V, E)$ is $V^* \subseteq V$ such that no two vertices in V^* have an edge between them. The MIS problem on G seeks to identify an IS with maximum cardinality, i.e. MIS. $V^* \subseteq V$ is a vertex cover, i.e. VC of $G = (V, E)$ if every $e \in E$ is incident upon at least one vertex in V^*. $V^* \subseteq V$ is dominating set, DS, of $G = (V, E)$ if $\forall v \in V$ either $v \in V^*$ or $\exists u \in V^*$ and $(u, v) \in E$. The MVC problem on G seeks to identify a vertex cover with minimum cardinality, i.e. MVC. Likewise, MCV seeks a connected vertex cover, i.e. VC which forms one component in G, with minimum cardinality, i.e. MCV. A connected DS, CDS, is a DS that forms a connected component in G. The problems MDS and MCD seek to identify a DS and a connected DS i.e. CDS respectively with minimum cardinalities. MIS, MVC, MDS, MCV and MCD on a general graph are known to be NP-complete. Polynomial time algorithms are known for bipartite graphs, chordal graphs, cycle graphs, comparability graphs, claw-free graphs, interval graphs and circular arc graphs for some of these problems. We introduce a novel graph class, layered graph, where each layer refers to a subgraph containing at most some k vertices. Inter layer edges are restricted to the vertices in adjacent layers. We show that if $k = \Theta(\log |V|)$ then MIS, MVC and MDS can be computed in polynomial time and if $k = O((\log |V|)^\alpha)$, where $\alpha < 1$, then MCV and MCD can be computed in polynomial time. If $k = \Theta((\log |V|)^{1+\epsilon})$, for $\epsilon > 0$, then MIS, MVC and MDS require quasi-polynomial time. If $k = \Theta(\log |V|)$ then MCV, MCD require quasi-polynomial time. Layered graphs do have constraints such as bipartiteness, planarity and acyclicity.

Keywords: NP-complete, graph theory, layered graph, polynomial time, quasi-polynomial time, dynamic programming, independent set, vertex cover, dominating set.

---

1 Introduction

The maximum independent set problem, the minimum vertex cover problem and the minimum dominating set problem are well studied problems on graphs with myriad applications. All of these problems are shown to be NP-complete. Thus, identifying more general graph classes that admit polynomial solutions to these problems is of interest.

The maximum independent set problem on a graph $G = (V, E)$ seeks to identify a subset of V with maximum cardinality such that no two vertices in the subset have an edge between them. If $V^* \subseteq V$ is a maximum independent set or MIS for short of G then $\forall u, v \in V^*$, $(u, v) \notin E$. In this article G is undirected, so, an edge (u, v) is understood to be an undirected edge.

Karp proposed a method for proving problems to be NP-complete [17]. The maximum independent set problem on a general graph is known to be NP-complete [15]. Certain classes of graphs admit a polynomial time solution for this problem. Such algorithms are known for trees and bipartite graphs [1], chordal graphs [2], cycle graphs [3], comparability graphs [6], claw-free graphs [7], interval graphs and circular arc graphs [8]. The

maximum weight independent set problem is defined on a graph where the vertices are mapped to corresponding weights. The maximum weight independent set problem seeks to identify an independent set where the sum of the weights of the vertices is maximized. On trees, the maximum independent set problem can be solved in linear time [10]. Thus, for several classes of graphs MIS can be efficiently computed.
Hsiao et al. design an $O(n)$ time algorithm to solve the maximum weight independent set problem on an interval graph with n vertices given its interval representation with sorted endpoints list [12]. Several articles improved the complexity of the exponential algorithms that compute an MIS on a general graph [5,9]. Lozin and Milanic showed that MIS is polynomially solvable in the class of $S_{1,2,k}$-free planar graphs, generalizing several previously known results where $S_{1,2,k}$ is the graph consisting of three induced paths of lengths 1, 2 and k, with a common initial vertex [13].

The minimum vertex cover problem on G seeks to identify a vertex cover with minimum cardinality, i.e. minimum vertex cover or MVC. If $V* \subseteq V$ is MVC of G then $\forall e = (u, v) \in E, u \in E \vee v \in E$. In this article G is undirected, so, an edge (u, v) is understood to be an undirected edge. The problems minimum dominating set, i.e MDS and the minimum connected dominating set i.e. MCD seek to identify a DS and a CDS respectively with minimum cardinalities. The MVC, MDS and MCD problems on a general graphs are known to be NP-complete [15]. Garey and Johnson showed that MVC is one first NP-complete problem [15]. In connected vertex cover problem i.e. MCV, given a connected graph G, a connected vertex cover i.e. a CVC with minimum cardinality is sought. Garey and Johnson proved that MCV is NP-complete [18]. For trees and bipartite graphs the minimum vertex cover can be identified in polynomial time [20,21]. Garey and Johnson proved that MCV problem is NP-hard in planar graphs with a maximum degree of 4 [15]. Li et. al. proved that for 4-regular graph MCV problem is NP-hard [19]. It is shown that for series-parallel graphs, which are a set of planar graphs, it shown that minimum vertex cover can be computed in linear time [23].

Garey and Johnson showed that MDS on planar graphs with maximum vertex degree 3 and planar graphs that are regular with degree 4 are NP-complete [15]. MCD is NP-complete even for planar graphs that are regular of degree 4 [15]. Bertossi showed that the problem of finding a MDS is NP-complete for split graphs and bipartite graphs [22]. Cockayne et. al. proved that MDS in trees can be computed in linear time [4]. Haiko and Brandstadt showed that MDS and MCD are NP-complete for chordal bipartite graphs [24]. Ruo-Wei et. al. proved that for a given circular arc graph with n sorted arcs, MCD is linear in time and space [25]. Fomin et. al. propose an algorithm with time complexity faster than 2^n for solving connected dominating set problem [26].

The term layered graph has been used in the literature. The hop-constrained minimum spanning tree problem related to the design of centralized telecommunication networks with QoS constraints is NP-hard [14]. A graph that they call a *layered graph* is constructed from the given input graph and authors show that hop-constrained minimum spanning tree problem is equivalent to a Steiner tree problem. In software architecture the system is divided into several layers, this has been viewed as a graph with several layers. In this article we define a new class of graphs that we call *layered graphs* and design an algorithm to identify the corresponding minimum vertex cover.

2 Layered Graph

Consider a set of undirected graphs $G_1, G_2, \ldots G_q$ on the corresponding vertex sets $V_1, V_2, \ldots V_q$ and the edge sets $E_1, E_2, \ldots E_q$ i.e. $G_i = (V_i, E_i)$. Consider a graph G that is formed from $\forall_i\ G_i$ with special additional edges called *inter-layer edges* denoted as E_{ij} where $j = i + 1$ and E_{ij} denotes the edges between V_i and V_j. We call such a graph a *layered graph* denoted as LG i-th layer is G_i. Note that for any given i, E_{ij} where $j = i + 1$ can be ϕ and $\forall_{l \notin \{i-1,i+1\}} E_{il} = \phi$. Every vertex within a given layer gets a label from $(1, 2, 3, \ldots, k)$. Thus, $V_i \in \{V_{i1}, V_{i2}, \ldots V_{ik}\}$. Note that V_{ix} is the vertex number x in layer i. However, in layer i the vertex number x need not exist. Further, if $(V_{ix}, V_{i+1\ y}) \in E_{i\ i+1}$ then it follows that vertex x is present in layer i and vertex y is present in layer $i + 1$.

We define the following restrictions on a layered graph. Several of the primary restrictions can be combined. Please see Figure 1.

- The size of all graphs is restricted such that $\mid V_i \mid \leq k$ then a *k-restricted layered graph* i.e. LG_k is obtained. LG_k^q denotes an LG with q layers. $LG_k^{n,q}$ denotes an LG_k^q with n vertices.
- If $\forall_t$ for V_{it} the only permissible edges are (V_{it}, V_{jt}) where $j \in \{i-1, i+1\}$ then a *linear layered graph* i.e. LLG is obtained. LLG_k denotes an LLG that is *k-restricted*. LLG_k^q denotes an LLG_k with q layers. $LLG_k^{n,q}$ denotes an LLG_k^q with n vertices.
- If every G_i is required to be a connected component then a *single component layered graph* i.e. SLG is obtained.
- If G is required to be a connected component then a *connected layered graph* i.e. CLG is obtained.

This article designs algorithms for LG_k where every vertex within a given layer gets a label from $\{1, 2, 3, \ldots k\}$. The results are applicable for any restrictions of LG_k like LLG, SLG etc.. Consider a layered graph G whose first a layers and the last b layers do not have any edges. The graph is not a CLG, however, a MCV of G is same as the MCV of the subgraph where the first a and the last b layers are removed. Further, if every layer has at least one edge then MCV also requires a CLG. MCD is well defined only for CLG because it must dominate all vertices.

The recursive process of generating a hypercube of dimension $n+1$ i.e. H_{n+1} from two copies of H_n consists of creating the *inter-H_n edges* $\forall_i$ (v_{1i}, v_{2i}) where v_{1i} and v_{2i} are the corresponding vertices from the first copy of H_n and the second copy of H_n respectively. Thus, the *inter-layer edges* of LLG are in fact akin to a subset of *inter-H_n* edges because an *inter-H_n* edge exists between every pair of corresponding edges. However, in an LLG the successive layers need not have all allowed edges; moreover, $\mid V_i \mid$ and $\mid V_{i+1} \mid$ need not be identical.

The complete graph on k vertices, a *clique* on k vertices, is denoted by K_k. Consider a graph G formed from several copies of K_k say $G_1, G_2, \ldots G_q$ where in addition to the edges that exist in each of G_i an edge is introduced between every pair u, v: $u \in G_i$ and $v \in G_{i+1}$. We denote this particular graph G that has q layers with K_k^q. The class of *k-restricted layered graphs* are in fact subgraphs of K_k^q. Thus, we call K_k^q as *full* LG_k^q. Likewise, a LLG that is defined on q cliques, where for any $i, i+1$ for all values of l an edge is introduced between vertex l of layer i and vertex l of layer $i+1$, is called as a *full* LLG_k^q. The number of layers in LG_k i.e. q is bounded by $n/k \leq q \leq n$.

A subgraph of G *induced* by vertices $u_1, u_2, \ldots u_i$ consists of all vertices $u_1, u_2, \ldots u_i$ and all the edges restricted to them. We design algorithms that compute the cardinalities of MVC, MIS and MDS of any subgraph of K_k^q i.e. $LG_k^{n,q}$ in polynomial time when $k = O(\log n)$ and the cardinalities of MCV and MCD in polynomial time when $k = O((\log n)^\alpha)$, $\alpha < 1$. Additionally, these algorithms report the corresponding numbers of MISs, MVCs, MDSs, MCVs and MCDs in $LG_k^{n,q}$.

3 Algorithm

Consider a layered graph with q layers i.e. $LG_k^{n,q}$ with layers $(1, 2, 3, \ldots, q)$. We design a generic dynamic programming algorithm for all the problems. However, certain restrictions exist corresponding to the problem at hand. The specific details pertaining to each problem are elucidated along with its solution. For example, MCD is meaningful only when the underlying graph is connected; i.e. the input graph is restricted to CLG.

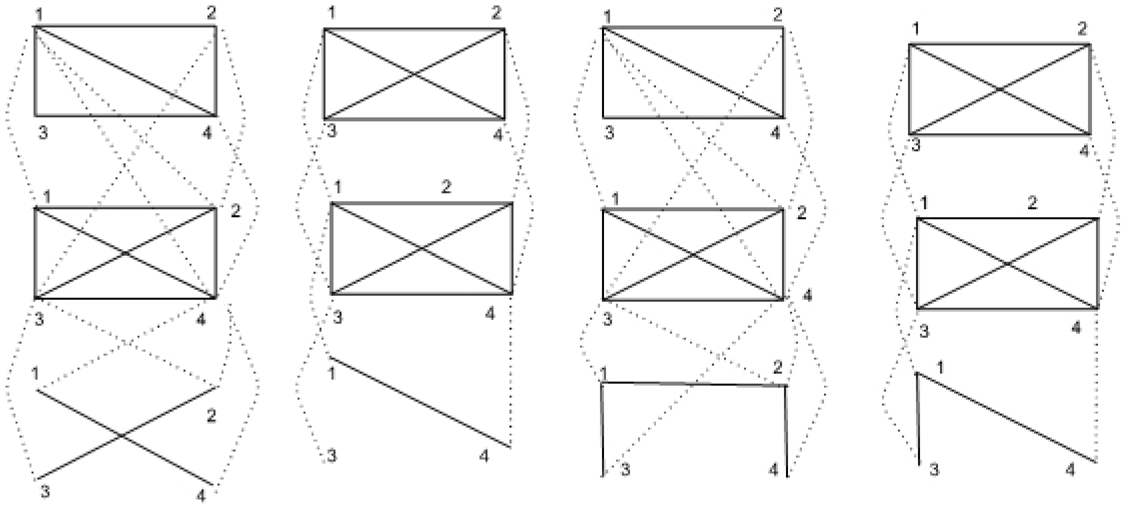

Figure 1. From left to right: 1a) $LG_{4r}^{3,12}$. 1b) $LLG_{4r}^{3,10}$. 1c) $SLG_{4r}^{3,11}$. 1d) $SLLG_{4r}^{3,11}$. In single component graphs, each layer has exactly one connected component. The vertices are labeled $1, 2, 3, 4$ within the given layer. The edges between the vertices of a given layer are shown with thick lines whereas an $e \in E_{i\ i+1}$ is shown with a dotted line. The graph is labeled. In a linear graph the edges $\in E_{i\ i+1}$ connect the vertices with identical labels from adjacent layers.

We denote the vertices chosen in a particular layer with a k-bit variable that we call as *mask*. The p^{th} bit of the mask is set to one to include p^{th} vertex. Otherwise, the bit is set to zero and the vertex is excluded. Let $S = \bigcup_{i=1}^{q} V_i^*$ be a candidate solution for a problem where V_i^* denotes the set of nodes that are chosen from layer i. The candidate sub-solution for layer i is denoted as cs_i. For layers $1 \ldots i$, we maintain a combined candidate sub-solution denoted as ccs_i. Likewise, $cs_{i,j}$ and $ccs_{i,j}$ each denote instances where the vertices chosen from layer i are denoted by mask j. We store only the cardinality of the best options; such cardinality is called an *optimum value*. This is stored in the variable $sol_{i,j}$ and the corresponding number of solutions that yield the optimum value is stored in $count_{i,j}$. In this article, an *optimal solution* is a solution that corresponds to the optimum value. We say that $cs_{i,j}$ and $ccs_{i-1,l}$ are *compatible* if $cs_{i,j} \bigcup ccs_{i-1,l} \in ccs_{i,j}$. That is the union of $cs_{i,j}$ and $ccs_{i-1,l}$ yields a ccs for the first i layers. Note that compatibility is determined by $cs_{i,j}$ and $cs_{i-1,l} \in ccs_{i-1,l}$ and the vertices chosen by $ccs_{i-1,l}$ in the earlier layers is irrelevant. This is a key feature.

3.1 Input

The input consists of $LG_k^{n,q}$ that is specified in terms of $M_1, \ldots, M_q$ and $I_1, \ldots, I_{q-1}$ where M_i is the 0-1 adjacency matrix for layer i i.e. G_i. I_i is the 0-1 adjacency matrix for $E_{i,i+1}$. The rows $1, 2, \ldots k$ of I_i correspond to the vertices $V_{i1}, V_{i2}, \ldots V_{ik}$ and the columns $1, 2, \ldots k$ of I_i are the vertices $V_{i+1\ 1}, V_{i+1\ 2}, \ldots V_{i+1\ k}$. It must be noted that for a linear graph, I_i can just be a k dimensional vector and the corresponding computation is less expensive where $I_i[a] = 1 \iff$ an edge between $a \in V_i, a \in V_{i+1}$ exists. The adjacency matrix M_i, for layer i, is a matrix of dimensions $k \times k$, which means it requires $O(k^2)$ space. Similarly, each of G_i also requires $O(k^2)$ space. Therefore, the total space required for the input graph would be $O(nk)$, since each layer requires $O(k^2)$ space and there are $O(n/k)$ layers.

The boolean valued function *compatible* is called to determine whether candidate sub-solutions (of the current layer and the subgraph induced up to the previous layer) can be combined; here the layer number i is implicit. For each mask j of a given layer i a function $valid(i, j)$ determines if j is a feasible option for layer i. The helper function $cardinality(j)$ returns the number of bits that are set in the binary representation of some

 doi:10.20944/preprints201805.0012.v1

mask j.

All algorithms consist of the following sequence of computational tasks.

- Repeat (i) and (ii) for all layers $1 \ldots q-1$.
- (i) Feasible: $\forall_j$ (if $valid(i,j)$) then go to step(ii).
- (ii) Extension: If j and l are compatible then store the cardinality of $cs_{i,j} \bigcup ccs_{i-1,l}$ in $sol_{i,j}$ and the count of $ccs_{i,j}$ in $count_{i,j}$. Corresponding to each $cs_{i,j}$ if 2^k additional variables are present then update them (e.g. DS problems).
- (iii) Summarize: At layer q : execute (i) and (ii). Identify the optimum cardinality among $\forall_j sol_{q,j}$ and the corresponding count.

Each problem has specific characteristics. The compatibility criteria and other specifics for each of the problems is elucidated below.

3.2 MIS

Consider the structure of a MIS on $LG_k^{n,q}$. Say, $V^* = \bigcup_{j=1}^{q} V_j^*$ where V_j^* are the vertices in MIS from layer j. Clearly, V_j^* must be an IS. Let G_1 be the subgraph of $LG_k^{n,q}$ induced by $V^1 = \bigcup_{j=1}^{i} V_j$ and let G_2 be the subgraph of $LG_k^{n,q}$ induced by $V^2 = \bigcup_{j=i+1}^{q} V_j$. Consider the IS of G. IF $M_1 = \bigcup_{j=1}^{i} V_j^*$ and $M_2 = \bigcup_{j=1+1}^{q} V_j^*$ then M_1 and M_2 are ISs. Let the set of edges crossing the cut $C = (M_1, M_2)$ be E^C. It follows that $M_1 \bigcup M_2$ is an IS of G with cardinality $\mid M_1 \mid + \mid M_2 \mid$ when there is no edge crossing C. Only edges in $E_{i\ i+1}$ need to be considered. Thus, the cardinality of an MIS of $LG_k^{n,q} = max(\forall_{E^C=\phi} \mid M_1 \mid + \mid M_2 \mid)$.

- $feasible(j)$: the mask j must denote an IS for G_i.
- $compatible(j,l)$: the union of two ISs must be an IS.
- Extension: if$(cardinality(j) + sol_{i-1,l} > sol_{i,j})$ $sol_{i,j} \leftarrow cardinality(j) + sol_{i-1,l}$.
- Summarize: Let $opt \leftarrow max(\forall_j sol_{q,j})$;$count \leftarrow 0$; $\forall_j$ $if(sol_{q,j} = opt) count \quad count + count_{q,j}$; Return $(opt, count_{q,j})$)

3.3 MVC and MCV

Consider the VC $V^* = \bigcup_{j=1}^{q} V_j^*$ of $LG_k^{n,q}$ where V_j^* denotes the set of vertices in V^* from layer j. Clearly, V_j^* is a VC for layer j. V_j^* depends only on V_{j-1}^* and V_{j+1}^*.

Consider two adjacent layers p and $p+1$. $V_p^* \bigcup V_{p+1}^*$ must cover all inter-layer edges between layers p and $p+1$. Specifically, $V^* = \bigcup_{j=1}^{p+1} V_j^*$ must cover all edges in the corresponding induced subgraph including $E_{p\ p+1}$. Similar constraints hold for MCV. Additionally the induced subgraph of V^* must be a single connected component. The time and space complexity analysis for both the problems is mentioned in later sections.

Clearly each layer must choose a mask that is a VC. In the case of MCV, when considering a mask j for the current layer i the following cases exist.

(a) The previous layer mask l corresponds to one component.

(b) l corresponds to more than one component.

Case(a): For layer i the mask j is infeasible if no vertex from j connects with l or all the edges in I_i are not covered. Otherwise, it is feasible.

If at least one edge exists across j and l: (i) j is a single connected component then the result is also a single component (consisting of all chosen vertices).

(ii) j has more than one connected component and all of them connect to l then the result is also a single component.

(iii) j has more than one connected component and only some of them connect to l then the result consists of many components. All components from j connected to l become one component all the rest are separate components.

(iv) Thus, for a j we store all partitions of vertices where when j is chosen and the current components are

 doi:10.20944/preprints201805.0012.v1

denoted by the sets in a partition the sub-solution with minimum cardinality is chosen.
(v) Thus, for each mask j we have at most Bell Number(k) solutions stored. When the mask x is chosen for the last layer then the vertices of the mask must be connected to the components of the previous layer and yield a single component.

- $feasible(j)$: the mask j must denote a VC for G_i. For MCV j must be connected.
- $compatible(j, l)$: the union of two VCs must be a VC for edges in G_i, G_{i+1} and $E_{i,j}$.
- Extension: if$(cardinality(j) + sol_{i-1,l} < sol_{i,j})$ $sol_{i,j} \leftarrow cardinality(j) + sol_{i-1,l}$. For MCV masks j and l must have at least one edge in between.
- Summarize: Let $opt \leftarrow min(\forall_j sol_{q,j})$;$count \leftarrow 0$; $\forall_j$ $if(sol_{q,j} = opt) count \quad count + count_{q,j}$; Return $(opt, count)$

3.4 MDS and MCD

Let the MDS on $LG_k^{n,q}$ say $V^* = \bigcup_{j=1}^{q} V_j^*$ where V_j^* are the vertices in this MDS from layer j. Clearly, V_j^* need not be a DS of layer j because the V_j can be dominated by any subset of $V_{j-1}^* \bigcup V_j^* \bigcup V_{j+1}^*$. It follows that $\bigcup_{j=1}^{p+1} V_j^*$ must dominate all vertices in $\bigcup_{j=1}^{p} V_j$. Further, V^* which is obtained by $V^* = \bigcup_{j=1}^{q-1} V_j^* \bigcup V_q^*$ must dominate $\bigcup_{j=1}^{q} V_j$. A vertex that is not dominated is *undominated*.

Consider $mask = j$ in layer i. Say, $cs_{i,j} \bigcup ccs_{i-1,l}$ dominates layer $i-1$. However, this particular union of vertices does not dominate some vertices in layer i. The number of such choices is 2^k; each choice is denoted by a k-bit variable that we call mask, here, a mask of exclusion. Further, when one processes layer $i+1$ this information is significant. We show that $O(2^k)$ triples stored for each mask of a given layer suffice to compute MDS of LG_k. For a chosen mask j in layer i it suffices to store 2^k triples of the form (u, s, c). Here u is the mask of the vertices that are *not* dominated in layer i, s is the cardinality of the vertices chosen so far and c is the number of choices corresponding to un for a particular j in layer i.

In the case of MCD, it suffices to store $O(B_k 2^k)$ triples of the form (lo, un, r) where B_k is the k-th Bell Number. This corresponds to $O(B_k)$ component layouts lo for a mask j and $O(2^k)$ masks un of the vertices that are *not* dominated in layer i, and $O(2^k)$ triples r of the form (m, s, c) for every unique pair of (lo, un). Here, m is the mask of the current layer that produced the respective (lo, un) pair i.e. mask j, while s and c are same as that for MDS, corresponding to mask m and pair (lo, un). The particular mask in the previous layer that is the cause for a particular triple in the current layer need not be carried forward. So, for MDS $sol_{i,j}$ indicates an array of 2^k triples. As for MCD it indicates $O(B_k 2^k)$ triples where $O(2^k)$ triples are associated with each of the $O(B_k 2^k)$ unique pairs of (lo, un). Also, we use $k = O(\log n)$ for MDS while $k = O(\log n)^{\alpha}$, $\alpha < 1$, for MCD, so that the algorithm runs in polynomial time.

Consider the following analysis for MDS. Let mask j be chosen in layer i, it can potentially be combined with every mask ($O(2^k)$ masks) of the previous layer. Thus, potentially $(O(2^k))$ triples need be stored. Further, the total number of triples of the form (un, s, c) is $\Omega(n.2^k)$ because un can potentially assume any of $0 \ldots 2^k - 1$, s is $O(n)$ and c can in fact be exponential in $\frac{n}{k}$. Here we make the following critical observations.

- Let the chosen mask for layer i is j. When all the compatible vertex sets of the previous layer are considered then let the resultant triples for the choice of j in layer i be set S.
- In S for any two triples with the same mask we need only retain the triples with the least size. The other triples cannot lead to an optimal solution.
- If two triples have the same mask and the minimum size then they can be combined into one triple where the respective counts are added.
- Thus, only 2^k triples suffice for a chosen mask for layer i. Which implies 2^{2k} triples suffice $\forall_j$ $cs_{i,j}$. We store the information of only two layers. Thus, the algorithm needs $O(k2^{2k})$ space. This is in addition to the space required by the input graph, which is $O(nk)$. For $k = O(\log n)$, $O(k2^{2k})$ is the dominating term, so the space complexity is $O(k2^{2k})$.
- Thus, for a chosen mask for layer i potentially 2^{2k} triples of previous layer must be processed. That is, for all masks of layer i, a total of 2^{3k} triples must be processed.

- Consider the mask j in layer i and mask l in layer $i-1$. Recall that there are 2^k triples stored corresponding to mask l in layer $i-1$. All the vertices that are covered by the combination of j and l in layer $i-1$ say A and not covered in layer i say B can be computed in $O(k^2)$. This needs to be computed only once. Subsequently, for each triples stored corresponding to l in layer $i-1$ we need only check if the undominated vertices are a subset of B in $O(k)$ time. Thus, $O(k2^k)$ is the dominating term in the time complexity yielding $O(k2^{2k})$ for all masks of the previous layer. So, for all masks of the current layer the time complexity is $O(k2^{3k})$. Thus, the time complexity of the algorithm is $O(\frac{n}{k}k2^{3k}) = O(n2^{3k})$.

Similar constraints hold for MCD. Additionally the induced subgraph of V^* must be a single connected component. Thus, $\forall_p \bigcap V_p^*$ is connected. We carry forward the existing connected components and eventually when the final layer is processed all the components must be connected. The MCD algorithm is explained in detail in Theorem 4 along with time and space complexity analysis.

- $feasible(j)$: For MCD j must be connected. For MDS any j is valid.
- $compatible(j,l)$: the union must dominate all vertices of V_{i-1}. For MCD masks j and l must have at least on edge in between.
- Extension: Performed as per critical observations listed above. The choice of the final layer must ensure that the final layer is dominated.
- Summarize: Let $opt \leftarrow min(\forall_j \forall_d size_{q,j,d})$;$count \leftarrow 0$; $\forall_j \forall_d$ if $(size_{q,j,d} = opt)$ then $count \leftarrow count + ccount_{q,j,d}$; Return $(opt, count)$

The function compatible receives two masks denoting chosen vertices from layers i and $i+1$. If the vertices in layer $i+1$ dominate the so far undominated vertices in layer i then the function returns true. Otherwise, it returns false.

3.5 Algorithm Compatible

Algorithm Compatible

Input: LG_k, j, l, and I. //The function call: $compatible(j,l)$. l: Mask for layer i.
Output: 0 (incompatible) or 1 (compatible). //j: Mask for layer $i+1$. I denotes matrix for $E_{i\ i+1}$.
// $bit_c(i)$ returns true if bit c is set in i else returns false.

Case MIS: // Input: two valid MISs of two adjacent layers
if $independent(j,l)$ **then** // $independent(j,l)$: for any $a,b : bit_a(l)$ and $bit_b(j)$:
  return 1; //if $I[a][b] = 1$ return 0; otherwise return 1; $O(k^2)$ algorithm.
else
  return 0; //$\exists$ a pair of vertices across the layers joined with an edge.
end if

Case MVC: // Input: two VCs of two adjacent layers
if $cover(j,l)$ **then** // $cover(j,l)$: $\forall_{a,b}$ where $I[a][b] = 1$: $bit_a(l) \vee bit_b(j) = 1$
  return 1; // then return 1; otherwise return 0; $O(k^2)$ algorithm.
else
  return 0;
end if

Case MCV: // Input: two masks of two adjacent layers; need not be MCVs of their respective layers.
if $ccover(j,l)$ **then** // $ccover(j,l)$: $\forall_{a,b}$ where $I[a][b] = 1$: $bit_a(l) \vee bit_b(j) = 1$
  return 1; // and $\exists_{c,d} : I[c][d] = 1 \wedge bit_c(l) \wedge bit_d(j)$
else // then return 1; otherwise return 0; $O(k^2)$ algorithm.
  return 0;
end if

Case MDS: // Input: two masks of two adjacent layers,
if $dom(j,l)$ **then** // $dom(j,l)$: $D \leftarrow cs_{i,l} \bigcup cs_{i+1,j} \bigcup Adj(cs_{i,l}) \bigcup Adj(cs_{i+1,j})$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2018 doi:10.20944/preprints201805.0012.v1

return 1; // $i < q-1$: if $V_i \subseteq D$ then return 1; otherwise return 0;
else // $i = q-1$: if $V_i \bigcup V_{i+1} \subseteq D$ then return 1; otherwise return 0;
return 0; //V_i or $V_i \bigcup V_{i+1}$ is not dominated. $O(k^2)$ algorithm.
end if // $Adj(V)$ is the set of all vertices neighboring any vertex in V

Case MCD: // Input: two masks of two adjacent layers,
// $\exists_{c,d} : I[c][d] = 1 \wedge bit_c(l) \wedge bit_d(j)$
if $dom(j,l)$ **then** // $dom(j,l)$: $D \leftarrow cs_{i,l} \bigcup cs_{i+1,j} \bigcup Adj(cs_{i,l}) \bigcup Adj(cs_{i+1,j})$
return 1; // $i < q-1$: if $V_i \subseteq D$ then return 1; otherwise return 0;
else // $i = q-1$: if $V_i \bigcup V_{i+1} \subseteq D$ then return 1; otherwise return 0;
return 0; //V_i or $V_i \bigcup V_{i+1}$ is not dominated. $O(k^2)$ algorithm.
end if // $Adj(V)$ is the set of all vertices neighboring any vertex in V

3.6 Algorithm Generic Optimum

The algorithms for MIS, MVC and MDS problems on $LG_k^{n,q}$ are similar while those for MCV and MCD require additional processing related to connected components. We give a generic dynamic programming based algorithm for both sets of problems. Some specific instances are shown in the Appendix.
Initialization: $\forall i$ $sol_{0i} = sol_{1i} = 0$; $\forall i$ $count_{0i} = count_{1i} = 0$; sol_{ij} : The optimum value (of IS, VC, MCD etc.) up to layer i where the chosen vertices of the layer i are given by the binary value of j. $count_{ij}$: the number of ways the j^{th} mask in layer i yields the corresponding optimum value.

Algorithm Generic Optimum

Input: $LG_k^{n,q}$
Output: The cardinality and corresponding count for the respective problem.
for $(i = 0, ..., 2^k - 1)$ **do**
  if $valid(1,i)$ **then** //for layer 1
    $count_{0i} = 1; sol_{0i} = cardinality(i)$; // For all valid masks set their count
  end if
end for
for $(i = 2, ...q)$ **do** //For layers 2 through maximum
  for $(j = 0, ...2^k - 1)$ **do** //For all masks of current layer
    Compose larger sub-solutions by considering all compatible masks of the
    previous layer and any accompanying information.
  end for//Masks of previous layer
end for//For all layers
The current layer being processed is the final layer.
$best \leftarrow 0; sum \leftarrow 0;$
for $(i = 0, ..., 2^k - 1)$ **do**
  Identify $best$, the cardinality of an optimal solution.
end for
for $(i = 0, ..., 2^k - 1)$ **do**
  Compute sum, the count of optimal solutions.
end for
$return(best, sum)$

4 Correctness and complexity

The Algorithm Generic Optimum when adapted to a specific problem, say MVC, is referred to as Algorithm MVC. The correctness is shown for MIS, MVC and MCD problems. The time complexities for MIS, MVC, and MDS are respectively $O(nk2^{2k})$, $O(nk2^{2k})$ and $O(n2^{3k})$, where $k = O(\log n)$, and the space complexities are $O(nk)$, $O(nk)$ and $O(k2^{2k})$ respectively. For MCV and MCD problems, the time complexity is $O(n^{1+\epsilon})$ for any $\epsilon > 0$, where the number of vertices in a layer is $k = O((\log n)^{\alpha})$ for $\alpha < 1$. The space complexity is $O(nk)$ for

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2018 doi:10.20944/preprints201805.0012.v1

MCD and MCV. The analysis is given for MVC and MCD. The proofs of correctness for the remaining problems are similar. The time complexity for MDS was presented earlier.

Theorem 1. *Algorithm MIS correctly computes the MIS on $LG_k^{n,q}$.*

Proof. Let $G = (V, E)$ be a graph and let V be partitioned into V^1, V^2. Further let I_1, I_2 be the ISs of the graphs induced by V^1, V^2 respectively and let $I = I_1 \bigcup I_2$. If you consider the cut $C = (I_1, I_2)$ on I where E^C is the set of edges crossing the cut then it follows that I is an IS of G if $E^C = \phi$. Further the cardinality of an MIS of G is $max(\forall_{E^C=\phi} \mid I_1 \mid + \mid I_2 \mid)$. It is possible that either $\mid I_1 \mid = 0$ or $\mid I_2 \mid = 0$.

Let G be $LG_k^{n,q}$. Let G_1 be the subgraph of $LG_k^{n,q}$ induced by $V^1 = \bigcup_{j=1}^{i} V_j$ and let G_2 be the subgraph of $LG_k^{n,q}$ induced by $V^2 = \bigcup_{j=i+1}^{q} V_j$. Consider the IS of G. Let I_1 and I_2 be the independent sets of G_1 and G_2 respectively. Let the set of edges crossing the cut $C = (I_1, I_2)$ be E^C. It follows that $I = I_1 \bigcup I_2$ is an IS of G with cardinality $\mid I_1 \mid + \mid I_2 \mid$ when there is no edge crossing C. Only edges in $E_{i\ i+1}$ need to be considered. Thus, the cardinality of an MIS of $LG_k^{n,q} = max(\forall_{E^C=\phi} \mid M_1 \mid + \mid M_2 \mid)$. When the last layer is processed the cardinalities of ISs of subgraphs induced by V and $V - V_q$ both are known. Further, these ISs have maximum cardinalities with respect to the vertices chosen in layers $q-1$ and q respectively. The theorem follows. Likewise, $count_{ij}$ gives the number of ways an independent set of maximum cardinality that can be formed when the vertices chosen in the layer i are given by j. Thus, $count_{qj}$ corresponding to the maximum value of sol_{qj} yields the total number of MISs. $\square$

Theorem 2. *Algorithm MVC correctly computes the MVC on $LG_k^{n,q}$.*

Proof. Consider the structure of MVC on $LG_k^{n,q}$. Let G_1 be the subgraph of $LG_k^{n,q}$ induced by $V^1 = \bigcup_{j=1}^{i} V_j$ and let G_2 be the subgraph of $LG_k^{n,q}$ induced by $V^2 = \bigcup_{j=i+1}^{q} V_j$. Consider a VC of G. Let M_1 and M_2 be the vertex covers of G_1 and G_2 respectively. Let the set of edges crossing the cut $C = (M_1, M_2)$ be E^C. It follows that the cardinality of a VC of G is $\mid M_1 \mid + \mid M_2 \mid$ when every edge crossing C is covered by either M_1 or M_2. Note that the only edges from $E_{i\ i+1} = E^C$ can go across the cut. Thus, the cardinality of MVC of $LG_k^{n,q} = min(\mid M_1 \mid + \mid M_2 \mid)$ for any such cut. When the last layer is processed this property is ensured. The theorem follows. Similarly, $count_{ij}$ gives the number of ways an vertex cover of minimum cardinality that can be formed when the vertices chosen in the layer i are given by j. Thus, $count_{qj}$ corresponding to the minimum value of sol_{qj} yields the total number of MVCs. $\square$

Theorem 3. *Algorithm MVC on $LG_k^{n,q}$ runs in polynomial time in n when $k = O(\log n)$. The space required is $O(nk)$.*

Proof. We presume that I_i, the 0-1 adjacency matrix for the subgraph induced by $V_i \bigcup V_{i+1}$ where the edges are restricted to $E_{i\ i+1}$ is given. Likewise, we assume that the 0-1 adjacency matrix M_i for each of G_i are given. Recall that $LG_k^{n,q}$ was formed from $G_1, G_2, \dots G_q$. For a linear graph, I_i is just a $k-$dimensional vector where if bit j is set then there is an edge between V_{ij} and $V_{i+1\ j}$.

- The initialization step requires $O(2^k)$ time.
- Given a mask for layer i it can be determined if it is a valid VC in $O(k^2)$ time with M_i. That is, for any two $M_i[a][b]$ that is set the mask should have either bit a or bit b set.
- Given two masks $mask1, mask2$ for layers $i, i+1$ respectively and I_i it can be directly determined if their union is a VC of a subgraph induced by $\bigcup_i^{i+1} V_j$ of $LG_k^{n,q}$ in $O(k^2)$ time.
- In order to determine the MVC up to layer i whose mask is j; j must be checked for compatibility with all masks of the previous layer. Thus, $O(k^2 2^k)$ time is required. For all masks of the current layer $O(k^2 2^{2k})$ time is required. For all layers, the time required is maximized when each layer has k vertices yielding $O(\frac{n}{k} k^2 2^{2k}) = O(nk2^{2k})$ time.

The time complexity is clearly exponential in k; however, if $k = O(1)$ the time complexity is $O(n)$. The time complexity remains polynomial when $k = O(\log n)$; specifically $O(n^3 \log n)$ when $k = \log n$. The additional space required is $O(k2^k)$ because for two layers we store 4.2^k mask and count variables each of size k. The space required is $O(nk)$ for storing the graph and an additional space of $O(k2^k)$ that is needed by the algorithm. When $k = O(\log n)$ the space complexity is $O(nk)$. □

Lemma 1. *Let* $0 \leq \alpha < 1.0$ *where* $\alpha \in R^+$. *If* $x = (\log n)^\alpha$ *then* $x! = O(n^\epsilon)$, *for any* $\epsilon > 0$.

Proof.

Let $f(n) = (\log n)^\alpha$, $\alpha < 1$

Let $h(n) = n^\epsilon, \epsilon > 0$

Now, consider $f(n)!$

$$\Rightarrow f(n)! = (\log n)^\alpha!$$

Taking log on both sides,

$$\begin{aligned}
\log(\lceil f(n)! \rceil) &= \log 1 + \log 2 + \cdots + \log(\lceil (\log n)^\alpha \rceil) \\
&= \sum_{x=1}^{\lceil (\log n)^\alpha \rceil} \log x \\
&\approx \int_1^{(\log n)^\alpha} \log x dx \\
&= [x \log x - x]_1^{(\log n)^\alpha} \\
&= \alpha (\log n)^\alpha \log \log n - (\log n)^\alpha + 1 \\
&\approx (\log n)^\alpha (\alpha \log \log n - 1) \\
&= g(n), \text{ say}
\end{aligned}$$

Assume that,

$$\begin{aligned}
g(n) &= O(\epsilon \log n) \\
\Rightarrow (\log n)^\alpha (\alpha \log \log n - 1) &\leq c\epsilon \log n \\
\Rightarrow \frac{(\alpha \log \log n - 1)}{(\log n)^{1-\alpha}} &\leq c\epsilon
\end{aligned}$$

Let $1 - \alpha = \beta, \beta > 0$ and $c\epsilon = \gamma$

$$\Rightarrow \frac{(\alpha \log \log n - 1)}{(\log n)^\beta} \leq \gamma$$

Let $logn = x$

$$\begin{aligned}
\Rightarrow \frac{(\alpha \log x - 1)}{(x)^\beta} &\leq \gamma \\
\Rightarrow (\alpha \log x - 1) &\leq \gamma (x)^\beta
\end{aligned}$$

We know that logarithmic functions grow slower than polynomial functions.

So, the above inequality holds which means our assumption was correct.

$$\begin{aligned}
\Rightarrow (\log n)^\alpha (\alpha \log \log n - 1) &= O(\epsilon \log n) \\
\therefore ((\log n)^\alpha !) &= O(n^\epsilon) \qquad \alpha < 1, \epsilon > 0
\end{aligned}$$

Hence, proved. □

Lemma 2. *If* $x = (\log n)$ *then* $x!$ *is quasi-polynomial and* $(x!) = O(n^{\log \log n})$.

Proof.
Let $f(n) = \log n$

$$\Rightarrow f(n!) = \log(n!)$$

From Stirling's Approximation, we have

$$\Rightarrow \log(n!) = \theta(n \log n)$$
$$\Rightarrow (\log(\log n)!) = \theta(\log n \log\log n)$$
$$\Rightarrow ((\log n)!) = 2^{\theta(\log n \log\log n)}$$

This can be written as,

$$((\log n)!) = n^{\log\log n}$$
$$\Rightarrow (f(n)!) = n^{\log\log n}$$

The above result is quasi-polynomial.

Hence, proved. □

Lemma 3. *If $k = \Theta((\log n)^{1+\epsilon})$, for any $\epsilon > 0$ then Algorithm MIS, Algorithm MVC and Algorithm MDS run in quasi-polynomial time.*

Proof. The time complexities of all these algorithms can be written as $O(f(n)g(k)2^{ck})$ where $f(n) = \Theta(n)$, $g(k) = O(k)$ and $c = O(1)$. Thus, when $k = \Theta((\log n)^{1+\epsilon})$ for $\epsilon > 0$ the complexities for all the algorithms will be quasi-polynomial. □

Theorem 4. *Algorithm MCD correctly computes the cardinality of a connected minimum dominating set for LG_k with a time complexity of $O(n^{1+\epsilon})$, for any $\epsilon > 0$ when $k = O(\log n)^\alpha$ and $\alpha < 1$. The space complexity of the algorithm is $O(nk)$.*

Proof: First, we show that the algorithm correctly computes the cardinality of a connected minimum dominating set. Consider the structure of CDS on a connected graph G. Let V be arbitrarily partitioned into V^1, V^2 where both $| V^1 | > 0$ and $| V^2 | > 0$. Let G_1 be the subgraph of G induced by V^1 and let G_2 be the subgraph of G induced by V^2. Let $M_1 \subseteq V^1$ and $M_2 \subseteq V^2$ be DSs of G_1 and G_2. Let C be the cut (M_1, M_2) and let E^C be the edges that cross this cut. Clearly $M = M_1 \bigcup M_2$ is DS for G. Further, M is a CDS for G if $| E^C | > 0$ and M forms a connected component in G. For a given partition V^1, V^2 of V, M is a MCD if it minimizes $| M_1 | + | M_2 |$ where M forms a connected component in G.

Let G be a $LG_k^{n,q}$ in particular let G be a $CLG_k^{n,q}$ let $V^1 = \bigcup_{j=1}^{q-1} V_j$ and $V^2 = V_q$. Let G_1 be the subgraph of G induced by V^1 and let G_2 be the subgraph of G induced by V^2. Let $M_1 \subseteq V^1$ and $M_2 \subseteq V^2$ be DSs of G_1 and G_2 respectively. Let C be the cut (M_1, M_2) and let E^C be the edges that cross this cut. Note that $E^C = E_{q-1\ q}$. When the algorithm processes layer q it chooses $M = M_1 \bigcup M_2$ such that $| M_1 | + | M_2 |$ is minimized where M forms a connected component in G. Thus, the theorem follows. Similarly, $count_{ij}$ gives the number of ways a CDS of minimum cardinality can be formed when the vertices chosen in the layer i are given by j. Thus, $\forall_j \Sigma count_{qj}$ corresponding to the minimum value of $\forall_j sol_{qj}$ yields the total number of MDSs.

Time complexity of the algorithm is analyzed below. We presume that similar prerequisites are provided as in Theorem 3 earlier. The steps are as below.

- A global structure *sol* consisting of sol_0 and sol_1 corresponding to the previous and current layers is maintained for the whole algorithm. The final solution for the problem can be determined just by using information from sol_0 and sol_1. This structure is maintained for the whole algorithm and not for every layer.

- sol_0 and sol_1 each consist of a maximum of $B_k 2^k$ triples of the form (lo, un, r). This corresponding to a maximum of B_k (k^{th} Bell number) component layouts (lo), 2^k masks, un, of undominated vertices of the current layer and a maximum 2^k triples, r of the form (m, s, c) for every unique pair (lo, un). Here, m: mask of the current layer that produced the respective (component layout, undominated vertices) pair, s minimum cardinality of the sub-solution corresponding to mask m and pair (lo, un), c: count of s corresponding to mask m and pair (lo, un).
- Throughout the algorithm, sol_0 and sol_1 are maintained by clearing sol_0 when the current layer is processed and using the information of sol_1 as sol_0 for the next layer.
- sol_0 is initialized with the triple (lo, un, r) corresponding to 2^k masks of the first layer. The initialization takes $O(k^2 2^k)$.
- A candidate sub-solution for layers $1 \ldots i$ induces connected components in layer i that are defined in terms of vertices of layer i. We call this as the component layout.
- Number of component layouts is upper bounded by Bell Number(k) or B_k, the number of ways of partitioning k vertices of a layer. Here $k = f(n)$, $f(n) = O(\log n)^\alpha$, $\alpha < 1$. $B_k = O(f(n)!)$. From Lemma 1, we know that $f(n)! = O(n^\epsilon)$, for any $\epsilon > 0$.
- A mask j of the current layer can be combined with a component layout for mask l of the previous layer to form a new component layout for the current layer. With the same mask l, j can form a new mask corresponding to the undominated vertices of the current layer.
- Every such unique pair of $(lo,\ un)$, where lo is component layout and un is mask of undominated vertices, is maintained and a list of triples r consisting of triples of the form (m, s, c) is associated with it. Here m is the current layer mask, s is the minimum cardinality of the sub-solution corresponding to m and c is the count of s. The number of such tuples (lo, un, r) is upper bounded by $B_k 2^2 k$, where $B_k 2^k$ is the possible number of unique pairs of (lo, un) and 2^k is the possible number of triples that can exist for each pair.
- Starting from the i-th layer, $i > 1$, every 2^k mask of the current layer and the tuple values from the previous layer are used to generate the tuples for the current layer.
- For a unique pair (lo, un) of the previous layer, if mask j dominates the undominated vertices of mask un and forms a connected component with the layout lo, then we consider that a sub-solution using mask j is feasible. Here, a mask j and a component layout lo are considered to form a connected component if every component in lo has at least one edge to a node in mask j. Each such check takes $O(k^2)$ time. So, the total time to determine if a sub-solution with mask j is feasible is $O(k^2)$.
- If a mask j is feasible to give a sub-solution, then it is combined with the component layout lo of the previous layer to form a new component layout for the current layer corresponding to mask j. This is performed using a DFS which takes $O(k^2)$ for the given input matrix.
- Mask j is then combined with mask l of the previous layer corresponding to the pair (lo, un), that is under consideration, to form a mask for the current layer vertices that are not dominated by j or l. This takes $O(k^2)$ time.
- Using the mask j of the current layer and minimum cardinality s for the pair (lo, un) of the previous layer, the new cardinality for the sub-solution is computed.
- The count of the new cardinality will be same as that of c of the (lo, un) pair for the previous layer.
- This new pair of component layout and undominated mask computed for mask j of the current layer is checked with the existing pairs of the current layer to determine if it is unique or not. We maintain the structure of the tuples such that an entry can be accessed in $O(1)$ time, indexed by the pair (lo, un) and the corresponding mask m for the previous and the current layer.
- If it is unique, the tuple value consisting of the newly computed (lo, un) pair and its corresponding triple consisting of the mask j, respective cardinality and the count are added as a new tuple for the current layer.
- Consider that the current mask j produces the new pair (lo, un) with values $s = s_x$ and $c = c_x$. If the new pair is not unique then there are three cases. Consider the existing entry of the (lo, un) pair and the corresponding j to have values $s = s_y$ and $c = c_y$.
  (a) if $s_y = s_x$ then $c_y \leftarrow c_y + c_x$;
  (b) if $s_y > s_x$ then $s_y \leftarrow s_x; c_y \leftarrow c_x$;
  (c) if $s_y < s_x$ then no update is required.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2018 doi:10.20944/preprints201805.0012.v1

- The above procedure is performed till the last layer where the final solution is computed from the current layer information corresponding to the last layer. Of all the $B_k 2^k$ pairs for the current layer, a solution is considered to be feasible if the mask for the undominated vertices for any of the B_k component layouts is 0, as this would mean all the vertices are dominated. The cardinality of MCD is the minimum value among all the feasible solutions. The count is then computed by considering each feasible entry with the minimum cardinality computed above and adding its corresponding count.
- Thus, the solution and the corresponding count of optimal solutions for MCD problem are computed.

For the whole algorithm, we maintain the global structure as mentioned above. It consists of a maximum of $O(B_k 2^k)$ entries corresponding to unique pairs of (lo, un) and another 2^k triples for each such pair. We maintain this information for only the previous and the current layers. So, the space used by the data structure is $O(B_k 2^{2k})$. This can be shown to be equal to $O(n^\epsilon)$, for any $\epsilon > 0$, based on the proof for Lemma 1. This space requirement is in addition to the space required by the input graph which is $O(nk)$. For $k = O((\log n)^\alpha)$, $O(nk)$ is the dominating term compared to $O(n^\epsilon)$. So, the space complexity is $O(nk)$. The following is the proof for time complexity of the algorithm.

First, we derive an expression for the runtime of the algorithm. The initialization using the first layer takes $O(k^2 2^k)$ time. For each layer after the first, the 2^k masks of the current layer is combined with the $B_k 2^k$ pairs of the previous layer. For each pair, a current layer mask is combined with a maximum of 2^k masks of the previous layer that generated this pair. Checking the feasibility of a mask of the current layer takes $O(k^2)$ time. Computing the new component layout and the new undominated mask takes $O(k^2)$ time each. The undominated mask is calculated for 2^k masks of the previous layer for each mask of the current layer. Accessing and updating an entry takes $O(1)$ time as mentioned above. This is done for $O(n/k)$ layers. So, the time complexity expression can be written as,

$$\begin{aligned} T &= O(\frac{n}{k} 2^k B_k 2^k (k^2 + 2^k k^2)) \\ &= O(\frac{n}{k} k! 2^{2k} (2^k k^2)) \quad \because (B_k = O(k!), \text{Lemma } 1) \\ &= O(nk 2^{3k} k!) \qquad (1) \end{aligned}$$

If $k = O(1)$, the time complexity becomes $T = O(n)$. If we assume the worst case number of nodes in each layer, i.e. $k = f(n)$ then the corresponding time complexity is $T = O(n^{1+\epsilon})$ as shown below.

Let $f(n) = (\log n)^\alpha \quad \alpha < 1$

Let $h(n) = n^\gamma \quad \gamma > 0$

From Lemma 1 we have

$$\begin{aligned} x! = O(n^\gamma) &\text{ for some } \gamma > 0, \text{ where } x = (\log n)^\alpha \\ &\Rightarrow f(n)! = O(n^\gamma) = O(h(n)) \end{aligned}$$

The running time of the algorithm, is given by

$$\begin{aligned} T &= O(nk 2^{3k} f(n)!) \\ &\le cn * k * 2^{3k} * h(n) \\ &\le cn^{1+\gamma} * (\log n)^\alpha * 2^{3(\log n)^\alpha} \qquad (1) \quad (\because h(n) = n^\gamma) \end{aligned}$$

Consider $F(n) = (\log n)^\alpha * 2^{3(\log n)^\alpha}$

Let $g_1(n) = n^\delta$ and $g_2(n) = n^\mu \qquad \delta > 0, \mu > 0$

We know that logarithmic functions grow slower than polynomial functions.

$$\Rightarrow (\log n)^\alpha \leq cg_1(n)$$
$$\Rightarrow (\log n)^\alpha = O(n^\delta)$$

Now, we claim that $2^{3(logn)^\alpha} \leq cg_2(n)$ for some $\alpha < 1$, a positive real number c and $n > n_0$, where n_0 is some positive integer

Consider the following proof.

Taking log on both sides, we get

$$\log(2^{3(logn)^\alpha}) \leq \log(cg_2(n))$$
$$\Rightarrow 3(\log n)^\alpha \leq \log c + \log g_2(n)$$
$$\Rightarrow 3(\log n)^\alpha \leq \mu \log n \qquad (\because g_2(n) = n^\mu)$$

Since $\alpha < 1$, $(\log n)^\alpha < \log n$

$$\Rightarrow 3(\log n)^\alpha = O(\mu \log n)$$
$$\Rightarrow 2^{3(logn)^\alpha} \leq cn^\mu$$

Hence, we proved our claim.

$$\therefore 2^{3(logn)^\alpha} = O(n^\mu)$$

From above we have,

$$F(n) = (\log n)^\alpha * 2^{3(\log n)^\alpha}$$
$$\Rightarrow F(n) \leq cn^\delta * n^\mu$$
$$\Rightarrow F(n) \leq cn^{\delta+\mu}$$
$$\therefore F(n) = O(n^{\delta+\mu}) \qquad \delta > 0, \mu > 0$$

From (1), we get

$$T \leq cn^{1+\gamma} * n^{\delta+\mu}$$
$$\leq cn^{1+\gamma+\delta+\mu}$$

We can write it as,

$$T \leq cn^{1+\epsilon} \qquad \epsilon = \gamma + \delta + \mu$$

By arbitrarily taking small values for μ, δ and γ, ϵ can be made a small value such that $\epsilon > 0$

$$\therefore T = O(n^{1+\epsilon}) \qquad \epsilon > 0$$

Hence, proved. □

Theorem 5. *Algorithm MCV correctly computes a connected VC of minimum cardinality for LG_k with a time complexity of $O(n^{1+\epsilon})$, for any $\epsilon > 0$ when $k = O(\log n)^\alpha$ and $\alpha < 1$. The space complexity is $O(nk)$.*

Proof. MCV algorithm is similar to MCD algorithm. A mask j of layer i must be a valid VC for layer i. The check takes $O(k^2)$ time additionally though the total time complexity can be proved to be same as that of MCD. So, the proofs of correctness and time complexity follow from the proofs for the same of the MCD algorithm. Hence, the time complexity is $O(n^{1+\epsilon})$ for any $\epsilon > 0$ when the number of vertices in each layer is k, where $k = O((\log n)^\alpha)$ and $\alpha < 1$. Similarly, the space complexity can be shown to be $O(nk)$. □

Lemma 2 proves that $(\log n)!$ is quasi-polynomial. Using this, we can show that if $k = \Theta(\log n)$ for MCV and MCD problems then the running time of algorithm is quasi-polynomial. Proving this is quite straightforward. By substituting $(\log n)!$ for $k!$ in equation (1) in Theorem 4, we get a product of quasi-polynomial factor and a polynomial factor. Thus, the time complexity is quasi-polynomial.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2018 doi:10.20944/preprints201805.0012.v1

4.1 Minor Enhancements

The current layer requires the information only from the previous layer. So, only the variables of the current layer i and the previous layer $i-1$ are maintained. In the pseudocode shown for all algorithms, for simplicity, the variables of current layer are stored at index 1 and the previous layer at index 0 of the data structure *sol*. When the current layer i is completely processed the variables from index 1 overwrite the corresponding variables in index 0. This can be avoided by alternating the index of current layer between indices 0 and 1 thereby reducing the execution time by a factor of $O(1)$.

We generate the optimum cardinalities for each of the problems by using minimal additional space. For example, Algorithm MVC employs only $O(k2^k)$ space in addition to the space required by the graph. If for each mask in each layer we store a best compatible mask from its previous layer then we can generate a solution. There are $O(n/k)$ layers each having $O(2^k)$ k-bit masks. This requires $O(n2^k)$ space instead of $O(k2^k)$ space. However, if we want to generate all solutions then for each mask of a given layer we need to store all compatible masks of its previous layer that yield the optimum value requiring $O(n2^{2k})$ space.

4.2 Cyclic Layered Graphs

A *cyclic layered graph* is a layered graph with one additional feature. In addition to the edges that are allowed for a layered graph, in a cyclic layered graph there can be edges between the first and the last layer. The problems that are solved on a layered graph in this article can be solved on a cyclic layered graph also by modifying the solution in the following manner. Along with every candidate sub-solution that is stored at a layer i the corresponding masks of layer 1 that can lead to the solution are also stored. Note that at most 2^k such masks exist. When the last layer is processed when choosing the mask for the last layer the edges between the vertices of the last and first layers are considered. This imposes an additional constraint on what masks are feasible for the last layer. These additional tasks that must be performed for cyclic layered graphs do not change the asymptotic time and space complexities of the existing algorithms for layered graphs.

5 Conclusions

A novel graph class called layered graph is defined. It includes a subset of bipartite graphs and a subset of trees on n vertices and can have exponential number of cycles. The typical restrictions on graph classes that admit polynomial time solutions for hard problems like bipartiteness, planarity, acyclicity are not applicable for this class. The known NP-complete problems on these graphs are shown to be in class P when layer size is $O(\log \mid V \mid)$ for MIS, MVC and MDS, and $O((\log \mid V \mid)^{\alpha})$, where $\alpha < 1$, for MCV and MCD. We also compute the count of the corresponding optimal solutions.

Acknowledgments: Authors thank Amma for her guidance. This article is based on technical report UTDCS-02-17. Sriram K helped with the literature survey.

Author Contributions: B.C. conceived the design of graphs and solution; performed analysis of the algorithms. S.B., S.S. and K.P. helped in testing the algorithms. K.P. worked on implementation and analysis of the algorithms. S.S. worked on extending the implementations and analysis. S.B. tested the implementations. B.C. and S.B. worked on the proofs for the algorithms and wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

A Appendix

The generic algorithm was presented earlier. Here, we present a detailed algorithm each for MIS and MVC. A relatively high-level description for the MCD algorithm is mentioned.

A.1 Algorithm MIS

Input: $LG_k^{n,q}$
Output: The cardinality of MIS and the count of the maximum independent sets.
Initialization: $\forall i\ sol_{0i} = sol_{1i} = 0$;
$\forall i\ count_{0i} = count_{1i} = 0$;
$//sol_{ij}$: The maximum value of an independent set up to layer i where the chosen
//vertices of the layer i are given by the binary value of j.
$//count_{ij}$: the number of ways the j^{th} mask in layer i yields the corresponding maximum value.
$//valid(i, j)$ is a boolean function that returns true if the vertex assignment corresponding to
//the binary value of j in layer i forms an IS. Otherwise it returns false.
$//\wedge$ is the bitwise AND operator.
$//cardinality(j)$ is the number of bits that are set in the binary representation of j.
// For each sol_{ij} one k-bit variable that remembers the mask of the layer $i-1$ that
// yielded sol_{ij} will help in constructing MISs. Union of such masks (1/layer) is an MIS.

for $(i = 0, ..., 2^k - 1)$ **do**
  if $valid(1, i)$ **then** // for layer 1
    $count_{0i} = 1; sol_{0i} = cardinality(i)$; // No. of valid ISs of layer 1
  end if
end for
for $(p = 2, ...q)$ **do** //For layers 2 through maximum
  for $(j = 0, ...2^k - 1)$ **do** //For all masks of current layer
    if $valid(p, j)$ **then** $//j$ is valid
      $size \leftarrow 0$
      for $(l = 0, ..., 2^k - 1)$ **do** //Masks of previous layer
        if $((count_{0l} > 0) \wedge (compatible(j, l)))$ **then** $//sol_{0l} = 0 \rightarrow$Invalid IS
          if $(cardinality(j) + sol_{0l} \geq size)$ **then** // Better IS for the current mask
            if $(cardinality(j) + sol_{0l} > size)$ **then**
              $size = cardinality(j) + sol_{0l}; count_{0l} = count_{0l} + 1$
            end if
            $count_{0l} \leftarrow count_{0l} + 1$
          end if
        end if
      end for//Masks of previous layer
      for $(l = 0, ..., 2^k - 1)$ **do** //Masks of previous layer
        if $(size = cardinality(j) + sol_{0l})$ **then** //Instance of max
          $count_{1j} \leftarrow count_{1j} + count_{0l}$; // Count corr. to max wrt mask=j
        end if
      end for//Masks of previous layer
      sol_{1j} $size$
    end if// j is valid
  end for//For all masks of current layer
  $\forall x\ count_{0x} \leftarrow count_{1x}; sol_{0x} \leftarrow sol_{1x}; count_{1x}$ $sol_{1x} \leftarrow 0$;
end for//For layers 2 through maximum
$best \leftarrow 0; sum \leftarrow 0$;
for $(i = 0, ..., 2^k - 1)$ **do**
  if $sol_{0i} > best$ **then** //Get the max value of $\forall_i sol_{pi}$
    $best = sol_{0i}$;
  end if
end for
for $(i = 0, ..., 2^k - 1)$ **do**
  if $sol_0 i = best$ **then** //Corr. to the best value of $MIS(LG_k^{n,q})$
    $sum \leftarrow sum + count_{1i}$; //Get the count of MISs

end if
end for
$return(best, sum)$ //MIS cardinality and the count of such MISs

A.2 Algorithm MVC

Input: $LG_k^{n,q}$
Output: The cardinality and the count for the resp. problem.
//sol_{ij} : The minimum value of a vertex cover up to layer i where the chosen
//vertices of the layer i are given by the binary value of j.
// $valid(i, j)$ is a boolean function that returns true if the vertex assignment corresponding to
//the binary value of j in layer i forms a VC. Otherwise it returns false.
//$count_{ij}$: the number of ways the j^{th} mask in layer i yields the corresponding minimum value.
//$cardinality(j)$ is the number of bits that are set in the binary representation of j.
for $(i = 0, ..., 2^k - 1)$ **do**
  if $valid(1, i)$ **then** //for layer 1
    $count_{0i} = 1; sol_{0i} = -1;$ // No. of valid VCs of layer 1
  end if
end for
for $(p = 2, ...q)$ **do** //For layers 2 through maximum
  for $(j = 0, ...2^k - 1)$ **do** //For all masks of current layer
    if $valid(p, j)$ **then** //j is valid
      $size \leftarrow (i + 1) * k$
      for $(l = 0, ..., 2^k - 1)$ **do** //Masks of previous layer
        if $((count_{0l} > 0) \wedge (compatible(j, l)))$ **then** //$sol_{0l} = 0$ →Invalid VC
          if $(cardinality(j) + sol_{0l} \le size)$ **then** // Better VC for the current mask
            $size = cardinality(j) + sol_{0l};$
            if $(cardinality(j) + sol_{0l} = size$ **then** $count_{1j} \leftarrow count_{1j} + count_{0l};$
            else $count_{1j} \leftarrow count_{0l}; sol_{1j} \leftarrow size)$
            end if
          end if
        end if
        sol_{1j} $size$
      end for//Masks of previous layer
      for $(l = 0, ..., 2^k - 1)$ **do** //Masks of previous layer
        if $(size = cardinality(j) + sol_{0l};)$ **then** //Instance of max
          $count_{1j} \leftarrow count_{1j} + count_{0l};$ // Count corr. to max wrt mask=j
        end if
      end for//Masks of previous layer
    end if// j is valid
  end for//For all masks of current layer
  $\forall x\ count_{0x} \leftarrow count_{1x}; sol_{0x} \leftarrow sol_{1x}; count_{1x}$ $sol_{1x} \leftarrow 0;$
end for//For layers 2 through maximum
$best \leftarrow \inf; sum \leftarrow 0;$
for $(i = 0, ..., 2^k - 1)$ **do**
  if $sol_{1i} < best$ **then** //Get the max value of $\forall_i sol_{pi}$
    $best = sol_{1i};$
  end if
end for
for $(i = 0, ..., 2^k - 1)$ **do**
  if $sol_{1i} = best$ **then** //Corr. to the best value of $MVC(LG_k^{n,q})$
    $sum \leftarrow sum + count_{1i};$ //Get the count of MVCs

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 May 2018 doi:10.20944/preprints201805.0012.v1

    end if
end for
$return(best, sum)$ //MVC cardinality and the count of such MVCs

A.3 Algorithm MCD

// A brief outline of the MCD algorithm
// The algorithm maintains a global structure, sol which consists of sol_0 and sol_1 corresponding to the previous and current layers. sol_1 consists of $B_k 2^k$ triples of the form (lo, un, r). This corresponding to a maximum of B_k (k^{th} Bell number) component layouts, 2^k masks of undominated vertices of the current layer and a maximum 2^k triples, r, of the form (m, s, c) for every unique pair (lo, un). lo: is a component layout, un: mask of undominated vertices of the current layer, r: triples of the form (m, s, c) where m: mask of the current layer that produced the respective (component layout, undominated vertices) pair, s minimum cardinality of the sub-solution corresponding to mask m and pair (lo, un), c: count of s corresponding to mask m and pair (lo, un). All unique pairs of (component layout, undominated vertices) need not yield a (sub)solution. sol_0 consists of the same information for the previous layer.
// Mask i refers to the mask of the vertices of current layer that can yield a sub-solution (with minimum value of s for some pair (lo, un)). The component layout refers to the list of the connected components of the current layer vertices (which can form a component employing some vertices from the previous layers). It is determined by the respective mask, and the corr. sub-solution from the previous layer whose combination yields the minimum value of s for some pair (lo, un) .
// If the current layer mask j produces (lo, un) pair with values $s = s_x$ and $c = c_x$ then we have two cases (i) There is no entry corr. (lo, un) and j. Here we just add (lo, un) and j with corr. s and c. (ii) There is an entry corr. (lo, un) and j with $s = s_y$ and $c = c_y$ then
(a) if $s_y = s_x$ then $c_y \leftarrow c_y + c_x$;
(b) if $s_y > s_x$ then $s_y \leftarrow s_x; c_y \leftarrow c_x$;
(c) if $s_y < s_x$ then no update is required.

for $(i = 0, ..., 2^k - 1)$ **do** //for layer 1
    Initialize $sol_{0i} \leftarrow (lo, un, r)$; $r \leftarrow (m, cardinality(i), 1)$
end for
for $(p = 2,...,q)$ **do** //for layers 2 through q
    for $(j = 0,...,2^k - 1)$ **do** //j: current layer mask
        for $(v = 0,...,$no. of (lo, un) pairs$)$ **do** // Of sol_0
            If j dominates the nodes of un of sol_{0v} then continue.
            If every component of lo of sol_{0v} has an edge to any node in j then continue.
            Compute the new component layout using mask j and layout lo.
            for $(x = 0,...,$ size of r corr. $(l, u))$ **do** // No. of triples in r
                Compute the new mask of the undominated vertices using masks j
                of current layer and m corresponding to x-th triple of sol_{0v}.
                Compute the minimum cardinality of the sub-solution corresponding to
                mask j for the current layer using s of the x-th triple of sol_{0v}.
                The count of the newly computed sub-solution will be equal to c
                of the x-th triple corresponding to mask m.
                    If component layout lo and the undominated mask un that are computed corr. j
                    do not exist in sol_1, then insert the tuple (lo, un, r) , into sol_1
                    where r has a single triple whose mask is j.
                    If the (lo, un) pair was already generated by j and a previous mask of the
                    previous layer, then if needed update the minimum cardinality
                    and the corresponding count.
                    Else, insert the new triple (m, s, c) for the corresponding (l, u) pair in sol_1.
            end for
        end for

    end for
end for
$best \leftarrow \text{inf}$, $sum \leftarrow 0$
Consider the values of sol_1 in layer q.
Here the component layout can be ignored as, an entry would mean that it forms a connected component.
For a solution to be considered, the undominated mask must be 0.
for ($i = 0$,...,no. of (lo, un) pairs) **do** // for sol_1
    Identify $best$, the cardinality of the optimal solution.
end for
for ($i = 0$,...,no. of (lo, un) pairs) **do** // size of sol_1
    Compute sum, the count of such optimal solutions.
end for
$return(best, sum)$

References

1. Harary, Frank. *Graph theory* **1969**.
2. Gavril, F. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum vertex cover of a chordal graph. *Siam J. Computing* **1972**, *1.2*, 180-187.
3. Gavril, F. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Netw.* **1973**, *3.3*, 261-273.
4. Cockayne, E.; Goodman, S.; Hedetniemi, S. A linear algorithm for the domination number of a tree. *Inf. Process. Lett.* **1975**, *4.2*, 41-44.
5. Tarjan, R. E.; Trojanowski, A. E. Finding a maximum independent set. *Siam J. on Comput.* **1977**, *6.3*, 537-546.
6. Golumbic, M.C. The complexity of comparability graph recognition and coloring. *Comput.* **1977**, *18.3*, 199-208.
7. Minty, G. J. On maximal independent sets of vertices in claw-free graphs. *J. of Comb. Theory* **1980**, *Series B 28.3*, 284-304.
8. Gupta, U. I.; Lee, D.T.; Leung, J.T. Efficient algorithms for interval graphs and circular arc graphs. *Netw.* **1982**, *12.4*, 459-467.
9. Robson, J. M. Algorithms for maximum independent sets. *J. of Algorithms* **1986**, *7.3*, 425-440.
10. Chen, G.H.; Kuo M. T.; Sheu J. P. An optimal time algorithm for finding a maximum weight independent set in a tree. *BIT Numerical Mathematics* **1988**, *28.3*, 353-356.
11. Gavril, F. Maximum weight independent sets and cliques in intersection graphs of filaments. *Inf. Process. Lett.* **2000**, *73.5-6* 181-188.
12. Hsiao, J. Y.; Tang, C. Y.; Chang, R. S. An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. *Inf. Process. Lett.* **1992**, *43.5*, 229-235.
13. Lozin, V.V.; Milanic, M. On the Maximum Independent Set Problem in Subclasses of Planar Graphs. *J. of Graph Algorithms and Appl.* **2010**, *14.2*, 269-286.
14. Gouveia, L.; Simonetti, L.; Uchoa, E. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. *Mathematical Program.* **2011**, *128.1*, 123-148.
15. Garey, M R.; Johnson, D.S. Computers and Intractability: A Guide to the Theory of NP-completeness, **1979**.
16. Bhadrachalam Chitturi. Layered graphs: a class that admits polynomial time solutions for some hard problems. *arXiv:1705.06425*, **2017**.
17. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of computer computations*, Springer: Boston, USA, 1972, pp. 85-103.
18. Garey, M.R.; Johnson, D. S. The rectilinear Steiner tree problem is NP-complete. *Siam J. Appl. Mathematics* **1977**, 826-834.
19. Li, Y.; Yang, Z.; Wang, W. Complexity and algorithms for the connected vertex cover problem in 4-regular graphs. *Appl. Mathematics and Comput.* **2017**, *301*, 107-114.
20. Bondy, J. A.; Murty, U. S. R. *Graph theory with application*, Macmillan: London, 1976.
21. Konig, D. Graphen und matrizen. *Mat. Fiz. Lapok* **1931**, *38*, 116-119.

22. Bertossi, A. A. Dominating sets for split and bipartite graphs. *Inf. Process. Lett.* **1984**, *19.1*, 37-40.
23. Takamizawa, K.; Nishizeki, T.; Saito, N. Linear-time computability of combinatorial problems on series-parallel graphs. *J. of ACM* **1982**, *29.3*, 623-641.
24. Müller, H.; Brandstädt, A. The NP-completeness of Steiner tree and dominating set for chordal bipartite graphs. *Theor. Computer Science* **1987**, *53.2*, 257-265.
25. Hung, R. W.; Chang, M. S.; Ming-Hsiung, C. A linear algorithm for the connected domination problem on circular-arc graphs. Proceedings of the 19th Workshop on Combinatorial Mathematics and Computation Theory, **2002**.
26. Fomin, F. V.; Grandoni, F.; Kratsch, D. Solving connected dominating set faster than 2^n. *Algorithmica* **2008**, *52.2*, 153-166.