algorithms

MDPI

Article

Layered Graphs: Applications and Algorithms

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

1 Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, Amritapuri 690525, India; srijithb27@gmail.com (S.B.); sansth4x496@gmail.com (S.S.); skritics2@gmail.com (K.P.)
2 Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083, USA
* Correspondence: bhadrachalam@am.amrita.edu

Received: 4 June 2018; Accepted: 21 June 2018; Published: 28 June 2018

Abstract: The computation of distances between strings has applications in molecular biology, music theory and pattern recognition. One such measure, called short reversal distance, has applications in evolutionary distance computation. It has been shown that this problem can be reduced to the computation of a maximum independent set on the corresponding graph that is constructed from the given input strings. The constructed graphs primarily fall into a class that we call layered graphs. In a layered graph, each layer refers to a subgraph containing, at most, some k vertices. The inter-layer edges are restricted to the vertices in adjacent layers. We study the MIS, MVC, MDS, MCV and MCD problems on layered graphs where MIS computes the maximum independent set; MVC computes the minimum vertex cover; MDS computes the minimum dominating set; MCV computes the minimum connected vertex cover; and MCD computes the minimum connected dominating set. MIS, MVC and MDS run in polynomial time if $k = \Theta(\log \mid V \mid)$. MCV and MCD run in polynomial time if $k = O((\log \mid V \mid)^{\alpha})$, where $\alpha < 1$. If $k = \Theta((\log \mid V \mid)^{1+\epsilon})$, for $\epsilon > 0$, then MIS, MVC and MDS run in quasi-polynomial time. If $k = \Theta(\log \mid V \mid)$, then MCV and MCD run in quasi-polynomial time.

Keywords: NP-complete; layered graph; quasi-polynomial time; dynamic programming; independent set; vertex cover; dominating set; string transformations; social networks

1. Introduction

A string is a sequence of symbols from an alphabet Σ, in which a symbol can be repeated. An *adjacent swap* exchanges two consecutive elements in a sequence [1,2]. In a *signed string* (π, $\forall_i\ \pi[i]$) the following signs are assigned: $+$ for normal orientation and $-$ for reverse orientation. Adjacent swap over positions $i,\ i+1$ are denoted by $(i\,i+1)$. For unsigned strings (π), where $\pi = \pi[1], \pi[2], \pi[3], \ldots, \pi[i], \pi[i+1], \ldots, \pi[n]$, π transforms into π', where $\pi' = \pi[1], \pi[2], \pi[3], \ldots, \pi[i-1], \pi[i+1], \pi[i], \pi[i+2], \ldots, \pi[n]$. For signed strings ($\pi$), where $\pi = \pi[1], \pi[2], \pi[3], \ldots, \pi[i], \pi[i+1], \ldots, \pi[n]$, π transforms into π', where $\pi' = \pi[1], \pi[2], \pi[3], \ldots, \pi[i-1], -\pi[i+1], -\pi[i], \pi[i+2], \ldots, \pi[n]$. Two strings are *compatible* under some operation if they can be transformed into each other with the operation. That is, the unsigned string (2, 1, 3, 2) is compatible with (2, 1, 2, 3), whereas it is not compatible with (3, 2, 1, 1). The computation of the minimum number of adjacent swaps required to transform one given string into another compatible string, i.e., the *adjacent swap distance*, has applications in genetics and music theory [1]. A 1-*flip* toggles the orientation of a particular $\pi[i]$; it is denoted by $f_1(i)$. It changes the sign of $\pi[i]$. When it is applied to a signed string (π), where $\pi = \pi[1], \pi[2], \pi[3], \ldots, \pi[i], \pi[i+1], \ldots, \pi[n]$, π transforms into π', where $\pi' = \pi[1], \pi[2], \pi[3], \ldots, \pi[i-1], -\pi[i], \pi[i+1], \pi[i+2], \ldots, \pi[n]$.

A *short reversal* is either a (1-*flip*) or an adjacent swap. The *short reversal distance* is the minimum number of short reversals required to transform a signed string into another compatible string. Two strings are compatible under short reversals if and only if their unsigned versions, i.e., the strings whose signs are disregarded, are compatible [1]. The computation of the short reversal distance between α and β is reduced to the computation of the cardinality of the maximum independent set on a conflict graph constructed from α and β. It has applications in HOX gene clusters in vertebrate evolution [2,3]. In music theory, a composition is represented as a string. The smaller the distance between two patterns (compositions), the more similar they are [4].

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 the 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 [5]. The maximum independent set problem on a general graph is known to be NP-complete [6]. Certain classes of graphs have a polynomial time solution for this problem. Such algorithms are known for trees and bipartite graphs [7], chordal graphs [8], cycle graphs [9], comparability graphs [10], claw-free graphs [11], interval graphs and circular arc graphs [12]. 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 weighted independent set problem can be solved in linear time [13]. Thus, for several classes of graphs, MIS can be efficiently computed.

Hsiao et al. designed 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 a sorted endpoints list [14]. Several articles improved the complexity of the exponential algorithms that compute an MIS on a general graph [15,16]. Lozin and Milanic showed that MIS is polynomially solvable for 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 [17].

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 the MVC of G, then $\forall e = (u, v) \in E$, $u \in V^* \vee v \in V^*$. In this article, G is undirected, so an edge (u, v) is understood to be an undirected edge. The minimum dominating set (i.e., MDS) and the minimum connected dominating set (i.e., MCD) problems seek to identify a dominating set i.e., DS and a connected dominating set i.e., CDS, respectively, with minimum cardinalities. The MVC, MDS and MCD problems on general graphs are known to be NP-complete [6]. Garey and Johnson showed that MVC is one of the first NP-complete problems [6]. In connected vertex cover problems (i.e., MCV), given a connected graph (G), a connected vertex cover (i.e., 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 [19,20]. Garey and Johnson proved that the MCV problem is NP-hard in planar graphs, with a maximum degree of 4 [6]. Li et al. proved that for 4-regular graphs, the MCV problem is NP-hard [21]. It has been shown that for series-parallel graphs, which are a set of planar graphs, the minimum vertex cover can be computed in linear time [22].

Garey and Johnson showed that the MDS problems on planar graphs with maximum vertex degree 3 and planar graphs that are regular with degree 4 are NP-complete [6]. MCD is NP-complete even for planar graphs that are regular with degree 4 [6]. Bertossi showed that the problem of identifying a MDS is NP-complete for split graphs and bipartite graphs [23]. Cockayne et al. proved that MDS in trees can be computed in linear time [24]. Baker designed various approximation algorithms for planar graphs [25]. Muller and Brandstadt showed that MDS and MCD are NP-complete for chordal bipartite graphs [26]. Ruo-Wei et al. proved that for a given circular arc graph with n sorted arcs, MCD is linear in time and space [27]. Fomin et al. proposed an algorithm with a time complexity faster than 2^n to solve the connected dominating set problem [28].

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 quality of service (QoS) constraints is NP-hard [29]. A graph known as a *layered graph* was constructed from a given input graph, and the authors showed that the hop-constrained minimum spanning tree problem is equivalent to the 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 algorithms for various graph-theoretic problems.

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, where the i-th layer is G_i. Note that for any given i, E_{ij}, where $j = i+1$ can be ϕ and $\forall_{l \neq i+1} E_{il} = \phi$. Every vertex within a given layer gets a label from $(1, 2, 3, \ldots, k)$. Thus, $V_i \subseteq \{V_{i1}, V_{i2}, \ldots, V_{ik}\}$. Note that V_{ix} is the vertex number, x, in layer i. However, in layer i, the vertex number, x, may 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 defined the following restrictions on a layered graph. Several of the these restrictions can be combined. Please see Figure 1.

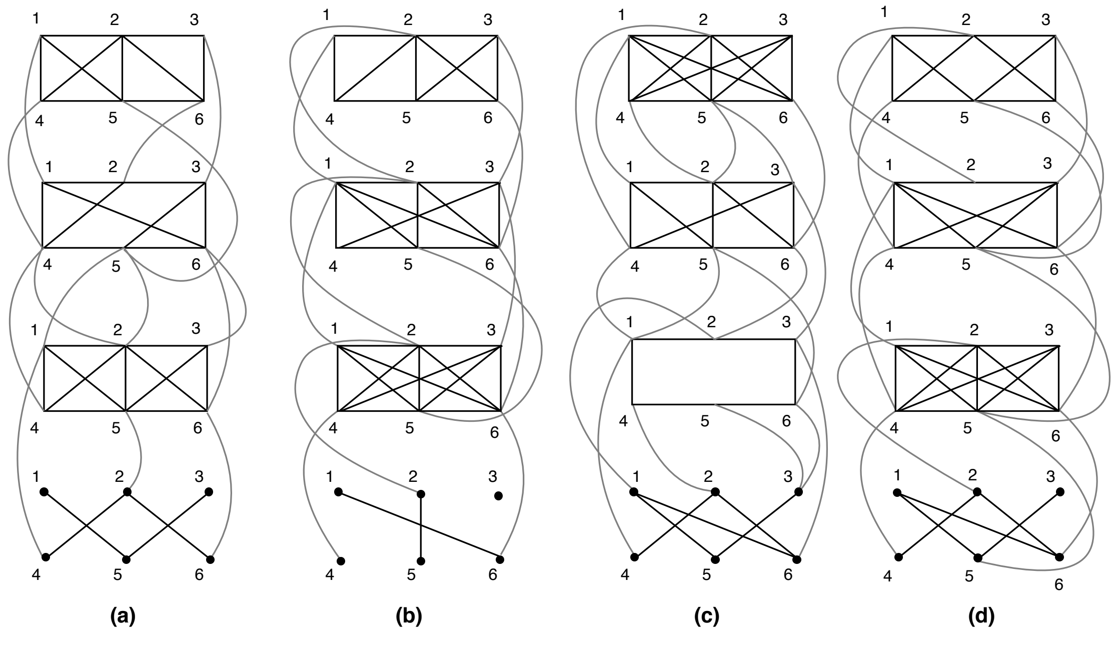

Figure 1. (**a**) $LG_6^{24,4}$. (**b**) $LLG_6^{23,4}$. (**c**) $SLG_6^{24,4}$. (**d**) $SLLG_6^{24,4}$. Layer 1 is the topmost and layer 4 is the bottommost. Vertices have labels from $\{1,2,3,4,5,6\}$ within a given layer. Intra-layer edge is a thick line whereas inter-layer edge is thinner. (**a**–**d**) are CLGs as well. (**a**,**b**) are not a SLGs ((**a**): layer 4 has two components {1,3,5} and {2,4,6}. (**b**): layer 4 has three components {1,6}, {2,5} and {4}; vertex 3 does not exist, only a placeholder is shown).

- If $\forall_i\ |\ V_i\ | \leq k$, 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 $\forall_t$, (V_{it}, V_{ju}) is an inter-layer edge $\rightarrow (t = u) \wedge j \in \{i-1, i+1\}$, then a *linear layered graph*, i.e., LLG, is obtained. LLG_k denotes an LLG that is *k-restricted*.
- If $\forall_i\ G_i$ is 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.

The problems of computing the adjacent swap distance between unsigned strings, adjacent swap distance between signed strings and short reversal distance were addressed in [1]. If the alphabet is Σ, $|\Sigma|$ is k, the source string is α and the destination string is β and the length of α (and β) is n, a *pairing diagram* can be drawn for α and β, where all elements of α and β are treated as vertices, and perfect matching is performed on all $2n$ vertices, where each edge corresponds to $(\alpha[i], \beta[j])$ (here, both $\alpha[i]$, $\beta[j]$ denote the same symbol [1]). The solutions to the above problems are based on *optimum pairing* (in contrast to any perfect matching), where there is an edge from the i-th occurrence of a symbol x in α to the i-th occurrence of x in β; the corresponding pairing diagram is the *optimum pairing diagram*. The solutions for adjacent swap distances are complete. However, the solution suggests that the short reversal distance is partial; it can solve very few sub-cases. Several distance problems on strings have been shown to be NP-complete [30]. However, the complexity of short reversal distance problem is unknown. The short reversal distance was studied in [2]. It was shown that the edges corresponding to two consecutive occurrences of a symbol x in optimum pairing form a special edge-pair if they meet certain criteria. A conflict graph, G, is constructed from an optimum pairing diagram where a vertex denotes a special edge-pair, and an edge exists between a pair of vertices that are in conflict. The computation of the short reversal distance is reduced to the computation of MIS on G. G consists of several subgraphs, $\forall_i\, G_i$, each having, at most, k vertices, where each vertex corresponds to a symbol in Σ. Each G_i is a subgraph of the K_k clique. The vertices, $u \in G_i, v \in G_j$ $(i \neq j)$, can have a conflict only if they correspond to the same symbol. Further, they share a common edge in the optimum pairing diagram. In this particular scenario, the layered graphs arise naturally. Further, such layered graphs are LLGs. In this framework, the computation of MIS on a LLG is a necessary component in the computation of the corresponding short reversal distance.

Considering a tribal society S consisting of some villages on a bank of a river, a village consists of a few families where each family has its own *family-head*. The family-heads of a given village know one another, and they also interact with specific family-heads of adjacent villages for trade (of produce) and partnership (collaboration in farming etc.). If one models this society as a social network, where a family-head is denoted by a vertex and an interaction (among family-heads) is denoted by an edge, then one obtains a layered graph. In this social network, identifying the smallest set of influencers is a natural pursuit (whose solution is given by computing MDS). These applications motivate the study of MIS, MDS and other graph theoretic problems on layered graphs. In general, graph theoretic problems, like subgraph isomorphism, and its variations have extensive applications in computational biology, e.g., references [31,32].

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 last b layers do not have any edges. The graph is not a CLG; however, the MCV of G is the 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 intra-layer edge, then MCV can be computed only on CLG. MCD is well defined only for CLG because it must dominate all vertices.

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, uv: $u \in G_i$ 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 a *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 a *full* LLG_k^q. The number of layers in LG_k i.e., q is bound 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 designed 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 designed a generic dynamic programming algorithm for all of 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.

We denoted the vertices chosen in a particular layer with a k-bit variable that we called *mask*. The pth bit of the mask was set to one to include the pth vertex. Otherwise, the bit was set to zero and the vertex was 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 cs_i. For layers $1, \ldots, i$, a combined candidate sub-solution is maintained, denoted ccs_i. Likewise, $cs_{i,j}$ and $ccs_{i,j}$ denote the sub-solutions (of layer i and first i layers respectively), where the vertices chosen from layer i are denoted by mask j. Only the cardinality of the best options is stored; 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 are irrelevant. This is a key feature.

3.1. Input

The input consists of $LG_k^{n,q}$ which 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 $E_{i,i+1}$. Rows $1, 2, \ldots, k$ of I_i correspond to vertices $V_{i1}, V_{i2}, \ldots, V_{ik}$, and columns $1, 2, \ldots, k$ of I_i are 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 G_i also requires $O(k^2)$ space. Therefore, the total space required for the input graph is $O(nk)$, since each layer requires $O(k^2)$ space, and there are $O(n/k)$ layers.

The Boolean valued function *compatible* (please see Algorithm 1) determines whether the candidate sub-solutions (of the current layer and the subgraph induced by vertices of all previous layers) can be combined; here, the layer number, i, is implicit. For each mask, j, of a given layer, i, the 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 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 $count_{i,j}$.
- (iii) Summarize: At layer q :, execute (i) and (ii). Identify the optimum cardinality among $\forall_j sol_{q,j}$ and the corresponding count.

A particular problem has specific characteristics. In the sequel, where a problem is dealt with in detail, the corresponding validity/compatibility and other specifics are elucidated.

Algorithm 1 Compatible Algorithm

```
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 $i$, otherwise, it 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 for each component of $l$, $\exists c \in l$: $(\exists_d : I[c][d] = 1) \wedge (bit_d(j) = 1)$
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           // $dom(j, l)$: $D \leftarrow cs_{i,l} \bigcup cs_{i+1,j} \bigcup Adj(cs_{i,l}) \bigcup Adj(cs_{i+1,j})$
    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,
                              // For each component of $l$, $\exists c \in l$: $(\exists_d : I[c][d] = 1) \wedge (bit_d(j) = 1)$
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 $V$
```

3.2. MIS

Consider the structure of an 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 independent set. (please see Figure 2). 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 $M_2 = \bigcup_{j=i+1}^{q} V_j^*$, 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 the edges in $E_{i\ i+1}$ need to be considered. Thus, the cardinality of an MIS of $LG_k^{n,q}$ is equal to $max(\forall_{E^C=\phi} \mid M_1 \mid + \mid M_2 \mid)$.

- $feasible(j)$: 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 $opt \leftarrow max(\forall_j sol_{q,j})$; $count \leftarrow 0$; $\forall_j$ $if(sol_{q,j} = opt) count \leftarrow count + count_{q,j}$; Return $(opt, count_{q,j}))$

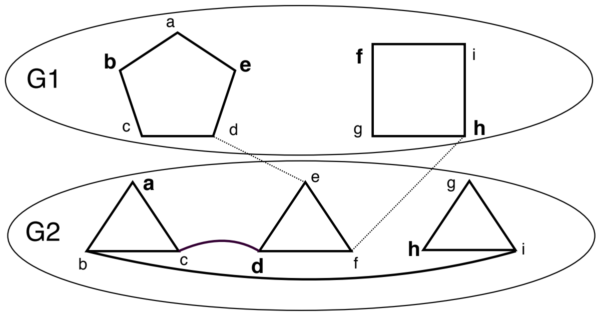

Figure 2. MIS: Graph G consists of two layers, G_1 and G_2. The same graph is employed for the illustration of other problems. The vertices of a maximum independent set are $\{b, e, f, h\}$ from G_1 and $\{a, d, h\}$ from G_2. The cardinality of any MIS of G is 7. The vertices of MIS are shown in larger bold font.

3.3. MVC and MCV

Consider a vertex cover $V^* = \bigcup_{j=1}^{q} V_j^*$ 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 (please see Figure 3). V_j^* depends only on V_{j-1}^* and V_{j+1}^*. Consider two adjacent layers, p and $p+1$. $V_p^* \cup V_{p+1}^*$ 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, for MCV, the induced subgraph of V^* must be a connected component (please see Figure 4). In the sequel, the time and space complexity analyses for these problems are presented.

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 has more than one component, i.e., the set of vertices denoting l is partitioned into several connected components.

Case (a): For layer i, mask j is infeasible if either (I) vertices corresponding to j and l have no edges among them or (II) all edges in I_i are not covered. Otherwise, j is feasible. If at least one edge exists across j and l:

(i) If j is a single connected component, then the result is also a single component (consisting of all chosen vertices).
(ii) If j has more than one connected component and all of them connect to l, then the result is also a single component.
(iii) If 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 and the rest are separate components.

Case (b): Every component from the previous layer corresponding to mask l must connect to some component in the current layer. Otherwise, the pair j and l is infeasible for layer i. For feasible pairs the following possibilities exist:

(i) Every component in l has an edge with exactly one component in j. Here, the partition is determined by j.
(ii) A component, C, in l has edges with $C_1, \ldots, C_a$ in j. Then, $C_1, \ldots, C_a$ can be merged into one component as they are connected through C.

A particular partitioning of the current layer can occur due to various choices of l. For each partition corresponding to j, the sub-solution is stored with minimum cardinality. Thus, for each mask, j, there are, at most, B_k (k-th Bell number) solutions stored. When 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.

- $feasible(j)$: Mask j must denote a VC for G_i.
- $compatible(j,l)$: The union of two VCs must be a VC for edges in G_{i-1}, G_i and I_{i-1}. i is the current layer. For MCV, all components of l must have edges with vertices in j. If $i = q$, then V^* must be one component.
- Extension: If $(cardinality(j) + sol_{i-1,l} < sol_{i,j})$ $sol_{i,j} \leftarrow cardinality(j) + sol_{i-1,l}$.
- Summarize: Let $opt \leftarrow min(\forall_j sol_{q,j})$; $count \leftarrow 0$; $\forall_j$ $if(sol_{q,j} = opt) count \leftarrow count + count_{q,j}$; Return $(opt, count)$

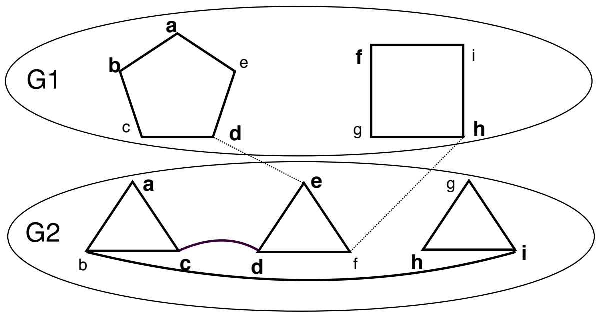

Figure 3. MVC: The vertices of a minimum vertex cover are $\{a, b, d, f, h\}$ from G_1 and $\{a, c, d, e, h, i\}$ from G_2. The cardinality of any MVC of G is 11. The vertices of MVC are shown in larger bold font.

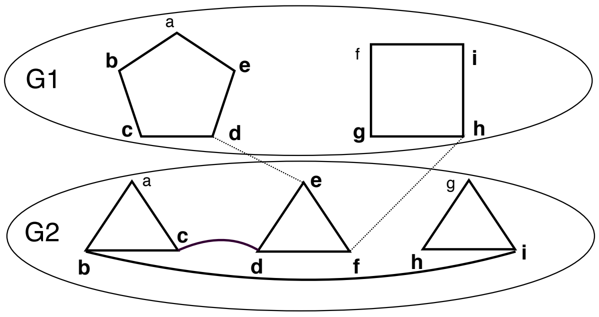

Figure 4. MCV: The vertices of a minimum connected vertex cover are $\{b, c, d, e, g, h, i\}$ from G_1 and $\{b, c, d, e, f, h, i\}$ from G_2. The cardinality of any MCV of G is 14. The vertices of MCV are shown in larger bold font.

3.4. MDS and MCD

Let an MDS of $LG_k^{n,q}$ be V^*, such that, $V^* = \bigcup_{j=1}^{q} V_j^*$, where V_j^* represents the vertices in this MDS from layer j. Clearly, V_j^* may not be a dominating set of layer j because the vertices of V_j can be dominated by any subset of $V_{j-1}^* \bigcup V_j^* \bigcup V_{j+1}^*$. In Figure 5, $e \in G_2$ is dominated by $d \in G_1$. It follows that $\bigcup_{j=1}^{p+1} V_j^*$ must dominate all vertices in $\bigcup_{j=1}^{p} V_j$. Further, $V^* = \bigcup_{j=1}^{q-1} V_j^* \bigcup V_q^*$ must dominate $\bigcup_{j=1}^{q} V_j$. A vertex that is not dominated is *undominated*.

Consider mask j in layer i, where $cs_{i,j} \bigcup ccs_{i-1,l}$ dominates layer $i-1$. This particular subset of vertices can leave some vertices in layer i undominated. The number of such choices is 2^k; each choice is denoted by a k-bit variable that we call a mask—here, a mask of *exclusion*. Further, when one processes layer $i+1$, this information is critical. We show that the $O(2^k)$ 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 *undominated* in layer i, s is the cardinality of the vertices chosen so far and c is the number of choices corresponding to u for a particular j in layer i.

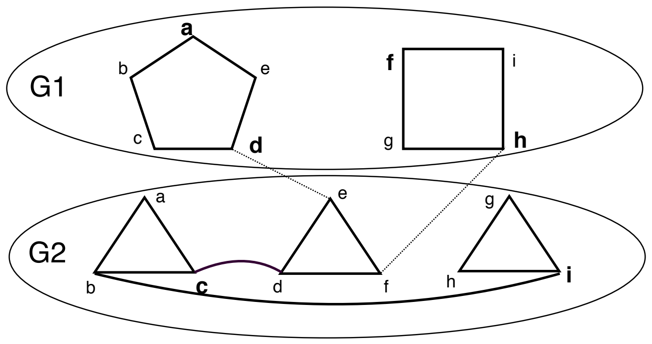

Figure 5. MDS: The vertices of a minimum dominating set are $\{a, d, f, h\}$ from G_1 and $\{c, i\}$ from G_2. The cardinality of any MDS of G is 6. The vertices of MDS are shown in larger bold font.

MCD has an additional requirement compared to MDS, i.e., V^* must form a single component (please see Figure 6). For possible combinations of component layouts of the current and previous layer masks, see MCV. For 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 produces 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 of a particular triple in the current layer needs 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, when $k = O(\log n)$ for MDS and $k = O(\log n)^\alpha$ where $\alpha < 1$ for MCD, the algorithm runs in polynomial time.

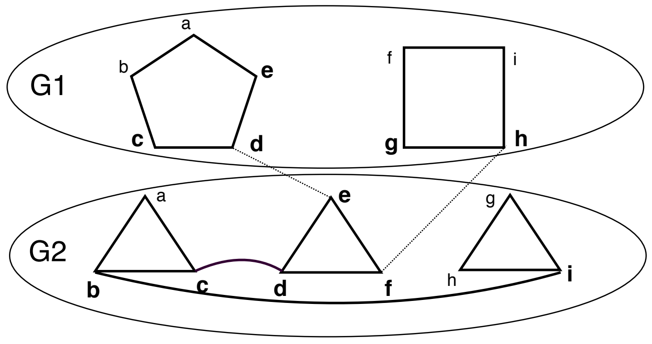

Figure 6. MCD: The vertices of a minimum connected dominating set are $\{c, d, e, g, h\}$ from G_1 and $\{b, c, d, e, f, i\}$ from G_2. The cardinality of any MCD of G is 11. The vertices of MCD are shown in larger bold font.

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 $0, \ldots, 2^k - 1$; 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 be 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 as S.
- In S, for any two triples with the same mask, we need only to retain the triples with the smallest 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 $\forall_j \; cs_{i,j}$. The information of only two layers is stored. 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 the previous layer must be processed. That is, for all masks of layer i, a total of 2^{3k} triples must be processed.
- Consider 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 of the triples stored corresponding to l in layer $i-1$, we need only to 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 in the previous layer. So, for all masks in 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. 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 analyses. The current layer in the following functions is i.

- $feasible(j)$: Any j is valid.
- $compatible(j,l)$: Tthe union must dominate all vertices of V_{i-1}. For MCD, all components of l must have edges with vertices in j. If $i = q$, then V^* must be one component and it must dominate V_q also.
- Extension: Performed as per the observations listed above.
- 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 $(opt, count)$.

3.5. Compatible Algorithm

Given candidate sub-solutions for consecutive layers one must determine if their union is a feasible sub-solution. The following algorithm determines the same for the problems addressed in this article.

3.6. Generic Optimum Algorithm

The algorithms (please see Algorithm 2) for the MIS, MVC and MDS problems on $LG_k^{n,q}$ are similar, while those for MCV and MCD must additionally ensure connectedness criterion. We give a generic dynamic programming based algorithm for both sets of problems. Some specific instances are shown in Appendix A.

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 layer i are given by the binary value of j. $count_{ij}$: The number of ways that the j-th mask in layer i yields the corresponding optimum value.

Algorithm 2 Generic Optimum Algorithm

```
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 the 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 the 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 $k = O(\log n)$. When $k = \log n$ these time complexities yield $O(n^3k)$, $O(n^3k)$ and $O(n^4)$ respectively. The space complexities are $O(nk)$, $O(nk)$ and $O(k2^{2k})$ respectively. When $k = \log n$ these space complexities yield $O(nk)$, $O(nk)$ and $O(kn^2)$ 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 MCD and MCV. The time and space complexities of MVC and MCD are analyzed. The proofs of correctness for the remaining problems are similar. The time complexity for MDS was presented earlier.

Theorem 1. *The MIS Algorithm 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 we 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 the edges in $E_{i\ i+1}$ need to be considered. Thus, the cardinality of an MIS of $LG_k^{n,q}$ is equal to $max(\forall_{E^C=\phi} \mid I_1 \mid + \mid I_2 \mid)$. When the last layer is processed, the cardinalities of the ISs of subgraphs induced by both V and $V - V_q$ 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 that an independent set of maximum cardinality can be formed when the vertices chosen in layer i are given by j. Thus, the $count_{qj}$ corresponding to the maximum value of sol_{qj} yields the total number of MISs. □

Theorem 2. *The MVC Algorithm 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 the MVC of $LG_k^{n,q}$ is equal to $min(\mid M_1 \mid + \mid M_2 \mid)$ 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 that a vertex cover of minimum cardinality 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. □

Theorem 3. *The MVC Algorithm 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 is 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 whether VC is valid in $O(k^2)$ time with M_i. That is, for any two $M_i[a][b]$ that are set, the mask should have either a bit a or a bit b set.
- Given I_i and two masks, *mask*1 and *mask*2, for layers i and $i+1$, respectively, it can be directly determined whether the union of the two masks is a VC of the 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, 4.2^k masks and count variables are stored, each of size k. The space required is $O(nk)$ to store 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 $0 \le \alpha < 1.0$, 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 $h(n) = n^\epsilon, \epsilon > 0$.

Thus, $f(n)! = (\log n)^\alpha!$. Taking log on both sides we obtain:

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

Disregarding smaller order terms we obtain the expression: $(\log n)^{\alpha}(\alpha \log\log n)$
$\alpha \log\log n = \log((\log n)^{\alpha})$. Note that $\log n = O(n^{\mu})$ for any $\mu > 0$. Thus, $(\log n)^{\alpha}(\alpha \log\log n) < ((\log n)^{\alpha}))^{(1+\mu)}$ for any $\mu > 0$. Given that $\alpha < 1$ one can always choose μ such that $\alpha + \alpha\mu < 1$. Thus, the expression is $O(\log n) = O(n^{\epsilon})$ for any $\epsilon > 0$.

Thus, $(\log n)^{\alpha}! = O(n^{\epsilon})$. □

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

Proof.
From Stirling's Approximation, we have $\log((\log n)!) \approx \theta(\log n \log\log n)$

$$\Rightarrow (\log n)! \approx e^{\theta(\log n \log\log n)}$$

Thus, for some constants μ and δ, $(n^{\delta \log\log n}) \leq ((\log n)!) \leq (n^{\mu \log\log n})$

Thus, $(f(n)!)$ is quasi-polynomial. □

Lemma 3. *If* $k = \Theta((\log n)^{1+\epsilon})$, *for any* $\epsilon > 0$ *then, the MIS Algorithm, MVC Algorithm and MDS Algorithm 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. *The MCD Algorithm 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 $\mid V^1 \mid > 0$ and $\mid V^2 \mid > 0$. 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 $\mid E^C \mid > 0$, 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 $\mid M_1 \mid + \mid M_2 \mid$, 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 $\mid M_1 \mid + \mid M_2 \mid$ 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 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.

The time complexity of the algorithm is analyzed below. We presumed that similar prerequisites as in Theorem 3 were provided. The steps are presented 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 consist of a maximum of $B_k 2^k$ triples of the form (lo, un, r). This corresponds to a maximum of B_k component layouts (lo), 2^k masks, un undominated vertices of the current layer and a maximum of 2^k triples, r, of the form (m, s, c), for every unique pair (lo, un). Here, m is the mask of the current layer that produced the respective (component layout, undominated vertices) pair; s is the minimum cardinality of the sub-solution corresponding to mask m and pair (lo, un); and c is the 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 the information of sol_1 is used 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 the component layout.
- The number of component layouts is upper bounded by 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 $B_k 2^{2k}$, 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 masks of the current layer and the triple values from the previous layer are used to generate the triples for the current layer.
- For a unique pair (lo, un) in 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 with 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, can feasibly 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 the component layout and undominated mask computed for mask j of the current layer are checked with the existing pairs of the current layer to determine if it is unique or not. We maintain the structure of the triples 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 triple 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 triple 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.

- The above procedure is performed until 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 maintained a global structure, as mentioned above. It consisted 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 maintainewa 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, an expression for the runtime of the algorithm is derived. 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 are 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 $O(k^2)$. Computing the new component layout and the new undominated mask takes $O(k^2)$ 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

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

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

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

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 the logarithmic functions grow slower than the 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 intege.

Taking the log of both sides, we get

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

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

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

Hence, we have proved our claim.

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

From above, we have

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

From (1), we get

$$\begin{aligned} T &\leq cn^{1+\gamma} * n^{\delta+\mu} \\ &\leq cn^{1+\gamma+\delta+\mu} \end{aligned}$$

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 any value, such that $\epsilon > 0$.

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

Hence, the theorem is proved. □

Theorem 5. *The MCV Algorithm 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. The MCV Algorithm is similar to the MCD Algorithm. A mask, j, of layer i must be a valid VC for layer i. The check takes an additional $O(k^2)$, 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. Thus, if $k = \Theta(\log n)$ then MCV and MCD are computed in quasi-polynomial time. Proving this is quite straightforward. By substituting $(\log n)!$ for $k!$ into Equation (1) in Theorem 4, we get a product of a quasi-polynomial factor and a polynomial factor. Thus, the time complexity is quasi-polynomial.

Minor Enhancements

The current layer requires 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 the current layer are stored at index 1 and the variables of the previous layer are stored 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 $O(1)$.

The optimum cardinalities for each of the problems are generated using minimal additional space. For example, the MVC Algorithm employs only $O(k2^k)$ space in addition to the space required by the graph. If, for each mask in each layer, we store the best compatible mask from the 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.

5. Conclusions

A novel graph class called layered graphs was defined. It includes a subset of bipartite graphs and a subset of trees with n vertices. A layered graph can have exponential number of cycles. The typical restrictions like bipartiteness, planarity and acyclicity on graph classes that admit polynomial time solutions for hard problems are not applicable for this class. The known NP-complete problems were shown to be in class P for these graphs when the 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. The count of the corresponding optimal solutions is also computed. We note that the identification of a maximum clique in a layered graph is direct as the inter-layer edges are limited to adjacent layers. Thus, one can independently solve $q-1$ instances of the maximum clique problem on graphs induced by $V_i \bigcup V_{i+1}$ (of size $\leq 2k$, each in $O(2^{2k}k^2)$ time) to obtain a maximum clique of $LG_k^{n,q}$.

A few applications of layered graphs in computational molecular biology and social networks were addressed in this article. One can explore other problems that can be modeled by layered graphs. The design of larger graph classes (that include layered graphs as a sub-class) that admit polynomial time solutions for the problems studied in this article is an open problem. For example, in a layered graph, the inter-layer edges are restricted to the vertices of adjacent layers. If this restriction is dropped for $O(1)$ edges originating from each layer, then one obtains a more general graph class. Such graphs can potentially model more scenarios than layered graphs.

Author Contributions: B.C. conceived the design of graphs and the solution, performed analysis and wrote the first draft. S.B., S.S. and K.P. helped to test the algorithms. K.P. worked on the 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 of the algorithms and wrote the later versions of the paper.

Funding: This research received no external funding.

Acknowledgments: Authors thank Amma for her guidance. Sriram K helped with the literature survey.

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

Appendix A

The generic algorithm was presented earlier. Here, we present detailed algorithms for MIS and MVC. A relatively high-level description of the MCD algorithm follows.

Appendix A.1. MIS Algorithm

Algorithm A1 MIS Algorithm

```
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. The 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
            $sol_{1j} \leftarrow size$
        end if// $j$ is valid
    end for//For all masks of current layer
    $\forall x\ count_{0x} \leftarrow count_{1x}; sol_{0x} \leftarrow sol_{1x}; count_{1x} \leftarrow sol_{1x} \leftarrow 0;$
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
```

Appendix A.2. MVC Algorithm

Algorithm A2 MVC Algorithm

```
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 $//sol_{0l} = 0 \rightarrow$Invalid VC
                    if $(cardinality(j) + sol_{0l} \leq size)$ 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
                $sol_{1j} \leftarrow size$
            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
    $\forall x\ count_{0x} \leftarrow count_{1x}; sol_{0x} \leftarrow sol_{1x}; count_{1x} \leftarrow sol_{1x} \leftarrow 0;$
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
    end if
end for
$return(best, sum)$ //MVC cardinality and the count of such MVCs
```

Appendix A.3. MCD Algorithm

Algorithm A3 MCD Algorithm

```
// 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 corresponds 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, $un$ is the mask of undominated vertices of the current layer; $r$ is triples of the form $(m, s, c)$ where $m$ is the mask of the current layer that produces the respective (component layout, undominated vertices) pair; $s$ is the minimum cardinality of the sub-solution corresponding to mask $m$ and pair $(lo, un)$; $c$ is the 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 as for the previous layer.
// Mask $j$ 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.
                    Otherwise, insert the new triple $(m, s, c)$ for the corresponding $(l, u)$ pair in $sol_1$.
            end for
        end for
    end for
end for
$best \leftarrow$ 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. Chitturi, B.; Sudborough, H.; Voit, W.; Feng, X. Adjacent Swaps on Strings. In Proceedings of the Computing and Combinatorics (COCOON 2008), Dalian, China, 27–29 June 2008; Springer: Berlin/Heidelberg, Germany, 2018; Volume 5092.
2. Chitturi, B.; Mohandas, M.; Sudborough, H.; Voit, W. Adjacent Swaps on Strings. *Algorithms Mol. Biol.* **2018**, submitted.
3. Alberts, B.; Johnson, A.; Lewis, J.; Raff, M.; Roberts, K.; Walter, P. *Molecular Biology of the Cell*; Garland Science: New York, NY, USA, 2002.
4. Feng, X.; Chitturi, B.; Sudborough, H. Sorting circular permutations by bounded transpositions. In *Advances in Computational Biology*; Springer: New York, NY, USA, 2010; pp. 725–736.
5. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*; Springer: Boston, MA, USA, 1972; pp. 85–103.
6. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman & Co.: New York, NY, USA, 1979.
7. Harary, F. *Graph Theory*; Addison-Wesley Pub. Co.: Boston, MA, USA, 1969.
8. Gavril, F. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum vertex cover of a chordal graph. *SIAM J. Comput.* **1972**, *1*, 180–187. [CrossRef]
9. Gavril, F. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks* **1973**, *3*, 261–273. [CrossRef]
10. Golumbic, M.C. The complexity of comparability graph recognition and coloring. *Computing* **1977**, *18*, 199–208. [CrossRef]
11. Minty, G.J. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory Ser. B* **1980**, *28*, 284–304. [CrossRef]
12. Gupta, U.I.; Lee, D.T.; Leung, J.T. Efficient algorithms for interval graphs and circular arc graphs. *Networks* **1982**, *12*, 459–467. [CrossRef]
13. Chen, G.H.; Kuo, M.T.; Sheu, J.P. An optimal time algorithm for finding a maximum weight independent set in a tree. *BIT Numer. Math.* **1988**, *28*, 353–356. [CrossRef]
14. 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*, 229–235. [CrossRef]
15. Tarjan, R.E.; Trojanowski, A.E. Finding a maximum independent set. *SIAM J. Comput.* **1977**, *6*, 537–546. [CrossRef]
16. Robson, J.M. Algorithms for maximum independent sets. *J. Algorithms* **1986**, *7*, 425–440. [CrossRef]
17. Lozin, V.V.; Milanic, M. On the Maximum Independent Set Problem in Subclasses of Planar Graphs. *J. Graph Algorithms Appl.* **2010**, *14*, 269–286. [CrossRef]
18. Garey, M.R.; Johnson, D.S. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.* **1977**, *32*, 826–834. [CrossRef]
19. Bondy, J.A.; Murty, U.S.R. *Graph Theory with Application*; Macmillan: London, UK, 1976.
20. Konig, D. Graphen und matrizen. *Mat. Fiz. Lapok* **1931**, *38*, 116–119. (In German)
21. Li, Y.; Yang, Z.; Wang, W. Complexity and algorithms for the connected vertex cover problem in 4-regular graphs. *Appl. Math. Comput.* **2017**, *301*, 107–114. [CrossRef]
22. Takamizawa, K.; Nishizeki, T.; Saito, N. Linear-time computability of combinatorial problems on series-parallel graphs. *J. ACM* **1982**, *29*, 623–641. [CrossRef]
23. Bertossi, A.A. Dominating sets for split and bipartite graphs. *Inf. Process. Lett.* **1984**, *19*, 37–40. [CrossRef]
24. Cockayne, E.; Goodman, S.; Hedetniemi, S. A linear algorithm for the domination number of a tree. *Inf. Process. Lett.* **1975**, *4*, 41–44. [CrossRef]
25. Baker, B.S. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM* **1994**, *41*, 153–180. [CrossRef]
26. Müller, H.; Brandstädt, A. The NP-completeness of Steiner tree and dominating set for chordal bipartite graphs. *Theor. Comput. Sci.* **1987**, *53*, 257–265. [CrossRef]
27. Hung, R.W.; Chang, M.S.; Ming-Hsiung, C. A linear algorithm for the connected domination problem on circular-arc graphs. In Proceedings of the 19th Workshop on Combinatorial Mathematics and Computation Theory, Kaohsiung, Taiwan, 29–30 March 2002.

28. Fomin, F.V.; Grandoni, F.; Kratsch, D. Solving connected dominating set faster than 2^n. *Algorithmica* **2008**, *52*, 153–166. [CrossRef]
29. Gouveia, L.; Simonetti, L.; Uchoa, E. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. *Math. Program.* **2011**, *128*, 123–148. [CrossRef]
30. Chitturi, B. A note on complexity of genetic mutations. Discrete Mathematics. *Discret. Math. Algorithms Appl.* **2011**, *3*, 269–286. [CrossRef]
31. Chitturi, B.; Bein, D.; Grishin, N. Complete enumeration of compact structural motifs in proteins. In Proceedings of the International Symposium on Biocomputing, Kerala, India, 15–17 February 2010; pp. 19–27.
32. Chitturi, B.; Shi, S.; Kinch, L.N.; Grishin, N. Compact Structure Patterns in Proteins. *J. Mol. Biol.* **2016**, *428*, 4392–4412. [CrossRef] [PubMed]