Finding a Maximum Weight Sequence with Dependency Constraints

by

Behrooz Sepehry

B.Sc., Sharif University of Technology, 2014

AN ESSAY SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2016

Abstract

In this essay, we consider the following problem: We are given a graph and a weight associated with each vertex, and we want to choose a sequence of vertices that maximizes the sum of the weights, subject to some constraints arising from dependencies between vertices. We consider several versions of this problem with different constraints. These problems have applications in finding the convergence rates for some optimization algorithms, including coordinate descent with Gauss-Southwell rule and greedy Kaczmarz.

Preface

The main problem considered in this essay, introduced in chapter 1, was first introduced in [1]. In that paper, the authors solved the problem for a special case. In this essay, I solved the problem for the general case. This solution was published in [2]. I also consider several generalizations of the problem in this essay and solved some of them.

Table of Contents

List of Figures

Chapter 1

Introduction

This essay is motivated by solving the following problem, which we call the *Maximum weight Sequence with Dependency constraints* and abbreviate it to MSD:

We are given a non-empty graph $G = (V, E)$ with at least one edge, a weight $W(v_i)$ associated with each vertex $v_i \in V$, and an iteration number T. We want to choose a sequence of vertices $\mathcal{V} = \{v_{i_t}\}_{t=1}^{T}$ that maximizes $\sum_{t=1}^{T} W(v_{i_t})$, subject to the following constraint: after each time vertex v_i is selected, it cannot be selected again until after a neighbor of vertex v_i has been selected.

This problem naturally arises while trying to find the convergence rates for coordinate descent with Gauss-Southsell rule and greedy Kaczmarz method.

In this essay, we will solve this problem and will give a polynomial time algorithm to find the solution. Then we consider several generalization of this problem, in which the constraints are different.

1.1 Coordinate Descent with Gauss-Southsell Rule[1]

In recent years, coordinate descent methods have become very useful in solving large-scale optimization problems. Nesterov has shown that coordinate descent can be faster than Gradient Descent in the following cases. Assuming we are optimizing n variables, the cost of performing n coordinate descent updates is similar to the cost of performing one full Gradient Descent update. These cases includes the family of functions

$$h(x) = \sum_{i \in V} g_i(x_i) + \sum_{(i,j) \in E} f_{i,j}(x_i, x_j),$$

where $f_{i,j}$ are smooth, $G(V, E)$ is a graph and all functions are convex. This family of functions includes quadratic functions, graph-based label propa-

[1]This section is based on [1]

gation algorithms for semi-supervised learning, and finding the most likely assignments in continuous pairwise graphical models [1].

Our objective is to find $\min_{x\in\mathbb{R}^n} h(x)$. In coordinate descent with Gauss-Southwell rule with exact optimization, at each step t, we choose a coordinate i_t as follows

$$i_t = \operatorname*{argmin}_i |\nabla_i h(x^t)|$$

and then we optimize the function with respect to coordinate i_t

$$\begin{aligned}\alpha_t &= \operatorname*{argmin}_\alpha \{h(x^t + \alpha e_{i_t})\}\\ x^{t+1} &= x^t + \alpha_t e_{i_t}.\end{aligned}$$

In [1], the authors show that the convergence rate of this method is

$$h(x^t) - h(x^*) \le \left[\prod_{t=1}^{T}\left(1 - \frac{\mu_1}{L_{i_t}}\right)\right][h(x^0) - h(x^*)], \tag{1.1}$$

where x^* is the optimal solution, μ_1 is the measure of strong convexity of the function h with respect to 1-norm, which means that for every two points x and y, we have

$$h(y) \ge h(x) + \langle \nabla h(x), y - x\rangle + \frac{\mu_1}{2}||y - x||_1^2,$$

and L_i is the Lipschitz constant for the partial derivative of h with respect to coordinate i,

$$|\nabla_i h(x + \alpha e_i) - \nabla_i h(x)| \le L_i|\alpha|.$$

Furthermore, after we have updated the coordinate i, the algorithm will never select it again until one of its neighbors has been selected. This is because when a coordinate i is selected at step t, the function is optimized with respect to coordinate i, so we would have $\nabla_i h(x^{t+1}) = 0$, and based on the definition of $h(x)$, the value of $\nabla_i h(x)$ will not be changed until one of the neighbors of i has been selected.

To find the worst case bound on convergence rate of the algorithm, we need to find the maximum value of $\prod_{t=1}^{T}\left(1 - \frac{\mu_1}{L_{i_t}}\right)$ in (1.1) subject to the constraint that a coordinate will never be selected again until after one of its neighbors has been selected. Note that if we do not have any constraints, then the maximum value of $\prod_{t=1}^{T}\left(1 - \frac{\mu_1}{L_{i_t}}\right)$ would be $\left(1 - \frac{\mu_1}{L_{\max}}\right)^T$. But because of the constraints, we can not repeatedly select the coordinate coressponding to $L_{\max}$, which means that the maximum value of $\prod_{t=1}^{T}\left(1 - \frac{\mu_1}{L_{i_t}}\right)$ might be lower than $\left(1 - \frac{\mu_1}{L_{\max}}\right)^T$.

To find the worst case bound, equivalently, if we take logarithm from both sides of (1.1), we need to find the maximum value of $\sum_{t=1}^{T} \log\left(1 - \frac{\mu_1}{L_{i_t}}\right)$, which is the MSD problem with graph G and weight function $W(v_i) = \log\left(1 - \frac{\mu_1}{L_{v_i}}\right)$ where v_i is the vertex corresponding to the coordinate i in graph G.

1.2 Greedy Kaczmarz Method[2]

Large scale linear systems of equations have many applications in machine learning, including least-squares problems, Gaussian processes, and graph-based semi supervised learning. The Kaczmarz method is an iterative algorithm for solving consistent linear system of equations of the form $Ax = b$.

At each step t, the Kaczmarz method selects a row i_t of the matrix A based on a "*selection rule*" and projects the current point x^t onto the hyperplane corresponding to the row i_t, i.e. the hyperplane $a_{i_t}^\top x = b_{i_t}$, to obtain the point x_{t+1} for the next step. Note that by $a_i^\top$ we mean the ith row of the matrix A.

There are several selection rules for the Kaczmarz method, such as cyclic, randomized, and greedy. Here we consider a particular greedy selection rule, named "*maximum residual*" rule that selects the row i_t according to

$$i_t = \underset{i}{\operatorname{argmax}} \, |a_i^\top x^t - b_i|.$$

and then updates the point x^t by projecting x^t onto the hyperplane corresponding to the row i_t according to the formula

$$x^{t+1} = x^t + \frac{b_{i_t} - a_{i_t}^\top x^t}{||a_{i_t}||^2} a_{i_t}.$$

In general, computing this greedy selection rule exactly is too computationally expensive, but there are several cases in which we can compute it efficiently. For example if A is sparse, in [2] the authors show an efficient way to compute the maximum residual rule. Their method is based on the fact that when we select a row j that is orthogonal to row i, then the residual with respect to row i will not change. So if the matrix A is sparse and many rows are orthogonal to each other, then at each step, many residuals remain the same and we do not need to recompute them. For simplicity, the authors defines a graph named "*orthogonality graph*" $G = (V, E)$, based on

[2]This section is based on [2]

the concept of orthogonal rows in matrix A. In graph G we have a vertex v_i corresponding to each row i of the matrix A. There is an edge between two vertices v_i and v_j if and only if the corresponding rows i and j in the matrix A are not orthogonal. So we only need to update the residual corresponding to a row i, if a neighbor of vertex v_i in graph G is selected.

In [2], the authors show that the convergence rate of this method is

$$||x^t - x^*||^2 \leq \left[\prod_{t=1}^{T} \left(1 - \frac{\sigma(A, \infty)}{||a_{i_t}||^2} \right) \right] ||x^0 - x^*||^2, \tag{1.2}$$

where x^* is the optimal solution, and $\sigma(A, \infty)$ is the Hoffman-like constant of matrix A. Furthermore, after we have selected the row i, the maximum residual rule will never select it again until one of its neighbors in the orthogonality graph has been selected. This is because when a row i is selected at step t, the residual of row i, i.e. $a_i^\top x^t - b_i$ becomes 0, and as we discussed, it will not be changed until one of the neighbors of i in the orthogonality graph has been selected. So the maximum residual rule will not select row i again until after one of its neighbors have been selected.

We can find the worst case convergence rate for the algorithm, we need to find the maximum value of $\prod_{t=1}^{T} \left(1 - \frac{\sigma(A,\infty)}{||a_{i_t}||^2} \right)$ in (1.2) subject to the constraint that a row will never be selected again until after one of its neighbors in the orthogonality graph has been selected. Note that if we do not have any constraints, then the maximum value of $\prod_{t=1}^{T} \left(1 - \frac{\sigma(A,\infty)}{||a_{i_t}||^2} \right)$ would be $\left(1 - \frac{\sigma(A,\infty)}{\max_i\{||a_i||^2\}} \right)^T$. But because of the constraints, we can not repeatedly select the row coressponding to $\max_i\{||a_i||^2\}$, which means that the maximum value of $\prod_{t=1}^{T} \left(1 - \frac{\sigma(A,\infty)}{||a_{i_t}||^2} \right)$ might be lower than $\left(1 - \frac{\sigma(A,\infty)}{\max_i\{||a_i||^2\}} \right)^T$.

To find the worst case bound, equivalently, if we take logarithm from both sides of (1.2), we need to find the maximum value of $\sum_{t=1}^{T} \log \left(1 - \frac{\sigma(A,\infty)}{||a_{i_t}||^2} \right)$, which is the MSD problem with graph G and weight function $W(v_i) = \log \left(1 - \frac{\sigma(A,\infty)}{||a_{i_t}||^2} \right)$ where v_i is the vertex corresponding to the row i in graph G.

Chapter 2

The Maximum Weight Sequence with Dependency Constraints Problem

2.1 The Problem

In this chapter, we try to solve the MSD problem, introduced in chapter 1.

2.2 Notations

Because the total number of sequences of vertices with length T is finite, for each T, there is at least one sequence with the highest average weight.

Let $V = \{v_1, v_2, ..., v_n\}$ where $n = |V|$. We define the binary vector $\mathbf{s}^t = (s^t_{v_1}, s^t_{v_2}, ..., s^t_{v_n})$ as the state of our structure at time t such that

$$\mathbf{s}^t = (s^t_{v_1}, s^t_{v_2}, ..., s^t_{v_n}), \quad \text{where} \quad s^t_{v_i} = \begin{cases} 1 & \text{vertex } v_i \text{ is selectable} \\ 0 & \text{vertex } v_i \text{ is not selectable} \end{cases}, \tag{2.1}$$

Note that a vertex is selectable, either from the beginning, or because one of it's neighbors have been selected. Also, note that when a vertex is selected, it becomes not selectable, until one of it's neighbors is selected.

For an arbitrary finite sequence of vertices $\mathcal{V} = \{v_{i_t}\}_{t=a}^{b}$, we define the average weight of the sequence as

$$W(\mathcal{V}) = \frac{\sum_{t=a}^{b} W(v_{i_t})}{\sum_{t=a}^{b} 1}. \tag{2.2}$$

We define

$$W_{\max} = \max\{W(v_i)|\ v \in V\}. \tag{2.3}$$

So for any sequence $\mathcal{V}$ of vertices V, we have

$$W(\mathcal{V}) \leq W_{\max}. \tag{2.4}$$

We denote the number of appearances of a vertex $v_i \in V$ in sequence $\mathcal{V}$ as $\text{count}(\mathcal{V}, v_i)$.

We denote the length of sequence $\mathcal{V}$ with $|\mathcal{V}|$.

For two sequences $\mathcal{V}_1$ and $\mathcal{V}_2$, we denote the sequence obtained by concatenating the sequence $\mathcal{V}_2$ to $\mathcal{V}_1$ as $\mathcal{V}_1\mathcal{V}_2$.

We define the set $\mathbb{F}(G)$ as the set of all *valid finite* sequences of vertices with respect to graph G. By valid, we mean that we can begin from some initial state $\mathbf{s}$ and perform the sequence with the constraint of MSD problem. By finite, we mean that it's length is finite.

We define the set $\mathbb{F}_{\max}(G, T) \subseteq \mathbb{F}(G)$ as the set of all valid, finite sequences with the highest average weight with length T that can be started from the initial state $\mathbf{s} = \mathbf{1}$. Note that in the state $\mathbf{s} = \mathbf{1}$, all vertices are selectable. Note that because we are assuming that the graph G has at least one edge, we have $\mathbb{F}_{\max}(G, T) \neq \varnothing$, as we can have valid sequences with any lengths, because we can repeat two adjacent vertices indefinitely.

We define $W(\mathbb{F}_{\max}(G, T))$ as the average weight of sequences in $\mathbb{F}_{\max}(G, T)$.

We define $\mathbb{C}(G) \subseteq \mathbb{F}(G)$ as the set of all *cyclical* sequences. By cyclical, we mean that it is valid and finite, and from some initial state $\mathbf{s}$, we can begin and repeat the sequence indefinitely. Also note that because $E \neq \varnothing$, then $\mathbb{C}(G) \neq \varnothing$.

We define $W_{\max}(\mathbb{C}(G)) = \max\{W(\mathcal{V})|\ \mathcal{V} \in \mathbb{C}(G)\}$. Note that $W_{\max}(\mathbb{C}(G)) \neq \infty$ because of (2.4).

We define $\mathbb{C}_{\max}(G)$ as the set of all cyclical sequences with the highest average weight among all cyclical sequences. Note that if $E \neq \varnothing$, then as $\mathbb{C}(G) \neq \varnothing$, we have $\mathbb{C}_{\max}(G) \neq \varnothing$.

For set of vertices $U \subseteq V$, we define $G(U)$ as the sub-graph of G whose vertices are U. Similarly, for a sequence of vertices $\mathcal{V}$, we define $G(\mathcal{V})$ as the sub-graph of G whose vertices have been appeared in the sequence. By $\text{diam}(G)$, we mean the diameter of the graph G. By $N(G, v)$ we mean the set of all neighbors of vertex v in graph G.

Let $\mathbf{U}_2$ be the collection of all sets of vertices such that their corresponding sub-graph is connected and has diameter 1 or 2, in other words, the collection of all star sub-graphs. Formally, let

$$\mathbf{U}_2 = \{U|\ U \subseteq V,\ G(U) \text{ is connected},\ 0 < \text{diam}(G(U)) \leq 2\}. \tag{2.5}$$

Note that if $E = \varnothing$, then $\mathbf{U}_2 = \varnothing$.

For each set of vertices $U \subseteq V$, we define a binary vector $\mathbf{e}_U$ denoting membership in U. Formally, let

$$\mathbf{e}_U = (e_1, e_2, ..., e_n), \quad \text{where} \quad e_i = \begin{cases} 1 & v_i \in U \\ 0 & v_i \notin U \end{cases}. \tag{2.6}$$

2.3 Solution

To solve the MSD problem, we prove several useful lemmas related to the problem.

In the next lemma, we show that if a sequence of vertexes is a combination of several smaller sequences, then at least one of the smaller sequences must have a higher average weight than the original sequence.

Lemma 2.1. *Let $\mathcal{V}$, and $\mathcal{V}_1, \mathcal{V}_2, ..., \mathcal{V}_m$ be sequences of vertices. If for every vertex $v \in V$ we have*

$$\operatorname{count}(\mathcal{V}, v) = \sum_{i=1}^{m} \operatorname{count}(\mathcal{V}_i, v), \tag{2.7}$$

then there is a sequence $\mathcal{V}_i$ such that

$$W(\mathcal{V}_i) \geq W(\mathcal{V}). \tag{2.8}$$

Proof. Because of (2.7), we have

$$W(\mathcal{V}) \sum_{i=1}^{m} |\mathcal{V}_i| = \sum_{i=1}^{m} W(\mathcal{V}_i)|\mathcal{V}_i| \;\Rightarrow\; \sum_{i=1}^{m} |\mathcal{V}_i| = \sum_{i=1}^{m} \frac{W(\mathcal{V}_i)}{W(\mathcal{V})}|\mathcal{V}_i|.$$

Assume for all $\mathcal{V}_i$ we have $M_{\mathcal{V}_i} < M_{\mathcal{V}}$. This yields $\sum_{i=1}^{m} |\mathcal{V}_i| < \sum_{i=1}^{m} |\mathcal{V}_i|$, which is a contradiction. So the result holds. □

Note that because of the constraints of the MSD problem, we can not select a vertex repeatedly. In the next lemma, we show that there is an upper bound on the number of times that a vertex can be selected in a sequence.

Lemma 2.2. *Let $\mathcal{V} \in \mathbb{F}(G)$ be a sequence that can be started from some initial state* $\mathbf{s}$*. Then for all vertices $v \in V$ we have*

$$\operatorname{count}(\mathcal{V}, v) \leq s_v + \sum_{u \in N(G,v)} \operatorname{count}(\mathcal{V}, u). \tag{2.9}$$

Proof. When vertex v is selected, it must be in selectable state. This means it was selectable from the beginning or one of its neighbor was selected. So the result is correct.

□

Note that in the lemma 2.2, the upper bound depends on the initial state. In the next lemma, we show that if the sequence is cyclical, i.e. it can be repeated indefinitely, then we can have an upper bound not depending on the initial state. This lemma will allow our proofs to not depend on the initial state.

Lemma 2.3. *If $\mathcal{V} \in \mathbb{C}(G)$, then for all vertices $v \in V$ we have*

$$\operatorname{count}(\mathcal{V}, v) \leq \sum_{u \in N(G,v)} \operatorname{count}(\mathcal{V}, u). \tag{2.10}$$

Proof. Because $\mathcal{V} \in \mathbb{C}(G)$, we can repeat the sequence $\mathcal{V}$ indefinitely. Consider repeating $\mathcal{V}$ twice beginning from some initial state $\mathbf{s}$. We denote this new sequence by $\mathcal{V}^2$.

As $\mathcal{V}^2 \in \mathbb{F}(G)$, from lemma 2.2, we can conclude that for all vertices $v \in V$ we have

$$\operatorname{count}(\mathcal{V}^2, v) \leq s_v + \sum_{u \in N(G,v)} \operatorname{count}(\mathcal{V}^2, u).$$

As $s_v \in \{0, 1\}$ and $\operatorname{count}(\mathcal{V}^2, v) = 2 \operatorname{count}(\mathcal{V}, v)$, we have

$$\begin{aligned} 2 \operatorname{count}(\mathcal{V}, v) &\leq 1 + \sum_{u \in N(G,v)} 2 \operatorname{count}(\mathcal{V}, u) \\ &< 2 + \sum_{u \in N(G,v)} 2 \operatorname{count}(\mathcal{V}, u). \end{aligned}$$

Dividing by 2, we have

$$\operatorname{count}(\mathcal{V}, v) < 1 + \sum_{u \in N(G,v)} \operatorname{count}(\mathcal{V}, u),$$

which yields our result, as $\operatorname{count}(\mathcal{V}, v)$ is always an integer.

□

The next lemma, is a core lemma, that when is combined with other lemmas, will show that we can decompose a valid cyclical sequence into several smaller cyclical sequences. Furthermore, the sub-graphs formed by each of these smaller cyclical sequences, is a star sub-graphs.

Lemma 2.4. *Let c_v be a non-negative integer associated with each vertex $v \in V$, and let $\mathbf{c} = (c_{v_1}, c_{v_2}, ..., c_{v_n})$ be the associated vector. Suppose that, for all $v \in V$,*

$$c_v \leq \sum_{u \in N(G,v)} c_u. \tag{2.11}$$

Let $\mathbf{U}_2$ be the set defined in (2.5). Then we can assign a non-negative integer a_S to each $S \in \mathbf{U}_2$ such that,

$$\mathbf{c} = \sum_{S \in \mathbf{U}_2} a_S \mathbf{e}_S. \tag{2.12}$$

Proof. If $\mathbf{c} = \mathbf{0}$, then for all $S \in \mathbf{U}_2$, we can assign $a_S = 0$ to satisfy (2.11).

So assume that $\mathbf{c} \neq \mathbf{0}$. As $\mathbf{c} \neq \mathbf{0}$, there must be a vertex x such that $c_x > 0$. Because of (2.11), for at least one of the neighbors of x such as y, we must have $c_y > 0$, because otherwise (2.11) will be violated for vertex x as the left-hand side is non-zero and right-hand side is zero. So there must be an edge $\{x, y\} \in E$ such that $c_x > 0,\ c_y > 0$.

Consider

$$L = \sum_{v \in V} c_v. \tag{2.13}$$

We use induction on L to prove the lemma.

First we prove the lemma is correct for $L = 2$. As we argued, there are two neighbor vertices x, y, such that $c_x > 0, c_y > 0$. As $L = 2$, we must have $c_x = 1, c_y = 1$ and for all other vertices v, we must have $c_v = 0$. Let $S^* = \{u, v\}$. As $\mathbf{c} = \mathbf{e}_{S^*}$ and $S^* \in \mathbf{U}_2$, by setting $a_{S^*} = 1$ and $a_S = 0$ for all other sets $S \in \mathbf{U}_2$ that $S \neq S^*$, we can satisfy (2.12). So the lemma is correct for $L = 2$.

Assume that the lemma holds for $L = 2, ..., k-1$. We show that the lemma holds for $L = k$.

For any vector $\mathbf{c} \in \mathbb{N}^n$ and graph G satisfying (2.11), we define the *remainder* operator $\mathrm{rem}_G(\mathbf{c}) = (r_{v_1}, r_{v_2}, ..., r_{v_n})$ such that for every $v \in V$ we have

$$r_v = -c_v + \sum_{u \in N(G,v)} c_u. \tag{2.14}$$

We can see that (2.11) is satisfied if and only if for all vertices $v \in V$, we have

$$r_v \geq 0. \tag{2.15}$$

Let $V_1 = \{v |\ c_v \geq 1\}$ and $G_1 = G(V_1)$. Let $r_{\min} = \min\{r_v |\ v \in V_1\}$. We divide the problem into different cases based on the value of $r_{\min}$. In each case we find some set $S^* \in \mathbf{U}_2$ such that the vector $\mathbf{c}' = \mathbf{c} - \mathbf{e}_{S^*}$ satisfies the constraint (2.11). To do this we show that all elements of vector $\mathbf{r}' = \mathrm{rem}_G(\mathbf{c}')$ are non-negative.

Note that because we have

$$c'_v = \begin{cases} c_v & v \notin S^* \\ c_v - 1 & v \in S^* \end{cases}, \tag{2.16}$$

we have

$$r'_v = \begin{cases} r_v - |S^* \cap N(G,v)| & v \in V - S^* \\ r_v - |S^* \cap N(G,v)| + 1 & v \in S^* \end{cases}. \tag{2.17}$$

For vertices $v \in V$ that $c_v = 0$, it is clear that $r'_v \geq 0$ is satisfied. So we should only consider the vertices in V_1. For vertices $v \in V_1 - S^*$ that don't have a neighbor in S^*, we have $S^* \cap N(G,v) = \varnothing$, so $r'_v = r_v \geq 0$. So we only need to prove that for all the vertices v in S^* or neighbors of S^* in V_1, $r' \geq 0$. We divide the problem into three cases: $r_{\min} = 0$, $r_{\min} = 1$ and $r_{\min} \geq 2$.

- **Case 1** ($r_{\min} = 0$):

  Consider a vertex $x \in V_1$ such that $r_x = 0$. As $c_x > 0$, then because of (2.11), x should have some neighbor in V_1, say y. Now consider the set $N_0(y) = \{v |\ v \in V_1 \cap N(G,y),\ r_v = 0\}$. We choose $S^* = \{y\} \cup N_0(y)$, which is in $\mathbf{U}_2$. Note that $x \in N_0(y)$, so there are some vertices other than y in S^*.

  - **Claim**: For all vertices $v \in N_0(y)$, we have $r'_v \geq 0$.

    Proof. First we prove that there are no two neighbor vertices $u, v \in N_0(y)$. By way of contradiction, assume $u, v \in N_0(y)$ are neighbors. Because $r_u = 0$ and $v \in N(G,u)$, we have $c_u \geq c_v$. Because $r_v = 0$ and $u \in N(G,v)$, we have $c_v \geq c_u$. So $c_v = c_u$. But as $u, v \in N(G,y)$, we have $r_v > 0, r_u > 0$ which contradicts the fact that $r_u = 0, r_v = 0$.

    So there are no two neighbor vertices $u, v \in N_0(y)$. So for all vertices $x \in N_0(y)$, y is their only neighbor in S^*. So for all vertices $v \in N_0(y)$, we have $|S^* \cap N(G,v)| = 1$ and because $N_0(y) \subseteq S^*$, based on (2.17), we have $r'_v = r_v = 0$. □

  - **Claim**: For all vertices $v \in V_1$ that have some neighbors in $N_0(y)$ (including y itself), we have $r_v \geq 0$.

Proof. Consider a vertex v that is neighbor of a vertex $u \in N_0(y)$. Because $r'_u = 0$, we have $c'_u \geq c'_v$, so $r'_v \geq 0$. $\square$

- **Claim**: For all neighbors v of y with $v \notin N_0(y)$, we have $r'_v \geq 0$.

  Proof. Note that $r_v \geq 1$, because if $r_v = 0$, then based on the definition of $N_0(y)$, we would have $v \in N_0(y)$ which contradict our assumption that $v \notin N_0^y$. If v has a neighbor in $N_0(y)$, we showed in previous claim that $r_v \geq 0$. If v is not a neighbor of any vertices of $N_0(y)$, then $|S^* \cap N(G, v)| = 1$ and because $v \in V - S^*$, then based on (2.17), $r'_v = r_v - 1$ and because $r_v \geq 1$, we have $r'_v \geq 0$. $\square$

- **Case 2** ($r_{\min} = 1$):

  We divide this case into different sub-cases.

  - **Case A**: There are no two neighbor vertices $u, v \in V_1$ such that $r_v = 1, r_u = 1$.

    Approach: We pick some vertex x such that $r_x = 1$. Then because of (2.11), x has some neighbor y such that $c_y > 0$. We choose $S^* = \{x, y\}$ which is in $\mathbf{U}_2$. Note that $r'_x = r_x \geq 0$, $r'_y = r_y \geq 0$. For all vertices v outside of S^* that are connected to S^*, if $|S^* \cap N(G, v)| = 1$, then as $v \in V - S^*$ and $r_v \geq r_{\min} = 1$, based on (2.17) we have $r'_v \geq 0$. If $|S^* \cap N(G, v)| = 2$, then because v is neighbor of x, then $r_v \geq 2$, otherwise our assumption will be violated. So based on (2.17) we have $r'_v \geq 0$.

  - **Case B**: There are two neighbor vertices $x, y \in V_1$ such that $r_x = 1, r_y = 1$.

    * **Case (i)**: For all $v \in V_1 - \{x, y\}$ connected to both of x, y, we have $r_v \geq 2$.

      Approach: In this case we choose $S^* = \{x, y\}$ which is in $\mathbf{U}_2$. We have $r'_x = r_x \geq 0$ and $r'_y = r_y \geq 0$. For all vertices v connected to one of x, y, as $r_v \geq r_{\min} = 1$ and $|S^* \cap N(G, v)| = 1$, we have $r'_v \geq 0$ based on (2.17). For vertices v connected to both of x, y we have $r_v \geq 2$, and as $|S^* \cap N(G, v)| = 2$ we have $r'_v \geq 0$ based on (2.17).

    * **Case (ii)**: There is some vertex $z \in V_1 - \{x, y\}$ connected to both of x, y, with $r_z = 1$.

Approach: In this case, using $r_x = 1$ and

$$c_x = -r_x + \sum_{u \in N(G,x)} c_u = -1 + c_y + c_z + \sum_{u \in N(G,x)-\{y,z\}} c_u,$$

as $c_z > 0$, we have $c_x \geq c_y$. Using a similar argument, we have $c_x \leq c_y$. So we have $c_x = c_y$. Similarly we can prove $c_x = c_z$. So $c_x = c_y = c_z$.
We choose $S^* = \{x, y, z\}$, which is in $\mathbf{U}_2$. We claim that x, y, z are not connected to any other vertex in V_1. For the sake of contradiction, assume that there is a vertex v connected to x. So we have

$$r_x = -c_x + c_y + c_z + \sum_{u \in N(G,x)-\{y,z\}} c_u = c_z + \sum_{u \in N(G,x)-\{y,z\}} c_u. \tag{2.18}$$

As v is a neighbor of x, we have

$$\sum_{u \in N(G,x)-\{y,z\}} c_u > 0,$$

and as $c_z > 0$, based on (2.18) we have $r_x > 1$, which is a contradiction. So $\{x, y, z\}$ has no neighbor in V_1 and based on (2.17) we have $r'_x = r'_y = r'_z = 0$ because $r_x = r_y = r_z = 1$.

- **Case** ($r_{\min} \geq 2$):

  As argued before, there are two neighbor vertices $x, y \in V_1$ because of (2.11). We choose $S^* = \{x, y\}$ which is in $\mathbf{U}_2$. Then we have $r'_x = r_x \geq 0$ and $r'_y = r_y \geq 0$. For all other vertices $v \in V_1 - S^*$, we have $|S^* \cap N(G, v)| \leq 2$. As $r_v \geq 2$, by (2.17) we have $r'_v \geq 0$.

So we proved that all vertices of S^* and neighbors of S^* in V_1 has non negative r' value. So we have shown that the vector $\mathbf{c}'$ satisfies the condition of (2.11). We assumed that the lemma is true for $L = 2, ..., k-1$. As $\sum_{u \in V} c'_u < L$, the lemma is true for vector $\mathbf{c}'$, so we have $\mathbf{c}' = \sum_{S \in \mathbf{U}_2} a'_S \mathbf{e}_S$. As $\mathbf{c} = \mathbf{c}' + \mathbf{e}_S$, we have our result. □

In the next lemma we show that if in the initial state, at least one of the vertexes of a star sub-graph is selectable, then we can form a cyclical sequence from the vertexes of the star sub-graph and repeat them indefinitely.

Lemma 2.5. *Let $S \in \mathbf{U}_2$, and $\mathbf{s} = (s_{v_1}, s_{v_2}, ..., s_{v_n})$ be a state such that for at least one of the vertices $x \in S$ we have $s_x = 1$. Then there exists a cyclical sequence $\mathcal{V} \in \mathbb{C}(G)$ with the following properties.*

- *For every vertex $v \in S$,* $\text{count}(\mathcal{V}, v) = 1$.
- *For every vertex $v \notin S$,* $\text{count}(\mathcal{V}, v) = 0$.
- *The sequence $\mathcal{V}$ can be started from the initial state* $\mathbf{s}$.

Proof. If $|S| = 2$, assume $S = \{x, y\}$. Based on the definition of $\mathbf{U}_2$ in 2.5, $G(S)$ is connected, so there is an edge between x and y. Let $\mathcal{V} = (x, y)$. Note that we can repeat this sequence indefinitely from state $\mathbf{s}$. For every vertex $v \in S$, $\text{count}(\mathcal{V}, v) = 1$, and for all other vertices $v \notin S$, $\text{count}(\mathcal{V}, v) = 0$. So the lemma holds for $|S| = 2$.

If $|S| > 2$, based on the definition of $\mathbf{U}_2$ in 2.5, there must be a vertex $y \in S$ such that the degree of y in the sub-graph $G(S)$ is greater than 1. Assume $S = \{x, y, z_1, z_2, ..., z_m\}$. Note that in the sub-graph $G(S)$, the degree of all vertices $\{z_1, z_2, ..., z_m\}$ is 1. If $x \neq y$, we set $\mathcal{V} = (x, y, z_1, z_2, ..., z_m)$, otherwise $\mathcal{V} = (x, z_1, z_2, ..., z_m)$. Note that we can repeat this sequence indefinitely from state $\mathbf{s}$. For every vertex $v \in S$, $\text{count}(\mathcal{V}, v) = 1$, and for all other vertices $v \notin S$, $\text{count}(\mathcal{V}, v) = 0$. So the lemma holds for $|S| > 2$. □

In the next lemma, we show that if a vertex of a star sub-graph is selectable, then in all subsequent steps, at least one of the vertexes of the star sub-graph will be selectable.

Lemma 2.6. *Let $S \in \mathbf{U}_2$ and $\mathbf{s}^0$ be an initial state in which for at least one of the vertices $v \in S$ we have $s^0_v = 1$. Suppose $\mathcal{V} \in \mathbf{F}(G)$ is a valid sequence that can be started from the initial state $\mathbf{s}^0$. Let $\mathbf{s}^1$ be the state after performing the sequence $\mathcal{V}$. Then there exists at least one vertex $v \in S$ such that $s^1_v = 1$.*

Proof. We use induction on the length of $\mathcal{V}$.

When $|\mathcal{V}| = 0$, the lemma clearly holds.

Assume the lemma holds for $|\mathcal{V}| = 0, 1, ..., k-1$. We show that it holds for $|\mathcal{V}| = k$.

So suppose $|\mathcal{V}| = k$. Assume the last vertex of $\mathcal{V}$ is x and the rest is the sequence $\mathcal{V}'$. Suppose after performing the sequence $\mathcal{V}'$, we are in state $\mathbf{s}'$. As $\mathcal{V}'$ is a valid sequence with length $k-1$, and the lemma is correct for the sequences with length $k-1$, there exists a vertex $y \in S$ such that $s'_y = 1$.

Now we choose vertex x to complete the sequence $\mathcal{V}$. If $x \neq y$, then after choosing x, we still have $s'_y = 1$. If $x = y$, note that based on the definition of $\mathbf{U}_2$, $G(S)$ is connected, so y has a neighbor, say $z \in S$. So after choosing y, z is selectable. As $z \in S$, so the lemma also holds in this case.

□

In the next lemma, we show how we can use lemma 2.4 to decompose a valid cyclical sequence into several smaller cyclical sequences.

Lemma 2.7. *Let c_v be a non-negative integer associated with each vertex $v \in V$, and let $\mathbf{c} = (c_{v_1}, c_{v_2}, ..., c_{v_n})$ be the associated vector. Suppose that, for all $v \in V$, (2.11) holds. Then there exists a valid cyclical sequence $\mathcal{V} \in \mathbb{C}(G)$ such that for all vertices $v \in V$, $\text{count}(\mathcal{V}, v) = c_v$. Furthermore, $\mathcal{V} = \mathcal{V}_1\mathcal{V}_2...\mathcal{V}_m$, where for all $1 \leq i \leq m$, we have $\mathcal{V}_i \in \mathbb{C}(G)$, $\text{diam}(G(\mathcal{V}_i)) \leq 2$, and for all vertices $v \in V$, $\text{count}(\mathcal{V}_i, v) \leq 1$.*

Proof. Using lemma 2.4, we can assign a non-negative integer a_S to each set $S \in \mathbf{U}_2$, such that (2.12) holds. Consider all sets $S \in \mathbf{U}_2$ with non-zero a_S. We denote these sets with $S_1, S_2, ..., S_l$.

To construct $\mathcal{V}$, we begin with the initial state $\mathbf{s}^0 = \mathbf{1}$. From lemma 2.5, we can find a cyclical sequence $\mathcal{S}_1$, corresponding to the set S_1, such that all vertices of S_1 are appeared in $\mathcal{S}_1$ exactly once, and no other vertex is appeared in it, and it can be started from the state $\mathbf{s}^0$. We repeat $\mathcal{S}_1$ for a_{S_1} times and set $\mathcal{V}_1, ..., \mathcal{V}_{a_{S_1}}$ to $\mathcal{S}_1$.

After repeating $\mathcal{S}_1$ for a_{S_1} times, suppose we are in state $\mathbf{s}^1$. By lemma 2.6, for all sets S_i, $i = 1, ..., l$, there exists a vertex v such that $s^2_v = 1$. By lemma 2.5, there exists a cyclical sequence $\mathcal{S}_2$, corresponding to S_2. We repeat $\mathcal{S}_2$ for a_{S_2} times.

Similarly, for each set S_i, $i = 1, ..., l$, we can continue and repeat a corresponding cyclical sequence $\mathcal{S}_i$ for a_{S_i} times. Because $\mathbf{c}$ satisfies 2.12, for all vertices $v \in V$ we have $\text{count}(\mathcal{V}, v) = c_v$. Note that based on lemma 2.5, each vertex v, appears at most once in $\mathcal{S}_i$. As $S_i \in \mathbf{U}_2$, $\text{diam}(G(\mathcal{S}_i)) \leq 2$. So the constructed sequence $\mathcal{V}$ satisfies all the properties required by the lemma. □

In the next lemma, we put together all of the previous lemmas to conclude that there is a cyclical sequence with the highest average weight whose vertexes form a star sub-graph. Hence, to find a cyclical sequence with the highest average weight, we can just limit our search to star sub-graphs.

Theorem 2.1. *If for graph $G = (E, V)$, $E \neq \varnothing$, then there exists a cyclical sequence with the highest average weight $\mathcal{V}^* \in \mathbb{C}_{\max}(G)$ such that* $\text{diam}(G(\mathcal{V}^*)) \leq 2$ *and for all vertices $v \in V$,* $\text{count}(\mathcal{V}, v) \leq 1$.

Proof. Because $E \neq \varnothing$, $\mathbb{C}_{\max}(G) \neq \varnothing$. So there exists a sequence $\mathcal{V} \in \mathbb{C}_{\max}(G)$. From lemma 2.3, this implies (2.10). We construct a vector $\mathbf{c} = (c_{v_1}.c_{v_2}, ..., c_{v_n})$, such that for all vertices $v \in V$, $c_v = \text{count}(\mathcal{V}, v)$. Under this construction, $\mathbf{c}$ satisfies (2.11). So based on lemma 2.4 we have (2.12). By lemma 2.7, $\mathcal{V} = \mathcal{V}_1\mathcal{V}_2...\mathcal{V}_m$, where for all $\mathcal{V}_i$, $i = 1, ..., m$, $\text{diam}(G(\mathcal{V}_i)) \leq 2$ and for all vertices v, $\text{count}(\mathcal{V}_i, v) \leq 1$. By lemma 2.1, there is some $\mathcal{V}_j$ such that $W(\mathcal{V}_j) \geq W(\mathcal{V})$. Note that as $\mathcal{V} \in \mathbb{C}_{\max}(G)$, for all i, we have $W(\mathcal{V}_i) \leq W(\mathcal{V})$, but as $W(\mathcal{V}_j) \geq W(\mathcal{V})$ we must have $W(\mathcal{V}_j) = W(\mathcal{V})$. So the result holds for $\mathcal{V}^* = \mathcal{V}_j$. □

Based on theorem 2.1, we can find cyclical sequences with the highest average weight in graphs efficiently. The basic idea is to search over all subgraphs of graph G that has diameter less than or equal to 2, and pick the one with the highest average weight.

Algorithm 1: Finding a cyclical sequence with the highest average weight in graphs.

Data: A graph $G = (E, V)$ and a weight function $W : V \to \mathbb{R}$
Result: A cyclical sequence $\mathcal{V}^* \in \mathbb{C}_{\max}(G)$ with the highest average weight.

Sort vertices of V by their weight in descending order;
for *All vertices* $v_i \in V$ **do**
    $\mathcal{V}_{v_i} \leftarrow (v_i)$;
end
for *All vertices* $v_i \in V$ *from highest weight to lowest* **do**
    for *All vertices* $v_j \in N(G, v_i)$ **do**
        if $W(v_i) \geq W(\mathcal{V}_{v_j})$ **then**
            Insert v_i to $\mathcal{V}_{v_j}$;
        end
    end
end
$v^* \leftarrow \text{argmax}_{v \in V} W(\mathcal{V}_v)$;
Return $\mathcal{V}_{v^*}$;

Theorem 2.2. *Given a graph $G = (V, E)$ and a weight function $W : V \to \mathbb{R}$, The algorithm 1 returns an cyclical sequence with the highest average* *weight in time $O(|E| + |V| \log |V|)$.*

Proof. In a sub-graph with diameter of exactly 2, there is one vertex that is connected to all other vertices. We call this special vertex, the center of the sub-graph. When the diameter of a sub-graph is exactly 1, which is simply two vertices connected to each other, we can assume any of them as the center. For simplicity, we refer to sub-graphs with diameter less than or equal 2 as *star* sub-graphs.

In the algorithm, for each vertex $v_i \in V$, we find the star sub-graph with the center of v_i with the highest average weight. We put the corresponding vertices in a list named $\mathcal{V}_{v_i}$.

One *inefficient* way to find $\mathcal{V}_{v_i}$ for each vertex $v_i \in V$, might be the following approach. Sort all the neighbors of v_i by their weights in descending order and beginning from the vertex with highest weight, insert them to $\mathcal{V}_{v_i}$ as long as the insertion increase the average weight of $\mathcal{V}_{v_i}$. In order to see why this approach works, consider two cases. First, if v_i has only one neighbor, then there is only one possible star graph with center v_i, and our approach will finds it. Seconds, if v_i has more than 1 neighbors, then consider two vertices x and y that are neighbors of v_i, where $W(x) \leq W(y)$. Note that if y is in the star sub-graph with highest average weight with center v_i, then x must also be in the sub-graph, because otherwise we could replace y with x and get a star sub-graph with higher average weight. So we can insert vertices to $\mathcal{V}_{v_i}$ in descending order based on their weights.

However this approach is not the most efficient way to solve the problem. As for each vertex we sort all it's neighbors, the running time would be

$$\sum_{v_i \in V} d(v_i) \log d(v_i),$$

where $d(v_i)$ is the degree of vertex v_i. If we assume our graph is the complete graph, then the running time would be $\Theta(|V|^2 \log |V|)$.

Now consider the approach used in the algorithm 1. In the algorithm, instead of sorting the neighbors of each vertex separately, we sort all vertices once, and then we consider vertices v_i in descending order, and for each vertex v_j that is neighbor of v_i, insert v_i to $\mathcal{V}_{v_j}$ if the insertion improves the average weight of $\mathcal{V}_{v_j}$. Note that again, for each $\mathcal{V}_{v_j}$, the vertices are inserted in descending order, like previous approach, so it works correctly. The running time of the algorithm is $\Theta(|V| \log |V|)$ for sorting the vertices, and $|E|$ for considering the neighbor for each vertex. So the total running time is be $\Theta(|V| \log |V| + |E|)$, which is better than the running time of previous approach. $\square$

In the next lemma, we show that although the sequence with the highest average weight with a particular length may have a higher average than the cyclical sequence with the highest average weight, when the length goes to infinity they would have similar average weight. Thus for long enough sequences, in order to find a sequence with high average weight, we can use a cyclical sequence with the highest average weight and repeat it.

Theorem 2.3.

$$\lim_{T\to\infty} W(\mathbb{F}_{\max}(G,T)) = W(\mathbb{C}_{\max}(G)). \tag{2.19}$$

Proof. Let $\mathcal{V}_0^T = \{v_{i_t}\}_{t=1}^T \in \mathbb{F}_{\max}(G)$ and $\{\mathbf{s}^t\}_{t=1}^T$ be the corresponding sequence of states where $\mathbf{s}^1 = \mathbf{1}$. If $|\mathcal{V}_0^T| > 2^n$, then by pigeon hole principle, there must be t_1 and t_2 such that $\mathbf{s}^{t_1} = \mathbf{s}^{t_2}$. Let $\mathcal{A}_0^T = \{i_t\}_{t=1}^{t_1-1}$ and $\mathcal{B}_0^T = \{i_t\}_{t=t_1}^{t_2-1}$ and $\mathcal{C}_0^T = \{i_t\}_{t=t_2}^{T}$, so that $\mathcal{V}_0^T = \mathcal{A}_0^T\mathcal{B}_0^T\mathcal{C}_0^T$. Now because $\mathbf{s}^{t_1} = \mathbf{s}^{t_2}$, $\mathcal{V}_1^T = \mathcal{A}_0^T\mathcal{C}_0^T$ is a valid sequence. Note that $\mathcal{B}_0^T \in \mathbb{C}(G)$, so $W(\mathcal{B}_0^T) \leq W(\mathbb{C}_{\max}(G))$. If $|\mathcal{V}_1^T| > 2^n$, we repeat the process and obtain a new sequence $\mathcal{V}_2^T$. As long as $|\mathcal{V}_j^T| > 2^{|V|}$, we repeat this process until we obtain a sequence $\mathcal{V}_m^T$ such that

$$|\mathcal{V}_m^T| \leq 2^n. \tag{2.20}$$

We denote the omitted sub-sequence from $\mathcal{V}_j^k$ in step j as $\mathcal{B}_j^k$. As we argued,

$$W(\mathcal{B}_j^T) \leq W(\mathbb{C}_{\max}(G)). \tag{2.21}$$

We have

$$W(\mathcal{V}_0^T) = \frac{1}{T}\left(|\mathcal{V}_m^T| W(\mathcal{V}_m^T) + \sum_{j=0}^{m-1} |\mathcal{B}_j^k| W(\mathcal{B}_j^T)\right). \tag{2.22}$$

Combining (2.4) and (2.20) and (2.21) and (2.22), we have

$$W(\mathcal{V}_0^T) \leq \frac{1}{T}\left(2^n W_{\max} + TW(\mathbb{C}_{\max}(G))\right). \tag{2.23}$$

Let $\mathcal{V}^*$ be a sequence satisfying the conditions of theorem 2.1. We construct the new sequence $\mathcal{V}_\downarrow^*$ by sorting the elements of $\mathcal{V}^*$ by their weight in descending order. Because $\operatorname{diam}(G(\mathcal{V}^*)) \leq 2$, $\mathcal{V}_\downarrow^*$ is also a valid cycle. Now we construct the sequence $\mathcal{Z}$ by repeating $\mathcal{V}_\downarrow^*$ until we obtain a sequence with length T. Note that in the last repeat of $\mathcal{V}_{\downarrow}^{*}$, all of it's elements may not be inserted. So

$$W(\mathbb{C}_{\max}(G)) \leq W(\mathcal{Z}). \tag{2.24}$$

And because $\mathcal{V}_0^k \in \mathbb{F}_{\max}(G)$ and $|\mathcal{Z}| = T$, we have

$$W(\mathcal{Z}) \leq W(\mathcal{V}_0^T). \tag{2.25}$$

Combining (2.23), (2.24) and (2.25), we get

$$W(\mathbb{C}_{\max}(G)) \leq W(\mathcal{V}_0^T) \leq \frac{1}{T}\left(2^n W_{\max} + TW(\mathbb{C}_{\max}(G))\right). \tag{2.26}$$

Because

$$\lim_{k\to\infty} W(\mathbb{C}_{\max}(G)) = \lim_{k\to\infty} \frac{1}{T}\left(2^n W_{\max} + TW(\mathbb{C}_{\max}(G))\right) = W(\mathbb{C}_{\max}(G)),$$

by the sandwich theorem we have our result.

□

So far, we are able to find cyclical sequences with average weight $W(\mathbb{C}_{\max}(G))$, using algorithm 1. Given a sequence length T, we can repeat the cyclical sequence to generate a sequence with length T. However this approach may not be optimal. For example consider the graph shown in figure 2.1. Using the algorithm 1, we can find the cyclical sequence with the highest average weight, which is the sequence $(1, 2, 3, 4, 0)$. So if we want a sequence with length 11, by repeating the cyclical sequence with the highest average weight, we would have the sequence $(1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1)$. However, the sequence $(1, 2, 3, 4, 0, 1, 2, 3, 9, 5, 6)$ has greater average weight. However, if the length of the desired sequence, goes to infinity, by theorem 2.3, we know that the average weight of the sequence that we get, by repeating the cyclical sequence with the highest average weight, is asymptotically tight.

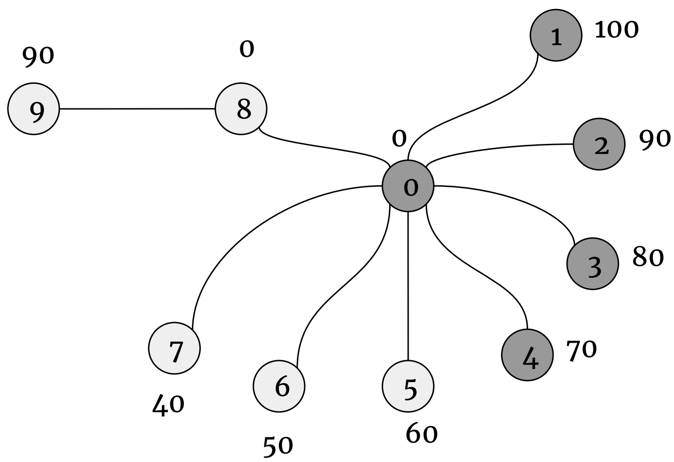

Figure 2.1: An example graph. In this figure, each circle corresponds to a vertex, the numbers insides the circles are the number of the vertex and the numbers outside of the circles are the weight corresponding to the vertex. The Grey vertices are the vertices of the cyclical sequence with the highest average weight.

Chapter 3

Some Generalizations Of The MSD Problem

In this chapter, we consider some generalizations of the the MSD problem. We use the notations introduced in section 2.2. Whenever we change the notations, we will explicitly indicate it.

3.1 The Maximum Weight Sequence with k-times Dependency Constraints Problem

The first generalization that we consider, is the following problem, which we call The Maximum weight Sequence with k-times Dependency constraints problem and abbreviate it to k-times-MSD:

We are given a graph $G = (V, E)$, a weight $W(v_i)$ associated with each vertex $v_i \in V$, and an iteration number T, and a positive integer k. Choose a sequence of vertices $\mathcal{V} = \{v_{i_t}\}_{t=1}^{T}$ that maximizes $\sum_{t=1}^{T} W(v_{i_t})$, subject to the following constraint: after k times that vertex v_i is selected, it cannot be selected again until after a neighbor of vertex v_i has been selected.

Theorem 3.1. *The k-times-MSD problem can be reduced to the MSD problem in polynomial time.*

Proof. Given a graph $G = (V, E)$ and weight function $W : V \to \mathbb{R}$, we construct a new graph $G' = (V', E')$ and weight function $W' : V' \to \mathbb{R}$, and use it as the input for the MSD problem, and we reconstruct the solution of k-times-MSD problem from the solution of MSD problem.

For every vertex $v_i \in V$, there are k vertices $v'_{i,1}, v'_{i,2}, ..., v'_{i,k}$ in V'. For every edge $\{v_i, v_j\} \in E$, we add k^2 edges to E' by connecting every vertex $v'_{i,1}, v'_{i,2}, ..., v'_{i,k}$ to every vertex $v'_{j,1}, v'_{j,2}, ..., v'_{j,k}$. We define $W'(v'_{i,l}) = W(v_i)$. Assume the sequence $\mathcal{V}'$ is the solution to the MSD with graph G' and weights W'. To reconstruct the solution of the k-times-MSD problem, denoted by $\mathcal{V}$, from $\mathcal{V}'$, we replace each vertex $v'_{i,l}$ in $\mathcal{V}'$ with v_i.

Note that the sequence $\mathcal{V}$ is valid, because a vertex in $\mathcal{V}$ can not be selected for more than k times before selecting any of it's neighbors, because otherwise we must have been selected a vertex for two times before selecting any of it's neighbors in $\mathcal{V}'$ which contradicts the assumption that $\mathcal{V}'$ is a valid sequence for the MSD problem. Also note that the average weight of $\mathcal{V}$ and $\mathcal{V}'$ is the same.

We claim that $\mathcal{V}$ is an optimal solution for k-times-MSD problem, if $\mathcal{V}'$ is an optimal solution for the MSD problem.

Assume to the contrary that $\mathcal{V}$ is not optimal. So there exists another sequence $\mathcal{U}$ which is optimal for the k-times-MSD problem. From $\mathcal{U}$, we construct a sequence $\mathcal{U}'$ for the MSD problem, with the same average weight of $\mathcal{U}$.

Whenever a vertex v_i is selected for the j^{th} time in the sequence $\mathcal{U}$ before selecting any of it's neighbors, we select the vertex $\sqsubseteq'_{\rangle,|}$ in sequence $\mathcal{U}'$. Sequence $\mathcal{U}'$ is valid, because no vertex in $\mathcal{U}'$ is selected for two times before selecting any of it's neighbors. Because otherwise, we must have been selected v_i for more than k times before selecting any of it's neighbors. Also note that the average weight of $\mathcal{U}$ and $\mathcal{U}'$ is the same. As the average weight of $\mathcal{V}$ and $\mathcal{V}'$ were the same, and average weight of $\mathcal{U}$ is higher than $\mathcal{V}$, the average weight of $\mathcal{U}'$ must be higher than $\mathcal{V}'$ which contradicts the fact that $\mathcal{V}'$ were optimal.

So we have found an optimal sequence $\mathcal{V}$ for the k-times-MSD problem by solving a k-times-MSD problem. □

3.2 The Maximum Weight Sequence with k-order Dependency Constraints Problem

The second generalization that we consider, is the following problem, which we call the Maximum weight Sequence with k-order Dependency constraints, and abbreviate it to k-order-MSD:

We are given a graph $G = (V, E)$, a weight $W(v_i)$ associated with each vertex $v_i \in V$, and an iteration number T, and a positive integer k. Choose a sequence of vertices $\mathcal{V} = \{v_{i_t}\}_{t=1}^{T}$ that maximizes $\sum_{t=1}^{T} W(v_{i_t})$, subject to the following constraint: after each time that vertex v_i is selected, it cannot be selected again until after a vertex v_j has been selected, such that the length of the shortest path between v_i and v_j, is less than or equal k.

Theorem 3.2. *The k-order-MSD problem can be reduced to the MSD problem in polynomial time.*

Proof. Given a graph $G = (V, E)$ and weight function $W : V \to \mathbb{R}$, we construct a new graph $G' = (V', E')$ and weight function $W' : V' \to \mathbb{R}$, and use it as the input for the MSD problem, and we show how to find the solution of k-order-MSD problem from the solution of MSD problem.

For every vertex $v_i \in V$, we add one vertex v'_i to V'. For every two vertices $v_i \in V$ and $v_j \in V$, if there is a path with length less than or equal k, we add the edge $\{v'_i, v'_j\}$ to E'. For every $v'_i \in V'$, we set $W'(v'_i) = W(v_i)$.

Assume the sequence $\mathcal{V}'$ is the solution to the MSD with graph G' and weights W'. To construct the solution of the k-order-MSD problem, denoted by $\mathcal{V}$ from $\mathcal{V}'$, we replace each vertex v'_i in $\mathcal{V}'$ with v_i.

Note that the sequence $\mathcal{V}$ is a valid sequence for the k-order-MSD problem. Because whenever a vertex v_i is selected in the sequence $\mathcal{V}$, one of the vertices that it's distance is not more than k must have been selected before. Because one of the vertices that are neighbor of v'_i, say v'_j must have been selected, so v_j whose distance from v_i is not more than k must have been selected in $\mathcal{V}$. Also note that the weight of $\mathcal{V}$ and $\mathcal{V}'$ is equal.

We claim that $\mathcal{V}$ is an optimal solution for k-order-MSD problem, if $\mathcal{V}'$ is an optimal solution for the MSD problem.

Assume to the contrary, that $\mathcal{V}$ is not optimal. So there is another sequence $\mathcal{U}$ which is optimal for the k-order-MSD problem. We construct a valid sequence $\mathcal{U}'$ for the MSD problem. For each vertex v_i that appears in the sequence $\mathcal{U}$, we replace v_i with v'_i to obtain the sequence $\mathcal{U}'$. Note that $\mathcal{U}'$ is valid, because whenever a vertex v'_i is selected in $\mathcal{U}'$, one of it's neighbors must have been selected before v'_i. Because when v_i is selected in $\mathcal{U}$, a vertex v_j that has a distance not more than k must have been selected before v_i. So the vertex v'_j which is a neighbor of v'_i must have been selected before v'_i. Also note that the average weight of the sequence $\mathcal{U}$ and $\mathcal{U}'$ is equal. So the average weight of $\mathcal{U}'$ is higher than the average weight of $\mathcal{V}'$ which contradicts the assumption that $\mathcal{V}'$ was optimal for the MSD problem.

So we have found an optimal sequence $\mathcal{V}$ for the k-order-MSD problem by solving a MSD problem. □

3.3 The Maximum Weight Sequence with k-neighbors Dependency Constraints problem

The third generalization that we consider, is the following problem, which we call the maximum weight Sequence with k-neighbors dependency constraints problem, and abbreviate it to k-neighbors-MSD:

We are given a graph $G = (V, E)$, a weight $W(v_i)$ associated with each vertex $v_i \in V$, and an iteration number T, and a positive integer k. Choose a sequence of vertices $\mathcal{V} = \{v_{i_t}\}_{t=1}^{T}$ that maximizes $\sum_{t=1}^{T} W(v_{i_t})$, subject to the following constraint: after each time vertex v_i is selected, it cannot be selected again until after k neighbors of vertex v_i has been selected.

Note that if $k = 1$, then this problem would be the MSD problem. So we focus on the case that $k \geq 2$.

3.3.1 Notations

In this section, we redefine some of the notations, introduced in section 2.2. For all notations that we don't explicitly redefine, we use the same notation introduced in section 2.2.

We define the state vector $\mathbf{s}^t = (s^t_{v_1}, s^t_{v_2}, ..., s^t_{v_n})$ as the state of our structure at time t such that $s_{v_i} \in [k]$ indicates the number of vertices adjacent to v_i that are selected since the last time v_i is selected. So, once $s_{v_i} = k$, the vertex v_i becomes selectable.

We define the set $\mathbb{F}(G, k)$ as the set of all *valid finite* sequences of vertices with respect to graph G. By valid, we mean that we can begin from some initial state $\mathbf{s}$ and perform the sequence with the constraint of k-neighbors-MSD problem. By finite, we mean that it's length is finite.

We define the set $\mathbb{F}(G, k) \subseteq \mathbb{F}(G, k)$ as the set of all valid, finite sequences that can be started from some initial state $\mathbf{s}$.

We define the set $\mathbb{F}(G, k, \mathbf{s}) \subseteq \mathbb{F}(G, k)$ as the set of all valid, finite sequences that can be started from the initial state $\mathbf{s}$.

We define the set $\mathbb{F}_{\max}(G, k) \subseteq \mathbb{F}(G, k)$ as the set of all valid, finite sequences with the highest average weight that can be started from some initial state $\mathbf{s}$.

We define $\mathbb{C}(G, k) \subseteq \mathbb{F}(G, k)$ as the set of all *cyclical* sequences. By cyclical, we mean that it is valid and finite, and from some initial state $\mathbf{s}$, we can begin and repeat the sequence indefinitely.

We define $W_{\max}(\mathbb{C}(G, k)) = \max\{W(\mathcal{V})| \ \mathcal{V} \in \mathbb{C}(G, k)\}$. Note that $W_{\max}(\mathbb{C}(G, k)) \neq \infty$ because of (2.4).

We define $\mathbb{C}_{\max}(G, k)$ as the set of all cyclical sequences with the highest average weight.

Consider the set of vectors $\mathbf{c} = (c_{v_1}, c_{v_2}, ..., c_{v_n}) \in \mathbb{N}^n$, in which for every vertex $v \in V$, there is a corresponding non-negative integer c_v. Let $\mathbf{H}(G)$ be the subset of this set of vectors, such that for all vectors $\mathbf{c} \in \mathbf{H}(G)$, and for all vertices $v \in V$ we have

$$k c_v \leq \sum_{u \in N(G,v)} c_u. \tag{3.1}$$

We define the set $\mathbf{U}(G,k)$ as the set of vectors in $\mathbf{H}(G)$, such that it is not the sum of two other vectors of $\mathbf{H}(G)$. Formally

$$\mathbf{U}(G,k) = \{\mathbf{c} | \ \mathbf{c} \in \mathbf{H}(G), \nexists \mathbf{c}', \mathbf{c}'' \in \mathbf{H}(G),\ \mathbf{c} = \mathbf{c}' + \mathbf{c}'',\ \mathbf{c}' \neq \mathbf{0},\ \mathbf{c}'' \neq \mathbf{0}\}. \tag{3.2}$$

3.3.2 Solution

In this section, we try an approach, similar to the one introduced in section 2.3.

The next lemma, is the lemma 2.2 counterpart.

Lemma 3.1. *Let $\mathcal{V} \in \mathbb{F}(G,k)$ be a sequence that can be started from some initial state $\mathbf{s}$. Then for all vertices $v \in V$ we have*

$$k \operatorname{count}(\mathcal{V}, v) \leq s_v + \sum_{u \in N(G,v)} \operatorname{count}(\mathcal{V}, u). \tag{3.3}$$

Proof. The proof is very similar to the proof of lemma 2.2.

□

The next lemma, is the lemma 2.3 counterpart.

Lemma 3.2. *If $\mathcal{V} \in \mathbb{C}(G,k)$, then for all vertices $v \in V$ we have*

$$k \operatorname{count}(\mathcal{V}, v) \leq \sum_{u \in N(G,v)} \operatorname{count}(\mathcal{V}, u). \tag{3.4}$$

Proof. The proof is very similar to the proof of lemma 2.3. Because $\mathcal{V} \in \mathbb{C}(G,k)$, we can repeat the sequence $\mathcal{V}$ indefinitely. Consider repeating $\mathcal{V}$ for $k+1$ times, beginning from some initial state $\mathbf{s}$. We denote this new sequence by $\mathcal{V}^{k+1}$.

As $\mathcal{V}^{k+1} \in \mathbb{F}(G,k)$, from lemma 3.1, we can conclude that for all vertices $v \in V$ we have

$$k \operatorname{count}(\mathcal{V}^{k+1}, v) \leq s_v + \sum_{u \in N(G,v)} \operatorname{count}(\mathcal{V}^{k+1}, u).$$

As $s_v \in [k]$ and $\operatorname{count}(\mathcal{V}^{k+1}, v) = k+1 \operatorname{count}(\mathcal{V}, v)$, we have

$$\begin{aligned} k(k+1) \operatorname{count}(\mathcal{V}, v) \leq k + & \sum_{u \in N(G,v)} (k+1) \operatorname{count}(\mathcal{V}, u) \\ & < (k+1) + \sum_{u \in N(G,v)} (k+1) \operatorname{count}(\mathcal{V}, u). \end{aligned}$$

Dividing by $k+1$, we have

$$k \,\mathrm{count}(\mathcal{V}, v) < 1 + \sum_{u \in N(G,v)} \mathrm{count}(\mathcal{V}, u),$$

which yields our result, as $\mathrm{count}(\mathcal{V}, v)$ is always an integer.

□

To generalize lemma 2.4, we propose the following conjecture.

Conjecture 3.1. *Let* $\mathbf{c} \in \mathbf{H}(G)$. *Then we can assign a non-negative integer* $a_{\mathbf{e}}$ *to each* $\mathbf{e} \in \mathbf{U}(G,k)$ *such that,*

$$\mathbf{c} = \sum_{\mathbf{e} \in \mathbf{U}(G,k)} a_{\mathbf{e}} \mathbf{e}. \tag{3.5}$$

Remember that lemma 2.4, was the key step in finding a cyclical sequence with the highest average weight for the problem MSD. We might think that if we could prove the conjecture 3.1, then we can apply a similar approach that we used in theorem 2.1 and algorithm 1 to solve the problem k-neighbors-MSD. However, proving the conjecture will not help us in developing an algorithm, as even deciding whether a vector $\mathbf{e} \in \mathbf{U}(G,k)$ is NP-complete.

Theorem 3.3. *The problem of deciding if a vector* $\mathbf{e}$ *is not in* $\mathbf{U}(G,k)$*, is NP-complete, for* $k \geq 2$.

Proof. Note that the problem is in NP, because when $\mathbf{e} \notin \mathbf{U}(G,k)$, then we can find two vectors $\mathbf{e}'$ and $\mathbf{e}''$ such that

$$\begin{aligned} \mathbf{e} &= \mathbf{e}' + \mathbf{e}'', \\ \mathbf{e}',\ \mathbf{e}'' &\in \mathbf{H}(G). \end{aligned} \tag{3.6}$$

Given such $\mathbf{e}'$ and $\mathbf{e}''$ as certificate, we can easily check in polynomial time if they satisfy the conditions (3.6).

We assume $k = 2$. In the end of the proof, we will show how to change the reduction to handle the case that $k > 2$.

We reduce the *3SAT* problem to deciding $\mathbf{e} \notin \mathbf{U}(G,k)$. In the *3SAT* problem, we are given a *3-cnf-formula* ϕ with Boolean variables $x_1, x_2, ..., x_m$ and we want to decide if ϕ is satisfiable. To reduce the *3SAT* problem to the problem of deciding $\mathbf{e} \notin \mathbf{U}(G,2)$, we construct a graph $G = (E,V)$ and a vector $\mathbf{e}$, such that

$$\phi \text{ is satisfiable } \Leftrightarrow \mathbf{e} \notin \mathbf{U}(G,2). \tag{3.7}$$

Let m be the number of variables of ϕ and l be the number of clauses.

Now we show how to construct the graph G from ϕ. First we add vertices and edges according to figure 3.1. Note that in the figure, for each clause i, we have a corresponding vertex C_i, and for each variable x_i, we have two vertices t_i and f_i. We refer to the vertices of the left connected component as *left vertices* and to the vertices of the right connected component as *right vertices* (note that these are not all of the vertices, as we will add some other vertices soon).

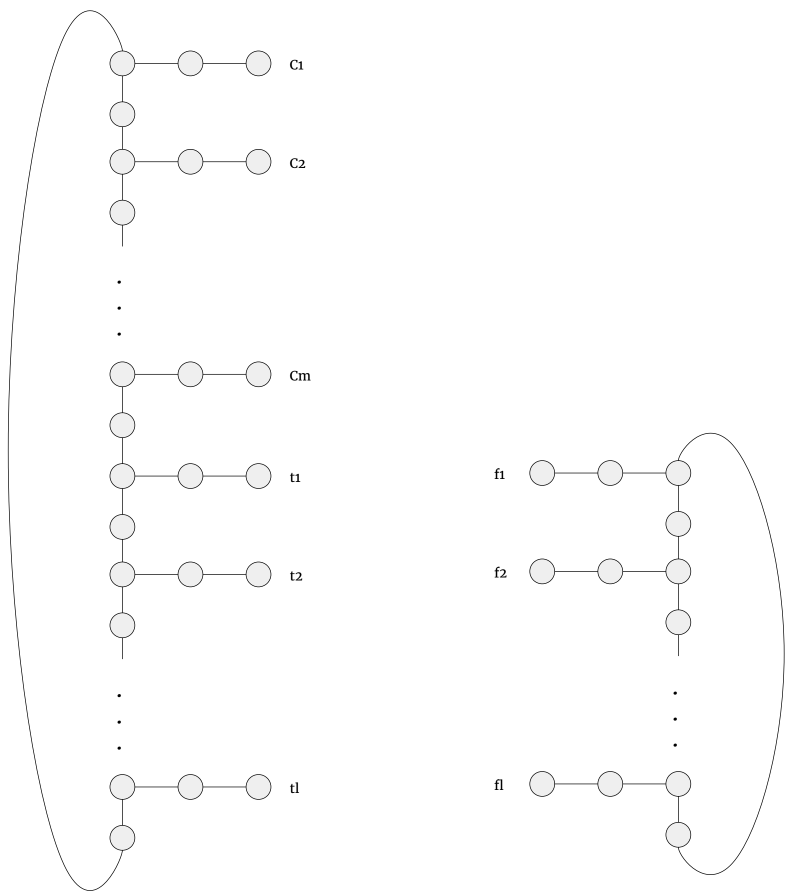

Figure 3.1: The vertices and edges corresponding clauses.

Then for each variable x_i, we add vertices and edges according to figure 3.2.

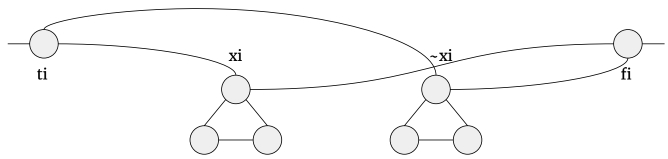

Figure 3.2: The vertices and edges corresponding variables.

We refer to these vertices as *middle vertices*.

Note that for each literal x_i and $\neg x_i$, we have a vertex, and they are connected to vertices t_i and f_i, the same vertices shown in figure 3.1.

Finally, for each clause i, we connect the vertex C_i to the vertices corresponding to the literals that appears in the clause.

We can see an example in figure 3.3

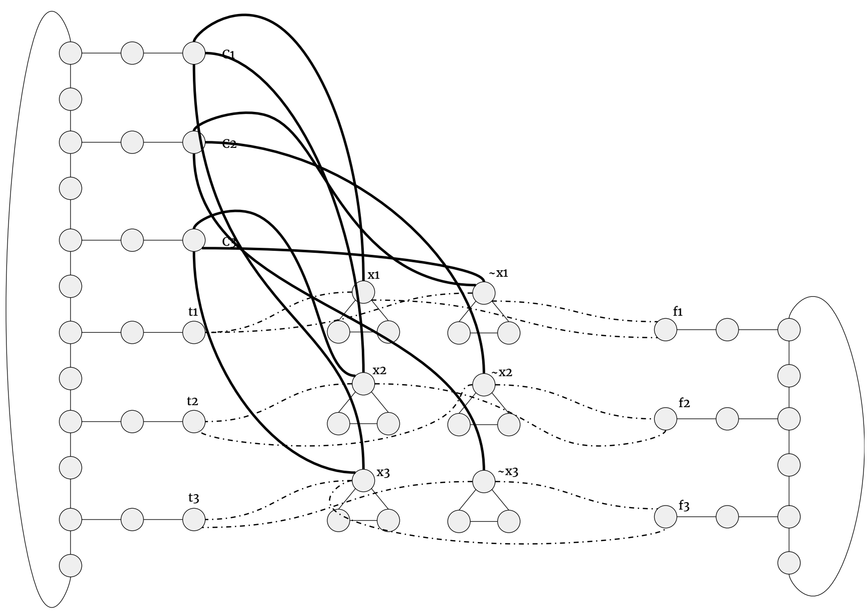

Figure 3.3: An example graph for reduction from *3SAT* to the problem of deciding $\mathbf{e} \notin \mathbf{U}(G, 2)$, where $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$.

Now our graph $G = (V, E)$ is completed. We set $\mathbf{e} = \mathbf{1}$.

To prove (3.7), first assume that ϕ is satisfiable. So there is an assignment for variables $x_1, ..., x_m$ such that ϕ is satisfied. We show that we can find two vectors $\mathbf{e}'$ and $\mathbf{e}''$ such that the condition (3.6) is held, which means that $\mathbf{e} \notin \mathbf{U}(G, 2)$. To construct $\mathbf{e}'$ and $\mathbf{e}''$, we color the vertices of the graph with colors BLUE and RED, such that each vertex has at least 2 neighbors with the same color. For all vertices $v \in V$, we set

$$e'_v = \begin{cases} 1 & \text{Color of } v \text{ is BLUE} \\ 0 & \text{Color of } v \text{ is RED} \end{cases}, \; e''_v = \begin{cases} 1 & \text{Color of } v \text{ is RED} \\ 0 & \text{Color of } v \text{ is BLUE} \end{cases}. \tag{3.8}$$

We color all left vertices with BLUE and all right vertices with RED. For middle vertices, if $x_i = \text{TRUE}$ then we color all vertices in the triangle corresponding to x_i with BLUE and all the vertices in the triangle corresponding to $\neg x_i$ with RED. If $x_i = \text{FALSE}$ then we color all vertices in the triangle corresponding to x_i with RED and all the vertices in the triangle corresponding to $\neg x_i$ with BLUE.

Note that for vertices other than $C_1, ..., C_l$ and $t_1, ..., t_m$ and $f_1, ..., f_m$, obviously each vertex has at least 2 neighbors with the same color.

For vertices t_i and f_i, as one of the triangles corresponding to x_i or $\neg x_i$ is colored with BLUE, and the other one with RED, t_i and f_i also has two neighbors with the same color.

For vertices C_i, as each clause of ϕ has at least one TRUE literal, and the triangle corresponding to the literal is colored with BLUE, and C_i is connected to the triangle, C_i has at least two neighbors with the same color.

So we have colored the vertices of the graph G with two colors, such that each vertex has at least two neighbors with the same color, and both colors are used. So the two vectors $\mathbf{e}'$ and $\mathbf{e}''$ obtained from the coloring, satisfy the condition 3.6. So $\mathbf{e} \notin \mathbf{U}(G, 2)$.

So assuming ϕ is satisfiable, we proved that $S \notin \mathbf{U}(G, 2)$.

Conversely assume that $\mathbf{e} \notin \mathbf{U}(G, 2)$. So we can find two vectors $\mathbf{e}'$ and $\mathbf{e}''$ such that the condition (3.6) is held. As $\mathbf{e} = \mathbf{1}$, so all elements of vectors $\mathbf{e}'$ and $\mathbf{e}''$ are either 0 or 1. We define the sets of vertices $U' = \{v | \; e'_v = 1\}$ and $U'' = \{v | \; e''_v = 1\}$. As $\mathbf{e} = \mathbf{e}' + \mathbf{e}'$, we have partitioned the set V into two set U' and U''. Note that all left vertices must be in the same partition, because otherwise we would have a vertex v_1 with degree 2 in one partition connected to a vertex v_2 in another partition. So it is not possible for v_1 to have two neighbors in the same partition, which violates the condition 3.6. Similarly, all right vertices are in the same partition. Furthermore, for each triangle corresponding to literals, all vertices of the triangle are in the same partition. Note that The left vertices and right vertices are not in the same partition. Because otherwise, then all triangles corresponding to the literals must also be in the same partition of the left and right vertices, because otherwise the vertices $t_1, ..., t_m$ or $f_1, ..., f_m$ would not have 2 neighbors in the same partition.

Without loss of generality, assume all left vertices are in U' and all right vertices are in U''. We color all vertices of U' with BLUE and all vertices of U'' with RED.

For each vertex x_i, if its color is BLUE, we assign the Boolean variable $x_i =$ TRUE, otherwise $x_i =$ FALSE.

We claim that with this assignment of the Boolean variables $x_1, ..., x_m$, ϕ must be satisfied. Note that the color of the vertices $C_1, ..., C_l$ is BLUE. Each vertex C_i, has at least two neighbors with the same color. So there must be at least one vertex x_j with color BLUE connected to C_i. So the corresponding clause of ϕ is satisfied, as $x_j =$ TRUE. So all clauses of ϕ is satisfied.

So we proved that assuming $\mathbf{e} \notin \mathbf{U}(G, 2)$, ϕ is satisfiable.

So (3.7) is correct and we can reduce the *3SAT* problem to $\mathbf{e} \notin \mathbf{U}(G, 2)$. Note that the reduction is polynomial as the number of vertices of the constructed graph G is $O(l + m)$.

So the problem of deciding if a $\mathbf{e} \notin \mathbf{U}(G, k)$ is NP-complete for $k = 2$. If $k > 2$, we can construct the same graph G described for the case $k = 2$, but add $2(k-2)$ special vertices $V^* = \{v_1^*, v_2^*, ..., v_{2(k-2)}^*\}$, and connect all vertices of V^*, to all vertices. Then we set $\mathbf{e} = \mathbf{1}$.

We must prove that

$$\phi \text{ is satisfiable } \Leftrightarrow \mathbf{e} \notin \mathbf{U}(G, k). \tag{3.9}$$

First, if ϕ is satisfiable, ignoring vertices of V^*, we color the vertices of G as described for the case $k = 2$. We color $k - 2$ vertices of V^* with BLUE and the other half with red. As based on this coloring for the case $k = 2$, every vertex had at least 2 vertices with the same color, now as vertices of V^* are connected to all vertices, every vertex has at least $2 + k - 2 = k$ neighbors with the same color.

Conversely, if $\mathbf{e} \notin \mathbf{U}(G, k)$, then we can find $\mathbf{e}'$ and $\mathbf{e}''$ such that they satisfy 3.6. We color the vertices based on $\mathbf{e}'$ and $\mathbf{e}''$ similar to the way we did for the case $k = 2$. Note that exactly $k - 2$ vertices of V^* must be BLUE and the other half must be RED. Because for the left vertices that are BLUE, there are some vertices with degree exactly 2, ignoring the vertices of V^* that are neighbor of every vertex. So these vertices must have at least $k - 2$ BLUE neighbors in V^*. So at least half of vertices of V^* must be blue. Also note that in the right vertices that are RED, there are some vertices with degree 2, ignoring the vertices of V^* that are neighbor of every vertex.

With the same argument that we used, we can conclude that at least half of vertices of V^* must be RED. So exactly half of vertices of V^* must be BLUE and the other half must be RED. As each vertex has at least k neighbors with the same color and exactly $k-2$ neighbors with the same color in V^*, every vertex has at least 2 neighbors with the same color in $V - V^*$. Hence we can use the same method that used for the case $k = 2$ to find a satisfying assignment for ϕ.

So we have shown how to reduce the *3SAT* problem to $\mathbf{e} \notin \mathbf{U}(G, k)$ for all $k \geq 2$. $\square$

Corollary 3.1. *The problem of deciding if a set* $\mathbf{e} \in \mathbf{U}(G, k)$*, is co-NP-complete.*

Unless P = NP, because of the corollary 3.1, even if the conjecture 3.1 is correct, it is not possible to apply the approach of algorithm 1 to get a polynomial time algorithm for the k-neighbors-MSD problem. However, we may think that there might be some other approach that works for the k-neighbors-MSD problem. But in the following theorem, we prove that the k-neighbors-MSD problem itself is NP-complete, at least for $k \geq 3$.

Theorem 3.4. *Consider the decision problem of k-neighbors-MSD, in which we have to decide whether there is a valid sequence of vertices that can be started from a given initial state* $\mathbf{s}$*, with average weight of at least a given number W_d. This problem is NP-complete for $k \geq 3$.*

Proof. First note that this problem is in NP, because if we have the sequence as the certificate, we can easily check in polynomial time that if it is a valid sequence of vertices with average weight of at least W_d.

We reduce the *3SAT* problem to the decision problem of k-neighbors-MSD. In the *3SAT* problem, we are given a *3-cnf-formula* ϕ with Boolean variables $x_1, x_2, ..., x_m$ and we want to decide if ϕ is satisfiable. To reduce the *3SAT* problem to the decision problem of k-neighbors-MSD, we construct a graph $G = (E, V)$, a weight function $W : V \to \mathbb{R}$, an initial state $\mathbf{s}$, and a value W_d, such that

$$\phi \text{ is satisfiable } \Leftrightarrow \text{ the } k\text{-neighbors-MSD has a solution with weight at least } W_d. \tag{3.10}$$

Let m be the number of variables of ϕ and l be the number of clauses.

Now we show how to construct the graph G from ϕ.

For each clause of ϕ, we add a clause gadget with 3 vertices with the initial states and weights as follows, such that the weight of c'_i, c''_i and c'''_i is $-4ml$ and the weight of c_i is $4ml$.

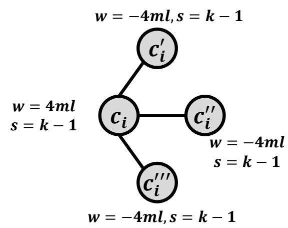

Figure 3.4: The clause gadget, with it's vertices, edges, weights and states.

Then considering the vertices $c_1, ..., c_l$ from clause gadgets, we construct a binary tree with leaves $c_1, ..., c_l$. We set the state of all of the inner vertices of the binary tree to $k-2$ and the weight of all of them except the root to 0, and the weight of the root to 1. So after adding the vertices of the binary tree, the graph look likes the following figure.

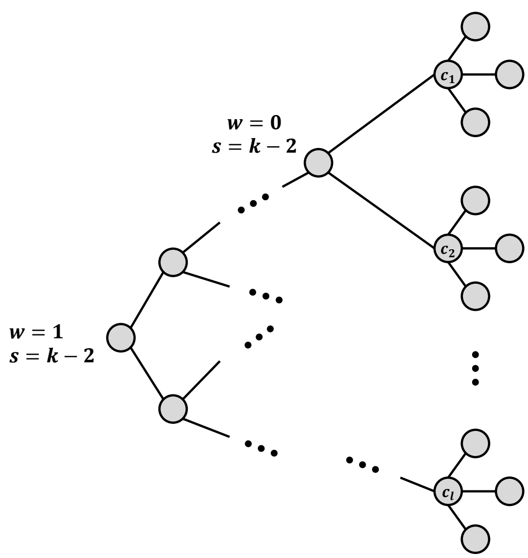

Figure 3.5: The binary tree constructed on vertices coressponding to the clauses.

For each variable x_i, we add a variable gadget with $2l+1$ vertices, as follows, such that the initial state of vertices $x_{i,1}$ and $\bar{x}_{i,1}$ is k and all other vertices are $k-1$, and the weight of x_i^+ is $2l$ and all other vertices are -2.

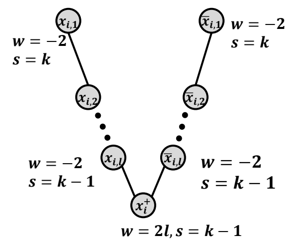

Figure 3.6: The variable gadget, with it's vertices, edges, weights and states.

Each clause of ϕ contains 3 literals. For each literal x_i appearing in clause j, we connect the vertex $x_{i,j}$ to one of the vertices c'_j, c''_j and c'''_j, such that each of c'_j, c''_j and c'''_j is connected to exactly one vertex coressponding to the literals. For example if vertex c_1 corresponds to the clause $(x_1 \vee \bar{x}_2 \vee x_3)$, then we will add the edges shown in the following figure.

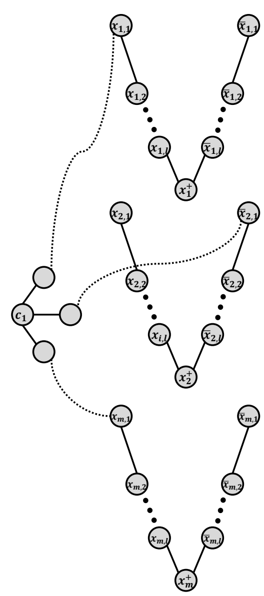

Figure 3.7: The edges corresponding to the literals of the clause $(x_1 \vee \bar{x}_2 \vee x_3)$ is shown with dotted lines.

So the whole graph G will look like the following figure.

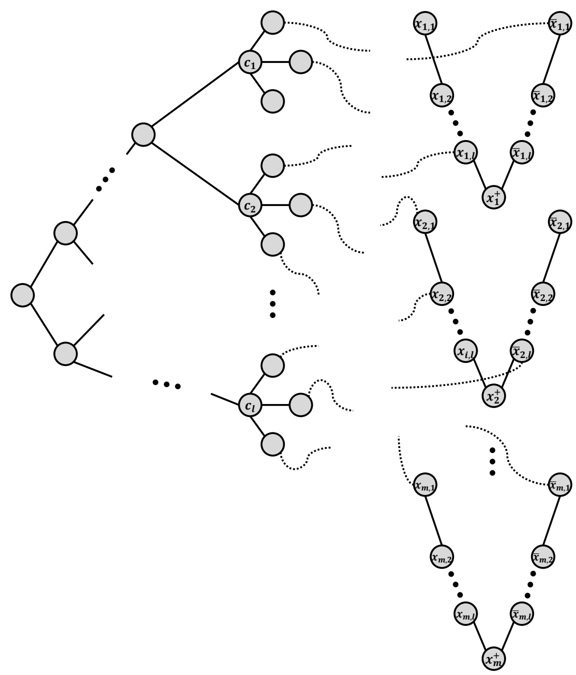

Figure 3.8: The complete graph

We set $W_d = 0$. We claim that

$$\phi \text{ is satisfiable} \Leftrightarrow \exists \mathcal{V} \in \mathbb{F}(G,k),\ W(\mathcal{V}) > W_d \tag{3.11}$$

First we try to prove

$$\phi \text{ is satisfiable} \Rightarrow \exists \mathcal{V} \in \mathbb{F}(G,k),\ W(\mathcal{V}) > W_d.$$

So assume that ϕ is satisfiable, so there is an assignment to the Boolean variables $x_1, ..., x_m$ such that ϕ is satisfied. From this assignment of values to variables $x_1, ..., x_m$, we construct a sequence $\mathcal{V} \in \mathbb{F}(G,k)$ such that $W(\mathcal{V}) \geq W_d$.

For each variable x_i, we have $2l+1$ vertices, as shown in figure 3.6. If the Boolean variable x_i = TRUE, we select vertices labeled $x_{i,1}, ..., x_{i,l}$ otherwise we select vertices labeled $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$, in $\mathcal{V}$. Then we select the vertex with label x_i^+ in $\mathcal{V}$. Because of the initial state of the vertices, this sequence is valid. Note that the sum of the weights of the selected vertices is 0. For each clause i, there is a literal a_i = TRUE, where a_i is a literal like x_j or $\bar{x}_j$. So among vertices with label c_i', c_i'' and c_i''', at least one of them, say c_i', is connected to a vertex corresponding to a literal which is TRUE. As the state of c_i' was initially $k-1$ and one of it's neighbors is selected, then it is selectable. We select c_i' (note that if among c_i', c_i'' and c_i''', more than one of them are selectable, we just select one of them) in $\mathcal{V}$. Because of this selection, c_i becomes selectable, and we select c_i in $\mathcal{V}$. Note that the sum of weights of c_i' and c_i is 0. As all clauses are satisfiable, all vertices c_i are selected. So all vertices of the binary tree, can be selected, by beginning from the leaves and going up to the root.

Now the sequence $\mathcal{V}$ is completed. To calculate the sum of weight of the vertices of the sequence, note that the sum of weight of all vertices other than the vertices of the binary tree are 0 and the sum of vertices of the binary tree is 1. So the average weight of the sequence is positive. So assuming that ϕ is satisfiable, we have found a valid seuqunce $\mathcal{V}$ with positive weight.

Now we try to prove

$$\exists \mathcal{V} \in \mathbb{F}(G,k),\ W(\mathcal{V}) > W_d \Rightarrow \phi \text{ is satisfiable.}$$

So assume that there exists a valid sequence $\mathcal{V}$ such that the sum of weights of vertices of $\mathcal{V}$ is positive. We prove that ϕ is satisfiable.

Let V_c be the set of vertices of the clause gadgets, shown in figure 3.4, except vertices $c_1, ..., c_l$. Let V_x be the set of all vertices of the variable gadgets shown in figure 3.6. Let V_t be the set of all vertices of the binary tree. Note that we have $V_t = V - (V_x \cup V_c)$. We claim that the vertices of $V_x \cup V_c$, must be selected at most once in $\mathcal{V}$. For the sake of contradiction, assume at least one of the vertices of $V_x \cup V_c$ is selected more than once.

Let u_2 be the first vertex of $V_x \cup V_c$ that is selected for the second time in $\mathcal{V}$. First consider the case that u_2 is a vertex of a V_x. Before u_2 is selected, each of it's neighbors are selected for at most once. If the initial state of u_2 is k, then it has 2 neighbors that are currently selected for at most once. So if u_2 is selected for the second time, then the equation (3.3) from lemma 3.1 would be violated, as the left hand side would be $2k$ and the right hand side would be at most 4, and we have $k \geq 3$. If the initial state of u_2 is $k-1$, then it has at most 3 neighbors, and as all of them are currently selected for at most once, again the lemma 3.1 would be violated.

Now consider the case that u_2 is a vertex of a V_c. Before reaching a contradiction for this case, we prove a claim.

We claim that if all vertices of V_c are selected for at most once, then all vertices of V_t must be selected for at most 2 times. For the sake of contradiction, assume at least one of the vertices of V_t is selected for more than 2 times. Let u_3 be a vertex of the binary tree with the highest depth that is selected for more than 2 times. Assume u_3 is selected for g times where $g \geq 3$. Based on the definition of u_3, the two children of u_3 must be selected for at most $g-1$. Based on lemma 3.1, the parent of u_3 must be selected for at least $g+1$ times. Let u_4 be the parent of u_3. Again using lemma 3.1, we can conclude that the parent of u_4 must be selected for at least $g+2$ times. Repeating this argument, we can conclude that if the root is selected for h times, where $h \geq g$, then the two children of the root must be selected for at most $h-1$ times. As the root doesn't have any other neighbors, lemma 3.1 would be violated for the root, which is a contradiction.

Now we backtrack to reach a contradiction for the case that u_2 is a vertex of a V_c. Consider the second time that u_2 is selected. One step before this selection, all vertices of the V_c are selected once. So all vertices of the binary tree must be selected for at most 2 times. As u_2 has one neighbor in V_x which is selected for at most once, and one neighbor in V_t which is selected for at most 2 times, 3.1 would be violated if u_2 is selected for the second time.

Note that the root of the binary tree can be selected for at most once. Because if it is selected for more than once, then as it's two children can be selected for at most 2 times, then lemma 3.1 would be violated.

We claim that for every clause gadget i, shown in figure 3.4, it is not possible to select a vertex c_i without selecting at least one of the vertices c'_i, c''_i, c'''_i beforehand. Assume to the contrary, that we can select c_i without selecting any of c'_i, c''_i, c'''_i beforehand. So the parent of c_i in the binary tree must be selected before the selection of c_i. Let p_1 be the parent of c_i. Consider the first time that p_1 is selected. Consider the child of p_1 which is not c_i. We name it p'_1. It must be selected for at most once, before selection of p_1. Because if we assume that it was selected for 2 times, then if p'_1 is an inner vertex of the binary tree, as it's two childern can be selected for at most 2 times, lemma 3.1 would be violated. If p'_1 is a leaf of the binary tree, the 3 vertices connected to it other than it's parent can be selected for at most once, so again lemma 3.1 would be violated. So based on 3.1, the parent of p_1, which we denote it by p_2, must be selected before p_1. Now consider the first time that p_2 is selected. We can repeat the same argument that we used for p_1, to conclude that the parent of p_2, denoted by p_3 must be selected before p_2. If we repeat this argument, then we can conclude that the root of the binary tree must be selected before at least one of it's children. But from our argument, we also know that the other child of the root must be selected for at most once. So based on lemma 3.1, it is not possible to select the root, which is a contradiction.

We claim that for each clause gadget i, shown in figure 3.4, the sum of weight gained by selecting vertices c_i, c_i', c_i'', c_i''' can not be positive. Assume to the contrary that it is positive. So the vertex c_i must be selected at least once, as it is the only vertex with positive weight among the vertices of the clause gadget. As we argued before, c_i can be selected for at most 2 times and c_i', c_i'', c_i''' for at most 1 time. If c_i is selected for 2 times, then as it's parent in the binary tree can be selected for at most 2 times, based on lemma 3.1, all of c_i', c_i'', c_i''' must be selected, but this would result in negative sum of weights of the vertices c_i, c_i', c_i'', c_i'''. If c_i is selected for 1 time, then as we argued in the previous paragraph, at least one of c_i', c_i'', c_i''' must be selected before c_i. So the sum of the weights of c_i, c_i', c_i'', c_i''' can be at most 0 which is not positive.

We claim that for each variable gadget shown in figure 3.6, the sum of weights of the vertices selected from $x_{i,1}, ..., x_{i,l}$, $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$, and x_i^+ can not be positive. Assume to the contrary that it is positive. As x_i^+ is the only vertex with positive weight, it must be selected. So at least one of it's neighbors must be selected before x_i^+. We denote the neighbor by $y_{i,l}$. $y_{i,l}$ was selectable because one of it's neighbors was selected before it. The neighbor is either a vertex of the variable gadget, or a vertex of a clause gadget. If the neighbor is a vertex of the variable gadget, we denote it by $y_{i,l-1}$. We repeat this process and find the neighbor of $y_{i,l-1}$ and repeat this as long as the neighbor is also a vertex of the variable gadget. In the end two case can happen. Either we reach the vertex $y_{i,1}$ or we reach a vertex like c_j' (without loss of generality) from some clause gadget. If we reach the vertex $y_{i,1}$, as the vertices $y_{i,1}, ..., y_{i,l}$ must have been selected, the sum of weights of the selected vertices of the variable gadget can not be positive, which is a contradiction. If c_j' was selected before some vertex $y_{i,p}$, then c_j' must have become selectable because of selecting c_j. As we argued before, if c_j is selected, then at least one of the vertices c_j', c_j'', c_j''' must have been selected before c_j. So at least two neighbors of c_j are selected. Note that based on weights of c_j and it's neighbors, c_j must be selected for 2 times. Because if it is selected for 1 time, then the sum of weights of the sequence can not be positive. Note that we proved that the sum of weights of vertices of each of the clause gadgets can not be positive. For the other vertices with positive weights, we proved that they can be selected for at most once. Now note that the sum of weights of these vertices is less than the negative sum of weights of the clause gadget. So the total sum of weights in the sequence would be negative which is a contradiction. So c_j must be selected for 2 times. In this case, exactly two vertices from c_j', c_j'', c_j''' must have been selected. Because otherwise, the sum of weights of the sequence would be negative. So based on lemma 3.1, the parent of c_j must have been selected for 2 times, before selecting c_j for the second time. Let p_1 be the parent of c_j. So when p_1 was selected for the second time, c_j was selected for at most 1 time. Based on lemma 3.1, the other two neighbors of p_1 was selected for 2 times. So the parent of p_1, denoted by p_2 must have been selected for the second time, before selecting p_1 for the second time. Repeating this argument, we can conclude that the root must have been selected for 2 times, which contradict the fact that the root must be selected for at most 1 time. So we reached a contradiction for the case that c_j is selected for 2 times. So our assumption that the sum of weights of the vertices of a variable gadget can be positive was false.

So it is not possible to gain any positive weight by selecting the vertices of clause gadgets and variable gadgets. The only vertex with positive weight that is remained, is the root of the binary tree. So it must be selected, otherwise the weight of the sequence would be 0 at most.

We claim that for a variable gadget i, if the vertex x_i^+ is selected, then either all of the vertices $x_{i,1}, ..., x_{i,l}$ and none of the vertices $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$ are selected or none of the vertices $x_{i,1}, ..., x_{i,l}$ and all of the vertices $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$ are selected. Assume to the contrary, that x_i^+ is selected and there exists two vertices x_{i,j_1} and $\bar{x}_{i,j_2}$ that are also selected. At least one of the neighbors of x_i^+ is selected. We denote it with $y_{i,l}$. $y_{i,l}$ have become selectable, because one of it's neighbors was previously selected. As we argued before, this neighbor can not be a vertex of clause gadgets, as it makes the sum of weights of the sequence negative. So the vertex that was selected and made $y_{i,l}$ selectable, is also a vertex of the variable gadget. We denote it with $y_{i,l-1}$. By repeating this argument, we can conclude that all vertices $y_{i,1}, ..., y_{i,l}$ must have been selected. As both of the vertices x_{i,j_1} and $\bar{x}_{i,j_2}$ are selected, one of them is not among $y_{i,1}, ..., y_{i,l}$. So at least $l + 1$ vertices other than x_i^+ is selected from the variable gadget. So the sum of weights of the variable gadget is negative. As we argued the sum of weights of the vertices of variable and clause gadgets can not be positive. The only vertex with positive weight is the root which can be selected at most once. As the sum of the weight of the root and the negative weight of the variable gadget i is negative, the total weight of the sequence would be negative, which is a contradiction. So our assumption was false, and if the vertex x_i^+ is selected, then either all of the vertices $x_{i,1}, ..., x_{i,l}$ and none of the vertices $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$ are selected or none of the vertices $x_{i,1}, ..., x_{i,l}$ and all of the vertices $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$ are selected.

We claim that for a variable gadget i, if the vertex x_i^+ is not selected, then no other vertices of the variable gadget is selected. Because if a vertex of the variable gadget, other than x_i^+ is selected, then the sum of weights of the variable gadget is negative and as the sum of weights of the vertices of clause and variable gadgets can not be positive, and the positive weight of the root is smaller than the negative weight of the variable gadget, the total sum of weights would be negative which contradicts our assumption.

We claim that all vertices $c_1, ..., c_l$ must have been selected, otherwise it is not possible to select the root. Assume to the contrary that a vertex c_i is not selected. Consider the path from the root of the binary tree to c_i. We know that the root is selected. In the path from root to c_i, consider the first vertex that is not selected. We denote this vertex with p_0. Such vertex exists, as c_i itself, is not selected. So the parent of p_0, denoted by p_1 is selected. Note that the other two neighbors of p_1 can be selected for at most 2 times. So based on lemma 3.1, p_1 can be selected for at most 1 time. Consider the time just before when p_1 was selected for the first time. One child of p_1, i.e. p_0 is not selected. We claim that the other child of p_1, denoted by p'_0 have not been selected for more than 1 time, at this moment. Because as p_1 is not selected at this moment, and the other neighbors of p'_0 can be selected for at most 4 times in total, based on lemma 3.1, p'_0 can be selected for at most 1 time, at this moment. So p'_0 is selected for at most 1 time at this moment. So based on 3.1, the parent of p_1, denoted by p_2 must have been selected for at least 1 time at this moment. So the p_2 must have been selected before the first selection of p_1. Using the argument that we used for p_1, we can conclude that the parent of p_2, denoted by p_3 must have been selected before p_2. Repeating this argument, we can conclude that the root of the binary tree must have been selected before at least one of it's two children. As the other child of the root can be selected for at most 1 time, this would contradict the lemma 3.1.

Now we find an assignment for the Boolean variables $x_1, ..., x_m$ that satisfies the formula ϕ. For each variable gadget i, as we argued, if the vertex x_i^+ is selected, then either all of the vertices $x_{i,1}, ..., x_{i,l}$ and none of the vertices $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$ are selected or none of the vertices $x_{i,1}, ..., x_{i,l}$ and all of the vertices $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$ are selected. If $x_{i,1}, ..., x_{i,l}$ are selected, then we assign the Boolean variable $x_i =$ TRUE. If $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$ are selected, then we assign the Boolean variable $x_i =$ FALSE. If x_i^+ is not selected, then none of $x_{i,1}, ..., x_{i,l}$ and $\bar{x}_{i,1}, ..., \bar{x}_{i,l}$ are selected. We can assign the Boolean variable x_i to whatever value. It's value is "*don't care*".

As we argued, all vertices $c_1, ..., c_l$ are selected in the sequence. As we proved in our arguments, for each of the vertices c_i, if c_i is selected, then at least one of the vertices c'_i, c''_i, c'''_i must have been selected before c_i. Without loss of generality assume c'_i is selected before c_i. Note that the only neighbor of c'_i, other than c_i, is a vertex from a variable gadget, like $x_{j,p}$. So based on our argument, all vertices $x_{j,1}, ..., x_{j,l}$ must have been selected. So the Boolean variable x_i which appears in clause i, is set to TRUE, and the clause i from ϕ is satisfied. So all clauses of ϕ are satisfied. So assuming that there exists a valid sequence for graph G with the initial state $\mathbf{s}$ with a positive weight, we proved that ϕ is satisfiable.

So we have proved both direction of (3.11). So we have reduced the *3SAT* problem to the decision version of k-neighbors-MSD problem. Note that the reduction is polynomial, as the size of the graph and all weights are polynomial in size of the input of the *3SAT* problem. So the decision version of k-neighbors-MSD is NP-complete.

□

In theorem 3.4 we proved that the decision version of the k-neighbors-MSD problem is NP-complete, for $k \geq 3$. It remains a question whether the problem is also NP-complete for $k = 2$ or not. Also in our proof of theorem 3.4, we used an initial state which would not allow to produce infinite sequences. When the initial state is such that we can have long sequences, it may be possible to find at least a good approximation of the sequence with the highest average weight.

3.4 The Probabilistic Sequence with Dependency Constraints Problem

The probabilistic sequence with dependency constraints problem, which we abbreviate it to probabilistic-SD is similar to the original MSD problem. The only difference is that we select the vertices probabilistically.

Formally, we have a function $f(\mathbf{s}, v)$ which gives the probability of selecting the vertex v when we are in state $\mathbf{s}$. If in state $\mathbf{s}$, no vertex is selectable, the value of $f(\mathbf{s}, v)$ would be 0 for all vertices $v \in V$. We would like $f(\mathbf{s}, v)$ to be a probability distribution over V. So we add a dummy vertex v_{dummy} with weight 0, so that when no vertex is selectable in an state $\mathbf{s}$, then v_{dummy} is selectable and selecting it would result in remaining in the same state. So in such state, we would have $f(\mathbf{s}, v_{\text{dummy}}) = 1$. So for every state $\mathbf{s}$, we have

$$\sum_{v \in V} f(\mathbf{s}, v) = 1.$$

Similar to the MSD problem, we have the constraint that after each time vertex v_i is selected, it cannot be selected again until after a neighbor of vertex v_i has been selected. So when a vertex v is not selectable in an state $\mathbf{s}$, we must have set $f(\mathbf{s}, v) = 0$.

It is possible to have a positive number $g(v)$ for each vertex $v \in V$ and define the function f as follows, which satisfies the condition of the MSD problem.

$$f(\mathbf{s}, v) = \begin{cases} g(v)/\sum_{u\in V,\ s_u=1} g(u) & s_v = 1 \\ 0 & s_v = 0 \end{cases},$$

But here we consider the general case and assume that the function f can be any probability function.

We denote the selected vertex at step t with v^t and the state at step t with $\mathbf{s}^t$. So $\Pr\left[\mathbf{s}^t = \mathbf{s}, v^t = v\right] = f(\mathbf{s}, v)$.

The question is, given the graph $G = (V, E)$ and a weight function $W : V \to \mathbb{R}$ and a probability function f, what is the expected average weight of the sequence generated by the following rule:

At each state $\mathbf{s}^t$, the vertex v is selected with probability of $f(\mathbf{s}^t, v)$, and based on the selected vertex, the state is changed to the state $\mathbf{s}^{t+1}$ by changing the state of the selected vertex to not selectable and changing the state of all neighbors of the selected vertex to selectable.

Let $\mathbb{S}_n$ be the set of all possible 2^n states. Let $\sigma : \mathbb{S}_n \to \{1, ..., 2^n\}$ be a mapping from the 2^n states to numbers $1, ..., 2^n$. Let $\beta : \mathbb{S}_n \times \mathbb{S}_n \to V$ be a function, such that if we are in state $\mathbf{s}_1$ and selecting a vertex $v \in V$ will result in the new state $\mathbf{s}_2$, then $\beta(\mathbf{s}_2, \mathbf{s}_1) = v$.

Based on the function f, we define a $2^n \times 2^n$ matrix F such that the value of each element is defined as follows. Assume from state $\mathbf{s}_1$, we can select a vertex v_1, and this selection results in the new state $\mathbf{s}_2$. Then we set the element in row $\sigma(\mathbf{s}_2)$ and column $\sigma(\mathbf{s}_1)$, to $f(\mathbf{s}_1, v_1)$. All other unassigned elements are assigned to 0. Note that the sum of the elements of each column of F is 1, so F is a "*Markov Matrix*".

Note that based on the definition of F, $\Pr\left[\mathbf{s}^t = \mathbf{s_2} | \mathbf{s}^{t-1} = \mathbf{s}_1\right] = F_{\sigma(\mathbf{s}_2),\sigma(\mathbf{s}_1)}$. So we have

$$\begin{aligned} \Pr\left[\mathbf{s}^t = \mathbf{s}_2\right] &= \sum_{\mathbf{s}_1 \in \mathbf{S}_n} \Pr\left[\mathbf{s}^t = \mathbf{s}_2, \mathbf{s}^{t-1} = \mathbf{s}_1\right] \\ &= \sum_{\mathbf{s}_1 \in \mathbf{S}_n} \Pr\left[\mathbf{s}^t = \mathbf{s}_2 | \mathbf{s}^{t-1} = \mathbf{s}_1\right] \Pr\left[\mathbf{s}^{t-1} = \mathbf{s}_1\right] \\ &= \sum_{\mathbf{s}_1 \in \mathbf{S}_n} F_{\sigma(\mathbf{s}_2),\sigma(\mathbf{s}_1)} \Pr\left[\mathbf{s}^{t-1} = \mathbf{s}_1\right] \end{aligned}. \qquad (3.12)$$

We define the column vector with 2^n elements $\mathbf{p}^t = [p_1^t, ..., p_{2^n}^t]^\top$ such

that
$$p_i^t = \Pr\left[\mathbf{s}^t = \sigma^{-1}(i)\right].$$

So, we can represent the equation(3.12), in the following compact form using matrix multiplication

$$\mathbf{p}^t = F\mathbf{p}^{t-1}. \tag{3.13}$$

We denote the initial state with $\mathbf{s}^0$. So all elements of $\mathbf{p}^0$ is 0 except the $\sigma(\mathbf{s}^0)$ one which is 1. Using (3.13) repeatedly, we have

$$\mathbf{p}^t = F^t\mathbf{p}^0. \tag{3.14}$$

Let $\mathcal{V}_T$ be a random sequence of vertices with length T selected according to the rule of the probabilistic-SD problem. In the next theorem, we give an equation to calculate the expected average weight of the sequence $\mathcal{V}_T$, when the length of the sequence goes to infinity.

Theorem 3.5. *Let $\mathcal{V}_T$ be a random sequence of vertices with length T selected according to the rule of the probabilistic-SD problem, started from the initial state $\mathbf{s}^0$. We have*
$$\lim_{t\to\infty} \mathbf{p}^t = \boldsymbol{\pi}, \tag{3.15}$$
where $\boldsymbol{\pi}$ is an eigenvector $\boldsymbol{\pi}$ of F with the coressponding eigenvalue of 1.

Moreover,

$$\lim_{T\to\infty} \mathrm{E}\left[W(\mathcal{V}_T)\right] = \sum_{\mathbf{s}\in\mathbb{S}_n} \pi_{\sigma(\mathbf{s})} \sum_{v\in V,\ s_v=1} f(\mathbf{s}, v)W(v). \tag{3.16}$$

Proof. Because the matrix F is a Markov Matrix, F^t converges as $t \to \infty$. Based on (3.14), we can conclude that $\mathbf{p}^t$ also converges to a vector $\boldsymbol{\pi}$ as $t \to \infty$.

Note that because $\lim_{t\to\infty} F^t = \lim_{t\to\infty} F^{t-1}$, then based on (3.14) we have $\lim_{t\to\infty} \mathbf{p}^t = \lim_{t\to\infty} \mathbf{p}^{t-1} = \boldsymbol{\pi}$. Based on (3.13), we have $\mathbf{p}^t = F\mathbf{p}^{t-1}$. So we must have $\boldsymbol{\pi} = F\boldsymbol{\pi}$. So $\boldsymbol{\pi}$ is an eigenvector of F with eigenvalue of 1.

To find the expected value of the average weight of the random sequence $\mathcal{V}_T$, we first find the expected value of the weight of the vertices selected at step t.

$$\begin{aligned}
\mathrm{E}\left[W(v^t)\right] &= \sum_{v \in V} \Pr[v^t = v] W(v_t) \\
&= \sum_{v \in V} W(v) \sum_{\mathbf{s} \in \mathbb{S}_n} \Pr[v^t = v, \mathbf{s^t} = \mathbf{s}] \\
&= \sum_{v \in V} W(v) \sum_{\mathbf{s} \in \mathbb{S}_n} \Pr\left[v^t = v | \mathbf{s^t} = \mathbf{s}\right] \Pr\left[\mathbf{s^t} = \mathbf{s}\right] \\
&= \sum_{\mathbf{s} \in \mathbb{S}_n} \Pr\left[\mathbf{s^t} = \mathbf{s}\right] \sum_{v \in V} \Pr\left[v^t = v | \mathbf{s^t} = \mathbf{s}\right] W(v).
\end{aligned}$$

So if $t \to \infty$, because $\lim_{t\to\infty} \Pr\left[\mathbf{s}^t = \mathbf{s}\right] = \pi_{\sigma(s)}$, we have

$$\lim_{t\to\infty} \mathrm{E}\left[W(v^t)\right] = \sum_{\mathbf{s} \in \mathbb{S}_n} \pi_{\sigma(\mathbf{s})} \sum_{v \in V,\ s_v = 1} f(\mathbf{s}, v) W(v)$$

Note that as $\lim_{t\to\infty} \mathrm{E}\left[W(v^t)\right]$ converges, we have $\lim_{T\to\infty} \mathrm{E}\left[W(\mathcal{V}_T)\right] = \lim_{t\to\infty} \mathrm{E}\left[W(v^t)\right]$. So we can get the result (3.16). $\square$

Note that to use the formula (3.16) in practice, we need the vector $\boldsymbol{\pi}$. But note that the size of the matrix F and vector $\boldsymbol{\pi}$ is 2^{2n} and 2^n respectively. So forming them explicitly is intractable. However, it might be possible for some specific type of functions f, to calculate the result of the formula (3.16) without forming F and $\boldsymbol{\pi}$ explicitly, maybe using ideas similar to the "*Kernel Trick*".

Chapter 4

Conclusion and Future Work

In chapter 1 we introduced the Maximum weight Sequence with Dependency constraints problem (MSD) which has application in finding the convergence rates of coordinate descent with Gauss-Southwell rule and exact optimization, and Kaczmarz method with the maximum residual rule.

In chapter 2 we solved the MSD problem when the length of the sequence goes to infinity. We showed that to find the solution, we only need to check the star sub-graphs of our graph, and we gave an algorithm that could find the solution in time $\Theta(|V|\log|V| + |E|)$. However, the problem for finite length sequences is still open.

In chapter 3, we considered 4 generalizations of the MSD problem.

The first generalization was the k-times-MSD problem, which was similar to the MSD problem, except that we could choose a vertex k times until it becomes unelectable. We showed that this problem is equivalent to the original MSD problem.

The second generalization was the k-order-MSD problem which was similar to the MSD problem, except that when we select a vertex, all vertices whose distance from the vertex is not grater than l becomes selectable. We also showed that this problem is equivalent to the MSD problem.

The third generalization that we considered was the k-neighbors-MSD problem which was similar to the MSD problem, except that we need to select k neighbors of a vertex to make it selectable. We showed that this problem is NP-Hard for $k \geq 3$. In our proof, we constructed a graph that it was not possible to have an infinite sequence. The problem for the graphs that we can have infinite sequences remains open. Also the k-neighbors-MSD problem for $k = 2$ remained unsolved (and the case $k = 1$ is the MSD problem that we solve in polynomial time). Also we introduced the conjecture 3.1 which can help in developing an algorithm to find an optimal solution for the k-neighbors-MSD problem, by limiting our search to just a particular class of sub-graphs in our graph.

The last generalization of our problem that we considered was the probabilistic-SD problem, which is similar to the MSD, except that the vertices are selected randomly. We derived a formula for the expected value of the average weights if we leave the system for a long time. However finding the value needs exponential time. Finding an approximation of the value, or the exact value for some special cases (that are useful) in polynomial time, remains open. Also finding the value for finite sequences remains unsolved.

Bibliography

[1] Julie Nutini, Mark Schmidt, Issam Laradji, Michael Friedlander, and Hoyt Koepke. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1632–1641, 2015.

[2] Julie Nutini, Behrooz Sepehry, Issam Laradji, Mark Schmidt, Hoyt Koepke, and Alim Virani. Convergence rates for greedy kaczmarz algorithms, and randomized kaczmarz rules using the orthogonality graph. In *Proceedings of the 32nd conference on Uncertainty in Artificial Intelligence (UAI)*, pages 547–556, 2016.