Performance testing and Engineering improvement on the current best Parameterized 2-Cluster Editing Algorithm

De-Ting Liu, Bang Ye Wu
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, 621 Taiwan

Abstract

We examine parameter algorithm for NP-hard minimum-sum 2-Clustering Editing Problem, in which given a graph and a parameter k, the goal is to determine if there exists a 2-partition of the vertex set such that the sum of conflict number is at most k, where the conflict number of a vertex u is the number of vertices v in the same cluster but the edge (u, v) $\notin$ E plus the number of vertices v in the different clusters but edge (u, v) $\in$ E. We implemented and tested the performance of the current best algorithm and made engineering improvements. The current best algorithm can solve the problem efficiently which has the larger value of vertices n. (e.g., n=500, n=1000.) Our algorithm saves about 20% of the running time. In addition, we propose a new measure of comparing parameterized algorithms.

1 Introduction

For an NP-hard problem, since it is unlikely to develop a polynomial time algorithm, designing good exponential-time algorithms is an important issue if one needs to find the optimal solutions. Parameterized algorithm, which measures the complexity as a function in input parameters, is another method to deal with NP-hard problems. It can solve the problem efficiently which has the small value of the fixed parameter.

An instance of parameterized problem is a 2-tuple (I, k), where I is the input and k is the parameter. A problem *is fixed-parameter tractable* (FPT) if the problem can be solved in $O(f(k) \cdot q(|I|))$, where f is a computable function in k and q is a polynomial in input size. We refer to the book of Downey and Fellows [5] for details. *Kernelization* is a well-known technique which is widely used for parameterized algorithms. A kernelization algorithm converts an instance (*I, k*) to a reduced instance (I', k') with $k' \leq k$ and $|I'| \leq g(k)$ for some computable function g, and the answer is not changed. That is, (*I, k*) is a yes-instance if and only if (I', k') is a yes-instance.

Cluster Editing is a well-known NP-hard problem with important applications in numerous fields, especially in bioinformatics and machine learning. The Cluster Editing problem is also known as Correlation Clustering [3, 4] and have been appeared in several variants of the problems [2]. A cluster graph is an undirected graph consisting of numerous disjoint maximal cliques. The maximal cliques in a cluster graph are also called clusters. The goal of the cluster editing problem is to modify a given input graph into a cluster graph such that the number of inserting and deleting edges is minimized. One of the variants is *p*-Cluster Editing, which modify the input graph to a *p*-Cluster graph, that is, the number of clusters is exactly *p*.

In this paper, we focus on the 2-cluster editing problem [1]. Given an input graph $G = (V, E)$, the goal is to partition the vertices into two vertex subsets, which is denoted by $\pi = (V_1, V_2)$ of V in the following. For (u, v) $\in$ V, u and v *conflict with each other* if *u* and *v* are in the same cluster but (u, v) $\notin$ E or they are in the different clusters but edge (u, v) $\in$ E. Let $c_\pi(v)$ denote the number of vertices conflicting with v in π. Let $h(\pi) = \sum_{v \in V} c_\pi(v)$ be the sum of conflict number in a partition π. The min-sum 2-clustering problem is to determine if there exists a 2-partition of the vertex set such that the total conflict number is at most k. The problem is defined as follows.

MIN-SUM 2-CLUSTERING
Instance: An undirected graph $G = (V, E)$ and a nonnegative integer *k*.
Question: Is there a 2-partition $\pi = (V_1, V_2)$ of V such that $h(\pi) \leq k$ and $V_1, V_2 \neq \emptyset$?

A parameterized algorithm for min-sum 2-clustering running $O(n \cdot \emptyset^{1-\frac{4r}{n}} + n^3)$ in time is given in [1], where $\emptyset \approx 1.618$ and $r = k/n$. The time complexity is $O(n \cdot 2.619^{k/n} + n^3)$ for $k \in o(n^2)$. This implies the problem can be solved in subexponential time. In addition, It can be solved in polynomial time when $k \in O(n \log n)$. The algorithm achieves the current best complexity of the problem.

In this paper we implement the parameterized algorithm for min-sum 2-clustering problem and test its practical performance on random graphs. Our experimental results indicate that the current

Algorithm 1 : Min-sum 2-clustering

Input: a graph $G = (V, E)$ and integer k.
Output: determining if existing π with $h_1(\pi) \leq k$.

```
1: if existing an extreme 2-partition with cost at most k then
2:     return True;
3: end if
4: for each s ∈ V do
5:     π_0 ← (N_G[s], V \ N_G[s]);
6:     call REDUCTION(G, π_0, k, (n − 1)/2, k/n) to compute (U, π, m);
7:     χ ← c_π(X, X), where X = V \ U;                ▷ Preparing for tree-search
8:     construct the list L_1 of C_π(v, U) and the list L_2 of c_π(v, X) for each v ∈ U;
9:     if SEARCH1(U, m, L_2, χ)=True then
10:        return True;
11:    end if
12: end for
13: return False.
```

Algorithm 1: The algorithm is presented by Wu et al. We call it Flipping algorithm in this paper.

best algorithm can solve the problem efficiently which has the larger value of vertices n. (e.g., n=500, n=1000.) We summarize Algorithm 1 in the following three steps: Step 1: Partition the vertex set by guessing the vertex with the smallest conflict number. Step 2: Apply kernelization-style rules, determine which vertices should be flipped to the other cluster. Step 3: Apply a standard branching algorithm to deal with the undetermined vertices. We improve the above algorithm by proposing a better partition at step 1 and finding a better way to pick the vertex in the tree searching at step 3. We implement our improvement methods and compare our performance with the original algorithm. Our experimental results indicate that our improved method is more efficient than the original one. In addition, we propose a new way of practical performance comparison for parameterized algorithms.

Organization of the paper. In Section 2, we give some notation and definitions used in this paper. In Section 3, we show that Flipping algorithm and our improved algorithm. We give some basic idea about Flipping algorithm in section 3.1. In section 3.2, we propose our improved methods. The experimental results are shown in Section 4. Finally some concluding remarks and future work are given in Section 5.

2 Preliminaries

Throughout this paper, a graph is always undirected and simple. For a given graph G, V(G) and E(G) denote the vertex and edge sets, respectively. In this paper, $G = (V, E)$ is the input graph and n= |V|. Let $N_G[v] = \{u|(u, v) \in E\} \cup \{v\}$ denote the closed neighborhood of v in G. We partition the vertex set V into a 2-partition of V, which is represented as $\pi = (V_1, V_2)$, that is, $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. We call two vertices *conflict*, if two vertices u and v are in the same cluster but the edge (u, v) $\notin$ E or they are in the different clusters but edge (u, v) $\in$ E. Let $C_\pi(v)$ denote the set of vertices in conflict with v in the partition π, and $c_\pi(v) = |C_\pi(v)|$ denote the conflict number of v. We use the function $h(\pi)$ to represent the sum of conflict number with $v \in V$ in π. Let $h(\pi) = \sum_{v \in V} c_\pi(v)$ represent the cost of a 2-partition π.

For two sets S_1 and S_2. Let $S_1 \setminus S_2$ denote the set difference. Let $S_1 \ominus S_2 = (S_1 \setminus S_2) \cup (S_2 \setminus S_1)$ denote the symmetric difference. To move a vertex to the other cluster in a 2-partition π is called *flip*. We flip a vertex in a 2-partition π, that is to change π to $\pi \ominus v \equiv (V_1 \ominus v, V_2 \ominus v)$. Conflicting-relations of a vertex are exchanged when it is flipped. We find the optimized 2-partition by finding a best flipping set F of an initial 2-partition.

3 Algorithm and Improvements

3.1 The original algorithm

Algorithm 1 is the proposed by Wu and Chen [1], we call it *Flipping algorithm* in this paper. The main idea is to solve the Min-Sum 2-Clustering problem by finding the flipping set. First, the algorithm yields a 2-partition π_0, where $\pi_0 = (N_G[s], V \setminus N_G[s])$ for each vertex $s \in V$. Second, the algorithm calls the reduction algorithm to

Algorithm 2 : Min-sum 2-clustering

Input: a graph $G = (V, E)$ and integer k.
Output: determining if existing π with $h(\pi) \leq k$.

1: **if** existing an extreme 2-partition with cost at most k **then**
2: **return** True;
3: **end if**
4: **for** each $s \in V$ **do**
5: $\pi_0 \leftarrow (N_G[s], V \setminus N_G[s])$;
6: call REDUCTION$(G, \pi_0, k, (n-1)/2, k/n)$ to compute (U, π, m);
7: $p(s) \leftarrow |U| + m$;
8: **end for**
9: **for** each increasing order of $p(s)$ **do**
10: $\pi_0 \leftarrow (N_G[s], V \setminus N_G[s])$;
11: call REDUCTION$(G, \pi_0, k, (n-1)/2, k/n)$ to compute (U, π, m);
12: $\chi \leftarrow c_\pi(X, X)$, where $X = V \setminus U$; ▷ Preparing for tree-search
13: construct the list $\mathcal{L}_1$ of $C_\pi(v, U)$ and the list L_2 of $c_\pi(v, X)$ for each $v \in U$;
14: **if** SEARCH1(U, m, L_2, χ)=True **then**
15: **return** True;
16: **end if**
17: **end for**
18: **return** False.

Algorithm 2: The main algorithm is presented in this paper.

reduce the instance by removing the vertices which must be flipped and the vertices which must not be flipped. The reduction algorithm moves the vertices which must be flipped to the other cluster in a 2-partition π_0. Let $\pi_0 \leftarrow \pi_0 \ominus v$. The reduction algorithm is used to compute $(\mathrm{U}, \pi, \mathrm{m})$, where U denotes the set of the vertices that is undetermined, π denotes the 2-partition after flipping the vertices which must be flipped, and m denotes the remaining *flipping quota*. Finally, the algorithm performs a branching algorithm to deal with the undetermined vertices. The reduction rules are listed as follows.

R1: If $c_\pi(v) > t + f$, then v must be in any feasible flipping set for π of size at most *f*.
R2: If $c_\pi(v) < n - t + f$, then v cannot be in any feasible flipping set for π of size at most *f*.
R3: If $c_\pi(v) \leq t + f$ for all $v \in V$ and $|U| > (K/(n - t - 2f)) - f$, where $U = \{v | c_\pi(v) \geq n - t + f\}$, then there is no feasible flipping set for π of size exactly *f*.

Let *f* denote the size of flipping sets. It is called *flipping quota*. *K* and *t* are nonnegative integers. The reduction algorithm is applied with $K = k, t = (n - 1)/2$ as shown in the proof of the original paper. For more details about Flipping algorithm, please refer to [1].

3.2 An improved Algorithm

We present two improvements in Flipping algorithm. Both of the improvements focus on the running time while still finding an exact solution. We start with two engineering improvements that save a constant factor in the running time. In order to reduce execution time, we modify the Algorithm 1 to Algorithm 2. The first improvement is to find a better partition order. By testing the performances, we propose that a vertex with smaller sum of $|U|$ and *m* should be chosen first, since the depth of search tree is affected by the sum of $|U|$ and *m*. Let p(s) denote the sum of $|U|$ and m for vertex s. This improvement is shown at steps 4-9 in Algorithm 2. Second, we improve the tree searching process by finding a better way to pick the vertex, which is described in Algorithm 3.

A 2-partition (V_1, V_2) is extreme if $|V_1| = 1$ or $|V_2| = 1$. In algorithm 2, we first exclude the extreme case, since we can easily calculate $h(\pi)$ of the extreme case in $O(n^2)$ time. After the reduction step, we calculate p(s) for every 2-partitio π_0, where $\pi_0 = (N_G[s], V \backslash N_G[s])$ at step 7. For all vertex $s \in V$, we can compute the p(s) in $O(n^3)$. Then we apply the following reduction algorithm and search tree algorithm by

Algorithm 3 Search-tree algorithm for MIN-SUM 2-CLUSTERING

Input: a set U of undetermined vertices, a flipping quota m, a list L_2 of $c(v, X)$ for each $v \in U$, and $\chi = c(X, X)$, where $X = V \setminus U$. In addition, a list $\mathcal{L}_1$ of $C(v, U)$ for each $v \in U$ is stored as a global variable.

1: **procedure** SEARCH1(U, m, L_2, χ)
2: **if** $\chi > k$ **then return** False;
3: **if** $U = \emptyset$ or $m = 0$ **then** ▷ no vertex can be flipped
4: $q \leftarrow \chi + \sum_{v \in U}(|C(v, U)| + 2c(v, X))$; ▷ total conflict number
5: **if** $q \leq k$ **then return** True **else return** False;
6: **end if**
7: pick an vertex $u \in U$, which $c_\pi(u, X)$ is maximum;
8: $U' \leftarrow U \setminus \{u\}$; ▷ $X' = V \setminus U'$
9: modify $\mathcal{L}_1$: $C(v, U') \leftarrow C(v, U) \setminus \{u\}, \forall v \in U'$; record the modifications in L_3;
10: $\chi' \leftarrow \chi + 2c_\pi(u, X)$; ▷ move u to X without flipping
11: construct L_2' from L_2 by $c(v, X') \leftarrow c(v, X) + c(v, u), \forall v \in U'$;
12: **if** SEARCH1(U', m, L_2', χ')=True **then**
13: **return** True;
14: **end if**
15: $\chi'' \leftarrow \chi + 2(|X| - c(u, X))$; ▷ flip and move u to X
16: construct L_2'' from L_2 by $c(v, X') \leftarrow c(v, X) + 1 - c(v, u), \forall v \in U'$;
17: **if** SEARCH1($U', m - 1, L_2'', \chi''$)=True **then**
18: **return** True;
19: **end if**
20: recover $\mathcal{L}_1$ by undoing the modifications in L_3;
21: **return** False;
22: **end procedure**

Algorithm 3: The Search-tree algorithm of our algorithm in this paper.

the ascending order of the summation value.

. We call the reduction algorithm at step 11, which returns (U, π, m), we define U as the set of the vertices that is undetermined, π as the 2-partition after flipping the vertices which must be flipped, m as the remaining *flipping quota*. The goal of the search-tree algorithm is to find a flipping set $F \subseteq U$ with $|F| \leq m$ and $h(\pi \ominus F) \leq k$.

The implementation of the search-tree algorithm is shown in Algorithm 3. We have a set U of vertices which is undetermined. The algorithm picks a vertex $u \in U$, which $c_\pi(u, X)$ is maximum. Then we recursively solve the problem for two cases: to flip u to the other cluster and not to flip u. When $U = \emptyset$ or $m = 0$, we compute the total conflict number. If the conflict is greater than the parameter k, return False; else return True.

The naïve search tree algorithm takes $O(n^2)$ time for each recursive call, and the time complexity will be $O(n^2)$ multiplied by the number of recursive calls. The implementation of search-tree algorithm takes $O(|U|^2)$ time for each recursive call, which is similar to the technique widely used in fixed-parameter algorithms [8]. In addition, we update χ, where $\chi = c_\pi(X, X)$ and $X = V \setminus U$, at each recursive call. When the value χ is greater than parameter k, we can return the output immediately. So we can determine that there is no solution in branching process when the value χ is more than parameter k. Therefore, we propose the following method to let χ increase faster by picking the vertex from the set U.

There are two cases as the value χ is increased at each recursive call. One is that the value χ increases by $2c_\pi(u, X)$ when we pick a vertex that is determined not to flip, and the other is that the value χ increases by $2(|X| - c_\pi(u, X))$ when we pick a vertex that is flipped. In addition, the value $c_\pi(v, X)$ is changed for each vertex $v \in U$ when we pick a vertex u that is removed from U. We experiment several methods to find the better way to pick the vertex, such as picking the vertex u which $c_\pi(u, X)$ is the largest, picking the vertex u

δ (K= $\delta \cdot n^2$)	0.15	0.16	0.17	0.18	0.19	0.191	0.192	0.1921
Parameter K	150000	160000	170000	180000	190000	191000	192000	192100
Basic (avg.)	11.241	11.724	12.105	12.075	23.608	93.127	511.059	644.088

Table 1: Average running time (wall time in s) of Flipping algorithm on the first dataset (n=1000). $\delta = \mathrm{K}/n^2$

Vertex number	100		500		1000	
Parameter	δ	K	δ	K	δ	K
Basic Alg.	0.1810	1800~1810	0.1924	48075~48100	0.1922	192100~192200

Table 2: The maximum parameter k for solving the problem within 1800 seconds on the first dataset.

δ (K= $\delta \cdot n^2$)	0.16	0.17	0.18	0.19	0.20	0.21	0.22	0.23	0.24	0.25
Basic Alg.	285	1536	8523	80986	398466	644960	3041552	7945769	11704959	24840947
Our Alg.	158	829	5130	51772	261219	415689	1957105	5418674	8134340	17273384
Improvement	44.56%	46.03%	39.81%	36.07%	34.44%	35.55%	35.65%	31.80%	30.51%	30.46%

Table 3: The improvement of average recursive time on the first dataset (n=35). $\delta = \mathrm{K}/n^2$

δ (K= $\delta \cdot n^2$)	0.16	0.17	0.18	0.19	0.20	0.21	0.22	0.23	0.24	0.25
Basic Alg.	0.015	0.018	0.035	0.366	1.736	2.812	12.129	30.121	44.792	90.899
Our Alg.	0.013	0.014	0.028	0.306	1.47	2.41	10.357	26.898	41.012	91.797
Improvement	13.33%	22.22%	20.40%	16.35%	15.30%	14.31%	14.61%	10.70%	8.44%	10.01%

Table 4: The improvement of average running time on the first dataset (n=35). $\delta = \mathrm{K}/n^2$

which ($|\mathrm{X}| - c_\pi(u, X)$) is the largest, finding the case that let χ increase the largest and so on. Finally, the method that picks the vertex u which $c_\pi(u, X)$ is the largest performs the best in our experimental results. The experiments show the method can decrease the recursive time by 30 to 40% and decrease the running time by 10 to 20%.

4 Experimental Results

We first evaluate Flipping algorithm. Flipping algorithm performs quite well when the parameter k is less than a certain amount. In addition, the algorithm can solve the instance efficiently which is given a larger value of vertices n. (e.g., n=500, n=1000.) Then, we compare our algorithms with Flipping algorithm on the running time and the number of recursive calls. The experiments show that we can reduce 30~40% of the recursive times and 10~20% of the running time.

Experimental Setup and Implementation Details.

The program is written in the C programming language and consists of about 600 lines of code. We tested our implementation on random inputs. The testing machine is a PC equipped an Intel(R) Core(TM) i7-4790 CPU @3.60GHz, 8MB cache, and 4GB main memory, running under the 64bit Windows 7 operating system. The source was compiled by the GNU GCC compiler (version 4.7.1, 32 bit).

Datasets and Experimental Design.

We generate two types of input dataset. The first dataset consists of 50% Yes-instances and 50% No-instances. The second dataset consists of all Yes-instances. Each dataset consists of 1000 random graphs, which we built with n vertices and the parameter k. We implemented the min-sum 2-Cluster Problem on the random graphs. We constructed the random graphs by the following steps: In step 1, we partition n vertices into two vertex sets randomly, and then we generate 2-clusters graph with the two vertex sets. In step 2, we generate an edge set randomly with n vertices by inserting edge incident to two random vertices. Finally, we combine the two graphs by taking exclusive-or.

For each random graph, we set the time limit to 1800 seconds for the 2-clustering parameterized problem. We aim at finding the maximum parameter k such that instances can be solved in the time limit.

4.1 Experimental Result of Flipping algorithm

Table 1 shows the average running time of Flipping algorithm solving the instances with number of vertices n=1000 and parameter k on the first dataset. Table 1 provides an overview of our experimental results. We obtained an effective performance for our implementation of Flipping algorithm. Table 1 shows that Flipping algorithm can solve the case of n=1000 in 23 seconds with

Vertex number	100		500		1000	
Parameter	δ	K	δ	K	δ	K
Basic Alg.	0.191	1910~1920	0.2041	51025~51050	0.2045	204500~204600
Our Alg.	0.244	2440~2450	0.233	58250~58275	0.2255	225500~225600

Table 5: The maximum parameter k for solving the problem within 1800 seconds on the second dataset.

$k \leq 190000$ ($\delta \leq 0.19$) and the running time is exponential growth when $k > 190000$ ($\delta > 0.19$). The result shows that our implementation of Flipping algorithm can run fast with the parameter k/n^2 less than a certain amount. In addition, the result shows that Flipping algorithm can solve the instance which has the larger value of vertices n than other exponential algorithms.

We propose a new way of practical performance comparison for parameterized algorithms. To gauge the practical merit of a *fixed-parameter algorithm*, we test the maximum parameter k for solving the case of the same number *n* within 1800 seconds. We show that the experimental result on Table 2. The interval of parameter k is due to that the data is generated randomly. Table 2 shows that we can solve the case of n=100 with $k \leq 1800$ in 1800 seconds, the case of n=500 with $k \leq 48075$ in 1800 seconds and the case of n=1000 with $k \leq 192100$ in 1800 seconds.

4.2 Experimental Result and Comparison of Improvement Algorithm

Our algorithm improves the branching process by finding the better way to pick the vertex. Because the running time of a *fixed-parameter algorithm* grows exponentially when the parameter k/n^2 is more than a certain amount. We want to observe the branching process for all kinds of parameter k, and so we test the case with a small number of vertices n=35 on the first dataset. Table 3 and Table 4 show that the running time and recursive times of the two algorithms. The improvement denotes the difference between the running time of Flipping algorithm and our algorithm divided by the running time of Flipping algorithm. The experiments show that the method can decrease the recursive times by 30 to 40% and decrease the running time by 10 to 20%. Fig.1 shows that the percentage of improvement on running time and recursive times. In addition, our algorithm is also implemented efficiently by improving the order of partitioning on the Yes-instances. Therefore, we experiment on the second dataset.

We can gauge the practical merit of algorithms by comparing the maximum parameter k of case that is solved within the same limit time. The Table 5 shows the experimental result for Flipping

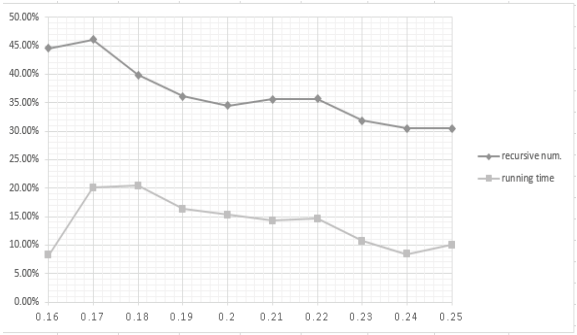

Fig.1: Horizontal axis denotes δ and vertical axis denotes the percentage of improvement.

algorithm and our algorithm on the second dataset. Table 5 shows that we can solve the case of n=100 with $k \leq 2440$ in 1800 seconds, the case of n=500 with $k \leq 58250$ in 1800 seconds and the case of n=1000 with $k \leq 225500$ in 1800 seconds. Our algorithm can solve the instances with larger parameter k than Flipping algorithm in the same time limit of 1800 seconds. The result shows that our algorithm is more efficient than Flipping algorithm.

5 Conclusions

In this paper, we implement the parameterized algorithm for min-sum 2-clustering problem and test its practical performance. The problem is the same as 2-Cluster Editing in the literature with a multiplication factor of two in the parameter. In addition, we present a method to get an engineering improvement for the original algorithm. We improve the branching process of the algorithm by finding a better way to pick the vertex and we also improve the running time by finding an order of partitioning. In addition, we propose a new way of practical performance comparison for parameterized algorithms. Our implementation of Flipping algorithm can run fast when the parameter k/n^2 is more than a certain amount. In addition, the Flipping algorithm can solve the instance efficiently which has the larger value of vertices n. (e.g., n=500, n=1000.) The result shows that our algorithm can solve the instances with larger parameter k than Flipping algorithm in the same time limit of 1800 seconds. Our algorithm is more efficient than Flipping algorithm.

References

[1] B. Y. Wu and L.-H. Chen. Parameterized algorithms for the 2-clustering problem with minimum sum and minimum sum of squares objective functions. Algorithmica, in print, on-line available, 2014.

[2] Chen, L.H., Chang, M.S., Wang, C.C., and Wu, B.Y.: On the min-max 2-cluster editing problem. J. Inf. Sci. Eng. 29(6), 1109–1120 (2013).

[3] Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Appl. Math. 144(1–2), 173–182 (2004).

[4] Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Mach. Learn. 56, 89–113 (2004).

[5] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Berlin (1999).

[6] Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer (2010)

[7] Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Springer-Verlag (2013)

[8] Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter-tractable algorithms. Inf. Process. Lett. 73(3–4), 125–129 (2000).