FPT-Approximation for FPT Problems

Hussein Houdrouge
houdrouge.hussein@gmail.com

April 2023

In the $\mathcal{NP}$ class, some problems are known to be $\mathcal{NP}$-hard, and most likely, there is no polynomial time algorithm that decides them. Therefore, under this assumption, approximation algorithms were designed to run in polynomial time. As their name suggests, these algorithms can only guarantee an approximate solution within some factor, ideally a small constant. Another approach to deal with $\mathcal{NP}$-complete problems is to design exact algorithms whose run-time is exponentially in a parameter different than the size of the problem. This type of algorithm is called a fixed-parameter tractable algorithm. The authors of [LMR$^+$] combine the two worlds and present several techniques to develop FPT approximation algorithms for FPT problem with better approximation guarantee than the polynomial algorithms. In other words, they designed approximation algorithms whose run times are exponential in a parameter different than the problem size, run faster than the best known FPT algorithm for the problems of interest in [LMR$^+$], and provide better approximation than the polynomial approximation algorithm. Thus, in the remaining of this document, we give an introduction to the FPT world and present the new technique in [LMR$^+$].

We first start by giving some preliminaries in Section 1, then we proceed to Section 2 to present common techniques in the design of fixed parameter algorithms. Lastly, we present an FPT-approximation algorithm for Subset Feedback Vertex Set in a directed graph. An experienced reader in the design of FPT-algorithm may start immediately from Section 3.

1 Preliminaries

In the remaining of this work, a graph $G = (V, E)$ stands for a simple undirected graph. A digraph $D = (V, A)$ stands for a directed graph, we usually refer to a directed edge by an arc.

The formal definition of a parameterized problem and a parameterized algorithm is the following.

Definition 1.1. (Parameterised Problem [CFK$^+$15]) A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed finite alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.

Example 1 (Vertex Cover)**.**
Input: a graph $G = (V, E)$, a parameter k.
Output: Yes, if there is a $V' \subseteq V$ of size at most k such that every edge $\{u, v\} \in E$ either $u \in V'$ or $v \in V'$. Otherwise, no.

In Example 1, the pair (G, k) belongs to the language of parameterised vertex cover if and only if G encodes an undirected graph in the alphabet Σ and G has a vertex cover of size k. In this case, we call k, a natural parameter. However, the parameter is not necessarily the size of the set in question. For example, it can be the tree-width of a graph.

Definition 1.2 (FPT problem [CFK$^+$15])**.** A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is fixed parameter tractable (FPT) if there is an algorithm A (called a fixed parameter algorithm) that decides if an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ is in L in time bounded by $f(k).|(x, k)|^{O(1)}$, where $f : \mathbb{N} \longrightarrow \mathbb{N}$ is a computable function.

Moreover, we call FPT the class of all problems that admit a fixed parameter algorithm.

Since approximation algorithms are concerned with optimisation problems, we introduce the following definition.

Definition 1.3 (NP-optimisation problem)**.** An NP-optimisation problem is a tuple $(I, sol, cost, goal)$. I denotes the set of instances. For an instance $x \in I$, $sol(x)$ is the set of feasible solutions for x, the length of each $y \in sol(x)$ is polynomially bounded in $|x|$, it can be decided in polynomial time in x whether $y \in sol(x)$ holds for a given x and y. Given an instance x and a feasible solution y, $cost(x, y)$ is a polynomial time computable positive integer. The goal is either min or max .

Example 2. The vertex cover optimisation problem can be described as $(I, sol, cost, \min)$. In this case, I will be the set of all graphs. For a graph $x \in I$, the $sol(x)$ is all subset of $V(G)$ that covers $E(G)$. For $y \in sol(x)$ is bounded by $|V(G)|$, It is possible to verify that y is a solution in polynomial time by checking that every vertex has an endpoint in y. $Cost(x, y)$ is the number of vertices in y. The goal is to find $\min_{y \in sol(x)} cost(x, y)$.

Next, we define what is an $FPT-$approximation Algorithm.

Definition 1.4 (FPT-approximation Algorithm)**.** Let $X = (I, sol, cost, goal)$ be a minimisation problem. A standard factor $c(k)$ FPT-approximation algorithm for X (where the parameterization is by solution size or value) is an algorithm that takes an input (x, k) satisfying $opt(x) \leq k$. Then, it runs in time $f(k) \cdot |x|^{O(1)}$ to compute a $y \in sol(x)$ such that $cost(x, y) \leq k.c(k)$. For inputs not satisfying $opt(x) \leq k$, the output can be arbitrary.

An analogous definition can be drawn for a maximisation problem.

2 Some Techniques for Designing FPT-Algorithms

In this section, we present techniques to mainly design FPT-algorithms. These techniques will be also used to design FPT-approximation algorithms in Section 3.

2.1 Branching

We begin with a very well known technique, branching (or bounded search tree). Informally, we try to build a solution for a given problem by taking a sequence of decisions such as including a vertex in a solution or not.

We will illustrates this techniques on the vertex cover problem introduced in Example 1. We start by the following observation.

Observation 1. Suppose $V' \subseteq V$ is a solution for the vertex cover problem. Then, for every edge $e = \{u, v\} \in E(G)$, either $u \in V'$ or $v \in V'$.

Given this observation, we can design the following branching algorithm that take a graph G and an a positive integer k, then returns true if there is a vertex cover of size k, otherwise it returns false.

Algorithm 1: VertexCover(G, k).

1. if G has no edges then return true
2. if $k = 0$ then return false
3. let $e = \{uv\} \in E(G)$
4. return VertexCover$(G - u, k - 1)$ or VertexCover$(G - v, k - 1)$

□

It is straightforward to show the correctness of this algorithm using Observation 1. In addition, it is easy to see that after k steps the algorithm halts. The execution tree of this algorithm is a binary tree with 2^k leaves. Therefore, the run-time of Algorithm 1 is $2^k n^{O(1)}$.

2.2 Iterative Compression

The idea behind iterative compression is to start from a given solution for the problem at hand, then try to exploit the knowledge gained from this solution to obtain a smaller one.

Consider the vertex cover problem again, suppose using an approximation algorithm we get a solution W of size at most $2k$ for some k. Suppose, we want to find a solution Z of size at most k. Z might contains element from W or not. This observation leads to the following problem that we will call VertexCover-Compression.

Definition 2.1 (VertexCover-Compression)**.**
Input: a graph G, a solution W of size at most $2k$, and a positive integer k.
Output: a solution S of size at most k.

Notice, if $2k < |W|$, then we can conclude that there is no instance. As noted before a solution Z might be constituted of $F \subseteq W$ and $X \subseteq V$ such that $X \cap (W - F) = \phi$. Suppose we can guess F in FPT-time, then what remains is to find X disjoint from $W - F$. This problem can be defined as follow.

Definition 2.2 (VertexCover-Disjoint)**.**
Input: a graph G, a solution W of size at most $2k$, $F \subseteq W$, and a positive integer k.
Output: A solution S for $G - F$ disjoint from $W - F$ such that $|S \cup F| \leq K$.

To solve the last problem, observe that the set $W - F$ cannot have an edge between its vertices i.e. it is an independent set. Otherwise, any disjoint vertex cover will miss this edge. Therefore, we can assume that $W - F$ is an independent set, then $N(W - F)$ is a disjoint solution, then if $F \cup N(W - F)$ has size at most k then we are done. Given this algorithm for the disjoint vertex cover problem we can solve the compression version as follows.

Algorithm 2: VertexCover-Compression.

1. Branch on all the subset F of W.
2. Solve Disjoint problem on $G - F$ given $W - F$.
3. Return the solution if it exists.

□

The run-time for Algorithm 2 is $2^{|W|}n^{O(1)} \leq 4^k n^{O(1)}$. This algorithm is slightly worst than the previous one because we started with a large solution W. However, the iterative part of the iterative compression method provides us with a mechanism to start with a solution of size at most $k+1$.

The main idea is to build the instance of the problem iteratively in order to obtain a solution of size at most $k+1$ and maintain a smaller solution at each step by applying the compression procedure.

Consider an arbitrary sequence of the vertices of G $v_1, ..., v_n$, and let $G_i = G[\{v_1, ..., v_i\}]$. The iterative compression algorithm is described as follows.

Algorithm 3: VertexCover Iterative Compression.

1. $S = \{v_1\}$
2. For $i = 1$ to n.
   (a) S = VertexCover-Compression(G_i, S, k)
   (b) if $S > k$ return NO. Otherwise set $S = S \cup \{v_{i+1}\}$ and proceed to the next iteration.

□

Notice that we can terminated as quickly as we find a no instance G_i for some i because if a subgraph requires a vertex cover larger than k then G has a larger vertex cover.

To summarize the run-time of the iterative compression, we provide the following remark as in [CFK$^+$15].

Remark 1.

- If there exists an algorithm solving $(*)$-Compression in time $f(k) \cdot n^c$, then there exists an algorithm solving problem $(*)$ in time $O(f(k) \cdot n^{c+1})$.
- If there exists an algorithm solving Disjoint-$(*)$ in time $g(k) \cdot n^{O(1)}$, then there exists an algorithm solving $(*)$-Compression in time
$$\sum_{i=0}^{k} \binom{k+1}{i} g(k-i) n^{O(1)}.$$
In particular, if $g(k) = \alpha^k$, then (*)-Compression can be solved in time $(1+\alpha)^k \cdot n^{O(1)}$.

2.3 Important Separators/Cuts

In this section, we will present another techniques that is used in designing FPT-algorithms for problems such as Multiway Cut and Feedback Vertex Set [CFK$^+$15]. We start by giving the essential definition in an undirected graph.

Definition 2.3 (Important Separators)**.** An $(X, Y)-$separator $\delta(R)$ is important if there is no (X, Y)-separator $\delta(R')$ with $R \subset R'$ (Cover) and $|\delta(R')| \leq |\delta(R)|$ (if both condition applies we say $\delta(R')$ dominates $\delta(R)$).

An important theorem that can be exploit to design FPT-algorithms is the following one.

Theorem 2.1. *There are at most 4^k important (X, Y)-separators of size at most k.*

This bound is essentially tight due to the example in Figure 2.

Now, we proceed to illustrate an FPT-algorithm based on the important separator technique. First, we introduce the problem of Multiway Cut.

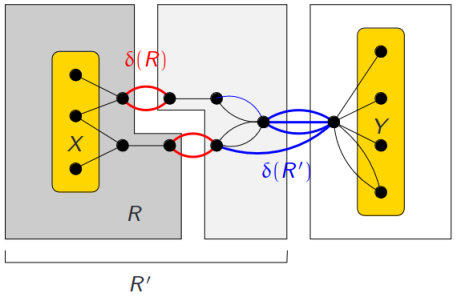

Figure 1: The blue edges are important separator of size 4, while the red edges are separator but not important [DML].

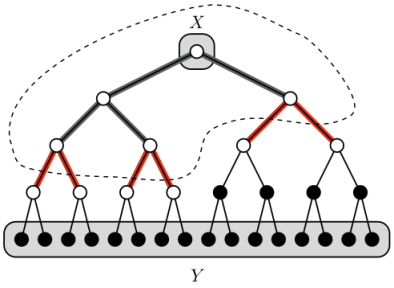

Figure 2: A graph with $\theta(4^k/k^{3/2})$ important (X, Y)-cuts of size k: every full binary sub-tree with k leaves gives rise to an important (X, Y)-cut of size k [CFK+15].

Definition 2.4 (Multiway Cut)**.**
Input: A graph $G = (V, E)$, a set $T \subseteq V$ of terminals, and an integer k.
Output: A set $S \subseteq E$ of size at most k such that each connected component of $G - S$ contains at most one $t \in T$.

In order to solve the Multiway Cut problem, we need the following lemma.

Lemma 2.2 (Pushing lemma [DML])**.** *Let $t \in T$. The Multiway Cut problem has a solution S that contains an important $(t, T - t)$-separator.*

The proof of this lemma is straightforward and it can be find in [DML]. Now, using Theorem 2.1, Lemma 2.2, and the branching technique, we can develop the following FPT-algorithm.

Algorithm 4: Multiway Cut:

- Choose a t, while t is not alone in a component.
- Enumerate all the important $(t, T - t)$-separators.
- Branch on a separator S of size at most k, set $k = k - |S|$, and repeat for another $t \in T$.

□

After giving a brief overview on important separator in undirected graph, we introduce analogous definition for important separator in digraph. Then, we will illustrate the use of such separators in the design of $FPT-$approximation algorithm in Section 3. For FPT-algorithms in digraph we refer the reader to Chapter 8.5 in [CFK+15].

Definition 2.5. (Separators in Digraph) A minimal vertex set S disjoint from $X \cup Y$ is called an $X - Y$ separator if there is no $X - Y$ path in $D - S$. Note S is minimal if no strict subset of S is also an $X - Y$ separator.

Given a set X and separator S as in the above definition, we will denote by $R_D(X, S)$ the set of vertices reachable from vertices of X via directed path in $D - S$. We will also denote $NR_D(X, S)$ the set of vertices not reachable from vertices of X in $D - S$.

Definition 2.6 (Covering and Dominating)**.** Let S_1, S_2 be $X - Y$ separators. We say S_2 covers S_1 (denoted by $S_1 \sqsubseteq S_2$) if $R(X, S_1) \subseteq R(X, S_2)$. We say S_2 dominates S_1 (denoted $S_1 \preccurlyeq S_2$) if S_2 covers S_1 and $|S_2| \leq |S_1|$.

We present the following observation about the separators in a digraph that will be a key element in proving several lemmas in the next section.

Observation 2. Let S_1 and S_2 be minimal $X-Y$ separators such that $S_1 \sqsubseteq S_2$. Then, $S_2-S_1 \subseteq NR(X, S_1)$. Similarly $Y \subseteq NR(S_1 - S_2, S_2)$.

Finally, we give the definition of important separator in a digraph.

Definition 2.7. (Important Separators) Let S be a minimal $X - Y$ separator.

- S is important $X - Y$ separator closest to Y, if there is no $X - Y$ separator S' such that $S \preccurlyeq S'$.
- S is important $X - Y$ separator closest to X, if there is no $X - Y$ separator S' such that $S' \preccurlyeq S$.

3 FPT-approximation Algorithms

In the proceeding, we introduce a technique called two-extremal separator technique using in the approximation of FPT-problems. This technique is initiated in [LMR$^+$]. It is applied on problems classified as $\mathcal{F}$-transveral.

Definition 3.1 ($\mathcal{F}$-transveral set)**.** Let $\mathcal{F} = \{F_1, F_2, ..., F_q\}$ be a fixed set of sub-graphs of a digraph D such that $\mathcal{F}$-free sub-graphs of D are closed under taking sub-graphs. An $\mathcal{F}$-transveral in D is a set of vertices that intersects every $F_i \in \mathcal{F}$.

The goal in an $\mathcal{F}$-transveral problem is to compute the minimum $\mathcal{F}-$transveral set. The following is an example of an $\mathcal{F}$-transveral problem.

Example 3 (Feedback Vertex Set)**.** Let $\mathcal{F}$ be the set of all directed cycle in a digraph D. Then, the feedback vertex set is $S \subseteq V(D)$ such that every directed cycle has a non-empty intersection with S.

Another example of $\mathcal{F}$-transveral problem is the Subset Feedback Vertex Set. Next, we will present the definition of this problem, and later we present an FPT-approximation algorithm.

Definition 3.2 (T-walk and T-cycle)**.** Let $D = (V(D), A(D))$ be a directed graph. A directed closed walk in D is an alternating sequence of vertices and arcs that starts and end on the same vertex. For a set $T \subseteq V(D) \cup A(D)$, a directed closed walk in D is said to be a T-closed walk if it contains an element from T. A T-closed walk is called a T-cycle if it is a simple cycle.

Definition 3.3 (T-sfvs (Subset Feedback Vertex Set))**.** A set $S \subseteq V(D)$ is called a $T - sfvs$ if it is intersects every T-cycle in D for a given $T \subseteq A(D) \cup V(D)$.

Thus, the goal in the Directed Subset Feedback Vertex Set is to compute a $T - sfvs$ of minimum size.

The following lemma will be crucial in the design on approximation algorithm for $\mathcal{F}$-transveral problems.

Lemma 3.1. *Let $\tilde{S}$ be an $\mathcal{F}$-transveral in D. Let $W = W_1 \uplus W_2$ be an $\mathcal{F}$-transveral in D such that for some $\phi \neq S \subseteq \tilde{S}$, S is a minimal $W_1 - W_2$ separator. Let X_{pre} and X_{post} be $W_1 - W_2$ separator in D such that $X_{pre} \sqsubseteq S \subseteq X_{post}$. Then $\tilde{S} - S$ is an $\mathcal{F}$-transveral in the graph $D' = D - (X_{pre} \cup X_{post})$.*

Proof. Suppose there is a graph $F \in \mathcal{F}$ such that $\tilde{S} - S$ does not intersect D'. Consider the graph $D'' = D' - (\tilde{S} - S)$, since both W and S are $\mathcal{F}$-transveral sets, and since F is assumed to be strongly connected, then there is a directed walk in F that intersects S and W_1, or S and W_2. Using observation 2, S is not reachable from W_1 which implies a contradiction. The other cases are analogous. □

Using the previous lemma, we can deduce the following one.

Lemma 3.2. *Let D,W_1,W_2, $\tilde{S}$, S be as defined in the previous lemma. Then, there exists an important $W_1 - W_2$ separator closest to W_1 of size at most $|S|$, call it X_{pre}, and an important $W_1 - W_2$ separator closest to W_2 of size at most $|S|$, call it X_{post}, such that $\tilde{S} - S$ is an important $\mathcal{F}$-transversal in $D' = D - (X_{pre} \cup X_{post})$.*

Now, we are ready to prove the following theorem by providing the necessary algorithms.

Theorem 3.3 ([LMR$^+$])**.** *There is a factor-2 FPT-approximation algorithm for Subset DFVS with running time $2^{O(k)}n^{O(1)}$.*

In order to give an algorithm for subset DFVS, first we give an algorithm for a special case. We call this special case Strict DFVS, and it is defined as follow.

Definition 3.4. [LMR$^+$] A factor-c FPT-approximation algorithm for Strict Subset DFVS in an algorithm take the input: (D, T, W, k). $D = (V(D), A(D))$ denotes a directed graph, $T \subseteq A(D)$, $W \subseteq V(D)$ is T-sfvs in D, and $k \in \mathbb{N}$. The output is a T-sfvs set in time $f(k).n^{O(1)}$ of size at most $c \cdot k$, if there is a T-sfvs S of size at most k, and W is contained in a unique strongly connected component of $D - S$. Otherwise, the output can be arbitrary.

Lemma 3.4 ([LMR$^+$])**.** *There is a factor-1 FPT-approximation algorithm for Strict Subset DFVS with running time $2^{O(k)}n^{O(1)}$. We call this algorithm Alg-Strict-SFVS.*

3.1 Proof of Lemma 3.4 [LMR$^+$]

Let $I = (D, T, W, k)$ be the given input. Suppose $(u, v) \in T$ such that $u, v \in W$, then the algorithm terminates with arbitrary output. S will break every cycle contains (u, v). Thus W will not be in a strongly connected component. Now, we construct a new tuple $I' = (D', T', w, k)$ where D' is obtained from D by identifying the vertices in W and T' is adjusted accordingly. w is the new vertex created in place of W.

Lemma 3.5. *The following statements hold.*

1. *w is a T-sfvs in D'*
2. *Every T-sfvs S in D that is disjoint from W such that W is contained in a unique strongly connected component of $D - S$, is a T'-sfvs in D' that is disjoint from w.*
3. *Conversely, every T'-sfvs in D' disjoint from w is a T-sfvs in D.*

Therefore, it is sufficient to give an algorithm for I'. But first we need the following lemma to complete the design of our algorithm.

Lemma 3.6. *Let S be a solution for I'. For every $(u,v) \in T'$, either $\{u,v\} \cap S \neq \phi$ or there is a solution for I' that contains an important $x-w$ separator closest to w for some $x \in \{u,v\}$.*

Proof. Suppose S is a T'-sfvs in D' of size at most k. We have S disjoint from w, and w lies in a strongly connected component C.

For every $e = \{u,v\} \in T'$, e cannot be in C because S will break all the cycle of C, e is an element of every cycle in C, therefore C will be no longer a strongly connected component.

This implies that either at least one of u or v is contained in S or S intersects all $w-x$ or $x-w$ paths for some $x \in \{u,v\}$.

Now, we will prove that if S intersects all $w-x$ or $x-w$ paths for some $x \in \{u,v\}$, then there is a solution S' that contains an important $w-x$ separator closest to w or an important $x-w$ separator closest to w for some $x \in \{u,v\}$.

Consider the case when S intersects all $w-x$ paths for some $x \in \{u,v\}$. The other case is similar. Let $\hat{S} \subseteq S$ be minimal $w-x$ separator in D'. Now, consider an important $w-x$ separator $\tilde{S}$ of size at most at most $|\hat{S}|$ that is covered by $\hat{S}$. We claim that $(S-\hat{S}) \cup \tilde{S}$ is a solution for I'. If it is not the case, then we can find a contradiction using Observation 2. □

Using Lemma 3.6 and 3.5 we can design the following algorithm for the Strict Subset DFVS.

Algorithm 5: Strict DFVS.
Recursively apply: for an arc $e = (u,v) \in T$

- Branch 1: add u to S, set $k = k-1$.
- Branch 2: add v to S ...
- Branch 3: Enumerate all important $u-w$ separator closest to w of size at most k.
- Branch 4: Enumerate all $v-w$ separators closest to w of size at most k.
- Branch 5: Enumerate all $w-u$ closest to w of size at most k.
- Branch 6: Enumerate all $w-v$ closest to w of size at most k.

□

Analysing the run-time for Algorithm 5, we get the following time complexity $(10+4\sqrt{6})^k n^{O(1)}$.

3.2 Proof of Theorem 3.3 [LMR$^+$]

We will make use of the following lemma.

Lemma 3.7. *Let D be a digraph, $T \subseteq A(D)$, and let W and S be disjoint T-sfvs in D. Let $\phi \neq W' \subseteq W$ be such that in $D-S$, there is a strongly connected component whose intersection with W is precisely W'. Consider the graph D' obtained from D by adding a bi-directed clique on W' (i.e., we add an arc (w,w') for every $w,w' \in W'$ such that $(w,w') \notin A(D)$). Then, W and S are both T-sfvs in D'.*

The algorithm follows the paradigm of Iterative Compression discussed in section 2.2.

Algorithm 6: Iterative-Compression-Subset DFVS.

1. $V(D) = \{v_1, ..., v_n\}$, $V_i = \cup_{j=1}^{i} v_j$ for $i \in [n]$.
2. For $X \subseteq V(D)$, let $T[X] = \{(x, y) \in T | x, y \in X\}$.
3. $W_1 = \{v_1\}$.
4. For $i = 1$ to n:
   (a) $I_i = (D[V_i], T[V_i], W_i, k)$
   (b) $S =$ Alg-Compression-SFVS(I_i).
   (c) $W_{i+1} := \{v_{i+1}\} \cup S$

□

The Alg-Compression-SFVS sub-routine is sepecified as follow.

Definition 3.5 (Alg-Compression-SFVS)**.**
Input: (D, T, W, k). D is digraph, $T \subseteq A(D)$, W is a solution of size at most $2k + 1$, and k is a positive integer.
Output: if there is a T-sfvs S in D of size at most k that is not necessarily disjoint from W then it outputs a T-sfvs in D of size at most $2k$ in $2^{O(k+|W|)}n^{O(1)}$.

As discussed in Section 2.2. we have to specify the disjoint problem in order to solve the compression problem.

Definition 3.6 (Alg-Disjoint-SFVS)**.**
Input: (D, T, W, k). D is digraph, $T \subseteq A(D)$, W is a solution of size at most $2k + 1$, and k is a positive integer.
Output: if there is a T-sfvs S in D of size at most k that is disjoint from W, then it outputs a T-sfvs in D of size at most $2k$ in $2^{O(k+|w|)}n^{O(1)}$.

Now, the main task is to design an algorithm for the disjoint Subset DFVS. We start by specifying the base case. If $k \leq 1$ or $|W| = 1$, we can solve the problem by brute force. If $k \leq 1$, it is sufficient to check whether there is a T-cycle in D and if yes, whether there is a T-sfvs in D of size at most 1. If $k > 1$, $|W| = 1$, then we can simply return W.

In order to describe the recursive case, we introduce the following definitions and notations.

Definition 3.7. $rel_D(X, Y)$ the set of all vertices that lie in a strongly connected component of $D - Y$ intersected by X.

Definition 3.8. Let $\mathcal{P}$ be the set of all 3-partitions of W into (X, Y, Z). For every $\tau = (X, Y, Z) \in \mathcal{P}$, we define the following tuples. Let $1 \leq i, j \leq k$.

- $\mathcal{L}^i[Z \longrightarrow XY]$ denotes the set of all important $Z - X \cup Y$ separators of size at most i closest $X \cup Y$.
- $\mathcal{L}^i[XY \longleftarrow Z]$ denotes the set of all important $Z - X \cup Y$ separators of size at most i closest to Z.

For $1 \leq i, j \leq k$, let $L_1 \in \mathcal{L}^j[Z \longrightarrow XY]$ and $L_2 \in \mathcal{L}^j[XY \longleftarrow Z]$.

- $\mathcal{L}^i[Y \longrightarrow X, L_1, L_2]$ denotes the set of all important $Y - X$ separators of size at most i closest to X in $D - (L_1 \cup L_2)$.

- $\mathcal{L}^i[X \longleftarrow Y, L_1, L_2]$ denotes the set of all important $Y - X$ separators of size at most i closest to Y in $D - (L_1 \cup L_2)$.

Moreover, we will use the following notation for the input. For $Q \subseteq V(D)$:

- $I[Q, Z, i]$ denotes the tuple $(D[rel(Z, Q)], T, Z, i)$.
- $I[Q, XY, i]$ denotes $(D[rel(X \cup Y), Q], T, X \cup Y, i)$.
- $I[Q, X, i]$ denotes $(D[rel(X, Q)], T, X, i)$.
- $\tilde{I}[Q, Y, i]$ denotes (D', T, Y, i), where D' is the graph obtained from $D[rel(Y, Q)]$ by adding a bi-directed clique on Y.

Now, we are ready to describe the remaining of the algorithm.

Algorithm 7: Alg-Disjoint-SFVS.

1. For every $(X, Y, Z) \in \mathcal{P}$ such that:
   - $|W|/3 \leq |X \cup Y|$ and $|Z| \leq 2|W|/3$.
   - $|Y| > |W|/3$.

   1.1 For every $1 \leq i_1 \leq k$, and for every i_2 such that $k_1 = i_1 + i_2 \leq k$:

   - **Step 1** Guess:
     - $L_1 \in \mathcal{L}^{i_1}[Z \longrightarrow XY]$.
     - $L_2 \in \mathcal{L}^{i_1}[XY \longleftarrow Z]$.
     - $L_3 \in \mathcal{L}^{i_2}[Y \longrightarrow X, L_1, L_2]$.
     - $L_4 \in \mathcal{L}^{i_2}[X \longleftarrow Y, L_1, L_2]$.
     - Set $Q = \cup_{q \in [4]} L_q$
   - **Step 2:** If $|W|/3 \leq |X \cup Y|, |Z| \leq 2|W|/3$, then:
     For every $i_3 + i_4 \leq k - k_1$:
     (a) $S_z =$ Alg-Dijoint-SFVS$(I[Q, Z, i_3])$
     (b) $S_{XY} =$ Alg-Disjoint-SFVS$(I[Q, XY, i_4])$.
     (c) $\Delta = Q \cup S_Z \cup S_{XY}$ is a T-sfvs in D of size at most $2k$, then we return Δ.
   - **Step 3:** if step 2 does not apply and $Y > |W|/3$, then:
     for every i_3, i_4, i_5 such that $i_3 + i_4 + i_5 = k - k_1$
     (a) $S_z =$ Alg-Dijoint-SFVS$(I[Q, Z, i_3])$.
     (b) $S_X =$ Alg-Dijoint-SFVS$(I[Q, X, i_4])$.
     (c) $S_Y =$ Alg-Strict-SFVS$(\tilde{I}[Q, Y, i_5])$.
     (d) if $\Delta = Q \cup S_Z \cup S_X \cup S_Y$ is a T-sfvs in D of size at most $2k$, then return Δ.

□

Now, we give the proof of correctness of the algorithm.

Proof. The correctness is proved by induction on $|W|$. The base case $|W| = 1$, in which case, the algorithm works on brute force and hence it is correct. Assume $|W| > 1$, suppose there is a T-sfvs S of size at most k in D Disjoint from W. We aim to show that the algorithm output a T-sfvs of size at most $2k$.

Let $(M_1, ..., M_r)$ denote the partition on W such that each M_i is contained in a strongly connected component of $D - S$, and for every $\ell_1 > \ell_2$, there is no path in $D - S$ from $rel_D(M_{\ell_1}, S)$ to $rel_D(M_{\ell_2,S})$. In other words, S is an $M_{\ell_1} - M_{\ell_2}$ separator for every $\ell_1 > \ell_2$.

Now, we will consider how W breaks after removing S.

Case 1: $|M_\ell| \leq \frac{|W|}{3}$ for every $\ell \in [r]$. Let $\ell' \in [r]$ denote the least value such that $|w|/3 < \sum_{i=1}^{\ell'} |M_i|$. Then,

$$\frac{|W|}{3} < \sum_{i=1}^{\ell'} |M_i| \leq \frac{2|W|}{3}.$$

Define $X = \phi$, $Y = \cup_{i=1}^{\ell'} M_i$, and $Z = \cup_{\ell'+1}^{r} M_i$. Therefore, we have

$$\frac{|W|}{3} \leq |X \cup Y|, |Z| \leq \frac{2|W|}{3}.$$

Then, when considering the partition (X, Y, Z), **step 2** would have been executed.

Let S_1 be a minimal subset of S that intersects all the $Z - X \cup Y$ paths in D. Let $i_1 = |S_1|$. Since D is strongly connected component it follows that $i_1 > 0$ by the two extremal separator lemma (Lemma 3.1 and 3.2), $L_1 \in \mathcal{L}^{i_1}[Z \longrightarrow XY]$ and $L_2 \in \mathcal{L}^{i_2}[XY \longrightarrow Z]$ such that $S' = S - S_1$ is a T-sfvs in $D - (L_1 \cup L_2)$. Let $i_2 = 0$ that implies $L_3 = L_4 = \phi$. We have $Q = \cup_{q \in [4]} L_q$, define:

- $S'_z = S' \cap rel(Z, Q)$, $i_3 = |S'_Z|$.
- $S'_{XY} = S' \cap rel(X \cup Y, Q), i_4 = |S'_{XY}|$.

By induction hypothesis we get a solution of size at most $2i_3 + 2i_4$. Combining all solutions we get one of size at most $2i_1 + 2i_3 + 2i_4 \leq 2k$.

Case 2: There is $\ell^* \in [r]$ such that $|M_{\ell^*}| > \frac{|W|}{3}$. Define $Y = M_{\ell^*}$. If $\ell^* = 1$, then define $X = \phi$. If $\ell^* = r$, then define $Z = \phi$. Otherwise, $X = \cup_{i=1}^{\ell^*-1} M_i$, and $Z = \cup_{i=\ell^*+1}^{r} M_i$. Then, the conditions of step 3 are satisfied, and **step 3** will be executed.

As before suppose S_1 is a minimal subset of S that intersects all $Z - X \cup Y$ paths in D. Let $i_1 = |S_1|$, by lemma 3.1 and 3.2, we have $L_1 \in \mathcal{L}^{i_1}[Z \longrightarrow XY]$ and $L_2 \in \mathcal{L}^{i_1}[XY \longleftarrow Z]$ such that $S' = S - S_1$ is a T-sfvs in $D - (L_1 \cup L_2)$.

Now, let S_2 be a minimal subset of S' that intersects all $Y - X$ paths in $D - (L_1 \cup L_2)$. Let $i_2 = |S_2|$. Then, Lemma 3.1 and 3.2 guarantees that there exists $L_3 \in \mathcal{L}^{i_2}[Y \longrightarrow X, L_1, L_2]$ and $L_4 \in \mathcal{L}^{i_2}[X \longleftarrow Y, L_1, L_2]$ such that $S'' = S' - S_2$ is a T-sfvs in $D - \cup_{q \in [4]} L_q$. Define $Q = \cup_{q \in [4]} L_q$, and

- $S''_Z = S'' \cap rel(Z, Q), i_3 = |S''_Z|$.
- $S''_X = S'' \cap rel(X, Q), i_4 = |S''_X$.
- $S''_Y = S'' \cap rel(Y, Q), i_5 = |S''_Y|$.

We have S''_Z is a T-sfvs of size at most i_3 in $D[rel(Z, Q)]$, S''_X is a T-sfvs of size at most i_4 in $D[rel(X, Q)]$. Using Lemma 3.7, we have S''_Y and Y are both T-sfvs in D' where D' is the graph by adding a bi-directed clique on Y in $D[rel(Y, Q)]$. Therefore, using the induction hypotheses and the correctness of the algorithm Alg-Strict-DFVS, we obtain S_Z of size at most $2i_3$, S_X of size at most $2i_4$, and S_Y of size i_5. In sum, $Q \cup S_X \cup S_Y \cup S_Z$ is a T-sfvs in D of size at most $2(i_1 + i_2 + i_3 + i_4 + i_5) \leq 2|S| \leq 2k$. □

Proving the run-time requires solving the following recurrence. Let $T(k,r)$ denotes the number of leaves generated by the instance (D,T,W,k) where $r=|W|$. Then,

$$T(k,r) \leq 3^r \sum_{k_1=1}^{k} 2^{5k_1} \cdot 2 \sum_{k_2+k_3 \leq k-k_1} T(k_2, \lfloor 2r/3 \rfloor) + T(K_3, \lfloor 2r/3 \rfloor).$$

and $T(1,r)=1$, $T(k,1)=1$. This complete the proof of Theorem 3.3

4 Conclusion

In this project, we presented some of the techniques used in FPT-algorithm. Then, we introducing the subject of FPT-approximation algorithm. Section 3 explains the two-extremal separator techniques used in [LMR$^+$] to design approximation algorithm. It illustrates the use of this techniques on the problem of Subset Feedback Vertex Set in directed graphs. However, this techniques is applied to other problems such as Directed Odd Cycle Transversal and Bi-directed Multicut. In addition, [LMR$^+$] leaves some open problems in this fields such as: is there a constant FPT-approximation for Planar Vertex Deletion or Chordal Vertex Deletion parameterised by the solution size that run in $2^{O(k)}n^{O(1)}$.

References

[CFK+15] M. Cygan, F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015.

[DML] Fixed parameter algorithms. `http://cs.bme.hu/~dmarx/papers/marx-warsaw-fpt1`.

[LMR+] Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. *FPT-approximation for FPT Problems*, pages 199–218.