Diversity of Solutions: An Exploration Through the Lens of Fixed-Parameter Tractability Theory*

Julien Baste[1], Michael R. Fellows[2], Lars Jaffke[2], Tomáš Masařík[3,4], Mateus de Oliveira Oliveira[2], Geevarghese Philip[5], and Frances A. Rosamond[2]

[1]University of Lille, France
julien.baste@univ-lille.fr
[2]University of Bergen, Norway
{michael.fellows,lars.jaffke,mateus.oliveira,frances.rosamond}@uib.no
[3]University of Warsaw, Poland
[4]Charles University, Czech Republic
masarik@kam.mff.cuni.cz
[5]Chennai Mathematical Institute and UMI Relax, India
gphilip@cmi.ac.in

Abstract

When modeling an application of practical relevance as an instance of a combinatorial problem X, we are often interested not merely in finding *one* optimal solution for that instance, but in finding a *sufficiently diverse* collection of good solutions. In this work we initiate a systematic study of *diversity* from the point of view of fixed-parameter tractability theory. First, we consider an intuitive notion of *diversity* of a collection of solutions which suits a large variety of combinatorial problems of practical interest. We then present an algorithmic framework which –*automatically*– converts a tree-decomposition-based dynamic programming algorithm for a given combinatorial problem X into a dynamic programming algorithm for the diverse version of X. Surprisingly, our algorithm has a polynomial dependence on the diversity parameter.

1 Introduction

In a typical combinatorial optimization problem, we are given a large space of potential solutions and an objective function. The task is to find a solution that maximizes or minimizes the objective function. In many situations of practical relevance, however, it does not really help to get just *one optimal* solution; it would be much better to have a small, but *sufficiently diverse* collection of *sufficiently good* solutions. Given such a small list of good solutions, we can select one which is best for our purpose, perhaps by taking into account external factors—such as aesthetical, political, or environmental—which are difficult or even impossible to formalize. An early, illustrative example is the problem of generating floor plans for evaluation by an architect [19].

Solution diversity is already a fundamental concept in many computational tasks. Take, for instance, a web search. Here, we do not want to find *the one* website that 'optimally fits' the search

*An extended abstract of this manuscript has appeared in the Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020 [4].

term, neither a ranking of a small number of 'best fits', but what is desirable is a *diverse set* of websites that fit the search term *reasonably well*.

Another advantage of considering a set of diverse solutions is that some of these solutions may find some use in contexts which are not specified à priori. For instance, in cutting problems [24], which are widely studied in the field of operations research, we are given a piece of material of standard size and a prescribed set of shapes. The goal is to cut the material into pieces of the specified shapes in such a way that the amount of leftover material is minimized. In this setting, a minimum-size leftover may be viewed as a solution. A set of sufficiently diverse solutions would give the user the opportunity to choose a suitable leftover that could be used later in the fabrication of pieces whose shapes have not been specified in the input of the program.

The notion of diversity has also been applied to solution sets of various types of combinatorial problems. For instance, the works [20] and [25, 26] seek to find solution sets to mixed-integer programming problems and constraint satisfaction problems, respectively, that are diverse. In other words, the solutions are far apart from each other in some mathematical notion of distance. We refer to [32] for a timely overview of the subject.

From a complexity-theoretic perspective, there are two immediate barriers to this approach. The first is that most combinatorial problems are already NP-hard when asking only for a single solution. The second is that the very basic Maximum Diversity problem, which given a set of n elements in a metric space and an integer $k < n$, asks for a size-k subset of the elements such that the sum of the pairwise distances is maximized, is NP-hard as well [28]. The theory of *fixed-parameter tractability* [14] provides a powerful framework to overcome these barriers. The key goal is to identify a secondary numerical measure of the inputs to an (NP-hard) computational problem, called the *parameter*, and to provide algorithms in whose runtime the combinatorial explosion is restricted to the parameter k. More formally, a problem is *fixed-parameter tractable (FPT)*, if it can be solved in time $f(k) \cdot n^c$, where f is a computable function, n the input size, and c a fixed constant. On instances where the parameter value is relatively small, FPT-algorithms are efficient. In an application context, we are naturally concerned with finding *small* diverse sets of solutions since the aim is to provide the user with a few alternatives that can then be compared manually. Therefore, the number of requested solutions is an ideal candidate for parameterization.

In this work, we propose to study the notion of solution diversity from the perspective of fixed-parameter tractability theory. We demonstrate the theoretical feasibility of this paradigm by showing that diverse variants of a large class of parameterized problems admits FPT-algorithms. Specifically, we consider *vertex-problems* on graphs, which are sets of pairs (G, S) of a graph G and a subset S of its vertices that satisfies some property. For instance, in the Vertex Cover problem, we require the set S to be a vertex cover of G (i.e., S has to contain at least one endpoint of each edge of G). One consequence of our main result which we discuss below in more detail is that the diverse variant of Vertex Cover, asking for r solutions, is FPT when parameterized by solution size plus r.

Before we proceed, we would like to point out promising future applications of the *Diverse FPT* paradigm in AI. The Vertex Cover problem itself naturally models conflict-resolution: the entities are the vertices of the graph, and a conflict is represented by an edge. Now, a vertex cover of the resulting graph is a set of entities whose removal makes the model conflict-free. An example of a potential use of Diverse Vertex Cover in a planning scenario is given in [5]. In general, in planning and scheduling problems, a large amount of *side information* is lost or intentionally omitted in the modeling process. Some side information could make the model too complex to be solved, and other information may even be impossible to model. Offering the user a small number of good solutions to a more easily computable 'base model', among which they can handpick their

favorite solution is a feasible alternative.

A Formal Notion of Diversity We choose a very natural and general measure as our notion of diversity among solutions. Given two subsets S and S' of a set V the *Hamming distance* between S and S' is the number

$$\text{HamDist}(S, S') = |S \backslash S'| + |S' \backslash S|.$$

We define the *diversity of a list* $S_1, \ldots, S_r$ of subsets of V to be

$$\text{Div}(S_1, \ldots, S_r) = \sum_{1 \leq i < j \leq r} \text{HamDist}(S_i, S_j).$$

We can now define the diverse version of vertex-problems:

Definition 1.1 (Diverse Problem). Let $\mathcal{P}_1, \ldots, \mathcal{P}_r$ be vertex-problems, and let $d \in \mathbb{N}$. We let

$$\text{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r) = \{(G, X_1, \ldots, X_r) \mid (G, X_i) \in \mathcal{P}_i, \\ \text{Div}(X_1, \ldots, X_r) \geq d\}.$$

Intuitively, given vertex-problems $\mathcal{P}_1, \ldots, \mathcal{P}_r$ and a graph G, we want to find subsets $S_1, \ldots, S_r$ of vertices of G such that for each $i \in \{1, \ldots, r\}$, S_i is a solution for problem $\mathcal{P}_i$ on input G, and such that the list $S_1, \ldots, S_r$ has diversity at least d. If all vertex-problems $\mathcal{P}_1, \ldots, \mathcal{P}_r$ are the same problem $\mathcal{P}$, then we write $\text{Div}^d_r(\mathcal{P})$ as a shortcut to $\text{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$.

Diversity and Dynamic Programming The treewidth of a graph is a structural parameter that quantifies how close the graph is to being a forest (i.e., a graph without cycles). The popularity of this parameter stems from the fact that many problems that are NP-complete on general graphs can be solved in polynomial time on graphs of constant treewidth. In particular, a celebrated theorem due to Courcelle [11] states that any problem expressible in the monadic second-order logic of graphs can be solved in polynomial time on graphs of constant treewidth. Besides this metatheorem, the notion of treewidth has found applications in several branches of Artificial Intelligence such as Answer Set Programs [6], checking the consistency of certain relational algebras in Qualitative Spacial Reasoning [7], compiling Bayesian networks [10], determining the winners of multi-winner voting systems [36], analyzing the dynamics of stochastic social networks [3], and solving constraint satisfaction problems [27]. A large number of these algorithms are in fact FPT-algorithms when treewidth is the parameter. Typically, such algorithms are dynamic programming algorithms which operate on a tree-decomposition in a bottom-up fashion by computing data from the leaves to the root.

Dynamic Programming Core Model We introduce a formalism for dynamic programming based on a tree decomposition, which we call the *Dynamic Programming Core* model. This notion captures a large variety of dynamic programming algorithms on tree decompositions. We use the model to derive our main result (Theorem 4.5) which is a framework to efficiently—and automatically—transform treewidth-based dynamic programming algorithms for vertex-problems into algorithms for the diverse versions of these problems. More precisely, we show that if $\mathcal{P}_1, \ldots, \mathcal{P}_r$ are vertex-problems where, for each $i \in \{1, \ldots, r\}$, $\mathcal{P}_i$ can be solved in time $f_i(t) \cdot n^{\mathcal{O}(1)}$, then $\text{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$ can be solved in time $(\prod_{i=1}^r f_i(t)) \cdot n^{\mathcal{O}(1)}$. In particular, if a vertex-problem $\mathcal{P}$ can be solved in time $f(t) \cdot n^{\mathcal{O}(1)}$, then its diverse version $\text{Div}^d_r(\mathcal{P})$ can be solved in time $f(t)^r \cdot n^{\mathcal{O}(1)}$. The surprising aspect of this result is that the running time depends only *polynomially* on d (which is at most $r^2 n$), while a naïve dynamic programming algorithm would have a runtime of $d^{\mathcal{O}(r^2)} \cdot f(t)^r \cdot n^{\mathcal{O}(1)}$.

Discussion of the Diversity Measure Various measures of diversity have been used, studied, and compared in different areas of computer science. We choose the *sum* of the Hamming distances over all pairs of elements for this work. This measure is commonly used for population diversity in genetic algorithms [18, 35]. Nonetheless, we would like to point out that it has some weaknesses. For instance, taking many copies of two disjoint solutions yields a relatively high diversity value, and such a solution set is not 'diverse' from an intuitive point of view. We refer to [5] for a more detailed discussion. Another natural measure using the Hamming distance is the *minimum* Hamming distance over all pairs in a set, as it is done e.g. in [25, 26]. We would like to point out that a straightforward adaptation of our algorithmic framework would result in a running time of $d^{\mathcal{O}(r^2)} \cdot f(t)^r \cdot n^{\mathcal{O}(1)}$, where d is the diversity, r the number of solutions, and t the treewidth. This remains FPT only when the diversity d is an additional component of the parameter, or when d is naturally upper bounded by t and r. Consider for instance Diverse Vertex Cover, asking for vertex covers of size at most k. In any nontrivial instance, t is at most k, and the Hamming distance between two solutions is at most $2k$, therefore we may assume that $d \leq 2k$. This implies that Diverse Vertex Cover can be solved in time $2^{\mathcal{O}(r^2 \log k)+kr} \cdot n^{\mathcal{O}(1)}$ using the minimum Hamming distance as a diversity measure.

Related Work The above-mentioned Maximum Diversity problem has applications in the generation of diverse query results, see e.g. [21, 1]. Besides mixed integer programming [20, 13, 31], binary integer linear programming [22, 34] and constraint programming [25, 26], diverse solution sets have been considered in SAT solving [30], recommender systems [2], routing problems [33], answer set programming [15], and decision support systems [29, 23].

2 Preliminaries

For positive integers a and b, with $a < b$, we use $[a, b]$ to denote the set $\{a, a+1, \ldots, b\}$. We use $V(G)$ and $E(G)$, respectively, to denote the vertex and edge sets of a graph G. For a tree T rooted at q we use T_t to denote the subtree of T rooted at a vertex $t \in V(T)$. A *rooted tree decomposition* of a graph G is a tuple $\mathcal{D} = (T, q, \mathcal{X})$, where T is a tree rooted at $q \in V(T)$ and $\mathcal{X} = \{X_t \mid t \in V(T)\}$ is a collection of subsets of $V(G)$ such that:

- $\bigcup_{t \in V(T)} X_t = V(G)$,
- for every edge $\{u, v\} \in E(G)$, there is a $t \in V(T)$ such that $\{u, v\} \subseteq X_t$, and
- for each $\{x, y, z\} \subseteq V(T)$ such that z lies on the unique path between x and y in T, $X_x \cap X_y \subseteq X_z$.

We say that the vertices of T are the *nodes* of $\mathcal{D}$ and that the sets in $\mathcal{X}$ are the *bags* of $\mathcal{D}$. Given a node $t \in V(T)$, we denote by G_t the subgraph of G induced by the set of vertices $\bigcup_{s \in V(T_t)} X_s$. The *width* of a tree decomposition $\mathcal{D} = (T, q, \mathcal{X})$ is defined as $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of a graph G, denoted by $\mathsf{tw}(G)$, is the smallest integer w such that there exists a rooted tree decomposition of G of width at most w. The *rooted path decomposition* of a graph is a rooted tree decomposition $\mathcal{D} = (T, q, \mathcal{X})$ such that T is a path and q is a vertex of degree 1. The *pathwidth* of a graph G, denoted by $\mathsf{pw}(G)$, is the smallest integer w such that there exists a rooted path decomposition of G of width at most w. Note that in a rooted path decomposition, every node has at most one child.

For convenience we will always assume that the bag associated to the root of a rooted tree decomposition is empty. For a node $t \in V(T)$ we use $\delta_{\mathcal{D}}(t)$, or $\delta(t)$ when $\mathcal{D}$ is clear from the

context, to denote the number of children of t in the tree T. For nodes t and t' of $V(T)$ where t' is the parent of t we use $\text{forg}(t) = X_t \setminus X_{t'}$ to denote the set of vertices of G which are *forgotten* at t. By convention, for the root q of T, we let $\text{forg}(q) = \emptyset$. For $t \in V(T)$ we denote by $\text{new}(t)$ the set $X_t \setminus \bigcup_{i=1}^{\delta(t)} X_{t_i}$ where $t_1, \ldots, t_{\delta(t)}$ are the children of t. Given a rooted tree decomposition $\mathcal{D}$ of a graph G one can obtain, in linear time, a tree decomposition $(T, q, \mathcal{X})$ of G of the same width as $\mathcal{D}$ such that for each $t \in V(T)$, $\delta(t) \leq 2$ and $|\text{new}(t)| \leq 1$ [12]. From now on we assume that every rooted tree decomposition is of this kind.

3 A First Example: Diverse Vertex Cover

The main result of this paper is a general framework to automatically translate tree-decomposition-based dynamic programming algorithms for vertex-problems into algorithms for the diverse versions of these problems. We develop this framework in Section 4. In this section we illustrate the main techniques used in this conversion process by showing how to translate a tree-decomposition-based dynamic programming algorithm for the Vertex Cover problem into an algorithm for its diverse version Diverse Vertex Cover. Given a graph G and three integers k, r, and d, the Diverse Vertex Cover problem asks whether one can find r vertex covers in G, each of size at most k, such that their diversity is at least d. Our algorithm for this problem will run in $2^{\mathcal{O}(kr)}|V(G)|$ time.

3.1 Incremental Computation of Diversity

Recall that we defined the diversity of a list $S_1, S_2, \ldots, S_r$ of subsets of a set V to be

$$\text{Div}(S_1, \ldots, S_r) = \sum_{1 \leq i < j \leq r} \text{HamDist}(S_i, S_j).$$

We will now describe a way to compute the diversity $\text{Div}(S_1, \ldots, S_r)$ in an incremental fashion, by incorporating the influence of each element of V in turn. For each element $v \in V$ and each pair of subsets S, S' of V, we define $\gamma(S, S', v)$ to be 1 if $v \in (S \setminus S') \cup (S' \setminus S)$, and to be 0 otherwise. Intuitively, $\gamma(S, S', v)$ is 1 if and only if the element v contributes to the Hamming distance between S and S'. Given this definition we can rewrite $\text{HamDist}(S, S')$ as

$$\text{HamDist}(S, S') = \sum_{v \in V} \gamma(S, S', v),$$

and the diversity of a list $S_1, \ldots, S_r$ of subsets of V as

$$\begin{aligned} \text{Div}(S_1, \ldots, S_r) &= \textstyle\sum_{1 \leq i < j \leq r} \sum_{v \in V} \gamma(S_i, S_j, v) \\ &= \textstyle\sum_{v \in V} |\{\ell \ : \ v \in S_\ell\}| \cdot |\{\ell \ : \ v \notin S_\ell\}|. \end{aligned}$$

Now, if we define the *influence* of v on the list $S_1, \ldots, S_r$ as

$$I(S_1, \ldots, S_r, v) = |\{\ell \ : \ v \in S_\ell\}| \cdot |\{\ell \ : \ v \notin S_\ell\}|,$$

then we have that

$$\text{Div}(S_1, \ldots, S_r) = \sum_{v \in V} I(S_1, \ldots, S_r, v). \tag{1}$$

3.2 From Vertex Cover to Diverse Vertex Cover

We solve Diverse Vertex Cover using dynamic programming over a tree decomposition of the input graph. An excellent exposition of tree-width-based dynamic programming algorithms can be found in [12, Chapter 7]. We sketch here how finding a single solution of the Vertex Cover problem can be easily achieved given a tree-decomposition of width w. We store all possible intersections of potential solutions with a particular node of the tree-decomposition while remembering the (smallest) size of each such partial solution on the whole subtree of the node. This information can be computed starting with the leaves of the tree-decomposition up to the root using dynamic programming. We refer to the information stored in each node of the decomposition as *table* and its *size* is the number of entries in the table which corresponds to the number of potential solutions. Then the resulting solution—the vertex cover of the smallest size—is the smallest solution among those stored in the root. For each node, the size of the table is at most 2^w, which is, in other words, at most 2^w possible solution intersections with the node. Then, the update step of the dynamic programming takes linear time. Observe that the table size does not depend upon the size of the vertex cover in the subtree of the node for the above approach, but it is sufficient to have a decomposition of bounded width. The existence of such a small width decomposition is implied by the existence of vertex cover of small size, but not vice-versa. As this section serves as an illustration of our approach, we will limit ourselves to vertex covers of size at most k for k being a parameter. However, such a limitation is not needed for the main theorem, where we rely only on the width of the decomposition.

Now, we modify the dynamic programming approach to compute r vertex covers each of size k that are as diverse as possible under the Hamming distance measure. Let us remark here that one can always compute all possible solutions, in this case, all possible vertex covers, but such an approach could be quite costly. Instead, we will store only several r-tuples composed of possible intersections of solutions with the particular node. As r becomes an additional parameter for our diversity paradigm, this translates to an FPT algorithm with only polynomial dependency on the target diversity d.

Let (G, k, r, d) be an instance of Diverse Vertex Cover and let $\mathcal{D} = (T, q, \mathcal{X})$ be a rooted tree decomposition of G. For each node $t \in V(T)$, we define the set

$$\mathcal{I}_t = \{((S_1, s_1), \ldots, (S_r, s_r), \ell) \mid \ell \in [0, d], \forall i \in [1, r], S_i \subseteq X_t, s_i \in [0, k]\}.$$

This set $\mathcal{I}_t$, $t \in V(T)$, is such that the partial solutions we will construct for the node t will always be a subset of $\mathcal{I}_t$. Note that for each $t \in V(T)$, $|\mathcal{I}_t| \leq (2^{|X_t|} \cdot (k+1))^r \cdot (d+1)$. Now, our dynamic programming algorithm for Diverse Vertex Cover consists in constructing for each $t \in V(T)$ a subset $\mathcal{R}_t \subseteq \mathcal{I}_t$ as follows. Let t be a node in $V(T)$ with children $t_1, \ldots, t_{\delta(t)}$. We recall that, by convention, this set of children is of size 0, 1, or 2. We let $\mathcal{R}_t$ be the set of all tuples $((S_1, s_1), \ldots, (S_r, s_r), \ell) \in \mathcal{I}_t$ satisfying the following additional properties:

1. For each $j \in [1, r]$, $E(G[X_t \setminus S_j]) = \emptyset$.
2. For each $i \in [1, \delta(t)]$ there exists a tuple $((S_1^i, s_1^i), \ldots, (S_r^i, s_r^i), \ell_i)$ in $\mathcal{R}_{t_i}$ such that
   (a) $S_j \cap X_{t_i} = S_j^i \cap X_t$ for each $i \in [1, \delta(t)]$ and each $j \in [1, r]$,
   (b) For each $j \in [1, r]$, $s_j = |\text{forg}(t) \cap S_j| + \sum_{i=1}^{\delta(t)} s_j^i$,
   (c) and $\ell = \min(d, m)$ where $m = \sum_{v \in \text{forg}(t)} I(S_1, \ldots, S_r, v) + \sum_{i=1}^{\delta(t)} \ell_i$.

Lemma 3.1. *(G, k, r, d) is a* Yes*-instance of* Diverse Vertex Cover *if and only if there is a tuple $((S_1, s_1), \ldots, (S_r, s_r), \ell)$ in $\mathcal{R}_q$ such that $\ell = d$.*

Proof. Using induction, one can see that for each $t \in V(T)$, $\mathcal{R}_t$ is the set of every element of $\mathcal{I}_t$ such that, with $Y_t = X_t \setminus \text{forg}(t)$, there exists $(\widehat{S}_1, \ldots, \widehat{S}_r) \in V(G_t)^r$, that satisfies:

- for each $i \in [1, r]$, $\widehat{S}_i$ is a vertex cover of G_t,
- for each $i \in [1, r]$, $\widehat{S}_i \cap X_t = S_i$,
- for each $i \in [1, r]$, $|\widehat{S}_i \setminus Y_t| = s_i$, and
- $\min(d, \text{Div}(\widehat{S}_1 \setminus Y_t, \ldots, \widehat{S}_r \setminus Y_t)) = \ell$.

As the root q of the tree decomposition $\mathcal{D}$ is such that $X_q = \emptyset$, we obtain that the elements in $\mathcal{R}_q$ are the elements $((\emptyset, s_1), \ldots, (\emptyset, s_r), \ell)$ of $\mathcal{I}_q$ such that there exists $(\widehat{S}_1, \ldots, \widehat{S}_r) \in V(G)^r$, that satisfy,

- for each $i \in [1, r]$, $\widehat{S}_i$ is a vertex cover of G_t,
- for each $i \in [1, r]$, $|\widehat{S}_i| = s_i \leq k$, and
- $\min(d, \text{Div}(\widehat{S}_1, \ldots, \widehat{S}_r)) = \ell$.

As such, a tuple $(\widehat{S}_1, \ldots, \widehat{S}_r)$ of subsets of $V(G)$ is a solution of Diverse Vertex Cover if and only if $\ell \geq d$, the lemma follows. □

Theorem 3.2. *Given a graph G, integers k, r, d, and a rooted tree decomposition $\mathcal{D} = (T, q, \mathcal{X})$ of G of width w, one can determine whether (G, k, r, d) is a* Yes*-instance of* Diverse Vertex Cover *in time*

$$\mathcal{O}(2^r \cdot (2^{w+1} \cdot (k+1))^{a \cdot r} \cdot d^a \cdot w \cdot r \cdot n),$$

where $a = \max_{t \in V(T)} \delta(t) \leq 2$ and $n = |V(T)|$.

Proof. Let us analyze the time needed to compute $\mathcal{R}_q$. We have that, for each $t \in V(\mathcal{D})$, $|\mathcal{I}_t| \leq (2^{\cdot |X_t|} \cdot (k+1))^r \cdot (d+1)$. Note that given $I_1, \ldots, I_{\delta(t)}$ be elements of $\mathcal{R}_{t_1}, \ldots, \mathcal{R}_{t_{\delta(t)}}$, there are at most $2^{|\text{new}(t)| \cdot r} \leq 2^r$ ways to create an element I of $\mathcal{R}_t$ by selecting, or not the (potential) new element of X_t for each set S_i, $i \in [1, r]$. The remaining is indeed fixed by $I_1, \ldots, I_{\delta(t)}$. Thus, $\mathcal{R}_t$ can be computed in time $\mathcal{O}(r \cdot |X_t| \cdot 2^r \cdot \prod_{i=1}^{\delta(t)} |\mathcal{R}_{t_i}|)$, where the factor $r \cdot |X_t|$ appears when verifying that the element we construct satisfy $\forall j \in [1, r], E(G[X_j \setminus S_j]) = \emptyset$. As we need to compute $\mathcal{R}_t$ for each $t \in V(\mathcal{D})$ and that $|V(\mathcal{D})| = \mathcal{O}(n)$ and we can assume that $\delta(t) \leq 2$ for each $t \in V(\mathcal{D})$, the theorem follows. □

Remark 3.3. Given a graph G and a vertex cover Z of G of size k, one can find a rooted path decomposition $\mathcal{D} = (T, q, \mathcal{X})$ of G of width k, in linear time.

This can be done by considering the bags $Z \cup \{v\}$ for each $v \in V(G)$ in any fixed order. Thus, from Theorem 3.2, we get the following corollary, which establishes an upper bound for the running time of our dynamic programming algorithm for Diverse Vertex Cover solely in terms of the size k of the vertex cover, the number r of requested solutions, and the diversity d.

Corollary 3.4. Diverse Vertex Cover *can be solved on an input (G, k, r, d) in time $\mathcal{O}((2^{k+2} \cdot (k+1))^r \cdot d \cdot k \cdot r \cdot |V(G)|)$.*

4 Computing Diverse Solutions using the Dynamic Programming Core model

In this section, we show that the process illustrated in Section 3, of lifting a dynamic programming algorithm for a combinatorial problem to an algorithm for its diverse version, can be generalized to a much broader context. As a first step, we introduce the notion of *dynamic programming core*, a suitable formalization of the *intuitive* notion of tree-width based dynamic programming that satisfies three essential properties. First, this formalization is general enough to be applicable to a large class of combinatorial optimization problems. Second, this formalization is compatible with the notion of diversity, in the sense that the lifting of an algorithm for a problem to an algorithm for the diverse version of this problem can be done automatically, without requiring human ingenuity. Third, the resulting lifted algorithm is fast when compared with the original one. In particular, the running time of the resulting algorithm is polynomial in the diversity (the pairwise sum of Hamming distances). So, the only important new factor in the running time is the number of the solutions, which we consider an additional parameter. This is a highly desired property since this allows our framework to be applied in the context where the sizes of the considered solution sets are not bounded.

Below, we let $\mathcal{G}$ be the set of simple, undirected graphs whose vertex set is a finite subset of $\mathbb{N}$. We say that a subset $\mathcal{P} \subseteq \mathcal{G}$ is a graph problem. Intuitively, a dynamic programming algorithm working on tree decompositions may be understood as a procedure that takes a graph $G \in \mathcal{G}$ and a rooted tree decomposition $\mathcal{D}$ of G as input, and constructs a certain amount of data for each node of $\mathcal{D}$. The data at node t is constructed by induction on the height of t, and in general, this data is used to encode the existence of a partial solution on the graph induced by bags in the sub-tree of $\mathcal{D}$ rooted at t. In Definition 4.1 below, this is captured in the relation $\text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$. Such an algorithm accepts the input graph G if the data associated with the root node contains a string belonging to a set of *accepting strings*, captured below in the set $\text{Accept}_{\mathfrak{C},G,\mathcal{D}}$. We formalize this intuitive notion in the following concept of *dynamic programming core*; see Figure 1.

Definition 4.1 (Dynamic Programming Core). A *dynamic programming core* is an algorithm $\mathfrak{C}$ that takes a graph $G \in \mathcal{G}$ and a rooted tree decomposition $\mathcal{D}$ of G as input, and produces the following data.

- A finite set $\text{Accept}_{\mathfrak{C},G,\mathcal{D}} \subseteq \{0,1\}^*$.
- A finite set $\text{Process}_{\mathfrak{C},G,\mathcal{D}}(t) \subseteq (\{0,1\}^*)^{\delta(t)+1}$ for each $t \in V(\mathcal{D})$.

We let $\tau(\mathfrak{C}, G, \mathcal{D})$ be the overall time necessary to construct the data associated with all nodes of $\mathcal{D}$. The size of $\mathfrak{C}$ on a pair $(G, \mathcal{D})$ is defined as

$$\text{Size}(\mathfrak{C}, G, \mathcal{D}) = \max\{|\text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| \mid t \in V(\mathcal{D})\}.$$

Next, we define the notion of a *witness* for a dynamic programming core. Intuitively such witnesses are certificates of the existence of a solution.

Definition 4.2. Let $\mathfrak{C}$ be a dynamic programming core, G be a graph in $\mathcal{G}$, and $\mathcal{D} = (T, q, \mathcal{X})$ be a rooted tree decomposition of G. A $(\mathfrak{C}, G, \mathcal{D})$-*witness* is a function $\alpha : V(T) \to \{0,1\}^*$ such that the following conditions are satisfied.

1. For each $t \in V(T)$, with children $t_1, \ldots, t_{\delta(t)}$, $(\alpha(t), \alpha(t_1), \ldots, \alpha(t_{\delta(t)})) \in \text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$.

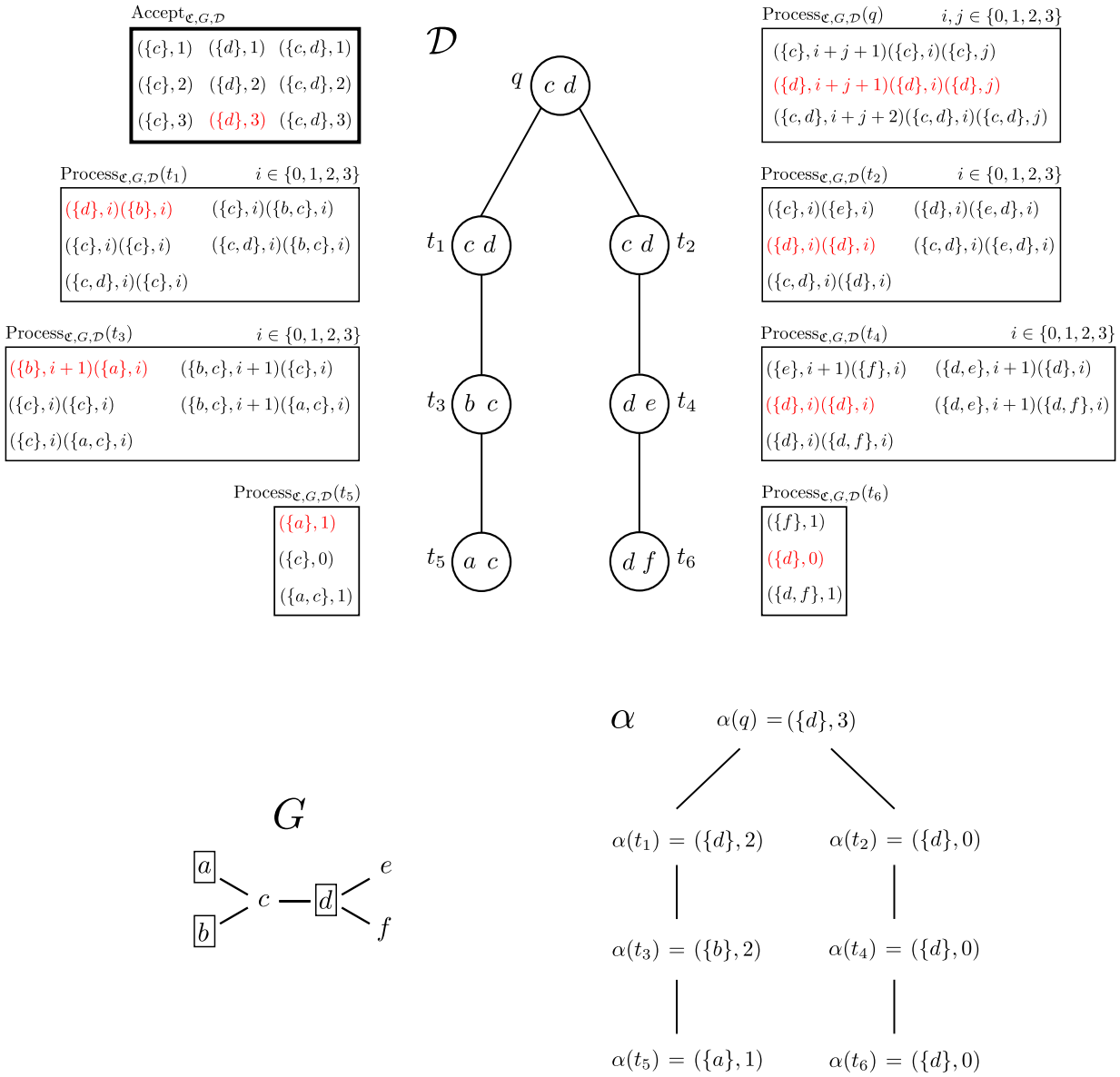

Figure 1: An illustration of the dynamic programming core model (Definition 4.1) using the vertex cover problem as an example. $\mathcal{D}$ is a tree decomposition of G rooted at q. In this example, the restriction of a partial solution to a given bag is represented by a pair (S, s) where S is a subset of the bag and $s \in \{0, 1, 2, 3\}$. Formally, such a pair would be encoded in binary. The dynamic programming core $\mathfrak{C}$ assigns to each node t of $\mathcal{D}$ a set Process$_{\mathfrak{C},G,\mathcal{D}}(t)$ of tuples of the form $((S, s), (S_1, s_1), \ldots, (S_{\delta(t)}, s_{\delta_t}))$, where $\delta(t)$ is the number of children of t. The function α is a $(\mathfrak{C}, G, D)$-witness (Definition 4.2) because $\alpha(q) = (\{d\}, 3)$ belongs to Accept$_{\mathfrak{C},G,\mathcal{D}}$ and for each node t, the tuple $(\alpha(t), \alpha(t_1), \ldots, \alpha(t_{\delta(t)}))$ belongs to Process$_{\mathfrak{C},G,\mathcal{D}}(t)$. For instance, the tuple $((\{d\}, 3), (\{d\}, 2), (\{d\}, 0))$ belongs to Process$_{\mathfrak{C},G,\mathcal{D}}(q)$ and $((\{d\}, 2), (\{b\}, 2))$ belongs to Process$_{\mathfrak{C},G,\mathcal{D}}(t_1)$. Intuitively, α represents a solution to the given problem. In this case, a vertex cover of size at most 3 (the vertex cover $\{a, b, d\}$ in the graph G). A formal specification of the core in this example will be given in Section 4.2.

2. $\alpha(q) \in \text{Accept}_{\mathfrak{C},G,\mathcal{D}}$.

Using the notion of witness, we define formally what it means for a dynamic programming core to solve a combinatorial problem.

Definition 4.3. We say that a dynamic programming core $\mathfrak{C}$ solves a problem $\mathcal{P}$ if for each graph $G \in \mathcal{G}$, and each rooted tree decomposition $\mathcal{D}$ of G, $G \in \mathcal{P}$ if and only if a $(\mathfrak{C}, G, \mathcal{D})$-witness exists.

In other words, if a dynamic programming core $\mathfrak{C}$ solves a problem $\mathcal{P}$, then for each each graph $G \in \mathcal{P}$ and each tree decomposition $\mathcal{D}$ of G there exists at least one $(\mathfrak{C}, G, \mathcal{D})$-witness. On the other hand, for each graph $G \notin \mathcal{P}$ and each tree decomposition $\mathcal{D}$ of G, no such a $(\mathfrak{C}, G, \mathcal{D})$-witness exists.

Theorem 4.4. *Let $\mathcal{P}$ be a graph problem and $\mathfrak{C}$ be a dynamic programming core that solves $\mathcal{P}$. Given a graph $G \in \mathcal{G}$ and a rooted tree decomposition $\mathcal{D}$ of G, one can determine whether $G \in \mathcal{P}$ in time*

$$\mathcal{O}\left(\sum_{t \in V(T)} |\text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| + \tau(\mathfrak{C}, G, \mathcal{D})\right).$$

Proof. Given $\mathfrak{C}$, G, and $\mathcal{D} = (T, q, \mathcal{X})$, we construct the set $\text{Accept}_{\mathfrak{C},G,\mathcal{D}}$ and the sets $\text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$ for each $t \in V(\mathcal{D})$. By definition, this can be done in time $\tau(\mathfrak{C}, G, \mathcal{D})$.

Given $t \in V(T)$ and $w \in \{0,1\}^*$, a $(\mathfrak{C}, G, \mathcal{D}, t, w)$-*witness* is a function

$$\beta : V(\text{subtree}(T, t)) \to \{0,1\}^*$$

such that for each $t' \in V(\text{subtree}(T, t))$, with children $t_1, \ldots, t_{\delta(t)}$,

$$(\beta(t'), \beta(t_1), \ldots, \beta(t_{\delta(t)})) \in \text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$$

and $\beta(t) = w$. Note that there exists a $(\mathfrak{C}, G, \mathcal{D})$-witness if and only if there exists a $(\mathfrak{C}, G, \mathcal{D}, q, w)$-witness for some $w \in \{0,1\}^*$.

For each $t \in V(T)$, we define $\Pi(G, \mathcal{D}, t)$ to be the set of every $w \in \{0,1\}^*$ such that there exists a $(\mathfrak{C}, G, \mathcal{D}, t, w)$-witness. Let $t \in V(T)$ and assume that we are able to construct $\Pi(G, \mathcal{D}, t_i)$ for every $i \in [1, \delta(t)]$ where $t_1, \ldots, t_{\delta(t)}$ are the children of t. We can then construct $\Pi(G, \mathcal{D}, t)$ as follows. For each $(w, w_1, \ldots, w_{\delta(t)}) \in \text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$, we add w to $\Pi(G, \mathcal{D}, t)$ if for each $i \in [1, \delta(t)]$, $w_i \in \Pi(G, \mathcal{D}, t_i)$. It is easy to see that for each such w, there exists a $(\mathfrak{C}, G, \mathcal{D}, t, w)$-witness that is an extension of the $(\mathfrak{C}, G, \mathcal{D}, t_i, w_i)$-witness, $i \in [1, \delta(t)]$. Moreover if there exists a $(\mathfrak{C}, G, \mathcal{D}, t, w)$-witness β for some $w \in \{0,1\}^*$, then, for each $i \in [1, \delta(t)]$, the restriction of β to subtree(T, t_i) is a $(\mathfrak{C}, G, \mathcal{D}, t_i, w_i)$-witness for some $w_i \in \{0,1\}^*$, and so, by induction hypothesis, $w_i \in \Pi(G, \mathcal{D}, t_i)$. This implies that our construction has correctly added w to $\Pi(G, \mathcal{D}, t)$. Thus $\Pi(G, \mathcal{D}, t)$ is correctly constructed. From Definition 4.3, we have that $G \in \mathcal{P}$ if and only if $\text{Accept}_{\mathfrak{C},G,\mathcal{D}} \cap \Pi(G, \mathcal{D}, q) \neq \emptyset$. Note that the time needed to construct $\Pi(G, \mathcal{D}, q)$ is $\mathcal{O}\left(\sum_{t \in V(T)} |\text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)|\right)$. Therefore, the theorem follows. □

4.1 Dynamic Programming Cores for Vertex Problems

Let $\mathfrak{C}$ be a dynamic programming core. A $\mathfrak{C}$-*vertex-membership function* is a function $\rho : \mathbb{N} \times \{0,1\}^* \to \{0,1\}$ such that for each graph G, each rooted tree decomposition $\mathcal{D} = (T, q, \mathcal{X})$ of G and each $(\mathfrak{C}, G, \mathcal{D})$-witness α, it holds that $\rho(v, \alpha(t)) = \rho(v, \alpha(t'))$ for each edge $(t, t') \in E(T)$ and

each vertex $v \in X_t \cap X_{t'}$. Intuitively, if G is a graph and $\mathcal{D}$ is a rooted tree decomposition of G, then a $\mathfrak{C}$-vertex-membership together with a $(\mathfrak{C}, G, \mathcal{D})$-witness, provide an encoding of a subset of vertices of the graph. More precisely, we let

$$S_\rho(G, \mathcal{D}, \alpha) = \{v \mid \exists t \in V(T_\mathcal{D}), \rho(v, \alpha(t)) = 1\}$$

be this encoded vertex set. Given a $\mathfrak{C}$-vertex-membership function ρ, we let $\hat{\rho} : \{0,1\}^* \to 2^\mathbb{N}$ be the function that sets $\hat{\rho}(w) = \{v \in \mathbb{N} \mid \rho(v, w) = 1\}$ for each $w \in \{0,1\}^*$.

Let $\mathcal{P}$ be a vertex-problem, $\mathfrak{C}$ be a dynamic programming core, and ρ be a $\mathfrak{C}$-vertex-membership function. We say that $(\mathfrak{C}, \rho)$ *solves* $\mathcal{P}$ if for each graph $G \in \mathcal{G}$, each subset $S \subseteq V(G)$, and each rooted tree decomposition $\mathcal{D}$, $(G, S) \in \mathcal{P}$ if and only if there exists a $(\mathfrak{C}, G, \mathcal{D})$-witness α such that $S = S_\rho(G, \mathcal{D}, \alpha)$.

The following theorem is the main result of this section. It shows how to transform dynamic programming cores for problems $\mathcal{P}_1, \ldots, \mathcal{P}_r$ into a dynamic programming core for the problem $\mathrm{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$.

Theorem 4.5. *Let $\mathcal{P}_1, \ldots, \mathcal{P}_r$ be vertex-problems, let $(\mathfrak{C}_i, \rho_i)$ be a dynamic programming core for $\mathcal{P}_i$, and let d be an integer. The problem $\mathrm{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$, on graph G with rooted tree decomposition $\mathcal{D} = (T, q, \mathcal{X})$, can be solved in time*

$$\mathcal{O}(d^a \cdot |V(T)| \cdot \prod_{i=1}^{r} \mathrm{Size}(\mathfrak{C}_i, G, \mathcal{D}) + \sum_{i=1}^{r} \tau(\mathfrak{C}_i, G, \mathcal{D})),$$

where $a = \max_{t \in V(T)} \delta(t) \leq 2$.

Let $w_1, \ldots, w_r \in \{0,1\}^*$ and $v \in V(G)$. We extend the definition of diverse influence to $w_1, \ldots, w_r$ such that

$$I(w_1, \ldots, w_r, v) = I(\hat{\rho}_1(w_1), \ldots, \hat{\rho}_r(w_r), v).$$

Before proving Theorem 4.5, we state and prove the following technical lemma.

Lemma 4.6. *Let G be a graph and $\mathcal{D} = (T, q, \mathcal{X})$ be a rooted tree decomposition of G. $(G, Z_1, \ldots, Z_r)$ belongs to $\mathrm{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$ if and only if there exist $\alpha_1, \ldots, \alpha_r : V(T) \to \{0,1\}^*$ such that the following conditions are satisfied.*

1. *For each $i \in [1, r]$, α_i is a $(\mathfrak{C}_i, G, \mathcal{D})$-witness and $Z_i = S_{\rho_i}(G, \mathcal{D}, \alpha_i)$.*
2. *$\sum_{t \in V(\mathcal{D})} \sum_{v \in \mathrm{forg}(t)} I(\alpha_1(t), \ldots, \alpha_r(t), v) \geq d$.*

Proof. First assume that $(G, Z_1, \ldots, Z_r)$ belongs to $\mathrm{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$. By Definition 1.1, for each $i \in [1, r]$, we have that $(G, Z_i) \in \mathcal{P}_i$, and so, there exists a $(\mathfrak{C}_i, G, \mathcal{D})$-witness α_i such that $Z_i = S_{\rho_i}(G, \mathcal{D}, \alpha_i)$. Thus Condition 1 is satisfied. Moreover, we have that for each $t \in V(\mathcal{D}) \setminus \{q\}$ and each $v \in X_t$, $I(\alpha_1(t), \ldots, \alpha_r(t), v) = I(Z_1, \ldots, Z_r, v)$. Together with the fact that each vertex is in exactly one set $\mathrm{forg}(t)$, $t \in V(\mathcal{D}) \setminus \{q\}$, and $\mathrm{Div}(Z_1, \ldots, Z_r) \geq d$ imply Condition 2.

Assume now that there exist $\alpha_1, \ldots, \alpha_r : V(T) \to \{0,1\}^*$ that satisfy Conditions 1 and 2. Condition 1 implies that for each $i \in [1, r]$, $(G, Z_i) \in \mathcal{P}_i$. Moreover, as for each $v \in V(G)$, there is exactly one node $t \in V(\mathcal{D}) \setminus \{q\}$ such that $v \in \mathrm{forg}(t)$, by definition of a rooted tree decomposition, Condition 2 implies that $\mathrm{Div}(Z_1, \ldots, Z_r) \geq d$. Thus, $(G, Z_1, \ldots, Z_r)$ belongs to $\mathrm{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$.□

Now we are in a position to prove Theorem 4.5. An intuitive account of what is going on in the proof of this theorem can be found in Figure 2.

Proof (Proof of Theorem 4.5). For each $i \in \{1, \ldots, r\}$, we start by constructing the data corresponding to the dynamic core $\mathfrak{C}_i$. The overall construction takes time $\sum_{i=1}^{r} \tau(\mathfrak{C}_i, G, \mathcal{D})$.

Subsequently, we define a dynamic core $\mathfrak{C}$ for the problem $\mathrm{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$. Let $G \in \mathcal{G}$ and $\mathcal{D} = (T, q, \mathcal{X})$ be a rooted tree decomposition of G. The dynamic core $\mathfrak{C}$ produces the following data.

- $\mathrm{Accept}_{\mathfrak{C},G,\mathcal{D}} = \{(w_1, \ldots, w_r, d) \mid \forall i \in [1, r], w_i \in \mathrm{Accept}_{\mathfrak{C}_i,G,\mathcal{D}}\}$.
- For each $t \in V(\mathcal{D})$, $\mathrm{Process}_{\mathfrak{C},G,\mathcal{D}}(t) =$

$$\begin{aligned}\{((w_1, \ldots, w_r, \ell), (w_1^1, \ldots, w_r^1, \ell^1), \ldots, (w_1^{\delta(t)}, \ldots, w_r^{\delta(t)}, \ell^{\delta(t)})) \mid \\ \forall i \in [1, r], (w_i, w_i^1, \ldots, w_i^{\delta(t)}) \in \mathrm{Process}_{\mathfrak{C}_i,G,\mathcal{D}}(t), \\ s = \textstyle\sum_{i \in [1,\delta(t)]} \ell^i + \sum_{v \in \mathrm{forg}(t)} I(w_1, \ldots, w_r, v), \ell = \min\{s, d\}\}.\end{aligned}$$

Let α be a $(\mathfrak{C}, G, \mathcal{D})$-witness of $(G, \mathcal{D})$, let α_i be the projection of α to its i-th coordinate, and let β be the projection of α to its last coordinate. Then we have that α is a $(\mathfrak{C}, G, \mathcal{D})$-witness for $(G, \mathcal{D})$ if and only if α_i is a $(\mathfrak{C}_i, G, \mathcal{D})$-witness for $(G, \mathcal{D})$, and for q being the root of $\mathcal{D}$,

$$\beta(q) = \min\{d, \sum_{t \in V(\mathcal{D})} \sum_{v \in \mathrm{forg}(t)} I(\alpha_1(t), \ldots, \alpha_r(t), v)\} \geq d.$$

By Lemma 4.6, we have that this happens if and only if

$$(G, S_{\rho_1}(G, \mathcal{D}, \alpha_1), \ldots, S_{\rho_r}(G, \mathcal{D}, \alpha_r))$$

belongs to $\mathrm{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$.

Let now analyze the running time of this procedure. When constructing $\mathrm{Process}_{\mathfrak{C},G,\mathcal{D}}(t)$ for some $t \in V(T)$, we need to combine every combination of elements of $\mathrm{Process}_{\mathfrak{C}_i,G,\mathcal{D}}(t)$, $i \in [1, r]$ and of values of ℓ^i, $i \in [1, \delta(t)]$. This can be done in time $\mathcal{O}(d^{\delta(t)} \cdot |V(T)| \cdot \prod_{i=1}^{r} \mathrm{Size}(\mathfrak{C}_i, G, \mathcal{D}))$. Thus constructing the data associated to $\mathfrak{C}$, G, and $\mathcal{D}$ takes

$$\mathcal{O}(d^{\delta(t)} \cdot |V(T)| \cdot \prod_{i=1}^{r} \mathrm{Size}(\mathfrak{C}_i, G, \mathcal{D}) + \sum_{i=1}^{r} \tau(\mathfrak{C}_i, G, \mathcal{D})).$$

Moreover, as for every $t \in V(T)$, $|\mathrm{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| \leq d^{\delta(t)} \cdot \prod_{i=1}^{r} \mathrm{Size}(\mathfrak{C}_i, G, \mathcal{D})$, then by Theorem 4.4, $\mathrm{Div}^d(\mathcal{P}_1, \ldots, \mathcal{P}_r)$ can be solved in time $\mathcal{O}(d^a \cdot |V(T)| \cdot \prod_{i=1}^{r} \mathrm{Size}(\mathfrak{C}_i, G, \mathcal{D}))$ where $a = \max_{t \in V(T)} \delta(t) \leq 2$. The theorem follows. □

4.2 An Illustrative Application of Theorem 4.5

In this subsection we show how to apply Theorem 4.5 in the construction of an improved dynamic programming algorithm for Diverse Vertex Cover. The first thing to do is to describe a dynamic programming core $\mathfrak{C}_{\mathrm{VC}}$ for k-Vertex Cover. Given a graph G and a rooted tree decomposition $\mathcal{D} = (T, q, \mathcal{X})$, this dynamic programming core $\mathfrak{C}_{\mathrm{VC}}$ produces (see Figure 1 for an illustration):

$$\begin{aligned}\mathrm{Accept}_{\mathfrak{C},G,\mathcal{D}} &= \{(S, s) \mid S \subseteq X_q, s \leq k\} \\ \mathrm{Process}_{\mathfrak{C},G,\mathcal{D}}(t) &= \{((S, s), (S^1, s^1), \ldots, (S^{\delta(t)}, s^{\delta(t)})) \mid \\ &\quad E(G[X_t \setminus S]) = \emptyset, \\ &\quad \forall i \in [1, \delta(t)] : S^i \cap X_t = S \cap X_{t_i}, \\ &\quad s = |\mathrm{forg}(t) \cap S| + \textstyle\sum_{t=1}^{\delta(t)} s^i\}\end{aligned}$$

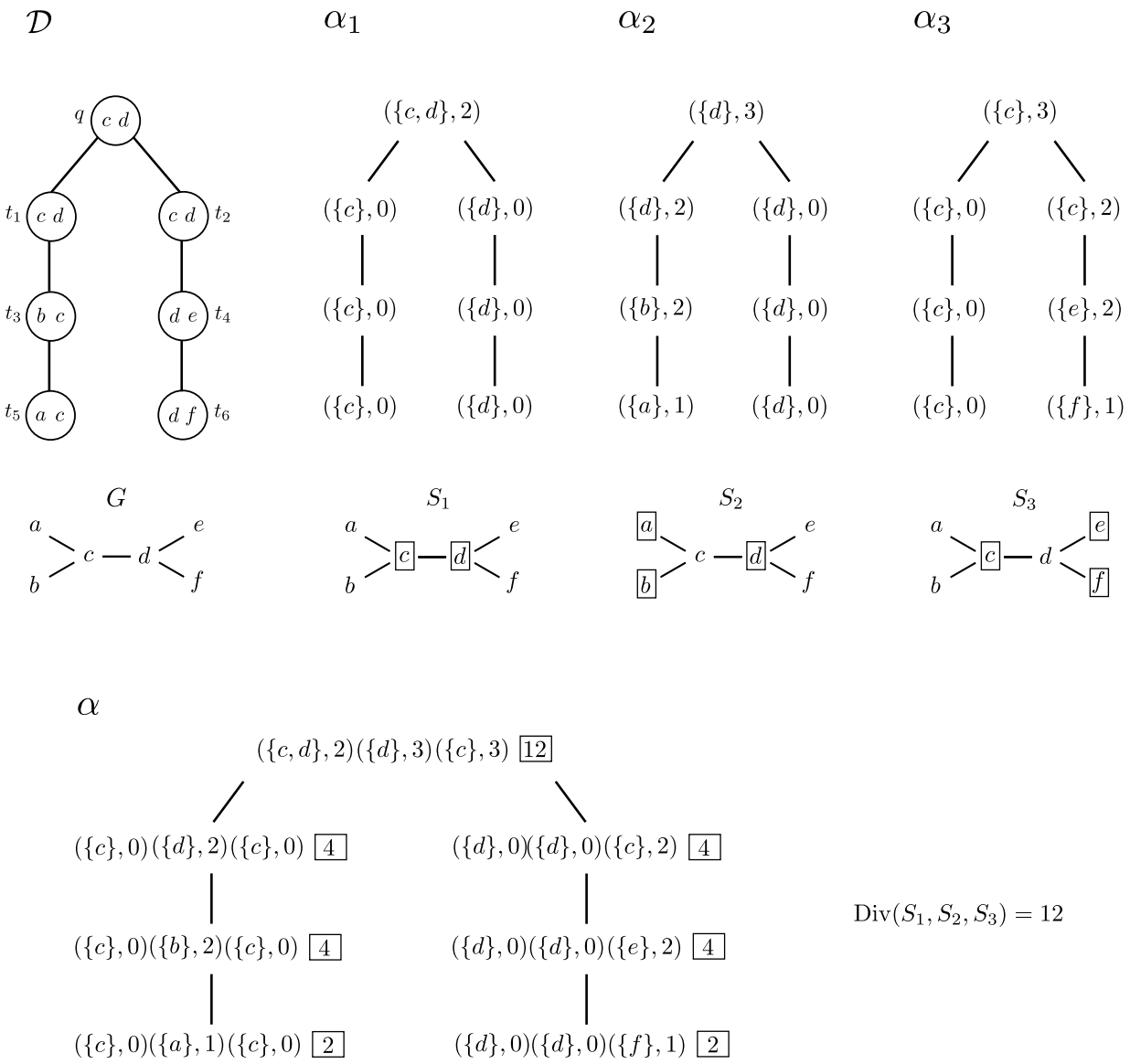

Figure 2: An illustration of the construction in the proof of Theorem 4.5. The graph G and the tree decomposition $\mathcal{D}$ are the same as in Figure 1. The sets $S_1 = \{c,d\}$, $S_2 = \{a,b,d\}$ and $S_3 = \{c,e,f\}$ are vertex covers of G with diversity $\mathrm{Div}(S_1,S_2,S_3) = 12$. Let $\mathfrak{C}_1 = \mathfrak{C}_2 = \mathfrak{C}_3$ be three instantiations of the vertex-cover core. Then, for each $i \in \{1,2,3\}$, α_i is a $(\mathfrak{C}_i, G, D)$-witness representing the vertex cover S_i. Now, let $\mathfrak{C}$ be the combined core defined in the proof of Theorem 4.5 and let α be a $(\mathfrak{C}, G, D)$-witness. Then, for each node t, $\alpha(t) = (\alpha_1(t), \alpha_2(t), \alpha_3(t), \ell)$ where ℓ is a counter that stores the diversity of the partial solutions represented by α_1, α_2 and α_3 up to node t. The value of this counter is obtained by summing the values at the counters of the children of t plus the influence of each vertex forgotten at t. For instance, the value at the counter of the root node q is 12 because the values of the counters of the children of q sum up to 8 and the vertices c and d that are forgotten at q have each influence 2.

Provided the width of the decomposition is at most k, this can be done in time $\mathcal{O}((2^{k+1} \cdot (k+1))^{\delta(t)} \cdot k \cdot \delta(t))$ for each $t \in V(T)$, where the factor $k \cdot \delta(t)$ appears as we need the conditions $E(G[X_t \setminus S]) = \emptyset$ and $\forall i \in [1, \delta(t)], S^i \cap X_t = S \cap X_{t_i}$ to be verified. It is easy to verify that $\mathfrak{C}_{\text{VC}}$ is a dynamic programming core for the Vertex Cover problem. As described in Remark 3.3, we know that we can construct a rooted path decomposition of G of width k. We are now considering this rooted path decomposition. Thus, for each $t \in V(T)$, $|\text{Process}_{\mathfrak{C},G,\mathcal{D}}(t)| \leq 2 \cdot 2^{k+1} \cdot (k+1)$. By Theorem 4.5, we obtain the following corollary, improving Corollary 3.4.

Corollary 4.7. Diverse Vertex Cover *can be solved on an input* (G, k, r, d) *in time*

$$\mathcal{O}(d \cdot |V(G)| \cdot (2^{k+2} \cdot (k+1))^r + |V(G)| \cdot 2^{k+1} \cdot (k+1) \cdot k).$$

Note that we obtain a slightly better running time than for Corollary 3.4. This is due to the fact that verifying the properties $E(G[X_t \setminus S]) = \emptyset$ and $\forall i \in [1, \delta(t)], S^i \cap X_t = S \cap X_{t_i}$ is done when constructing $\mathfrak{C}_{\text{VC}}$ and not when constructing $\mathfrak{C}$. Note also that, formally, we need to construct $\mathfrak{C}_{\text{VC}}$ r times but as it is r times the same, we do the operation only once.

5 Diversity in Kernelization

Another key concept in the field of parameterized complexity is that of a *kernelization* algorithm [17]. We have obtained some parallel results about the kernelization complexity of diverse problems as well that we want to briefly sketch in this section. A polynomial kernel of a parameterized problem is a polynomial-time algorithm that given any instance either solves it or constructs in polynomial time an equivalent[1] instance whose size is polynomial in the parameter. It is known that a parameterized problem is FPT if and only if it has a (not necessarily polynomial) kernel, and a natural step after proving a parameterized problem to be FPT is to decide whether or not it has a polynomial kernel.

We show that the diverse variants of several basic problems parameterized by the number of requested solutions plus solution size admit polynomial kernels as well. This is done via a variant of the recently introduced notion of *loss-less kernels* [9] which are a special class of kernelizations that - very roughly speaking - for each but polynomially (in the parameter) many bits of the input can either decide whether it has to be part of every solution or if it may be added to a solution without 'destroying' it.

For instance, consider the famous Buss kernel for Vertex Cover [8]: Given a graph G and an integer k, we want to decide if G has a vertex cover of size k. Each vertex of degree at least $k+1$ must be in each solution. Otherwise, we have to include its (at least) $k+1$ neighbors, exceeding the size constraint. On the other hand, each isolated (degree-0) vertex can be included in a vertex cover without destroying it, but it does not cover any edge. In the 'non-diverse' variant, we may remove these isolated vertices, and in the diverse variant, we have to keep some of them as they may be used to increase the diversity. However, polynomially (in k and r) many such vertices suffice.

We now turn to the technical description of this framework. All problems that fall into our framework have to be *subset minimization problems*. In a *subset minimization problem*, one part of the input is a set, called the *domain* of the instance, and the objective is to find a minimum size subset of the domain that satisfies a certain property. For a subset minimization problem Π, and an instance I of Π, we denote by $\mathcal{D}(I)$ the *domain* of I. E.g., in the Vertex Cover problem, an

[1] Meaning that the constructed instance is a Yes-instance if and only if the original instance was.

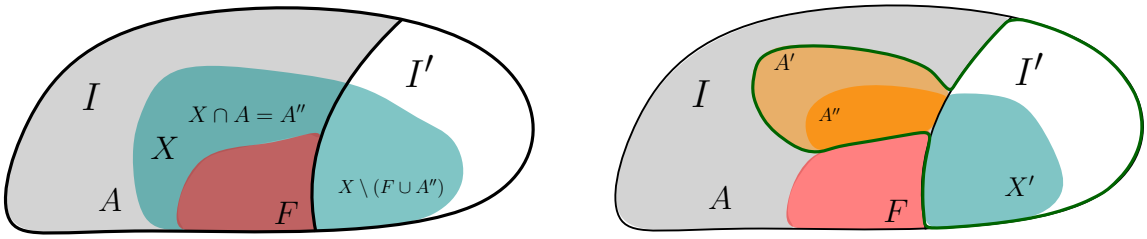

Figure 3: Illustration of Definition 5.2. Observe that (F, A) is a partition of $\mathcal{D}(I) \setminus \mathcal{D}(I')$. The left side shows part ii: Each solution X to I of size $k' \leq k$ contains F, and $X \setminus (F \cup A'')$ is a solution to I' of size $k' - |F \cup A''|$. The right side shows part iii: Given a solution X' to I' of size $k' \leq k - |F|$, $X' \cup A''$ is a solution of size $k' + |A''|$ to the instance obtained from reintroducing A' to I' via the domain recovery algorithm (which is denoted by $\mathsf{rec}_I(I', A')$.)

instance consists of a graph G and an integer k and the domain of the instance is $V(G)$. For an instance (I, k) of a parameterized problem, we denote its domain by $\mathcal{D}(I)$.

The following definition is a technical requirement to adapt loss-less kernelization to the setting of diverse problems. Domain recovery algorithms will be used to reintroduce some elements of the domain that have been removed during the kernelization process, in a controlled manner.

Definition 5.1. Let Π be a subset minimization problem. A *domain recovery algorithm* takes as input two instances of Π, I and I', with $\mathcal{D}(I') \subseteq \mathcal{D}(I)$, and a set $S \subseteq \mathcal{D}(I) \setminus \mathcal{D}(I')$ and outputs in polynomial time an instance $\mathsf{rec}_I(I', S)$ on domain $\mathcal{D}(I') \cup S$, such that $|\mathsf{rec}_I(I', S)| \leq |I'| + g(|S|)$ for some computable function g.

We give the definition of a loss-less kernel [9], tailored to our purposes as follows.[2] We use the following notation: For an instance I of a subset minimization problem and an integer k, we denote by $\mathrm{sol}(I, k)$ the solutions of I of size at most k. We illustrate the following definition in Figure 3.

Definition 5.2. Let Π be a parameterized subset minimization problem. A *loss-less kernelization* of Π is a pair of a domain recovery algorithm and an algorithm that takes as input an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ and either correctly concludes that (I, k) is a No-instance, or outputs a tuple (I', F, A) with the following properties. $(I', k - |F|)$ is an equivalent[3] instance to (I, k) with $\mathcal{D}(I') \subseteq \mathcal{D}(I)$ and (F, A) is a partition of $\mathcal{D}(I) \setminus \mathcal{D}(I')$, and the following hold.

(i) There is a computable function f such that $|I'| \leq f(k)$.

(ii) For all $k' \leq k$, for all $X \subseteq \mathcal{D}(I)$, the following holds. Let $A'' := X \cap A$. Then,

$$X \in \mathrm{sol}(I, k') \Leftrightarrow F \subseteq X \text{ and}$$
$$X \setminus (F \cup A'') \in \mathrm{sol}(I', k' - |F \cup A''|).$$

(iii) For all $k' \leq k - |F|$, for all $X' \subseteq \mathcal{D}(I')$, and for all $A'' \subseteq A' \subseteq A$ we have that:

$$X' \in \mathrm{sol}(I', k') \Leftrightarrow X' \cup A'' \in \mathrm{sol}(\mathsf{rec}_I(I', A'), k' + |A''|).$$

We call $f(k)$ the *size* and $g(\cdot)$[4] the *recovery cost* of the loss-less kernel, F the *forced* items and A the *allowed* items.

[2] Due to technical reasons and at a potential cost of slightly increased kernel sizes, we do not keep track of the *restricted* items that are forbidden in any solution of size k.

[3] I.e. (I, k) is a Yes-instance if and only if $(I', k - |F|)$ is.

[4] Function g is given implicitly in the domain recovery algorithm $\mathsf{rec}_I(I', A')$.

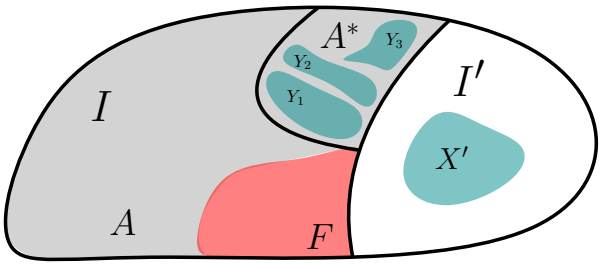

Figure 4: Illustration of how loss-less kernels are used to obtain kernels for diverse problems. While the kernelized instance I' preserves the YES/NO-answer of I, we may need to reintroduce a subset of A, here A^*, to be able to obtain a diverse set of solutions. In this example, X' is a solution for I', but only (X_1, X_2, X_3), where $X_i = X' \cup Y_i \cup F$ for all $i \in \{1, 2, 3\}$, is a diverse set of solutions for the instance I. Without reintroducing A^* to the instance, we may not be able to obtain a diverse set of solutions for I from the kernelized instance.

We now show that for all problems Π that admit a loss-less kernel, DIVERSE Π admits a (slightly larger) kernel. The idea is as follows: When kernelizing and instance of a non-diverse problem (I, k) to a smaller instance (I', k'), we preserve the YES/NO-answer, but not necessarily diverse sets of solutions. To make sure that we have sufficiently many elements in the kernelized instance to obtain a diverse solution if the original instance had one, we reintroduce a subset A^* of A (the allowed items in the loss-less kernel) to (I', k'). (Note that each element in A can be added to any solution of I without destroying it.) It suffices to use a set A^* of size at most kr, which implies that the resulting instance remains small with respect to the cost of reintroducing A^*, k and r. For an illustration see Figure 4.

Theorem 5.3. *Let Π be a parameterized subset minimization problem that admits a loss-less kernel of size $f(k)$ and recovery cost $g(\cdot)$. Then,* DIVERSE Π *admits a kernel of size at most $f(k) + g(kr)$.*

Proof. Let (I, k, r, d) be an instance of DIVERSE Π. Our algorithm works as follows. We apply the loss-less kernel to (I, k) and obtain (I', F, A). Let $k' := k - |F|$. Then, we simply return $(\mathsf{rec}_I(I', A^*), k', r, d)$ where $A^* = A$ if $|A| \leq kr$ and otherwise, A^* is an arbitrary size-kr subset of A. We now show that $(\mathsf{rec}_I(I', A^*), k', r, d)$ is indeed an instance of DIVERSE Π that is equivalent to (I, k, r, d).

Suppose (I, k, r, d) is a YES-instance. Then, there is a tuple $\mathcal{S} = (S_1, \ldots, S_r) \in \mathcal{D}(I)^r$ such that for all $i \in [1, r]$, $S_i \in \mathrm{sol}(I, k)$ and $\mathrm{Div}(\mathcal{S}) \geq d$.

Case 1 ($|A| \leq kr$). In this case, $A^* = A$. For all $i \in [1, r]$, let $A_i := S_i \cap A$, $S'_i := S_i \setminus A_i$ and $S^*_i := S'_i \setminus F$. By Definition 5.2(ii), we have that $S^*_i \in \mathrm{sol}(I', k - |F| - |A_i|)$. By Definition 5.2(iii), this implies that $S'_i \in \mathrm{sol}(\mathsf{rec}_I(I', A^*), k')$ (recall that $k' = k - |F|$). Furthermore, since $F \subseteq S_i$ for all $i \in [1, r]$ by Definition 5.2(ii), we have that $\mathrm{Div}(S'_1, \ldots, S'_r) = \mathrm{Div}(\mathcal{S}) \geq d$, and hence $(\mathsf{rec}_I(I', A^*), k', r, d)$ is a YES-instance in this case.

Case 2 ($|A| > kr$). In this case, A^* is an arbitrary size-kr subset of A. For all $i \in [1, r]$, let $A_i := S_i \cap A$, $S^*_i := S_i \setminus (F \cup A_i)$. By Definition 5.2(ii) we have that $S^*_i \in \mathrm{sol}(I', k' - |A_i|)$. Furthermore, since removing an element from some S_i can decrease the diversity of the resulting solution by at most $(r - 1)$, and since $F \subseteq S_i$ for all $i \in [1, r]$ by Definition 5.2(ii), we have that

$$\mathrm{Div}(S^*_1, \ldots, S^*_r) \geq \mathrm{Div}(\mathcal{S}) - (r - 1) \sum_{i=1}^{r} |A_i|.$$

We construct a tuple of solutions to $\mathsf{rec}_I(I', A^*)$ as follows. Let $(B_1, \ldots, B_r)$ a tuple of pairwise disjoint subsets of A^* such that for all $i \in [1, r]$, $|B_i| = |A_i|$. Such a tuple exists since $\sum_{i=1}^{r} |A_i| \leq$

$kr = |A^*|$. For $i \in [1, r]$, let $S'_i := S^*_i \cup B_i$ and $\mathcal{S}' := (S'_1, \ldots, S'_r)$. Let $i \in [1, r]$. Since $S^*_i \in \mathrm{sol}(I', k' - |A_i|)$, $|A_i| = |B_i|$ and $B_i \subseteq A^*$, we use Definition 5.2(iii) to conclude that $S'_i \in \mathrm{sol}(\mathsf{rec}_I(I', A^*), k')$.

Now, adding B_i to S^*_i increased the diversity of the resulting solution by $(r-1) \cdot |A_i|$, since no element of B_i is added to any other solution. Hence,

$$\begin{aligned}\mathrm{Div}(\mathcal{S}') &= \mathrm{Div}(S^*_1, \ldots, S^*_r) + (r-1)\sum\nolimits_{i=1}^{r} |A_i| \\ &\geq \mathrm{Div}(\mathcal{S}) \geq d.\end{aligned}$$

We have shown that $(\mathsf{rec}_I(I', A^*), k', r, d)$ is a YES-instance in this case as well.

For the other direction, suppose $(\mathsf{rec}_I(I', A^*), k', r, d)$ is a YES-instance. Then, (ii) and (iii) of Definition 5.2 immediately imply that (I, k, r, d) is a YES-instance as well.

To bound the size of $\mathsf{rec}_I(I', A^*)$, we have that $|I'| \leq f(k)$ by the definition of the (loss-less) kernel, and $|\mathsf{rec}_I(I', A^*)| \leq |I'| + g(|A^*|) \leq f(k) + g(kr)$ by the definition of a domain recovery algorithm. □

We now exemplify the use of Theorem 5.3 by showing that several well-known kernels hold in the diverse setting as well, giving polynomial kernels in the parameterization solution size plus the number of requested solutions.

We briefly introduce these problems. In the d-HITTING SET problem, we are given a hypergraph H, each of whose hyperedges contains at most d elements, and an integer k, and the goal is to find a set $S \subseteq V(H)$ of vertices of H of size at most k such that each hyperedge contains at least one element from S. In the POINT LINE COVER problem, we are given a set of points in the plane and an integer k, and we want to find a set of at most k lines such that each point lies on at least one of the lines. A directed graph D is called a *tournament*, if for each pair of vertices $u, v \in V(D)$, either the edge directed from u to v or the edge directed from v to u is contained in the set of arcs of D. In the FEEDBACK ARC SET IN TOURNAMENTS problem we are given a tournament and an integer k, and the goal is to find a set of at most k arcs such that after removing this set, the resulting directed graph does not contain any directed cycles.

Theorem 5.4. *The following diverse subset minimization problems parameterized by $k + r$ admit polynomial kernels.*

(i) DIVERSE VERTEX COVER*, on $\mathcal{O}(k(k+r))$ vertices.*

(ii) DIVERSE d-HITTING SET *for fixed d, on $\mathcal{O}(k^d + kr)$ vertices.*

(iii) DIVERSE POINT LINE COVER*, on $\mathcal{O}(k(k+r))$ points.*

(iv) DIVERSE FEEDBACK ARC SET IN TOURNAMENTS*, on $\mathcal{O}(k(k+r))$ vertices.*

Proof. (i)[5] The classical kernelization for VERTEX COVER due to [8] consists of the following two reduction rules. Let (G, k) be an instance of VERTEX COVER. First, we remove isolated vertices from G; since they do not cover any edges of the graph, we do not need them to construct a vertex cover. To obtain the loss-less kernel, we put these vertices into the set A. Second, if there is a vertex of degree more than k, this vertex has to be included in any solution; otherwise we would have to include its more than k neighbors, resulting in a vertex cover that exceeds the size bound. We add this vertex to F, remove it from G and decrease the parameter value by 1. This second reduction rule finishes the description of the kernel. It is not difficult to argue that after

[5] This was also observed in [9].

an exhaustive application of these two rules, the resulting kernelized instance (G', k') is such that either $k' < 0$, in which case we are dealing with a No-instance, or $|V(G')| = \mathcal{O}(k^2)$. For the domain recovery algorithm, we can use a trivial algorithm that reintroduces some of the vertices in A to the graph G'.

We now argue that this is indeed a loss-less kernel. Consider Definition 5.2. Item (ii) follows immediately from the fact that each vertex cover of G of size at most k has to contain all vertices in F and that each vertex in A has no neighbors in $V(G')$. The latter also implies (iii). The result now follows from Theorem 5.3.

(ii) We show that the kernel on $\mathcal{O}(k^d)$ vertices presented in [12, Section 2.6.1] is a loss-less kernel. This kernel is essentially a generalization of the one presented in the proof of (i), so we will skip some of the details. It is based on the following reduction rule: If there are $k+1$ hyperedges $e_1, \ldots, e_{k+1}$ with $Y := \bigcap_{i=1}^{k+1} e_i$ such that for each $i \in [k+1]$, $e_i \setminus Y \neq \emptyset$, then any solution has to contain Y; otherwise, to hit the hyperedges $e_1, \ldots, e_{k+1}$, we would have to include at least $k+1$ elements in the hitting set. Moreover, if $Y = \emptyset$, we can immediately conclude that we are dealing with a No-instance. If Y is nonempty, then we add all elements of Y to F and decrease the parameter value by $|Y|$. The set A consists of all vertices that are isolated (i.e. not contained in any hyperedge) after exhaustively applying the previous reduction rule. Following the same argumentation above (and using the same domain recovery algorithm), we can conclude that this procedure is a loss-less kernel on $\mathcal{O}(k^d)$ vertices, and the result follows from Theorem 5.3.

(iii) Let (P, k) be an instance of Point Line Cover. We consider the set of the lines defined by all pairs of points of P as the domain of (P, k), and we denote this set by $L(P)$. All solutions to (P, k) can be considered a subset of $L(P)$. We obtain a kernel on $\mathcal{O}(k^2)$ points as follows, cf. [12, Exercise 2.4]. The idea is again similar to the kernel presented in (i). If there are $k+1$ points on a line, then we have to include this line in any solution; we add such lines to the set F and remove all points on them from P, and decrease the parameter value by 1. We finally add to A all lines that have no points on them. We can argue in the same way as above that this gives a kernel with at most $\mathcal{O}(k^2)$ points and with Theorem 5.3, the result follows.

(iv) We observe that the kernel given in [12, Section 2.2.2] is a loss-less kernel. Its first reduction rule states that if there is an arc that is contained in at least $k+1$ triangles, then we reverse this arc and decrease the parameter value by 1, and the second reduction rule states that any vertex that is not contained in a triangle can be removed. Any arc affected by the former rule will be put in the set F and any arc affected by the latter rule will be put in the set A. We now describe the domain recovery algorithm. Let (T, k) be the original instance and (T', k') the kernelized instance, and let $(u, v) = a \in A$ be an arc. Then, we add a to T' and to ensure that the resulting directed graph is a tournament, for any $x \in \{u, v\} \setminus V(T')$, we add all arcs $(x, y) \in E(T)$ and $(y, x) \in E(T)$ to T'. Since $a \in A$, we know that one of its endpoints was not contained in any triangle, and hence adding the endpoints of a and all their incident arcs does not add any triangles to the tournament. □

We would like to remark that the crucial part to use loss-less kernels in the diverse setting was that *any* solution of size at most k has to contain all vertices of F, and arbitrarily adding vertices from A does not destroy a solution. In the 'classical' kernelization setting, to argue that a reduction rule is safe it is sufficient to show that the existence of a vertex cover in the original instance implies the existence of *some* vertex cover in the reduced instance and vice versa, see e.g., [16, 17]. This alone is usually not enough to argue that a reduction preserves diverse solutions.

6 Conclusion

In this work, we considered a formal notion of diversity of a set of solutions to combinatorial problems in the setting of parameterized algorithms. We showed how to emulate treewidth based dynamic programming algorithms in order to solve diverse problems in FPT time, with the number r of requested solutions being an additional parameter.

This line of research is now wide open, with many natural questions to address. As all our results are of a positive nature, we ask: when can diversity be a source of hardness? Concretely, a natural target in parameterized complexity would be to identify a parameterized problem Π that is FPT, however Diverse Π being W[1]-hard when r is an additional parameter. For positive results, an interesting research direction would be to generalize our framework for diverse problems to other well-studied width measures for graphs, as well as to other structures, such as matroids.

In this work, we considered the *sum* of all pairwise Hamming distances of a set as a measure of diversity. As pointed out, this measure has some weaknesses, and another widely used measure is the *minimum* Hamming distance. In this setting, we only obtain FPT-results when the diversity is bounded by a function of the treewidth and the number of solutions, but not in general. So, a natural follow-up question is whether or not we can obtain FPT-results under the minimum Hamming distance, even if the diversity is unbounded.

Acknowledgements. M. Fellows and F. Rosamond acknowledge support from the Norwegian NFR Toppforsk Project, "Parameterized Complexity and Practical Computing" (PCPC) (no. 813299). M. Fellows, F. Rosamond, L. Jaffke, M. de Oliveira Oliveira, and G. Philip acknowledge support from the Bergen Research Foundation grant "Putting Algorithms Into Practice" (BFS, no. 810564). G. Philip acknowledges support from the Research Council of Norway grants "Parameterized Complexity for Practical Computing" (NFR, no. 274526d), "MULTIVAL", and "CLASSIS", and European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC, no. 819416). T. Masařík acknowledges support from the European Research Council (ERC grant agreement no. 714704), and from Charles University's grants GAUK 1514217 and SVV-2017-260452. He recently started a postdoc at Simon Fraser University, Canada. M. de Oliveira Oliveira acknowledges support from the Research Council of Norway (NFR, no. 288761) and from the Sigma2 network (NN9535K). J. Baste acknowledges support from the German Research Foundation (DFG, no. 388217545).

References

[1] Zeinab Abbassi, Vahab S. Mirrokni, and Mayur Thakur. Diversity maximization under matroid constraints. In *19th KDD*, pages 32–40, 2013.

[2] Gediminas Adomavicius and YoungOk Kwon. Optimization-based approaches for maximizing aggregate recommendation diversity. *INFORMS Journal on Computing*, 26(2):351–369, 2014.

[3] Chris Barrett, Harry B Hunt, Madhav V Marathe, SS Ravi, Daniel J Rosenkrantz, Richard E Stearns, and Mayur Thakur. Computational aspects of analyzing social network dynamics. In *20th IJCAI*, pages 2268–2273, 2007.

[4] Julien Baste, Michael R. Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. In *29th IJCAI*, pages 1119–1125, 2020.

[5] Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12:254, 2019.

[6] Bernhard Bliem, Marius Moldovan, Michael Morak, and Stefan Woltran. The impact of treewidth on ASP grounding and solving. In *26th IJCAI*, pages 852–858, 2017.

[7] Manuel Bodirsky and Stefan Wölfl. RCC8 is polynomial on networks of bounded treewidth. In *22nd IJCAI, Vol. 2*, pages 756–761, 2011.

[8] Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993.

[9] Clément Carbonnel and Emmanuel Hebrard. Propagation via kernelization: The vertex cover constraint. In *22nd CP*, pages 147–156, 2016.

[10] Mark Chavira and Adnan Darwiche. Compiling Bayesian networks using variable elimination. In *20th IJCAI*, pages 2443–2449, 2007.

[11] Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inform. Comput.*, 85(1):12–75, 1990.

[12] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[13] Emilie Danna and David L. Woodruff. How to select a small set of diverse solutions to mixed integer programming problems. *Operations Research Letters*, 37(4):255–260, 2009.

[14] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

[15] Thomas Eiter, Esra Erdem, Halit Erdogan, and Michael Fink. Finding similar/diverse solutions in answer set programming. *Theory and Practice of Logic Programming*, 13(3):303–359, 2013.

[16] Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? volume 11011 of *LNCS*, pages 330–356. 2018.

[17] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization*. Cambridge University Press, 2019.

[18] Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th ICAC*, pages 131–140, 2018.

[19] Per Galle. Branch & sample: A simple strategy for constraint satisfaction. *BIT Numer. Math.*, 29(3):395–408, 1989.

[20] Fred Glover, Arne Løkketangen, and David L. Woodruff. Scatter search to generate diverse mip solutions. In *Computing Tools for Modeling, Optimization and Simulation*, pages 299–317. Springer, 2000.

[21] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *18th WWW*, pages 381–390, 2009.

[22] Peter Greistorfer, Arne Løkketangen, Stefan Voß, and David L. Woodruff. Experiments concerning sequential versus simultaneous maximization of objective function and distance. *Journal of Heuristics*, 14(6):613–625, 2008.

[23] Tarik Hadžić, Alan Holland, and Barry O'Sullivan. Reasoning about optimal collections of solutions. In *15th CP*, pages 409–423. Springer, 2009.

[24] Robert W Haessler and Paul E Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54(2):141–150, 1991.

[25] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In *20th AAAI*, pages 372–377, 2005.

[26] Emmanuel Hebrard, Barry O'Sullivan, and Toby Walsh. Distance constraints in constraint satisfaction. In *IJCAI*, pages 106–111, 2007.

[27] Philippe Jégou, Samba Ndojh Ndiaye, and Cyril Terrioux. Dynamic heuristics for backtrack search on tree-decomposition of CSPs. In *20th IJCAI*, pages 112–117, 2007.

[28] Ching-Chung Kuo, Fred Glover, and Krishna S. Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming*. *Decis. Sci.*, 24(6):1171–1185, 1993.

[29] Arne Løkketangen and David L. Woodruff. A distance function to support optimized selection decisions. *Decision Support Systems*, 39(3):345–354, 2005.

[30] Alexander Nadel. Generating diverse solutions in sat. In *14th SAT*, pages 287–301. Springer, 2011.

[31] Thierry Petit and Andrew C. Trapp. Finding diverse solutions of high quality to constraint optimization problems. In *24th IJCAI*, pages 260–267, 2015.

[32] Thierry Petit and Andrew C. Trapp. Enriching solutions to combinatorial problems via solution engineering. *INFORMS Journal on Computing*, 31(3):429–444, 2019.

[33] Patrick Schittekat and Kenneth Sörensen. OR practice — supporting 3PL decisions in the automotive industry by generating diverse solutions to a large-scale location-routing problem. *Operations Research*, 57(5):1058–1067, 2009.

[34] Andrew C. Trapp and Renata A. Konrad. Finding diverse optima and near-optima to binary integer programs. *IIE Trans.*, 47(11):1300–1312, 2015.

[35] Mark Wineberg and Franz Oppacher. The underlying similarity of diversity measures used in evolutionary computation. In *GECCO '03, Part II*, pages 1493–1504, 2003.

[36] Yongjie Yang and Jianxin Wang. Multiwinner voting with restricted admissible sets: complexity and strategyproofness. In *27th IJCAI*, pages 576–582, 2018.