FPT-Algorithms for the ℓ-Matchoid Problem with a Coverage Objective

[1], Chien-Chung Huang[1], and Justin Ward[2]

[1]CNRS, DI ENS, PSL, France. villars@gmail.com
[2]School of Mathematical Sciences, Queen Mary University of London, United Kingdom.
justin.ward@qmul.ac.uk

Abstract

We consider the problem of optimizing a coverage function under a ℓ-matchoid of rank k. We design fixed-parameter algorithms as well as streaming algorithms to compute an exact solution. Unlike previous work that presumes linear representativity of matroids, we consider the general oracle model.

For the special case where the coverage function is linear, we give a deterministic fixed-parameter algorithm parameterized by ℓ and k. This result, combined with the lower bounds of Lovász [37], and Jensen and Korte [27], demonstrates a separation between the ℓ-matchoid and the matroid ℓ-parity problems in the setting of fixed-parameter tractability.

For a general coverage function, we give both deterministic and randomized fixed-parameter algorithms, parameterized by ℓ and z, where z is the number of points covered in an optimal solution. The resulting algorithms can be directly translated into streaming algorithms. For unweighted coverage functions, we show that we can find an exact solution even when the function is given in the form of a value oracle (and so we do not have access to an explicit representation of the set system). Our result can be implemented in the streaming setting and stores a number of elements depending only on ℓ and z, but completely indpendent of the total size n of the ground set. This shows that it is possible to circumvent the recent space lower bound of Feldman et al. [19], by parameterizing the solution value. This result, combined with existing lower bounds, also provides a new separation between the space and time complexity of maximizing an arbitrary submodular function and a coverage function in the value oracle model.

1 Introduction

A (weighted) coverage function $f : 2^X \to \mathbb{R}_+$ is defined by a collection X of subsets of points[1] from some underlying universe, each with a weight. Given some $A \subseteq X$, $f(A)$ is simply the total weight of all points that appear in at least one set in A. Here we consider the problem of maximizing a coverage function subject to one or more matroid constraints, which are captured by the notion of an ℓ-matchoid. Formally, suppose we are given a ground set X and a coverage function $f : 2^X \to \mathbb{R}_+$. The goal is to compute a *feasible* set $S \subseteq X$ with $f(S)$ being maximized. The feasible sets of X are defined by an ℓ-matchoid $\mathcal{M}$ over X. The ℓ-matchoid $\mathcal{M}$ is a collection $\{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$ of matroids, each defined on some (possibly distinct) subset $X_i \subseteq X$, in which each element $e \in X$ appears in at most ℓ of the sets X_i. We then say that a set $S \subseteq X$ is feasible if and only if $S \cap X_i \in \mathcal{I}_i$ for each $1 \leq i \leq s$. Intuitively, an ℓ-matchoid can be regarded as the intersection of several matroid constraints, in which any element "participates" in at most ℓ of the constraints. The *rank* of an ℓ-matchoid $\mathcal{M}$, is defined as the maximum size of any feasible set. When the coverage function f is an unweighted linear function, our problem is usually called the ℓ-Matchoid problem in the literature [26].

The family of ℓ-matchoid constraints includes several other commonly studied matroid constraints. A 1-matchoid is simply a matroid, and for $\ell > 1$, letting $s = \ell$ and $X_i = X$ for all i gives an intersection of ℓ matroid constraints. Additionally, the ℓ-set packing or ℓ-uniform hypergraph matching problems can be captured by letting the elements of X correspond to the given sets or hyperedges (each of which contains at most ℓ vertices) and defining one uniform matroid of rank 1 for each vertex, allowing at most 1 hyperedge containing that vertex to be selected. It is NP-hard to maximize a coverage function even under a single, uniform matroid constraint, and the ℓ-Matchoid problem (in our setting, this is the special case in which f is an unweighted, linear function) is NP-hard when $\ell \geq 3$. Thus, various approximation algorithms have been introduced for both coverage functions and, more generally, submodular objective functions in a variety of special cases e.g., [6, 20, 21, 33, 32, 45, 54].

In this work, we study the problem from the point of view of *fixed-parameter tractability*, in which some underlying parameter of problem instances is assumed to be a fixed-constant. A variety of fixed-parameter algorithms have been obtained for matroid constrained optimization problems, under the assumption that the matroids have a linear representation. Another approach is to require only that we are able to test whether or not any given set is feasible for each matroid (i.e. *independence oracle*). This is typically the case in the *streaming* setting, in which the ground set X is not known in advance but instead arrives one element at a time. In this setting the algorithm may only store a small number of elements throughout its execution but must produce a solution for the entire instance at the end of the stream. Motivated by such settings, we consider what can be accomplished for such problems in the oracle model without access to a linear representation of the entire matroid.

1.1 Our Contributions

We give FPT-algorithms for maximizing a coverage function f under an ℓ-matchoid $\mathcal{M}$ of rank k, given only independence oracles for the matroids in $\mathcal{M}$. Here the coverage function f can be either given in the form of a value oracle, or explicitly as a family of sets over points. We accomplish our goal by constructing a *joint k-representative set* for $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$ with respect to a subset $T \subseteq X$. This is a set $R \subseteq T$ with the property that given a feasible set B of $\mathcal{M}$ with $|B| \leq k$ and any $e \in T \cap B$, there is some representative $e' \in R$ for e so that $B - e + e'$ remains feasible in $\mathcal{M}$. Our construction also works in the case in which each element $e \in X$ has some weight $w(e) \in \mathbb{R}$, in which case we guarantee that an element e's representative e' has $w(e') \geq w(e)$. Note that the set R contains representatives only for those elements in T, but we ensure that these representatives provide valid exchanges with respect to *any* sets $B_i \in \mathcal{I}_i$, which may include elements not in T. This allows us to easily employ our construction in the streaming setting, in which we can treat $T \subseteq X$ as the set of elements that is currently available to the algorithm at some time. Table 1 gives a summary of our results. We emphasize that if we use any *strict* subset of the parameters proposed, the problems become at least $W[1]$-hard—see Table 2 for a summary and Appendix A for further details.

As a warm-up, we show that a simple, combinatorial branching procedure is sufficient to produce a kernel of size $\Gamma_{\ell,k} \triangleq \sum_{q=0}^{(k-1)\ell} \ell^q$ for the general, weighted ℓ-Matchoid problem, parameterized by ℓ and the rank k

[1] Here and throughout, we use the term "points" when discussing elements of the underlying universe of a coverage function to avoid confusion with the elements of X.

Objective	Params	Kernel Size ($\ell = 1$)	Kernel Size ($\ell > 1$)	Type	Theorem
Linear	ℓ, k	k	$\mathcal{O}(\ell^{(k-1)\ell})$	D	Thm 3.8
Unweighted Coverage (Oracle)	ℓ, z	$\mathcal{O}(2^{(z-1)^2} z^{2z+1})$	$\mathcal{O}(2^{(z-1)^2} \ell^{z(z-1)\ell} z^{z+1})$	D	Thm 4.4
Weighted Coverage (Explicit)	ℓ, z	$(4e)^z \ln(\epsilon^{-1})$	$\mathcal{O}((4e)^z \ell^{(z-1)\ell} \ln(\epsilon^{-1}))$	R	Thm 5.2
Weighted Coverage (Explicit)	ℓ, z	$2^{\mathcal{O}(z)} z \log^2(m)$	$2^{\mathcal{O}(z)} \ell^{(z-1)\ell} \log^2(m)$	D	Thm 5.2

Table 1: A summary of our results. All problems are constrained by an ℓ-matchoid $\mathcal{M}$ of rank k, and for coverage problems z denotes the number of points covered in some optimal solution. The kernel size is stated as number of elements. In the last row, m refers to the size of the underlying universe of the coverage function (i.e., the number of points). In the second last column, D indicates a deterministic algorithm and R a randomized algorithm with success probability $(1 - \epsilon)$. In the offline setting, our algorithms require a number of independence oracle queries at most n times the stated bounds on the kernel size.

Objective	Params	Hardness	Source
Linear	ℓ	Para-NP-hard	[29]
Linear	k	W[1]-hard	[13]
Unweighted Coverage (Explicit)	ℓ, k	W[2]-hard	[5]
Unweighted Coverage (Explicit)	z	W[1]-hard	[13]
General Submodular (Oracle)	$\ell, k, f(OPT)$	$\max\big(\Omega(n^k), \Omega(n^{f(OPT)/2})\big)$ queries	[27, 37]

Table 2: Hardness results for subsets of the parameters we consider. Here, ℓ is the number of matroids defining our ℓ-matchoid and k is the size of the solution. For coverage functions, z is the number of points to cover. For the first four hardness results, see Appendix A for the reductions. For the last result, see discussion below.

of the ℓ-matchoid. This set can be computed *deterministically* using $\Gamma_{\ell,k} \cdot |X|$ independence oracle queries plus the time required to sort the elements of X by weight. To see this result in a larger context, we point out that for fixed-parameterized tractability, the ℓ-Matchoid problem is in a sense the most general problem one can handle using only an independence oracle, as the "equivalent" ℓ-Matroid Parity problem cannot be solved with such an oracle.

More precisely, in the ℓ-Matroid Parity problem, we are given disjoint blocks of k elements whose union must be independent in a single matroid. Although both this problem and the ℓ-Matchoid are reducible to one another [28, 38], Lovász [37] and Jensen and Korte [27] show that even when $\ell = 2$, any algorithm finding k blocks whose union is independent (implying a solution of value $2k$ in our setting) needs $\Omega(n^k)$ independence queries. Therefore, our results give a new separation between the ℓ-Matchoid and the Matroid ℓ-Parity problems in the parameterized setting. To reconcile this apparent contradiction, we note that the classical reduction between these problems takes an instance of Matroid ℓ-Parity with optimal solution k to an ℓ-Matchoid instance with optimal solution $n+k$. Thus, the given lower-bound for Matroid ℓ-Parity indeed does not apply if we parameterize by k. It is also interesting to observe how critically the linear representability of matroids affects the tractability. Marx [40] gives a randomized FPT-algorithm for Matroid ℓ-Parity, parameterized by ℓ and k, thus showing that it is possible to circumvent the lower bound of [27, 37] when the matroid is linear.

Building on this, our main result considers the parameterized Maximum $(\mathcal{M}, z)$-Coverage problem, in which now f is a general coverage function and we must select $S \subseteq X$ that is feasible for $\mathcal{M}$ and covers either z points or, in the weighted variant, z points of maximum weight. We obtain FPT-algorithms for this problem, parameterized by ℓ and z (Theorems 4.4 and 5.2). Here, it is known that parameterizing by ℓ and k causes the problem to be at least $W[2]$-hard. Coverage functions often serve as a motivating example for the study of submodular functions, and so it is tempting to ask whether one might obtain a similar FPT algorithm for an arbitrary submodular function by parameterizing by $z = f(OPT)$. However, this is impossible due to the aforementioned lower bound of [27, 37]. To see this, observe that the objective for unweighted Matroid 2-Parity is a 2-polymatroid rank function, which is submodular. The lower bound construction of [27, 37] thus can be interpreted as follows: given a submodular function $f : 2^X \to \mathbb{Z}_+$, computing a set S with $|S| \leq k$ and $f(S) \geq 2k$ requires $\Omega(n^k)$ queries. In our context, by setting $z = 2k$ implies that we need $\Omega(n^{z/2})$ queries even when $\mathcal{M}$ is a single, uniform matroid.

In order to make this distinction rigorous, we again encounter questions of representation—for a general submodular function one must typically assume that the objective f is available via a *value oracle* that, for any set S, returns the value $f(S)$. In contrast, for coverage functions, f can be given explicitly as a family of sets over the points in the universe, which may provide additional information not available in the value oracle model. In our main result, we show that it is possible to obtain a fixed-parameter algorithm for Maximum $(\mathcal{M}, z)$-Coverage even when f is given only as a value oracle, which reports only the number of points covered by a set $S \subseteq X$. This algorithm is technically the most demanding part of this paper. Here the lack of point representation requires a sophisticated data structure to store the elements properly. Moreover, we need to guarantee that the stored elements are not only compatible with the rest of the elements in the optimal solution under the ℓ-matchoid, but also cover the points that are "diffuse" enough (so that at least one of them covers the points that are not already covered by the rest of the optimal solution). The latter goal is achieved by an extensive use of the joint k-representative sets.

As a result, we demonstrate a new separation between what is possible for an arbitrary, integer-valued submodular function and an unweighted coverage function in the context of fixed-parameter tractability. A similar separation between coverage functions and arbitrary submodular functions was shown by Feige and Tennenholtz [17] and by Dughmi and Vondrák [14] in different contexts. In the former, a separation of the approximability between a general and a submodular function is shown under a single uniform matroid, but applies only in a restricted setting in which the algorithm may only query the value of sets of size *exactly* k. In the latter, a separation is established in the specific setting of truthful mechanism design. Our results imply a clean separation between what is possible for a coverage function and a general submodular function in the setting of fixed-parameter tractability, without any restriction on what types of sets the algorithm can query. Although our algorithm has rather high space and time complexity, here we emphasize that its main interest lies in the above theoretical implications. In Section 5, we give more efficient randomized and deterministic algorithms via a color-coding technique when we have explicit access to the underlying representation of a coverage function. These algorithms work even in the general, weighted case.

All of our algorithms may be implemented in the streaming setting as well, which results in new consequences in the recently introduced setting of *fixed-parameter streaming* algorithms e.g., see [10, 11, 12, 15] and the references therein. Here, the idea is to allow the *space* available for a streaming algorithm to scale as $g(p)\,\mathrm{poly}(\log n)$, where p is a parameter. Just as the original motivation of fixed-parameterized complexity is to identify the parameters that cause a problem to have large running time, here we want to identify the parameters that cause a problem to require large space.

Recently, Feldman et al. [19] showed that given an unweighted coverage function f and a uniform matroid of rank k as contraint, a streaming algorithm attaining an approximation ratio of $1/2 + \epsilon$ must use memory $\Omega(\epsilon n/k^3)$. Our Theorem 4.4 implies that one can circumvent this lower bound by parameterizing by the solution value z. As before, it is natural to ask when f is an arbitrary submodular function in the value oracle model, can one solve the problem using the same amount of space? However, Huang et al. [24], shows that to obtain the approximation ratio of $2 - \sqrt{2} + \epsilon$, one requires $\Omega(n/k^2)$ space, under a uniform matroid of rank k. This lower-bound construction uses a submodular function $f : 2^X \to \mathbb{Z}_+$ whose maximum value is $\mathcal{O}(k^2)$. It implies that it is impossible to obtain an exact solution by storing only $g(z)$ elements, where $z = \mathcal{O}(k^2)$, even for a single uniform matroid. Theorem 4.4 thus again shows a separation in the space complexity between an arbitrary submodular function and a coverage function in the parameterized streaming setting.

1.2 Related Work

Marx was the first to initiate the study of Matroid ℓ-Parity from the perspective of fixed-parameterized tractability [40], using the idea of representative families. Fomin et al. [22] gave an improved algorithm for constructing representative families that, when combined with the techniques from [40], leads to a randomized FPT algorithm for *weighted* Matroid ℓ-Parity in linear matroids. Their algorithm was subsequently derandomized by Lokshtanov et al. [35], by showing that truncation can be performed on the representation of a linear matroid deterministically.

The above results presume that the matroids in question are linearly representable. A related issue is how to compute the linear representation of such a matroid efficiently. Deterministic algorithms for finding linear representation of transversal matroids and gammoids are given by Misra et al. [43] and Lokshtanov et al. [36].

The notion of *union representation*, a generalization of linear representation, is also introduced in [36].

In another line of work, van Bevern et al. [52] considered a matroid constrained variant of facility location. Their approach can be shown to yield an FPT algorithm for weighted coverage functions subject to ℓ matroid constraints. Their algorithm requires linear representation for the underlying matroid when $\ell > 1$. Moreover, as their approach is involved and uses an offline algorithm for 2-matroid intersection, it is unclear if it can be applied in the streaming setting without further insights.

The Maximum k-Coverage problem is an extensively studied special case of our problem, when $\mathcal{M}$ is a single uniform matroid of rank k. Although this problem is known [16, 45] to be NP-hard to approximate beyond $(1 - 1/e)$, it is FPT when parameterized by the number z of points to cover [4] or by the maximum of k and the size of the largest set in X [5]. However, it is $W[2]$-hard when parameterized by k alone and $W[1]$-hard when parameterized by k and the maximum number of sets any point appears in (sometimes called the *frequency* of a point) [5], and FPT approximation schemes are known [50, 51] when the maximum frequency is bounded. Recently, Manurangsi [39] has shown that the problem cannot be approximated to better than $(1 - 1/e)$ in FPT time when parameterized by k, assuming the Gap-ETH.

In the streaming setting, approximation algorithms for Maximum k-Coverage and more generally submodular optimization under special cases of the ℓ-matchoid constraint were given in [2, 3, 7, 8, 18, 23, 25, 30, 31, 34, 41, 42, 46, 47]. Recently, McGregor et al. [41] gave streaming exact and approximate algorithms for Maximum k-Coverage, as well as for the variant in which the goal is to maximize the number of points covered by *exactly* one set (where, again, we may choose any collection of at most k sets). Their algorithm stores $O(d^{d+1}k^d)$ elements, where d is the maximum value of $f(e)$. For comparison, we parameterize by the total number of z of points to be covered and use $2^{O(z)}$ space for a general matroid constraint (Thm 5.2) in the same explicit model.

2 Preliminaries

Henceforth, we will use $A + e$ and $A - e$ to denote the sets $A \cup \{e\}$ and $A \setminus \{e\}$, respectively. For a set function $f : 2^X \to \mathbb{R}$, a set $A \subseteq X$ and an element $e \in X \setminus A$ we also use the shorthands $f(e)$ to denote $f(\{e\})$ and $f(e|A)$ to denote $f(A + e) - f(A)$.

A *matroid* $M = (X, \mathcal{I})$ over ground set X is given by a family $\mathcal{I} \subseteq 2^X$ of *independent sets* such that: (1) $\emptyset \in \mathcal{I}$, (2) $\mathcal{I}$ is downward closed: for all $A \subseteq B \subseteq X$, $B \in \mathcal{I}$ implies that $A \in \mathcal{I}$, and (3) $\mathcal{I}$ satisfies the augmentation property: if $A, B \in \mathcal{I}$ with $|A| < |B|$, there is some $e \in B \setminus A$ such that $A + e \in \mathcal{I}$. Here, we assume that matroids are given by an *independence oracle*, which, when given a query set A, answers whether or not $A \in \mathcal{I}$.

For any set $A \subseteq X$, the *rank* of A in M is given by $\text{rank}_M(A) = \max\{|B| : B \subseteq A, B \in \mathcal{I}\}$. That is, $\text{rank}_M(A)$ is the size of the largest independent set contained in A, and the rank of M is simply $\text{rank}_M(X)$, which is the common size of all maximal independent sets.

Here we will primarily work with the characterization of matroids in terms of *spans*. Formally, the span of A in M is defined as $\text{span}_M(A) = \{e \in X : \text{rank}_M(A + e) = \text{rank}_M(A)\}$. Note that for any $T \subseteq X$, we have $T \subseteq \text{span}_M(T)$, and for independent $T \in \mathcal{I}$, $\text{rank}_M(\text{span}_M(T)) = |T|$. Additionally, for $T \in \mathcal{I}$, $\text{span}_M(T) = T \cup \{e \in X \setminus T : T + e \notin \mathcal{I}\}$. Thus, it is straightforward to compute the span of an independent set T by using an independence oracle for M. The following additional facts will be useful in our analysis:

Proposition 2.1. Let $M = (X, \mathcal{I})$ be a matroid. Then,

1. For any sets $S, T \subseteq X$, if $S \subseteq \text{span}_M(T)$, then $\text{span}_M(S) \subseteq \text{span}_M(T)$.
2. For any $S, T \in \mathcal{I}$ with $S \subseteq \text{span}_M(T)$ and $|S| = |T|$, $\text{span}_M(S) = \text{span}_M(T)$.

Proof. The first claim is well-known (see e.g. [49, Theorem 39.9]). For the second, note that since $S \subseteq \text{span}_M(T)$, we must have $\text{span}_M(S) \subseteq \text{span}_M(T)$ by the first claim. Suppose for the sake of contradiction that there is some element $e \in \text{span}_M(T) \setminus \text{span}_M(S)$. Then, $S + e \in \mathcal{I}$. Moreover, $S \subseteq \text{span}_M(S) \subseteq \text{span}_M(T)$, so $S + e \subseteq \text{span}_M(T)$. However, this means that $\text{span}_M(T)$ contains an independent set $S + e$ of size $|S| + 1 = |T| + 1$, and so $\text{rank}_M(\text{span}_M(T)) \geq |T| + 1 > |T|$—a contradiction. □

Recall that we define an ℓ-matchoid on X as a collection $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$ of matroids, where each $X_i \subseteq X$ and any $e \in X$ appears in at most ℓ of the X_i. For every element $e \in X$, we let $X(e)$ denote the collection of the (at most ℓ) ground sets X_i with $e \in X_i$. We say that a set $S \subseteq X$ is *feasible* for $\mathcal{M}$ if

$S \cap X_i \in \mathcal{I}_i$ for all $1 \leq i \leq s$. The *rank* of an ℓ-matchoid $\mathcal{M}$ is the maximum size of a feasible set for $\mathcal{M}$. We suppose without loss of generality that for each element $e \in X$, $\{e\} \in \mathcal{I}_i$ for all $X_i \in X(e)$, (i.e. none of the matroids in $\mathcal{M}$ has loops), in other words, $\{e\}$ is feasible in $\mathcal{M}$. Note that any element e for which this is not the case cannot be part of any feasible solution and so can be discarded.

3 Joint k-Representative Set

Our main construction will involve the following notion of a representative set for a collection $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$ of matroids.

Definition 3.1 (Joint k-representative set for $(T, \mathcal{M}, w)$)**.** Let X be a set and suppose that each element $e \in X$ has some weight $w(e) \in \mathbb{R}$. Let $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$ be a matchoid with $X_i \subseteq X$ for all $1 \leq i \leq s$. Finally, let T be a fixed subset of X.

We say that some subset $R \subseteq T$ is a *joint k-representative set for* $(T, \mathcal{M}, w)$ if for any feasible set B of $\mathcal{M}$, with $|B| \leq k$, and for any element $b \in T \cap B$, there exists some $e \in R$ with $w(e) \geq w(b)$ and $B - b + e$ feasible for $\mathcal{M}$.

Note that such a joint k-representative set $R \subseteq T$ has the property that for any feasible solution O of size at most k in $\mathcal{M}$, and each $b \in O \cap T$: either R contains b already (in which case, we let $e = b$), or R contains some other element e so that $O - b + e$ remains feasible and the new weight $w(O - b + e) \geq w(O)$. As we show in Theorem 3.7, if R is a joint k-representative set for X, then it then follows that R must contain a feasible solution of size at most k with total weight at least as large as any feasible set $O \subseteq X$ with size at most k.

We now give an algorithm for computing a joint k-representative set for $(T, \mathcal{M}, w)$. Our main procedure, RepSet is presented in Algorithm 1. We suppose that we are given access to independence oracles for all matroids in $\mathcal{M}$, as well as a weight function $w : X \to \mathbb{R}$. In order to compute a joint k-representative set for $(T, \mathcal{M}, w)$, the procedure RepSet(T) makes use of an auxiliary procedure Guess that takes a pair (J, Y) as input. The first input J to Guess is a multi-dimensional set $J = (J_1, \cdots, J_s)$, where each $J_i \subseteq X_i$ for $1 \leq i \leq s$, and the second input is a subset $Y \subseteq T$. For a multi-dimensional set J, we define $\|J\| \triangleq \sum_{i=1}^s |J_i|$, and let $J +_i e$ denote the multi-dimensional set obtained from J by adding e to the set J_i. That is, $J +_i e = (J_1, \ldots, J_{i-1}, J_i + e, J_{i+1}, \ldots, J_s)$. Given a pair of inputs (J, Y), the procedure Guess first selects a maximum weight element e of Y and adds e to the output set R. If $\|J\| < (k-1)\ell$, it then considers each of the matroids $M_i = (X_i, \mathcal{I}_i)$ for which $X_i \in X(e)$. For each of these, it makes a recursive call in which e has been added to the corresponding set J_i of J and all elements of Y spanned by $J_i + e$ in M_i have been removed from Y.

Algorithm 1: FPT-algorithm

```
Input: parameters $k, \ell$, independence oracles for $\ell$-matchoid $\mathcal{M} = \{M_i\}_{i=1}^s$ of rank $k$, weight function
       $w : X \to \mathbb{R}$.
procedure RepSet($T$)
    return the output of Guess($(\emptyset, \ldots, \emptyset), T$);
procedure Guess($J = (J_1, \cdots, J_s), Y$)
    if $Y = \emptyset$ then return $\emptyset$;
    Let $e = \arg\max_{a \in Y} w(a)$;
    $R = \{e\}$;
    if $\|J\| < (k-1)\ell$ then
        for each $X_i \in X(e)$ do
            Define $Y_i = Y \setminus \mathrm{span}_{M_i}(J_i + e)$;
            $R = R \cup$ Guess($J +_i e, Y_i$);
    return $R$;
```

In our analysis, it will be helpful to consider the tree of recursive calls to Guess made during the execution of RepSet(T). Each node in this tree corresponds to some call Guess(J, Y), where $Y \subseteq T$ and J is a multi-dimensional set.

The following proposition is a rather straightforward consequence of our algorithm.

Proposition 3.2. For any call $\textsc{Guess}(J, Y)$ in the tree of recursive calls made by $\textsc{RepSet}(T)$,

1. $J_i \in \mathcal{I}_i$ for $1 \leq i \leq s$.
2. $e \in Y$ if and only if $e \in T$ and for every $X_i \in X(e)$, $e \notin \mathrm{span}(J_i)$.

Proof. We prove the proposition to be true by induction on the depth of the tree node corresponding to a call $\textsc{Guess}(J, Y)$. In the root, the proposition holds trivially as $J = (\emptyset, \dots, \emptyset)$. Consider now a non-root node corresponding to some call $\textsc{Guess}(J, Y)$. Such a call is invoked by the parent node corresponding to some other call $\textsc{Guess}(J', Y')$, where $J = J' +_i e$ for some $e \in Y'$ and $X_i \in X(e)$.

For all $i' \neq i$, $J_{i'} = J'_{i'} \in \mathcal{I}_{i'}$, by induction hypothesis. For J_i, we note that by the induction hypothesis, $e \notin \mathrm{span}(J'_i)$ and $J'_i \in \mathcal{I}_i$. Thus, $J_i = J'_i + e \in \mathcal{I}_i$ and part (1) of the proposition is proved. For part (2), by induction hypothesis, $e \in Y'$ if and only if $e \in T$ and for every $X_i \in X(e)$, $e \notin \mathrm{span}(J_i)$ and by the operation of the algorithm, $e \in Y' \backslash Y$ if and only if $e \in Y'$ and $e \in \mathrm{span}_{M_i}(J'_i + e)$. The proof then follows. □

Note that we can compute $Y \backslash \mathrm{span}_M(J_i + e)$ in line 9 of Algorithm 1 by using at most $|Y| \leq |T|$ independence oracle calls for M_i. Each call to $\textsc{RepSet}$ will result in several recursive calls to $\textsc{Guess}(J, Y)$. In our analysis, it will be useful to consider inputs J, Y that satisfy the following property:

Definition 3.3. Given a feasible set B in the matchoid $\mathcal{M}$ and $b \in B \cap T$, we call a pair of inputs (J, Y) *legitimate for* (B, b) if $b \in Y$ and $J_i \subseteq \mathrm{span}_{M_i}(B_i - b)$, where $B_i = B \cap X_i$, for all $1 \leq i \leq s$.

Using this definition, we now formally analyze the behavior of our algorithm.

Lemma 3.4. *Suppose that the input (J, Y) to the call* $\textsc{Guess}$ *is legitimate for (B, b). Consider the element $e = \arg\max_{a \in Y} w(a)$ selected in this call to* $\textsc{Guess}$. *If $e \in \mathrm{span}_{M_i}(B_i - b)$ for some $X_i \in X(e)$, then $|J_i| < |B_i - b|$.*

Proof. Suppose that $e \in \mathrm{span}_{M_i}(B_i - b)$ for some $X_i \in X(e)$ and assume for the sake of contradiction that $|J_i| \geq |B_i - b|$. By Proposition 3.2(1), $J_i \in \mathcal{I}_i$ and by definition $B_i - b \in \mathcal{I}_i$. Moreover, since (J, Y) is legitimate for (B, b), we have $J_i \subseteq \mathrm{span}_{M_i}(B_i - b)$, and, as J_i and $B_i - b$ are independent, $|J_i| \leq |B_i - b|$. Thus, $|J_i| = |B_i - b|$. Proposition 2.1(2) then implies that $\mathrm{span}_{M_i}(B_i - b) = \mathrm{span}_{M_i}(J_i)$. But then by Proposition 3.2(2), $e \notin \mathrm{span}_{M_i}(J_i)$ and so $e \notin \mathrm{span}_{M_i}(B_i - b)$—a contradiction. □

Lemma 3.5. *Consider a legitimate input (J, Y) for (B, b), and let $e = \arg\max_{a \in Y} w(a)$. Then, $w(e) \geq w(b)$ and either $B - b + e$ remains feasible or there is some $X_i \in X(e)$ such that $J' = J +_i e$, $Y' = Y \setminus \mathrm{span}_{M_i}(J_i + e)$ is a legitimate input for (B, b), with $\|J'\| = \|J\| + 1$.*

Proof. Since (J, Y) is legitimate for (B, b), we have $b \in Y$ and so $w(e) \geq w(b)$. If $B - b + e$ is not feasible, then $e \in \mathrm{span}_{M_i}(B_i - b)$ for some $X_i \in X(e)$. By Lemma 3.4, $|J_i| < |B_i - b|$. Since (J, Y) is legitimate for (B, b), $J_i \subseteq \mathrm{span}_{M_i}(B_i - b)$ and so in fact $J'_i = J_i + e \subseteq \mathrm{span}_{M_i}(B_i - b)$, and for all $i' \neq i$, $J'_{i'} = J_{i'} \subseteq \mathrm{span}_{M_{i'}}(B_i - b)$.

What remains to argue is that $b \in Y' = Y \setminus \mathrm{span}_{M_i}(J_i + e)$. If $b \notin B_i$, then $b \notin X_i$, implying that b cannot be part of $\mathrm{span}_{M_i}(J_i + e)$. So assume that $b \in B_i$. If $b \in \mathrm{span}_{M_i}(J_i + e)$ then, since $J_i + e \subseteq \mathrm{span}_{M_i}(B_i - b)$, Proposition 2.1(1) implies that $b \in \mathrm{span}_{M_i}(B_i - b)$, contradicting that $B_i = B \cap X_i \in \mathcal{I}_i$. □

We are now ready to prove our first main result: $\textsc{RepSet}(T)$ constructs a joint k-representative set for $(T, \mathcal{M}, w)$.

Theorem 3.6. *Consider an ℓ-matchoid $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$ and weight function $w : X \to \mathbb{R}$. Then, for any subset $T \subseteq X$,* $\textsc{RepSet}(T)$ *returns a joint k-representative set R for $(T, \mathcal{M}, w)$, with $|R| \leq \Gamma_{\ell,k} \triangleq \sum_{q=0}^{(k-1)\ell} \ell^q$ using at most $\Gamma_{\ell,k} \cdot |T|$ independence oracle queries. For $\ell = 1$, $|R| \leq k$ and for $\ell > 1$, $|R| \leq \frac{\ell}{\ell-1} \ell^{(k-1)\ell} = \mathcal{O}\big(\ell^{(k-1)\ell}\big)$.*

Proof. We begin by showing that the set R returned by $\textsc{RepSet}(T)$ is a joint k-representative set for $(T, \mathcal{M}, w)$. Let B be a feasible set in $\mathcal{M}$ and $|B| \leq k$. We need to show that for any $b \in T \cap B$, there must exist some $e \in R$ with $w(e) \geq w(b)$ and $B - b + e$ remains feasible in $\mathcal{M}$. In the following we fix an arbitrary element $b \in T \cap B$.

First, we note that any (J, Y) that is legitimate for (B, b) must have $\|J\| \leq (k-1)\ell$. To see this, let $B^{-} = B - b$. Then $|B^{-}| \leq k - 1$. Since $\mathcal{M}$ is an ℓ-matchoid, each element in B^{-} appears in at most ℓ of the sets $B^{-} \cap X_i$. Then, from Definition 3.3 together with Proposition 3.2(1), a legitimate input (J, Y) must have: $\|J\| = \sum_{i=1}^{s} |J_i| \leq \sum_{i=1}^{s} |B^{-} \cap X_i| \leq \ell |B^{-}| \leq \ell(k-1)$.

Now, we consider the set of all recursive calls made to $\textsc{Guess}(J, Y)$ by $\textsc{RepSet}(T)$. We first show that for any $0 \leq d \leq (k-1)\ell$, either an element e with the desired properties is added to R by call $\textsc{Guess}(J, Y)$ with $\|J\| < d$ or there is some call to $\textsc{Guess}(J, Y)$ with (J, Y) legitimate for b and $\|J\| = d$. We proceed by induction on d. Initially $\textsc{RepSet}(T)$ makes a call to $((\emptyset, \ldots, \emptyset), Y = T)$, which is legitimate for b, since $b \in T$ by assumption. For the induction step, suppose that no call to $\textsc{Guess}(J, Y)$ with $\|J\| < d < (k-1)\ell$ adds an element e with the desired property to R. Then, by the induction hypothesis, there is some call to $\textsc{Guess}(J, Y)$ with (J, Y) legitimate for b and $\|J\| = d$. Consider the element e selected by this call. By Lemma 3.5, either e has the desired properties or there is some $X_i \in X(e)$ such that $J' = J +_i e$, $Y' = Y \setminus \mathrm{span}_{M_i}(J_i + e)$ is legitimate for (B, b) and $\|J'\| = \|J\| + 1 = d + 1$. In the latter case, since $\|J\| < (k-1)\ell$, the procedure $\textsc{Guess}(J, Y)$ will make a recursive call $\textsc{Guess}(J', Y')$, for this legitimate input (J', Y'). This completes the proof of the induction step.

Suppose now, for the sake of contradiction, that no call to $\textsc{Guess}(J, Y)$ made by $\textsc{RepSet}(T)$ adds an element e with the desired properties to R. Then by the claim above (with $d = (k-1)\ell$), there is some call to $\textsc{Guess}(J, Y)$ with (J, Y) legitimate for (B, b) and $\|J\| = (k-1)\ell$. As this call must not have selected an element e with the desired properties, Lemma 3.5 implies that there must be some (J', Y') that is legitimate for (B, b) with $\|J'\| = (k-1)\ell + 1$, contradicting our bound on the size of any legitimate $\|J'\|$. Thus, for any arbitrary $b \in T \cap B$ there is indeed some $e \in R$ with $w(e) \geq w(b)$ and $B - b + e$ feasible for $\mathcal{M}$ and so R is then a joint k-representative set for $(T, \mathcal{M}, w)$.

Finally, we consider the complexity of the procedure $\textsc{RepSet}(T)$. Consider the tree of recursive calls to $\textsc{Guess}$ made by $\textsc{RepSet}(T)$. Each call in this tree contributes at most 1 additional element to the final output set. For all calls except the root, we also make at most $|Y| \leq |T|$ independence queries in Line 9 of Algorithm 1 immediately before making this call. It follows that the total number of independence oracle queries is at most $|T|$ times the size of the recursion tree. Now, we note that each non-leaf call $\textsc{Guess}(J, Y)$ of the tree has at most ℓ children $\textsc{Guess}(J', Y')$ and for each child, $\|J'\| \geq \|J\| + 1$. Thus the depth of the recursion tree is at most $(k-1)\ell$ and so contains at most $\Gamma_{\ell,k} = \sum_{q=0}^{(k-1)\ell} \ell^q$ calls. The stated bounds then follow. □

We now show that any joint k-representative set for $(X, \mathcal{M}, w)$ can be used as a kernel for maximizing a linear function under an ℓ-matchoid constraint $\mathcal{M}$.

Theorem 3.7. *Let $\mathcal{M} = \{M_i\}_{i=1}^{s}$ be an ℓ-matchoid of rank k. Then, the procedure* $\textsc{RepSet}(X)$ *computes a kernel R for finding a maximum weight feasible set for $\mathcal{M}$ with $|R| \leq \Gamma_{\ell,k} \triangleq \sum_{q=0}^{(k-1)\ell} \ell^q$. The procedure requires the time to make $\Gamma_{\ell,k} \cdot |X|$ independence oracle queries, plus the time required to sort the elements of X by weight.*

Proof. Suppose that we are given a feasible set $O = \{b_1, \ldots, b_{k'}\}$, where $k' \leq k$. We show by induction on $0 \leq r \leq k'$ that there is some set $S_r \subseteq R$ such that $O_r = O \setminus \{b_1, \ldots, b_r\} \cup S_r$ is feasible for $\mathcal{M}$, $|O_r| = |O|$ and $w(O_r) \geq w(O)$. If $r = 0$, then the claim holds trivially with $S_0 = \emptyset$, and $O_0 = O$.

In the general case, suppose that $r > 0$. By the induction hypothesis, there is some set of elements $S_{r-1} \subseteq R$ such that $O_{r-1} = O \setminus \{b_1, \ldots, b_{r-1}\} \cup S_{r-1}$ is feasible for $\mathcal{M}$, $|O_{r-1}| = |O|$, and $w(O_{r-1}) \geq w(O)$. By Theorem 3.6, there is then some element e_r in the output of $\textsc{RepSet}(X)$ with $O_{r-1} - b_r + e_r$ feasible and $w(e_r) \geq w(b_r)$. Let $S_r = S_{r-1} + e_r$ so that $O_r = O \setminus \{b_1, \ldots, b_r\} \cup S_r = O_{r-1} - b_r + e_r$. Then, $|O_r| = |O_{r-1}| = |O|$ and $w(O_r) \geq w(O_{r-1}) \geq w(O)$.

The bounds on the number of oracle queries follows directly from Theorem 3.6. Additionally, we note that each call to $\textsc{Guess}(J, Y)$ requires finding the maximum weight element $e \in Y$. This can be accomplished by sorting X at the beginning of the algorithm, and then storing each Y according to this sorted order. □

3.1 Joint k-Representative Sets in the Streaming Setting

We next show that joint k-representative sets in the preceding section can be implemented in the streaming setting. Here, we suppose that the elements of X are initially unknown, and at each step a new element e

Algorithm 2: Streaming FPT-algorithm

```
Input: parameters $\ell, k$, independence oracles for $\ell$-matchoid $\mathcal{M} = \{M_i\}_{i=1}^s$ of rank $k$, weight function
       $w : X \to \mathbb{R}$
procedure StreamingRepSet
    $R \leftarrow \emptyset$;
    for each $e \in X$ arriving in the stream do
        Let $R'$ be the result of running RepSet$(R + e)$;
        $R \leftarrow R'$;
    return $R$;
```

arrives in the stream, together with the indices of the ground sets $X_i \in X(e)$. Recall that we are parameterizing by ℓ, so we can assume that $\ell \leq n$. Furthermore, since each element participates in at most ℓ sets X_i of an ℓ-matchoid $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$, we can assume that $s \leq n\ell$.

Our algorithm, shown in Algorithm 2, maintains a representative set for all the elements that have previously arrived. When a new element arrives, we show that a new representative set for the entire stream can be obtained by applying the procedure RepSet to the set T containing the representative set for the elements that have previously arrived together with this new element.

Theorem 3.8. *Consider an ℓ-matchoid $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$ and weight function $w : X \to \mathbb{R}$. Then, the set R produced Algorithm 2 is a joint k-representative set for $(T, \mathcal{M}, w)$, where T is the subset of X arriving in the stream so far. $|R| \leq \Gamma_{\ell,k} \triangleq \sum_{q=0}^{(k-1)\ell} \ell^q$ and at all times during its execution, processing the arrival of an additional element requires temporarily storing this element together with an additional $\mathcal{O}(k\ell \log n)$ bits. For $\ell = 1$, $|R| \leq k$ and for $\ell > 1$, $|R| = \mathcal{O}\big(\ell^{(k-1)\ell}\big)$.*

Proof. We proceed by induction on the stream of elements, in order of arrival. Let B be a feasible set in $\mathcal{M}$. For each $0 \leq t \leq n$, let A_t be the first t elements that arrive in the stream and R_{t-1} be the current set R immediately before the t-th element arrives. We show by induction that for each $0 \leq t \leq |T|$, for any $b \in A_t \cap B$ there is some $e \in R_t$, such that $B - b + e$ is feasible and $w(e) \geq w(b)$. For $t = 0$, we have $A_t = \emptyset$ and so the claim follows trivially.

Let $t > 0$ and consider the arrival of the t-th element e_t in the stream. Then, $A_t = A_{t-1} + e_t$. Fix any element $b \in A_t \cap B$. We consider first the case that $b \in A_{t-1} \cap B$. By the induction hypothesis there is some $e \in R_{t-1}$ with $B - b + e$ feasible and $w(e) \geq w(b)$. Let $B' = B - b + e$. By Theorem 3.6, R_t is a joint k-representative set for $(R_{t-1} + e_t, \mathcal{M}, w)$. Then, since $e \in R_{t-1}$ there is some $e' \in R_t = \text{RepSet}(R_{t-1} + e_t)$ such that $w(e') \geq w(e) \geq w(b)$ and $B' - e + e' = B - b + e'$ feasible.

Next consider the case $b = e_t$. Again, since R_t is a joint k-representative set for $(R_{t-1} + e_t)$ there must exist some $e' \in R_t$ with $w(e') \geq w(e_t) = w(b)$ and $B - e_t + e' = B - b + e'$ feasible. This completes the proof of the induction step. The first claim in the theorem then follows by letting $t = |T|$, and noting that $A_{|T|} = T$ and $R_{|T|}$ is the set R at the moment all of T have arrived.

We note that by Theorem 3.6, the size of the set R computed in any step of the algorithm is always at most $\Gamma_{\ell,k}$. In order to process the arrival of an element e, the algorithm computes RepSet$(R + e)$. This makes a tree of recursive calls Guess(J, Y), where J is a multidimensional set and $Y \subseteq R + e$. As shown in the proof of Theorem 3.6, this tree has depth at most $(k-1)\ell$ and so at any time we must maintain at most $(k-1)\ell$ such inputs (J, Y) appearing on the path from the current call to the root of the tree. To store each J, we note that each recursive call made by Guess(J, Y) adds some element $\overline{e} \in Y$ to a set $J_i \in J$. Thus, we can represent J implicitly by storing $\overline{e}$, together with a currently selected index i at each such call in the tree. Storing this index requires $\log(s) \leq \log(n\ell) = O(\log(n))$ bits. Moreover, given J, we can easily determine Y, since it is precisely the set of elements $e' \in R + e$ such that $e' \notin \text{span}_{M_i}(J_i)$ for all $X_i \in X(e')$. Altogether then, to process the arrival of an element we must temporarily use at most $\mathcal{O}(k\ell \log(n))$ additional bits of storage, together with the space required to temporarily store this single element. □

4 Unweighted Coverage Functions in the Value Oracle Model

In the previous section, we have focused on the problem of maximizing a *linear* function subject to an ℓ-matchoid constraint. In this section and the next, we consider the more general MAXIMUM $(\mathcal{M}, z)$-COVERAGE problem. Here we are given an ℓ-matchoid $\mathcal{M}$, together with a universe U of size m, and each element $e \in X$ corresponds to some subset of U. The goal is then to find a set of elements S that is feasible in $\mathcal{M}$ and whose union contains at least z points of the universe U. To avoid confusion, we refer to the elements of U as *points* and reserve the term *element* for those elements of X and set variables and functions related to points in $\mathsf{sans\ serif}$. For each element $e \in X$, we denote by $\mathsf{P}(e)$ the set of points in U that corresponds to e. Similarly, for any subset $T \subseteq X$, we let $\mathsf{P}(T)$ denote the set of points $\bigcup_{e \in T} \mathsf{P}(e)$ that are covered by at least one element of T. In the streaming setting, we suppose that U and X are not known in advance, and the elements of X arrive one at a time.

In this section, we consider the case of an *unweighted* coverage function, in which the objective is simply to find a set $S \subseteq X$ of elements that is independent in the given ℓ-matchoid $\mathcal{M}$ so that $f(S) = |\mathsf{P}(S)|$ is maximized. We further suppose that the representation of each element e as a subset $\mathsf{P}(e) \subseteq \mathsf{U}$ is not directly available, but instead we are given a value oracle for f. For any $S \subseteq X$, this oracle returns only the value $f(S)$ (that is, the number of points covered by the union of all elements in S). We give a fixed-parameter streaming algorithm constructing a kernel for the problem of finding a feasible set S for an ℓ-matchoid $\mathcal{M}$ with $f(S) \geq z$, where $z, \ell \in \mathbb{Z}_+$ are the parameters. Recall that we can assume that for each $e \in X$ we have that $\{e\}$ feasible for $\mathcal{M}$.

4.1 An intuitive description of our approach

Due to the limitations of the value oracle model, we require a rather sophisticated data structure to achieve our goal. Here we give some informal discussion and intuition; a formal description will follow.

Consider any feasible set O for our ℓ-matchoid $\mathcal{M}$, with $f(O) \geq z$. Fix some $b_r \in O$. Under what conditions are we justified in throwing away b_r when it arrives in the stream? Here we are primarily concerned with the case in which b_r is critically contributing to the value $f(O)$, so that $f(O) \geq z$ but $f(O - b_r) < z$. Intuitively, even in this case we can throw away b_r if we have stored enough elements to ensure that there exists an element e with the properties that

(i) $O - b_r + e$ is a feasible set in $\mathcal{M}$;

(ii) e covers at least as many points outside of $O - b_r$ as b_r itself, i.e., $f(e|O - b_r) \geq f(b_r|O - b_r)$.

To achieve (i) we can simply utilize the joint representative sets introduced in the preceding section. However, guaranteeing (ii) is trickier. Here, we must ensure that our replacement e covers at least as many points outside of $\mathsf{P}(O - b_r)$ as b_r does and, unlike in the case of linear functions, this marginal coverage will, in general, depend on how both e and b_r interacts with $O - b_r$. One simple approach would be to ensure that we store a representative e for b_r that covers a *superset* of the points covered by b_r. However, this may require storing a prohibitively large number of elements: consider the case in which each element that arrives covers some *distinct* set of t points.

Thus, we adopt a different approach. First, let us do some wishful thinking: imagine that after processing b_r, we have z disjoint z-representative sets $R_1, \cdots, R_z$ for the set of elements T that have arrived so far, with the following three properties:

(a) Each element e in $\cup_{i=1}^{z} R_i$ has the same value $f(e) = f(b_r)$;

(b) There exists a set $\mathsf{A} \subseteq \mathsf{U}$ of points that are shared by all elements in $\cup_{i=1}^{z} R_i$ and the element b_r;

(c) No two elements in $\cup_{i=1}^{z} R_i$ share any point outside A.

Note that these properties are more relaxed than the requirement that all elements e in our representative set have $\mathsf{P}(b_r) \subseteq \mathsf{P}(e)$: here we require only that e covers some subset A of the points in $\mathsf{P}(b_r)$. However, we now further require that there are z distinct such representative sets, and that the stored elements e each cover a disjoint set of points in $\mathsf{U} \setminus \mathsf{A}$.

We now show briefly why this suffices to satisfy property (ii). Given a collection of representative sets $R_1, \ldots, R_z$ satisfying (a)–(c), we can find z distinct representatives (one from each R_i) for b_r. By the given properties, each of these elements will cover the same set of points A as b_r, together with $f(b_r) - |\mathsf{A}|$ unique points outside A. Then, since $f(O - b_r) < z$, property (b) and the pigeonhole principle imply that for at least one such representative element e, the set $\mathsf{P}(e) \setminus \mathsf{A}$ must be disjoint from $\mathsf{P}(O - b_r)$. This element then covers all the points of A that b_r covers, together with a new set of $f(b_r) - |\mathsf{A}|$ points not covered by any set in $O - b_r$. Thus (as we will formally show) it must satisfy property (ii).

The question now becomes how we can efficiently ensure that some collection of representative sets satisfying the above properties with respect to some set of points A exists for any possible b_r. To do this, we maintain a tree of such collections for each possible value of $f(b_r) \in \{1, \ldots, z-1\}$. The nodes of each such tree will correspond to some set A of commonly covered points, as above, and each node will store a collection of z representative sets satisfying our properties (a)–(c) (with respect to the set A of points) for each element that has previously arrived. Note that the algorithm only has access to a value oracle, so we do not know the precise value of the set A, only that some such common set *exists*. The root node in each tree corresponds to $\mathsf{A} = \emptyset$. Suppose that $\mathbf{n}$ is a general node in the tree and that all elements stored in $\mathbf{n}$ cover the common set A of points. Then, for each element e stored in $\mathbf{n}$, we will potentially create a child node $\mathbf{n}'$ of $\mathbf{n}$ associated with e. The elements stored in each such child will cover a common set of points A' where $\mathsf{A} \subset \mathsf{A}' \subseteq \mathsf{P}(e)$. Then, note that as we descend the tree, the set A associated with our current node grows larger, and so at depth (at most) z, we will have $|\mathsf{A}| = z$.

When a new element b_r arrives, we then consider the tree corresponding to $f(b_r)$. We will then descend through this tree until a node corresponding to some set A with desired properties (a)–(c) is found. For any tree node, we must determine whether b_r covers some same set of points A as all elements in $\cup_{i=1}^{z} R_i$ and b_r does not cover any point outside of A that is covered by elements in $\cup_{i=1}^{z} R_i$—we show that this can be accomplished even if we have only a value oracle (and so do not know the points in the underlying set A). Once a tree node is found whose representative sets $R_1, \ldots, R_z$ satisfy properties (a)–(c) for b_r, we can try to add b_r into exactly one of the z-representative sets, $R_1, \cdots, R_z$. If b_r cannot be added to any of these sets, then we must already have a set of z representatives for b_r as described above, and so b_r can be safely thrown away.

4.2 Formal description of the algorithm

We now describe our algorithm formally. See the code in Algorithm 3. After introducing the required notation, we give a concrete example of how the algorithm behaves for the tree structure shown in Figure 1. Our algorithm maintains a collection of $z - 1$ trees, each storing multiple joint z-representative sets. More precisely, for each $1 \leq j \leq z - 1$, we maintain a tree that stores only elements e with $f(e) = j$. Each node $\mathbf{n}$ of our trees will maintain a collection of z disjoint representative sets $R_1, \ldots, R_z$, with the property that each element in these sets covers some common set A of at least d points—where d is the depth of $\mathbf{n}$ in the tree—and no other points in common with any other element. Let $\textsc{AllReps}(\mathbf{n})$ denote the union $\bigcup_{j=1}^{z} R_j$ of all the joint z-representative sets stored at $\mathbf{n}$. A node $\mathbf{n}$ has multiple children associated with each element $e \in \textsc{AllReps}(\mathbf{n})$. For each such child node $\mathbf{n}'$ of $\mathbf{n}$, we let $\textsc{ParentElem}(\mathbf{n}')$ denote the element $e \in \textsc{AllReps}(\mathbf{n})$ that $\mathbf{n}'$ is associated with. Specifically, consider some $e \in \textsc{AllReps}(\mathbf{n})$. Then e, as well as all other elements of $\textsc{AllReps}(\mathbf{n})$, cover some common set of points A. For each subset Q of points in $\mathsf{P}(e) \setminus \mathsf{A}$, we will potentially create a new child node $\mathbf{n}'$ of $\mathbf{n}$, with $\textsc{ParentElem}(\mathbf{n}') = e$. This child node will store only elements that cover precisely this set Q of points in $\mathsf{P}(e) \setminus \mathsf{A}$, together with all points of A, and cover no other points in common. That is, all elements in this child $\mathbf{n}'$ thus cover precisely the set of points $\mathsf{A} \cup \mathsf{Q} \subseteq \mathsf{P}(e) = \mathsf{P}(\textsc{ParentElem}(\mathbf{n}'))$ and the sets of points $\mathsf{P}(e') \setminus (\mathsf{A} \cup \mathsf{Q})$ for all elements $e' \in \textsc{AllReps}(\mathbf{n}')$ are disjoint. For convenience, at the root node $\mathbf{n}$ of each tree, we let $\textsc{ParentElem}(\mathbf{n})$ be a single dummy element $\perp$, with $f(\perp) = 0$ (so $\perp$ covers no points in the underlying representation of f).

In the value oracle mode, we do not have access to the actual sets of points covered by any element. We show (in Lemma 4.1) that given elements $a, b, x \in X$, we can determine whether a and b cover the same subset of points in $\mathsf{P}(x)$, and whether they cover no other points in common outside of $\mathsf{P}(x)$ using only queries to the value oracle for f. When a new element e arrives in the stream, we then first check if $f(e) \geq z$. If this is the case, then we can simply return $\{e\}$ as a kernel for the problem. Otherwise, algorithm will update the tree corresponding to $f(e)$ as follows. First, we find a node $\mathbf{n}$ in the tree so that both e and each element in $\textsc{AllReps}(\mathbf{n})$ cover some common set of points in $P(\textsc{ParentElem}(\mathbf{n}))$ and have no other points in common (note that this common set of points in $\textsc{ParentElem}(\mathbf{n})$ corresponds to the set A in our previous discussion). Starting with $\mathbf{n}$ as the root node $\mathbf{n}_{f(e)}$ of the tree corresponding to $f(e)$, we test if this is the case. If not, there must be some element $r \in \textsc{AllReps}(\mathbf{n})$ so that $P(e) \setminus P(\textsc{ParentElem}(\mathbf{n}))$ and $P(r) \setminus P(\textsc{ParentElem}(\mathbf{n}))$ are not disjoint. We then descend the tree and recursively test whether our desired property holds for e at each child node $\mathbf{n}'$ associated with r. This is accomplished by the procedure $\textsc{FindNode}(e, \mathbf{n})$, which ultimately either returns a suitable node in which e should be stored or creates a new child of some node in the tree. In Lemma 4.2, we show that the node ultimately returned by this procedure indeed satisfies the required analogues of our intuitive properties (a)–(c) from Section 4.1.

Once we have an appropriate such node $\mathbf{n}$, we iteratively attempt to add e into each joint z-representative set R_j stored in this node, stopping as soon as we succeed. This is accomplished by the procedure $\textsc{ProcessElem}(e, \mathbf{n})$. In Lemma 4.3, we give several invariants that are maintained by our algorithm as a whole. In particular, all elements stored in the joint z-representative sets of any node have the desired properties (with respect to the points they cover), that all the joint z-representative sets R_j at a node are disjoint, and that once an element is stored in some set R_j it is never removed from this set later (and so also remains in the associated tree). In order to ensure this last property, we assign elements dummy weights in descending order of arrival. As we will show, this ensures that when rebuilding a representative set after the arrival of some element e, the procedure $\textsc{RepSet}$ (which chooses a maximum weight element in each call to $\textsc{Guess}$) will never exclude a previously selected element in favor of choosing e. At the end of the procedure, we return the union of all the joint z-representative sets stored at all the nodes of all the trees.

4.3 An example

In order to illustrate the operation of our algorithm and the desired properties of our data structure we now provide a small example. Figure 1 shows a tree $\mathbf{n}_3$ storing elements e with $f(e) = 3$ in the case that the parameter $z = 4$. Each node in the tree stores 4 representative sets, illustrated as a rounded box with 4 separate regions. For each child $\mathbf{n}'$ of a node $\mathbf{n}$ in a tree, we have illustrated the set Q of points in the parent element $\textsc{ParentElem}(\mathbf{n})$ with which this child is associated. Suppose that elements $e_1, \ldots, e_{14}$ have already arrived and been stored in the tree. We will explain how e_{15} is processed and stored. Because we are working in the value oracle model, the algorithm does not have direct access to the underlying set of points corresponding to each element. However, in order to concretely illustrate our main ideas, here we show this underlying representation. In Lemma 4.1, we argue that all of the tests we perform here can be effectively carried out using only the value oracle for f.

Since $f(e_{15}) = 3$, we begin at the root $\mathbf{n}_3$ of the tree shown. At the root $\mathbf{n}_3$, $\textsc{AllReps}(\mathbf{n}_3) = \{e_1, e_2, e_3\}$ and $p = \textsc{ParentElem}(\mathbf{n}_3) = \perp$ (recall that $\perp$ is a dummy element with $P(\perp) = \emptyset$). For each element of $r \in \textsc{AllReps}(\mathbf{n}_3) = \{e_1, e_2, e_3\}$, we check in turn whether $P(e_{15}) \setminus P(p)$ and $P(r) \setminus P(p)$ are disjoint. We find that $P(e_{15}) \setminus \emptyset = \{b_1, d_2, g_2\}$ and $P(e_2) \setminus \emptyset = \{b_1, b_2, b_3\}$ are not disjoint. Thus, we will attempt to store e_{15} in some child of e_2.

We consider each child node $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ associated with e_2 in turn. To select an appropriate child node, we check whether e_{15} covers the same set of points in $P(e_2)$ as all of the other elements stored in this child. This is the case for $\mathbf{a}$, since e_{15}, e_4, and e_5 all cover precisely b_1 and no other element from $P(e_2)$. Thus, we attempt to insert e into $\mathbf{a}$. At this stage, we have $\mathbf{n} = \mathbf{a}$, and $p = e_2$.

We check whether $P(e_{15}) \setminus P(p) = P(e_{15}) \setminus P(e_2)$ is disjoint from $P(r) \setminus P(p) = P(r) \setminus P(e_2)$ for each $r \in \textsc{AllReps}(\mathbf{a})$. This is not the case, since e_{15} and e_4 both cover the point $d_2 \notin P(e_2)$. Thus, we descend the tree again and consider all child nodes associated with element e_4. There is exactly one such child $\mathbf{f}$. Now, we check if e_{15} covers the same set of points in e_5 as each element $r \in \textsc{AllReps}(\mathbf{f})$. Indeed, each of these elements (and e_{15}) covers precisely the same set of points $\{b_1, d_2\} \subseteq P(e_5)$. Thus, we will attempt to insert e_{15} into $\mathbf{n} = \mathbf{f}$ with $p = \textsc{ParentElem}(\mathbf{f}) = e_5$.

Now, we find that $P(e_{15}) \setminus P(p)$ and $P(r) \setminus P(p)$ are disjoint for all $r \in \textsc{AllReps}(\mathbf{f})$. Thus, we process e_{15} at this node. Suppose we try to insert e_{15} into $R_1 = \{e_{13}, e_{14}\}$, but do not succeed. Then, we will try to insert e_{15} into $R_2 = \{\}$ and succeed, giving the tree shown.

Observe that the way in which elements are processed ensures that the elements of $\textsc{AllReps}(\mathbf{n})$ at any node $\mathbf{n}$ cover precisely the same set of points in $\textsc{ParentElem}(\mathbf{n})$ and, except for these points, are otherwise pairwise disjoint.

Algorithm 3: Streaming FPT-algorithm for unweighted maximum coverage

```
Input: Parameters $\ell$,$z$, independence oracles for $\ell$-matchoid $\mathcal{M} = \{M_i\}_{i=1}^{s}$, value oracle for
       $f : 2^X \to \mathbb{R}_+$.
procedure StreamingCoverage
    Let $\perp$ be a dummy element with $f(\perp) = 0$ that covers no points;
    for $1 \leq j \leq z - 1$ do
        Let $\mathbf{n}_j$ be a new root node with ParentElem($\mathbf{n}_j$) $= \perp$
    for each $e \in X$ arriving in the stream do
        if $f(e) \geq z$ then return $\{e\}$;
        Let $\hat{\mathbf{n}}$ = FindNode($e, \mathbf{n}_{f(e)}$);
        ProcessElem($e, \hat{\mathbf{n}}$);
    return the set of all elements stored in any node of the trees $\mathbf{n}_1, \ldots, \mathbf{n}_{z-1}$;
procedure FindNode($e, \mathbf{n}$)
    Let $p$ = ParentElem($\mathbf{n}$);
    for each $r \in$ AllReps($\mathbf{n}$) do
        if $f(e|p) \neq f(e|\{r, p\})$ then          ▷ $P(e) \setminus P(p)$ and $P(r) \setminus P(p)$ not disjoint.
            for each child node $\mathbf{n}'$ of element $r$ do
                if $f(r|r') = f(r|e) = f(r|\{r', e\})$ for all $r' \in$ AllReps($\mathbf{n}'$) then
                    ▷ $e$ covers the same points in $P(r)$ as each $r' \in$ AllReps($\mathbf{n}'$).
                    return FindNode($e, \mathbf{n}'$);
            ▷ $e$ covers a new, distinct set of points in $P(r)$.
            Create a new child node $\mathbf{n}'$, with ParentElem($\mathbf{n}'$) $= r$ and $R_j = \emptyset$ for $1 \leq j \leq z$;
            return $\mathbf{n}'$;
    return $\mathbf{n}$;          ▷ $P(e) \setminus P(p)$ and $P(r) \setminus P(p)$ were disjoint for all $r \in$ AllReps($\mathbf{n}$)
procedure ProcessElem($e, \mathbf{n}$)
    Let $R_1, \ldots, R_z$ be the sets stored in $\mathbf{n}$;
    for $1 \leq j \leq z$ do
        Let $w$ assign decreasing weights to points of $R_j + e$ in order of arrival.;
        Let $R'_j$ be the output of RepSet($R_j + e$) for $\mathcal{M}$ and $w$, with parameters $k = z$ and $\ell$;
        Replace $R_j$ by $R'_j$;
        if $e \in R'_j$ then return;          ▷ $e$ was successfully added
```

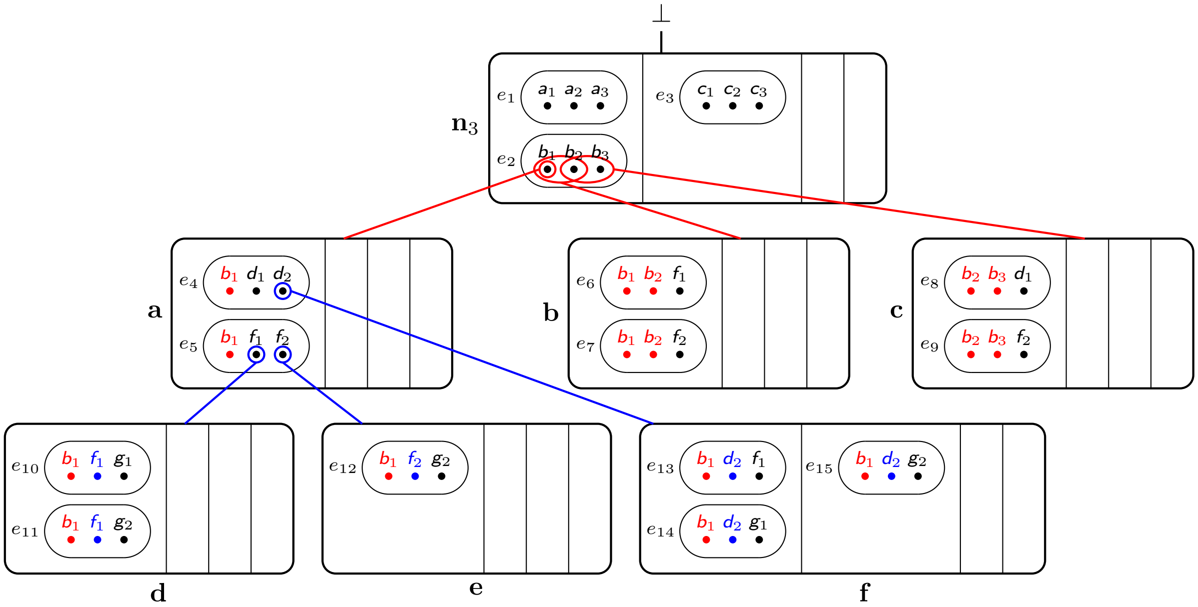

Figure 1: Let $z = 4$. Here we demonstrate a tree of depth 2 rooted at node $\mathbf{n}_3$. This tree contains elements e with $f(e) = 3$. Each node in the tree has 4 z-representative sets.

4.4 Analysis

We now give the formal statements of the lemmas discussed above, and carry out our analysis. First we argue that we can determine the required properties of the (unknown) sets of points corresponding to each element by using only a value oracle for the associated coverage function.

Lemma 4.1. *For any elements $a, b, x \in X$,*

1. *$\mathsf{P}(a) \cap \mathsf{P}(x) = \mathsf{P}(b) \cap \mathsf{P}(x)$ if and only if $f(x|a) = f(x|b) = f(x|\{a, b\})$.*
2. *$\mathsf{P}(a) \setminus \mathsf{P}(x)$ and $\mathsf{P}(b) \setminus \mathsf{P}(x)$ are disjoint if and only if $f(a|x) = f(a|\{b, x\})$.*

Proof. For part 1, note that

$$\begin{aligned} f(x|a) &= |\mathsf{P}(x)| - |\mathsf{P}(x) \cap \mathsf{P}(a)| \\ f(x|b) &= |\mathsf{P}(x)| - |\mathsf{P}(x) \cap \mathsf{P}(b)| \\ f(x|\{a, b\}) &= |\mathsf{P}(x)| - |(\mathsf{P}(x) \cap \mathsf{P}(a)) \cup (\mathsf{P}(x) \cap \mathsf{P}(b))| \,. \end{aligned}$$

If a and b share the same set of points in x, then $\mathsf{P}(x) \cap \mathsf{P}(a) = \mathsf{P}(x) \cap \mathsf{P}(b)$ and so all three of the above quantities are equal. On the other hand, if a and b do not share the same set of points in x then there must be some point in only one of $\mathsf{P}(a) \cap \mathsf{P}(x)$ and $\mathsf{P}(b) \cap \mathsf{P}(x)$. Suppose without loss of generality that this point is in $\mathsf{P}(a) \cap \mathsf{P}(x)$. Then, we have $|(\mathsf{P}(a) \cap \mathsf{P}(x)) \cup (\mathsf{P}(b) \cap \mathsf{P}(x))| > |\mathsf{P}(b) \cap \mathsf{P}(x)|$ and so the above 3 quantities are *not* equal.

For part 2, note that

$$\begin{aligned} f(a|x) &= |\mathsf{P}(a)| - |\mathsf{P}(a) \cap \mathsf{P}(b) \cap \mathsf{P}(x)| - |\mathsf{P}(a) \cap (\mathsf{P}(x) \setminus \mathsf{P}(b))| \\ f(a|\{b, x\}) &= |\mathsf{P}(a)| - |\mathsf{P}(a) \cap \mathsf{P}(b) \cap \mathsf{P}(x)| - |\mathsf{P}(a) \cap (\mathsf{P}(x) \setminus \mathsf{P}(b))| - |\mathsf{P}(a) \cap (\mathsf{P}(b) \setminus \mathsf{P}(x))| \,. \end{aligned}$$

The above two quantities are equal if and only if $|\mathsf{P}(a) \cap (\mathsf{P}(b) \setminus \mathsf{P}(x))| = 0$ and so a and b do not share any points other than those in x. □

We now show that our procedure FindNode returns a node $\mathbf{n}$ of the appropriate tree satisfying the formal analogues of the intuitive properties (a)–(c) described in Section 4.1.

Lemma 4.2. *Let $\hat{\mathbf{n}}$ be the node returned by* FindNode$(e, \mathbf{n}_{f(e)})$*. Let $d \geq 0$ be the depth of $\hat{\mathbf{n}}$ and $\hat{p} =$* ParentElem$(\hat{\mathbf{n}})$*. Then,*

1. *$|P(e) \cap P(\hat{p})| \geq d$ and $P(e) \cap P(\hat{p}) = P(r) \cap P(\hat{p})$ for all $r \in \textsc{AllReps}(\mathbf{n})$.*
2. *$P(e) \setminus P(\hat{p})$ and $P(r) \setminus P(\hat{p})$ are disjoint for all $r \in \textsc{AllReps}(\mathbf{n})$.*

Proof. Consider any recursive call made to $\textsc{FindNode}(e, \mathbf{n})$ during the execution $\textsc{FindNode}(e, \mathbf{n}_{f(e)})$ and let $p = \textsc{ParentElem}(\mathbf{n})$. We claim that in any such call $|P(e) \cap P(p)| \geq d$ and $P(e) \cap P(p) = P(r) \cap P(p)$ for all $r \in \textsc{AllReps}(\mathbf{n})$. This is true for $\mathbf{n} = \mathbf{n}_{f(e)}$, as $p = \textsc{ParentElem}(\mathbf{n}_{f(e)}) = \perp$ and $P(\perp) = \emptyset$. Suppose that the claim holds for all nodes of depth at most d, and let $\mathbf{n}$ be a node of depth $d + 1$ such that executing $\textsc{FindNode}(e, \mathbf{n}_{f(e)})$ results in a call to $\textsc{FindNode}(e, \mathbf{n}')$. This call was made in line 17 when executing some immediate predecessor call $\textsc{FindNode}(e, \mathbf{n})$ in the recursion, where the depth of $\mathbf{n}$ was d. Let $r = \textsc{ParentElem}(\mathbf{n}')$ and $p = \textsc{ParentElem}(\mathbf{n})$. Then, $r \in \textsc{AllReps}(\mathbf{n})$ and by the induction hypothesis, $|P(e) \cap P(p)| \geq d$ and $P(e) \cap P(p) = P(r) \cap P(p)$. Additionally, due to line 13 we must have $f(e|p) \neq f(e|\{r, p\})$ and so by Lemma 4.1(2), $P(e) \setminus P(p)$ and $P(r) \setminus P(p)$ share at least one point. Thus, $|P(e) \cap P(r)| \geq d + 1$, as required. Due to line 15, we must also have $f(r|r') = f(r|e) = f(r|\{r', e\})$ for all $r' \in \textsc{AllReps}(\mathbf{n}')$, and so by Lemma 4.1(1), $P(e) \cap P(r) = P(r') \cap P(r)$ for all $r' \in \textsc{AllReps}(\mathbf{n}')$, as required. This completes the proof of the induction step.

Now to prove the lemma, we note that any node $\hat{\mathbf{n}}$ returned by $\textsc{FindNode}(e, \mathbf{n}_{f(e)})$ must either be returned directly by some call $\textsc{FindNode}(e, \hat{\mathbf{n}})$ in line 21 or be a new child node of some $\mathbf{n}$, returned by $\textsc{FindNode}(e, \mathbf{n})$ in line 20. If the former case happens, then the first part of the lemma follows immediately from the claim above. Moreover, in this case, we must have $f(e|\hat{p}) = f(e|\{r, \hat{p}\})$ for all $r \in \textsc{AllReps}(\hat{\mathbf{n}})$ due to line 13 and so by Lemma 4.1(2), $P(e) \setminus P(\hat{p})$ and $P(r) \setminus P(\hat{p})$ are disjoint for all $r \in \textsc{AllReps}(\hat{\mathbf{n}})$.

On the other hand, suppose that $\hat{\mathbf{n}}$ is returned as a new child of some node $\mathbf{n}$, and let $p = \textsc{ParentElem}(\mathbf{n})$ and d be the depth of $\mathbf{n}$. Then, due to line 13, we must have $f(e|p) \neq f(e|\{r, p\})$ for $r = \textsc{ParentElem}(\hat{\mathbf{n}}) = \hat{p}$ and so, by Lemma 4.1(2), there is at least one point in both $P(e) \setminus P(p)$ and $P(r) \setminus P(p)$. By our induction claim above, $|P(e) \cap P(p)| \geq d$ and $P(e) \cap P(p) = P(r) \cap P(p)$. Thus, $|P(e) \cap P(\hat{p})| = |P(e) \cap P(r)| \geq d + 1$, which is the depth of $\hat{\mathbf{n}}$. Moreover, $\textsc{AllReps}(\hat{\mathbf{n}}) = \emptyset$, so the rest of the lemma holds trivially. □

Let $\mathbf{n}$ be the node returned by $\textsc{FindNode}(e, \mathbf{n}_{f(e)})$ when e arrives. The next lemma summarizes the property guaranteed by $\mathbf{n}$. The first two items are easy consequences of Lemma 4.2, while the last two items are the invariants guaranteed by the way in which we update representative sets in $\textsc{ProcessElem}(e, \mathbf{n})$.

Lemma 4.3. *Suppose that e is processed at a node $\mathbf{n}$ by $\textsc{ProcessElem}(e, \mathbf{n})$ during Algorithm 3, and let $p = \textsc{ParentElem}(\mathbf{n})$. Then,*

1. *$P(e) \cap P(p) = P(r) \cap P(p)$ for all $r \in \textsc{AllReps}(\mathbf{n})$ and $|P(e) \cap P(p)|$ is at least the depth of $\mathbf{n}$.*
2. *$P(e) \setminus P(p)$ is disjoint from $P(r) \setminus P(p)$ for all $r \in \textsc{AllReps}(\mathbf{n})$.*
3. *e is added to at most one set R_j stored in $\mathbf{n}$. Thus, all sets R_j stored in $\mathbf{n}$ are mutually disjoint.*
4. *If e is added to some R_j stored in $\mathbf{n}$ then e stays always as part of R_j even after any subsequent call to $\textsc{ProcessElem}(x, \mathbf{n})$ for $x \in X$.*

Proof. Parts (1) and (2) follow directly from Lemma 4.2. Part (3) follows immediately from the fact that we return from the loop in $\textsc{ProcessElem}(e, \mathbf{n})$ the first time that e is added to one of the sets $R_1, \ldots, R_z$, and so e is added to at most one set. Intuitively, part (4) follows from the fact that we assign elements dummy weights in descending order of arrival in line 25 of the algorithm. Thus, the procedure $\textsc{RepSet}$ will always prefer adding an element already in the representative set R_j over the new element x.

Formally, consider some set R_j stored at node $\mathbf{n}$ at some time during the algorithm and consider any element $e \in R_j$. Let T be the set of elements for which R_j was the output of $\textsc{RepSet}(T)$. Consider the next element x for which $\textsc{ProcessElem}(x, \mathbf{n})$ is called to update R_j, and let R'_j be the resulting set output by $\textsc{RepSet}(R_j + x)$. We will show that $e \in R'_j$ as well. Part (4) then follows by induction on the stream of elements processed at node $\mathbf{n}$.

To prove that $e \in R'_j$ as claimed, we show by induction on $\|J\|$ that if there is some call $\textsc{Guess}(J, Y)$ in $\mathcal{T}$, then there must also be a corresponding call $\textsc{Guess}(J, A)$ in $\mathcal{T}'$ with $(Y \cap R_j) \subseteq A \subseteq (Y \cap R_j) + x$. Then, since the elements of R_j are precisely those that are added to J by some call $\textsc{Guess}(J, Y)$ in $\mathcal{T}$, we must have $e \in J$ for some call $\textsc{Guess}(J, Y)$ in $\mathcal{T}$. The corresponding call $\textsc{Guess}(J, A)$ in $\mathcal{T}'$ will then also have $e \in J$ and so e will be included in R'_j.

To prove the inductive claim, first we note that $\|J\| = 0$, the roots of $\mathcal{T}$ and $\mathcal{T}'$ correspond to calls $\textsc{Guess}((\emptyset, \ldots, \emptyset), T)$ and $\textsc{Guess}((\emptyset, \ldots, \emptyset), R_j + x)$, respectively, and so the claim follows, as $R_j \subseteq T$. Suppose now that the claim holds for all $\|J\| \leq d$ and consider some call $\textsc{Guess}(J, Y)$ in $\mathcal{T}$, where $\|J\| = d + 1 > 0$.

Consider the parent $\textsc{Guess}(J', Y')$ of this call in $\mathcal{T}$ so that $Y = Y' \setminus \mathrm{span}_{M_i}(J'_i + e')$ and $J = J' +_i e'$ for some $e' = \arg\max_{a \in Y'} w(a)$ with $X_i \in X(e')$, and $\|J'\| \leq d$. By the induction hypothesis, there is then some corresponding call $\textsc{Guess}(J', A')$ in $\mathcal{T}'$, where $(Y' \cap R_j) \subseteq A' \subseteq (Y' \cap R_j) + x$. Since e' was selected by $\textsc{Guess}(J', Y')$, $e' \in Y' \cap R_j$ and so $e' = \arg\max_{a \in Y'} w(a) = \arg\max_{a \in Y' \cap R_j} w(a)$. Moreover, since $e' \in R_j$, x must arrive after e', and so $w(x) < w(e')$ and $Y' \cap R_j \subseteq A' \subseteq (Y' \cap R_j) + x$. Thus $e' = \arg\max_{a \in A'} w(a)$ and so $\textsc{Guess}(J', A')$ will also select e', resulting in a child call $\textsc{Guess}(J, A)$, with $J = J' +_i e'$ and $A = A' \setminus \mathrm{span}_{M_i}(J'_i + e')$. To complete the induction step, it remains show that $(Y \cap R_j) \subseteq A \subseteq (Y \cap R_j) + x$. For any $y \in (Y \cap R_j)$ we must have $y \in (Y' \cap R_j) \subseteq A'$ and also $y \notin \mathrm{span}_{M_i}(J'_i + e')$, since $y \in Y = Y' \setminus \mathrm{span}_{M_i}(J'_i + e')$. Then, $y \in A$ as well. Similarly, for any $a \in A - x$ we must have $a \in A' - x \subseteq (Y' \cap R_j)$ and $a \notin \mathrm{span}_{M_i}(J'_i + e')$, since $A = A' \setminus \mathrm{span}_{M_i}(J'_i + e')$. Then, $a \in (Y \cap R_j)$ as well. Thus, $(Y \cap R_j) \subseteq A \subseteq (Y \cap R_j) + x$, as required. This completes the induction step. □

Using the above properties, we now prove our main result. Note that once the set R has been computed, the following theorem implies that we can find a set S that is feasible for $\mathcal{M}$ with $f(S) \geq z$ by using at most $|R|^z$ value oracle queries, as in the proof we will show that our kernel R must in fact contain a feasible set of *at most* z elements S with $f(S) \geq z$. It follows that, when parameterized by $z = f(O)$, we can optimize an unweighted coverage function f in a general ℓ-matchoid using at most a polynomial number of value queries to f.

Theorem 4.4. *Consider an ℓ-matchoid $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^s$ and let $f : 2^X \to \mathbb{Z}_+$ be a value oracle for an unweighted coverage function. Then Algorithm 2 produces a kernel R for finding a feasible set S in $\mathcal{M}$ with $f(S) \geq z$. For $\ell = 1$, $|R| \leq N_{1,z} \triangleq \mathcal{O}\big(2^{(z-1)^2} z^{2z+1}\big)$ and the algorithm requires storing $\mathcal{O}(N_{1,z} \log n)$ bits in total and makes at most $\mathcal{O}(\Gamma_{1,z} 2^{z-1} z^2 n)$ value queries to f, where $\Gamma_{1,z} = z$. For $\ell > 1$, $|R| \leq N_{\ell,z} \triangleq \mathcal{O}\big(2^{(z-1)^2} \ell^{z(z-1)\ell} z^{z+1}\big)$ and the algorithm requires storing at most $\mathcal{O}(N_{\ell,z} \log n)$ bits in total and makes at most $\mathcal{O}(\Gamma_{l,z} 2^{z-1} z^2 n)$ value queries to f, where $\Gamma_{l,z} = \mathcal{O}(l^{(z-1)l})$.*

Proof. Let $O = \{b_1, \ldots, b_k\}$ with $f(O) \geq z$. We may suppose without loss of generality that $k \leq z$, as follows. Fix some set Z of z points covered by O. Then, as long as $|O| \geq z$, there must exist some $b_r \in O$ that can be removed from O while leaving all of Z covered. Then, $O - b_r$ is feasible for $\mathcal{M}$ with $f(O - b_r) \geq z$.

Let R be the output of Algorithm 3. If $f(b_r) \geq z$, for some b_r, then Algorithm 3 will return an element e so that $f(e) \geq z$ and the theorem holds trivially. So in the following, we suppose that $f(b_r) \leq z - 1$ for all $1 \leq r \leq k$ and so every $b_r \in O$ is processed by some call to $\textsc{ProcessElem}(b_r, \mathbf{n})$ upon arrival, where $\mathbf{n}$ is the node returned by $\textsc{FindNode}(b_r, \mathbf{n}_{f(b_r)})$.

We show by induction on $0 \leq r \leq k$ that there is some set $S_r \subseteq R$ with $O_r = O \setminus \{b_1, \ldots, b_r\} \cup S_r$ feasible for $\mathcal{M}$, $|S_r| \leq r$, and $f(O_r) \geq z$. For $r = 0$, this holds trivially by setting $S_0 = \emptyset$ and $O_0 = O$. For the general case $r > 0$, We show how to construct O_r, assuming that O_{r-1} with the desired properties exists. If $f(O_{r-1} - b_r) \geq z$, then letting $S_r = S_{r-1}$ we have $f(O_r) = f(O_{r-1} - b_r) \geq z$ and so the claim holds easily. Thus in the following we assume that $f(O_{r-1} - b_r) \leq z - 1$.

Let $\mathbf{n}$ be the node that processes element b_r (i.e. the node for which $\textsc{ProcessElem}(b_r, \mathbf{n})$ is called in Algorithm 3) and let $p = \textsc{ParentElem}(\mathbf{n})$. If b_r is added to some set stored in $\mathbf{n}$, then letting $S_r = S_{r-1} + b_r \subseteq R$ we have $|S_r| = |S_{r-1}| + 1 \leq r$ and $f(O_r) = f(O_{r-1}) \geq z$ as required. On the other hand if b_r is not added to any of the sets $R_1, \ldots, R_z$ stored in $\mathbf{n}$, then Algorithm 3 must have called $\textsc{RepSet}(R_j + b_r)$ to construct a joint z-representative set for $(R_j + b_r, \mathcal{M}, w)$ for each $1 \leq j \leq z$. By Theorem 3.6 there is then some element e_j in the output of each of these sets so that $O_{r-1} - b_r + e_j$ is feasible in $\mathcal{M}$. By Lemma 4.3(1), (2), and (3), each e_j has $\mathsf{P}(e_j) \cap \mathsf{P}(p) = \mathsf{P}(b_r) \cap \mathsf{P}(p)$ and the sets $\{\mathsf{P}(e_j) \setminus \mathsf{P}(p)\}_{j=1}^z$ are mutually disjoint. As $|\mathsf{P}(O_{r-1} - b_r)| = f(O_{r-1} - b_r) \leq z - 1$, we have at least one element $e \in \{e_1, \ldots, e_z\}$ for which $\mathsf{P}(e) \setminus \mathsf{P}(p)$ is disjoint from $\mathsf{P}(O_{r-1} - b_r)$. Let $S_r = S_{r-1} + e$. Then $|S_r| = |S_{r-1}| + 1 \leq r$ as required. Additionally, $O_r = O_{r-1} - b_r + e$ is feasible for $\mathcal{M}$. By Lemma 4.3(4), we will also have $e \in R$ for the final set of elements R produced by Algorithm 3.

It remains to show that $f(O_r) \geq z$. We note that:

$$\begin{aligned} f(e|O_{r-1} - b_r) &= |P(e)| - |P(e) \cap P(O_{r-1} - b_r)| \\ &= |P(e)| - |(P(e) \cap P(p)) \cap P(O_{r-1} - b_r)| - |(P(e) \setminus P(p)) \cap P(O_{r-1} - b_r)| \\ &= |P(e)| - |(P(e) \cap P(p)) \cap P(O_{r-1} - b_r)| \\ &= |P(b_r)| - |(P(b_r) \cap P(p)) \cap P(O_{r-1} - b_r)| \\ &\geq |P(b_r)| - |P(b_r) \cap P(O_{r-1} - b_r)| \\ &= f(b_r|O_{r-1} - b_r)\,, \end{aligned}$$

where the third equation follows from the fact that $P(e) \setminus P(p)$ is disjoint from $P(O_{r-1} - b_r)$, and the fourth from $P(e) \cap P(p) = P(b_r) \cap P(p)$, as well as $|P(b_r)| = f(b_r) = f(e) = |P(e)|$ since both e and b_r were stored in the same tree. Thus:

$$f(O_r) = f(e|O_{r-1} - b_r) + f(O_{r-1} - b_r) \geq f(b_r|O_{r-1} - b_r) + f(O_{r-1} - b_r) = f(O_{r-1}) \geq z,$$

as required. This completes the proof of the induction step. The first claim of the theorem then follows by setting $r = k$ and noting that $O_k = S_k \subseteq R$ and $|S_k| \leq k \leq z$.

We next discuss the space requirement. Consider some tree with root $\mathbf{n}_x$, where $1 \leq x \leq z - 1$. We note that by Theorem 3.7, each of the z sets $R_1, \ldots, R_z$ stored at any node of a tree has size at most $\Gamma_{\ell,z} \triangleq \sum_{q=0}^{(z-1)\ell} \ell^q$. Thus each node in the tree stores at most $\Gamma_{\ell,z} z$ elements. Whenever $\textsc{FindNode}(e, \mathbf{n})$ returns a new child $\mathbf{n}'$ with $\textsc{ParentElem}(\mathbf{n}') = r \in \textsc{AllReps}(\mathbf{n})$, we must have $P(e) \cap P(r) \neq P(e) \cap P(r')$ for all r' stored in the child nodes of $\mathbf{n}$ associated with r. By Lemma 4.3(1), all the elements r' in each such existing child node cover some common set points covered by $P(r)$. Thus, any element r can have at most $2^{|P(r)|} = 2^{f(r)} \leq 2^{z-1}$ associated child nodes. Altogether, then, a node in the tree has at most $2^{z-1}\Gamma_{\ell,z} z$ children. Additionally, Lemma 4.3(1) implies that the tree has depth at most $x \leq z - 1$. Thus, the total number of nodes in the tree is at most $N \triangleq \sum_{d=0}^{z-1} (2^{z-1}\Gamma_{\ell,z} z)^d = \mathcal{O}\big(2^{(z-1)^2} (\Gamma_{\ell,z})^{z-1} z^{z-1}\big)$, with each storing at most $\Gamma_{\ell,z} z$ elements. We maintain $z - 1$ such trees, so the total number of elements stored across all trees is at most $\mathcal{O}\big(N\Gamma_{\ell,z} z^2\big) = \mathcal{O}\big(2^{(z-1)^2} (\Gamma_{\ell,z})^z z^{z+1}\big)$. The total memory required by the algorithm is at most that required to maintain all of the stored elements in a dynamic tree, which requires $\mathcal{O}(N\Gamma_{\ell,z} z \log n)$ total bits per tree and so $\mathcal{O}(N\Gamma_{\ell,z} z^2 \log n)$ bits in total. When an element e arrives, we make several calls to $\textsc{RepSet}(R_i + e)$. As shown in the proof of Theorem 3.8, this can be accomplished by temporarily storing only $\mathcal{O}(z\ell \log n) = \mathcal{O}(N \log n)$ further bits. Altogether then, the algorithm stores at most $\mathcal{O}(N\Gamma_{\ell,z} z^2 \log n)$ bits at all times during its execution. For $\ell = 1$, $\Gamma_{\ell,z} = z$, and so $N\Gamma_{\ell,z} z^2 = \mathcal{O}\big(2^{(z-1)^2} z^{2z+1}\big)$. For $\ell > 1$, $\Gamma_{\ell,z} = \mathcal{O}(\ell^{(z-1)\ell})$ and so $N\Gamma_{\ell,z} z^2 = \mathcal{O}\big(2^{(z-1)^2} (\Gamma_{\ell,z})^z z^{z+1}\big) = \mathcal{O}\big(2^{(z-1)^2} \ell^{z(z-1)\ell} z^{z+1}\big)$.

Finally, we consider the number of value queries. When descending the tree in the procedure $\textsc{FindNode}(e, \mathbf{n})$, inside each node $\mathbf{n}$, we need to check possibly all elements stored in $\mathbf{n}$ in Line 13 using value queries, and there can be at most $\Gamma_{\ell,z} z$ of these. Furthermore, if the condition in Line 13 holds for some element r, we need to check all its child nodes. There can be 2^{z-1} such child nodes, and for each one, in Line 15, we need to check all its elements using value queries. Each such child again has at most $\Gamma_{\ell,z} z$ elements. In summary, inside each node we need $O(\Gamma_{l,z} 2^{z-1} z)$ oracle calls, implying a total of $O(\Gamma_{l,z} 2^{z-1} z^2)$ value queries for processing one new element. □

5 Improved Algorithms in the Explicit Model

We now consider the weighted version of Maximum $(\mathcal{M}, z)$-Coverage, in which we additionally have a weight function $w : U \to \mathbb{R}_+$ and now must find a feasible set S for $\mathcal{M}$ that covers (up to) z points of maximum total weight. Unlike Section 4, here a critical difference is that we assume that the sets of points corresponding to each element are given explicitly. Note that here we allow our solution to cover more than z points, but consider only the z heaviest points in computing the objective. Thus, if O is some optimal solution, by setting z to be the total number of points covered by O, then our results imply that we can find a solution S that has $f(S) \geq f(O)$. In fact, our result implies a stronger guarantee, as it ensures that the heaviest z points covered by S have alone total weight at least as large as those covered by O.

We combine multiple joint z-representative sets with a color coding procedure to obtain our results for Maximum $(\mathcal{M}, z)$-coverage. To this end, we consider a hash function $h : U \to [\bar{z}]$, where $\bar{z}$ is the smallest power of 2 that is at least z. For each point p, we call the value $h(p) \in [\bar{z}]$ the *color* assigned to p. For an element $e \in X$, we further define $h(e) = \{h(p) : p \in e\}$ to be the set of all colors that are assigned to the points covered by e.

For any set of points $T \subseteq U$, we let $w(T) = \sum_{p \in T} w(p)$ denote the total weight assigned to these points by the weight function w. We fix any solution O to the problem and let Z be a set of up to z points covered by O. Fix $h : U \to [\bar{z}]$. We say that Z is *well-colored* by h if h is injective on Z (i.e. h assigns each point $p \in Z$ a unique color in $[\bar{z}]$). Suppose now that Z is well-colored by h. For each possible subset $C \subseteq [\bar{z}]$ of colors, and each set $S \subseteq X$ of elements such that $C \subseteq h(P(S))$, we define

$$f_C(S) = \sum_{c \in C} \max\{w(p) : p \in P(S),\, h(p) = c\}$$

to be the sum of the weights of the single heaviest point of each color in C that is covered by some element of S. Let $X_C = \{e \in X : C \subseteq h(e)\}$ be the set of all elements containing at least one point assigned each color of C. Then, we can define $w_C(e) : X_C \to \mathbb{R}$ by $w_C(e) = f_C(\{e\})$. Note that to compute $w_C(e)$ it is enough to simply remember the "heaviest" point in $P(e)$ of each color. Thus, in the streaming setting we can maintain all of our constructions by using only the set of $|h(e)| \leq \bar{z} = \mathcal{O}(z)$ points and all other points can be discarded.

Let C and C' be two disjoint sets of colors and suppose $A \subseteq X$ with $C \subseteq h(P(A))$ and $b \in X_{C'}$ (so $C' \subseteq h(P(b))$). Then, $C \cup C' \subseteq h(P(A + b))$ and so

$$\begin{aligned} f_C(A) + w_{C'}(b) &= \sum_{c \in C} \max\{w(p) : p \in P(A),\, h(p) = c\} + \sum_{c \in C'} \max\{w(p) : p \in P(b),\, h(p) = c\} \\ &\leq \sum_{c \in C \cup C'} \max\{w(p) : p \in P(A + b),\, h(p) = c\} = f_{C \cup C'}(A + b)\,, \end{aligned} \tag{1}$$

since for any point $p \in P(A)$ with $h(p) = c \in C$ or any point $p \in P(b)$ with $h(p) = c \in C'$, we must also have $p \in P(A + b)$ with $h(p) = c \in C \cup C'$.

Using the above constructions, we now show how to combine multiple z-representative sets to obtain a kernel for Maximum $(\mathcal{M}, z)$-Coverage.

Lemma 5.1. *Let $\mathcal{M} = \{M_i = (X_i, \mathcal{I}_i)\}_{i=1}^{s}$ be an ℓ-matchoid. Suppose that $R = \bigcup_{C \subseteq [\bar{z}]} R_C$, where each R_C is a joint z-representative set for $(X_C, \mathcal{M}, w_C)$. Let O be any set that is feasible for $\mathcal{M}$. Let Z be a set of up to z points of maximum weight covered by O and suppose that Z is well-colored by h. Then, there is some $S \subseteq R$ that is feasible for $\mathcal{M}$ and covers $|Z|$ points of total weight at least as large as that of Z.*

Proof. Suppose that $O = \{b_1, \ldots, b_{k'}\}$ and fix some set Z of z points covered by O that is well-colored by h. For each $1 \leq r \leq k'$, let $P_r = \bigcup_{j=1}^{r} (Z \cap P(b_j))$ be the set of points from Z covered by the first r elements of O according to our indexing and let $C_r = h(P_r)$ be the set of colors assigned to these points. Note that the sets $\{P_r \setminus P_{r-1}\}_{r=1}^{k'}$ form a partition of Z. We can suppose without loss of generality that $k' \leq z$ as otherwise there must be some element $b_r \in O$ with $P_r \setminus P_{r-1} = \emptyset$. Any such element can be removed from O to obtain a feasible solution that still covers all of Z.

We now show by induction on $0 \leq r \leq k'$, that there exists a set of elements $S_r \subseteq R$ such that $O_r = O \setminus \{b_1, \ldots, b_r\} \cup S_r$ is feasible for $\mathcal{M}$, $|O_r| = k'$, $C_r \subseteq h(P(S_r))$, and $f_{C_r}(S_r) \geq w(P_r)$. In the case that $r = 0$, this follows trivially by letting $S_0 = \emptyset$ and $O_0 = O$.

In the general case $r > 0$, the induction hypothesis implies that there is a set S_{r-1} such that $O_{r-1} = O \setminus \{b_1, \ldots, b_{r-1}\} \cup S_{r-1}$ is feasible for $\mathcal{M}$, $|O_{r-1}| = k'$, $C_{r-1} \subseteq h(P(S_{r-1}))$, and $f_{C_{r-1}}(S_{r-1}) \geq w(P_{r-1})$. We will consult the representative set R_C associated with the set of colors $C = C_r \setminus C_{r-1}$. Note that $b_r \in X_C$ and so by Theorem 3.6, there must exist some element e_r in R_C such that $O_{r-1} - b_r + e_r$ is feasible, and $w_C(e_r) \geq w_C(b_r)$. Let $S_r = \{e_1, \ldots, e_r\}$ so that $O_r = O \setminus \{b_1, \ldots, b_r\} \cup S_r = O_{r-1} - b_r + e_r$. Then O_r is feasible for $\mathcal{M}$ and $|O_r| = |O_{r-1}| = k'$, as required. Since $e_r \in X_C$, e_r contains a point assigned each color in $C = C_r \setminus C_{r-1}$. Thus, $C_r \subseteq h(P(S_{r-1} + e_r)) = h(P(S_r))$, as required.

$$\begin{aligned} w(P_r) = w(P_{r-1}) + w(P_r \setminus P_{r-1}) &\leq w(P_{r-1}) + w_C(b_r) \\ &\leq f_{C_{r-1}}(S_{r-1}) + w_C(b_r) \leq f_{C_{r-1}}(S_{r-1}) + w_C(e_r) \leq f_{C_r}(S_r)\,, \end{aligned}$$

Algorithm 4: Streaming FPT-algorithm for the Maximum $(\mathcal{M}, z)$-Coverage

```
Input: Parameters $\ell, z$, independence oracles for $\ell$-matchoid $\mathcal{M} = \{M_i\}_{i=1}^{s}$, weight function
       $w : U \to \mathbb{R}$, hash function $h : U \to [\bar{z}]$.
procedure StreamingMaxCoverage
    for each $C \subseteq [\bar{z}]$ do
        Let StreamingRepSet$_C$ be an instance of the procedure StreamingRepSet for $\mathcal{M}$, with
          output $R_C$;
    for each $e \in X$ arriving in the stream do
        Color the points $P(e)$ using $h$;
        Discard all points from $e$ except for the maximum weight point of each color;
        for each $C \subseteq h(P(e))$ do
            Define $w_C(e) = \sum_{p \in P(e) : h(p) \in C} w(p)$;
            Process the arrival of $e$ in StreamingRepSet$_C$ with weight $w_C(e)$;
    return $R = \bigcup_{C \subseteq [\bar{z}]} R_C$;
```

where the first inequality follows since $P_r \setminus P_{r-1}$ is a subset of points of color C covered by b_r, the second from the induction hypothesis, the third from $w_C(e_r) \geq w_C(b_r)$, and the last from (1). This completes the induction step.

To complete the proof of the lemma, we set $r = k'$. Then, we have $O_{k'} = S_{k'} \subseteq R$ and $|S_{k'}| = |O_{k'}| = k'$. Moreover, by definition $C_{k'} = h(p_{k'}) = Z$. We have $S_{k'}$ feasible for $\mathcal{M}$, and $C_{k'} \subseteq h(P(S_k))$ so $S_{k'}$ contains a distinct point of U colored with each color $c \in C_{k'}$. For each color c, consider the heaviest such point. The total weight of these points is precisely $f_{C_{k'}}(S_{k'}) \geq w(P_{k'}) = w(Z)$. □

We now give a streaming algorithm for computing the collection of joint z-representative sets required by Lemmas 5.1. Our procedure StreamingMaxCoverage is shown in Algorithm 4. Thus far, we have supposed that some set of points Z in a solution O was well-colored by a given $h : U \to [\bar{z}]$. Under this assumption, our procedure simply runs a parallel instance of the streaming procedure StreamingRepSet for each $C \subseteq [\bar{z}]$. When a new element $e \in X$ arrives, we assign each point $p \in P(e)$ a color $h(p)$. In order to limit the memory required by our algorithm, we will remember only the maximum-weight point of each color in e. As noted in the proof of Lemmas 5.1, the resulting collection of elements R that we produce will still contain a feasible solution S with the necessary properties. It is clear that if we later consider the corresponding set of all points in $P(e)$, we can only cover *more* points of U.

We now consider the problem of ensuring that the given set Z of up to z points in $P(O)$ is well-colored by h. In the offline setting, letting h assign each point $p \in U$ a color uniformly at random guarantees that this will happen with probability depending on z. In the streaming setting, however, we cannot afford to store a color for each point of U, but must still ensure that a point receives a consistent color in each set that it appears in. To accomplish our goal, we use a *z-wise independent family $\mathcal{H}$* of hash functions $h : X \to [\bar{z}]$. Such a family $\mathcal{H}$ has the property that for every set of at most z distinct elements $(e_1, \ldots, e_z) \in X^z$, and any z (not necessarily distinct) values $(c_1, \ldots, c_z) \in [\bar{z}]^z$, the probability that $h(e_1) = c_1, h(e_2) = c_2, \ldots,$ and $h(e_z) = c_z$ is precisely $\bar{z}^{-z}$. A classical result of Wegman and Carter [53] provides a construction of such a family $\mathcal{H}$ of functions $h : [\bar{m}] \to [\bar{z}]$ when both $\bar{m}$ and $\bar{z}$ are prime powers and storing and computing each function requires a random seed of only $\mathcal{O}(z \log \bar{m})$ bits. In our setting, it suffices to set $\bar{m}$ to the smallest power of 2 larger than $|U|$ to obtain a family of functions $h : U \to [\bar{z}]$, each of which can be stored in $\mathcal{O}(z \log \bar{m}) = \mathcal{O}(z \log m)$ bits. Then, for any set Z of z points, the probability Z will be well colored by an h chosen uniformly at random from $\mathcal{H}$ is $\binom{\bar{z}}{z}\frac{z!}{\bar{z}^z} > \frac{\bar{z}^z}{z^z}\frac{z!}{\bar{z}^z} = \frac{z!}{z^z} > e^{-z}$. Thus, if we choose u functions $h \in \mathcal{H}$ independently and uniformly at random, then the probability that Z is not well-colored by at least one of them is at most $(1 - e^{-z})^u$, which is at most ϵ for $u = e^z \ln(\epsilon^{-1})$. Each such choice can be done in parallel, invoking a separate instance of the procedure in Algorithm 4.

Alternatively, we can obtain a deterministic algorithm by making use of a *z-perfect family $\mathcal{H}$* of hash functions from $U \to [\bar{z}]$. Such a family has the property that for any subset $Z \subseteq U$ of size at most z, some function $h \in \mathcal{H}$ is injective on Z. Schmidt and Siegal [48] give a construction of such a family in which each function can be specified by $\mathcal{O}(\bar{z}) + 2\log\log|U|$ bits.[2] Thus, we can simply run our streaming algorithm in parallel for each of the $2^{\mathcal{O}(\bar{z})}\log^2(m)$ such functions.

Combining the above observations, we have the following:

Theorem 5.2. *Let $\mathcal{M}$ be an ℓ-matchoid and $z \in \mathbb{Z}_+$. For any $\epsilon > 0$, there is a randomized streaming algorithm that succeeds with probability $(1-\epsilon)$ and computes a kernel R for* Maximum $(\mathcal{M}, z)$-Coverage. *Moreover, $|R| \leq (4e)^z \Gamma_{\ell,z} \ln(\epsilon^{-1})$, where $\Gamma_{\ell,z} \triangleq \sum_{q=0}^{(z-1)\ell} \ell^q$. At all times during its execution, the algorithm stores at most $|R|+1$ sets of at most $\mathcal{O}(z)$ points each and requires at most $\mathcal{O}(z\ell\log(n) + (4e)^z \ln(\epsilon^{-1}) z\log(m))$ additional bits of storage. For $\ell = 1$, we have $|R| \leq (4e)^z z\ln(\epsilon^{-1})$ and for $\ell > 1$, $|R| = \mathcal{O}\big((4e)^z \ell^{(z-1)\ell}\ln(\epsilon^{-1})\big)$.*

There is also a deterministic algorithm producing a kernel R for the same problem with $|R| \leq 2^{\mathcal{O}(z)}\Gamma_{\ell,z}\log^2(m)$. At all times during its execution, it stores at most $|R|+1$ sets of at most $\mathcal{O}(z)$ points each and uses at most $\mathcal{O}(z\ell\log n) + 2^{\mathcal{O}(z)} z\log^2(m)\log\log(m)$ additional bits of storage. For $\ell = 1$, we have $|R| = 2^{\mathcal{O}(z)} z\log^2(m)$ and for $\ell > 1$, $|R| = 2^{\mathcal{O}(z)}\ell^{(z-1)\ell}\log^2(m)$.

Proof. Let O be an optimal solution for the problem, and let Z be the set of up to z points of maximum weight covered by O. For the randomized algorithm, we process each element of the input stream with $e^z\ln(\epsilon^{-1})$ parallel executions of the procedure StreamingMaxCoverage from Algorithm 4, each with a function h sampled uniformly and independently at random from the described $\bar{z}$-wise independent family $\mathcal{H}$. We then let R be the union of all the sets produced by these processes. With probability at least $(1-\epsilon)$, Z is well-colored by one such h. Consider the process StreamingMaxCoverage corresponding to this choice of h and let R be its output. For every $C \subseteq [\bar{z}]$, all elements $e \in X_C$ will be processed by a procedure StreamingRepSet$_C$ in this instance. By Theorem 3.8, each process StreamingRepSet$_C$ used in StreamingMaxCoverage then produces a joint z-representative set R_C for $(X_C, \mathcal{M}, w_C)$. Thus, by Lemma 5.1, the output $R = \bigcup_{C\subseteq[\bar{z}]} R_C$ for this procedure is a kernel for Maximum $(\mathcal{M}, z)$-Coverage.

In total, the algorithm maintains $2^{\bar{z}}e^z\ln(\epsilon^{-1}) \leq (4e)^z\ln(\epsilon^{-1})$ procedures StreamingRepSet$_C$. By Theorem 3.8, each such procedure returns a set of R_C containing at most $\Gamma_{\ell,z} \triangleq \sum_{q=0}^{(z-1)\ell} \ell^q$ elements. For each element, we discard all but the heaviest point of each color class. Thus $|R| \leq (4e)^z\Gamma_{\ell,z}\ln(\epsilon^{-1})$ and for each element of R, we must store at most $\bar{z} = \mathcal{O}(z)$ points. When a new element arrives, we can perform the updates in each procedure sequentially, temporarily storing at most one element and using at most $\mathcal{O}(z\ell\log n)$ bits of additional storage, as shown in Theorem 3.8. Additionally, we must store $\mathcal{O}(z\log m)$ bits for the hash function in each of the $(4e)^z\ln(\epsilon^{-1})$ procedures StreamingMaxCoverage. Thus, the total number of additional bits required is at most $\mathcal{O}\big(z\ell\log(n) + (4e)^z\ln(\epsilon^{-1})z\log(m)\big)$.

For the deterministic algorithm, we proceed in the same fashion, but instead use each of the $2^{\mathcal{O}(\bar{z})}\log^2(m) = 2^{\mathcal{O}(z)}\log^2(m)$ functions in the z-perfect hash family $\mathcal{H}$, each of which requires at most $\mathcal{O}(z + \log\log(m))$ bits to store. Then, Z will be well-colored by at least one of these functions. By a similar argument as above, the union R of the $2^{\mathcal{O}(z)}\log^2(m)$ procedures StreamingMaxCoverage will then be a kernel. By a similar calculation, $|R| \leq 2^{\mathcal{O}(z)}\Gamma_{\ell,z}\log^2(m)$ and the total number of additional bits required is at most $\mathcal{O}(z\ell\log n) + 2^{\mathcal{O}(z)}z\log^2(m)\log\log(m)$. □

In Theorem 5.2, we have stated our results in the streaming setting where the primary concern is the space used by the algorithm. However, we note that our algorithms also translate directly to fixed-parameter tractable algorithms for the offline setting, in which the primary concern is computation time. Specifically, instead of processing elements in a stream using multiple instances of StreamingRepSet we can simply execute multiple instances of the offline procedure RepSet. Combining Theorem 3.7 with our analyses from the streaming setting then immediately gives the following.

Theorem 5.3. *There are fixed-parameter tractable algorithms computing a kernel R for* Maximum $(\mathcal{M}, z)$-Coverage *requiring a number of independence oracle calls proportional to n times the stated upper bounds on $|R|$ in Theorem 5.2 plus the time required to sort the input by weight for each of the $(4e)^z\ln(\epsilon)$, or $2^{\mathcal{O}(z)}\log^2(m)$ representative sets maintained, respectively.*

[2] There have been several subsequent improvements obtaining smaller families $\mathcal{H}$ of z-perfect hash functions (e.g. [1, 44, 9]). For simplicity, we use the result of [48], which gives explicit bounds on the space required for storing and computing such functions and suffices to obtain poly-logarithmic space in our setting.

References

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[2] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *KDD*, pages 671–680, 2014.

[3] MohammadHossein Bateni, Hossein Esfandiari, and Vahab S. Mirrokni. Almost optimal streaming algorithms for coverage problems. In *SPAA*, pages 13–23. ACM, 2017.

[4] Markus Bläser. Computing small partial coverings. *Information Processing Letters*, 85(6):327–331, 2003.

[5] Édouard Bonnet, Vangelis Th. Paschos, and Florian Sikora. Parameterized exact and approximation algorithms for maximum k-set cover and related satisfiability problems. *RAIRO - Theor. Inf. and Applic.*, 50(3):227–240, 2016.

[6] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

[7] Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1-2):225–247, 2015.

[8] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *ICALP*, pages 318–330, 2015.

[9] Jianer Chen, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Improved algorithms for path, matching, and packing problems. In *SODA*, pages 298–307, 2007.

[10] Rajesh Chitnis and Graham Cormode. Towards a theory of parameterized streaming algorithms. In *IPEC*, pages 7:1–7:15, 2019.

[11] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *SODA*, pages 1326–1344, 2016.

[12] Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *SODA*, pages 1234–1251, 2015.

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

[14] Shaddin Dughmi and Jan Vondrák. Limitations of randomized mechanisms for combinatorial auctions. *Games Econ. Behav.*, 92:370–400, 2015.

[15] Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In *MFCS*, pages 275–286, 2014.

[16] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45:634–652, 1998.

[17] Uriel Feige and Moshe Tennenholtz. Optimization with uniform size queries. *Algorithmica*, 78(1):255–273, 2017.

[18] Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In *NeurIPS*, pages 730–740, 2018.

[19] Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *STOC*, pages 1363–1374, 2020.

[20] Yuval Filmus and Justin Ward. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM Journal on Computing*, 43(2):514–542, 2014.

[21] Marshall L. Fisher, George L Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions—ii. *Mathematical Programming Studies*, 8:73–87, 1978.

[22] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *Journal of the ACM*, 63(4):29:1–29:60, 2016.

[23] Paritosh Garg, Linus Jordan, and Ola Svensson. Semi-streaming algorithms for submodular matroid intersection. In *IPCO*, 2021.

[24] Chien-Chung Huang, Naonori Kakimura, Simon Mauras, and Yuichi Yoshida. Approximability of monotone submodular function maximization under cardinality and matroid constraints in the streaming model. *SIAM Journal on Discrete Mathematics*, 36(1), 2022.

[25] Chien-Chung Huang, Theophile Thiery, and Justin Ward. Improved multi-pass streaming algorithms for submodular maximization with matroid constraints. In *APPROX*, 2020.

[26] T.A. Jenkyns. *Matchoids: A Generalization of Matchings and Matroids.* PhD thesis, University of Waterloo, 1974.

[27] Per M. Jensen and Bernhard Korte. Complexity of matroid property algorithms. *SIAM J. Computing*, 11(1):184–190, 1982.

[28] Konstantinos Kaparis, Adam N. Letchford, and Ioannis Mourtos. On matroid parity and matching polytopes. *Discrete Applied Mathematics*, 284:322 – 331, 2020.

[29] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[30] Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *ICML*, pages 3311–3320, 2019.

[31] Nitish Korula, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM Journal on Computing*, 47(3):1056–1086, 2018.

[32] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Mathematics of Operations Research*, 35(4):795–806, 2010.

[33] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Matroid matching: the power of local search. *SIAM Journal on Computing*, 42(1):357–379, 2013.

[34] Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. In *SODA*, pages 1914–1933, 2021.

[35] Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Transaction on Algorithms*, 14(2):14:1–14:20, 2018.

[36] Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Quasipolynomial representation of transversal matroids with applications in parameterized complexity. In *ITCS*, pages 32:1–32:13, 2018.

[37] László Lovász. The matroid matching problem. In László. Lovász and Vera T Sós, editors, *Algebraic Methods in Graph Theory, Vol. II (Colloquium Szeged 1978)*, pages 495–517, 1981.

[38] László Lovász and M. D. Plummer. *Matching theory.* North-Holland, 1986.

[39] Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k-coverage, unique set cover and related problems (via t-wise agreement testing theorem). In *SODA*, pages 62–81, 2020.

[40] Dániel Marx. A parameterized view on matroid optimization problems. *Theoretical Computer Science*, 410(44):4471–4479, 2009.

[41] Andrew McGregor, David Tench, and Hoa T. Vu. Maximum coverage in the data stream model: Parameterized and generalized. In *ICDT*, pages 12:1–12:20, 2021.

[42] Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory of Computing Systems*, 63(7):1595–1619, 2019.

[43] Pranabendu Misra, Fahad Panolan, M.S. Ramanujan, and Saket Saurabh. Linear representation of transversal matroids and gammoids parameterized by rank. *Theoretical Computer Science*, 818:51–59, 2020.

[44] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *FOCS*, pages 182–191, 1995.

[45] George L Nemhauser and Laurence A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[46] Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *ICML*, pages 3826–3835, 2018.

[47] Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SDM*, pages 697–708, 2009.

[48] Jeanette P. Schmidt and Alan Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM Journal on Computing*, 19(5):775–786, 1990.

[49] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency.* Springer, 2003.

[50] Piotr Skowron. FPT approximation schemes for maximizing submodular functions. *Information and Computation*, 257:65–78, 2017.

[51] Piotr Skowron and Piotr Faliszewski. Chamberlin-Courant rule with approval ballots: Approximating the MaxCover problem with bounded frequencies in FPT time. *Journal of Artificial Intelligence Research*, 60:687–716, 2017.

[52] René van Bevern, Oxana Yu. Tsidulko, and Philipp Zschoche. Fixed-parameter algorithms for maximum-profit facility location under matroid constraints. In *CIAC*, pages 62–74, 2019.

[53] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

[54] Laurence Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research*, 7(3):pp. 410–425, 1982.

A Hardness Results for Alternative Parameterizations

Here, we provide some justification for our choice of parameters for each of the problems we consider by showing that the problems become hard if we use any strict subset of the parameters proposed.

First, we note that the 3-dimensional matching problem, which the 3-matchoid problem generalizes, is one of Karp's original NP-hard problems [29]. It follows that all of our problems remain NP-hard when parameterized by ℓ alone.

For linear objectives, we parameterize by ℓ and k. Here, we note that if we parameterize by k alone, we can encode an arbitrary instance of the Independent Set problem, where k is the size of the independent set. This problem is known to be $W[1]$-hard [13]. Given an arbitrary graph $G = (V, E)$, we let our ground set X be V, and use an unweighted objective that sets $w(e) = 1$ for each $e \in V$. Then, we introduce a uniform matroid of rank 1 on $\{u, v\}$ with each edge $(u, v) \in E$. Note that some $S \subseteq X = V$ is then independent in all matroids if and only if no pair of vertices in S share an edge. Moreover, we have a solution of value at least k for our problem if and only if we can select k elements from S and so have an independent set of size k in G.

For coverage functions, parameterizing by the number k of elements chosen immediately gives the Maximum k-Coverage problem, which is $W[2]$ hard [5]. Here, we parameterize instead by the number of points z that are covered and ℓ. If instead we parameterize by only z, we can again encode an arbitrary instance of Independent Set as described above. We encode our unweighted objective by letting each element of X cover a single, unique point. Then, similar to the discussion for the case of linear functions, we have an independent set of size z in G if and only if we have a set of elements that is independent in all our matroids covering z points.