Journal of Artificial Intelligence Research 60 (2017) 687–716 Submitted 07/17; published 11/17

Chamberlin–Courant Rule with Approval Ballots: Approximating the MaxCover Problem with Bounded Frequencies in FPT Time

Piotr Skowron P.K.SKOWRON@GMAIL.COM
TU Berlin,
Berlin, Germany

Piotr Faliszewski FALISZEW@AGH.EDU.PL
AGH University,
Kraków, Poland

Abstract

We consider the problem of winner determination under Chamberlin–Courant's multiwinner voting rule with approval utilities. This problem is equivalent to the well-known NP-complete MaxCover problem and, so, the best polynomial-time approximation algorithm for it has approximation ratio $1-1/e$. We show exponential-time/FPT approximation algorithms that, on one hand, achieve arbitrarily good approximation ratios and, on the other hand, have running times much better than known exact algorithms. We focus on the cases where the voters have to approve of at most/at least a given number of candidates.

1. Introduction

We study the complexity of winner determination under the Chamberlin–Courant multiwinner voting rule with approval ballots. Intuitively, the Chamberlin–Courant rule proceeds as follows. Consider a setting where we are given a set C of candidates, a collection V of voters, and the goal is to pick a committee of K candidates that, in some sense, best represent the voters. We consider the approval model, i.e., the case where the voters express their preferences by approving subsets of candidates. The Chamberlin–Courant rule picks a set of K candidates that maximizes the number of voters who approve at least one of the selected candidates.

On the formal level, the approval-based variant of the Chamberlin–Courant rule is equivalent to the MaxCover problem—i.e., to the problem where one aims to cover as many elements as possible by using a given number of sets (we will elaborate on this relation in Section 1.3). While most of the technical part of the paper is written in the language of the MaxCover problem, our main motivating assumptions and discussions stem from the voting setting.

1.1 Applications of the Chamberlin–Courant Rule

While Chamberlin and Courant (1983) proposed their rule as a mean of electing committees of representatives, such as parliaments, university senates, etc., its applicability in political domains requires some comment. Indeed, it has been often discussed that this rule is not perfectly suited for electing committees that make decisions on behalf of societies. For

instance, consider the case where the goal is to select $K = 3$ committee members, and where 98% of voters approve candidates c_1, c_2, and c_3, 1% approve candidate c_4, and 1% approve candidate c_5. In this case, the Chamberlin and Courant rule will elect committee $\{c_1, c_4, c_5\}$, which clearly underrepresents the overwhelming majority of 98% of the voters. One solution to this problem has already been discussed by Chamberlin and Courant: They suggested that when the goal is to select a decision-making body, the elected committee members should use weighted voting to make their decisions. Another one was proposed by Monroe (1995), who complemented the Chamberlin–Courant rule with a mechanism to ensure proportional representation of the voters (not discussed in this paper). Without Monroe's modification, the Chamberlin–Courant rule is far more appealing in the context of *deliberative democracy*—where the primary goal is to elect committees that represent as diverse sets of opinions as possible—than in the context of proportional representation.

However, applications of the Chamberlin–Courant rule reach far beyond the political domain. For instance, consider a company aiming to choose a few types of a certain product to advertise to its customers (e.g., a company that tries to decide which models of cell phones it should start to produce) (Lu & Boutilier, 2011; Oren & Lucier, 2014). Since a typical customer would either buy only one unit of a given product or would not buy it at all, it is natural to model the decision faced by the company as the problem of finding winners under the Chamberlin–Courant rule (where the potential customers are modeled as the voters and the product types are modeled as the candidates). The Chamberlin–Courant rule found many other applications, for example, in resource allocation (Skowron, Faliszewski, & Slinko, 2015a; Skowron, Faliszewski, & Lang, 2016), in certain variants of facility location problems—see, for example, discussions provided by Procaccia, Rosenschein and Zohar (2008) and Betzler, Slinko and Uhlmann (2013)—in diversifying search results (Skowron, Lackner, Brill, Peters, & Elkind, 2017), in personalized recommendation and advertisement (Lu & Boutilier, 2015), or in improving genetic algorithms (Faliszewski,Sawicki, Schaefer, & Smołka, 2017a; Sawicki, Smołka, Łoś, Schaefer, & Faliszewski, 2017). The MaxCover problem, which as we will explain later, is equivalent to the problem of finding winners according to the Chamberlin–Courant rule, is important for many other areas, such as computational biology (Vandin, Upfal, & Raphael, 2011) and networking (Kuo, Lin, & Tsai, 2015).

Finally, the Chamberlin–Courant rule can be considered as a prime example of a multiwinner rule aimed at achieving *diversity* of outcomes in multiwinner elections. For a more detailed discussion on the principle of diversity and its significance, and for more discussion on the Chamberlin–Courant rule we refer the reader to the work of Elkind, Faliszewski, Skowron and Slinko (2017), to a recent book chapter by Faliszewski, Skowron, Slinko and Talmon (2017c), and to an axiomatic study by Lackner and Skowron (2017).

1.2 Approval Ballots

The Chamberlin–Courant rule was originally defined for the case where the voters express their preferences by ranking the candidates from the most to the least desirable ones. However, the rule can be naturally extended to the case where instead of ranking the candidates, the voters assign cardinal utilities to them (Skowron et al., 2016). Approval ballots are an extreme example of cardinal utilities, where for each candidate a voter can either give his or her full support for the candidate or indicate complete lack of support. While very

simple, the approval model has a number of advantages (Kilgour, 2010). First, providing approval information puts much less cognitive burden on the voters than providing preference rankings or more involved cardinal utilities. Second, using approval ballots can positively influence the willingness of voters to participate in elections and can reduce the effect of negative campaigning (Brams & Herschbach, 2001). Third, approval ballots are relatively similar to the first-past-the-post ballots (where each voter provides a single candidate, one that he or she votes for) and, thus, may be more easily adopted in real-life elections. For more arguments regarding advantages of using approval ballots we refer the reader to the works of Aragones, Gilboa and Weiss (2011), Brams and Fishburn (2010), and Laslier and Van der Straeten (2016).

Unfortunately, the approval-based variant of the Chamberlin–Courant rule is also harder to compute than the variant with preference rankings based on the Borda scoring function. Indeed, while it is known that computing winners under the Chamberlin–Courant rule is NP-hard for both variants—see the work of Procaccia et al. (2008) and of Lu and Boutilier (2011)—there are far stronger approximation algorithms for the Borda-based one. In particular, for the Borda-based variant there is a (practically useful) polynomial-time approximation scheme (Skowron et al., 2015a), and for the approval-based variant it is known that unless P = NP, there exists no polynomial-time algorithm that achieves approximation ratio better than $1 - 1/e$ (Feige, 1998).

1.3 Our Contribution

The approval-based variant of the Chamberlin–Courant rule turns out to be equivalent to the MaxCover problem. In the MaxCover problem we are given a set N of n elements, a family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of m subsets of N, and an integer K. The goal is to find a size-at-most-K subcollection of $\mathcal{S}$ that contains (covers) as many elements from N as possible. Each instance of the MaxCover problem one-to-one corresponds to an approval-based Chamberlin–Courant election as follows: Each set $S_j \in \mathcal{S}$ is a candidate, each element from N is a voter, and each element e "approves" those sets S_j that include it. Choosing a set S_j to be included in the winning committee means "covering" all the elements that belong to it. Thus a winning size-K committee directly corresponds to a solution for the MaxCover instance. In the technical part of the paper we focus on studying the more abstract MaxCover problem, but we explain our various assumptions in terms of the Chamberlin–Courant voting rule.

The standard greedy algorithm for MaxCover that iteratively picks sets that cover most yet-uncovered elements has an approximation ratio $1 - 1/e$ and this is optimal unless P = NP—see, e.g., the textbook of Hochbaum (1996) for the algorithm and the work of Feige (1998) for the approximation lower bound. Thus in our work we focus on exponential-time algorithms that, on one hand, achieve arbitrarily good approximation ratios (i.e., are *approximation schemes*) and, on the other hand, have running times significantly better than the known exact algorithms—indeed, we are primarily interested in fixed-parameter tractable (FPT) approximation schemes, parameterized by the number of sets allowed in the solution.

Due to the motivations stemming from voting, we found three domain restrictions for the MaxCover problem, which, on the one hand, are quite natural and well-motivated, and, on the other hand, allow us to design better approximation algorithms, though running in

super-polynomial time. We consider three variants of the MaxCover problem, depending on the restrictions regarding elements' frequencies (an element's frequency is the number of sets it belongs to; in the voting domain, it corresponds to the number of candidates a voter approves):

Upper-bounded frequencies. In this variant we assume that there is some constant p such that each element appears in at most p sets. This variant corresponds to winner determination under the Chamberlin–Courant rule where each voter approves of at most p candidates (this is quite a natural restriction; often the voters have energy to express approvals for a small set of candidates only, and sometimes such upper bounds are even put forward by election laws[1]). For this case we show FPT approximation schemes (deterministic and randomized) with respect to the parameter $K + p$.

Lower-bounded frequencies. In this variant we require that there is some constant p such that each element belongs to at least p sets. This corresponds to a setting where each voter is required to approve of at least p candidates. For this case we show a slightly improved analysis of the standard, polynomial-time, greedy algorithm.

Unrestricted case. In this variant we put no restrictions on the MaxCover inputs. While we were unable to find FPT approximation schemes in this case (randomized or not), using an approach introduced by Cygan, Kowalik and Wykurz (2009), we show exponential-time approximation schemes that seamlessly exchange running time for the quality of the approximation.

We conclude the paper by discussing the consequences of our results from the point of view of winner determination under the Chamberlin–Courant rule. We also provide a number of interesting research questions that stem from our work.

1.4 Related Work

Chamberlin and Courant (1983) proposed their rule for votes expressed in the form of preference rankings. Later, based on their work, Procaccia et al. (2008) suggested the approval-based variant. Recently it became known that, in fact, the approval-based variant was already mentioned as an appealing tool for selecting representative bodies already in the 19th century, by Thiele (1895).

Winner determination under the Chamberlin–Courant rule received quite a lot of attention in recent years. Its worst-case complexity was studied by Procaccia et al. (2008) (for the case of approval utilities) and by Lu and Boutilier (2011) (for the case of Borda utilities). Lu and Boutilier have also shown the greedy polynomial-time $(1 - 1/e)$-approximation algorithm for the rule. Skowron et al. (2015a) have shown a polynomial-time approximation scheme for the case of Borda utilities and evaluated it empirically. Other authors have studied the parameterized complexity of the rule and its complexity in restricted domains (Betzler et al., 2013; Yu, Chan, & Elkind, 2013; Skowron, Yu, Faliszewski, & Elkind, 2015b;

1. Indeed, the ballots in Polish parliamentary elections require each voter to provide three candidates that this voter approves of. While Polish parliamentary elections are not based on the Chamberlin–Courant rule itself, this shows that our assumptions are realistic.

Talmon, 2017) as well as in online settings (Oren & Lucier, 2014; Dey, Talmon, & Handel, 2017), or using heuristic algorithms (Faliszewski, Slinko, Stahl, & Talmon, 2016; Faliszewski, Skowron, Slinko, & Talmon, 2017b).

The Chamberlin–Courant rule is not the only natural way of selecting committees based on approval information. Various such rules are reviewed by Kilgour (2010) and are studied algorithmically, e.g., by LeGrand, Markakis and Mehta (2007), Caragiannis, Kalaitzis and Markakis (2010), Aziz, Gaspers, Gudmundsson, Mackenzie, Mattei and Walsh (2015), and Skowron et al. (2016). Naturally, there are also many other multiwinner voting rules.

On the technical side, our work provides parameterized complexity analysis of the MaxCover problem. Somewhat surprisingly. this problem received relatively little attention in the literature. Independently from our work, Bonnet, Paschos and Sikora (2016) also studied its complexity. In particular, they showed that under the standard complexity assumptions, for any constant $c > 1$ the MaxCover problem cannot be approximated within ratio $1 - \frac{c}{n}$, where n is the number of elements to be covered, in time parameterized by the size of the solution K. On the other hand, they provided an algorithm that for each $\mu > \frac{e-2}{e-1}$ and each $\epsilon > \left(1 - \frac{1}{e}\right)\mu^2 - \left(1 - \frac{2}{e}\right)\mu$ approximates MaxCover with ratio $1 - \frac{1}{e} + \epsilon$, in an FPT time for a parameter (K, Δ), where Δ is the maximum cardinality of a set. It is also known that there exists an FPT algorithm for MaxCover parameterized by the number of elements to cover (Bläser, 2003).

On the other hand, researchers often consider MaxVertexCover, a much-restricted variant of MaxCover in which we are given a graph $G = (V, E)$ and an integer K, and we ask for at most K vertices that jointly cover as many edges as possible (i.e., it is a "Max" variant of the standard VertexCover problem). We stress that MaxVertexCover is considerably simpler even than MaxCover with frequencies bounded by two (we will explain this in more detail in the discussion below Proposition 4.1 in Section 4.1). Currently the best polynomial-time approximation algorithm for MaxVertexCover, due to Ageev and Sviridenko (1999), has an approximation ratio of $3/4$. Parameterized complexity of MaxVertexCover was first studied by Guo, Niedermeier and Wernicke (2007). The problem was also studied by Cai (2008), who gave the currently best exact algorithm for it, and by Marx (2008), who gave an FPT approximation scheme; interestingly, our more general algorithm is faster than that of Marx (see the discussion after Proposition 4.1).

Leaving the realm of FPT running time, Croce and Paschos (2012) provided an exponential-time approximation strategy for MaxVertexCover, based on combining exact (exponential-time) algorithms with (polynomial-time) approximation ones. We use a similar idea—also based on the work of Cygan et al. (2009)—for the case of the unrestricted MaxCover problem and compare it to their approach.

2. Preliminaries

In the introduction we have presented a natural one-to-one correspondence between the Chamberlin–Courant rule and the MaxCover problem. Thus, throughout the paper, we focus on the MaxCover problem as this way our results are useful both to people interested in multiwinner voting and to those interested in the abstract MaxCover problem only. We assume familiarity with standard notions regarding (approximation) algorithms and (pa-

rameterized) complexity theory, but we provide a brief review. For each positive integer n, we write $[n]$ to mean $\{1, \ldots, n\}$.

Let $\mathcal{P}$ be an algorithmic problem where, given some instance I, the goal is to find a solution s that maximizes a certain function f. Given an instance I of $\mathcal{P}$, we refer to the value $f(s)$ of an optimal solution s as $\mathrm{OPT}(I)$ (or simply as OPT if the instance I is clear from the context). Let γ, $0 < \gamma \leq 1$, be some fixed constant. An algorithm $\mathcal{A}$ that given instance I returns a solution s' such that $f(s') \geq \gamma \mathrm{OPT}(I)$ is called a γ-approximation algorithm for $\mathcal{P}$. Analogously, we define $\mathrm{OPT}(I)$ and the notion of a γ-approximation algorithm, $\gamma > 1$, for the case of a problem $\mathcal{P}'$, where the task is to find a solution that minimizes a given goal function g. Given instance I of some algorithmic problem, we write $|I|$ to denote the length of the standard, efficient encoding of I. In this paper we focus on the following two problems (the former directly models winner determination under the Chamberlin–Courant rule, whereas the latter models a variant of the rule where we measure voter's dissatisfaction; the number of voters that are not represented by someone they approve of).

Definition 2.1. *An instance $I = (N, \mathcal{S}, K)$ of the MaxCover problem consists of a set N of n elements, a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of m subsets of N, and nonnegative integer K. The goal is to find a subcollection $\mathcal{C}$ of $\mathcal{S}$ of size at most K that maximizes $|\bigcup_{S \in \mathcal{C}} S|$.*

Definition 2.2. *The MinNonCovered problem is defined in the same way as the MaxCover problem, but the goal is to find a subcollection $\mathcal{C}$ such that $|N| - |\bigcup_{S \in \mathcal{C}} S|$ is minimal.*

In the decision variant of MaxCover (of MinNonCovered) we are additionally given an integer T (an integer T') and we ask if there is a collection of up to K sets from $\mathcal{S}$ that cover at least T elements (that leave at most T' elements uncovered). MaxVertexCover is a variant of MaxCover where we are given a graph $G = (V, E)$, the edges are the elements to be covered, and vertices define the sets that cover them (a vertex covers all the incident edges). SetCover and VertexCover are special cases of the decision variants of MaxCover and MaxVertexCover, where we have to cover all the elements (all the edges).

MaxCover and MinNonCovered are equivalent as far as optimal solutions go, but they are quite different in terms of their approximation properties. For example, if there is a solution that covers all the elements, then a γ-approximation algorithm for MaxCover can cover a γ fraction of them, but a γ-approximation algorithm for MinNonCovered has to cover them all.

Given an instance I of MaxCover (MinNonCovered), we say that an element e has frequency t if it appears in exactly t sets. We mostly focus on the variants of MaxCover and MinNonCovered where there is a given constant p such that each element's frequency is at most p. We refer to these problems as variants with upper-bounded frequencies. (It is tempting to think that MaxCover with frequencies equal to two is simply MaxVertexCover, but in fact it is a considerably richer problem. In the former, two sets can share many elements, while in the latter two vertices may be connected by at most one edge. Thus, MaxCover with frequencies equal to two is closer in spirit to MaxVertexCover on multigraphs.)

Our focus is on (approximation) algorithms that run in FPT time—see the books of Downey and Fellows (2013), Niedermeier (2006), Flum and Grohe (2006), and Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk, Pilipczuk and Saurabh (2015) for details on parameterized complexity theory and fixed-parameter tractable algorithms. To speak of an FPT

algorithm for a given problem, we declare a part of the problem as the so-called parameter. Here, for MaxCover and MinNonCovered problems, we take the parameter to be the number K of sets that we are allowed to use in the solution (in Section 3 we briefly consider MaxCover/MinNonCovered with parameters T, T', and their combinations with K). Given an instance I of a problem with parameter K, an FPT algorithm is required to run in time $f(K)\text{poly}(|I|)$, where f is some computable function and $\text{poly}(\cdot)$ is some polynomial.

We say that an algorithm is an FPT approximation scheme for a maximization problem $\mathcal{P}$, if for each fixed $\gamma \in (0,1)$ it finds γ-approximate solutions for instances of $\mathcal{P}$ in FPT time. An FPT approximation scheme for a minimization problem is defined analogously.

From the point of view of parameterized complexity, FPT is seen as the class of tractable problems. There is also a whole hierarchy of hardness classes, $\text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \subseteq \cdots \text{W[P]} \subseteq \cdots$. The standard definitions of $\text{W[1]}, \text{W[2]}, \ldots$ are quite involved; we point interested readers to appropriate overviews (Cygan et al., 2015; Downey and Fellows, 2013; Flum and Grohe, 2006; Niedermeier, 2006). We use equivalent definitions that use an appropriate reduction notion and these classes' complete problems.

Definition 2.3. *Let $\mathcal{P}$ and $\mathcal{P}'$ be two decision problems. For given parameters of $\mathcal{P}$ and $\mathcal{P}'$, we say that $\mathcal{P}$ reduces to $\mathcal{P}'$ through a parameterized reduction if there exist a mapping $F\colon \mathcal{P} \to \mathcal{P}'$ (computable in FPT time) and two computable functions, $g\colon \mathbb{R}_+ \to \mathbb{R}_+$ and $h : \mathbb{R}_+ \to \mathbb{R}_+$, such that:*

(i) for each instance $I \in \mathcal{P}$ the answer to I is "yes" if and only if the answer to $F(I) = I'$ is "yes",

(ii) K and K' are the values of the parameters for instances I and I', respectively,

(iii) $|I'| \leq g(K)\text{poly}(|I|)$, and

(iv) $K' \leq h(K)$.

W[1] is the class of all problems for which there is a parameterized reduction to the Clique problem (i.e., the problem where we ask if a given graph $G = (V, E)$ has a clique of size at least K, where K is the parameter). W[2] is the class of all problems with parameterized reductions to SetCover (with parameter K). Interestingly, VertexCover is well-known to be in FPT, but MaxVertexCover is W[1]-complete (Guo et al., 2007).

There are several standard techniques for showing membership in W[1], W[2], etc. In some of our proofs we show W[1]- and W[2]-membership. To show W[2]-membership, we reduce our problem to SetCover. For W[1]-membership, we reduce to the Short-Nondeterministic-Turing-Machine-Computation problem, shown to be W[1]-complete for parameter k by Cesati (2003).

Definition 2.4. *In the Short-Nondeterministic-Turing-Machine-Computation problem we are given a single-tape nondeterministic Turing machine M (described as a tuple including the input alphabet, the work alphabet, the set of states, the transition function, the initial state and the accepting/rejecting states), a string x over M's input alphabet, and an integer k. The question is whether there is an accepting computation of M that accepts x within k steps.*

The Bounded-Nondeterministic-Turing-Machine-Computation problem is defined similarly, but in addition we are also given an integer m, and we ask if M accepts its input within m steps, of which at most k are nondeterministic. Cesati has shown that this problem is W[P]-complete (Cesati, 2003); we omit the exact definition of W[P]; the reader can think of W[P] as the set of problems that have parameterized reductions to the Bounded-Nondeterministic-Turing-Machine-Computation problem.

3. Worst-Case Complexity Results

To justify seeking FPT approximation algorithms, we first investigate the parametrized complexity of the MaxCover problem. For upper-bounded frequencies, the problem is W[1]-hard—this follows from the fact that MaxVertexCover is W[1]-hard (Guo et al., 2007)—and we show that it, indeed, is W[1]-complete. For lower-bounded frequencies it is W[2]-hard and in W[P].

Theorem 3.1. *(1) For each constant p greater or equal to 2, the MaxCover problem with frequencies upper-bounded by p is* W[1]*-complete (when parameterized by the number of sets in the solution). (2) For each constant p, $p \geq 1$, MaxCover where each element belongs to at least p sets is* W[2]*-hard and belongs to* W[P] *(when parameterized by the number of sets in the solution).*

Proof. Let us consider part (1) first. The hardness follows directly from the W[1]-hardness of the MaxVertexCover problem (Guo et al., 2007). We prove membership in W[1] by reducing MaxCover with bounded frequencies to the Short-Nondeterministic-Turing-Machine-Computation problem.

Let p be some fixed constant and let $I = (N, \mathcal{S}, K, T)$ be our input instance, where N is a set of elements, $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a family of subsets of N (each element from N appears in at most p sets from $\mathcal{S}$), and K and T are two integers. This is the decision variant of the problem, thus we have T in the input; we ask if there is a collection of up to K sets from $\mathcal{S}$ that jointly cover at least T elements. W.l.o.g., we assume that $K < m$. We form a single-tape nondeterministic Turing machine M to execute the following algorithm (on empty input string); the idea of the algorithm is to employ the standard inclusion-exclusion principle:

1. Guess the indices $i_1, \ldots, i_K$ of K sets from $\mathcal{S}$.
2. Set $b = 0$.
3. For each subset A of $\{i_1, \ldots, i_K\}$ of size up to p, do the following: If $|A|$ is odd, add $|\bigcap_{i \in A} S_i|$ to b, and otherwise subtract $|\bigcap_{i \in A} S_i|$ from b.
4. If $b \geq T$ then we accept and otherwise we reject.

It is easy to see that this algorithm can indeed be implemented on a single-tape nondeterministic Turing machine with a sufficiently large (but polynomially bounded) work alphabet and state space (recall that we treat p as a constant). The only issue that might require a comment is the computation of $|\bigcap_{i \in A} S_i|$. Since sets A contain at most p elements, we can precompute these values and store them in M's transition function.

The correctness of the algorithm follows directly from the inclusion-exclusion principle and the fact that each element appears in at most p sets:

$$|S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_K}| = \sum_{\ell \in [K]} |S_{i_\ell}| - \sum_{\substack{\ell' \in [K] \\ \ell'' \in [K] \\ \ell' \neq \ell''}} |S_{i_{\ell'}} \cap S_{i_{\ell''}}| + \sum_{\substack{\ell' \in [K] \\ \ell'' \in [K] \\ \ell''' \in [K] \\ \ell' \neq \ell'' \\ \ell' \neq \ell''' \\ \ell'' \neq \ell'''}} |S_{i_{\ell'}} \cap S_{i_{\ell''}} \cap S_{i_{\ell'''}}| - \cdots$$

In general, the above formula should include intersections of up to K sets. However, since in our case each element appears in at most p sets, the intersection of more than p sets are always empty. This shows that the algorithm is correct and concludes the proof.

Let us now consider part (2) of the theorem. To show W[2]-hardness, we give a reduction from SetCover. In the SetCover problem we ask whether there exist K subsets that cover all the elements (we give a reduction for the parameter K). Let $I = (N, \mathcal{S})$ be an input instance of SetCover. W.l.o.g., we can assume that each element from N belongs to at least one set in $\mathcal{S}$. We form an instance I' of MaxCover which is identical to I, except (a) for each $e \in N$, we modify $\mathcal{S}$ to additionally include $p-1$ copies of set $\{e\}$, and (b) we require all elements to be covered (i.e., we set $T = |N|$). Clearly, in I' each element belongs to at least p sets and I' is a yes-instance of MaxCover if and only if I is a yes-instance of SetCover.

To show W[P]-membership, we give a reduction from MaxCover to the Bounded-Nondeterministic-Turing-Machine-Computation problem. On input $I = (N, \mathcal{S}, K, T)$, where N, $\mathcal{S}$, and K are as usual and T is the lower bound on the number of elements that we should cover, we produce a machine that on empty input executes the following algorithm:

1. It nondeterministically guesses up to K names of sets from $\mathcal{S}$ and writes these names on the tape (each name of a set from $\mathcal{S}$ is a single symbol).

2. Deterministically, for each name of the set produced in the previous step, the machine writes on the tape the names of those elements from this set that have not been written on the tape yet.

3. The machine counts the number of names of elements written on the tape. If there were at least T of them, it accepts. Otherwise it rejects.

It is easy to see that we can produce a description of such a machine in polynomial time with respect to $|I|$. Further, it is clear that its nondeterministic running time is bounded by some polynomial of $|I|$ and that it makes at most K nondeterministic steps. □

For parameter T, the number of elements that we should cover, Bläser (2003) gave an FPT algorithm for MaxCover. On the other hand, for parameter $T' = n - T$, i.e., the number of elements we can leave uncovered (this means considering the MinNonCovered problem), we show that the problem is para-NP-complete (i.e., it is NP-complete even for a constant value of the parameter), but becomes W[2]-complete for the joint parameter (K, T').

Theorem 3.2. *(1) The MaxCover problem is para-*NP*-complete when parameterized by the number T' of elements that can be left uncovered. This holds even if each element's frequency is upper-bounded by some constant p, $p \geq 2$. (2) MaxCover is* W[2]*-complete when parameterized by both the number K of sets that can be used in the solution and the number T' of elements that can be left uncovered.*

Proof. We start with part (1). The following trivial reduction from SetCover to MinNonCovered suffices: Given an input instance $I = (N, \mathcal{S}, K)$, output an instance $(N, \mathcal{S}, K, 0)$, i.e., an identical one, where we require that the number of elements left uncovered is zero. Since the reduction is clearly correct and works for the constant value of the parameter, we get para-NP-completeness. To obtain the result for upper-bounded frequencies, simply use VertexCover instead of SetCover in the reduction.

We now consider part (2) of the theorem. We obtain W[2]-hardness by simply observing that the reduction given in part (1) suffices. To prove W[2]-membership, we give a reduction from MaxCover (with parameter (K, T')) to SetCover (with parameter K).

Let $I = (N, \mathcal{S}, K, T')$ be an input instance of MaxCover. We form an instance $I' = (N', \mathcal{S}', K + T')$ of SetCover as follows: Let $N' = N \cup D' \cup D''$, where $D' = \{d'_1, \ldots, d'_K\}$ and $D'' = \{d''_1, \ldots, d''_{T'}\}$. We set $\mathcal{S}' = \mathcal{S}'_1 \cup \mathcal{S}'_2$, where:

(a) $\mathcal{S}'_1 = \{S \cup \{d_i\} : (S \in \mathcal{S}) \wedge (d'_i \in D')\}$, and

(b) $\mathcal{S}'_2 = \{\{e, d''_i\} : e \in N, d''_i \in D''\}$.

We observe that if I is a yes-instance of MaxCover then I' is a yes-instance of SetCover: If for I it is possible to cover $n - T'$ elements of N using K sets, then for I it is possible to (a) use K sets from $\mathcal{S}'_1$ to cover $n - T'$ elements from N and all the elements from D', and (b) use T' sets from $\mathcal{S}'_2$ to cover all the elements from D'' and the remaining T' elements from N. For the other direction, assume that I' is a yes-instance of SetCover. We will show that covering the elements from D' requires one to use at least K sets from $\mathcal{S}'_1$ (which correspond to the sets from $\mathcal{S}$) and that covering the elements in D'' requires at least T' sets from $\mathcal{S}'_2$. Since each set from $\mathcal{S}'_2$ covers exactly one element from N, we see that if I' is a yes-instance, then it must be possible to cover at least $|N| - T'$ elements from N using K sets from $\mathcal{S}$. □

The results from this section are summarized in Table 1. Indeed, our problems are hard from the parameterized complexity point of view and, thus, it is reasonable to seek FPT approximation algorithms for them (which we do in the next section).

From the point of view of the approval-based Chamberlin–Courant rule, our results say that: (a) computing small winning committees is computationally hard, even if we put upper bounds on the number of candidates approved by each voter (hardness for MaxCover with parameter K), (b) computing committees that are satisfying to almost all voters also is computationally difficult (hardness for MinNonCovered with parameter T'), even if these committees are small (hardness for MinNonCovered with joint parameter K, T'), but (c) computing committees that are satisfying for few voters only is tractable (FPT result for MaxCover with parameter T; this is hardly an appealing setting, though). This reinforces our desire to find FPT approximation algorithms.

parameter	the complexity for the case of unrestricted frequencies	upper-bounded frequencies
K	W[2]-hard, in W[P]	W[1]-complete
T	FPT (Bläser, 2003)	FPT (Bläser, 2003)
T'	para-NP-complete	para-NP-complete
(K, T')	W[2]-complete	W[1]-complete

Table 1: Parameterized worst-case complexity results for unrestricted MaxCover and MinNonCovered. The parameters are as follows: K is the number of sets we can use in the solution, T is the number of elements we are required to cover, and $T' = n - T$ is the number of elements we can leave uncovered (considering this parameter, in essence, means considering the MinNonCovered problem).

4. Algorithms for the Bounded Frequencies Cases

In this section we present our approximation algorithms for the MaxCover and MinNonCovered problems, for the case where we either upper-bound or lower-bound the frequencies of the elements. We first consider the MaxCover problem, both with upper-bounded frequencies and with lower-bounded frequencies, and then move on to the MinNonCovered problem with upper-bounded frequencies.

4.1 The MaxCover Problem with Upper Bounded Frequencies

We start by presenting an FPT approximation scheme for MaxCover with upper-bounded frequencies. While Marx (2008) has already shown an FPT approximation scheme for MaxVertexCover, his approach cannot be generalized to the MaxCover problem with bounded frequencies (it relies on the fact that two sets can have at most one element in common, which is not true in our case). Also, when we consider inputs from the MaxVertexCover only, our algorithm turns out to be faster than his. We give a more detailed comparison of the two algorithms after presenting our approach.

Our algorithm works in a very simple way. Given an instance $I = (N, \mathcal{S}, K)$ of MaxCover (with frequencies bounded by some constant p) and a required approximation ratio γ, the algorithm simply picks some of the sets from $\mathcal{S}$ with highest cardinalities (the exact number of these sets depends only on K, p, and γ), tries all K-element subcollections of sets from this group, and returns the best one. This approach is formalized as Algorithm 1. The following theorem explains that indeed the algorithm achieves the required approximation ratio.

Theorem 4.1. *Fix a positive integer p. For each instance $I = (N, \mathcal{S}, K)$ of MaxCover, where each element from N appears in at most p sets in $\mathcal{S}$, for each $\gamma \in (0, 1)$, Algorithm 1 outputs a γ-approximate solution in time* $\text{poly}(|I|) \cdot \binom{\frac{2pK}{(1-\gamma)}+K}{K}$.

Proof. Establishing the running time is immediate and so we focus on showing the approximation ratio. Consider an input instance I. Let $\mathcal{C}$ be the solution returned by Algorithm 1 and let $\mathcal{C}^*$ be some optimal solution. Let c be an arbitrary function such that for each element e for which there is some $S \in \mathcal{C}^*$ containing e, $c(e)$ is some $S \in \mathcal{C}^*$ containing e. We

Algorithm 1: An algorithm for the MaxCover problem with upper-bounded frequencies.

Parameters:
$(N, \mathcal{S}, K)$ — input MaxCover instance
p — bound on the number of sets each element can belong to
γ — the required approximation ratio of the algorithm

$\mathcal{A} \leftarrow \lceil \frac{2pK}{(1-\gamma)} + K \rceil$ sets from $\mathcal{S}$ with highest cardinalities ;
return *K-element subset of $\mathcal{A}$ that covers the largest number of elements* ;

refer to c as the *coverage function*. Intuitively, the coverage function assigns to each element covered under $\mathcal{C}^*$ (by, possibly, many different sets) a set "responsible" for covering it. We say that S covers e if and only if $c(e) = S$. Let OPT be the number of elements covered by $\mathcal{C}^*$.

We will show that $\mathcal{C}$ covers at least γOPT elements. Naturally, the reason why $\mathcal{C}$ might cover fewer elements than $\mathcal{C}^*$ is that some sets from $\mathcal{C}^*$ may not be present in $\mathcal{A}$, the set of the subsets considered by the algorithm. We will show an iterative procedure that starts with $\mathcal{C}^*$ and, step by step, replaces those members of $\mathcal{C}^*$ that are not present in $\mathcal{A}$ with the sets from $\mathcal{A}$. The idea of the proof is to show that each such replacement decreases the number of covered element by at most a small amount.

Let $\ell = |\mathcal{C}^* \setminus \mathcal{A}|$. Our procedure will replace the ℓ sets from $\mathcal{C}^*$ that do not appear in $\mathcal{A}$ with ℓ sets from $\mathcal{A}$. We rename the sets so that $\mathcal{C}^* \setminus \mathcal{A} = \{S_1, \ldots, S_\ell\}$. We will replace the sets $\{S_1, \ldots, S_\ell\}$ with sets $\{S'_1, \ldots, S'_\ell\}$ defined through the following algorithm. Assume that we have already computed sets $S'_1, \ldots, S'_{i-1}$ (thus for $i = 1$ we have not yet computed anything). We take S'_i to be a set from $\mathcal{A} \setminus (\mathcal{C}^* \cup \{S'_1, \ldots, S'_{i-1}\})$ such that the set $(\mathcal{C}^* \setminus \{S_1, \ldots, S_i\}) \cup \{S'_1, \ldots, S'_i\}$ covers as many elements as possible. During the i'th step of this algorithm, after we replace S_i with S'_i, we modify the coverage function as follows: (1) for each element e such that $c(e) = S_i$, we set $c(e)$ to be undefined; (2) for each element $e \in S'_i$, if $c(e)$ is undefined then we set $c(e) = S'_i$.

After replacing S_i with S'_i, it may be the case that fewer elements are covered by the resulting collection of sets. Let x_i denote the difference between the number of elements covered by $(\mathcal{C}^* \setminus \{S_1, \ldots, S_i\}) \cup \{S'_1, \ldots, S'_i\}$ and by $(\mathcal{C}^* \setminus \{S_1, \ldots, S_{i-1}\}) \cup \{S'_1, \ldots, S'_{i-1}\}$ (or 0, if by a fortunate coincidence there are more elements covered after replacing S_i with S'_i). By the construction of the set $\mathcal{A}$ and the fact that $S_i \notin \mathcal{A}$, each set from $\mathcal{A}$ contains more elements than S_i. We infer that every set from $\mathcal{A} \setminus (\mathcal{C}^* \cup \{S'_1, \ldots, S'_{i-1}\})$ must contain at least x_i elements covered by $(\mathcal{C}^* \setminus \{S_1, \ldots, S_{i-1}\}) \cup \{S'_1, \ldots, S'_{i-1}\}$. Indeed, if some set $S' \in \mathcal{A} \setminus (\mathcal{C}^* \cup \{S'_1, \ldots, S'_{i-1}\})$ contained fewer than x_i elements covered by $(\mathcal{C}^* \setminus \{S_1, \ldots, S_{i-1}\}) \cup \{S'_1, \ldots, S'_{i-1}\}$, S' would have to cover at least $|S'| - (x_i - 1) \geq |S_i| - (x_i - 1)$ elements uncovered by $(\mathcal{C}^* \setminus \{S_1, \ldots, S_{i-1}\}) \cup \{S'_1, \ldots, S'_{i-1}\}$. But this would mean that after replacing S_i with S', the difference between the number of covered elements would be at most $(x_i - 1)$.

Let $\mathcal{C}^*_2$ denote the set obtained after the above-described ℓ iterations. Since for each i the set $(\mathcal{C}^* \setminus \{S_1, \ldots, S_{i-1}\}) \cup \{S'_1, \ldots, S'_{i-1}\}$ is a subset of $\mathcal{C}^* \cup \mathcal{C}^*_2$, we know that for each i each set from $\mathcal{A} \setminus (\mathcal{C}^* \cup \{S'_1, \ldots, S'_\ell\})$ (there are $|\mathcal{A}| - K$ such sets) must contain at least x_i elements covered by $\mathcal{C}^* \cup \mathcal{C}^*_2$ (there are at most 2OPT such elements). Since each element is contained in at most p sets, we infer that for each i, $x_i(|\mathcal{A}| - K) \leq 2\text{OPT}p$ and,

as a consequence, $x_i \leq \frac{2\text{OPT}p}{|\mathcal{A}|-K} = \frac{2\text{OPT}p(1-\gamma)}{2pK}$. Since $\ell \leq K$, we conclude that $\sum_{i=1}^{\ell} x_i \leq 2\text{OPT}pK\frac{(1-\gamma)}{2pK} = (1-\gamma)\text{OPT}$. That is, after replacing the sets from $\mathcal{C}^*$ that do not appear in $\mathcal{A}$ with sets from $\mathcal{A}$, at most $(1-\gamma)\text{OPT}$ elements fewer are covered. This means that there are K sets in $\mathcal{A}$ that together cover at least γOPT elements. Since the algorithm tries all size-K subsets of $\mathcal{A}$, it finds a solution that covers at least γOPT elements. □

A natural question is whether our analysis of approximation ratio is tight. Below we present a family of parameters γ and instances of MaxCover with upper-bounded frequencies on which our algorithm achieves approximation ratio $(\frac{3}{4}+\frac{1}{4}\gamma)$.

Proposition 4.1. *There is a family $\mathcal{I}$ of pairs (I,γ), where I is an instance of MaxCover with bounded frequencies and γ is a real number, $0<\gamma<1$, such that for each $(I,\gamma)\in\mathcal{I}$, if we use Algorithm 1 to find a γ-approximate solution for I, it outputs an at-most $(\frac{3}{4}+\frac{1}{4}\gamma)\text{OPT}(I)$-approximate one.*

Proof. We describe how to construct pairs (I,γ) from the set $\mathcal{I}$. We let p be the bound of the frequencies of elements in I and we let K be the number of sets that we can use in the solution. We choose p and K to be sufficiently large, and γ to be sufficiently close to 1 (the exact meaning of "sufficiently large" and "sufficiently close to 1" will become clear at the end of the proof; elements of $\mathcal{I}$ differ in the particular choices of p, K, and γ). We require that $\frac{1}{1-\gamma}$ is an integer and that p divides K.

We now proceed with the construction of instance $I=(N,\mathcal{S},K)$ for our choice of p, K, and γ. We set $x=\frac{2pK}{(1-\gamma)}+K$; x is the number of highest-cardinality sets from $\mathcal{S}$ that Algorithm 1 will consider on instance I. By our choice of γ and K, x is an integer and is divisible by p. We form N, the set of elements to be covered, to consist of two disjoint subsets, N_1 and N_2, such that $|N_1|=\binom{x}{p}$ and $|N_2|=\binom{x}{p}\frac{Kp}{x}$. We form the family $\mathcal{S}$ to consist of two subfamilies, $\mathcal{S}_1$ and $\mathcal{S}_2$, defined as follows:

1. There are x subsets in $\mathcal{S}_1$, $\mathcal{S}_1=\{S_1,\ldots,S_x\}$. We form the sets in S_1 so that: (a) sets from $\mathcal{S}_1$ are subsets of N_1, (b) each element from N_1 belongs to exactly p different sets from $\mathcal{S}_1$, and (c) no two elements from N_1 belong to the same p sets from $\mathcal{S}_1$. Specifically, we build sets $(S_1,\ldots,S_m)$ as follows. Let f be some one-to-one mapping between elements in N_1 and p-element subsets of $[x]$. For each $e\in N_1$, e belongs exactly to the sets $S_{i_1},\ldots,S_{i_p}$ such that $f(e)=\{i_1,\ldots,i_p\}$. Note that each set $S_i\in\mathcal{S}_1$ contains exactly $\binom{x-1}{p-1}=\binom{x}{p}\frac{p}{x}$ elements.

2. $\mathcal{S}_2$ contains K sets, each covering exactly $\binom{x}{p}\frac{p}{x}$ different elements from N_2 (and no other elements) so that no two sets from $\mathcal{S}_2$ overlap.

This completes our description of I. It is easy to see that each optimal solution for I covers exactly $K\binom{x}{p}\frac{p}{x}$ elements; each set from $\mathcal{S}$ contains exactly $\binom{x}{p}\frac{p}{x}$ elements and there are (more than) K sets that are pairwise disjoint (for example the K sets in $\mathcal{S}_2$).

Nonetheless, Algorithm 1 is free to choose any x sets from $\mathcal{S}$ to include within $\mathcal{A}$, the collection of sets from which it forms the solution. In particular, it is free to pick the x sets from $\mathcal{S}_1$.[2]

2. We could also ensure that each set in $\mathcal{S}_1$ contained one of $\frac{x}{p}$ additional elements, forcing the algorithm to pick exactly the sets from $\mathcal{S}_1$, but that would obscure the presentation of our argument.

Let us fix some arbitrary collection $\mathcal{S}'$ of K sets from $\mathcal{S}_1$. For each j, $0 \leq j \leq K$, let $h(j)$ be the number of elements from N_1 that belong to exactly j sets in $\mathcal{S}'$. The number of elements covered by $\mathcal{S}'$ is exactly $K\binom{x}{p}\frac{p}{x} - \sum_{j=2}^{K}(j-1)h(j)$. How to compute $h(j)$? Using mapping f, it suffices to count the number of p-element subsets of $[x]$ that contain the indices of exactly j sets from $\mathcal{S}'$. In effect, we have $h(j) = \binom{K}{j}\binom{x-K}{p-j}$. We upper-bound the number of sets covered by $\mathcal{S}'$ with:

$$K\binom{x}{p}\frac{p}{x} - h(2) = K\binom{x}{p}\frac{p}{x} - \binom{K}{2}\binom{x-K}{p-2}.$$

Consequently, Algorithm 1 achieves the following approximation ratio on instance I:

$$\frac{K\binom{x}{p}\frac{p}{x} - \binom{K}{2}\binom{x-K}{p-2}}{K\binom{x}{p}\frac{p}{x}} = 1 - \frac{\binom{K}{2}\binom{x-K}{p-2}}{K\binom{x}{p}\frac{p}{x}} = 1 - \frac{\binom{K}{2}\binom{x-K}{p}\frac{p(p-1)}{(x-K-p+2)(x-K-p+1)}}{K\binom{x}{p}\frac{p}{x}}.$$

Now, if x is large in comparison with p and K (which happens for sufficiently large γ), then $\frac{\binom{x-K}{p}}{\binom{x}{p}} \approx 1$. Also, for sufficiently large x and p (and for $x \gg p, K$) we have $\frac{p}{x-K-p+2} \approx \frac{p}{x}$ and $\frac{p-1}{x-K-p+1} \approx \frac{p}{x}$. Finally, for sufficiently large K we have $\binom{K}{2} \approx \frac{K^2}{2}$. Thus, for large values of γ, K, and p, we can approximate the above ratio with the following expression:

$$\begin{aligned}
1 - \frac{\frac{K^2}{2}\cdot\frac{p^2}{x^2}}{K\frac{p}{x}} &= 1 - \frac{1}{2}\cdot\frac{Kp}{\frac{2pK}{(1-\gamma)} + K} \\
&\approx 1 - \frac{1}{2}\cdot\frac{Kp}{\frac{2pK}{(1-\gamma)}} \\
&= 1 - \frac{1}{4}\cdot(1-\gamma) = \frac{3}{4} + \frac{1}{4}\gamma.
\end{aligned}$$

This completes our argument. □

Let us now restrict the setting to the MaxVertexCover problem and compare our algorithm to that of Marx (2008). Briefly put, the idea behind Marx's algorithm is as follows: Consider vertices in the order of nonincreasing degrees. If the degree of the vertex with the highest degree is large enough, then K vertices with the highest degrees already cover sufficiently many edges to give a desired approximate solution. If the highest degree is not large enough, then there is an exact color-coding-based FPT algorithm that solves the problem optimally. Our algorithm is similar in the sense that we also focus on a group of sets with highest cardinalities (sets' cardinalities in MaxCover correspond to vertex degrees in MaxVertexCover). However, instead of simply picking K largest ones, we make a careful decision as to which exactly to take. Marx's approach works for the case of MaxVertexCover but it seems that it cannot be easily adapted to the case of MaxCover with bounded frequencies (even if the frequencies are bounded by two). The reason is that in the former problem each two sets have at most one element in common (i.e., each two vertices share at most one edge), whereas in the latter one the cardinality of an intersection of two sets can be of an arbitrary size. In effect, it is not easy to accurately estimate the number of elements covered by simply picking *some* sets with highest cardinalities; there can be very

many overlaps among them. An algorithm that focuses on picking sets with highest cardinalities has to minimize the number of such overlaps and our algorithm achieves exactly that by considering all possible K-element groups of large sets from the instance.

Further, our algorithm has a better running time than that of Marx. To achieve approximation ratio γ, the algorithm presented by Marx requires at least time $\Omega((\frac{k^3}{1-\gamma})^{(\frac{k^3}{1-\gamma})})$. For us, the exponential factor in the running time is $\binom{\frac{2pK}{(1-\gamma)}+K}{K}$. On the other hand, we should point out that Marx's algorithm's running time stems mostly from the exact part and the algorithm given there is interesting in its own right.

To conclude the discussion of Algorithm 1, let us consider it in the context of the approval-based Chamberlin–Courant rule. The assumption that elements' frequencies are upper-bounded by p means that each voter can approve of up to p candidates. The algorithm chooses a group of most-often approved candidates and from them chooses a committee such that as many voters as possible approve at least one of its members. Our results show that this committee is very good (though not optimal) and that it can be found efficiently, provided that the committee is small. In practice, however, for $\gamma = 0.9$, the algorithm would be feasible only for $K \approx 4, 5$ and p of similar value. Thus it should be viewed mostly as a theoretical result.

4.2 The MaxCover Problem with Lower-Bounded Frequencies

Let us now move on to the case of MaxCover with lower-bounded frequencies. In this case the standard algorithm (to which we refer as "the greedy algorithm") which in each iteration greedily extends the solution with a set that contains the most yet-uncovered elements, can achieve a better approximation ratio than in the unrestricted case (and our analysis is tight).

Theorem 4.2. *The greedy algorithm is a polynomial-time* $\left(1 - e^{-\max\left(\frac{pK}{m},1\right)}\right)$*-approximation algorithm for the MaxCover problem with frequencies lower-bounded by* p*, on instances with* m *sets where we can pick up to* K *of them.*

Proof. The algorithm clearly runs in polynomial time and so we show it's approximation ratio. Let $I = (N, \mathcal{S}, K)$ be an input instance of MaxCover and let p be an integer such that each element from N belongs to at least p sets from $\mathcal{S}$.

We prove by induction that for each i, $0 \leq i \leq K$, after the i'th iteration of the greedy algorithm's main loop, the number of uncovered elements is at most $n(1-\frac{p}{m})^i$. Naturally, for $i = 0$ the number of uncovered elements is exactly n, the total number of elements. Suppose that the inductive assumption holds for some $(i-1)$, $1 \leq i < K$, and let x be the number of elements still uncovered after the $(i-1)$'th iteration (by the inductive assumption, we have $x \leq n(1-\frac{p}{m})^{i-1}$). Since each element belongs to at least p sets and neither of the sets containing the uncovered elements is yet selected, by the pigeonhole principle there is a not-yet-selected set that contains at least $\lceil x\frac{p}{m} \rceil$ of the uncovered elements. In consequence, the number of elements still uncovered after the i'th iteration is at most:

$$x - x\frac{p}{m} = x\left(1 - \frac{p}{m}\right) \leq n\left(1 - \frac{p}{m}\right)^i.$$

Thus after K iterations the number of uncovered elements is at most:

$$n\left(1 - \frac{p}{m}\right)^K = n\left(1 - \frac{p}{m}\right)^{\frac{m}{p}\cdot\frac{pK}{m}} \leq ne^{-\frac{pK}{m}}.$$

That is, at least $n - ne^{-\frac{pK}{m}}$ elements are covered. Since the number of covered elements in the optimal solution is at most n, the algorithm's approximation ratio is $1 - e^{-\frac{pK}{m}}$.

Naturally, the standard approximation ratio of $(1 - e^{-1})$ of the greedy algorithm still applies and we get approximation guarantee of $1 - e^{-\max(\frac{pK}{m},1)}$. □

The analysis given in Theorem 4.2 is tight. Below we present a family of instances on which the algorithm reaches exactly the promised approximation ratio.

Proposition 4.2. *For each rational α, $\alpha \geq 1$, there is an instance $I(\alpha)$ of MaxCover (with m sets, element frequencies lower-bounded by p, K sets to use, and $\frac{pK}{m} = \alpha$) such that on input $I(\alpha)$, the greedy algorithm achieves approximation ratio no better than $1 - e^{-\frac{pK}{m}}$.*

Proof. Let us fix some α, $\alpha > 1$. We choose integers p, K, and m so that: (a) $p = \frac{\alpha m}{K}$, (b) $m \gg K$ (and, thus, $p \gg K$), and (c) p, m, and K are sufficiently large (the exact meaning of "sufficiently large" will become clear at the end of the proof).

We form the instance $I(\alpha) = (N, \mathcal{S}, K)$ as follows. We let $N = N_1 \cup \cdots \cup N_K$, where $N_1, \ldots, N_K$ are pairwise-disjoint sets, each of cardinality $\binom{m-K}{p-1}$ (thus $|N| = K\binom{m-K}{p-1}$). The family $\mathcal{S}$ consists of two subfamilies, $\mathcal{S}_1$ and $\mathcal{S}_2$:

1. $\mathcal{S}_1$ consists of $m-K$ sets, $S_1, \ldots, S_{m-K}$, constructed as follows. For each i, $1 \leq i \leq K$, let f_i be some one-to-one mapping from N_i to $(p-1)$-element subsets of $[m-K]$. For each i, $1 \leq i \leq K$, and each $e \in N_i$, if $f_i(e) = \{j_1, \ldots, j_{p-1}\}$ then we include e in sets $S_{j_1}, S_{j_2}, \ldots, S_{j_{p-1}}$. Note that for each S_j in $\mathcal{S}_1$, $|S_j| = K\binom{m-K-1}{p-2}$; for each i, $1 \leq i \leq K$, S_j contains $\binom{m-K}{p-2}$ elements from N_i; to see this, it suffices to count how many $(p-1)$-element subsets of $[m-K]$ there are that contain j (the number of ways in which we can complement j with $p-2$ elements).

2. $\mathcal{S}_2 = \{N_1, \ldots, N_K\}$.

Note that, by our construction, each element from N belongs to exactly p sets from $\mathcal{S}$ (namely, $p-1$ from $\mathcal{S}_1$ and one from $\mathcal{S}_2$).

Naturally, the K disjoint sets from $\mathcal{S}_2$ form the optimal solution and cover all the elements. We will now analyze the operation of the greedy algorithm on input $I(\alpha)$.

We claim that the greedy algorithm will select sets from $\mathcal{S}_1$ only. We show this by induction. Fix some ℓ, $1 \leq \ell \leq K$, and suppose that until the beginning of the ℓ'th iteration the algorithm chose sets from $\mathcal{S}_1$ only. This means that, for each i, $1 \leq i \leq K$, each set N_i contains exactly $\binom{m-K-\ell}{p-1}$ uncovered elements. Why is this the case? Assume that the algorithm selected sets $S_{j_1}, \ldots, S_{j_\ell}$. An element $e \in N_i$ is uncovered if and only if $f_i(e) \cap \{j_1, \ldots, j_\ell\} = \emptyset$; $\binom{m-K-\ell}{p-1}$ is the number of $(p-1)$-element subsets of $[m-K]$ that do not contain any members of $\{j_1, \ldots, j_\ell\}$. So, if in the ℓ'th iteration the algorithm chose some set from $\mathcal{S}_2$, it would cover these additional $\binom{m-K-\ell}{p-1}$ elements. On the other hand, if it chose a set from $\mathcal{S}_1$, it would additionally cover Kx elements, where $x = \binom{m-K-\ell}{p-1} - \binom{m-K-\ell-1}{p-1}$. By our choice of p, K and m, we have $pK > m$ and, thus, $K > \frac{m-K}{p-1}$ (we use this fact in the final line of the calculations below). We see that the following holds:

$$Kx = K\left(\binom{m-K-\ell}{p-1} - \binom{m-K-\ell-1}{p-1}\right)$$

$$= K\binom{m-K-\ell-1}{p-2}$$
$$= \frac{K(p-1)}{m-K-\ell}\binom{m-K-\ell}{p-1}$$
$$\geq K\frac{p-1}{m-K}\binom{m-K-\ell}{p-1} > \binom{m-K-\ell}{p-1}.$$

That is, in the ℓ'th iteration the greedy algorithm picks a set from $\mathcal{S}_1$. This proves our claim.

Let us now assess the approximation ratio the greedy algorithm achieves on $I(\alpha)$. By the above reasoning, we know that it leaves $\binom{m-2K}{p-1}$ uncovered elements in each N_i, $1 \leq i \leq K$. Thus the ratio of the uncovered elements to all elements is bounded by the following expression (see the explanation below):

$$\frac{K\binom{m-2K}{p-1}}{K\binom{m-K}{p-1}} = \frac{(m-2K)!(m-p-K+1)!}{(m-K)!(m-p-2K+1)!}$$
$$= \frac{(m-p-K+1)(m-p-K)\cdots(m-p-2K+2)}{(m-K)(m-K-1)\cdots(m-2K+1)}$$
$$\geq \left(\frac{m-2K-p+2}{m-2K+1}\right)^K = \left(1-\frac{p-1}{m-2K+1}\right)^K \approx e^{-\frac{pK}{m}}.$$

The first inequality holds by iterative application of the simple observation that if $1 \leq x \leq y$ then $\frac{x-1}{y-1} \leq \frac{x}{y}$. To obtain the final estimate, we observe that for sufficiently large p and m (where $m \gg K$), we have $\frac{p-1}{m-2K+1} \approx \frac{p}{m} = \frac{\alpha}{K}$. For sufficiently large K, $(1-\frac{\alpha}{K})^K \approx e^{-\alpha} = e^{-\frac{pK}{m}}$ (by the fact that $p = \frac{\alpha m}{K}$). Since the optimal solution covers all the elements, we have that the greedy algorithm on input $I(\alpha)$ achieves approximation ratio no better than $1 - e^{-\frac{pK}{m}}$. □

Theorem 4.2 has some interesting implications. For each α, $0 < \alpha < 1$, let α-MaxCover be a variant of MaxCover where for each instance the ratio $\frac{p}{m}$ is at least α. This problems arises, e.g., if we use the approval-based variant of the Chamberlin–Courant rule with the requirement that each voter must approve at least some constant fraction of the candidates (e.g., 10%). There exists a polynomial-time approximation scheme (PTAS) for this version of the problem.

Theorem 4.3. *For each α, $0 < \alpha \leq 1$, there is a PTAS for α-MaxCover.*

Proof. Fix some α, $0 < \alpha \leq 1$. Let $I = (N, \mathcal{S}, K)$ be an input instance of α-MaxCover and let γ be our desired approximation ratio. We let m be the number of sets in $\mathcal{S}$ and we let p be the lower bound on the element frequencies. By definition, we have $\frac{p}{m} \geq \alpha$. If $K > -\frac{m}{p}\ln(1-\gamma)$ then we can run the greedy algorithm and, by Theorem 4.2, we obtain approximation ratio γ. Otherwise, K is bounded by a constant and enumerating all K-element subsets of $\mathcal{S}$ gives a polynomial-time exact algorithm for the problem. □

The exact complexity of α-MaxCover is quite interesting. Using the greedy algorithm, we can show that it belongs to the second level of Kintala and Fisher's β-hierarchy of limited

nondeterminism (Kintala & Fisher, 1980). (A problem belongs to the class β^2 if it can be solved using at most $O(\log^2 n)$ nondeterministic bits, where n is the size of the input.) In effect, it is unlikely that α-MaxCover is NP-complete.

Theorem 4.4. *For each α, $0 < \alpha < 1$, α-MaxCover is in β^2.*

Proof. Fix some α, $0 < \alpha < 1$. We give a β^2-algorithm for α-MaxCover. Let $I = (N, \mathcal{S}, K, T)$ be an instance of α-MaxCover (recall that T is the number of elements we are required to cover). We let p be the lower bound on elements' frequencies in I, we let $m = |\mathcal{S}|$, and we let $n = |N|$. By definition, we have $\frac{p}{m} \geq \alpha$. W.l.o.g., we assume that $|I| \geq n + m$.

Our algorithm works as follows. If $K > 1/\alpha \ln(n)$ then we run the greedy algorithm and output its solution. Otherwise, we nondeterministically guess K names of the sets from $\mathcal{S}$ and check if these sets cover at least T elements. If so, we accept and otherwise we reject on this computation path.

First, it is clear that the algorithm uses at most $O(\log^2 |I|)$ nondeterministic bits. We execute the nondeterministic part of the algorithm only if $K < 1/\alpha \ln(n) \leq 1/\alpha \ln |I|$ and each set's name requires at most $\log m \leq \log |I|$ bits. Altogether, we use at most $O(\log^2 |I|)$ bits of nondeterminism.

Second, we need to show the correctness of the algorithm. Clearly, if the algorithm uses the nondeterministic part then certainly it finds an optimal solution. Thus consider the case that the algorithm uses the deterministic part, based on the greedy algorithm. In this case we know that $K > 1/\alpha \ln(n)$. Thus, the approximation ratio of the greedy algorithm is greater than: $1 - e^{-\alpha K} > (1 - e^{-\ln n}) = 1 - \frac{1}{n}$. That is, the algorithm returns a solution that covers more than $\mathrm{OPT}(1 - \frac{1}{n})$ elements and, since $\mathrm{OPT} \leq n$ and the number of covered elements is integer, the algorithm must find an optimal solution. □

So far, we were not able to show β^2-hardness of α-MaxCover. We leave establishing the exact complexity of of α-MaxCover as an interesting open problem.

4.3 The MinNonCovered Problem with Upper-Bounded Frequencies

In this section we consider the MinNonCovered problem, that is, a version of MaxCover where the goal is to minimize the number of elements left uncovered. In this case we give a randomized FPT approximation scheme (presented as Algorithm 2).

Intuitively, the idea behind our approach is to extend a simple bounded-search-tree algorithm for SetCover with upper-bounded frequencies to the case of MaxCover. An FPT algorithm for SetCover with frequencies upper-bounded by some constant p could work recursively as follows: If there still is some uncovered element e, then nondeterministically guess one of the at-most-p sets that contain e and recursively solve the smaller problem. The recursion tree would have at most K levels and at most p^K leaves. The same approach does not work directly for MaxCover because we do not know which element e to pick (in SetCover the choice is irrelevant because we have to cover all the elements). However, it turns out that if we choose e randomly then, in expectation, we achieve a good result. It remains an open question whether our algorithm can be efficiently derandomized.

Algorithm 2: Algorithm for the MinNonCovered problem with frequencies upper-bounded by p.

Parameters:
$(N, \mathcal{S}, K)$ — input MinNonCovered instance
p — upper bound on the number of sets each element can belong to
γ — the required approximation ratio of the algorithm
ϵ — the allowed probability of achieving worse than γ approximation ratio

`Search`(s, *partial*):
  if $s = 0$ **then return** *partial* ;
  $e \leftarrow$ randomly select element not-yet covered by *partial*;
  best $\leftarrow \{\}$;
  foreach $S \in \mathcal{S}$ *such that* $e \in S$ **do**
    sol $\leftarrow$ `Search`$((s-1)$, *partial* $\cup \{S\})$;
    if *sol* **is better than** *best* **then** *best* $\leftarrow$ *sol* ;
  return *best*;

`Main()`:
  run `Search`(K, $\{\}$) for $\lceil -\ln \epsilon / (\frac{\gamma-1}{\gamma})^K \rceil$ times;
  return the best solution;

Theorem 4.5. *Fix a positive integer p. For each instance $I = (N, \mathcal{S}, K)$ of MinNonCovered, where each element from N appears in at most p sets in $\mathcal{S}$, for each $\gamma > 1$, and each $\epsilon \in (0, 1)$, Algorithm 2 outputs a γ-approximate solution with probability $(1 - \epsilon)$ in time* $\mathrm{poly}(|I|) \cdot \lceil -\ln \epsilon / (\frac{\gamma-1}{\gamma})^K \rceil \cdot p^K$.

Proof. Let $I = (N, \mathcal{S}, K)$ be an input instance of MinNonCovered and fix some γ, $\gamma > 1$, and ϵ, $0 < \epsilon < 1$. Each element from N appears in at most p sets from $\mathcal{S}$.

By p_s we denote the probability that a single invocation of the function `Search` (from the `Main` function) returns a γ-approximate solution. We will first show that p_s is at least $(\frac{\gamma-1}{\gamma})^K$, and then we will use the standard argument that if we make $\lceil \frac{-\ln \epsilon}{p_s} \rceil$ calls to `Search`, then the best output is a γ-approximate solution with probability $(1 - \epsilon)$.

Let $\mathcal{C}^*$ be some optimal solution for I, let $N^* \subseteq N$ be the set of elements covered by $\mathcal{C}^*$, and let $U^* = N \setminus N^*$ be the set of the remaining, uncovered elements. Consider a single call to `Search` from the "for" loop within the function `Main`. Let Ev denote the event that during such a call, at the beginning of each recursive call, at least a $\frac{\gamma-1}{\gamma}$ fraction of the elements not covered by the constructed solution (i.e., the solution denoted *partial* in the algorithm) belongs to N^*. Note that if the complementary event, denoted $\overline{Ev}$, occurs, then `Search` definitely returns a γ-approximate solution. Why is this the case? Consider some tree of recursive invocations of `Search`, and some invocation of `Search` within this tree. Let X be the number of elements not covered by *partial* at the beginning of this invocation. If at most $\frac{\gamma-1}{\gamma} X$ of the not-covered elements belong to N^*, then—of course—the remaining at least $\frac{1}{\gamma} X$ of them belong to U^*. In other words, then we have $\frac{1}{\gamma} X \leq |U^*|$ and, equivalently, $X \leq \gamma |U^*|$. This means that *partial* already is a γ-approximate solution, and so the solution returned by the current invocation of `Search` will be γ-approximate as well. (Naturally, the same applies to the solution returned at the root of the recursion tree.)

Now, consider the following random process $\mathcal{P}$. (Intuitively, $\mathcal{P}$ models a particular branch of the `Search` recursion tree.) We start from the set N' of all the elements, $N' = N$, and in each of the next K steps we execute the following procedure: We randomly select an element e from N' and if e belongs to N^*, we remove from N' all the elements covered by the first[3] set from $\mathcal{C}^*$ that covers e. Let p_{opt} be the probability that a call to `Search` (within `Main`) finds an optimal solution for I, and let $p_{\text{opt}|Ev}$ be the same probability, but under the condition that Ev takes place. It is easy to see that p_{opt} is greater than or equal to the probability that in each step $\mathcal{P}$ picks an element from N^*. Let p_{hit} be the probability that in each step $\mathcal{P}$ picks an element from N^*, under the condition that at the beginning of every step more than $\frac{(\gamma-1)}{\gamma}$ fraction of the elements in N' belong to N^*. Again, it is easy to see that $p_{\text{opt}|Ev} \geq p_{hit}$. Further, it is immediate to see that $p_{hit} \geq (\frac{\gamma-1}{\gamma})^K$.

Altogether, combining all the above findings, we get that the probability that `RecursiveSearch` returns a γ-approximate solution is at most

$$p_s \geq \mathrm{P}(\overline{Ev}) + \mathrm{P}(Ev)p_{\text{opt}|Ev} \geq p_{\text{opt}|Ev} \geq (\frac{\gamma-1}{\gamma})^K.$$

(That is, either the event Ev does not take place and `Search` definitely returns a γ-approximate solution, or Ev does occur, and then we lower-bound the probability of finding a γ-approximate solution by the probability of finding the optimal one.) To conclude, the probability of finding a γ-approximate solution in one of the $x = \lceil -\ln \epsilon / (\frac{\gamma-1}{\gamma})^K \rceil$ independent invocations of `Search` from `Main` is at least $1 - (1 - (\frac{\gamma-1}{\gamma})^K)^x \geq 1 - e^{\ln \epsilon} = 1 - \epsilon$. Establishing the running time is clear. □

Algorithm 2 is very useful, especially in conjunction with Algorithm 1. The former one has to provide a very good solution if it is possible to cover almost all the elements and the latter one has to provide a very good solution if in every solution many elements must be left uncovered. Given some input instance, we can run both these algorithms and pick the better solution.

Algorithm 2 has very desirable properties from the point of view of the Chamberlin–Courant rule. For small committee sizes K it finds committees that (approximately) minimize the number of voters that do not approve any of the committee members. Further, as opposed to Algorithm 1, its running time is far more manageable. A single execution of function `Search` requires time $O(p^K)$, which is feasible for some realistic values of p and K. Running this function $\lceil -\ln \epsilon / (\frac{\gamma-1}{\gamma})^K \rceil$ times may be challenging, but in practice one can break the algorithm after any number of `Search` executions (this, of course, would deteriorate the approximation guarantee, but the computed committee may still be sufficiently good).

5. Algorithms for the Unrestricted Variant

So far we have focused on the MaxCover problem where element frequencies were either upper- or lower-bounded. Now we consider the completely unrestricted variant of the problem. In this case we give exponential-time approximation schemes that, unfortunately, are not FPT.

3. We assume the sets in $\mathcal{C}^*$ are ordered in some arbitrary way.

Algorithm 3: An approximation algorithm for the unrestricted MaxCover problem.

Parameters:
$(N, \mathcal{S}, K)$ — input MaxCover instance
X — a parameter of the algorithm
$\mathcal{A}(\cdot)$ — an exact algorithm for MaxCover (returns the set of sets to be used in the cover)
$C = \{\}$;
for $i \leftarrow 1$ **to** X **do**
  $Cov \leftarrow \{e \in N : \exists_{S \in C} e \in S\}$;
  $S_{best} \leftarrow \text{argmax}_{S \in \{S_1, \ldots, S_m\} \setminus C} |\{e \in N \setminus Cov : e \in S\}|$;
  $C \leftarrow C \cup \{S_{best}\}$
$uCov \leftarrow N \setminus \{e \in N : \exists_{S \in C} e \in S\}$;
$C' \leftarrow \mathcal{A}(uCov, (K - X), S \setminus C)$;
return $C \cup C'$

The main idea, which is similar to that of Cygan et al. (2009) and of Croce and Paschos (2012), is to solve part of the problem using an exact algorithm and to solve the remaining part using the greedy algorithm (i.e., the algorithm that we focus on in Section 4.2). There are two possible ways in which this idea can be implemented: Either we can first run the exact algorithm and then solve the remaining part of the instance using the greedy algorithm, or the other way round. We consider both approaches, though a variant of the "brute-force-first-then-greedy" approach appears to be superior (at least as long as we do not have exact algorithms that are significantly faster than a brute-force approach). We start with an analysis of Algorithm 3, which first runs the greedy part and then completes it using an exact algorithm.

Theorem 5.1. *Let $\mathcal{A}$ be an exact algorithm for the MaxCover problem with time complexity $f(K, n, m)$. For each instance $I = (N, \mathcal{S}, K)$ of MaxCover and for each X, $0 \leq X \leq K$, Algorithm 3 returns an $\left(1 - \frac{X}{K} e^{-\frac{X}{K}}\right)$-approximate solution for I and runs in time $f(K - X, |N|, |\mathcal{S}|) + \text{poly}(|I|)$.*

Proof. Establishing the running time of the algorithm is immediate and, thus, below we focus on showing the approximation ratio.

Let $I = (N, \mathcal{S}, K)$ be an instance of MaxCover and let X be an integer, $1 \leq X \leq K$. We rename the sets in $\mathcal{S}$ so that $\mathcal{S} = \{S_1, \ldots, S_m\}$ and $S_1, \ldots S_X$ are the consecutive sets selected in the for loop in Algorithm 3 (i.e., in the greedy part of the algorithm). For each i, $1 \leq i \leq m$, let $c_i = |S_i \setminus (S_1 \cup \cdots \cup S_{i-1})|$. Let N_{OPT} denote the set of elements covered by some optimal solution and set $\text{OPT} = |N_{\text{OPT}}|$. Let Cov_i denote the set $S_1 \cup \cdots \cup S_{i-1}$. (That is, Cov_i is the set of elements in the variable Cov in Algorithm 3 right before executing the i'th iteration of the for loop. Of course, $Cov_1 = \emptyset$.) Naturally, for each i, $1 \leq i \leq m$, we have $|Cov_i| = \sum_{j=1}^{i-1} c_i$.

We claim that for each i, $1 \leq i \leq X$, there exist $(K - i)$ sets from $\mathcal{S} \setminus \{S_1, \ldots S_{i-1}\}$ that cover at least $\frac{K-i}{K}$ fraction of the elements from $N_{\text{OPT}} \setminus Cov_{i-1}$. Why is this the case? First, note that there are some K sets from $\mathcal{S} \setminus \{S_1, \ldots S_{i-1}\}$ that cover $N_{\text{OPT}} \setminus Cov_{i-1}$ (it suffices to take the K sets from some optimal solution, if need be, replace those that belong to $\{S_1, \ldots, S_{i-1}\}$ with some arbitrarily chosen ones from $\mathcal{S} \setminus \{S_1, \ldots, S_{i-1}\}$). Let $Q_1, \ldots, Q_K$ be these K sets. Consider some arbitrary assignment of the elements from $N_{\text{OPT}} \setminus Cov_{i-1}$ to

the sets $Q_1, \ldots, Q_K$, such that each element is assigned to exactly one set. Further, consider an ordering of these sets according to the increasing number of assigned elements. If the i'th set in the ordering is assigned at most fraction $\frac{1}{K}$ of the elements, then each of the sets preceding the i'th one in the ordering also is assigned at most fraction $\frac{1}{K}$ of the elements. In consequence, the last $K-i$ sets from the ordering cover at least a fraction $\frac{K-i}{K}$ of the elements. On the other hand, if the i'th set in the order is assigned more than fraction $\frac{1}{K}$ of the elements then the following sets also are and, once again, the last $K-i$ sets cover at least a fraction $\frac{K-i}{K}$ of the elements.

In consequence, we see that for each i, $1 \leq i \leq X$, $c_i \geq \frac{1}{K}(\mathrm{OPT} - \sum_{j=1}^{i-1} c_j)$. The reason is that since there are $K-i$ sets among $\mathcal{S} \setminus \{S_1, \ldots, S_{i-1}\}$ that cover fraction $\frac{K-i}{K}$ of elements from $N_{\mathrm{OPT}} \setminus Cov_i$, at least one of them must cover $\frac{1}{K}(\mathrm{OPT} - |Cov_i|)$ such elements. S_i is chosen as a set that covers the largest number of elements from $N - Cov_i$. It covers c_i elements from $N - Cov_i$, and, thus:

$$c_i \geq \frac{1}{K}(\mathrm{OPT} - |Cov_i|) = \frac{1}{K}(\mathrm{OPT} - \sum_{j=1}^{i-1} c_j).$$

We can now proceed with computing the algorithm's approximation ratio. By the above reasoning, we observe that the solution provided by Algorithm 3 covers at least

$$c = \sum_{i=1}^{X} c_i + \frac{K-X}{K}(\mathrm{OPT} - \sum_{i=1}^{X} c_i) = \frac{X}{K}\sum_{i=1}^{X} c_i + \frac{K-X}{K}\mathrm{OPT}$$

elements. Now, we assess the minimal value of $\sum_{i=1}^{X} c_i$. Minimization of $\sum_{i=1}^{X} c_i$ can be viewed as a linear programming task with the following constraints: for each i, $1 \leq i \leq X$, $c_i \geq \frac{1}{K}(\mathrm{OPT} - \sum_{j=1}^{i-1} c_j)$. Since we have X variables and X constraints, we know that the minimum is achieved when each constraint is satisfied with equality—see, e.g., the textbook of Vazirani (2001). Thus a solution to our linear program consists of values $c_{1,\min}, \ldots, c_{X,\min}$ that, for each i, $1 \leq i \leq X$, satisfy $c_{i,\min} = \frac{1}{K}(\mathrm{OPT} - \sum_{j=1}^{i-1} c_{j,\min})$. By induction, we show that for each i, $1 \leq i \leq X$, $c_{i,\min} = \frac{1}{K}\left(\frac{K-1}{K}\right)^{i-1}\mathrm{OPT}$. Indeed, the claim is true for $i = 1$:

$$c_{1,\min} = \frac{1}{K}\mathrm{OPT}$$

For the inductive step, let us assume that our claim holds for $c_{1,\min}, \ldots, c_{i,\min}$. We show that it also holds for $c_{(i+1),\min}$:

$$\begin{aligned} c_{(i+1),\min} &= \frac{1}{K}\left(\mathrm{OPT} - \sum_{j=1}^{i} c_{j,\min}\right) \\ &= \frac{1}{K}\mathrm{OPT}\left(1 - \frac{1}{K}\sum_{j=1}^{i}\left(\frac{K-1}{K}\right)^{j-1}\right) \\ &= \frac{1}{K}\mathrm{OPT}\left(1 - \frac{1}{K}\cdot\frac{1-\left(\frac{K-1}{K}\right)^{i}}{1-\left(\frac{K-1}{K}\right)}\right) \end{aligned}$$

Algorithm 4: An approximation algorithm for the MaxCover problem.

Parameters:
$(N, \mathcal{S}, K)$ — input MaxCover instance
X — the parameter of the algorithm
$C = \{\}$;
$C_{best} = \{\}$;
foreach $(K - X)$-*element subset* C *of* $\mathcal{S}$ **do**
  for $i \leftarrow (K - X + 1)$ **to** K **do**
    $Cov \leftarrow \{e \in N : \exists_{S \in C} e \in S\}$;
    $S_{best} \leftarrow \text{argmax}_{S \in \{S_1, \ldots, S_m\} \setminus C} \{e \in N \setminus Cov : e \in S\}|$;
    $C \leftarrow C \cup \{S_{best}\}$
  $C_{best} \leftarrow$ better solution among C_{best} and C;
return C_{best}

$$= \frac{1}{K}\text{OPT}\left(1 - \left(1 - \left(\frac{K-1}{K}\right)^i\right)\right) = \frac{1}{K}\text{OPT}\left(\frac{K-1}{K}\right)^i.$$

Thus we can lower-bound the number of elements covered by Algorithm 3 as follows:

$$\begin{aligned}
c &= \frac{X}{K}\sum_{i=1}^{X} c_i + \frac{K-X}{K}\text{OPT} \\
&= \text{OPT}\left(\frac{X}{K^2}\sum_{i=1}^{X}\left(\frac{K-1}{K}\right)^{i-1} + \frac{K-X}{K}\right) \\
&= \text{OPT}\left(\frac{X}{K^2}\cdot\frac{1-\left(\frac{K-1}{K}\right)^X}{1-\left(\frac{K-1}{K}\right)} + \frac{K-X}{K}\right) \\
&= \text{OPT}\left(\frac{X}{K}\left(1-\left(\frac{K-1}{K}\right)^X\right) + \frac{K-X}{K}\right) \\
&\geq \text{OPT}\left(1 - \frac{X}{K}e^{-\frac{X}{K}}\right).
\end{aligned}$$

This completes the proof. □

The idea of the proof of Theorem 5.1 is similar to that used by Cygan et al. (2009) for the problem of weighted set cover. Theorem 5.1 gives a good-quality result provided that we know an optimal algorithm with better time complexity than exhaustive search. Otherwise, we can obtain even better results using Algorithm 4, which first performs brute-force search and then completes it using the greedy algorithm.

Theorem 5.2. *For each instance* $I = (N, \mathcal{S}, K)$ *of MaxCover and each integer* X, $0 \leq X \leq K$, *Algorithm 4 computes an* $\left(1 - \frac{X}{K}e^{-1}\right)$*-approximate solution for* I *in time* $\binom{m}{K-X} + \text{poly}(|I|)$.

Proof. Let $I = (N, \mathcal{S}, K)$ be our input instance and let $\mathcal{C}^*$, $\mathcal{C}^* \subseteq \mathcal{S}$, denote some optimal solution. Let $\mathcal{C}^*_X$ denote a subset of $(K - X)$-elements from $\mathcal{C}^*$ that together cover the

largest number of elements. Thus the sets from $\mathcal{C}^*_X$ cover at least a fraction $\frac{K-X}{K}$ of all the elements covered by the optimal solution. Consider the problem of covering the elements uncovered by $\mathcal{C}^*_X$ with X sets from $(\mathcal{S} \setminus \mathcal{C}^*_X)$. We know that $(\mathcal{C}^* \setminus \mathcal{C}^*_X)$ is an optimal solution for this problem. On the other hand, we also know that the greedy algorithm achieves approximation ratio $(1 - 1/e)$ for the problem (Hochbaum, 1996). Thus, the approximation ratio for the original problem is:

$$\left(\frac{K-X}{K} + \frac{X}{K}\left(1 - \frac{1}{e}\right)\right) = \left(1 - \frac{X}{K}e^{-1}\right).$$

It is immediate to establish the running time of the algorithm and so the proof is complete. □

If we wish to solve MaxVertexCover rather than MaxCover, then in Algorithm 4 we should replace the greedy approximation algorithm with the $\frac{3}{4}$-approximation algorithm of Ageev and Sviridenko (1999).

Corollary 5.1. *There exists an algorithm that given an instance $I = (G, K)$ of MaxVertexCover and an integer X, $1 \leq X \leq K$, outputs a $\left(1 - \frac{X}{4K}\right)$-approximate solution for I in time $\binom{m}{K-X} + \text{poly}(|I|)$, where m is the number of vertices of graph G.*

It is quite evident that as long as algorithm $\mathcal{A}$ used within Algorithm 3 is the simple brute-force algorithm that tries all possible solutions, then Algorithm 4 is superior; in the same time it achieves a better approximation ratio. It turns out that, for the case of MaxVertexCover, Algorithm 4 (in the variant from Corollary 5.1) is also better than the algorithm of Croce and Paschos (2012).[4]

The idea behind the algorithm of Croce and Paschos (2012) for MaxVertexCover is similar to that behind our Algorithm 4. Specifically, given two algorithms for MaxVertexCover, an approximation algorithm $\mathcal{A}_a$ and an exact algorithm $\mathcal{A}_e$, for a given value X it first uses $\mathcal{A}_e$ to find an optimal solution that uses $K - X$ vertices (out of the K vertices that we are allowed to use in the full solution), then it removes these $K - X$ vertices and solves the remaining part of the problem using $\mathcal{A}_a$. Assuming that γ_a is the approximation ratio of the algorithm $\mathcal{A}_a$, this approach results in the approximation ratio equal to $\left(\frac{X}{K} + \gamma_a \left(1 - \frac{X}{K}\right)^2\right)$.

Below we compare Algorithm 4 (version from Corollary 5.1) with the algorithm of Croce and Paschos (2012). As the components $\mathcal{A}_a$ and $\mathcal{A}_e$ we use, respectively, the $\frac{3}{4}$-approximation algorithm of Ageev and Sviridenko (1999) and the brute-force algorithm that tries all possible solutions. The best known exact algorithm for MaxVertexCover is due to Cai (2008) and has the complexity $O(m^{0.792K})$, but this algorithm uses an exponential amount of space. Since exponential space complexity might be much less practical than exponential time complexity, we decided to use the brute-force approach (to the best of our knowledge, there is no better exact algorithm running in polynomial space). We present our comparison in Figure 1. The

4. Algorithm 3 cannot be directly compared to the algorithm of Croce and Paschos (2012) for the following reason. Algorithm 3 uses specifically a greedy algorithm which is the best known approximation algorithm for MaxCover, but which is suboptimal for MaxVertexCover. In contrast, the algorithm of Croce and Paschos (2012) can use, e.g., the $\frac{3}{4}$-approximation algorithm of Ageev and Sviridenko (1999). One could, of course, try to use the algorithm of Ageev and Sviridenko in Algorithm 3, but our analysis does not work for this case.

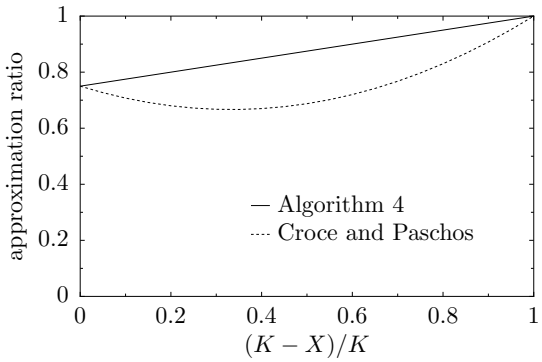

Figure 1: Comparison of the approximation ratios of Algorithm 4 and the algorithm of Croce and Paschos (2012) for MaxVertexCover.

x-axis represents the parameter $\frac{K-X}{K}$, measuring the fraction of the solution obtained using the exact algorithm (for 0 we use the approximation algorithm alone and for 1 we use the exact algorithm alone). On the y-axis we give the guaranteed approximation ratios. In other words, for each point on the x-axis we set the X parameters of the algorithms to be equal, so that their running times are the same, and we compare their approximation guarantees.

We conclude that, as long as we use the brute-force algorithm as the exact one, Algorithm 4 gives considerably better approximation guarantees than that of Croce and Paschos. Figure 1 also exposes one potential weakness of the algorithm of Croce and Paschos. Apparently, for some cases increasing the complexity of the algorithm results in the decrease of its approximation guarantee.

It is quite interesting to understand the reasons behind the differing performance of Algorithm 4 and that of Croce and Paschos. In some sense, the algorithms are very similar. If we use the brute-force algorithm as the exact one in the algorithm of Croce and Paschos, then the main difference is that our algorithm runs the approximation algorithm for each possible solution tried by the brute-force algorithm, and Croce and Paschos's algorithm only runs the approximation algorithm once, for the best partial solution. In effect, our algorithm can exploit situations where it is better when the exact algorithm does not find an optimal solution for the subproblem, but rather leaves ground for the approximation algorithm to do well. Naturally, such strategy is only possible if we have additional knowledge of the structure of the exact algorithm (here, the brute-force algorithm). The result of Croce and Paschos pays the price for being more general and being able to use any combination of the approximation algorithm and the exact algorithm.

6. Conclusions and Future Work

We have studied approximation algorithms for the Chamberlin–Courant voting rule, for the case of approval utilities. We have phrased our technical results in terms of the MaxCover problem, but now we take a step back and consider the results from the point of view of multiwinner voting: As long as the elected committee is small (that is, the value K in the given MaxCover instance is small) and each voter approves of a small, bounded, number of

candidates, the Chamberlin–Courant rule can be very well approximated (the exponential parts of the running times of our algorithms are low in this setting; at least from a theoretical perspective). If we require that each voter approves of some fraction of the candidates, the standard greedy algorithm becomes a polynomial-time approximation scheme. For the completely unrestricted case, we gave an exponential approximation scheme, in which it is possible to seamlessly exchange the quality of the solution for the running time. While the reader may feel queasy about using approximation algorithms in place of an election rule, Skowron et al. (2015a) give several examples where this indeed would be practical and desirable; Faliszewski et al. (2016) also discuss this issue carefully. For example, the use of approximation algorithms can be easily justified for applications reaching beyond the political domain (e.g., for using election rules in recommendation systems or as a tool in resource allocation), where it is important to find as good a solution as possible, but it is not crucial to find exactly the best one. For the high-stake domains, including, e.g., political elections, it is sometimes appealing to view approximation algorithms as full-fledged election rules. Indeed, it is common to consider greedy algorithms for known rules as new election methods. Further, such algorithms often share good properties of the original rules, and sometimes even exhibit new desired characteristics (Elkind et al., 2017).

There are several interesting directions for future research that stem from our work. For example, is it possible to obtain FPT approximation schemes for MaxCover with lower-bounded element frequencies? What is the exact parameterized complexity of MaxCover (with or without lower-bounded frequencies; we have quickly observed its W[2]-hardness and its membership in W[P])? We are also interested in the exact complexity of MaxCover with frequencies lower-bounded by p, for the case where we require the ratio p/m to be at least some given value α, $0 < \alpha < 1$ (this corresponds to the scenario where we require each voter to approve of at least a certain fraction of the candidates)? We have given a PTAS for this variant of the problem (see Theorem 4.3) and have shown its membership in β^2, but we did not manage to prove its completeness for any particular complexity class.

Acknowledgements

We would like to thank the very helpful reviewers who commented on the early version of this paper. The authors were supported by the National Science Centre, Poland, under project 2012/06/M/ST1/00358. During the later parts of the project, Piotr Faliszewski was supported by AGH grant 11.11.230.337 (statutory research) and Piotr Skowron was supported by a Humboldt Research Fellowship for Postdoctoral Researchers.

References

Ageev, A. and Sviridenko, M. (1999). Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *Proceedings of Integer Programming and Combinatorial Optimization*, pages 17–30. Springer Berlin Heidelberg.

Aragones, E., Gilboa, I., and Weiss, A. (2011). Making statements and approval voting. *Theory and decision*, 71(4):461–472.

Aziz, H., Gaspers, S., Gudmundsson, J., Mackenzie, S., Mattei, N., and Walsh, T. (2015). Computational aspects of multi-winner approval voting. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems*, pages 107–115.

Betzler, N., Slinko, A., and Uhlmann, J. (2013). On the computation of fully proportional representation. *Journal of Artificial Intelligence Research*, 47:475–519.

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

Bonnet, E., Paschos, V., and Sikora, F. (2016). Parameterized exact and approximation algorithms for maximum k-set cover and related satisfiability problems. *RAIRO-Theoretical Informatics and Applications*, 50(3):227–240.

Brams, S. J. and Fishburn, P. C. (2010). Going from theory to practice: The mixed success of approval voting. In Laslier, J.-F. and Sanver, M. R., editors, *Handbook on Approval Voting*, pages 19–37. Springer.

Brams, S. J. and Herschbach, D. R. (2001). The science of elections. *Science*, 292(5521):1449.

Cai, L. (2008). Parameterized complexity of cardinality constrained optimization problems. *The Computer Journal*, 51(1):102–121.

Caragiannis, I., Kalaitzis, D., and Markakis, E. (2010). Approximation algorithms and mechanism design for minimax approval voting. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pages 737–742. AAAI Press.

Cesati, M. (2003). The Turing way to parameterized complexity. *Journal of Computer and System Sciences*, 67(4):654–685.

Chamberlin, B. and Courant, P. (1983). Representative deliberations and representative decisions: Proportional representation and the Borda rule. *American Political Science Review*, 77(3):718–733.

Croce, F. and Paschos, V. (2012). Efficient algorithms for the max k-vertex cover problem. *Theoretical Computer Science*, 28(3):295–309.

Cygan, M., Fomin, F., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2015). *Parameterized Algorithms*. Springer.

Cygan, M., Kowalik, Ł., and Wykurz, M. (2009). Exponential-time approximation of weighted set cover. *Information Processing Letters*, 109(16):957–961.

Dey, P., Talmon, N., and Handel, O. (2017). Proportional representation in vote streams. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, pages 15–23.

Downey, R. and Fellows, M. (2013). *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

Elkind, E., Faliszewski, P., Skowron, P., and Slinko, A. (2017). Properties of multiwinner voting rules. *Social Choice and Welfare*, 48(3):599–632.

Faliszewski, P., Sawicki, J., Schaefer, R., and Smołka, M. (2017a). Multiwinner voting in genetic algorithms for solving ill-posed global optimization problems. *IEEE Intelligent Systems*, 32(1):40–48.

Faliszewski, P., Skowron, P., Slinko, A., and Talmon, N. (2017b). Multiwinner rules on paths from k-Borda to Chamberlin–Courant. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. To appear.

Faliszewski, P., Skowron, P., Slinko, A., and Talmon, N. (2017c). Multiwinner voting: A new challenge for social choice theory. In Endriss, U., editor, *Trends in Computational Social Choice*. AI Access. To appear.

Faliszewski, P., Slinko, A., Stahl, K., and Talmon, N. (2016). Achieving fully proportional representation by clustering voters. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems*, pages 296–304.

Feige, U. (1998). A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652.

Flum, J. and Grohe, M. (2006). *Parameterized Complexity Theory*. Springer-Verlag.

Guo, J., Niedermeier, R., and Wernicke, S. (2007). Parameterized complexity of vertex cover variants. *Theoretical Computer Science*, 41(3):501–520.

Hochbaum, D. (1996). Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems. In Hochbaum, D., editor, *Approximation Algorithms for NP-Hard Problems*, pages 94–143. PWS Publishing.

Kilgour, D. (2010). Approval balloting for multi-winner elections. In Laslier, J. and Sanver, R., editors, *Handbook on Approval Voting*, pages 105–124. Springer.

Kintala, C. and Fisher, P. (1980). Refining nondeterminism in relativized polynomial-time bounded computations. *SIAM Journal on Computing*, 9(1):46–53.

Kuo, T., Lin, K. C., and Tsai, M. (2015). Maximizing submodular set function with connectivity constraint: Theory and application to networks. *IEEE/ACM Transactions on Networking*, 23(2):533–546.

Lackner, M. and Skowron, P. (2017). Consistent approval-based multi-winner rules. Technical Report arXiv:1704.02453 [cs.DS], arXiv.org.

Laslier, J.-F. and Van der Straeten, K. (2016). Strategic voting in multi-winners elections with approval balloting: a theory for large electorates. *Social Choice and Welfare*, 47(3):559–587.

LeGrand, R., Markakis, E., and Mehta, A. (2007). Some results on approximating the minimax solution in approval voting. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1193–1195.

Lu, T. and Boutilier, C. (2011). Budgeted social choice: From consensus to personalized decision making. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 280–286.

Lu, T. and Boutilier, C. (2015). Value directed compression of large-scale assignment problems. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 1182–1190.

Marx, D. (2008). Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78.

Monroe, B. (1995). Fully proportional representation. *American Political Science Review*, 89(4):925–940.

Niedermeier, R. (2006). *Invitation to Fixed-Parameter Algorithms*. Oxford University Press.

Oren, J. and Lucier, B. (2014). Online (budgeted) social choice. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1456–1462.

Procaccia, A., Rosenschein, J., and Zohar, A. (2008). On the complexity of achieving proportional representation. *Social Choice and Welfare*, 30(3):353–362.

Sawicki, J., Smołka, M., Łoś, M., Schaefer, R., and Faliszewski, P. (2017). Multiwinner voting in genetic algorithms for solving ill-posed global optimization problems. In *Proceedings of the 20th International Conference on the Applications of Evolutionary Computation*, pages 266–281.

Skowron, P., Faliszewski, P., and Lang, J. (2016). Finding a collective set of items: From proportional multirepresentation to group recommendation. *Artificial Intelligence*, 241:191–216.

Skowron, P., Faliszewski, P., and Slinko, A. (2015a). Achieving fully proportional representation: Approximability results. *Artificial Intelligence*, 222:67–103.

Skowron, P., Lackner, M., Brill, M., Peters, D., and Elkind, E. (2017). Proportional rankings. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. To appear.

Skowron, P., Yu, L., Faliszewski, P., and Elkind, E. (2015b). The complexity of fully proportional representation for single-crossing electorates. *Theoretical Computer Science*, 569:43–57.

Talmon, N. (2017). Structured proportional representation. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*, pages 633–641.

Thiele, T. N. (1895). Om flerfoldsvalg. In *Oversigt over det Kongelige Danske Videnskabernes Selskabs Forhandlinger*, pages 415–441.

Vandin, F., Upfal, E., and Raphael, B. J. (2011). Algorithms for detecting significantly mutated pathways in cancer. *Journal of Computational Biology*, 18(3):507–522.

Vazirani, V. (2001). *Approximation Algorithms*. Springer-Verlag.

Yu, L., Chan, H., and Elkind, E. (2013). Multiwinner elections under preferences that are single-peaked on a tree. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 425–431. AAAI Press.