CLOSEST SUBSTRING PROBLEMS WITH SMALL DISTANCES[*]

DÁNIEL MARX[†]

Abstract.
We study two pattern matching problems that are motivated by applications in computational biology. In the CLOSEST SUBSTRING problem k strings s_1, ..., s_k are given, and the task is to find a string s of length L such that each string s_i has a consecutive substring of length L whose distance is at most d from s. We present two algorithms that aim to be efficient for small fixed values of d and k: for some functions f and g, the algorithms have running time $f(d) \cdot n^{O(\log d)}$ and $g(d,k) \cdot n^{O(\log \log k)}$, respectively. The second algorithm is based on connections with the extremal combinatorics of hypergraphs. The CLOSEST SUBSTRING problem is also investigated from the parameterized complexity point of view. Answering an open question from [13, 14, 20, 21], we show that the problem is W[1]-hard even if both d and k are parameters. It follows as a consequence of this hardness result that our algorithms are optimal in the sense that the exponent of n in the running time cannot be improved to $o(\log d)$ or to $o(\log \log k)$ (modulo some complexity-theoretic assumptions).

CONSENSUS PATTERNS is the variant of the problem where, instead of the requirement that each s_i has a substring that is of distance at most d from s, we have to select the substrings in such a way that the average of these k distances is at most δ. By giving an $f(\delta) \cdot n^9$ time algorithm, we show that the problem is fixed-parameter tractable. This answers an open question from [14].

Key words. closest substring, consensus pattern, parameterized complexity, fixed-parameter tractability, computational complexity

AMS subject classifications. 68W01, 68Q17

1. Introduction.

Computational biology applications provide a steady source of interesting stringology problems. In this paper we investigate two pattern matching problems that received considerable attention lately. Finding similar regions in multiple DNA, RNA, or protein sequences plays an important role in many applications, for example, in locating binding sites [27] and in finding conserved regions in unaligned sequences [24, 28, 34]. This task can be formalized the following way. Given k strings s_1, ..., s_k over an alphabet Σ and an integer L, the task is to find a pattern that appears (possibly with some errors) in each string s_i. More precisely, we have to find a length L string s and a length L substring s'_i of each s_i such that s is "close" to every s'_i. We investigate two variants of the problem that differ in how closeness is defined. In the CLOSEST SUBSTRING problem the goal is to find a string s such that the Hamming-distance of s is at most d from every s'_i. An equally natural optimization goal is to minimize the sum of the distances of s from the substrings s'_i: in the CONSENSUS PATTERNS problem we have to find a string s such that this sum is at most D. An equivalent way of formulating this problem is to require that the average distance is at most $\delta := D/k$.

The CLOSEST SUBSTRING problem is NP-hard even in the special case when $\Sigma = \{0,1\}$ and every string s_i has length L [18]. This means that most probably there are only exponential-time algorithms for the problem. However, an exponential-time algorithm can still be efficient if the exponential dependence is restricted to parameters that are typically small in practice (for example, the size of the alphabet or the maximum number of mismatches that we allow) and the running time depends poly-

[*]A preliminary version of the paper was presented at FOCS 2005. Research partially supported by the Magyary Zoltán Felsőoktatási Közalapítvány and the Hungarian National Research Fund (Grant Number OTKA 67651).

[†]Department of Computer Science and Information Theory, Budapest University of Technology and Economics, Budapest H-1521, Hungary. dmarx@cs.bme.hu

nomially on all the other parameters (such as the lengths of the strings, length of the pattern). Parameterized complexity is the systematic study of problem parameters, with the goal of restricting the exponential increase of the running time to as few parameters of the instance as possible.

1.1. Parameterized complexity. In classical complexity theory, the running time of an algorithm is usually expressed as a function of the input size. Parameterized complexity provides a more refined, two-dimensional analysis of the running time: the goal is to study how the different parameters of the input instance affect the running time. We assume that every input instance has an integer number k associated to it, which will be called the *parameter*. For example, in the case of (the decision version of) Maximum Clique, we can associate to each instance the size of the clique that has to be found. When evaluating an algorithm for a parameterized problem we take into account both the input size n and the parameter k, and we try to express the running time as a function of n and k. The goal is to develop algorithms that run in *uniformly polynomial time*: the running time is $f(k) \cdot n^c$, where c is a constant and f is a (possibly exponential) function depending only on k. We call a parameterized problem *fixed-parameter tractable* if such an algorithm exists. This means that the exponential increase of the running time can be restricted to the parameter k. It turns out that several NP-hard problems are fixed-parameter tractable, for example Minimum Vertex Cover, Longest Path, and Disjoint Triangles. Therefore, for small values of k, the $f(k)$ term is just a constant factor in the running time, and the algorithms for these problems can be efficient even for large values of n. This has to be contrasted with algorithms that have running time such as n^k: in this case the algorithm becomes practically useless for large values of n even if k is as small as 10. Analogously to NP-completeness in classical complexity, the theory of W[1]-hardness can be used to show that a problem is unlikely to be fixed-parameter tractable, which means that for every algorithm the parameter has to appear in the exponent of n. For example, for Maximum Clique and Minimum Dominating Set the running time of the best known algorithms is $n^{\Omega(k)}$, and the W[1]-hardness of these problems tells us that it is unlikely that an algorithm with running time, say, $O(2^k \cdot n)$ can be found.

For a particular problem, there are many possible parameters that can be defined. For example, in the case of the Maximum Clique problem, the maximum degree of the graph, the genus of the graph, or the treewidth of the graph are also natural choices for the parameter. Different applications might suggest different parameters: whether a particular choice of parameter is relevant to an application depends on whether it can be assumed that this parameter is typically "small" in practice. The theory can be extended in a straightforward way to the case when there are more than one parameters: if there are two parameters k_1 and k_2, then the goal is to develop algorithms with running time $f(k_1, k_2) \cdot n^c$. For more details, see Section 2 and [12, 16].

1.2. Previous work on Closest Substring. The NP-completeness of Closest Substring was first shown by Frances and Litman [18] by considering an equivalent problem in coding theory. Li et al. [30] presented a polynomial-time approximation scheme, but the running time of their approximation algorithm is prohibitive. Heuristic approaches for the problem are discussed in [6, 31, 32, 26]; see also the references therein.

Under the standard complexity-theoretic assumptions, the NP-completeness of Closest Substring means that any exact algorithm has to run in exponential time.

However, there can be great qualitative differences between exponential-time algorithms: for example, it can be a crucial difference whether the running time is exponential in the length of the strings or in the number of the strings. This question was investigated in the framework of parameterized complexity by several papers. Formally, the following problem is studied:

> CLOSEST SUBSTRING
>
> *Input:*
> k strings $s_1, \ldots, s_k$ over an alphabet Σ, integers d and L.
>
> *Parameters:*
> k, $|\Sigma|$, d, L
>
> *Task:*
> Find a string s of length L such that for every $1 \le i \le k$, the string s_i has a length L consecutive substring s'_i with $d(s, s'_i) \le d$.

The Hamming-distance of two strings w_1 and w_2 (i.e., the number of positions where they differ) is denoted by $d(w_1, w_2)$. The string s in the solution is called the *center string*. Observe that for a given center string s, it is easy to check in polynomial time whether the substrings s'_i exist: we have to try every length L consecutive substring of the strings s_i. Therefore, the real difficulty of the problem lies in finding the best center string s. We will denote by n the size of the input, which is an upper bound on the total length of the strings. In the following, "substring" will always mean consecutive substring (and not an arbitrary subsequence of the symbols).

The problem can be solved in polynomial time if k, d, or L is fixed to a constant. For every fixed value of L, the problem can be solved in polynomial time by enumerating all the $|\Sigma|^L = O(n^L)$ possible center strings. If d is a fixed constant, then the problem can be solved in polynomial time by making a guess at s'_1 (at most n possibilities) and then trying every center string s that is of distance at most d from s'_1 (at most $(|\Sigma|L)^d = O(n^{2d})$ possibilities). For fixed values of k, the problem can be solved in polynomial time as follows. First we guess the k substrings s'_k (at most n^k possibilities). Now we have to find a center string s that is of distance at most d from each s'_i. This can be done by dynamic programming in $O(n^k)$ time or by applying the linear-time algorithm of Gramm et al. [21] for CLOSEST STRING that is based on integer linear programming. Therefore, for fixed values of L, d, or k, the problem can be solved in polynomial time. However, the algorithms described above are not uniformly polynomial: the exponent of n increases as we consider greater and greater fixed values. The parameterized complexity analysis of the problem can reveal whether it is possible to remove these parameters from the exponent of n, and obtain algorithms with running time such as $f(k) \cdot n^c$.

In [14] and [13] it is shown that the problem is W[1]-hard even if all three of k, d, and L are parameters. Therefore, if the size of the alphabet Σ is not bounded in the input, then we cannot hope for an efficient exact algorithm for the problem. Fortunately, in the computational biology applications the strings are typically DNA or protein sequences, hence the number of different symbols is a small constant (4 or 20). Therefore, we will focus on the case when the size of Σ is a parameter. Restricting $|\Sigma|$ only does not make the problem tractable, since CLOSEST SUBSTRING is NP-hard even if the alphabet is binary. On the other hand, if $|\Sigma|$ and L are both parameters, then the problem becomes fixed-parameter tractable: we can enumerate and check all the $|\Sigma|^L$ possible center strings. However, if the strings are long (which is often

TABLE 1.1

Complexity of CLOSEST SUBSTRING *with different parameterizations. Asterisk denotes the new results of the paper.*

Parameters	$\lvert\Sigma\rvert$ is constant	$\lvert\Sigma\rvert$ is parameter	$\lvert\Sigma\rvert$ is unbounded
d	W[1]-hard (*)	W[1]-hard (*)	W[1]-hard
d, k	W[1]-hard (*)	W[1]-hard (*)	W[1]-hard
k	W[1]-hard	W[1]-hard	W[1]-hard
L	FPT	FPT	W[1]-hard
d, k, L	FPT	FPT	W[1]-hard

the case in practical applications), then it makes much more sense to assume that the number of strings k or the distance constraint d are parameters. In [14] it is shown that CLOSEST SUBSTRING is W[1]-hard with parameter k, even if the alphabet is binary. However, the complexity of the problem with parameter d or with combined parameters d, k remained an open question.

1.3. New results for CLOSEST SUBSTRING**.** We show that the problem is W[1]-hard with combined parameters k and d, even if the alphabet is binary. This resolves an open question asked in [13, 14, 20, 21]. Therefore, even in the binary case, there is no $f(k,d) \cdot n^c$ algorithm for CLOSEST SUBSTRING (unless FPT = W[1]); the exponential increase cannot be restricted to the parameters k and d. This completes the parameterized complexity analysis of CLOSEST SUBSTRING (see Table 1.1; the results of this paper are marked with an asterisk.)

As a first step of the reduction, we introduce a technical problem called SET BALANCING, and prove W[1]-hardness for this problem. This part of the proof contains most of the new combinatorial ideas. The SET BALANCING problem is reduced to CLOSEST SUBSTRING by a reduction very similar to the one presented in [14].

We present two exact algorithms for the CLOSEST SUBSTRING problem. These algorithms can be efficient if d, or both d and k are small (say, $o(\log n)$). The first algorithm runs in $|\Sigma|^{d(\log d+2)} n^{O(\log d)}$ time. Notice that this algorithm is not uniformly polynomial, but only the logarithm of the parameter appears in the exponent of n. Therefore, the algorithm might be efficient for small values of d. The second algorithm has running time $|\Sigma|^d \cdot 2^{kd} \cdot d^{O(d \log\log k)} \cdot n^{O(\log\log k)}$. Here the parameter k appears in the exponent of n, but $\log\log k$ is a very slowly growing function. This algorithm is based on defining certain hypergraphs and enumerating all the places where one hypergraph appears in the other. Using some results from extremal combinatorics, we develop techniques that can speed up the search for hypergraphs. It turns out that if hypergraph H has bounded fractional edge cover number, then we can enumerate in uniformly polynomial time all the places where H appears in some larger hypergraph G. This result might be of independent interest.

Notice that the running times of our two algorithms are incomparable. Assume that $|\Sigma| = 2$. If $d = \log n$ and $k = \sqrt{n}$, then the running time of the first algorithm is $n^{O(\log\log n)} \cdot n^{O(\log\log n)} = n^{O(\log\log n)}$, while the second algorithm needs at least $2^{kd} = 2^{\sqrt{n}\log n} = n^{\sqrt{n}}$ steps, which can be much larger. On the other hand, if $d = k = \log\log n$, then the first algorithm runs in something like $n^{O(\log\log\log n)}$ time, while the running time of the second algorithm is dominated by the $n^{O(\log\log k)}$ factor, which is only $n^{O(\log\log\log\log n)}$.

Our W[1]-hardness proof combined with some recent results on subexponential algorithms shows that the two exact algorithms are in some sense best possible. The exponents are optimal: we show that if there is an $f_1(k, d, |\Sigma|) \cdot n^{o(\log d)}$ or an $f_2(k, d, |\Sigma|) \cdot n^{o(\log\log k)}$ algorithm for Closest Substring, then n-variable 3-Sat can be solved in $2^{o(n)}$ time. It is widely believed that 3-Sat does not have subexponential-time algorithms; this conjecture is called the Exponential Time Hypothesis (cf. [25, 35]).

1.4. Relation to approximability. Li et al. [30] studied the optimization version of Closest Substring, where the task is to find the smallest d that makes the problem feasible. They presented a *polynomial-time approximation scheme (PTAS)* for the problem: for every $\epsilon > 0$, there is an $n^{O(1/\epsilon^4)}$ time algorithm that produces a solution that is at most $(1 + \epsilon)$-times worse than the optimum. This PTAS was improved to $n^{O(\log(1/\epsilon)/\epsilon^2)}$ time by Andoni et al. [2] using an idea from an earlier version of this paper. However, such a PTAS becomes practically useless for large n, even if we ask for an error bound of, say, 20%. As pointed out in [11], there are numerous approximation schemes in the literature where the degree of the algorithm increases very rapidly as we decrease ϵ: having $O(n^{1,000,000})$ or worse for 20% error is not uncommon. Clearly, such approximation schemes do not yield efficient approximation algorithms. Nevertheless, these results show that there are no theoretical limitations on the approximation ratio that can be achieved.

An *efficient PTAS (EPTAS)* is an approximation scheme that produces a $(1+\epsilon)$-approximation in $f(\epsilon) \cdot n^c$ time for some constant c. If $f(\epsilon)$ is e.g., $2^{1/\epsilon}$, then such an approximation scheme can be practical even for $\epsilon = 0.1$ and large n. A standard consequence of W[1]-hardness is that there is no EPTAS for the optimization version of the problem [7, 4]. Hence our hardness result shows that the approximation schemes of [30] and [2] for Closest Substring cannot be improved to an EPTAS.

1.5. Previous work on Consensus Patterns. The Consensus Patterns problem is the same as Closest Substring, but instead of minimizing the maximum distance between the center string and the substrings s'_i, now the goal is to minimize the sum of the distances. Similarly to Closest Substring, the problem is NP-complete and it admits a polynomial-time approximation scheme [29]. Heuristic algorithms for Consensus Patterns and some generalizations are given in e.g., [31, 23, 17, 33, 5].

We will study the decision version of the problem:

Consensus Patterns

Input:
k strings $s_1, \ldots, s_k$ over an alphabet Σ, integers D and L.

Parameters:
k, $|\Sigma|$, D, L

Task:
Find a string s of length L, and a length L consecutive substring s'_i of s_i for every $1 \leq i \leq k$ such that $\sum_{i=1}^{k} d(s, s'_i) \leq D$ holds.

The string s in the solution is called the *median string*. Similarly to Closest Substring, the problem is fixed-parameter tractable if both $|\Sigma|$ and L are parameters: we can enumerate and test every possible median string. Fellows et al. [14] showed

Table 1.2

Complexity of Consensus Patterns *with different parameterizations. Asterisk denotes the new results of the paper.*

Parameters	$\lvert\Sigma\rvert$ is constant	$\lvert\Sigma\rvert$ is parameter	$\lvert\Sigma\rvert$ is unbounded
δ	FPT (*)	FPT (*)	W[1]-hard
D	FPT (*)	FPT (*)	W[1]-hard
k	W[1]-hard	W[1]-hard	W[1]-hard
L	FPT	FPT	W[1]-hard
k, L	FPT	FPT	W[1]-hard
D, k, L	FPT	FPT	W[1]-hard

that their hardness results for Closest Substring can be adapted for Consensus Patterns. Thus the problem is W[1]-hard with combined parameters L, k, D in the unbounded alphabet case, and W[1]-hard with parameter k in the binary alphabet case. The complexity of the problem in the binary alphabet case with parameter D or combined parameters k and D remained open.

Notice that if $D < k$, then the problem can be solved in polynomial time. To see this, observe that $\sum_{i=1}^{k} d(s, s'_i) \le D < k$ is only possible if $d(s, s'_i) = 0$ for at least one i. This means that the median string is a substring of some s_i, thus a solution can be found by trying every length L substring of the input strings. Therefore, we can assume that $D \ge k$ holds in the problem instance. It follows that the complexity of Consensus Patterns is the same with parameter D and with combined parameters k, D.

1.6. New results for Consensus Patterns**.** We define and investigate the new parameter $\delta := D/k$, which is the average error that is allowed between the median string and the substrings s'_i. Parameterization by δ (and not by k) is relevant for applications where we want to find a solution with small average error, but the number of strings is allowed to be large.

By presenting an algorithm with running time $\delta^{O(\delta)} \cdot |\Sigma|^{\delta} \cdot n^9$, we show that Consensus Patterns is fixed-parameter tractable if both $|\Sigma|$ and δ are parameters. The algorithm uses similar hypergraph techniques as the $f(k, d, |\Sigma|) \cdot n^{O(\log \log k)}$ time algorithm for Closest Substring. However, a subtle difference in the combinatorics of the two problems allows us to replace the $O(\log \log k)$ term in the exponent of n with a constant.

Since parameter δ is not greater than parameter D, it follows trivially that the problem is fixed-parameter tractable with combined parameters $|\Sigma|$ and D. This settles another open question from [14]. The results for Consensus Patterns are summarized in Table 1.2, with an asterisk marking the results of the current paper.

1.7. Organization. The paper is organized as follows. Section 2 briefly reviews the most important notions of parameterized complexity. The first algorithm for Closest Substring is presented in Section 3. In Section 4 we discuss techniques for finding one hypergraph in another. In Section 5 we present the second algorithm for Closest Substring. This section introduces a new hypergraph property called *half-covering,* which plays an important role in the algorithm. The algorithm for

Consensus Patterns is presented in Section 6. We define the Set Balancing problem in Section 7 and prove that it is W[1]-hard. In Section 8 the Set Balancing problem is used to show that Closest Substring is W[1]-hard with combined parameters d and k. We conclude the paper with a summary in Section 9.

Algorithm 1 (Section 3) and Algorithm 2 (Sections 4 and 5) for the Closest Substring problem are independent from each other. The algorithm for Consensus Patterns (Section 6) is very similar to the algorithm in Section 5, but it is presented in a self-contained way. The algorithm of Section 6 is also based on the hypergraph techniques developed in Section 4.

The hardness results in Section 7 and 8 are independent from the algorithms; the reductions can be understood without the preceding sections. However, the combinatorics of the reduction in Section 7 has subtle connections with the half-covering property discussed in Section 5. In some sense, Section 5 explains why the reduction in Section 7 has to be done that way.

2. Parameterized complexity.

We follow [16] for the standard definitions of parameterized complexity. Let Σ be a finite alphabet. A decision problem is represented by a set $Q \subseteq \Sigma^*$ of strings over Σ. A *parameterization* of a problem is a polynomial-time computable function $\kappa : \Sigma^* \to \mathbb{N}$. A *parameterized decision problem* is a pair (Q, κ), where $Q \subseteq \Sigma^*$ is an arbitrary decision problem and κ is a parameterization. Intuitively, we can imagine a parameterized problem as a decision problem where each input instance $x \in \Sigma^*$ has a positive integer $\kappa(x)$ associated with it. A parameterized problem (Q, κ) is *fixed-parameter tractable (FPT)* if there is an algorithm that decides whether $x \in Q$ in time $f(\kappa(x)) \cdot |x|^c$ for some constant c and computable function f. An algorithm with such running time is called an *fpt-time algorithm* or simply *fpt-algorithm.*

Many NP-hard problems were investigated in the parameterized complexity literature, with the goal of identifying fixed-parameter tractable problems. There is a powerful toolbox of techniques for designing fpt-algorithms: kernelization, bounded search trees, color coding, well-quasi ordering—just to name some of the more important ones. On the other hand, certain problems resisted every attempt at obtaining fpt-algorithms. Analogously to NP-completeness in classical complexity, the theory of W[1]-hardness can be used to give strong evidence that certain problems are unlikely to be fixed-parameter tractable. We omit the somewhat technical definition of the complexity class W[1], see [12, 16] for details. Here it will be sufficient to know that there are several problems, including Maximum Clique, that were proved to be W[1]-hard. Furthermore, we also expect that there is no $n^{o(k)}$ (or even $f(k) \cdot n^{o(k)}$) algorithm for Maximum Clique: recently it was shown that if there exists an $f(k) \cdot n^{o(k)}$ algorithm for n-vertex Maximum Clique, then n-variable 3-Sat can be solved in time $2^{o(n)}$ (see [8] and [15]).

To prove that a parameterized problem (Q', κ') is W[1]-hard, we have to present a parameterized reduction from a known W[1]-hard problem (Q, κ) to (Q', κ'). A *parameterized reduction* from problem (Q, κ) to problem (Q', κ') is a function that transforms a problem instance x of Q into a problem instance x' of Q' in such a way that

1. $x' \in Q'$ if and only if $x \in Q$,
2. $\kappa'(x)$ can be bounded by a function of $\kappa(x)$, and
3. the transformation can be computed in time $f(\kappa(k)) \cdot |x|^c$ for some constant c and function $f(k)$.

It is easy to see that if there is a parameterized reduction from (Q, κ) to (Q', κ'), and (Q', κ') is fixed-parameter tractable, then it follows that (Q, κ) is fixed-parameter tractable as well. The most important difference between parameterized reductions and classical polynomial-time many-to-one reductions is the second requirement: in most NP-completeness proofs the new parameter is not a function of the old parameter. Therefore, finding parameterized reductions is usually more difficult, and the constructions have somewhat different flavor than classical reductions.

There are many possible parameters that can be defined for a particular problem; different parameters can be relevant in different applications. Usually, the parameter is either some property of the solution we seek (number of vertices, quality of the solution, etc.) or describes some aspect of the input structure (degree/genus/treewidth of the input graph, number of variables/clauses in the input formula, etc.) The complexity of the problem can be different with different parameters. Observe that if parameter k_1 is never greater than parameter k_2, then the problem cannot be easier with parameter k_1 than with k_2: an $f(k_1) \cdot n^c$ time algorithm implies the existence of an $f(k_2) \cdot n^c$ time algorithm.

In some cases we want to investigate the complexity of the problem by considering two or more parameters at the same time, i.e., we assume that both parameter k_1 and parameter k_2 are typically small in applications. The problem is fixed-parameter tractable with combined parameters k_1 and k_2 if there is an algorithm with running time $f(k_1, k_2) \cdot n^{O(1)}$. For a particular problem, we can investigate several different combination of parameters. In general, if we increase the set of parameters, then we cannot make the problem harder: for example, if the problem is fixed-parameter tractable with parameter k_1, then clearly it is fixed-parameter tractable with combined parameters k_1 and k_2.

3. Finding generators.

In this section we present an algorithm for Closest Substring that has running time proportional to roughly $n^{\log d}$. The algorithm is based on the following observation: if all the strings s'_1, …, s'_k agree at some position p in the solution, then we can safely assume that the same symbol appears at the p-th position of the center string s. However, if we look at only a subset of the strings s'_1, …, s'_k, then it is possible that they all agree at some position, but the center string contains a different symbol at this position. We will be interested in sets of strings that do not have this problem:

Definition 3.1. *Let $G = \{g_1, g_2, \ldots, g_\ell\}$ be a set of length L strings. We say that G is a* generator *of the length L string s if whenever every g_i has the same character at some position p, then string s has this character at position p. The* size *of the generator is ℓ, the number of strings in G. The* conflict size *of the generator is the number of those positions where not all of the strings g_i have the same character.*

As we have argued above, it can be assumed that the strings s'_1, …, s'_k of a solution form a generator of the center string s. Furthermore, these strings have a subset of size at most $\log d + 2$ that is also a generator:

Lemma 3.2. *If an instance of* Closest Substring *is solvable, then there is a solution s that has a generator G having the following properties:*

1. *each string in G is a substring of some s_i,*
2. *G has size at most $\log d + 2$,*
3. *the conflict size of G is at most $d(\log d + 2)$.*

Proof. Let s, s'_1, …, s'_k be a solution such that $\sum_{i=1}^{k} d(s, s'_i)$ is minimal. We prove by induction that for every j we can select a subset G_j of j strings from $\{s'_1, \ldots, s'_k\}$ such that there are less than $(d+1)/2^{j-1}$ *bad positions* where the strings in G_j all agree, but this common character is different from the character in s at this position.

The lemma follows from $j = \lceil \log(d+1) \rceil + 1 \leq \log d + 2$: the set G_j has no bad positions, hence it is a generator of s. Furthermore, each string in G_j is at distance at most d from s, thus the conflict size of G_j can be at most $d(\log d + 2)$.

For the case $j = 1$ we can set $G_1 = \{s'_1\}$, since s'_1 differs from s at not more than d positions. Now assume that the statement is true for some j. Let P be the set of bad positions, where the j strings in G_j agree, but they differ from s. We claim that there is some string s'_t in the solution and a subset $P' \subseteq P$ with $|P'| > |P|/2$ such that s'_t differs from all the strings in G_j at every position of P'. If this is true, then we add s'_t to the set G_j to obtain G_{j+1}. Only the positions in $P \setminus P'$ are bad for the set G_{j+1}: for every position p in P', the strings cannot all agree at p, since s'_t do not agree with the other strings at this position. Thus there are at most $|P \setminus P'| < |P|/2 < (d+1)/2^j$ bad positions, completing the induction.

Assume that there is no such string s'_t. In this case we modify the center string s the following way: for every position $p \in P$, let the character at position p be the same as in string s'_1. Denote by s^* the new center string. We show that $d(s^*, s'_i) \leq d(s, s'_i) \leq d$ for every $1 \leq i \leq k$, hence s^* is also a solution. By assumption, every string s'_i in the solution agrees with s'_1 on at least $|P|/2$ positions of P. Therefore, if we replace s with s^*, the distance of s'_i from the center string decreases on at least $|P|/2$ positions, and the distance can increase only on the remaining at most $|P|/2$ positions. Therefore, $d(s^*, s'_i) \leq d(s, s'_i)$ follows. Furthermore, $d(s^*, s'_1) = d(s, s'_1) - |P|$ implies $\sum_{i=1}^{k} d(s^*, s'_i) < \sum_{i=1}^{k} d(s, s'_i)$, which contradicts the minimality of s. ☐

We note that Lemma 3.2 (appearing in an earlier version of this paper) was used by Andoni et al. [2] to improve the running time of the PTAS of Li et al. [30] to $n^{O(\log(1/\epsilon)/\epsilon^2)}$ time.

Our algorithm first creates a set S containing all the length L substrings of s_1, $\ldots$, s_k. For every subset $G \subseteq S$ of $\log d + 2$ strings, we check whether G generates a center string s that solves the problem. Since $|S| \leq n$, there are at most $n^{\log d + 2}$ possibilities to try. By Lemma 3.2 we have to consider only those generators whose conflict size is at most $d(\log d + 2)$, hence at most $|\Sigma|^{d(\log d + 2)}$ possible center strings have to be tested for each G.

THEOREM 3.3. CLOSEST SUBSTRING *can be solved in time* $|\Sigma|^{d(\log d+2)} n^{\log d + O(1)}$.

Proof. The algorithm is presented in pseudocode in Figure 3.1. Let S be the set of all length L substrings in s_1, $\ldots$, s_k, clearly $|S| \leq n$ (recall that n is the total length of the input). If there is a solution s, then Lemma 3.2 ensures that there is a subset $G \subseteq S$ of size at most $\log d + 2$ that generates s. We test every size $\log d + 2$ subset of S whether it can generate a solution. First, by Lemma 3.2 we can restrict our attention to those G where the strings in G agree on all but at most $d(\log d + 2)$ positions. If such a G generates a string s, then the characters of s are determined everywhere except on the conflicting positions of G. Therefore, G can be the generator of at most $|\Sigma|^{d(\log d+2)}$ different strings. We try all the possible combinations of assigning characters on the conflicting positions of G, and we check for each resulting string s whether it is true for every $1 \leq i \leq k$ that there is a substring s'_i of s_i such that $d(s, s'_i) \leq d$. This method will eventually find a solution, if there exits one.

We try $O(n^{\log d+2})$ different subsets G (Line 2), and each G can generate at most $|\Sigma|^{d(\log d+2)}$ different center strings s (Line 4). It can be checked in polynomial time whether a center string s is a solution (Line 5), hence the total running time is $|\Sigma|^{d(\log d+2)} n^{\log d + O(1)}$. ☐

We remark here that the algorithm can be made slightly more efficient: it is suffi-

```
Closest-Substring-1$(k, L, d, (s_1, \ldots, s_k))$
    1. Construct $S$, the set of all length $L$ substrings of the input strings.
    2. for every $G \subseteq S$ with $|G| = \log d + 2$ do
    3.   if the strings in $G$ agree on all but at most $d(\log d + 2)$ positions
    4.     for every string $s$ that is generated by $G$ do
    5.       if $\max_{i=1}^{k} \min_{\{s'_i \text{ is a substring of } s_i\}} d(s, s'_i) \le d$ then
    6.         $s$ is a solution, STOP.
    7. There is no solution, STOP.
```

Figure 3.1. *Algorithm 1 for* Closest Substring

cient to check those generators where the $\log d + 2$ strings come from different strings s_i. However, this observation does not improve the asymptotics of the running time, and we did not want to complicate the notation to accommodate this improvement.

4. Finding hypergraphs. Let us recall some standard definitions concerning hypergraphs. A *hypergraph* $H(V_H, E_H)$ consists of a set of *vertices* V_H and a collection of *edges* E_H, where each edge is a subset of V_H. Let $H(V_H, E_H)$ and $G(V_G, E_G)$ be two hypergraphs. We say that H *appears at* $V' \subseteq V_G$ *as partial hypergraph* if there is a bijection π between the elements of V_H and V' such that for every edge $E \in E_H$ we have that $\pi(E)$ is an edge of G (where the mapping π is extended to the edges the obvious way). For example, if H has the edges $\{1, 2\}$, $\{2, 3\}$, and G has the edges $\{a, b\}$, $\{b, c\}$, $\{c, d\}$, then H appears as a partial hypergraph at $\{a, b, c\}$ and at $\{b, c, d\}$. We say that H *appears at* $V' \subseteq V_G$ *as subhypergraph* if there is such a bijection π where for every $E \in E_H$, there is an edge $E' \in E_G$ with $\pi(E) = E' \cap V'$. For example, let the edges of H be $\{1, 2\}$, $\{2, 3\}$, and let the edges of G be $\{a, c, d\}$, $\{b, c, d\}$. Now H does not appear in G as partial hypergraph, but H appears as subhypergraph at $\{a, b, c\}$ and at $\{a, b, d\}$. If H appears at some $V' \subseteq V_G$ as partial hypergraph, then it appears there as subhypergraph as well.

A *stable set* in $H(V_H, E_H)$ is a subset $S \subseteq V_H$ such that every edge of H contains at most one element from S. The *stable number* $\alpha(H)$ is the size of the largest stable set in H. A *fractional stable set* is an assignment ϕ: $V_H \to [0, 1]$ such that $\sum_{v \in E} \phi(v) \le 1$ for every edge E of H. The *fractional stable number* $\alpha^*(H)$ is the maximum of $\sum_{v \in V_H} \phi(v)$ taken over all fractional stable sets ϕ. The incidence vector of a stable set is a fractional stable set, hence $\alpha^*(H) \ge \alpha(H)$.

An *edge cover* of H is a subset $E' \subseteq E_H$ such that each vertex of V_H is contained in at least one edge of E'. The *edge cover number* $\rho(H)$ is the size of the smallest edge cover in H. (The hypergraphs considered in this paper do not have isolated vertices, hence every hypergraph has an edge cover.) A *fractional edge cover* is an assignment ψ: $E_H \to [0, 1]$ such that $\sum_{E : v \in E} \psi(E) \ge 1$ for every vertex v. The *fractional cover number* $\rho^*(H)$ is the minimum of $\sum_{E \in E_H} \psi(E)$ taken over all fractional edge covers ψ, clearly $\rho^*(H) \le \rho(H)$. It follows from the duality theorem of linear programming that $\alpha^*(H) = \rho^*(H)$ for every hypergraph H with no isolated vertices.

Friedgut and Kahn [19] determined the maximum number of times a hypergraph $H(V_H, E_H)$ can appear as partial hypergraph in a hypergraph G with m edges. That is, we are interested in the maximum number of different subsets $V' \subseteq V_G$ where H can appear in G. A trivial upper bound is $m^{|E_H|}$: if we fix $\pi(E) \in E_G$ for each edge $E \in E_H$, then this uniquely determines $\pi(V_H)$. This trivial bound can be improved to $m^{\rho(H)}$: if edges E_1, E_2, …, $E_{\rho(H)}$ cover every vertex of V_H, then by fixing $\pi(E_1)$,

$\pi(E_2)$, ..., $\pi(E_{\rho(H)})$ the set $\pi(V_H)$ is determined. The result of Friedgut and Kahn says that ρ can be replaced with the (possibly smaller) ρ^*:

THEOREM 4.1. *[19] Let H be a hypergraph with fractional cover number $\rho^*(H)$, and let G be a hypergraph with m edges. There are at most $|V_H|^{|V_H|} \cdot m^{\rho^*(H)}$ different subsets $V' \subseteq V_G$ such that H appears in G at V' as partial hypergraph. Furthermore, for every H and sufficiently large m, there is a hypergraph with m edges where H appears $m^{\rho^*(H)}$ times.*

We remark here that Theorem 4.1 was proved for the special case of graphs in the first published paper of Alon [1].

To appreciate the strength of Theorem 4.1, it is worth pointing out that $\rho^*(H)$ can be much smaller than $\rho(H)$, hence the upper bound can be much stronger than $m^{\rho(H)}$. For example, consider the hypergraph where the vertices correspond to the k-element subsets of $\{1, 2, \dots, n\}$, and edge E_i $(1 \le i \le n)$ contains those vertices that correspond to sets containing i. Now $\rho = n - k + 1$: if we select less than $n - k + 1$ edges, then there is a k-element set that is not covered by the less than $n-k+1$ elements corresponding to the edges. On the other hand, we can construct a fractional edge cover of total weight n/k by assigning weight $1/k$ to each edge. This is a fractional edge cover, since each vertex is contained in exactly k edges. Therefore, the ratio $\rho/\rho^* = (n - k + 1)/(n/k)$ can be arbitrarily large.

Theorem 4.1 does not remain valid if we replace "partial hypergraph" with "subhypergraph." For example, let H contain only one edge $\{1, 2\}$, and let G have one edge E of size ℓ. Now H appears at each of the $\binom{\ell}{2}$ two element subsets of E as subhypergraph. However, if we bound the size of the edges in G, then we can state a subhypergraph analog of Theorem 4.1:

COROLLARY 4.2. *Let H be a hypergraph with fractional cover number $\rho^*(H)$, and let G be a hypergraph with m edges, each of size at most ℓ. Hypergraph H can appear in G as subhypergraph at most $|V_H|^{|V_H|} \cdot \ell^{|V_H|\rho^*(H)} \cdot m^{\rho^*(H)}$ times.*

Proof. Let $G'(V_G, E_{G'})$ be a hypergraph over V_G where $E' \in E_{G'}$ if and only if $|E'| \le |V_H|$ and E' is a subset of some edge $E \in E_G$. An edge of G contributes at most $\ell^{|V_H|}$ edges to G', hence G' has at most $\ell^{|V_H|} \cdot m$ edges. If H appears as subhypergraph at $V' \subseteq V_G$ in G, then H appears as partial hypergraph at V' in G'. By Theorem 4.1, hypergraph H can appear at most $|V_H|^{|V_H|} \cdot \ell^{|V_H|\rho^*(H)} \cdot m^{\rho^*(H)}$ times in G' as partial hypergraph, proving the lemma. ☐

Given hypergraphs $H(V_H, E_H)$ and $G(V_G, E_G)$, we would like to find all the places $V' \subseteq V_G$ in G where H appears as subhypergraph. If there are t such places, then obviously we cannot enumerate all of them in less than t steps. Therefore, our aim is to find an algorithm with running time polynomial in the upper bound $|V_H|^{|V_H|} \cdot \ell^{|V_H|\rho^*(H)} \cdot m^{\rho^*(H)}$ on t given by Corollary 4.2. The proof of Theorem 4.1 is not algorithmic (it is based on Shearer's Lemma [10], which is proved by entropy arguments), hence it does not directly imply an efficient way of enumerating all the places where H appears. However, in Theorem 4.3, we show that there is a very simple algorithm for enumerating all these places. Corollary 4.2 is used to bound the running time of the algorithm. This result might be useful in other applications as well.

THEOREM 4.3. *Let $H(V_H, E_H)$ be a hypergraph with fractional cover number $\rho^*(H)$, and let $G(V_H, E_H)$ be a hypergraph where each edge has size at most ℓ. There is an algorithm that enumerates in time $|V_H|^{O(V_H)} \cdot \ell^{|V_H|\rho^*(H)+1} \cdot |E_G|^{\rho^*(H)+1} \cdot |V_G|^2$ every subset $V' \subseteq V_G$ where H appears in G as subhypergraph.*

Proof. Let $V_H = \{1, 2, \dots, r\}$. For each $1 \le i \le r$, let $H_i(V_i, E_i)$ be the subhypergraph of H induced by $V_i = \{1, 2, \dots, i\}$; that is, if E is an edge of H, then $E \cap V_i$ is an edge of H_i. For each $i = 1, 2, \dots, r$, we find all the places where H_i appears in G as subhypergraph. Since $H = H_r$ this method will solve the problem.

For $i = 1$ the problem is trivial, since V_i has only one vertex. Assume now that we have a list L_i of all the i-element subsets of V_G where H_i appears as subhypergraph. The important observation is that if H_{i+1} appears as subhypergraph at some $(i+1)$-element subset $V' \subseteq V_G$, then V' has an i-element subset $V'' \in L_i$ where H_i appears as subhypergraph. Thus for each set $X \in L_i$, we try all the $|V_G \setminus X|$ different ways of extending X to an $(i+1)$-element set X', and check whether H_{i+1} appears at X' as subhypergraph. This can be checked by trying all the $(i+1)!$ possible bijections π between V_{i+1} and X', and by checking for each edge E of H_{i+1} whether there is an edge E' in G with $\pi(E) = E' \cap X'$.

The structure of the algorithm is presented in Figure 4.1. Let us make a rough estimate of the running time. The loop in Step 2 consists of $|V_H| - 1$ iterations. Notice first that $\rho^*(H_i) \le \rho^*(H)$, since a fractional edge cover of H can be used to obtain a fractional edge cover of H_i. Therefore, by Corollary 4.2, each list L_i has size at most $|V_H|^{|V_H|} \cdot \ell^{|V_H|\rho^*(H)} \cdot |E_G|^{\rho^*(H)}$, which bounds the maximum number of times the loop in Step 3 is iterated. When we determine the list L_{i+1}, we have to check for at most $|L_i| \cdot |V_G|$ different sets X' of size $i+1$ whether H_{i+1} appears at X' as subhypergraph (Step 4). Adding duplicate entries into the list L_{i+1} should be avoided, otherwise we would not have the bound on the size of L_i claimed above. Therefore, in Step 6, we check whether X' is already in L_i. If the list L_i is implemented as a trie structure, then the test in Step 6 can be performed in time $O(|V_H| \cdot |V_G|)$. The trie structure can increase the time required to enumerate the list L_i by a factor of $|V_H|$. Checking one X' requires us to test $(i+1)!$ different bijections π (Step 7). Testing a bijection π means that for each $E \in E_{i+1}$ (Step 8), it has to be checked whether there is a corresponding $E' \in E_G$ (Step 9) such that $E' \cap X' = E$ (Step 10). Hypergraph H has at most $2^{|V_H|}$ edges, hence the loop of Step 8 is iterated at most $2^{|V_H|}$ times. If the edges of G are represented as lists of vertices, then the check in Step 10 can be implemented in $O(\ell)$ time. Adding a new element into the trie structure (Step 13) can be done in $O(|V_H| \cdot |V_G|)$ time.

The dominating part of the running time comes from Steps 7–13, which are repeated $|V_H|^{O(V_H)} \cdot \ell^{|V_H|\rho^*(H)} \cdot |E_G|^{\rho^*(H)} \cdot |V_G|$ times. The loop in Steps 7–12 takes $O(|V_H|! \cdot 2^{|V_H|} \cdot |E_G| \cdot \ell) = |V_H|^{O(V_H)} \cdot |E_G| \cdot \ell$ time, while Step 13 takes $O(|V_H| \cdot |V_G|)$ time. Therefore, the total running time can be bounded by $|V_H|^{O(V_H)} \cdot \ell^{|V_H|\rho^*(H)+1} \cdot |E_G|^{\rho^*(H)+1} \cdot |V_G|^2$. $\square$

We can use a similar technique to find all the places where H appears in G as partial hypergraph. This result is not used in this paper, but might be useful in some other applications.

Corollary 4.4. *Let $H(V_H, E_H)$ be a hypergraph with fractional cover number $\rho^*(H)$, and let $G(V_G, E_G)$ be an arbitrary hypergraph. There is an algorithm that enumerates in time $|V_H|^{O(|V_H|\rho^*(H))} \cdot |E_G|^{\rho^*(H)+1} \cdot |V_G|^2$ all the subsets $V' \subseteq V_G$ where H appears in G as partial hypergraph.*

Proof. We can throw away from G every edge larger than $|V_H|$ without changing the problem. Now Theorem 4.3 can be used to find in time $|V_H|^{O(|V_H|\rho^*(H))} \cdot |E_G|^{\rho^*(H)+1} \cdot |V_G|^2$ the list L of all the subsets $V' \subseteq V_G$ where H appears in G as subhypergraph. If H appears at V' as partial hypergraph, then this is only possible if H appears at V' as subhypergraph. Therefore, the algorithm returns a list that is a superset of the expected result. Let us modify the algorithm of Theorem 4.3 such

```
FIND-SUBHYPERGRAPH(H,G)
 1. $L_1$ := all the places where $H_1$ appears in $G$
 2. for $i := 1$ to $r-1$ do
 3.   for every $X \in L_i$ do
 4.     for every $x \in V_G \setminus X$ do
 5.       $X' := X \cup \{x\}$
 6.       if $X' \notin L_{i+1}$ then
 7.         for every bijection $\pi : V_{i+1} \to X'$ do
 8.           for every $E \in E_{i+1}$ do
 9.             for every $E' \in E_G$ do
10.               if $\pi(E) = E' \cap X'$ then
11.                 Go to Step 8, select next $E$
12.             Go to Step 7, select next $\pi$
13.         Add $X'$ to $L_{i+1}$
14. return $L_r$
```

FIGURE 4.1. *Algorithm for enumerating all the places where hypergraph H appears in G as subhypergraph.*

that in iteration $i = r-1$, Step 10 tests $\pi(E) = E'$ instead of $\pi(E) = E' \cap X'$. This ensures that L_r contains only those positions where H appears as partial hypergraph. □

5. Half-covering and the CLOSEST SUBSTRING problem.

The following hypergraph property plays a crucial role in our second algorithm for the CLOSEST SUBSTRING problem:

DEFINITION 5.1. *We say that a hypergraph $H(V,E)$ has the* half-covering *property if for every non-empty subset $Y \subseteq V$ there is an edge $X \in E$ with $|X \cap Y| > |Y|/2$.*

Theorem 4.3 says that finding a hypergraph H is easy if H has small fractional cover number. In our algorithm for the CLOSEST SUBSTRING problem (described later in this section), we have to find hypergraphs satisfying the half-covering property. The following combinatorial lemma shows that such hypergraphs have small fractional cover number, hence they are easy to find:

LEMMA 5.2. *If $H(V,E)$ is a hypergraph with m edges satisfying the half-covering property, then the fractional cover number ρ^* of H is $O(\log\log m)$.*

Proof. The fractional cover number equals the fractional stable number, thus there is a function ϕ: $V \to [0,1]$ such that $\sum_{v \in X} \phi(v) \le 1$ holds for every edge $X \in E$, and $\sum_{v \in V} \phi(v) = \rho^*$. The lemma is proved by a probabilistic argument: we show that if a random subset $Y \subseteq V$ is selected such that the probability of selecting a vertex v is proportional to $\phi(v)$, then with nonzero probability no edge covers more than half of Y, unless the number of edges is double exponential in ρ^*. The idea is to show that for each edge X, the expected size of Y is ρ^* times the expected size of $Y \cap X$, hence the Chernoff Bound can be used to show that there is only a small probability that X covers more than half of Y. However, the straightforward application of this idea gives only an exponential lower bound on the number of edges. To improve the bound to double exponential, we have to restrict our attention to a suitable subset T of vertices, and scale the probabilities appropriately.

Let v_1, v_2, …, $v_{|V|}$ be an ordering of the vertices by nonincreasing value of $\phi(v_i)$. First we give a bound on the sum of the largest $\phi(v_i)$'s:

PROPOSITION 5.3. $\sum_{j=1}^{i} \phi(v_j) \le -4\log_2 \phi(v_i) + 4$ *holds for every* $1 \le i \le |V|$.

Proof. The proof is by induction on i. Since $\phi(v_1) \leq 1$, the claim is trivial for $i = 1$. For an arbitrary $i > 1$, let $i' \leq i$ be the smallest value such that $\phi(v_{i'}) \leq 2\phi(v_i)$. By assumption, there is an edge X of H that covers more than half of the set $S = \{v_{i'}, \ldots, v_i\}$. Every weight in S is at least $\phi(v_i)$, hence X can cover at most $1/\phi(v_i)$ elements of S. Thus $|S| \leq 2/\phi(v_i)$, and $\sum_{j=i'}^{i} \phi(v_j) \leq 4$ follows from the fact that $\phi(v_j) \leq 2\phi(v_i)$ for $i' \leq j \leq i$. If $i' = 1$, then we are done. Otherwise $\sum_{j=1}^{i'-1} \phi(v_j) \leq -4\log_2 \phi(v_{i'-1}) + 4 < -4(\log_2 \phi(v_i) + 1) + 4$ follows from the induction hypothesis and from $\phi(v_{i'-1}) > 2\phi(v_i)$. Therefore, $\sum_{j=1}^{i} \phi(v_j) = \sum_{j=1}^{i'-1} \phi(v_j) + \sum_{j=i'}^{i} \phi(v_j) \leq -4\log_2 \phi(v_i) + 4$, what we had to show. ☐

In the rest of the proof, we assume that ρ^* is sufficiently large, say $\rho^* \geq 100$. Let i be the largest value such that $\sum_{j=i}^{|V|} \phi(v_j) \geq \rho^*/2$. By the definition of i, $\sum_{j=i+1}^{|V|} \phi(v_j) < \rho^*/2$, hence $\sum_{j=1}^{i} \phi(v_j) \geq \rho^*/2$. Thus by Prop. 5.3, the weight of v_i (and every v_j with $j \geq i$) is at most $2^{-(\rho^*/2-4)/4} \leq 2^{-\rho^*/10}$ (assuming that ρ^* is sufficiently large). Define $T := \{v_i, \ldots, v_{|V|}\}$, and let us select a random subset $Y \subseteq T$: independently each vertex $v_j \in T$ is selected into Y with probability $p(v_j) := 2^{\rho^*/10} \cdot \phi(v_j) \leq 1$. We show that if H does not have $2^{2^{\Omega(\rho^*)}}$ edges, then with nonzero probability every edge of H covers at most half of Y, contradicting the assumption that H satisfies the half-covering property.

The size of Y is the sum of $|T|$ independent 0-1 random variables. The expected value of this sum is $\mu = \sum_{j=i}^{|V|} p(v_j) = 2^{\rho^*/10} \cdot \sum_{j=i}^{|V|} \phi(v_j) \geq 2^{\rho^*/10} \cdot \rho^*/2$. We show that with nonzero probability $|Y| \geq \mu/2$, but $|X \cap Y| \leq \mu/4$ for every edge X. To bound the probability of the bad events, we use the following form of the Chernoff Bound:

Theorem 5.4. *[3] Let $X_1, X_2, \ldots, X_n$ be independent 0-1 random variables with $\Pr[X_i = 1] = p_i$. Denote $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathrm{E}[X]$. Then*

$$\begin{aligned} &\Pr[X \leq (1-\beta)\mu] \leq \exp(-\beta^2\mu/2) \quad \text{for } 0 < \beta \leq 1, \\ &\Pr[X \geq (1+\beta)\mu] \leq \begin{cases} \exp(-\beta^2\mu/3) \text{ for } 0 < \beta \leq 1, \\ \exp(-\beta^2\mu/(2+\beta)) \text{ for } \beta > 1. \end{cases} \end{aligned}$$

Thus by setting $\beta = \frac{1}{2}$, the probability that Y is too small can be bounded as

$$\Pr[|Y| \leq \mu/2] \leq \exp(-\mu/8).$$

For each edge X, the random variable $|X \cap Y|$ is the sum of $|X \cap T|$ independent 0-1 random variables. The expected value of this sum is $\mu_X = \sum_{v \in X \cap T} p(v) = 2^{\rho^*/10} \cdot \sum_{v \in X \cap T} \phi(v) \leq 2^{\rho^*/10} \leq \mu/(\rho^*/2)$, where the first inequality follows from the fact that ϕ is a fractional stable set, hence the total weight X can cover is at most 1. Notice that if ρ^* is sufficiently large, than the expected size of $X \cap Y$ is much smaller than the expected size of Y. We want to bound the probability that $|X \cap Y|$ is at least $\mu/4$. Setting $\beta = (\mu/4)/\mu_X - 1 \geq \rho^*/8 - 1$, the Chernoff Bound gives

$$\begin{aligned} \Pr\bigl[|X \cap Y| \geq \mu/4\bigr] = \Pr\bigl[|X \cap Y| \geq (1+\beta)\mu_X\bigr] \leq \exp(-\beta^2\mu_X/(2+\beta)) \leq \\ \exp(-\beta^2\mu_X/(2\beta)) = \exp(-\mu/8 + \mu_X/2) \leq \exp(-\mu/16). \end{aligned}$$

Here we assumed that ρ^* is sufficiently large that $\beta \geq 2$ (second inequality) and $\mu_X/2 \leq \mu/16$ (third inequality) hold. If H has m edges, then the probability that $|Y| \leq \mu/2$ holds or an edge X covers at least $\mu/4$ vertices of Y is at most

$$(5.1)\quad \exp(-\mu/8) + m \cdot \exp(-\mu/16) \leq (m+1)\exp(-2^{\rho^*/10} \cdot \rho^*/32) \leq m \cdot 2^{-2^{\Omega(\rho^*)}}.$$

If H satisfies the half-covering property, then for every Y there has to be at least one edge that covers more than half of Y. Therefore, the upper bound (5.1) cannot be smaller than 1. This is only possible if m is $2^{2^{\Omega(\rho^*)}}$, and it follows that $\rho^* = O(\log\log m)$, what we had to show. ☐

The following example shows that the bound $O(\log\log m)$ is tight in Lemma 5.2. Fix an integer r, and consider the $2^r - 1$ vertices $V := \{1, 2, \ldots, 2^r - 1\}$. We construct a hypergraph that has not more than 2^{2^r} edges and its fractional cover number is at least $r/2$. Given a finite nonempty set F of positive integers, define $\mathrm{up}(F)$ to be the largest $\lceil(|F|+1)/2\rceil$ elements of this set. For every nonempty subset X of V, add the edge $\mathrm{up}(X)$ to the set system. This results in not more than $2^{2^r-1} - 1$ edges. (There will be lots of parallel edges, but let us not worry about that.) Clearly, the set system satisfies the half-covering property: for every set Y, the set $\mathrm{up}(Y)$ covers more than half of Y.

We claim that the fractional cover number of the hypergraph is at least $r/2$. This can be proved by presenting a fractional stable set of weight $r/2$. Let the weight of v_1 be $1/2$, the weight of v_2 and v_3 be $1/4$, the weight of v_4, v_5, v_6, v_7 be $1/8$, and so on. It is easy to see that the total weight assigned is exactly $r/2$. Furthermore, observe that the weight of v_t is at most $1/(t+1)$ (there is equality if t is of the form $2^k - 1$, otherwise v_t is strictly smaller). To show that this weight assignment is indeed a fractional stable set, suppose that the vertices covered by some edge have total weight more than 1. Let this edge be $\mathrm{up}(X)$ for some subset X of V. Let t be the smallest element in $\mathrm{up}(X)$. Vertex v_t has weight at most $1/(t+1)$, and if t is the smallest element in $\mathrm{up}(X)$, then $\mathrm{up}(X)$ contains at most $t+1$ elements. Therefore, the total weight of the vertices covered by this edge is at most $(t+1)/(t+1) = 1$. We remark that the W[1]-hardness proof in Section 7 is essentially based on this example (see the construction of the enforcer systems in the proof of Prop. 7.2).

Now we are ready to prove the main result of this section:

THEOREM 5.5. CLOSEST SUBSTRING *can be solved in time* $|\Sigma|^d \cdot 2^{kd} \cdot d^{O(d\log\log k)} \cdot n^{O(\log\log k)}$.

Proof. Let us fix the first substring $s'_1 \in s_1$ of the solution. We will repeat the following algorithm for each possible choice of s'_1. Since there are at most n possibilities for choosing s'_1, the running time of the algorithm presented below has to be multiplied by a factor of n, which is dominated by the $n^{O(\log\log k)}$ term.

The center string s can differ on at most d positions from s'_1. Therefore, if we can find the set P of these positions, then the problem can be solved by trying all the $|\Sigma|^{|P|} \le |\Sigma|^d$ possible assignments on the positions in P. We show how to enumerate efficiently all the possible sets P.

We construct a hypergraph G over the vertex set $\{1, \ldots, L\}$. The edges of the hypergraph describe the possible substrings in the solution. If w is a length L substring of some string s_i and the distance of w is at most $2d$ from s'_1, then we add an edge E to G such that $p \in E$ if and only if the p-th character of w differs from the p-th character of s'_1. Clearly, G has at most n edges, each of size at most $2d$. If $(s, s'_1, \ldots, s'_k)$ is a solution, then let H be the partial hypergraph of G that contains only the $k-1$ edges corresponding to the $k-1$ substrings $s'_2, \ldots, s'_k$. (Note that the distance of s'_1 and s'_i is at most $2d$, hence G indeed contains the corresponding edges.) Denote by P the set of at most d positions where s and s'_1 differ. Let H_0 be the subhypergraph of H induced by P: the vertex set of H_0 is P, and for each edge E of H there is an edge $E \cap P$ in H_0. Hypergraph H_0 is subhypergraph of H and H is partial hypergraph of G, thus H_0 appears in G at P as subhypergraph.

We say that a solution is *minimal* if $\sum_{i=1}^{k} d(s, s'_i)$ is minimal. In Prop. 5.6, we show that if the solution $(s, s'_1, \dots, s'_k)$ is minimal, then H_0 has the half-covering property. Therefore, we can enumerate all the possible P's by considering every hypergraph H_0 on at most d vertices that has the half-covering property (there are only a constant number of them), and for each such H_0, we enumerate all the places in G where H_0 appears as subhypergraph. Lemma 5.2 ensures that every H_0 considered has small fractional cover number. By Lemma 4.3, this means that we can enumerate efficiently all the places P where H_0 appears in G as subhypergraph. As discussed above, for each such P we can check whether there is a solution where the center string s differs from s'_1 only on P. By repeating this method for every hypergraph H_0 having the half-covering property, we eventually find a solution, if exists.

PROPOSITION 5.6. *For every minimal solution $(s, s'_1, \dots, s'_k)$, the corresponding hypergraph H_0 has the half-covering property.*

Proof. To see that H_0 has the half-covering property, assume that for some $Y \subseteq P$, every edge of H_0 covers at most half of Y. We show that in this case the solution is not minimal. Modify s such that it is the same as s'_1 on every position of Y, let s^* be the new center string. Clearly, $d(s^*, s'_1) = d(s, s'_1) - |Y|$. Furthermore, we show that this modification does not increase the distance for any i, that is, $d(s^*, s'_i) \le d(s, s'_i)$ for every i. It follows that s^* is also a good center string, contradicting the minimality of the solution.

Let E_i be the edge of H_0 corresponding to the substring s'_i. This means that s'_1 and s'_i differ on $Y \cap E_i$, and they are the same on $Y \setminus E_i$. Therefore, $d(s^*, s'_i) \le d(s, s'_i) + |Y \cap E_i| - |Y \setminus E_i|$. By assumption, E_i can cover at most half of Y, hence $d(s^*, s'_i) \le d(s, s'_i)$, as required. $\square$

The overall algorithm is presented in Figure 5.1. There are at most n different possibilities for the string s'_1 in Step 1. The construction of the hypergraph G in Step 2 takes polynomial time. There are not more than 2^{kd} different hypergraphs on at most d vertices having at most k edges, since there are at most 2^d possibilities for each edge. Therefore, the loop in Step 3 is iterated at most 2^{kd} times. In Step 4 the half-covering property can be tested by complete enumeration: we have to test for at most 2^d different subsets whether there is an edge that covers more than half of it. If H_0 satisfies the half-covering property, then by Lemma 5.2 its fractional cover number is at most $O(\log\log k)$. Therefore, by Theorem 4.3, Step 5 takes $d^{O(d \log\log k)} \cdot n^{O(\log\log k)}$ time. If H_0 appears at P in G as subhypergraph, then in Step 6 we have to try at most $|\Sigma|^d$ possible center strings. Testing each center string can be done in polynomial time (Step 7). Therefore, the total running time is $n \cdot 2^{kd} \cdot d^{O(d \log\log k)} \cdot n^{O(\log\log k)} \cdot |\Sigma|^d$. $\square$

6. Algorithm for CONSENSUS PATTERNS**.** The aim of this section is to show that CONSENSUS PATTERNS is fixed-parameter tractable in the bounded alphabet case if the parameter is $\delta := D/k$, the average distance. The algorithm is very similar to the algorithm of Theorem 5.5. The crucial difference is that here we can obtain a constant bound on the fractional cover number of the small hypergraphs H_0, instead of the weaker $O(\log\log k)$ bound coming from the half-covering property. This means that the exponent of n in the running time of the FIND-SUBHYPERGRAPH algorithm of Theorem 4.3 is a constant and we obtain a uniformly polynomial algorithm.

THEOREM 6.1. CONSENSUS PATTERNS *can be solved in time $\delta^{O(\delta)} \cdot |\Sigma|^{\delta} \cdot n^9$.*

Proof. If $\{s, s'_1, \dots, s'_k\}$ is a solution for an instance of CONSENSUS PATTERNS, then $d(s, s'_i) \le \delta$ for at least one i. Therefore, if there is a solution for the instance, then it can be found by considering every string s_0 that is a length L substring of

```
Closest-Substring-2($k, L, d, (s_1, \dots, s_k)$)
  1. for each substring $s'_1$ of $s_1$ having length $L$ do
  2.   Construct the hypergraph $G$ on $\{1, \dots, L\}$
  3.   for every hypergraph $H_0$ having $\le d$ vertices and $\le k$ edges do
  4.     if $H_0$ has the half-covering property then
  5.       for every place $P$ where $H_0$ appears in $G$ as subhypergraph do
           (Algorithm Find-Subhypergraph of Theorem 4.3)
  6.         for every string $s$ that differs from $s'_1$ only at $P$ do
  7.           if $\max_{i=1}^{k} \min_{\{s'_i \text{ is a substring of } s_i\}} d(s, s'_i) \le d$ then
  8.             $s$ is a solution, STOP.
  9. There is no solution, STOP.
```

Figure 5.1. *Algorithm 2 for* Closest Substring

some s_i, and by checking for each such s_0 whether there is a solution with $d(s, s_0) \le \delta$. Below we describe how to perform this check for a particular s_0. There are at most n possibilities for s_0, hence the total running time is at most n times greater than for a single s_0. We will assume that $\delta \ge 2$, otherwise it is easy to check every possible s with $d(s, s_0) \le \delta$.

We construct a hypergraph G over the vertex set $\{1, \dots, L\}$. If w is a length L substring of some string s_i, then we add an edge E to G such that $p \in E$ if and only if the p-th character of w differs from the p-th character of s_0. (Note that, unlike in the proof of Theorem 5.5, the hypergraph G can have edges larger than $2d$.) If $(s, s'_1, \dots, s'_k)$ is a solution, then let H be the partial hypergraph of G that contains only the k edges corresponding to the k substrings s'_1, …, s'_k.

Let $(s, s_1, \dots, s_k)$ be a minimal solution, that is, $\sum_{i=1}^{k} d(s, s'_i)$ is as small as possible. Denote by P the set of positions where s and s_0 differ. Let H_0 be the subhypergraph of H induced by P: the vertex set of H_0 is P, and for each edge E of H there is an edge $E \cap P$ in H_0. Hypergraph H_0 is subhypergraph of H and H is partial hypergraph of G, thus H_0 appears in G at P as subhypergraph.

We follow the same path as in the proof of Theorem 5.5. It can be shown that the fractional cover number of H_0 is at most 2 (see the proof of Prop. 6.2 below). Therefore, we can find all the possible places P by enumerating every suitable hypergraph H_0, and by using Theorem 4.3 to enumerate all the places where H_0 appears in G as subhypergraph. The problem is that there can be large edges in G, and the algorithm of Theorem 4.3 can be used only if the size of the edges is bounded by the parameter. However, we argue that the same technique works even if the large edges are thrown away from G.

Remove every edge of size greater than 20δ from G (resp., H), let G^* (resp., H^*) be the resulting hypergraph and let H_0^* be the subhypergraph of H^* induced by P. It is clear that H_0^* is a subhypergraph of G^*. Furthermore, the fractional edge cover number of H_0^* can be bounded by a constant:

Proposition 6.2. *For every minimal solution $(s, s'_1, \dots, s'_k)$, the corresponding hypergraph H_0^* has fractional cover number at most* $5/2$.

Proof. We claim that every element of P is covered by at least $k/2$ edges of H_0. Assume that only $k' < k/2$ edges of H_0 cover some $p \in P$. This means that only k' of the strings s'_1, …, s'_k differ from s_0 at position p. Let us change position p of the median string s: let this character be the same as the character at position p of

s_0. Now $d(s, s'_i)$ decreases for $k - k' > k/2$ values of i, and it increases for at most $k' < k/2$ values of i. Therefore, $\sum_{i=1}^{k} d(s, s'_i)$ strictly decreases, contradicting the minimality of s. This shows that every vertex of H_0 is covered by at least $k/2$ of the k edges, hence the fractional cover number of H_0 is at most 2.

From $d(s, s_0) \le \delta$ and $\sum_{i=1}^{k} d(s, s'_i) \le D = k\delta$, it follows that $\sum_{i=1}^{k} d(s_0, s'_i) \le 2k\delta$. Therefore, the total size of the edges in H is at most $2k\delta$, which means that there are at most $2k/20 \le k/10$ edges in H that have size greater than 20δ. Each element of P is covered by at least $k/2$ edges of H_0, hence even if the edges greater than 20δ are thrown away, there remain at least $k/2 - k/10 = 2k/5$ edges in H_0^* to cover each element. Therefore, if we set the weight of each edge to $(5/2) \cdot (1/k)$, then we obtain a fractional edge cover with total weight $5/2$. □

Prop. 6.2 shows that we can find all the possible places P by enumerating every hypergraph H_0^* on δ vertices having fractional cover number at most $5/2$, and then enumerating every place in G^* where H_0^* appears. To reduce the number of hypergraphs H_0^* that has to be considered, we show that it is sufficient to restrict our attention to hypergraphs having $O(\log \delta)$ edges:

Proposition 6.3. *Assume $\delta \ge 2$. If $(s, s'_1, \ldots, s'_k)$ is a minimal solution and H_0^* is the corresponding hypergraph, then it is possible to select $200 \ln \delta$ edges of H_0^* in such a way that if we delete all the other edges, then the resulting hypergraph H_0^{**} has fractional cover number at most* 5.

Proof. Let us select each edge of H_0^* independently with probability $(150 \ln \delta)/k$. The expected number of selected edges is $150 \ln \delta$; from Theorem 5.4 ($\beta = 1/3$) it follows that the probability of selecting more than $200 \ln \delta$ edges is at most $\exp((-150 \ln \delta)/27) < 1/\delta^2$. We have seen in Prop. 6.2 that each vertex of H_0^* is covered by at least $2k/5$ edges, thus the expected number of edges that cover a given vertex of H_0^{**} is at least $60 \ln \delta$. Furthermore, by Theorem 5.4 ($\beta = 1/3$) the probability that a given vertex of H_0^{**} is covered by less than $40 \ln \delta$ edges is at most $\exp(-60 \ln \delta/18) \le 1/\delta^3$. Therefore, with probability at least $1 - 1/\delta^2 - \delta \cdot 1/\delta^3 > 0$, we select not more than $200 \ln \delta$ edges and each vertex is covered by at least $40 \ln \delta$ edges. This means that the fractional cover number of H_0^{**} is at most 5: setting the weight of each edge to $1/(40 \ln \delta)$ gives a fractional edge cover. □

The overall algorithm is presented in Figure 6.1. There are at most n different possibilities for the string s_0 in Step 1. The rest of the algorithm checks whether there is a solution where the median string differs from s_0 on at most δ positions. The construction of the hypergraph G^* can be done in $O(Ln)$ time in Step 2. Since we try to find solutions with $d(s_0, s) \le \delta$, it can be assumed that H_0^{**} has at most δ vertices. There are not more than $2^{O(\delta \ln \delta)} = \delta^{O(\delta)}$ different hypergraphs on at most δ vertices having at most $200 \ln \delta$ edges, since there are at most 2^δ possibilities for each edge. Therefore, the loop in Step 3 is iterated at most $2^{O(\delta \ln \delta)}$ times. The test in Step 4 is trivial. Since the fractional cover number of H_0^{**} is at most 5 and every edge of G^* has size at most 20δ, Step 5 takes $\delta^{O(\delta)} \cdot n^6 L^2$ time. If H_0^{**} appears at P in G^* as subhypergraph, then in Step 6 we have to try at most $|\Sigma|^\delta$ possible median strings. Testing each median string can be done in $O(Ln)$ time (Step 7). Therefore, the total running time is $\delta^{O(\delta)} \cdot |\Sigma|^\delta \cdot n^9$.
□

7. Set Balancing.

In this section we introduce a new problem called Set Balancing. The problem is somewhat technical, it is not motivated by practical applications. However, as we will see it in Section 8, the problem is useful in proving the W[1]-hardness of Closest Substring.

```
CONSENSUS-PATTERNS($k, L, \delta, (s_1, \dots, s_k)$)
  1. for each substring $s_0$ of $s_1, \dots, s_k$ having length $L$ do
  2.   Construct the hypergraph $G^*$ on $\{1, \dots, L\}$
  3.   for every hypergraph $H_0^{**}$ having $\leq \delta$ vertices and $\leq 200 \ln \delta$ edges do
  4.     if every vertex of $H_0^{**}$ is covered by at least 1/5 part of the edges then
  5.       for every place $P$ where $H_0^{**}$ appears in $G^*$ as subhypergraph do
           (Algorithm FIND-SUBHYPERGRAPH of Theorem 4.3)
  6.         for every string $s$ that differs from $s_0$ only at $P$ do
  7.           if $\sum_{i=1}^{k} \min_{\{s'_i \text{ is a substring of } s_i\}} d(s, s'_i) \leq \delta k$ then
  8.             $s$ is solution, STOP
  9. There is no solution, STOP
```

FIGURE 6.1. *Algorithm for* CONSENSUS PATTERNS

SET BALANCING

Input:
A collection of m set systems $\mathscr{S}_i = \{S_{i,1}, \dots, S_{i,|\mathscr{S}_i|}\}$ $(1 \leq i \leq m)$ over the same ground set A and a positive integer d. The size of each set $S_{i,j}$ is at most ℓ, and there is an integer weight $w_{i,j}$ associated to each set $S_{i,j}$.

Parameters:
m, d, ℓ

Task:
Find a set $X \subseteq A$ of size at most d and select a set $S_{i,a_i} \in \mathscr{S}_i$ for every $1 \leq i \leq m$ in such a way that

$$|X \bigtriangleup S_{i,a_i}| \leq w_{i,a_i} \tag{7.1}$$

holds for every $1 \leq i \leq m$.

Here $X \bigtriangleup S_{i,a_i}$ denotes the symmetric difference $(X \setminus S_{i,a_i}) \cup (S_{i,a_i} \setminus X)$. We have to select a set X and a set from each set system in such a way that the balancing requirement (7.1) is satisfied: every selected set is close to X. The weight $w_{i,j}$ of each set $S_{i,j}$ prescribes the maximum distance of X from this set. The smaller the weight, the more restrictive the requirement. The distance is measured by symmetric difference; therefore, adding an element outside $S_{i,j}$ to X can be compensated by adding an element from $S_{i,j}$ to X. If (7.1) holds for some set S_{i,a_i}, then we say that S_{i,a_i} is *balanced*, or X *balances* S_{i,a_i}.

It can be assumed that the weight of each set is at most $\ell + d$, otherwise the requirement would be automatically satisfied for every possible X. If a set appears in multiple set systems, then it can have different weights in the different systems.

In this section we show that SET BALANCING is W[1]-hard even when all of m, d, and ℓ are parameters. It is not very difficult to show that the problem is W[1]-hard if we consider the variant of the problem where the size of X has to be *exactly* d. However, the proof becomes significantly more complicated if we only have the requirement $|X| \leq d$. Intuitively, now the problem is that we have to ensure that the reduction does not construct instances that can be solved by a "small" X, since such an X could be found with an exhaustive search. An easy way to ensure that X is large would be to have a set that can be balanced only by selecting d elements from this set. However, this would reduce the search space to the d-element subsets of this set, and the problem would be easy, since there are at most $\binom{\ell}{d}$ such sets. The main combinatorial challenge in the proof is to ensure that there are no small solutions, but there are lots of possible sets that could form a solution. It should be the combined effect of several set systems that prevent $|X|$ from being small. Furthermore, each set in a set system should be useful for many possible solutions, since the set systems cannot be too large.

THEOREM 7.1. SET BALANCING *is* W[1]*-hard with combined parameters* m*,* d*, and* ℓ*.*

Proof. The proof is by reduction from the MAXIMUM CLIQUE problem. Assume that a graph $G(V, E)$ is given with n vertices and e edges, the task is to find a clique of size t. It can be assumed that $n = 2^{2^C}$ for some integer C: we can ensure that the number of vertices has this form by adding at most $|V|^2$ isolated vertices. Furthermore, we can assume that $C \geq t$ (i.e., $n \geq 2^{2^t}$): if $n < 2^{2^t}$, then MAXIMUM CLIQUE can be solved directly in time $O((2^{2^t})^t \cdot n)$ by enumerating every set of size t.

The ground set A of the constructed instance of SET BALANCING is partitioned into t groups A_0, ..., A_{t-1}. The group A_i is further partitioned into 2^i blocks $A_{i,1}$, ..., $A_{i,2^i}$; the total number of blocks is $2^t - 1$. The block $A_{i,j}$ contains $n^{1/2^i} = 2^{2^{C-i}}$ elements. Set $d := 2^t - 1$. Later we will argue that it is sufficient to restrict our attention to solutions where X contains exactly one element from each block $A_{i,j}$. Let us call such a solution a *standard solution.* We construct the set systems in such a way that there is one-to-one correspondence between the standard solutions and the size t cliques of G. In a standard solution X contains exactly 2^i elements from group A_i, and there are $(n^{1/2^i})^{2^i} = n$ different possibilities for selecting these 2^i elements from the blocks of A_i. Let $X_{i,1}$, ..., $X_{i,n}$ be these n different 2^i-element sets. These n possibilities will correspond to the choice of the i-th vertex of the clique.

The set systems are of two types: the verifier systems and the enforcer systems. The role of the verifier systems is to ensure that every standard solution corresponds to a clique of size t, while the enforcer systems ensure that there are only standard solutions.

For each $0 \leq i_1 < i_2 \leq t-1$ the verifier system $\mathscr{S}_{i_1,i_2}$ ensures that the i_1-th and the i_2-th vertices of the clique are adjacent. The set system $\mathscr{S}_{i_1,i_2}$ contains $2e$ sets of size $2^{i_1} + 2^{i_2}$ each. If vertices u and v are adjacent in G, then $X_{i_1,u} \cup X_{i_2,v}$ is in $\mathscr{S}_{i_1,i_2}$. The weight of every set in $\mathscr{S}_{i_1,i_2}$ is $(2^t - 1) - (2^{i_1} + 2^{i_2})$.

PROPOSITION 7.2. *There is a standard solution if and only if* G *has a size* t *clique.*

Proof. Assume that v_0, ..., v_{t-1} is a clique in G. Let

$$X = \bigcup_{i=0}^{t-1} X_{i,v_i}.$$

The size of X is $\sum_{i=0}^{t-1} 2^i = 2^t - 1$. Select the set $X_{i_1,v_{i_1}} \cup X_{i_2,v_{i_2}}$ from the verifier system $\mathscr{S}_{i_1,i_2}$. This set is balanced by X: it is a size $2^{i_1} + 2^{i_2}$ subset of X having weight $(2^t - 1) - (2^{i_1} + 2^{i_2})$.

To prove the other direction, assume now that there is a standard solution X. In a standard solution, $X \cap A_i$ is a 2^i-element set X_{i,v_i} for some v_i. We claim that these v_i's form a size t clique in G.

Suppose that for some $i_1 < i_2$ vertices v_{i_1} and v_{i_2} are not connected by an edge (including the possibility $v_{i_1} = v_{i_2}$). Consider the set $S \in \mathscr{S}_{i_1,i_2}$ selected in the solution. The size of X is $2^t - 1$ in a standard solution, thus the set X contains at least $2^t - 1 - (2^{i_1} + 2^{i_2})$ elements outside the set S. Therefore, S can be balanced only if all the $2^{i_1} + 2^{i_2}$ elements of S are in X. Assume that the set S selected from $\mathscr{S}_{i_1,i_2}$ is $X_{i_1,u} \cup X_{i_2,v}$. Now $X_{i_1,u} \cup X_{i_2,v} \subseteq X$, which means that $u = v_{i_1}$ and $v = v_{i_2}$. By construction, if $X_{i_1,u} \cup X_{i_2,v}$ is in $\mathscr{S}_{i_1,i_2}$, then u and v are adjacent, hence v_{i_1} and v_{i_2} are indeed neighbors. ☐

The job of the enforcer systems is to ensure that every solution of weight at most $d = 2^t - 1$ is standard. The $2^t - 1$ blocks $A_{i,j}$ are indexed by two indices i and j. In the following, it will be more convenient to index the blocks by a single variable. Let $B_1, \ldots, B_{2^t-1}$ be an ordering of the blocks such that B_1 is the only block of group A_0, the blocks B_2, B_3 are the blocks of A_1, the next four blocks after that are the blocks of A_2, etc.

A naive way of constructing the enforcer set systems would be to have for each block B_i a corresponding set system $\mathscr{S}_i$ that contains $|B_i|$ one-element sets: there is one set of weight $2^t - 2$ for each element of B_i. This ensures that if a solution contains at least one element from every block other than B_i (i.e., it contains at least $2^t - 2$ elements outside B_i), then it has to contain an element of B_i as well (otherwise the symmetric difference is at least $2^t - 1$). The problem with this construction is that every set of $\mathscr{S}_i$ is balanced by the solution $X = \emptyset$, hence such systems cannot ensure that every solution is standard.

There are $2^{2^t-1} - 1$ enforcer set systems: there is a set system $\mathscr{S}_F$ corresponding to each nonempty subset F of $\{1, 2, \ldots, 2^t - 1\}$. The job of $\mathscr{S}_F$ is to rule out the possibility that a solution X contains no elements from the blocks indexed by F, but X contains at least one element from every other block. Clearly, these systems will ensure that no block is empty in a solution, hence every solution of weight $2^t - 1$ is standard. One possible way of constructing the system $\mathscr{S}_F$ is to have one set of size $|F|$ and weight $2^t - 1 - |F|$ for each possible way of selecting one element from each block indexed by F. Clearly, this makes it impossible to have at least one element in each of the $2^t - 1 - |F|$ blocks outside F, but none in F. Now the problem is that the size of $\mathscr{S}_F$ can be too large, in particular when $F = \{1, 2, \ldots, 2^t - 1\}$. We use a somewhat more complicated construction to keep the size of the systems small.

Recall the definition of $\mathrm{up}(F)$ from Section 5: given a finite set F of positive integers, $\mathrm{up}(F)$ is defined to be the largest $\lceil (|F| + 1)/2 \rceil$ elements of this set. The enforcer system corresponding to F is defined as

$$\mathscr{S}_F = \prod_{p \in \mathrm{up}(F)} B_p. \tag{7.2}$$

That is, we consider the blocks indexed by the upper half of F, and put into $\mathscr{S}_F$ all the possible combinations of selecting one element from each block. Let the weight of each set in $\mathscr{S}_F$ be $2^t - 1 - |\mathrm{up}(F)|$. Notice that it is possible that $\mathrm{up}(F_1) = \mathrm{up}(F_2)$ for some $F_1 \neq F_2$, which means that for such F_1 and F_2 the systems $\mathscr{S}_{F_1}$ and $\mathscr{S}_{F_2}$ are in fact the same. However, we do not care about that.

We have to verify that these set systems are not too large, i.e., they can be constructed in uniformly polynomial time:

Proposition 7.3. *For every nonempty $F \subseteq \{1, 2, \ldots, 2^t - 1\}$, the enforcer system $\mathscr{S}_F$ contains at most n^2 sets.*

Proof. Let x be the smallest element of $\mathrm{up}(F)$, assume that $2^p \leq x < 2^{p+1}$ for some integer p. There is one block of size n, there are 2 blocks of size $n^{1/2}$, ..., there are 2^i blocks of size $n^{1/2^i}$, hence the size of B_{2^p} is $n^{1/2^p}$. The size of the blocks are decreasing, thus all the sets in the product (7.2) are of size at most $n^{1/2^p}$. If the smallest element of $\text{up}(F)$ is x, then it can contain at most $x+1$ elements. This means that we take the direct product of at most $x+1$ sets of size at most $n^{1/2^p}$ each. Therefore, the total number of sets in $\mathscr{S}_F$ is at most $(n^{1/2^p})^{x+1} \le (n^{1/2^p})^{2^{p+1}} = n^2$. ▯

The following proposition completes the proof of the first direction: if the solution is standard, then we can select a set from each enforcer system. Together with Prop. 7.2, it follows that if there is a clique of size t, then there is a (standard) solution for the constructed instance of Set Balancing.

Proposition 7.4. *If X is a standard solution, then each $\mathscr{S}_F$ contains a set that is balanced by X.*

Proof. For the enforcer system $\mathscr{S}_F$, let us select the set

$$S_F = X \cap \bigcup_{p \in \text{up}(F)} B_p.$$

That is, S_F contains those elements of X that belong to the blocks indexed by $\text{up}(F)$. The set S_F is a size $|\text{up}(F)|$ subset of X. Therefore, $|X \triangle S_F| = 2^t - 1 - |\text{up}(F)|$, which is exactly the weight of the selected set. Thus S_F is balanced. ▯

On the other hand, if there is a solution for the constructed instance of Set Balancing with $|X| \le d = 2^t - 1$, then this solution has to be standard, and by Prop. 7.2 there is a clique of size t in G. This completes the proof of the second direction.

Proposition 7.5. *If $|X| \le 2^t - 1$, then X contains exactly one element from each block.*

Proof. Assume first that X does not contain elements from some of the blocks. Let F contain the indices of those blocks that are disjoint from X. This means that X contains at least one element from each block not in F, hence $|X| \ge 2^t - 1 - |F|$. Assume that some set S is selected from $\mathscr{S}_F$ in the solution. This set contains elements only from blocks indexed by $\text{up}(F) \subseteq F$, hence S is disjoint from X. Thus $|X \triangle S| = |X| + |S| \ge 2^t - 1 - |F| + |\text{up}(F)| > 2^t - 1 - |\text{up}(F)|$, which means that S is not balanced (here we used $|F| - |\text{up}(F)| < |\text{up}(F)|$). Therefore, each block contains at least one element of X. Since there are $2^t - 1$ blocks, this is only possible if each block contains exactly one element of X. ▯

The distance $d = 2^t - 1$ is a function of the original parameter t. The number m of the constructed set systems is $\binom{t}{2} + 2^{2^t-1} - 1$, which is also a function of t. Each set in the constructed systems has size at most $\ell := 2^t - 1$. We have seen that the size of each set system is polynomial in n, hence the reduction is a correct parameterized reduction. ▯

8. Hardness of Closest Substring**.** In this section we show that Closest Substring is W[1]-hard with combined parameters k and d. The reduction is very similar to the reduction presented in [14], where it is proved that problem is W[1]-hard with parameter k only. As in that reduction, the main technical trick is that each string s_i is divided into blocks and we ensure that the string s_i' is one of these blocks in every solution. However, here the reduction is from Set Balancing, and not from Maximum Clique. This allows us to construct an instance of Closest Substring where the distance parameter d is bounded by a constant.

Theorem 8.1. Closest Substring *is* W[1]*-hard with parameters d and k, even if $\Sigma = \{0, 1\}$.*

Proof. The reduction is from the Set Balancing problem, whose W[1]-hardness was shown in Section 7. Assume that m set systems $\mathscr{S}_i = \{S_{i,1}, \ldots, S_{i,|\mathscr{S}_i|}\}$ and an integer d are given. Let $0 \le w_{i,j} \le d + \ell$ be the weight of $S_{i,j}$ in $\mathscr{S}_i$, and assume that each set has size at most ℓ. We construct an instance of Closest Substring with distance parameter $d' := d + \ell$ where $d' + 1$ strings $s_{i,1}$, $s_{i,2}$, …, $s_{i,d'+1}$ correspond to each set system $\mathscr{S}_i$, and there is one additional string s_0 called the *template string.* Thus there are $k := (d' + 1)m + 1$ strings in total.

Set $L := 6d' + 3d'(3d' + 1) + |A| + d' - d + 2d'm(d' + 1)$, where A is the common ground set of the set systems. The template string s_0 has length L, hence $s'_0 = s_0$ in every solution. The string $s_{i,j}$ is the concatenation of *blocks* $B_{i,j,1}$, …, $B_{i,j,|\mathscr{S}_i|}$ of the same length L, each block corresponds to a set in $\mathscr{S}_i$. We will ensure that in a solution the substring $s'_{i,j}$ is one complete block from $s_{i,j}$. Therefore, selecting $s'_{i,j}$ from $s_{i,j}$ in the constructed Closest Substring instance plays the same role as selecting a set S_i from $\mathscr{S}_i$ in Set Balancing.

Each block $B_{i,j,k}$ of the string $S_{i,j}$ is the concatenation of four parts: the front tag, the core, the complete tag, and the back tag. The *front tag* is the same in every block: $1^{3d'}(10^{3d'})^{3d'}1^{3d'}$. The *core* corresponds to the ground set A in the Set Balancing problem. The length of the core is $|A|$, and the p-th character of the core in block $B_{i,j,k}$ is 1 if and only if the set $S_{i,k} \in \mathscr{S}_i$ contains the p-th element of A. The *complete tag* is $1^{d'-d}$ in every block. The *back tag* is the concatenation of $m(d' + 1)$ segments $C_{i,j}$ $(1 \le i \le m,\ 1 \le j \le d' + 1)$ (the order in which these segments are concatenated will not be important). The length of each segment is $2d'$. In block $B_{i,j,k}$ of string $s_{i,j}$ the back tag contains 1's only in segment $C_{i,j}$: there is 1 on the first $d' - w_{i,k} \ge 0$ positions of $C_{i,j}$, the rest of $C_{i,j}$ is 0. This completes the description of the strings $s_{i,j}$. Notice that the blocks $B_{i,j_1,k}$ and $B_{i,j_2,k}$ differ only in the back tag. The length L template string s_0 is similar to the blocks defined above: it has the same front tag as all the other blocks, but its core, complete tag, and back tag contain only 0's.

The first direction of the proof is shown in the following proposition:

Proposition 8.2. *If the* Set Balancing *instance has a solution, then the constructed instance of* Closest Substring *also has a solution.*

Proof. Let $X \subseteq A$ and $S_{1,a_1} \in \mathscr{S}_1$, …, $S_{m,a_m} \in \mathscr{S}_m$ be a solution of Set Balancing. Let the center string s be the concatenation of the front tag, the incidence vector of X, the string $1^{d'-d}$, and the string $0^{2d'm(d'+1)}$. The distance of s and s_0 is $|X| + d' - d \le d'$: the distance is $|X|$ on the core and $d' - d$ on the complete tag. Furthermore, we claim that the block B_{i,j,a_i} in string $s_{i,j}$ is at distance at most d' from s. If we can show this, then it follows that Closest Substring has a solution.

The front tag of B_{i,j,a_i} is the same as the front tag of s. In the core the distance is the symmetric difference of X and S_{i,a_i}. The complete tag is the same in s and B_{i,j,a_i}. The back tag of s is all 0, while the back tag of B_{i,j,a_i} contains $d' - w_{i,k}$ characters 1 (in the segment $C_{i,j}$). Therefore,

$$d(s, B_{i,j,a_i}) = |X \bigtriangleup S_{i,a_i}| + d' - w_{i,k} \le d',$$

where the inequality follows from the fact that X balances the set S_{i,a_i}, that is, $|X \bigtriangleup S_{i,a_i}| \le w_{i,k}$. □

To prove the reverse direction, first we show that each substring $s'_{i,j}$ has to be a complete block of the string $s_{i,j}$. By the triangle inequality, $d(s_0, s'_{i,j}) = d(s'_0, s'_{i,j}) \le d(s'_0, s) + d(s, s'_{i,j}) \le 2d'$ has to hold in every solution. We show that $d(s_0, s'_{i,j}) \le 2d'$ implies that $s'_{i,j}$ is a complete block:

PROPOSITION 8.3. *If $d(s_0, s'_{i,j}) \leq 2d'$ for some substring $s'_{i,j}$ of $s_{i,j}$, then $s'_{i,j}$ is the block $B_{i,j,b}$ for some b.*

Proof. Assume that $s'_{i,j}$ starts on the p-th character of some block $B_{i,j,b}$. This means that $s'_{i,j}$ contains the last $L-p+1$ characters from $B_{i,j,b}$ and the first $p-1$ characters from the next block $B_{i,j,b+1}$. We show that if $p \neq 1$, then $d(s_0, s'_{i,j}) > 2d'$. Denote by $f = 6d' + 3d'(3d'+1)$ the length of the front tag. Assume first that $3d' < p \leq L - f$. In this case the first $3d'$ characters of $B_{i,j,b+1}$ (all of them are 1's) are aligned with characters $L+1-p, \ldots, L+3d'-p$ of s_0 (all of them are 0's), hence $d(s_0, s'_{i,j}) > 2d'$ follows. Assume now that $L - f < p \leq L - 3d'$, a similar argument shows that the last $3d'$ characters in the front tag of $B_{i,j,b+1}$ causes $3d'$ mismatches. If $1 < p \leq 3d'$, then the 1's in the $(10^{3d'})^{3d'}$ part of the front tag are not aligned, which increases the difference to more than $2d'$. The same thing happens if $L - 3d' < p \leq L$ holds. This concludes the proof that $p = 1$, that is, the string $s'_{i,j}$ is one complete block $B_{i,j,b}$. ☐

Since the template string and each block begins with the front tag, it cannot hurt if the center string also begins with the front tag:

PROPOSITION 8.4. *If there is a solution for the constructed instance of* CLOSEST SUBSTRING, *then there is such a solution where the front tag of the center string s is the same as the front tag of s_0.*

It can be assumed that the back tag of the center string s contains only 0's:

PROPOSITION 8.5. *If there is a solution for the constructed instance of* CLOSEST SUBSTRING, *then there is such a solution where the back tag of the center string s contains only* 0*'s.*

Proof. Let s be the center string of a solution. Since the back tag of s_0 contains only 0's, in the back tag of s at most d' characters can be 1. This means that with the exception of at most d' segments, the segments of the back tag contain only 0's. Thus for every $1 \leq i \leq m$, there is a $1 \leq x_i \leq d'+1$ such that segment C_{i,x_i} of the back tag of s contains only 0's. Let s^* be the same as s but with the back tag set to 0's. It is clear that $d(s^*, s_0) \leq d(s, s_0) \leq d'$: the back tag of s_0 is empty, hence setting the back tag to 0 cannot increase the distance.

We claim that a block can be selected from each string $s_{i,j}$ in such a way that the distance of each selected block is at most d' from s^*. For the string s_{i,x_i} we can select the same s'_{i,x_i} as before: the back tag of s'_{i,x_i} contains 1's only in segment C_{i,x_i}, but s does not contain any 1's in C_{i,x_i}. This means that setting to 0 the back tag of s does not increase the distance between s and s'_{i,x_i}. Assume that s'_{i,x_i} is block $B_{i,x_i,t}$ for some t. For every $j \neq x_i$, select block $B_{i,j,t}$ from the string $s_{i,j}$. The only difference between blocks $B_{i,x_i,t}$ and $B_{i,j,t}$ is in the back tag: they have the same number of 1's in the back tag, but in different segments. However, s^* has only 0's in the back tag, hence $d(B_{i,j,t}, s^*) = d(B_{i,x_i,t}, s^*) \leq d'$. Therefore, s^* and the selected blocks form a solution where the back tag of the center string s^* contains only 0's. ☐

We can assume that the complete tag is $1^{d'-d}$ in s:

PROPOSITION 8.6. *If there is a solution for the constructed instance of* CLOSEST SUBSTRING, *then there is such a solution where the complete tag of the center string s contains only 1's.*

Proof. Let s be a solution where the number of 0's in the complete tag of the center string is minimal. Assume first that there is a 1 in the core of s. In this case replace this 1 with a 0, and set one of the 0's in the complete tag to 1. This modification does not change the difference of s from s_0. Furthermore, it does not increase the distance of s from $s'_{i,j}$: replacing the 0 with a 1 in the complete tag decreases the distance, while replacing the 1 with 0 in the core may or may not increase the distance. Therefore, the new center string contradicts the minimality of s.

Assume now that the core of s contains only 0's. We have already seen that the front tag of s is the same as the front tag of s_0 (Prop. 8.4), and the back tag contains only 0's (Prop. 8.5). Therefore, s differs from s_0 only in the complete tag. This means that in the complete tag of s we can replace every 0 with 1: the distance between s and s_0 increases only to $d' - d$, while the distance decreases between s and every string $s'_{i,j}$. □

The proofs of Prop. 8.4–8.6 are independent in the sense that we can assume that there is a solution where the center string s satisfies all three requirements at the same time. Assuming that s is of this form, it is not difficult to prove the converse of Prop. 8.2:

Proposition 8.7. *If there is a solution for the constructed instance of* Closest Substring*, then there is a solution for the* Set Balancing *problem.*

Proof. Consider a solution where the complete tag of s contains only 1's, and the back tag of s contains only 0's. Define the set $X \subseteq A$ based on the core of s: let an element of A be in X if and only if the corresponding character is 1 in the core of s. The string s differs from the template string s_0 at $|X|$ positions in the core and at $d' - d$ positions in the complete tag. Since $d(s, s_0) \le d'$, it follows that $|X| \le d$.

We claim that for every $1 \le i \le s$, a set can be selected from $\mathscr{S}_i$ that is balanced by X. Assume that $s'_{i,1}$ is the block $B_{i,1,t}$ of $s_{i,1}$ for some t. We show that $S_{i,t} \in \mathscr{S}_i$ is balanced by X. Let us determine the distance $d(B_{i,1,t}, s)$, which is by assumption at most d'. In the core, the two strings differ on the symmetric difference of $S_{i,t}$ and X. The strings do not differ on the complete tag, but they differ on every position of the back tag where $B_{i,1,t}$ is 1. There are exactly $d' - w_{i,t}$ such positions, hence

$$d(s, s'_{i,j}) = |X \bigtriangleup S_{i,k}| + d' - w_{i,t} \le d',$$

which means that $|X \bigtriangleup S_{i,t}| \le w_{i,t}$ and the set $S_{i,t}$ is balanced. □

Prop. 8.2 and 8.7 together prove the correctness of the reduction. □

Putting together Theorem 7.1 and 8.1 gives a two-step reduction from Maximum Clique to Closest Substring. Let us follow how the parameters depend on each other during this reduction. If an instance of Maximum Clique is given with parameter t, then Theorem 7 constructs an instance of Set Balancing with parameters

$$\begin{aligned} d &:= 2^t - 1 \\ m &:= \binom{t}{2} + 2^{2^{t-1}} - 1 = 2^{2^{O(t)}} \\ \ell &:= 2^{t-1}. \end{aligned}$$

Theorem 8.1 transforms this instance into an instance of Closest Substring with the following parameters:

$$\begin{aligned} k &:= (d + \ell + 1)m + 1 = 2^{2^{O(t)}} \\ d' &:= d + \ell = 2^{O(t)}. \end{aligned}$$

Theorem 3.3 gives an $|\Sigma|^{d(\log d+2)} n^{\log d + O(1)}$ time algorithm for Closest Substring and Theorem 5.5 gives an algorithm with running time $2^{kd} \cdot d^{O(d \log\log k)} \cdot n^{O(\log\log k)} \cdot |\Sigma|^d$. We argue that in some sense these algorithms are best possible:

the exponent of n cannot be improved to $o(\log d)$ or to $o(\log\log k)$ (modulo some complexity-theoretic assumptions).

Assume that there is an $f_1(k,d)\cdot n^{o(\log d)}$ time algorithm for CLOSEST SUBSTRING. We can construct an algorithm for MAXIMUM CLIQUE by reducing it to CLOSEST SUBSTRING and using our assumed algorithm for the latter problem. The running time of this algorithm for finding a size t clique is $f_1(k,d)\cdot n^{o(\log d)} = f_1(2^{2^{O(t)}}, 2^t)\cdot n^{o(\log 2^t)} = f_1'(t)\cdot n^{o(t)}$ (it can be assumed that the running time of the reduction to CLOSEST SUBSTRING is dominated by the time required to solve the CLOSEST SUBSTRING instance.) By a result of [9], the existence of an $f_1'(t)\cdot n^{o(t)}$ algorithm for MAXIMUM CLIQUE would imply that n-variable 3-SAT can be solved in $2^{o(n)}$ time, i.e., the Exponential Time Hypothesis would be violated. Therefore, it is highly unlikely that there is an algorithm for CLOSEST SUBSTRING with $o(\log d)$ in the exponent.

COROLLARY 8.8. *There is no $f_1(k,d)\cdot n^{o(\log d)}$ time algorithm for* CLOSEST SUBSTRING, *unless n-variable* 3-SAT *can be solved in time* $2^{o(n)}$.

Similarly, an $f_2(k,d)\cdot n^{o(\log\log k)}$ time algorithm for CLOSEST SUBSTRING would imply that there is an $f_2(2^{2^{O(t)}}, 2^t)\cdot n^{o(\log\log 2^{2^{O(t)}})} = f_2'(t)\cdot n^{o(t)}$ algorithm for MAXIMUM CLIQUE.

COROLLARY 8.9. *There is no $f_2(k,d)\cdot n^{o(\log\log k)}$ time algorithm for* CLOSEST SUBSTRING, *unless n-variable* 3-SAT *can be solved in time* $2^{o(n)}$.

In our reduction from MAXIMUM CLIQUE to CLOSEST SUBSTRING, the blow-up of the parameters is unusually large: double exponential. It might seem that with some more careful construction we could give a simpler reduction, where the parameters of the constructed instance are smaller. However, the connection with subexponential algorithms show that the double exponential increase cannot be avoided, it is a necessary part of any reduction from MAXIMUM CLIQUE to CLOSEST SUBSTRING. Assume that there is an $f(t)\cdot n^c$ time parameterized reduction where $d = g_1(t)$ and $k = g_2(t) = 2^{2^{o(t)}}$. This reduction and the $h(k,d)\cdot n^{O(\log\log k)}$ time algorithm of Theorem 5.5 would give an algorithm for MAXIMUM CLIQUE with running time

$$\begin{aligned} f(t)n^c + h(g_1(t), g_2(t))\cdot (f(t)n^c)^{O(\log\log g_2(t))} \\ = h'(t)\cdot n^{O(\log\log 2^{2^{o(t)}})} = h'(t)\cdot n^{o(t)}, \end{aligned}$$

which is not possible, unless 3-SAT has subexponential algorithms.

Cesati and Trevisan [7] and Bazgan [4] have shown that if a problem is W[1]-hard, then the corresponding optimization problem cannot have an EPTAS (i.e., a PTAS with running time $f(\epsilon)\cdot n^c$), unless FPT = W[1]. Let us recall the argument here. Assume that there is an approximation scheme with running time $f(\epsilon)\cdot n^c$ for CLOSEST SUBSTRING. Running the algorithm with $\epsilon = 1/2k$ decides whether there is a solution with $d \le k$: if there is such a solution, then the approximation scheme always produces a solution with d at most $(1+\epsilon)k < k+1$. This would imply an $f(1/2k)\cdot n^c$ algorithm for CLOSEST SUBSTRING, and it would follow that the problem is fixed-parameter tractable, which is not possible, unless FPT = W[1].

We can push this argument a bit further: it can be shown that there is no PTAS with running time $f(\epsilon)\cdot n^{o(\log 1/\epsilon)}$. The same reasoning as in the previous paragraph shows that such a PTAS would imply an $f(1/2k)\cdot n^{o(\log 2k)}$ algorithm for CLOSEST SUBSTRING. In Corollary 8.8, we have seen that this is not possible, unless there are subexponential algorithms for 3-SAT.

COROLLARY 8.10. *There is no $f(\epsilon)\cdot n^{o(\log 1/\epsilon)}$ time PTAS for* CLOSEST SUBSTRING, *unless n-variable* 3-SAT *can be solved in time* $2^{o(n)}$.

The lower bound of Corollary 8.10 does not match the known approximation schemes for the problem. Using a different approach, Andoni et al. [2] proved an essentially tight lower bound. However, in a strict technical sense, their lower bound is not directly comparable with Corollary 8.10.

9. Conclusions.

We have presented algorithms and complexity results for two string matching problems, Closest Substring and Consensus Patterns. We have proved that Closest Substring parameterized by the distance parameter d and by the number of strings k is W[1]-hard, even if the alphabet is binary. This improves the previous result of [14], where it is proved that the problem is W[1]-hard with parameter k only (and binary alphabet). Our hardness result also improves [20], where it is proved that Distinguishing Substring Selection (a generalization of Closest Substring) is W[1]-hard with parameters k and d (again with binary alphabet). In our reduction we used some of the techniques from these results, but new ideas were also required. In particular, we had to ensure that in the constructed instance of Closest Substring there is no solution where the center string is very close to some substring. This is easy to ensure if d is unbounded, or if we can use the additional features of Distinguishing Substring Selection. However, if d is a parameter, then we have to develop new combinatorial machinery to make sure that no solution can be close to some substring.

The W[1]-hardness of a problem is usually interpreted as evidence that the problem is unlikely to be fixed-parameter tractable, that is, the parameter has to appear in the exponent of n. Furthermore, using recent connections with subexponential algorithms, we can even give a lower bound on the exponent of n. Our reduction for Closest Substring is "weak" in the sense that the parameters are significantly increased (exponentially and double exponentially). Therefore, we obtain only weak lower bounds on the exponent of n: all we can show is that the exponent cannot be $o(\log d)$ or $o(\log\log k)$. However, it turned out that these bounds are tight: we presented two algorithms where the exponent of n is $O(\log d)$ and $O(\log\log k)$, respectively. The second algorithm is based on some surprising connections with the extremal combinatorics of hypergraphs. We have introduced and investigated the half-covering property, which played an important role in the algorithm. Furthermore, we have shown that all the copies of hypergraph H in hypergraph G can be efficiently found if H has small fractional cover number. This result might be useful in some other applications as well. More generally, the fractional cover number and Shearer's Lemma (which is the main combinatorial idea behind Lemma 4.1 and hence behind Theorem 4.3) can be useful algorithmic tools in other contexts, see [22].

The same hypergraph techniques can be used in the case of the Consensus Patterns problem. However, the combinatorial structure of this problem is slightly different, and this slight difference allows us to obtain a uniformly polynomial time algorithm with running time $f(|\Sigma|, \delta) \cdot n^9$. Therefore, in the constant alphabet case the problem is fixed-parameter tractable with parameter δ (and also with the larger parameter D). This resolves another open question from [14].

The algorithms of Theorem 5.5 and Theorem 6.1 are based on the same idea: we want to enumerate all the "small" places P in a large hypergraph G that are "well-covered" in a certain sense. Our algorithms do this in a somewhat cumbersome way: first every small well-covered hypergraph is enumerated, and then for each such H, the algorithm enumerates all the places where H appears in G. It might be possible to do this in a more direct and elegant way. What we need is an algorithm that enumerates maximal subset of vertices having the property that they can be fractionally covered by weight k, and the running time is something like $n^{O(k)}$. Developing such an algorithm could improve the running time of our algorithms, and, more importantly, would give us more insight into the nature of fractional edge covers.

Our results present an example where parameterized complexity and subexponential algorithms are closely connected. First, a weak parameterized reduction might be the sign that some kind of subexponential algorithm is possible for the problem. On the other hand, a parameterized reduction can be used to show the optimality of a subexponential algorithm. It is possible that this interplay between parameterized complexity and subexponential algorithms appears in the case of some other problems as well.

Acknowledgment. I'm grateful to Mike Fellows for directing my attention to this problem and to Ildi Schlotter for reading the manuscript.

REFERENCES

[1] N. Alon, *On the number of subgraphs of prescribed type of graphs with a given number of edges*, Israel J. Math., 38 (1981), pp. 116–130.

[2] A. Andoni, P. Indyk, M. Pǎtraşcu, *On the optimality of the dimensionality reduction method*, in Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), 2006, pp. 449–458.

[3] D. Angluin and L. G. Valiant, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155–193.

[4] C. Bazgan, *Schémas d'approximation et complexité paramétrée*, tech. report, Université Paris Sud, 1995.

[5] M. Blanchette, B. Schwikowski, and M. Tompa, *Algorithms for phylogenetic footprinting*, J. Comput. Biol., 9 (2002), pp. 211–223.

[6] J. Buhler and M. Tompa, *Finding motifs using random projections*, in Proceedings of the Fifth Annual International Conference on Computational Biology (RECOMB'01), 2001, pp. 69–76.

[7] M. Cesati and L. Trevisan, *On the efficiency of polynomial time approximation schemes*, Inform. Process. Lett., 64 (1997), pp. 165–171.

[8] J. Chen, B. Chor, M. Fellows, X. Huang, D. Juedes, I. Kanj, and G. Xia, *Tight lower bounds for certain parameterized NP-hard problems*, in Proceedings of 19th Annual IEEE Conference on Computational Complexity, 2004, pp. 150–160.

[9] J. Chen, X. Huang, I. A. Kanj, and G. Xia, *Linear FPT reductions and computational lower bounds*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing, New York, 2004, ACM, pp. 212–221.

[10] F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer, *Some intersection theorems for ordered sets and graphs*, J. Combin. Theory Ser. A, 43 (1986), pp. 23–37.

[11] R. G. Downey, *Parameterized complexity for the skeptic*, in Proceedings of the 18th IEEE Annual Conference on Computational Complexity, 2003, pp. 147–169.

[12] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, Monographs in Computer Science, Springer-Verlag, New York, 1999.

[13] P. A. Evans, A. D. Smith, and H. T. Wareham, *On the complexity of finding common approximate substrings*, Theoret. Comput. Sci., 306 (2003), pp. 407–430.

[14] M. R. Fellows, J. Gramm, and R. Niedermeier, *On the parameterized intractability of motif search problems*, Combinatorica, 26 (2006), pp. 141–167.

[15] J. Flum and M. Grohe, *Parameterized complexity and subexponential time*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, (2004), pp. 71–100.

[16] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Springer-Verlag, Berlin, 2006.

[17] Y. M. Fraenkel, Y. Mandel, D. Friedberg, and H. Margalit, *Identification of common motifs in unaligned DNA sequences: application to Escherichia coli Lrp regulon*, Computer Applications in the Biosciences, 11 (1995), pp. 379–387.

[18] M. Frances and A. Litman, *On covering problems of codes*, Theory Comput. Syst., 30 (1997), pp. 113–119.

[19] E. Friedgut and J. Kahn, *On the number of copies of one hypergraph in another*, Israel J. Math., 105 (1998), pp. 251–256.

[20] J. Gramm, J. Guo, and R. Niedermeier, *On exact and approximation algorithms for distinguishing substring selection*, in Fundamentals of computation theory, vol. 2751 of Lecture Notes in Comput. Sci., Springer, Berlin, 2003, pp. 195–209.

[21] J. Gramm, R. Niedermeier, and P. Rossmanith, *Fixed-parameter algorithms for closest string and related problems*, Algorithmica, 37 (2003), pp. 25–42.

[22] M. Grohe and D. Marx, *Constraint solving via fractional edge covers*, in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06), New York, NY, USA, 2006, ACM Press, pp. 289–298.

[23] D. Gusfield, *Algorithms on strings, trees, and sequences*, Cambridge University Press, Cambridge, 1997.

[24] G. Hertz and G. Stormo, *Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps.*, in Proceedings of the 3rd International Conference on Bioinformatics and Genome Research, 1995, pp. 201–216.

[25] R. Impagliazzo, R. Paturi, and F. Zane, *Which problems have strongly exponential complexity?*, J. Comput. System Sci., 63 (2001), pp. 512–530. Special issue on FOCS 98 (Palo Alto, CA).

[26] U. Keich and P. A. Pevzner, *Finding motifs in the twilight zone*, in Proceedings of the Sixth Annual International Conference on Computational Biology (RECOMB'02), 2002, pp. 195–204.

[27] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, *Distinguishing string selection problems*, Inform. and Comput., 185 (2003), pp. 41–55.

[28] C. Lawrence and A. Reilly, *An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences.*, Proteins, 7 (1990), pp. 41–51.

[29] M. Li, B. Ma, and L. Wang, *Finding similar regions in many sequences*, J. Comput. System Sci., 65 (2002), pp. 73–96. Special issue on STOC, 1999 (Atlanta, GA).

[30] M. Li, B. Ma, and L. Wang, *On the closest string and substring problems*, J. ACM, 49 (2002), pp. 157–171.

[31] P. A. Pevzner and S.-H. Sze, *Combinatorial approaches to finding subtle signals in DNA sequences*, in Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, AAAI Press, 2000, pp. 269–278.

[32] A. Price, S. Ramabhadran, and P. Pevzner, *Finding subtle motifs by branching from sample strings*, Bioinformatics, 19 (2003), pp. II149–II155.

[33] I. Rigoutsos and A. Floratos, *Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm*, Bioinformatics, 14 (1998).

[34] G. Stormo, *Consensus patterns in DNA.*, Methods in Enzymology, 183 (1990), pp. 211–221.

[35] G. J. Woeginger, *Exact algorithms for NP-hard problems: a survey*, in Combinatorial optimization - eureka, you shrink!, Springer-Verlag New York, Inc., 2003, pp. 185–207.