An Efficient Fixed Parameter Algorithm for 3-Hitting Set

Rolf Niedermeier*
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
niedermr@informatik.uni-tuebingen.de

Peter Rossmanith
Institut für Informatik, Technische Universität München,
Arcisstr. 21, D-80290 München, Fed. Rep. of Germany
rossmani@in.tum.de

Abstract

Given a collection C of subsets of size three of a finite set S and a positive integer k, the 3-Hitting Set problem is to determine a subset $S' \subseteq S$ with $|S'| \leq k$, so that S' contains at least one element from each subset in C. The problem is NP-complete, and is motivated, for example, by applications in computational biology. Improving previous work, we give an $O(2.270^k + n)$ time algorithm for 3-Hitting Set, which is efficient for small values of k, a typical occurrence in some applications. For d-Hitting Set we present an $O(c^k + n)$ time algorithm with $c = d - 1 + O(d^{-1})$.

Keywords: NP-complete problems, parameterized complexity, fixed parameter tractability, exact algorithms, Hitting Set.

1 Introduction

The *Hitting Set for Size Three Sets* or *3-Hitting Set* problem (*3HS* for short) is defined as follows:

*Work partially done within the PEAL project ("Parameterized Complexity and Exact ALgorithms"), supported by the Deutsche Forschungsgemeinschaft (NI 369/1-1).

Input: A collection C of subsets of size three of a finite set S and a positive integer k.
Question: Is there a subset $S' \subseteq S$ with $|S'| \leq k$ which allows S' to contain at least one element from each subset in C?
The natural generalization to sets of size d is called d-Hitting Set, while 3HS is itself a natural generalization of the well-known Vertex Cover problem (which is the same as 2HS). Both are NP-complete [21]. Put another way, 3HS can be seen as a Vertex Cover problem for hypergraphs, where there is an hyperedge between three instead of two vertices and the task is to determine a minimal subset of vertices covering all edges.

Approximability vs parameterized complexity. A computational problem once classified as NP-hard, it has become "every day routine" to ask for its approximability [4, 13, 14, 28]. The task is to find the best approximation factor possible within polynomial time where, as a rule, in theoretical studies the degree of the polynomial is less important than the approximation factor. Until recently, there was little known about lower bounds for the approximation factors. With the advent of probabilistically checkable proofs [3], however, this situation has changed. For example, it is now known that Vertex Cover cannot be approximated better than 1.1666 unless $P = NP$ [26]. Unfortunately, the best known upper bound is, basically, only 2, and it is a longstanding open problem whether or not a factor $2 - \epsilon$ for a constant ϵ is achievable. Despite the indisputable importance of approximation algorithms, the emergence of parameterized complexity [18] has opened an important and necessary new line of attacking computational intractability. That is, instead of searching for approximations one is looking for *exact* solutions of hard problems when the problem carries some (small) parameter. Of course, this shift of perspective means taking into account exponential running times—however, often the exponential growth can be restricted to the parameter. As a consequence, if the parameter can be kept small (as, e.g., often is the case in problems occurring in computational biology), the "parameterized point of view" [2] may be superior to the "approximation point of view." Studying mainly 3HS, in this paper we give one example for a hard problem where the parameterized algorithm (or, more commonly, "fixed parameter algorithm") may, in some applications, be preferable to the corresponding approximation algorithm(s).

Exact algorithms for *NP*-hard problems. The NP-complete Maximum Independent Set problem was studied in a series of papers [40, 29, 36, 9], the best known general bound so far being $O(1.211^n)$, where n is the number of vertices in the given graph. Also, special cases (e.g., bounded vertex degree) have recently been studied [9, 11]. Another, even more prominent field of investigations on upper bounds are Satisfiability problems, in particular, 3SAT (see [27, 37, 15] for some recent results). Very recently, maximum satisfiability problems also received considerable attention [30, 34, 7, 22]. Studying the different "potentials" for (exponential) upper bounds for hard problems has, thus, proven to be of both theoretical and practical interest. Studying parameterized complexity bounds (only exponential in some (small) parameter, see Vertex Cover [6, 32, 11, 39]) can lead to additional, new insights and improve the practicality of exact algorithms for hard problems.

Approximability of 3HS. The Hitting Set problem with no restriction on the subset size has a well-known one to one relationship with the Minimum Set Cover problem [5, 13]. The approximability of variants of Minimum Set Cover where the subsets are restricted in size, most importantly the subset size three case, have recently been considered [13, 24]. Unfortunately, there is no longer a one to one relationship between 3HS and Minimum Set Cover for size three subsets, so approximation results do not transfer in this case. The known approximation algorithms for 3HS so far only achieve an approximation factor of three [8, 13, 28]. Hence, it is particularly interesting to develop efficient (fixed parameter) algorithms providing optimal solutions. Finally, we only mention in passing that 3HS is also mentioned in [25] and that an average case analysis of a greedy algorithm for d-Hitting Set for constant d has been done in [16].

Motivation from computational biology. In computational biology, it is very natural that data sets are incomplete or faulty. Often, the corresponding problem of "cleaning up data" can be formulated as a covering problem: Given a set of experimental data points, some of which are in conflict. Is there a way to determine a minimum size set of data points such that, if "deleted" from the experimental data, this would remove ("explain") all inconsistencies? Associating vertices with data points and edges with conflicting pairs, this naturally leads to the classical Vertex Cover problem. It also motivates the related parameterized studies of this problem [6, 11, 32, 39]. In the context of studying phylogenies and, in particular, when trying to combine different phylogenetic trees, 3-Hitting Set naturally appears. The basic idea (as explained in detail in [20]) is that when comparing different trees, conflicts in the tree structures can be modeled as conflicting triples and the question then is to delete a minimum number of species (i.e., to find a minimum subset $S' \subseteq S$ when formulating it as a 3HS problem) in order to avoid all conflicts in the tree structures. Clearly, one assumes that there are not too many conflicts in the given data (otherwise the data would probably be worthless), making the parameterized point of view a promising solution strategy.

Results. Downey *et al.* [20] stated that 3HS can be solved using a search tree of size $(1+\sqrt{17}/2)^k \approx 2.5616^k$, referring to unpublished work of Bryant *et al.* [10]. This gives an $O(2.5616^k + n)$ algorithm for 3HS. Note that a size 3^k search tree is easy to obtain: Since each subset has to be covered by the hitting set, we have to take at least one of the three elements of a subset. It follows that the search tree will branch into three children. Hence, we can build a search tree of depth k and size 3^k (each inner node having three children). Clearly, this generalizes to d-Hitting Set for constant d, yielding an $O(d^k+n)$ algorithm in this case. While concentrating on the more important 3HS, we also show how to solve d-hitting set in exponential time, where the base of the exponential function is only $d-1+O(d^{-1})$. By way of contrast, the general Hitting Set problem with unbounded subset size is known to be $W[2]$-complete [18] and, hence, there is no hope for an efficient fixed parameter algorithm in general. In this paper, we improve the previous bounds on the search tree size for 3HS to a value of 2.270^k, yielding an $O(2.270^k + n)$ time algorithm for 3HS. Our result shows a significant improvement: Let us compare the exponential factors in the time complexities of the algorithms (i.e., the search tree size) for a reasonable value such as $k = 30$. We have $3^k \approx 2.06 \cdot 10^{14}$, $2.5616^k \approx 1.8 \cdot 10^{12}$, and $2.270^k \approx 4.795 \cdot 10^{10}$. Hence, for $k = 30$ the new algorithm is better than that of 3^k by a factor of 4300 and, even for 2.5616^k, better by a factor of 38. The factor increases with the value of k. Moreover, observe that our result holds in the worst case—it is to be expected that our algorithm runs significantly faster on average. Hence, like the simpler Vertex Cover problem (cf. [6, 11, 32, 39]), 3HS is a natural and important member of the class of problems with efficient fixed parameter algorithms [2, 18, 31].

Structure of the paper. In Section 2, we introduce some basic notation and give a short account of parameterized complexity. In Section 3, we survey the main parts of our algorithm for 3HS. In Section 4, we describe an important preprocessing for our algorithm—reduction to problem kernel—a well-known technique in parameterized complexity. The central part of our paper is Section 5, where we present our improved bounded search tree for 3HS. Section 6 contains the algorithm for d-Hitting Set. In Section 7, we draw some final conclusions.

2 Preliminaries

We assume familiarity with the basic notions and concepts of algorithms and complexity [12, 35]. In the problem *3HS* which we will study, we are given a collection of subsets of size three of a finite set S and a positive integer k, the question being whether or not there is a hitting set of size at most k. We assume that no three element subset occurs more than once within the collection. By n, we denote the length of the encoding of the input.

Equally, 3HS can be seen as a vertex cover problem for *hypergraphs:* Interpret the elements of S as vertices and interpret the size three subsets as hyperedges. Thus, a hyperedge now joins three vertices instead of two. As a result, 3HS requires the *covering* of all these three element sets (hyperedges) by elements (vertices), which is completely analogous to the well-known Vertex Cover problem. From this point of view, it is natural to speak of the *degree* of elements in S. It simply means the number of subsets in which it occurs. Moreover, we call a given collection of subsets *d-regular* if each element x has exactly degree d. Finally, we call an element $y \in S$ *dominated* by an element $x \in S$ if each subset containing y also contains x.

Our algorithm is based on two key techniques of parameterized complexity [18]: *reduction to problem kernel* and *bounded search tree.* The first technique deals with reducing the size of the search space and the second with a clever search through the search space. Both will be explained in detail in Sections 4 and 5. To estimate the size of bounded search trees (and, thus, the algorithm's overall complexity), we make use of *recurrence relations.* As a rule, we use linear recurrences with constant coefficients for whose solution there are some well-known techniques [23, 38]. It will be only briefly mentioned here that, in contrast to previous estimates of search tree sizes in parameterized complexity, we use a *system* of recurrences. For example, we may have recurrences such as

$$T_k = 1 + T_{k-1} + T_{k-2} + B_{k-1}$$

and

$$B_k = 1 + B_{k-1} + T_{k-1}$$

with $T_1 = B_1 = 1$ and $T_2 = 2$, where we start with T_k (and not B_k). Since for non-degenerate trees (inner nodes have at least two children) the number of leaves is at least half of all tree nodes, to get an asymptotic solution for the recurrences, we may drop the additive term "1+." Solving these simplified recurrences, we obtain the so-called *branching number* α. This simply tells us that $T_k = O(\alpha^k)$, thereby giving an upper bound for the search tree size. Note that we will have to study several cases for our algorithm, each yielding some recurrence(s).

Without going into details, let us briefly say a few words about *parameterized complexity theory* [18]. Parameterized complexity, as chiefly developed by Downey and Fellows, is one of the latest approaches to attack problems that are *NP*–complete. The basic observation is that for many hard problems the seemingly inherent combinatorial explosion can be restrained to a "small part" of the input, the *parameter.* So, for instance, the Vertex Cover problem can be solved by an algorithm with running time $O(kn + 1.32472^k k^2)$ [6], where the parameter k is a bound on the maximum size of the vertex cover set we are looking for and n is the number of vertices in the given graph.[1] The fundamental assumption is $k \ll n$. As can easily be seen, this yields an efficient, practical algorithm for small values of k, "relativizing" to some extent the meaning of *NP*-hardness in some settings occurring in practice. A problem is called *fixed parameter tractable* if it can be solved in time $f(k)n^{O(1)}$ for an arbitrary function f which depends only on k, where n is the input size. The corresponding complexity class is called FPT. Unfortunately, this $f(k)$ is usually not as small as in the case of Vertex Cover, but grows much faster. For instance, $f(k) = 11^k$ for Planar Dominating Set [17] is still a comparatively slowly growing function, already making the algorithm impractical for small values of k. Concerning Planar Dominating Set, however, there was a recent breakthrough showing that it can be solved with $f(k) = c^{\sqrt{k}}$ [1], a significant asymptotic improvement. The given constant $c = 3^{6\sqrt{34}}$ is still much too big to imply directly practical significance of this result. This could be one of the primary deficiencies of parameterized complexity theory. As Downey *et al.* [20] said, "the extent to which FPT is really useful is unclear." So far, there are only relatively few examples of problems that are fixed parameter tractable and possess algorithms of comparable (practical) efficiency to that of Vertex Cover [31]. In this paper, we contribute to the list of efficient, nontrivial FPT algorithms for important parameterized problems. Finally, it is worth mentioning that a number of heuristic algorithms that are widely used in practical computing are actually FPT algorithms [19, 20]. Thus, as a foreseeable line of future research, parameterized complexity theory might lead to a new approach in designing, analyzing, and understanding heuristic algorithms. Whether or not this is a useful and widespread phenomenon has to be decided by studies to come.

[1] Further improvements of, in particular, the exponential base significantly below 1.3 have been developed in [32, 11, 39].

3 Overview of our algorithm

Our algorithm consists of two main components: reduction to problem kernel and a bounded search tree. Reduction to problem kernel shows us how to transform the given input instance of size n into a new one of size $O(k^3)$. That is, the size of the new problem instance depends exclusively on the parameter k. Hence, the bounded search tree algorithm can work on instances whose size depends only on k. Typically, parameterized algorithms based on reduction to problem kernel and bounded search tree methods have a time complexity of the form $O(t(k)p(k) + q(n, k))$, where $t(k)$ is the size of the search tree (number of tree nodes), $p(k)$ is the size of the problem kernel, and $q(n, k)$ is the time needed to perform the reduction to problem kernel. Very recently, it was shown how to improve this to $O(t(k) + q(n, k))$ [33], provided that p and q are polynomial functions. The idea is to interleave reduction to problem kernel and the bounded search trees. In our case, we have $t(k) = 2.270^k$ (Section 5), $p(k) = k^3$, and $q(n, k) = n$ (Section 4). Hence, as a result we find that 3HS can be solved in time $O(2.270^k + n)$.

In what follows, we consider the structure of the bounded search tree, the heart of our algorithm. Our fundamental aim is to undertake a case distinction concerning the degree of vertices x in the base set S. However, there are also some special cases, which are always considered first. For example, if there is a singleton $\{x\}$ in our collection, we clearly have to take x into our hitting set. A simple but important concept is that of *domination*. An element x is dominated by an element y if whenever x occurs in a set of the collection, then y will occur in this set as well. In this case, we can delete x from the sets without repercussion (see Subsection 5.1). Lastly, before coming to degree considerations, a case of special importance is when we have subsets with two elements in our collection (Subsection 5.2). In this case, which can easily be dealt with, the size of the search tree is at most B_k, whereas when there is no such subset (i.e., all subsets have size three), its size is at most T_k.

Summarizing, the structure of our search tree algorithm is as follows. Observe that the subsequent order of the steps is important. In each step, the algorithm always executes the applicable step with the lowest possible number:

1. Deal with simple cases, that is, one element subsets, elements occurring in only one set, or dominated elements (cf. Case 5.1).

2. Deal with subsets of size two (cf. Case 5.2).

3. Deal with elements of degree 3 (cf. Case 5.3).

$$B_k \le \max \begin{cases} 2B_{k-1} & \text{Case 5.2} \\ T_{k-1} + T_{k-2} + T_{k-3} & \text{Case 5.2} \\ 2B_{k-2} + T_{k-1} & \text{Case 5.2} \end{cases}$$

$$T_k \le \max \begin{cases} B_k + T_{k-2} + T_{k-3} & \text{Case 5.3} \\ 4B_{k-2} + T_{k-1} & \text{Case 5.3} \\ B_{k-1} + T_{k-1} + T_{k-2} & \text{Case 5.4} \\ 8B_{k-3} + T_{k-1} & \text{Case 5.4} \\ 3B_{k-1} & \text{Case 5.5} \end{cases}$$

Table 1: Upper bounds on T_k and B_k in various cases.

4. Deal with elements of degree at least 4 (cf. Case 5.4).

5. Deal with the case that the collection of subsets is 2-regular (cf. Case 5.5).

It is easily verified that the above case distinction takes all cases that may occur into consideration. As a rule, each case leads to some recursive calls. In Table 1 we list all upper bounds for T_k and B_k that determine the size of the search tree. Looking for an upper bound for this set of recurrences, we obtain size $O(2.270^k)$ for our search tree.

4 Reduction to problem kernel

In this section, we show how to reduce the original instance to a new one consisting of only $O(k^3)$ elements. The idea is that, without needing a case distinction which would lead to a branching of the recursion, we *must* put "high degree elements" into the hitting set.

We assume a standard RAM to be acting as our model of computation. The word size is big enough to hold numbers such as n or k in one word. Two words can be added, subtracted, multiplied, etc. in constant time. Furthermore, n is the size of the input. We assume that an instance of 3HS is encoded in straightforward way using a finite alphabet. The length of this encoding is called n. The *elements* in the collection can be encoded arbitrarily as strings over a finite alphabet, for example, as binarily encoded integers, but every other encoding is also permitted. We can, however, convert the encoding to integers between 1 and n by sorting all elements by radix-sort and then replacing elements according to rank. This conversion only requires linear time. In the following, we assume that elements are integers between

1 and n.[2]

Proposition 1 *There is a problem kernel of size $O(k^3)$ for 3HS, and it can be found in linear time.*

Proof. Firstly, let us consider two fixed elements $x, y \in S$:
Claim 1: There can be at most k size three subsets in the collection C that contain both x and y.
Claim 1 is seen as follows. Assume that there are more than k subsets containing x and y. Since each set appears only once in C, this implies that there are more than k different "third" elements in the corresponding sets. Hence, to cover these more than k different sets with at most k elements from the base set S, we *have* to bring at least one of x and y into our hitting set S'. This means, however, that all sets containing both x and y can be deleted from our collection C. This proves Claim 1.

Next, we consider the case of only one fixed element $x \in S$:
Claim 2: There can be at most k^2 size three subsets in the collection C that contain x.
Claim 2 is seen as follows. Assume that there are more than k^2 subsets containing x. From Claim 1, we know that x can occur in a subset together with another element y at most k times. Hence, if there were more than k^2 subsets containing x, these could not be covered by some $S' \subseteq S$ with $|S'| \leq k$ without taking x. Thus, x must be in S' and the corresponding sets can be deleted. This proves Claim 2.

Now, from Claim 2, we can conclude that each element x from S can occur in at most k^2 subsets in C. (Otherwise, x *had* to be in the hitting set S'.) Clearly, because $|S'| \leq k$, this means (provided that C has a hitting set of size $\leq k$) that C can consist of at most $k \cdot k^2 = k^3$ size three subsets. This is the size of the problem kernel.

Finally, we remark that we can easily count in how many sets each element occurs and throw away in linear time all elements (and their corresponding subsets) occurring in more than k^2 subsets. □

5 Bounded search tree

This section forms the heart of our paper. Analyzing several cases, we show that we can construct a search tree of size $O(2.270^k)$. Due to the reduction

[2]This conversion yields an encoding that allows us later to use a dictionary indexed by elements with constant update and lookup times and linear initialization time.

to problem kernel in Section 4, we may assume that the given input instance has size $O(k^3)$.

5.1 Simple cases

There exist some simple cases that we always consider first. Firstly, assume that there is a singleton, say $\{x\}$. Then, we clearly have to take x into the hitting set without any branching of the recursion.

Secondly, assume that there is an element $x \in S$ that occurs in only one set, e.g., of size three: $\{x, a, b\}$. Then it suffices to consider the covering of $\{a, b\}$, leading to the recursive call B_k. Thus, we find the recurrences as shown in the following Subsection 5.2.

Thirdly, if an element y is dominated by an element x, it never occurs in a set without x occurring in the same set. This implies, however, that it would not make sense to take y and not to take x into the hitting set. As a consequence, we can simply throw away all occurrences of y, thus obtaining sets of size two or one instead of three element ones.

In the cases handled in the following subsections, it will be of key importance to rely on the absence of dominated elements, degree 1 elements, and sets of size one in the given instance.

5.2 Subsets of size two

We distinguish whether the size of all sets is at least three and whether there is at least one set with size two. In this subsection we assume the latter. An upper bound on the number of leaves in a branching tree whose root has this property will be called B_k. First of all, note that we can discard from further consideration the case when all sets have size two, because we can simply apply the much better results for 2HS, i.e., Vertex Cover [11, 20, 32, 39]. Hence, in the following subcases, at least one subset of size three occurs.

Let us first handle the special case where there are two sets

$$\{x, y\} \text{ and } \{x, a, b\},$$

where x only occurs in these two sets and b may be missing from the second set. We can branch according to y: If y is in the hitting set, then x occurs only in $\{x, a, b\}$ and x can be deleted from this subset. This must happen because $y \notin \{a, b\}$ (otherwise x would be dominated by y). The corresponding subtree has at most B_{k-1} leaves. If y is not in the hitting set, then x is. Since there are no elements occurring in only one subset, y occurs in some other set from which it is deleted, leaving a set of size at most two. Hence, this subtree has at most B_{k-1} leaves, as well. Altogether, the corresponding upper bound for this case is $2B_{k-1}$.

We continue with the case that x occurs in at least two sets of size two and one set of size three, that is

$$\{x, y_1\},\ \{x, y_2\},\ \{x, a, b\}.$$

If x is in the hitting set, we get a T_{k-1} branch. If x is not in the hitting set, y_1 and y_2 have to be in the hitting set and (since, without loss of generality, $y_1 \neq y_2$) we trivially get a T_{k-2} branch. The corresponding upper bound reads $T_{k-1} + T_{k-2}$.

Next, we consider the case of three sets

$$\{x, y\},\ \{x, a, b\},\ \{x, a, c\},$$

where x may occur in other sets as well. If x is in the hitting set, we get a T_{k-1} branch. Otherwise, y must be in the hitting set. Among others, the sets $\{a, b\}$ and $\{a, c\}$ remain. Now we branch according to a. If a is in the hitting set, we obtain a T_{k-2} branch and, otherwise, b and c must be in the hitting set, yielding a T_{k-3} branch. (Note that $b \neq c$.) The corresponding upper bound is $T_{k-1} + T_{k-2} + T_{k-3}$.

Finally, the remaining case is three sets

$$\{x, y\},\ \{x, a, b\},\ \{x, c, d\},$$

where x may again occur elsewhere, but $\{a, b\} \cap \{c, d\} = \emptyset$. If x is in the hitting, set we get a T_{k-1} branch. Otherwise, we branch according to a. Note that now, y must be in the hitting set. If a is in the hitting set, we still have the set $\{c, d\}$ and, hence, a B_{k-2} branch. If a is not, then b is in the hitting set and we are also left with $\{c, d\}$: Another B_{k-2} branch. The corresponding upper bound reads $T_{k-1} + 2B_{k-2}$.

5.3 Degree 3

In this subsection, we assume that there is an element x that occurs in exactly three sets

$$\{x, a_1, a_2\}, \{x, b_1, b_2\}, \{x, c_1, c_2\}.$$

We can assume that all sets have size three because, if they did not, the considerations in Subsection 5.2 would apply.

We consider two subcases: Firstly, that there is another element besides x that occurs in at least two of the three sets (A.) and, secondly, that there is no such element (B.).

A. Let us assume $a_1 = b_1$. Then $a_1 \neq c_1$ and $a_1 \neq c_2$ because, otherwise, x would be dominated by $a_1 = b_1$. We make three branches: Either c_1 and a_1 are in the hitting set or c_1 is and a_1 is not or, finally, c_1 is not in the hitting set.

If c_1 and a_1 are in the hitting set, we get a T_{k-2} branch because the size of the hitting set grows by two.

If only c_1 is in the hitting set, but a_1 is not, then the elimination of $\{x, c_1, c_2\}$ will leave $\{x, a_1, a_2\}, \{x, a_1, b_2\}$ as the only sets that contain x. Hence, x is dominated by a_1 and, since we assume that a_1 is not in the hitting set, x won't be either. This leaves the two singletons $\{a_2\}$ and $\{b_2\}$. They are, in fact, *two* sets, since $a_2 \neq b_2$ (otherwise x would not have occurred in three sets, but only in two). We can include a_2 and b_2 together with c_1 in the hitting set and, consequently, obtain a T_{k-3} branch.

Finally, if c_1 is *not* in the hitting set, then $\{x, c_2\}$ remains after c_1 has been eliminated, yielding at least a B_k branch.

Summarizing, the upper bound reads $T_k \leq T_{k-2} + T_{k-3} + B_k$.

B. Now we may assume that $a_1, a_2, b_1, b_2, c_1, c_2$ are pairwise different. We branch on x. By a T_{k-1} branch, we deal with the case where x is in the hitting set. Otherwise we can eliminate x, leaving $\{a_1, a_2\}, \{b_1, b_2\}, \{c_1, c_2\}$. Now we branch according to whether a_1 or a_2 is in the hitting set and according to whether b_1 or b_2 is in the hitting set. Hence, we get four branches, each putting two elements in the hitting set and leaving the two element set $\{c_1, c_2\}$. Therefore, the corresponding recurrence reads $T_k \leq T_{k-1} + 4B_{k-2}$.

5.4 Degree at least 4

In this subsection, we assume that there is an element x that occurs in at least fours sets

$$\{x, a_1, a_2\}, \{x, b_1, b_2\}, \{x, c_1, c_2\}, \{x, d_1, d_2\}.$$

Subsequently, we distinguish between two cases.

Firstly, assume that a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2 are not all pairwise different. Without loss of generality, assume that $a_1 = b_1$. (Of course, $a_1 \neq a_2$, $b_1 \neq b_2$, $c_1 \neq c_2$, and $d_1 \neq d_2$.) Clearly, $a_1 = b_1$ implies that $a_2 \neq b_2$ because, otherwise, we would have two times the same size three subset in our given collection. We claim the upper bound $T_{k-1} + B_{k-1} + T_{k-2}$ for this case. We branch on x as follows. If x is part of the hitting set, all four sets above are covered. We obtain a T_{k-1} branch. If we do not include x in the hitting set, then we will, in particular obtain the sets $\{a_1, a_2\}$ and $\{a_1, b_2\}$

which are to be covered. Upon branching on a_1, we end up with the following situation. If a_1 is in the hitting set, both of these sets are covered and there must be an additional two-element set, without loss of generality, $\{c_1, c_2\}$, remaining. This is due to the following fact: Since we may assume that a_1 is not dominated by x and vice versa (see Subsection 5.1), there must be a set containing x and not containing a_1. Without loss of generality, let this set be $\{x, c_1, c_2\}$. Hence, since we decided not to take x in the hitting set, $\{c_1, c_2\}$ remains to be covered. Thus, we can continue with a B_{k-1} branch here. Eventually, if a_1 is not in the hitting set, then only $\{a_2\}$ and $\{b_2\}$ remain to be covered and, clearly, we have to take both, leading to a T_{k-2} branch.

Clearly, here and in the following case, the situation (and, consequently, the branching number) improves if we have degree greater than 4.

Now let us turn to the second case, that is, assuming that a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2 are pairwise distinct. In this case, we claim the upper bound $T_{k-1} + 8B_{k-3}$. Again, we branch on x. Bringing x into the hitting set leads to a T_{k-1} branch. If x is not in the hitting set, we have to cover the four sets

$$\{a_1, a_2\}, \{b_1, b_2\}, \{c_1, c_2\}, \{d_1, d_2\}.$$

Since these contain eight distinct elements, we can branch in the manner of a binary tree of height 3, yielding eight possibilities, each putting three elements in the hitting set and, without loss of generality, leaving in each branch the two element set $\{d_1, d_2\}$. Hence, we get eight B_{k-3} branches.

5.5 The collection is 2-regular

Finally, we end up with 2-regular collections, that is, each element $x \in S$ occurs in exactly two subsets of the given collection. Hence, we have

$$\{x, a, b\}, \{x, c, d\}.$$

We may assume that a, b, c, d all are pairwise distinct, because if, e.g., $a = c$, then x would be dominated by a. Therefore, we branch according to a. If a is in the hitting set, then $\{x, a, b\}$ is covered and $\{x, c, d\}$ will be replaced by $\{c, d\}$, because x is dominated after $\{x, a, b\}$ has been removed. This leads to a B_{k-1} branch. If a is not in the hitting set, then we will branch additionally according to x. If x is in the hitting set, then both of the above sets are covered. However, since we assume 2-regularity, we also know that a has to appear in some other set $\{a, e, f\}$. Since this now is the only remaining occurrence of a, this set is replaced by $\{e, f\}$. Consequently, we find the recursive call B_{k-1}. Finally, suppose that neither a nor x are in the hitting set. Then b has to be in the hitting set. Furthermore, $\{x, c, d\}$ is replaced by $\{c, d\}$, yielding a B_{k-1} branch. Hence, the size of this subtree is at most $3B_{k-1}$ in the case of 2-regular collections.

5.6 Summarizing: The main theorem

Summing up, we have the following result.

Theorem 2 *3HS can be solved in time $O(2.270^k + n)$.*

Proof. In Section 4, we gave a reduction to a problem kernel of size $O(k^3)$ (Proposition 1). In Section 5, we described a search tree whose size is bounded from above by the recurrences given in Table 1 (see Section 3). These yield the search tree size $O(2.270^k)$. Since for each node of the search tree, we have to process the collection of subsets (i.e., throwing out subsets or deleting elements from them) in time linear in the input size, we obtain the time complexity $O(2.270^k k^3 + n)$. Now, we apply the interleaving technique from [33]. This reduces the running time to $O(2.270^k + n)$, replacing k^3 with a small constant. □

6 *d*-Hitting Set for general *d*

In this section, we present a more general algorithm that works for general d. It is quite efficient, but, of course, outperformed by the above algorithm for the most important case $d = 3$.

The trivial algorithm that tries all d possibilities for a set of d elements has running time $O(d^k + n)$. Our algorithm is better, having running time $O(\alpha^k + n)$, where

$$\alpha = \frac{d-1}{2} + \frac{d-1}{2}\sqrt{1 + \frac{4}{(d-1)^2}} = d - 1 + \frac{1}{d-1} + O(d^{-3}) = d - 1 + O(d^{-1}).$$

The algorithm works as follows: First eliminate all dominating elements. Choose some set $s = \{x_1, x_2, \ldots, x_d\}$. Then branch according to the following possibilities: (1) Choose x_1 for the hitting set, and (2) choose that x_1 is not in the hitting set, but x_i is for $i = 2, \ldots, d$. That makes d branches in total. If $T(k)$ is the number of leaves in a branching tree, then the first branch has at most $T(k-1)$ leaves. Let $B(k)$ be the number of leaves in a branching tree where there is at least one set of size $d-1$ or smaller. For each $i = 2, \ldots, d$, there is some set s' in the given collection such that $x_1 \in s'$, but $x_i \notin s'$. Therefore, the size of s' is at most $d-1$ after

d	3	4	5	6	7	8	9	10	20	50	100
$T(k)$	2.41^k	3.30^k	4.23^k	5.19^k	6.16^k	7.14^k	8.12^k	9.11^k	19.05^k	49.02^k	99.01^k

Table 2: Running times for d-Hitting Set.

excluding x_1 from and including x_i in the hitting set. Altogether we get $T(k) \leq T(k-1) + (d-1)B(k-1)$.

If there is already a set with at most $d-1$ elements, we can play the same game and get $B(k) \leq T(k-1) + (d-2)B(k-1)$. The branching number of this recursion is α from above.

Table 2 shows the resulting running times for several values of d. Please note that even for $d = 3$ the result is much better than the best known special algorithm for 3-Hitting Set (cf. [10, 20]) and that α is always *smaller* than $d - 1 + (d-1)^{-1}$.

7 Conclusion

In this paper, we have given the best fixed parameter algorithms for the d-Hitting Set problem thus far for all values of d, emphasizing the most important case, $d = 3$. Observe that the general Hitting Set problem (unbounded subset size) is $W[2]$-complete, so there is little chance of getting an *FPT* algorithm for the general problem [18]. A general, completely unsolved problem in parameterized complexity is whether it is possible to show (relative) lower bounds for the exponential terms achievable. For instance, how likely or unlikely is it that 3HS can be solved in time $O(2^k + n)$? Or would solvability of 3HS in that time imply something unlikely? This problem, however, seems hard, even when restricted to the framework of bounded search tree algorithms.

Furthermore, it would be interesting to give a competitive implementation of our algorithm. Note that compared with other efficient *FPT* algorithms such as, for example, Vertex Cover [6, 11, 32, 39], our algorithm requires a relatively small number of case distinctions. Thus, there is a good chance that the realization of our algorithm will be fairly easy and efficient. Moreover, this could give insight into the heuristic qualities of our approach.

Acknowledgment. We thank Magnus M. Halldórsson for informing us on the approximability results for 3HS. We are grateful to Jochen Alber for his observation showing a kernel size of $O(k^3)$ instead of $O(k^6)$ (Proposition 1). Finally, we profited from the insightful remarks of two anonymous referees of *Journal of Discrete Algorithms.*

References

[1] J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for Planar Dominating Set and related problems. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, number 1851 in Lecture Notes in Computer Science, pages 97–110, Bergen, Norway, July 2000. Springer-Verlag.

[2] J. Alber, J. Gramm, and R. Niedermeier. Faster exact solutions for hard problems: a parameterized point of view. Invited for submission to a special issue of *Discrete Mathematics*, January 2000.

[3] S. Arora and C. Lund. Hardness of approximation. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, chapter 10, pages 399–446. PWS Publishing Company, Boston, 1997.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation—Combinatorial Optimization Problems and their Approximability Properties.* Springer-Verlag, 2000.

[5] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21:136–153, 1980.

[6] R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

[7] N. Bansal and V. Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation*, number 1741 in Lecture Notes in Computer Science, pages 247–258, Chennai, India, December 1999. Springer-Verlag.

[8] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the Weighted Vertex Cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[9] R. Beigel. Finding maximum independent sets in sparse and general graphs. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 856–857, 1999.

[10] D. Bryant, M. Fellows, V. Raman, and U. Stege. On the parameterized complexity of MAST and 3-Hitting Sets. Unpublished manuscript, 1998.

[11] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, number 1665 in Lecture Notes in Computer Science, pages 313–324, Ascona, Switzerland, June 1999. Springer-Verlag.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[13] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, August 1998.

[14] P. Crescenzi and V. Kann. How to find the best approximation results—a follow-up to Garey and Johnson. *ACM SIGACT News*, 29(4):90–97, 1998.

[15] E. Dantsin, A. Goerdt, E. A. Hirsch, and U. Schöning. Deterministic algorithms for k-SAT based on covering codes and local search. In *Proceedings of the 27th International Conference on Automata, Languages, and Programming*, Lecture Notes in Computer Science. Springer-Verlag, July 2000. To appear.

[16] W. Fernandez de la Vega, V. Th. Paschos, and R. Saad. Average case analysis of a greedy algorithm for the minimum hitting set problem. In I. Simon, editor, *Proceedings of the 1st Symposium on Latin American Theoretical Informatics*, number 583 in Lecture Notes in Computer Science, pages 130–138, São Paulo, Brazil, April 1992. Springer-Verlag.

[17] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 219–244. Boston: Birkhäuser, 1995.

[18] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.

[19] R. G. Downey and M. R. Fellows. Parameterized complexity after (almost) ten years: Review and open questions. In *Combinatorics, Computation & Logic, DMTCS'99 and CATS'99*, Australian Computer Science Communcations, Volume 21 Number 3, pages 1–33. Springer-Verlag Singapore, 1999.

[20] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 of *AMS-DIMACS*, pages 49–99. AMS Press, 1999.

[21] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, 1979.

[22] J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith. New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. Invited for submission to a special issue of *Discrete Applied Mathematics.* Preliminary version available as ECCC Technical Report R00-037, Trier, Fed. Rep. of Germany, May 2000.

[23] D. H. Greene and D. E. Knuth. *Mathematics for the analysis of algorithms.* Progress in computer science. Birkhäuser, 2d edition, 1982.

[24] M. M. Halldórsson. Approximating k-set cover and complementary graph coloring. In *5th International Conference on Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 118–131, June 1996.

[25] M. M. Halldórsson and K. Tanaka. Approximation and special cases of common subtrees and editing distances. In T. Asano, Y.Igarashi und H. Nagamochi, S. Miyano, and S. Suri, editors, *Proceedings of the 7th International Symposium on Algorithms and Computation*, number 1178 in Lecture Notes in Computer Science, pages 75–84, Osaka, December 1996. Springer-Verlag.

[26] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.

[27] E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.

[28] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems.* Boston, MA: PWS Publishing Company, 1997.

[29] T. Jian. An $o(2^{0.304n})$ algorithm for solving Maximum Independent Set problem. *IEEE Transactions on Computers*, 35(9):847–851, 1986.

[30] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31:335–354, 1999.

[31] R. Niedermeier. Some prospects for efficent fixed parameter algorithms (invited paper). In B. Rovan, editor, *Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, number 1521 in Lecture Notes in Computer Science, pages 168–185. Springer-Verlag, 1998.

[32] R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In C. Meinel and S. Tison, editors, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, number 1563 in Lecture Notes in Computer Science, pages 561–570. Springer-Verlag, 1999.

[33] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000.

[34] R. Niedermeier and P. Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36:63–88, 2000.

[35] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[36] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[37] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 410–414, 1999.

[38] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Weseley Publishing Company, 1996.

[39] U. Stege and M. Fellows. An improved fixed-parameter-tractable algorithm for vertex cover. Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.

[40] R. E. Tarjan and A. E. Trojanowski. Finding a Maximum Independent Set. *SIAM Journal on Computing*, 6(3):537–550, 1977.