Parameterized Algorithms for Double Hypergraph Dualization with Rank Limitation and Maximum Minimal Vertex Cover

Peter Damaschke
Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
ptr@chalmers.se

Abstract

Motivated by the need for succinct representations of all "small" transversals (or hitting sets) of a hypergraph of fixed rank, we study the complexity of computing such a representation. Next, the existence of a minimal hitting set of at least a given size arises as a subproblem. We give one algorithm for hypergraphs of any fixed rank, and we largely improve an earlier algorithm (by H. Fernau, 2005) for the rank-2 case, i.e., for computing a minimal vertex cover of at least a given size in a graph. We were led to these questions by combinatorial aspects of the protein inference problem in shotgun proteomics.

1 Introduction

1.1 Some Terminology

A *hypergraph* $H = (V, E)$ consists of a set V of *vertices* and a set E of *edges*, each being a subset of V. If not stated otherwise, we consider only *simple* hypergraphs H, which means that no edge is subset of another edge. (Simple hypergraphs are also known as Sperner families.) The *support* of a hypergraph is the set of all vertices covered by edges, the remaining vertices are called *isolated*. A *hitting set* (synonymously: *transversal*) of H is a subset of V that intersects every edge. A *minimal hitting set (minimal transversal)* is a hitting set such that no proper subset is a hitting set, too. Note that isolated vertices are never contained in a minimal transversal. The *rank* of H is the largest number of vertices in an edge of H. *Restricting* a hypergraph H to a subset $W \subset V$ means to throw out all vertices that are not in W. Note that a restricted hypergraph is in general not simple even if H was;

some edges may even be emptied. Hypergraphs of rank 2 are *graphs*, and hitting sets in graphs are also known as *vertex covers*.

We consider the following two operators on hypergraphs that do not change the vertex set but the edge set. The edges of the *dual hypergraph* H^D are the minimal transversals of H. For a positive integer k, the edges of the *rank-limited* hypergraph H^k are all edges of H of size at most k. Recall that we consider only simple hypergraphs. H^D is always a simple hypergraph by definition, and H^k is simple if H was. A well known duality theorem (see any standard reference like [1]) states $H^{DD} = H$, and trivially we have $H^{kl} = H^{\min(k,l)}$. However, alternating applications of dualization and rank limitation can yield new hypergraphs with interesting interpretations, as we will see below. H^D itself describes implicitly the family of all hitting sets of H, since it is obviously sufficient to know the minimal transversals.

The Hitting Set problem is: Given a hypergraph H and a number k, find some hitting set of size at most k in H. The Maximum Minimal Hitting Set (MMHS) problem is: Given a hypergraph H and a number k, is there some minimal transversal with more than k vertices in H? (In MMHS we do not attempt to compute the exact size of a maximum minimal transversal, we only want to figure out whether it is larger than the given k.) The problems Vertex Cover and Maximum Minimal Vertex Cover (MMVC) are similarly defined for graphs.

Notice that, by definition and by the duality theorem, the following three statements are equivalent for simple hypergraphs H: (i) MMHS for H, k has a negative answer; (ii) $H^D = H^{Dk}$; (iii) $H = H^{DkD}$.

1.2 An Inference Problem from Bioinformatics

In shotgun proteomics, an unknown sample of proteins is digested into peptides which are identified (e.g., by mass spectrometry), and then a database of candidate proteins is searched for proteins consisting of the peptides found, in order to infer the proteins in the sample. Instead of peptides one may use their masses only. Among several technical issues, shotgun proteomics applied to mixtures of proteins involves a principal combinatorial problem: One cannot see which peptides (or masses) come from which protein. It is implicit in [14] and other work that this leads to the problem of computing transversals in hypergraphs.

The formalization as a hypergraph H works as follows: V is the set of candidate proteins (whose peptide sets are contained in the observed set), and for every observed peptide, an edge represents the set of candidate proteins containing this peptide. Some upper bound k on the number of proteins in the sample is known, and a wealth of edges has a size limited by some small constant r, that is, the corresponding peptide appears in very few candidate proteins. We consider the hypergraph consisting of these small

edges and try to infer the possible protein mixtures. Every edge has to be "explained" by the presence of some of its vertices in the mixture, hence the possible solutions are exactly the hitting sets of size at most k. (Thus, in the following, the term *solution* refers to hitting sets of size at most k in H, just for brevity). Note that the hypergraph may be non-simple in the first place, but any edge containing a smaller edge can be deleted without changing the solution space. Also, it is sensible to ignore edges larger than some predefined size r: Peptides occuring in many candidate proteins are not very informative. The size limitation k on the solutions easily renders them redundant, namely if every transversal of size at most k already intersects these large edges. In the worst case, of course, the removal of some large edges might add extra solutions, but then we trade specificity for speed: Only the limitation to some fixed rank makes the problem computationally feasible in terms of parameterized complexity (see Section 1.3). As an illustration we give a trivial example of the above issues: If a hypergraph contains $r+1$ disjoint edges $\{u_i, v_i\}$ $(i = 0, \ldots, r)$ and another edge $\{u_0, \ldots, u_r\}$, then ignoring this large edge allows for 2^r different transversals with r vertices in this part, and re-insertion of the large edge excludes only one of them: $\{v_0, \ldots, v_r\}$.

Next, to characterize the solutions it is enough to know the minimal transversals of size at most k. In other words, we wish to compute H^{rDk}. We may straight away work only with the rank-limited input hypergraph, that is, we assume that $H^r = H$. Now our problem can be formulated as to compute H^{Dk} for H of rank r.

The section "Assembling peptides into proteins" in [14] discusses the issue of presenting the solution space, H^{Dk} in our notation. The concerns may be summarized as follows. A plain list of all (minimal) transversals is often long and incomprehensible, as it typically consists of many slight variations of similar sets. One may select a few representative transversals based on randomness or heuristics, but this causes information loss and possible misinterpretation, moreover it is not even clear when a vertex (protein) is "identified" as being undoubtedly in the mixture. Some ways to overcome this problem are proposed in [14], but only a few cases are discussed.

Our starting point was the question: Which formats to describe the complete solution space of a shotgun proteomics experiment is both succinct and intuitive, if the list itself is too long from a practical point of view? Secondly, how can we compute such output formats as efficiently as possible from an experimentally given hypergraph and a size bound k?

Quantitative and probabilistic approaches to the inference of proteins with shared peptides also take the abundance and detectability of peptides into account [4], other heuristics only aim at the computation of one most likely mixture relative to some scoring function [12], but in the present paper we are concerned with the *representation* of the set of all possible mixtures

(without quantification) once the hypergraph of peptides and candidate proteins is given.

In inference problems like this one in shotgun proteomics, one is primarily interested in the "identified" proteins that must appear in every solution. In ambiguous cases one would like to know the sets of $s = 2$ proteins such that at least one of these two must be chosen, then the same for $s = 3$, etc., until some maximum s. We argue that H^{DkDs} serves the purpose: From the properties in Section 1.1 it follows immediately that $(H^{DkD})^D = H^{DkDD} = (H^{Dk})^{DD} = H^{Dk}$, in words: The family of minimal transversals of H^{DkD} equals the family of minimal transversals of size at most k in H, i.e., the full solution space we are interested in. Consequently, H^{DkDs} is the hypergraph of the choice sets of at most s elements mentioned above. In particular, H^{DkD1} contains the identified elements.

Thus we propose H^{DkD} as one possible representation of the solution space, and we state the following computational problem for any constants r, s which are thought of as small: *Given a hypergraph H of rank r and a number k, compute H^{DkDs}.* Naturally we are also interested in computing the full H^{DkD}.

Studies on a large set of protein digestion data revealed that H^{DkD} is typically very small and structurally simple, that is, this object meets the aforementioned requirements of being succinct and intuitive. (Actually, the whole idea of computing H^{DkD} and H^{DkDs} was born when doing simulation experiments with the data and assumed protein mixtures.) We are working with a subset of nearly 200,000 proteins from Swissprot, digested *in silico* by the enzyme trypsin. (As the cleavage sites are known, digestion can be simulated.) Then we use only the masses of the resulting peptides and try to reconstruct random mixtures of proteins from their peptide mass spectra. In a typical run, H^{DkD} consists of 5–50 edges (depending on the mixture size), where the vast majority has small sizes, just 1–3 vertices, and more importantly, they are grouped into connected components with single edges or a few edges. In other words, the various solutions have many independent alternatives, and this makes H^{DkD} a particularly well suited representation. (To make the point, consider a toy example where H^{DkD} is merely a graph of k disjoint edges. A plain list of all minimal vertex covers would have length 2^k whereas it is easy to explain to a user that, independently, one vertex from every edge must be taken.)

One might object that H^{DkD} represents the solution space in the same way as input H itself. But there is a crucial difference: One can easily "see" the solutions. Note that it is in general hard to recognize the hitting sets *of a given maximum size* in H (this problem is NP-hard after all!), but everyone who understands the notion of a hitting set can see the hitting sets in H^{DkD} (provided that the latter hypergraph is simple) and thus immediately get a full imagination of the solution space. Otherwise we may drop the large

edges and present H^{DkDs} for some s. Edges in H^{DkD} are either subsets of edges from H, and trivially these have size at most r, or edges that are not subsets of edges from H and reflect further constraints imposed by the size limit k. Thus H^{DkDs} may have some more transversals of size at most k than H, but still the smallest edges are the most informative ones, and we also do not lose solutions.

The following example (contributed by an anonymous referee) illustrates the types of edges. To simplify notation we drop the set parantheses and write edges as "strings" of vertices: Let $H = \{ace, acf, ade, adf, bce, bcf, bde\}$ and $s = 2$. Then $H^D = \{ab, cd, ef, ace\}$, hence $H^{D2} = \{ab, cd, ef\}$. Now H^{D2D} also contains the edge bdf that is not found in H.

Taking H^{DkD} one can also count solutions (see [3] for the use of certain problem kernels for counting transversals), to assess how likely every single protein is, or use biological background knowledge to figure out the most likely solution once the alternatives are nicely presented.

One intention of this paper is to propose a useful formal tool for the application, however, the main technical content is algorithmical. We build on ideas developed in [3]. Since hypergraph dualization is a fundamental problem in many areas, the results may also be interesting there. Such applications are well surveyed in [11] and include: models of formal theories, diagnosis, classical optimization problems like covering, cut sets in networks, phylogeny reconstruction, data mining problems in numerical data, attacking anonymity, key identification, functional dependencies, and query processing in databases, channel assignment in communication systems – to mention some. To avoid an extensive bibliography we refer to [11] for pointers.

1.3 Summary of Results

Our algorithms are given in the framework of fixed-parameter tractability (FPT). A problem with input size n and another input parameter k is in FPT if it can be solved in $O(f(k)\dot{p}(n))$ time where f is any computable function and p is a polynomial. Sometimes we adopt the O^* notation that emphasizes only the superpolynomial and parameterized part of the complexity, that is, we write $O(f(k) \cdot p(n)) = O^*(f(k))$. For general introductions to parameterized algorithms we refer to [5, 15]. We assume familiarity with search tree algorithms and their analysis, the notions of branching vector, branching number, etc.

While Hitting Set in arbitrary hypergraphs is unlikely to be in FPT, Hitting Set is an FPT problem for every fixed rank r. Let $t(r)$ be any number such that an $O^*(t(r)^k)$ time algorithm exists for Hitting Set in hypergraphs of rank r. Bases $t(r)$ considerably smaller than the trivial bound r are known for all $r \geq 2$, see details below in Section 3.

A rather obvious algorithm computes H^{DkDs} in $O^*(r^k)$ time (for any fixed s), this will follow from Proposition 1 below. Our first technical contribution is that we can also compute the same hypergraph in $O^*(t(r)^k)$ time for constant r and s, which reduces the exponential part of the complexity notably. Basically we show that any algorithm for Hitting Set can be used as an oracle, and the number of calls is polynomial, as well as the other auxiliary computations.

Constructing H^{DkD} seems to be intrinsically more difficult. As we will see, the difficulty is to test for the equation $H^{DkDs} = H^{DkD}$, if the former hypergraph is already computed for some s, and this amounts to problem MMHS. Whereas the parameterized complexity of the minimization problem Hitting Set for hypergraphs of rank r and the special case Vertex Cover is well studied, the corresponding max-min problem received much less attention, probably because motivations are less obvious. In our setting MMHS comes out naturally as a subproblem.

Using a result from [6] about output-sensitive enumeration of minimal transversals we can compute H^{DkD} in $O^*(t(r)^k)$ time if the result happens to have a rank smaller than r. For the opposite case it remains open whether the straightforward algorithm can be improved.

The only parameterized result about MMHS we are aware of is an $O^*(2^k)$ time algorithm for MMVC that was announced in [8]. Here we present as another main contribution an $O^*(1.62^k)$ algorithm for MMVC.

There is much related literature about hypergraph dualization (computing H^D) and its applications, see the survey in [11]. Several parameterized cases of hypergraph dualization are treated, e.g., in [7, 11]. In the present work we are obviously interested in enumerations as such, but efficient enumerations of solutions are also useful as a computational tool, in FPT algorithms for the decision or optimization versions of several problems, see [10, 13] for instance.

2 Preliminaries

Let n and h denote the number of vertices and edges, respectively, of our input hypergraph H, and r its rank.

An obvious branching algorithm computes H^{Dk} from H and k in $O^*(r^k)$ time, and explicit enumeration of all minimal transversals cannot be faster in the worst case since, e.g., the hypergraph with k disjoint edges of size r has r^k minimal transversals. For the sake of completeness we sketch the simple algorithm here, although it is implicit in other literature:

Proposition 1 *If H has rank r then H^{Dk} has at most r^k edges and can be computed in $O^*(r^k)$ time.*

Proof. (Sketch.) We generate partial hitting sets P in a search tree. The root represents $P = \emptyset$. If, in a tree node, $|P| = k$ or P is a hitting set, we abort the search path with P as a leaf. The leaf is either a dead end or a solution. If $|P| < k$ and P is not a hitting set, we take any edge e of H with $P \cap e = \emptyset$. For all $v \in e$ we generate the children $P \cup \{v\}$ of P.

Clearly, the leaves eventually contain at most r^k sets, among them repeated, partial, and non-minimal hitting sets that can be recognized and discarded in $O^*(r^k)$ time. It remains to show that we did not miss any minimal transversal P of size at most k. For such P, let Q be a maximal set with the property that $Q \subseteq P$ and Q was generated in our search tree. (Such Q exists, since at least $\emptyset$ is in the search tree.) Assume that Q is a proper subset of P. Since Q is not a hitting set, the algorithm picked some edge e disjoint to Q and generated all $Q \cup \{v\}$, $v \in e$. Moreover, P must contain some vertex $u \in e$. Hence we have $Q \cup \{u\} \subseteq P$, contradicting the choice of Q. □

However, one can find just one (arbitrary) transversal of size at most k faster: Let $t(r)$ be defined as in Section 1.3. It is known that $t(2) \leq 1.2738$ [2], $t(3) \leq 2.076$ [18], and $t(r) \leq r - 1 + 1/(r-1)$ for all r [16]; the latter result was further improved in [9], and generalized to the weighted case. More precisely, the time bounds for Vertex Cover and Hitting Set in these works are at most $O(t(r)^k + kn)$. Further progress in the future is quite possible, but we can notice that $t(r)$ is more than just marginally smaller than r, and since bases are taken to the k-th power, this yields considerable time savings in practice, compared to the simple $O^*(r^k)$ algorithm. In the present paper we achieve the same for our more ambitious double dualization problem.

In our technical results we will use that $t(r) \geq r - 1$ for the $t(r)$ values known so far. A special case is $r = 3$. An algorithm using less than $O(2^k)$ time by A. Soleimanfallah and A. Yeo is announced in [17], but there does not seem to be a published version so far. However, since the $t(r)$ are just upper bounds, we can still redefine $t(3) := 2$ if an improvement comes out.

As a prerequisite we also need the following result shown in [3]:

Theorem 2 *The union of all minimal transversals of size at most k in a hypergraph of rank r has at most $(1+o(1))k^r$ vertices, and it can be computed in $O(k^{r-1}h + k^{2r}t(r)^k)$ time.* □

Since this union is the support of H^{Dk} and H^{DkD} (recall that the support consists of the vertices covered by edges), the number of non-isolated vertices there is bounded by $(1 + o(1))k^r$, no matter what the size of H was.

3 Algorithms for Double Dualization with Rank Limitations

Now we study the complexity of computing H^{DkDs}. First we state a naive algorithm as a benchmark: Using Proposition 1 twice, first compute H^{Dk} in $O^*(r^k)$ time, and from this compute H^{DkDs} in $O^*(k^s)$ time. The latter time bound holds because H^{Dk} has rank at most k. In total we need $O^*(r^k + k^s)$ time which is $O^*(r^k)$ since s is constant. In order to improve this to $O^*(t(r)^k)$ we exploit Theorem 2 and the following simple characterization of H^{DkD}:

Lemma 3 *A set e of vertices is an edge of H^{DkD} if and only if: e is contained in the support of H^{DkD}, no minimal transversal of H with size at most k is disjoint to e, and for every vertex $v \in e$, some minimal transversal of H of size at most k is disjoint to $e - v$.*

Proof. By definition, the support is the union of edges. Moreover, edges e of H^{DkD} are the minimal transversals of H^{Dk}, that is, e must intersect each minimal transversal of H of size at most k, and be minimal with this property. □

Theorem 4 *H^{DkDs} can be computed in $O(k^{r-1}h + k^{rs+1}n + k^{rs}t(r)^k)$ time, if H has rank r.*

Proof. The high-level description of our algorithm is simple: Compute the support of H^{DkD}. Then test every subset of the support with at most s vertices for being an edge of H^{DkD}.

By Lemma 3, e is an edge of H^{DkDs} if and only if e fulfills the conditions stated there, and has at most s vertices. According to Theorem 2, the support has $O(k^r)$ vertices and can be computed in $O(k^{r-1}h + k^{2r}t(r)^k)$ time. Now consider any subset e of the support. We can check in $O(t(r)^k + kn)$ time whether some minimal transversal of H of size at most k is disjoint to e, simply by seeking a minimal transversal of size at most k, in H restricted to $V - e$. The same is done for all $e - v$.

Thus we have to call an $O(t(r)^k + kn)$ time algorithm for Hitting Set $O(k^{rs})$ times. □

Computing H^{DkD} looks harder, note the difference to the previous problem of computing H^{DkDs} with a prescribed s. Here problem MMHS comes into play. To avoid distraction by the lengthy polynomial-time expressions we formulate the following results only in O^* notation.

Lemma 5 *If we can solve MMHS in $O^*(\mu(s)^k)$ time in hypergraphs of rank s, we can also compute H^{DkD} in $O^*(t(r)^k + t(s)^k + \mu(s)^k)$ time, provided that s is the rank of the result.*

Proof. We run the algorithm from Theorem 4 for $s = 1, 2, 3, \ldots$ and test in each iteration whether we have already reached $H^{DkDs} = H^{DkD}$. The equality we have to test can be rephrased, through the following chain of equivalent statements:

- $H^{DkDsD} = H^{DkDD}$ (because of the duality theorem).
- Every minimal transversal of H^{DkDs} is also a minimal transversal of H^{DkD}. (The other direction is, obviously, always true.)
- Every minimal transversal of H^{DkDs} is a transversal of H and has size at most k.

Finally this is equivalent to the negation of:

- H^{DkDs} has a minimal transversal T such that: either (a) T is disjoint to some edge e of H and has size at most k, or (b) T has more than k vertices.

We test the negation of the latter condition. We can exclude the existence of small transversals of type (a) in $O^*(t(s)^k)$ time as earlier, by seeking a transversal of size at most k in H^{DkDs} restricted to each $V - e$. Deciding the existence of minimal transversals of type (b) means precisely to solve MMHS on H^{DkDs}, which is a hypergraph of rank s. Now Theorem 4 yields the time bound. □

We also need a result that was apparently first proven in [6]: For hypergraphs of any fixed rank, the minimal transversals can be enumerated in incremental polynomial time.

Lemma 6 *MMHS for hypergraphs of any constant rank s can be solved (constructively) in $O^*(s^k)$ time.*

Proof. Using [6] we enumerate s^k+1 minimal transversals in $O^*(s^k)$ time, or we stop earlier when there are fewer minimal transversals. In the latter case we can check directly in $O^*(s^k)$ time whether some of them is larger than k. If $s^k + 1$ minimal transversals exist then, by Proposition 1, by Proposition 1, at least one of them is larger than k. □

Note that the result of Lemma 6 cannot be achieved by trivial enumeration of all minimal transversals of size at most k and some further "outlook" steps, because there may be jumps: The next larger minimal transversal might be *much* larger than k.

Theorem 7 *For hypergraphs of any constant rank r we can compute H^{DkD} in $O^*(t(r)^k)$ time, provided that the rank of the result is $s < r$.*

Proof. Combine Lemma 5 and 6, and note that $s \leq r - 1 < t(r)$. □

Note that the straightforward algorithm mentioned in the beginning of this section can also be used to compute H^{DkD} in $O^*(r^k + k^s)$ time. But the algorithm from Theorem 7 has a better time bound in the addressed case $s < r$ which often appears. As soon as the test in Lemma 5 reveals $s \geq r$ in an instance H, we can still switch back to the $O^*(r^k + k^s)$ time algorithm. It remains open whether we can do better if $s \geq r$. Any improvement of Lemma 6 would help immediately.

In the next section we address the special case of MMHS with rank 2. We show that MMVC is solvable in $O^*(1.62^k)$ time. Hence, if H^{DkD} happens to have rank 2, we can verify that $H^{DkD2} = H^{DkD}$ in $O^*(1.62^k)$ time.

4 Maximum Minimal Vertex Cover

An $O^*(2^k)$ time algorithm for MMVC was announced in [8], however not fully explained, and we are not aware of a faster one either. Here we state a different search tree algorithm for MMVC. It is still simple but improves the time bound from [8]. We remark that the algorithm in [8] involves a test whether a partial vertex cover is extendible to some minimal vertex cover, but it is not said how this test shall be performed efficiently. Luckily, our algorithm does not even need such a test, since by construction our partial vertex covers are always extendible (see below).

We use some standard notation: For a vertex u in a graph G, let $N(u)$ be the set of neighbors of u, and $N[u] = N(u) \cup \{u\}$. For a subset X of vertices of G, let $G - X$ denote the graph G without the vertices of X and all incident edges. The degree of a vertex u is the size of $N(u)$.

Theorem 8 *MMVC can be solved in $O^*(1.62^k)$ time.*

Proof. Let C be a variable for a minimal vertex cover in the input graph G, and u be a vertex. We cannot have $N[u] \subseteq C$, since u would be redundant in C, that is, $C - u$ would still be a vertex cover and C not minimal. Hence we can branch on $N[u]$ as follows: For each $v \in N[u]$ we create a branch where $v \notin C$ and therefore $N(v) \subseteq C$. In every branch we are left with the graph $G - N[v]$, and we subtract from k the size of $N(v)$. It is easy to see that the minimal vertex covers C of G with $N(v) \subseteq C$ are exactly the sets $C' \cup N(v)$ where C' is a minimal vertex cover of $G - N[v]$.

Hence our branching rule has split the given instance of MMVC into several smaller instances of the same problem, which are then processed recursively. By an inductive argument it follows that G has a minimal vertex cover larger than k if and only if, in some branch of the resulting search tree, k vertices of G are already put in C but some edges remain

uncovered (or a total of more than k vertices in C is enforced in the last branching step). That is, we can abort any search tree path as soon as the parameter value has dropped to zero.

In the following we use the trivial observation that, if we can split off a connected component of size $O(1)$, then a maximum minimal vertex cover therein can be computed in $O(1)$ time, and no branching is needed for that.

So far we have not specified how to pick a vertex u for branching. Note that the branching vector of the basic rule above is precisely the vector of degrees of the vertices in $N[u]$. Let us choose some u with minimum degree d. Then the branching vector has length $d+1$, with all entries being at least d. The worst case, for fixed d, is a branching vector with $d+1$ entries d, giving the branching number $\sqrt[d]{d+1}$. This expression is a monotone decreasing function of d. Hence the worst case altogether is $d=1$, that is, branching vector $(1,1)$, and we are back to the old $O^*(2^k)$ bound. However, we can easily improve this bottleneck case: Let u be a vertex of degree 1, and v its only neighbor. If v has degree 1 as well, then u and v form a connected component. But if v has at least one more neighbor, the branching vector is $(1,2)$ with branching number 1.62.

Now the bottleneck case is $d=2$ and branching vector $(2,2,2)$, with branching number 1.74, but we can further improve on this case as well. Consider a vertex u of degree 2, with neighbors v,w of degree 2. Let x and y be the other neighbor of v and w, respectively. If $x=w$ and $y=v$ (in simpler words: vw is an edge) then u,v,w form a connected component of size $O(1)$ that is processed in time $O(1)$, as said above. If $x=y$ then simple case inspection in the cycle x,v,u,w shows that we can always decide on either $u \notin C$, $v,w \in C$ or $v,w \notin C$, $u,x \in C$. (All other cases would insert a redundant vertex into C.) The branching vector is $(2,2)$ with branching number 1.42. If $x \neq y$ then we have a path x,v,u,w,y. Again, simple case inspection yields that we need to consider only the following four branches; vertices that are chosen are displayed in bold:

(i) x–**v**–u–**w**–y
(ii) **x**–v–**u**–w–**y**
(iii) **x**–v–**u**–**w**–y
(iv) x–**v**–u–w–**y**

The branching vector $(2,3,3,3)$ has branching number 1.68, however, at least one of x,y must have further neighbors (or u,v,w,x,y form a connected component). Hence we must add to C another neighbor of y in branch (iii), or another neighbor of x in branch (iv). In both cases this improves the branching vector to $(2,3,3,4)$ and the branching number to 1.62. Note carefully that, for every vertex we put in C, we have also excluded some neighbor from C. This guarantees that no selected vertex can become

redundant by vertices added to C later on.

Above we have achieved, for any neighborhood $N[u]$ with degrees $(2,2,2)$, a branching number 1.62 or better. If the degrees in $N[u]$ are $(2,2,3)$ or larger, we take $(2,2,3)$ directly as the worst branching vector and obtain 1.62, too. Finally note that the branching numbers for $d \geq 3$ are all better than 1.59, because the worst case among them is $(3,3,3,3)$. □

Further refinement of the "1.62 bottleneck cases" seems possible, but we leave it here. Another conjecture is that the "$\mathcal{C}$-cover" schem used in [13] may lead to a faster MMVC algorithm. The polynomial term in Theorem 8 is obviously moderate. We skip the tedious analysis and just remark that a vertex with minimum degree can be found quickly in each step, using a data structure that updates the degrees and puts vertices of equal degree in a designated queue. Also note that it is enough to work in graphs with $O(k^2)$ vertices, since in larger graphs MMVC always has a positive answer [8]. In other words, a problem kernel is immediately given.

A natural question now is whether MMHS for hypergraphs H of rank $s \geq 3$ can be solved in less than $O^*(s^k)$ time. This appears difficult even for $s = 3$ (as opposed to the minimization problem Hitting Set where such improvements are easy to obtain). We cannot simply generalize our approach for MMVC to MMHS, since even for rank 3 we do not have these enforced decisions in the neighborhood of a decided vertex. Another approach to this open question comes into mind: Remember that H has no minimal transversal larger than k if and only if $H^{DkD} = H$. The latter equation implies $H^{DkDs} = H$ since H has rank s. Moreover H^{DkDs} is computable in $O^*(t(s)^k)$ time using Theorem 4. However, to turn this observation into an MMHS algorithm we would also need the reverse direction: Does $H^{DkDs} = H$ imply that all minimal transversals of H have at most k vertices?

5 Conclusions and Further Research

We found that alternating applications of the operators dualization (computing the transversal hypergraph) and rank limitation (erasing all hyperedges larger than a given threshold) has interesting applications and gives rise to new algorithmic questions. It is also intimately related to the max-min version of transversal computation. The algebra of these mixed operators is not yet fully understood, and some open questions were mentioned. Our main algorithmic insight was that, for hypergraphs of any constant rank, computations are only polynomially harder than finding just one minimal transversal (Theorem 4). Still these polynomial factors can be significant, they depend on the exact size of the union of all minimal transversals of size at most k. As we pointed out in [3], this quantity is known only subject

to a constant factor that is widely undetermined. Progress on this purely combinatorial question would give a better idea of the practical runtimes to expect. Another obvious goal is to improve the complexity bounds for the other subproblems we considered. In the protein inference application, more studies of examples may reveal further types of solution spaces where other succinct representations are better suited.

Acknowledgment

The author would like to thank: Ferdinando Cicalese for providing protein digestion data, Leonid Molokov for the simulation experiments and discussions that inspired this work, and the anonymuous referees for a number of helpful comments. Support has been received from the Swedish Research Council (Vetenskapsrådet), grant 2007-6437, "Combinatorial inference algorithms – parameterization and clustering".

References

[1] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*, North Holland Mathematical Library, vol. 45. Elsevier, 1989

[2] J. Chen, I.A. Kanj, G. Xia. Improved parameterized upper bounds for vertex cover, *31st Symp. on Math. Foundations of Computer Science MFCS 2006*, *LNCS* 4162, 238-249

[3] P. Damaschke, L. Molokov. The union of minimal hitting sets: Parameterized combinatorial bounds and counting, *Journal of Discrete Algorithms* 7 (2009), 391-401

[4] B. Dost, V. Bafna, N. Bandeira, X. Li, Z. Shen, S. Briggs. Shared peptides in mass spectrometry based protein quantification, *13th Conf. on Research in Comp. Molecular Biology RECOMB 2009*, *LNCS* 5541, 356-371

[5] R.G. Downey, M.R. Fellows. *Parameterized Complexity*, Springer, 1999

[6] T. Eiter, G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems, *SIAM Journal on Computing* 24 (1995), 1278-1304

[7] K.M. Elbassioni, M. Hagen, I. Rauf. Some fixed-parameter tractable classes of hypergraph duality and related problems, *3rd Int. Workshop on Parameterized and Exact Computation IWPEC 2008*, *LNCS* 5018, 91-102

[8] H. Fernau. Parameterized algorithmics: a graph-theoretic approach, Habilitation thesis, Univ. Tübingen, 2005

[9] H. Fernau. Parameterized algorithms for hitting set: The weighted case, *6th Italian Conf. on Algorithms and Complexity CIAC 2006*, *LNCS* 3998, 332-343

[10] H. Fernau. Edge dominating set: efficient enumeration-based exact algorithms, *2nd Int. Workshop on Parameterized and Exact Computation IWPEC 2006*, *LNCS* 4169, 142-153

[11] M. Hagen. Algorithmic and computational complexity issues of MONET, PhD thesis, Univ. Jena, 2008

[12] Z. He, Ch. Yang, C. Yang, R. Qi, J.P.M. Tam, W. Yu. Optimization-based peptide mass fingerprinting for protein mixture identification, *13th Conf. on Research in Comp. Molecular Biology RECOMB 2009*, *LNCS* 5541, 16-30

[13] D. Mölle, S. Richter, P. Rossmanith. Enumerate and expand: New runtime bounds for vertex cover variants, *12th Conf. on Computing and Combinatorics COCOON 2006*, *LNCS* 4112, 265-273

[14] A.I. Nesvizhskii, R. Aebersold. Interpretation of shotgun proteomic data: The protein inference problem, *Molecular and Cellular Proteomics* 4 (2005), 1419-1440

[15] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press 2006

[16] R. Niedermeier, P. Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set, *Journal of Discrete Algorithms* 1 (2003), 89-102

[17] F. Rosamond (ed.). Parameterized Complexity News 4 (2009)

[18] M. Wahlström. Algorithms, measures, and upper bounds for satisfiability and related problems, PhD Thesis 1079, Linköping Studies in Science and Technology, 2007