Theoretical Computer Science 452 (2012) 39–46

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Parameterized reductions and algorithms for a graph editing problem that generalizes vertex cover☆

Peter Damaschke *, Leonid Molokov

Department of Computer Science and Engineering, Chalmers University, S-41296 Göteborg, Sweden

ARTICLE INFO

Article history:
Received 30 August 2011
Received in revised form 5 May 2012
Accepted 12 May 2012
Communicated by J. Díaz

Keywords:
Vertex cover
Hitting set
Graph editing
Error correction
Parameterized complexity
Problem kernel

ABSTRACT

We study a novel generalization of the Vertex Cover problem which is motivated by, e.g., error correction (data cleaning) prior to inference of chemical mixtures by their observable reaction products. We focus on the important case of deciding on one of two candidate substances. This problem has nice graph-theoretic formulations situated between Vertex Cover and 3-Hitting Set. In order to characterize its parameterized complexity we devise parameter-preserving reductions, and we show that some minimum solution can be computed faster than by solving 3-Hitting Set in general. More explicitly, we introduce the Union Editing problem: In a hypergraph with red and blue vertices, edit the colors so that the red set becomes exactly the union of some hyperedges. The case of degree 2 is equivalent to Star Editing: in a graph with red and blue edges, edit the colors so that the red set becomes exactly the union of some stars, i.e., vertices with all their incident edges. Our time bound is $O^*(1.84^c)$ where c denotes the total number of recolored edges.

1. Introduction

1.1. Definitions

A computational problem with input size n and another input parameter k is fixed-parameter tractable (FPT) if it can be solved in $p(n) \cdot f(k)$ time where p is a polynomial and f any function. Since we focus on the $f(k)$ factor, we adopt the $O^*(f(k))$ notation that suppresses polynomial factors. (The polynomial factor cannot be neglected in practice, but usually it is moderate, such that $f(k)$ "dominates" the complexity.) We deal with graph problems and denote by n the number of vertices. For introductions to parameterized algorithms we refer to [8,15]. However, for readers not being familiar with the basic notions of FPT we introduce some of them very briefly. Many FPT algorithms follow the *bounded search tree* paradigm: in a branching step, alternative choices are made regarding some items of the solution to a given problem, and the problem instance is split into several instances (branches) that are further processed independently and recursively, resulting in a tree of instances. Branching rules must be designed in such a way that the parameter value is properly reduced in every branch. The *branching vector* indicates how much the parameter is reduced in every branch. For instance, branching vector (2, 2, 3, 3, 4) means that 2 is deducted from the parameter in two branches, 3 is deducted in two other branches, and 4 is deducted in yet another branch. The *branching number*, informally, is the base x in the number x^k of leaves of the search tree when the initial parameter value is k. It can be calculated via the characteristic equation of the branching vector. For instance, the characteristic equation of (2, 2, 3, 3, 4) is $x^k = 2x^{k-2} + 2x^{k-3} + x^{k-4}$, or simpler $x^4 = 2x^2 + 2x + 1$, with the

☆ This is an extended version of a conference paper that appeared in: *12th Algorithms and Data Structures Symposium WADS 2011*, New York, *Lecture Notes in Computer Science* (Springer), vol. 6844, pp. 279–289.

* Corresponding author. Tel.: +46 31 772 5405; fax: +46 31 772 3663.
E-mail addresses: ptr@chalmers.se (P. Damaschke), molokov@chalmers.se (L. Molokov).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.05.031

solution $x < 1.84$. For brevity, a branching rule is said to be a *bv rule* if its branching number is less than or equal to that of branching vector bv.

The following is a classical FPT problem, and a tremendous amount of work has been devoted to its parameterized complexity.

Vertex Cover: in a graph, find a set C of at most k vertices being incident to all edges.

A hypergraph is a vertex set equipped with a family of subsets of vertices called hyperedges. The degree of a vertex is the number of hyperedges it belongs to, and the degree of a hypergraph is the maximum vertex degree. We introduce the following problem.

Union Editing: given a hypergraph whose vertices are colored red and blue, paint at most k blue vertices red, and paint at most m red vertices blue, such that the set of red vertices becomes exactly the union of some of the hyperedges.

Note that the hyperedges in that union are in general not disjoint. Union Editing (Degree r) is, as the name suggests, the Union Editing problem on hypergraphs of degree at most r. The generalization of Vertex Cover to hypergraphs is known as

Hitting Set: in a hypergraph, find a set C of at most k vertices intersecting all hyperedges.

The rank of a hypergraph is the maximum size of its hyperedges. r-Hitting Set is the Hitting Set problem for rank r.

1.2. Motivation

Union Editing arises from chemical analytics. In fact, it was proposed in [13], however without algorithmic analysis. Every hyperedge represents a possible substance in an unknown mixture, and the vertices therein represent the reaction products that can be directly identified by experiments. For instance, unknown protein mixtures can be analyzed by splitting the proteins enzymatically into peptides which are then identified by, e.g., mass spectrometry. A database of proteins and their peptides is finally used to identify the proteins from the set of peptides. Many peptides, especially those with large masses, are unique for a protein, while others appear in r different proteins, very often just $r = 2$. The problem to resolve these ambiguities [7,11,14] is easily seen to be the Set Cover or Hitting Set problem, as also observed by biologists. Case $r = 2$ is Vertex Cover.

However, here we do not address this inference step but another problem that appears prior to the actual mixture reconstruction. Ideally, the set of observed vertices should be exactly the union of some of the hyperedges (those representing the substances in the mixture). But in practice, observations may be corrupted in two ways: by at most k vertices that should appear but are not observed, and by at most m vertices that are observed but should not appear. (Our red vertices are the observed ones.) The question arises, how we can efficiently correct a limited number of such errors, assuming that reliable bounds k and m are known from experience. Different problem versions may be considered: decide the existence of a solution with k and m errors, enumerate them all, or enumerate all vertices being recolored in some solution (the error candidates).

1.3. Contributions and related work

Union Editing (Degree 1) is a trivial problem, in a sense: as the hyperedges are pairwise disjoint, we only have to decide for every hyperedge whether to color it completely red or blue. For each pair (k, m) we can solve the problem in polynomial time by dynamic programming, as this is essentially a knapsack problem with unary representations of numbers. We can even enumerate all possible solutions in an implicit way, as the system of all paths in a certain directed graph of partial solutions. All this is a straightforward exercise in dynamic programming, therefore we skip the details. In this paper we mainly deal with Union Editing (Degree 2) which is the "smallest nontrivial case" but is already important in the intended application.

In Section 2 we introduce several equivalent formulations as (hyper)graph editing problems. We relate them to each other and to established problems. These problems are NP-complete but also in FPT, with the solution size as the parameter. Union Editing (Degree 2) turns out to be a special case of 3-Hitting Set. In Section 3 we give a parameterized $O^*(1.84^{k+m})$ time algorithm for minimizing $k + m$ (corresponding to the total number of error corrections). This is clearly faster than the state-of-the-art result for 3-Hitting Set in general. The gain is not very high, nevertheless the result indicates that our problem is also an interesting subcase from an algorithmic point of view. We can also compute, within the same time bound, a kernel that contains all optimal solutions. Enumeration results and kernels are given in Section 4, and Section 5 concludes the paper with open questions.

The currently fastest Vertex Cover algorithm [3] runs in $O^*(1.2738^k)$ time. Several other Vertex Cover variants and generalizations addressed in the literature [1,9,10,12,16] include vertex covers with additional constraints and partial vertex covers. One of our problem formulations is a case of partial vertex covers where, however, the number of uncovered edges is also limited by a parameter.

2. Parameterized graph problems equivalent to star editing

In the following we translate Union Editing (Degree 2) into an equivalent graph problem. Since we are then on more familiar grounds, this will make it easier to characterize the complexity of the problem. Our graphs may contain loops and

parallel edges. In a graph, we call the set of *all* edges incident to some vertex v the *star with center* v. (This should not be confused with other notions of "star" in graph theory.) Specifically we define:
Star Editing: in a graph with red and blue edges, recolor at most k blue edges and m red edges, such that the union of some stars becomes exactly the set of red edges, or equivalently, the edge set of some induced subgraph becomes exactly the set of blue edges.

Theorem 1. Union Editing (Degree 2) *and* Star Editing *are equivalent, through some polynomial-time reduction that preserves parameters k and m.*

Proof. Given any hypergraph H of degree 2, we construct a graph G as follows. For every hyperedge of H create a vertex of G. Every vertex of H that belongs to the intersection $u \cap v$ of hyperedges u and v becomes an edge between u and v in G. (Thus, parallel edges may appear.) Every vertex of H that belongs to only one hyperedge v becomes a loop at v in G. This defines a one-to-one correspondence between hypergraphs of degree 2 and graphs with loops and parallel edges, as we can also reconstruct H from G. Every edge of G becomes a vertex of H, and every star in G becomes a hyperedge in H. Obviously, the solutions of the two problems in H and G correspond to each other. □

Alternatively we may view Star Editing as a vertex selection problem, rather than an edge selection problem. This is what we show next.
Deficient Vertex Cover with Cost Edges: in a graph with red and blue edges, find a subset C of vertices incident to at most k blue edges and non-incident to at most m red edges.

This problem extends the ordinary Vertex Cover problem, in graphs with only red edges, in two directions: m red edges may remain uncovered by C, and the "cost" k of C is the number of incident blue edges. In the special case when all blue edges are loops, they merely encode an integer-valued cost function on the vertices. The new twist is that any two vertices u and v joined by a blue edge uv share one unit of cost, in the sense that only one of them, say u, pays for this blue edge when $u \in C$, while the cost of adding v to C is reduced by one. Because of this role of the blue edges we also call them *cost edges*. These pairwise dependencies of costs have no counterpart in the ordinary Vertex Cover problem. Also note that we may add to any vertex cover C, at zero cost, vertices whose incident blue edges are already incident with C.

Let Vertex Cover with Cost Edges denote the special case when $m = 0$. We remark that the other natural special case, Deficient Vertex Cover where all blue edges are loops, was also studied in [1], though in a weighted version and under a different name. Both FPT and W-hardness results are shown in [16] for another interesting Vertex Cover generalization called Vector Dominating Set where the number of uncovered incident edges is individually prescribed for each vertex.

Theorem 2. *Problems* Star Editing *and* Deficient Vertex Cover with Cost Edges *are equivalent, via a polynomial-time reduction that preserves the parameters k and m.*

Proof. Consider a set C that solves Deficient Vertex Cover with Cost Edges for parameters k and m. By recoloring the blue edges incident to C and the red edges non-incident to C we get a solution to Star Editing for parameters k and m, where C is the set of centers of stars whose union consists of exactly the red edges. For the opposite direction, consider a solution to Star Editing where at most k blue and m red edges are recolored. Then, the set C of all centers of entirely red stars, after recoloring, solves Deficient Vertex Cover with Cost Edges. □

Vertex Cover with Cost Edges still appears as a proper generalization of Vertex Cover, so the following reduction to Vertex Cover might be a little surprising.

Definition 3. A *conflict triple* as a set of three edges: two blue edges uv and xy joined by a red edge vx, where $v \neq x$. The other vertices are not necessarily distinct, i.e., we can have $u = y$, or the blue edges may be loops, or parallel to vx.

The *red degree* and *blue degree* of a vertex is the number of incident red edges and blue edges, respectively, where a loop counts only once.

Theorem 4. Vertex Cover *and* Vertex Cover with Cost Edges *are equivalent, through some polynomial-time reductions that preserve parameter k.*

Proof. The reduction from Vertex Cover to Vertex Cover with Cost Edges is trivial, as already mentioned: Attach to every vertex a blue loop that encodes the vertex cost.

Conversely, consider any instance of Vertex Cover with Cost Edges, i.e., an edge-colored graph G and a parameter k. If a red loop is attached to some vertex v, clearly we must put v in the solution C; moreover we can remove v and all incident edges and subtract from k the blue degree of v. If parallel red and blue edges join two vertices u and v then some of u and v must eventually be in C, hence we can immediately remove the blue edges uv and subtract their number from k. After these data reductions, G has neither red loops nor blue edges parallel to red edges.

Now we construct a graph G' as follows. Every blue edge of G becomes a vertex of G'. For any conflict triple uv, vx, xy, we create an edge between vertices uv and xy in G'. Note that the same red edge vx may give rise to several edges of G'; we call them copies of vx.

Consider any vertex cover C of cost k in G. Let C' be the set of blue edges incident to C. Observe that C' has size k and is a vertex cover in G'. For any red edge vx in G, we have $v \in C$ or $x \in C$, by symmetry assume $v \in C$. Hence all blue edges incident with v are in C'. It follows that all copies, in G', of the edge vx are covered by vertices from C'.

To see the opposite direction of the equivalence, consider any vertex cover C' of size k in G'. For every red edge vx from G, all blue edges at v or all blue edges at x must belong to C', in order to cover all copies of the edge vx in G'. Define C as the set of all vertices v of G such that all blue edges incident to v are in C'. Due to the sentence before, C is a vertex cover in G. The cost of C is the number of incident blue edges, which is at most k, as these edges were in C'. □

Recall the r-Hitting Set problem. Next we refine it to a two-parameter problem which is also closely related to the problems discussed here.

s, r-Hitting Set: given is a hypergraph whose vertices are colored red and blue, where every hyperedge consists of at most s blue and r red vertices. We also say that the *blue rank* and *red rank* is s and r, respectively. Find a hitting set C of at most k blue and m red vertices.

Theorem 4 essentially relies on $m = 0$. For $m > 0$ we do not see a parameter-preserving reduction from Deficient Vertex Cover with Cost Edges to Vertex Cover. It seems that Vertex Cover with both missed edges and cost edges is intrinsically more difficult. However we can reduce it to the next higher problem in the "hitting set hierarchy":

Theorem 5. Deficient Vertex Cover with Cost Edges *is reducible to* 2,1-Hitting Set *through some polynomial-time reduction that preserves the parameters k and m.*

Proof. Every edge of the given graph G becomes a vertex of a hypergraph H. The hyperedges of H are the following sets of size 2 or 3: every red loop with every incident blue edge; every red edge with every parallel blue edge; and every conflict triple. We call the first two cases *conflict pairs*.

Let C be any solution to Deficient Vertex Cover with Cost Edges in G. We claim that the set F consisting of all blue edges incident to C and all red edges non-incident to C is a hitting set in H. To prove the claim, consider any red edge vx. Assume that some end vertex is in C, say $v \in C$. If vx forms a conflict pair with some blue edge, then either $v = x$ (red loop), or vx and the blue edge are parallel. In both cases the blue edge is incident to C, thus F intersects the conflict pair. If vx forms a conflict triple with blue edges uv and xy, then the blue edge uv is incident to C, too. If $v, x \notin C$ then $vx \in F$. Note that F intersects every conflict pair/triple containing vx.

Conversely, let F be any hitting set of k blue edges and m red edges in H. We construct a vertex set C in G and show that C is incident to at most k blue edges and non-incident to at most m red edges. Let vx be any red edge with $vx \notin F$. If $v = x$, we put this vertex in C. Note that the incident blue edges are all in F, since F intersects all conflict pairs with the red loop vx. In the following let be $v \neq x$. All blue edges at v or all blue edges at x are in F, since otherwise some conflict triple with vx in the middle, or some conflict pair with a parallel blue edge vx, would be disjoint to F. We put v in C if all blue edges incident to v are in F, and similarly for x. The construction of C implies: a blue edge is incident to C only if it belongs to F. Similarly, a red edge is non-incident to C only if it belongs to F. □

3. Solving Star Editing faster than 3-Hitting Set

We defined Star Editing as a problem with two separate parameters k and m. Nevertheless, let us consider the simplest version of the problem where we only want to find some solution minimizing the total number $k + m$ of edits. In the application mentioned in Section 1 this means to find the minimum number of error corrections that would possibly explain the data.

By Theorems 2 and 5 we can reduce this problem to 2,1-Hitting Set, and trivially we can further reduce it to 3-Hitting Set, since colors can be ignored as long as we only aim for a minimum $k + m$. Thus, some solution can be found by using any parameterized algorithm for 3-Hitting Set, such as the $O^*(2.076^{k+m})$ time algorithm from [17]. (Slightly faster algorithms have been announced but are apparently unpublished.) However, the hypergraphs of rank 3 from our reduction enjoy a special structure; thus we might be able to solve Star Editing significantly faster than 3-Hitting Set in general. In fact, we will now devise an $O^*(1.84^{k+m})$ time branching algorithm.

We use the Deficient Vertex Cover with Cost Edges formulation which is more convenient for stating the branching rules. Accordingly, we use C as a generic variable for a solution. We describe our algorithm as a list of rules, and we always apply the first applicable rule in the list. We denote our input graph by $G = (V, E)$.

(1) We always remove every edge incident to some vertex just added to C, and we subtract from parameter k the number of incident blue edges. If we decided not to put some vertex v in C, we turn every incident edge into a loop at its other end vertex. If both vertices of a red edge (or the vertex of a red loop) are not put in C, we remove this red edge and subtract 1 from parameter m. These trivial actions are silently added to the rules below.

(2) If two parallel red edges connect some vertices v and x, then removing one of them can only lower the deductions from parameter m in branching rules, in those branches where $v, x \notin C$. Hence, in a worst-case analysis we can safely assume that no parallel red edges exist. (That is, for bookkeeping of parameter m we actually keep them in the graph, but otherwise we ignore all edges but one, of any bundle of parallel red edges.)

(3) If some vertex v has red degree 0, we immediately decide $v \notin C$. A vertex v with blue degree 0 is put in C at no cost. Thus we can assume henceforth that all red and blue degrees are positive.

(4) Denote by V_1, V_2, V_3 the set of vertices with blue degree 1, 2, and at least 3, respectively. Since by (3) all blue degrees are positive, we have $V = V_1 \cup V_2 \cup V_3$. If some red edge vx connects two vertices $v \in V_2$ and $x \in V_3$, we decide whether

$v, x \notin C$ or $v \in C$ or $x \in C$. (The last two branches do not rule out insertion of the other vertex in C later on.) Obviously this gives us a (1, 2, 3) rule, and its branching number evaluates to 1.84. A red edge within V_3, either a normal edge or a loop, gives even better branching numbers, as is easy to check. When these rules are exhaustively applied, we obtain a graph with red edges only within $V_1 \cup V_2$ and between V_1 and V_3.

(5) Next we make a key observation that holds only because this problem version asks for some (arbitrary) solution with minimum total cost $k + m$. – Consider any red edge vx with $v \in V_1$. If we decide $v, x \notin C$, this red edge costs 1. If we take $v \in C$ instead, we pay 1 for the blue edge incident to v, and this decision is no worse than the case $v, x \notin C$. This exchange argument shows that we can ignore the branch $v, x \notin C$. Due to this domination rule we can set the red edge vx *permanent*: that is, we commit to put v or x in C, but we defer the actual choice of v or x. The same reasoning holds for a red loop at any $v \in V_1$, but there we can decide $v \in C$ instantly. Now all red edges except those in V_2 are permanent and non-loops.

(6) As long as some red edge is incident with V_3, being permanent due to (5), we obviously have a (1, 3) rule. By exhaustive application of this rule, recall also (1) and (3), we eventually make $V_3 = \emptyset$. Similarly, as long as some red edges connect V_1 and V_2, they are permanent as well, which gives us a (1, 2) rule with branching number 1.62. After exhaustive application of this rule, our graph has blue degree at most 2, and red edges only within V_2 and within V_1, and those within V_1 are permanent.

(7) If any two vertices $u, v \in V_2$ are joined by two parallel blue edges, we can assume that both u and v or none of them are in C, because if we put one vertex in C, we can add the other vertex for free. Hence we can merge u and v, thereby turning the blue edges in two blue loops at the new vertex. Parallel red edges are already excluded by (2). (Alternatively we might argue as follows: if any two vertices $u, v \in V_2$ are joined by two parallel red edges, we have a (2, 2, 2) rule with branching number 1.74.) Thus we get rid of parallel edges of equal color in V_2.

(8) Consider any $w \in V_2$ of red degree at least 2, and let wu and wv be red edges. Due to the structure already established, we have $u, v \in V_2$ and $u \neq v$.

(8.1) Suppose that no blue edge uv exists. Now we can decide $w \in C$ with cost 2, or $w \notin C$. In the latter case we also make the decisions for u and v. For each of them the cost is either 2 for the incident blue edges (if we put the vertex in C), or 1 for the red edge (if we don't). Since the blue edges incident to u and v are distinct, this yields a (2, 2, 3, 3, 4) rule with branching number 1.84.

Now we can assume that for every $w \in V_2$ of red degree at least 2, and any two of its neighbors (via red edges) $u, v \in V_2$, exactly one blue edge uv exists.

(8.2) Suppose that some $w \in V_2$ has red degree at least 3. Let N be the set of neighbors joined to w by red edges. Any two vertices of N are joined by a blue edge. Since w and N are all in V_2, it also follows $|N| = 3$. Again we can decide $w \in C$ with cost 2, or $w \notin C$. In the latter case we also make the decisions for N. If we put two vertices of N in C, we can also take the third, as all blue edges incident to N are already paid. Together this yields the branching vector (2, 3, 3, 4, 4, 4) rule with branching number smaller than 1.77.

(8.3) Now all vertices in V_2 have red degree 1 or 2, hence the red edges form simple paths and cycles. Note that every red cycle is interweaved with one or two blue cycles, and it forms a connected component in the entire red–blue graph, so that the problem restricted to each cycle can be solved trivially. What remains are simple red paths. Let y be an end of a path of at least two red edges. Denote by yx and yz the two incident blue edges, and by yw the incident red edge. Note that one of x and z is the next vertex following w on the red path. If $x \notin C$ or $z \notin C$, there is no reason to have $y \in C$, since y would cover only one red edge but would have to pay for some blue edge. By contraposition, if we put $y \in C$ then we also take $x, z \in C$. The cost is at least 3 for the incident blue edges. If we decide $y \notin C$ then we also decide about w. The cost is 1 if $w \notin C$, and 2 if $w \in C$. Hence a (1, 2, 3) rule with branching number 1.84 is available.

By applying these rules exhaustively, all vertices in V_2 get red degree 1.

(9) As said in (2), we can suppose that no parallel red edges exist in V_1 (in particular). Next assume that some edge uv with $u, v \in V_1$ is blue. As earlier, if we put either of u and v in C, the other vertex can be added to C for free, hence we merge u and v and shrink uv to a blue loop.

(10) Consider any $w \in V_1$ of red degree at least 2, and let wu and wv be red edges. From (6) we have that $u, v \in V_1$ and $u \neq v$, and these red edges are permanent. Hence we must take $w \in C$ or $u, v \in C$. Since no blue edge uv exists due to (9), vertices u and v together are incident to two blue edges, hence this branching rule is a (1, 2) rule.
It is helpful to summarize the current situation. All vertices are now in $V_1 \cup V_2$. All red degrees are 1, by (10) and (12), i.e., the red edges form a matching. Red edges exist only within V_2 and within V_1, cf. (6). All red edges within V_1 are permanent. Blue edges exist only within V_2, between V_2 and V_1, and as blue loops in V_1.

(11) Blue loops in V_1 are removed as follows. When some $v \in V_1$ has a blue loop attached, and vx is a red edge (note that $x \in V_1$), we can safely decide $x \in C$, since the option $v \in C$ is only worse.

(12) Let vx be a red edge in V_1. Since vx is permanent, we have to put some vertex in C, say $v \in C$, the other case is symmetric. Vertex v is also involved in a blue edge uv, where $u \in V_2$. Vertex u in turn is incident to some red edge yu, where $y \in V_2$. Since we decided $v \in C$, the blue degree of u drops to 1, thus u moves to V_1, so that we can make uy permanent, using (5). Since we now have to choose $u \in C$ or $y \in C$ which has blue degree 1 and 2, respectively, this gives us a (1, 2) rule. We argue similarly in the symmetric case starting with $x \in C$. This way we have appended a (1, 2) rule to both branches of a (1, 1) rule. Altogether this makes a (2, 2, 3, 3) rule. Exhaustive application of this rule empties V_1.

After application of all the preceding rules, it remains a graph where every vertex has exactly red degree 1 and blue degree 2. At this point, an optimal solution consists in not adding any further vertices to C. The cost of this claimed optimal solution is the number of remaining red edges. We consider any different solution that adds something to C and show that, in fact, it cannot be cheaper: Every vertex added to C reduces the cost of the red edges by at most 1 but has to pay for two blue edges. And trivially, at most two vertices in C can share the cost of a blue edge. Consequently the total cost has not improved. Since 1.84 was the worst branching number among our rules, this finally shows:

Theorem 6. *A solution to* Deficient Vertex Cover with Cost Edges *(or* Star Editing*) with minimal $k + m$ can be found in $O^*(1.84^{k+m})$ time.* □

4. Enumerations and problem kernels

The considered optimization version of our problem only asks for *some* solution with minimum $k + m$. But the optimal solution might not be unique, and in the error correction application one cannot simply assume that an arbitrary optimal solution explains the data correctly. It would be more appropriate to return all possible minimum solutions, but if these are exponentially many, this raises new issues. A nice compromise is to return just *the set of all edges that are recolored* in all possible solutions with minimum $k + m$, i.e., the potential errors. Enumerating these edges does not cost essentially more time than finding one solution:

Theorem 7. *All edges recolored in all solutions to an instance of* Star Editing *with minimum $k+m$ can be found in $O^*(1.84^{k+m})$ time.*

Proof. The basic idea is natural, only its correct implementation needs a little bit of care. Let $c = k + m$ be the minimum number of recolorings. For every edge $e = uv$ we test from scratch whether there exists a solution where e is recolored, and hence $c - 1$ other edges are recolored. In the following we use the equivalent Deficient Vertex Cover with Cost Edges formulation again.

If e is red, we color it blue, that is, we decide $u, v \notin C$, and apply the postprocessing steps mentioned above in step (1). If e is blue, we color it red and mark it permanent. In both cases we solve the residual problem with parameter value $c - 1$. To handle the latter case we have to extend the Deficient Vertex Cover with Cost Edges problem in yet another direction. We allow permanent red edges already in the input graph. Now we argue that this generalization can still be solved in $O^*(1.84^{k+m})$ time, using the algorithm from Theorem 6 with slight modifications. In all branchings we abandon the branches with $u, v \notin C$, if there are some. Trivially, deletion of some branches can only improve the branching numbers. Once we reach a graph where all vertices have red degree 1 and blue degree 2, and some red edge is permanent, we clearly have a $(2, 2)$ rule and can continue. All other situations are resolved as in Theorem 6. □

Next it is natural to ask how many edges can be affected in total. The following result is based on an old Vertex Cover kernelization [2]; however the two colors make it more tricky.

Theorem 8. *All solutions to an instance of* Star Editing*, with a given $k + m$, are together contained in a set of $O((k + m)^2)$ red and $O((k + m)^3)$ blue edges, being incident to $O((k + m)^2)$ vertices, which can be computed in polynomial time.*

Proof. First, remember a few trivial reduction rules: all blue degrees are positive, otherwise we can put the gratis vertices in C. No blue edge joins two vertices of blue degree 1, otherwise we can contract it to a blue loop. Now any p vertices are incident to at least $2p/3$ blue edges (including loops), where in the worst case the blue edges pair up to components of two incident edges. Also the red degrees are positive.

Let c be a maximum allowed cost $k + m$ of solutions C. If some vertex v is incident to $2.5c + 1$ or more red edges, and we decide $v \notin C$, then $1.5c + 1$ of its neighbors must be taken, in order to leave at most c red edges uncovered. But these $1.5c + 1$ neighbors are incident to at least $c + 1$ blue edges, thus too expensive. Hence $v \in C$ is enforced. If some vertex v is incident to $c + 1$ or more blue edges, clearly $v \notin C$ is enforced. Hence we obtain, in polynomial time, a kernel where all vertices have red and blue degrees bounded by $2.5c$ and c, respectively, and where still any p vertices are incident to at least $2p/3$ blue edges. Thus at most $1.5k \leq 1.5c$ vertices can be put in C, which are able to cover at most $1.5c \cdot 2.5c = 3.75c^2$ red edges, and $m \leq c$ red edges may exist in addition. It follows that at most $7.5c^2 + 2c$ vertices and $7.5c^3 + 2c^2$ blue edges are in the kernel. □

Since our problem generalizes Vertex Cover, the quadratic bound cannot be improved, due to our result for Vertex Cover in [6]. However, it remains open whether a cubic number of blue edges is needed.

Next we compute a kernel for all minimal solutions of s, r-Hitting Set, extending a result from [4] for r-Hitting Set. Naive application of the earlier result would only yield an $O((k + m)^{s+r})$ kernel size bound. By an adaptation of the proof we get a better bound which is even worst-case optimal, subject to a constant factor.

Theorem 9. *For any fixed ranks s and r we have the following: a set of $(s+r)\frac{(s+r)!}{s!r!}k^s m^r$ vertices containing all minimal solutions to a given instance of s, r-*Hitting Set *can be computed in polynomial time in the size of the hypergraph. Moreover, $\Theta(k^s m^r)$ vertices is the optimal upper bound.*

Proof. Given is a hypergraph of blue and red rank s and r, respectively, and integers k and m to limit the number of blue and red vertices, respectively, in a solution.

We consider sets X consisting of $s - i$ blue and $r - j$ red vertices. Suppose we have some upper bound function $h(i, j)$ for the number of hyperedges that contain X. We want to construct $h(i, j)$ in such a way that either X is a subset of at most $h(i, j)$ hyperedges, or the instance can be reduced preserving the solution space. Specifically, in the latter case we insert X as a new hyperedge and remove the, now redundant, hyperedges that contain X. Then the value $h(s, r)$ corresponding to $X = \emptyset$ will be the total number of hyperedges in the (reduced) hypergraph.

Once we have such a function $h(i, j)$, then if more than $h(s, r)$ hyperedges are present, we can simply loop through all $O(n^{r+s})$ sets consisting of at most s blue and r red vertices, test whether they are contained in at most $h(i, j)$ hyperedges, and otherwise do the replacements. That can be done in $O^*(n^{r+s})$ time, which is polynomial when the ranks are fixed.

Now let us derive the upper bound function h, by induction on i, j. We can set $h(0, 0) := 1$, since duplicate hyperedges can be safely removed. For any fixed i and j, let X be a set as specified above. Let $H(X)$ be the family of all hyperedges containing X. Consider any hitting set with no vertex from X. Since it must intersect, in particular, all hyperedges of $H(X)$, some set Z of at most k blue and m red vertices must intersect all $Y - X$, $Y \in H(X)$. We build the at most k sets $X \cup \{z\}$, $z \in Z$ blue, which have $s - (i - 1)$ blue and $r - j$ red vertices. Similarly we build the at most m sets $X \cup \{z\}$, $z \in Z$ red, which have $s - i$ blue and $r - (j - 1)$ red vertices. All hyperedges of $H(X)$ contain some of these sets. Thus $h(i, j) := k \cdot h(i - 1, j) + m \cdot h(i, j - 1)$ is safe. In the special case when some of the arguments are 0, say $j = 0$, our $H(X)$ consists only of hyperedges Y where all vertices in $Y - X$ are blue, hence all of Z is blue, and we can set $h(i, 0) := k \cdot h(i - 1, 0)$. Solving the recurrence yields $h(i, j) = \frac{(i+j)!}{i!j!} k^i m^j$, and finally $h(s, r) = \frac{(s+r)!}{s!r!} k^s m^r$.

So far we assumed that for every set X, with at most s blue and r red vertices, some hitting set with the given size limits is disjoint to X. The other case is that, for some X, every such hitting set must intersect X. But then we can create a hyperedge X, and remove all hyperedges in $H(X)$. A hyperedge is removed only due to insertion of a smaller hyperedge, hence a removed hyperedge is never inserted again. Thus, after finitely many steps we reach a hypergraph where the number of hyperedges (that fulfill the void condition to contain $X = \emptyset$) is $h(s, r)$. Clearly, they have at most $(s + r) \frac{(s+r)!}{s!r!} k^s m^r$ vertices.

To show optimality of $\Theta(k^s m^r)$, we construct a hypergraph where the union of all minimal solutions has $\Theta(k^s m^r)$ vertices. We take a rooted tree of depth $s + r$. The root is at level 0 is not a vertex of the hypergraph. All vertices in the first s levels below the root are blue and have parents with outdegree k/s, and all vertices in the last r levels are red and have parents with outdegree m/r (fractions are rounded). Every path from the root to a leaf forms a hyperedge. For any vertex v on the second last level, let P be the path from the root to v, and let S be the set of all siblings of vertices on P and all children of v. Note that S is a minimal hitting set with at most k blue and m red vertices, and every vertex belongs to such a set S. □

A difficult combinatorial problem is the optimal hidden factor in $\Theta(k^s m^r)$, for any fixed s and r. This matter is already intricate for the one-colored r-Hitting Set problem [6].

5. Open questions

It would be desirable to improve the base 1.84 for Star Editing further. Our branching algorithm is already lengthy, but a completely different FPT algorithm design technique like iterative compression may avoid even more intricate branching.

Also, our Star Editing result does *not* imply that we can find in $O^*(1.84^{k+m})$ time a solution where each of k and m separately respects some prescribed values: rule (5) does not apply to this stricter problem version, as it might be beneficial to pay for an uncovered red edge if k is "too small".

Our focus on degree 2 is, of course, a limitation, therefore it is worth looking at the general case. Union Editing (Degree s), for any fixed degree s, can be transformed into hypergraph editing problems analogously to the case $s = 2$, where *edge* is replaced with *hyperedge* of size at most s, and *vertex cover* is replaced with *hitting set*.

Then, our results raise some further questions for general s: can we solve the corresponding optimization problem significantly faster than s, 1-Hitting Set and $(s + 1)$-Hitting Set, and even faster than $O^*(s^{k+m})$? Can we enumerate all solutions (compute the transversal hypergraph) faster than in these Hitting Set instances?

Furthermore: is there a linear kernel for the optimization version of Star Editing (cf. [1])? What is the parameterized complexity of our problems when k and m are limited separately, as in the Pareto framework we proposed in [5]?

Acknowledgments

The first author has been supported by the Swedish Research Council (Vetenskapsrådet), through grant 2010-4661, "Generalized and fast search strategies for parameterized problems". The second author has been supported by a Bioscience grant from his co-supervisor Devdatt Dubhashi. We thank Sebastian Böcker for discussions around the topic, Ferdinando Cicalese for stepping in and presenting the WADS version, and the referees of both versions for their careful comments.

References

[1] R. Bar-Yehuda, D. Hermelin, D. Rawitz, An extension of the Nemhauser-Trotter theorem to generalized vertex cover with applications, SIAM J. Discrete Math. 24 (2010) 287–300.
[2] J.F. Buss, J. Goldsmith, Nondeterminism within P, SIAM J. Comput. 22 (1993) 560–572.

[3] J. Chen, I.A. Kanj, G. Xia, Improved upper bounds for vertex cover, Theoret. Comput. Sci. 411 (2010) 3736–3756.
[4] P. Damaschke, Parameterized enumeration, transversals, and imperfect phylogeny reconstruction, Theoret. Comput. Sci. 351 (2006) 337–350.
[5] P. Damaschke, Pareto complexity of two-parameter FPT problems: a case study for partial vertex cover, in: J. Chen, F. Fomin (Eds.), IWPEC 2009, in: LNCS, vol. 5917, Springer, Heidelberg, 2009, pp. 110–121.
[6] P. Damaschke, L. Molokov, The union of minimal hitting sets: parameterized combinatorial bounds and counting, J. Discrete Algorithms 7 (2009) 391–401.
[7] B. Dost, V. Bafna, N. Bandeira, X. Li, Z. Shen, S. Briggs, Shared peptides in mass spectrometry based protein quantification, in: S. Bazoglou (Ed.), RECOMB 2009, in: LNCS, vol. 5541, Springer, Heidelberg, 2009, pp. 356–371.
[8] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, New York, 1999.
[9] H. Fernau, D. Manlove, Vertex and edge covers with clustering properties: complexity and algorithms, J. Discrete Algorithms 7 (2009) 149–167.
[10] J. Guo, R. Niedermeier, S. Wernicke, Parameterized complexity of vertex cover variants, Theory Comput. Syst. 41 (2007) 501–520.
[11] Z. He, Ch. Yang, C. Yang, R. Qi, J.P.M. Tam, W. Yu, Optimization-based peptide mass fingerprinting for protein mixture identification, in: S. Bazoglou (Ed.), RECOMB 2009, in: LNCS, vol. 5541, Springer, Heidelberg, 2009, pp. 16–30.
[12] J. Kneis, A. Langer, P. Rossmanith, Improved upper bounds for partial vertex cover, in: H. Broersma, T. Erlebach, T. Friedetzky, D. Paulusma (Eds.), WG 2008, in: LNCS, vol. 5344, Springer, Heidelberg, 2008, pp. 240–251.
[13] L. Molokov, Application of combinatorial methods to protein identification in peptide mass fingerprinting, in: KDIR 2010, SciTePress, 2010, pp. 307–313.
[14] A.I. Nesvizhskii, R. Aebersold, Interpretation of shotgun proteomic data: the protein inference problem, Mol. Cellular Proteomics 4 (2005) 1419–1440.
[15] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, in: Oxford Lecture Series in Math. and its Appl., Oxford Univ. Press, 2006.
[16] V. Raman, S. Saurabh, S. Srihari, Parameterized algorithms for generalized domination, in: B. Yang, D.Z. Du, C.A. Wang (Eds.), COCOA 2008, in: LNCS, vol. 5165, Springer, Heidelberg, 2008, pp. 116–126.
[17] M. Wahlström, Algorithms, measures, and upper bounds for satisfiability and related problems, Ph.D. Thesis 1079, Linköping Studies in Science and Technol., 2007.