Multiple Hypernode Hitting Sets and Smallest Two-Cores with Targets

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

Abstract

The multiple weighted hitting set problem is to find a subset of nodes in a hypergraph that hits every hyperedge in at least m nodes. We extend the problem to a notion of hypergraphs with so-called hypernodes and show that, for $m = 2$, it remains fixed-parameter tractable (FPT), parameterized by the number of hyperedges. This is accomplished by a nontrivial extension of the dynamic programming algorithm for hypergraphs. The algorithm might be interesting for certain assignment problems, but here we need it as a tool to solve another problem motivated by network analysis: A d-core of a graph is a subgraph in which every vertex has at least d neighbors. We give an FPT algorithm that computes a smallest 2-core including a given set of target vertices, where the number of targets is the parameter. This FPT result is best possible in the sense that no FPT algorithm for 3-cores can be expected.

Keywords: hitting set, job assignment, parameterized algorithms, dynamic programming on subsets, cores in graphs

1 Introduction and Contributions

Hitting set problems are fundamental in various branches of computer science and in combinatorial optimization. Since the problems are NP-hard, the complexity of several parameterized versions is interesting. We refer to recently developed parameterized algorithms for hitting sets in hypergraphs of fixed rank [7] and for enumerations of all hitting sets [6], which is also known as monotone dualization [5].

In the Multiple Weighted Hitting Set problem we are given a family of hyperedges (a hypergraph) on a set V of nodes with positive real weights, and an integer m, and we seek a subset of nodes with minimum total weight that intersects every hyperedge in at least m distinct nodes. (We use the words "intersect" and "hit" interchangeably.) The unweighted case with $m = 1$ is the ordinary Hitting Set problem.

We further generalize the former problem as follows. Besides the hyperedges we are given another family of subsets of V that we call *hypernodes*. Every hypernode (rather than every node) has a positive weight. We seek a subset S of hypernodes with minimum total weight such that the union of all hypernodes in S intersects every hyperedge in at least m distinct nodes. We refer to this problem as Multiple Weighted Hypernode Hitting Set (MWHHS). The aforementioned Multiple Weighted Hitting Set problem is the special case of MWHHS where the the hypernodes are singleton sets, one for each node in V.

In MWHHS we may assume without loss of generality that all hyperedges are pairwise disjoint: If they are not, we "disjunctify" them as follows. For each pair v, e of a node v and hyperedge $e \ni v$, we replace v with a copy v_e of v that appears exclusively in e. The weights of hypernodes remain unchanged. While this transformation makes the hypernodes larger, it does not change the problem, and the blow-up is obviously polynomial in the size of the

hypergraph. Disjoint hyperedges are easier to handle in algorithms for the problem. Note that, if we apply the above transformation to a usual weighted hypergraph (an instance of Multiple Weighted Hitting Set), then the obtained hypernodes are also disjoint, as they come from single nodes. But we stress that hypernodes may still overlap in the general case of MWHHS. In the following, k always means the number of hyperedges which we denote $V(1), \ldots, V(k)$. For easier orientation we also give the definition more formally:

Multiple Weighted Hypernode Hitting Set (MWHHS)
Input: a set V of nodes, k hyperedges, i.e., sets $V(1), \ldots, V(k) \subset V$ which can be assumed to be pairwise disjoint, another family of subsets of V called hypernodes, each with a positive weight, and a fixed integer m (multiplicity).
Output: a set S of hypernodes with minimum total weight, such that at least m distinct nodes of every hyperedge are contained in the hypernodes in S.

We assume that the reader is familiar with the theory of fixed-parameter tractable (FPT) problems; introductions can be found, for example, in [3, 10]. Hitting Set is known to be $W[2]$-complete (thus probably not in FPT), but Hitting Set is in FPT in parameter k, the number of hyperedges. (This was observed, e.g., in [8] in dual form, i.e., for the Set Cover problem). This result can be easily extended to MWHHS with $m = 1$: The idea is to use dynamic programming on the subsets of the index set $\{1, \ldots, k\}$. We process the hypernodes successively. Every new hypernode is used in the solution or not. We only have to keep track of the minimum weights of solutions that hit every subfamily of $V(1), \ldots, V(k)$. The time bound is $O^*(2^k)$. (The O^* notation suppresses polynomial factors.) We withhold the straightforward details, because we will treat the more general case $m = 2$ in Section 2.

The difficulty increases dramatically in the case $m = 2$. Already for $m = 2$ it is not easy to see an FPT algorithm, even a bad one, let alone an FPT algorithm for combined parameters (k, m). As opposed to the Hitting Set problem, it is not of much help to classify the hypernodes in 2^k types according to which hyperedges they intersect. Since we want to hit every hyperedge in at least two *distinct* nodes, we have to memorize the nodes that are already used up, but we cannot afford to do that for every possible combination of nodes in the already selected hypernodes, as the $|V(i)|$ are not bounded by any function of parameter k. In Section 2 we develop an FPT algorithm for MWHHS with $m = 2$, by a nontrivial extension of the known dynamic programming scheme for Hitting Set.

Our original motivation for MWHHS was a network problem that we define below and treat in Section 3 and 4. However, hypernode hitting sets may be of more general interest in combinatorial optimization, especially for job or machine assignment problems where we have multiple demands or the need of redundancy. To be a little more concrete, consider the following problem as an illustration. A number of project groups works during certain time slots. Every group shall get supervision during at least m time slots. Potential supervisors are also available during certain time slots. The schedules of all project groups and supervisors are known in advance, and every supervisor demands a certain fee if she is hired for the time slots she offered. Assuming that times and fees are fixed and cannot be further negotiated, the problem is now to select a cheapest subset of supervisors to get the desired coverage. The nodes, hyperedges and hypernodes are the time slots, project groups, and supervisors, respectively. In a similar scenario, a set of installations has to be checked by experts, but this is possible only at certain times when the installations are not in use. The experts being available for the task have only certain free time slots. For security reasons we want for every installation the independent opinions of at least m experts. They have to come at different times, in order to avoid that they influence each other.

There is also a logical formulation of MWHHS: Given a monotone Boolean formula in CNF, satisfy at least m variables in every clause by a cheapest subset of variables. Clearly, the hyperedges, hypernodes and nodes are the clauses, variables, and occurrences of variables, respectively.

Next we introduce the announced network problem. To avoid confusion, we use the term *vertex* in graphs and *node* in hypergraphs. For a vertex v in a graph, $N(v)$ denotes the neighborhood of v, that is, the set of vertices adjacent to v.

For any fixed integer d, a d-core in a graph $G = (V, E)$ is a subset $C \subseteq V$ such that every vertex $v \in C$ has at least d neighbors in C. We do not strictly distinguish between C and the subgraph induced by C. Cores are interesting in network design (as robust subnetworks), but mainly in graph-theoretic approaches to clustering, as a relaxation of cliques. A recent bioinformatics application of cores is the prediction of protein complexes in protein interaction networks [2].

The maximal d-core in a graph is uniquely determined and can be computed by a trivial elimination algorithm: Starting with $C = V$, vertices in C that lack enough neighbors in C are removed from C, in any order. Since the degree of a vertex in C can only get smaller, we never have to reinsert vertices. However, the problem becomes difficult if we are interested in *minimum* d-cores that contain a given set of target vertices. (This is similar to the Steiner Tree problem that seeks a minimal connected set including a set of target vertices). We define:

Minimum d-Core
Input: a graph $G = (V, E)$ and a set $T \in V$ of t target vertices (also simply called "targets").
Output: a d-core C with $C \supseteq T$ and minimum $|C|$.

We call a d-core C *minimal* if C does not contain a smaller nonempty d-core. Likewise, $C \supseteq T$ is a *minimal d-core including T* if no d-core $C' \subset C$ includes T. Carefully distinguish between *minimal* and *minimum* which refers to set inclusion and cardinality, respectively.

Minimum 0-Core is a trivial problem. Minimal 1-cores in a graph are exactly the pairs of adjacent vertices. In an instance of Minimum 1-Core, only vertices being isolated in T need another neighbor in C. Thus we may assume without loss of generality that T is an independent set, therefore Minimum 1-Core is equivalent to the Hitting Set problem for the family of hyperedges $N(v)$, $v \in T$. Since Hitting Set is in FPT, with the number of hyperedges as the parameter, it follows that Minimum 1-Core is in FPT with parameter t. In Section 4 we will show that Minimum 2-Core is still in FPT with parameter t, using the preceding FPT result for MWHHS with $m = 2$. Prior to this final contribution, Section 3 provides some results on the hardness of Minimum d-Core. In particular, our FPT result for $d = 2$ is best possible in the sense that Minimum 3-Core is not in FPT, under common complexity-theoretic assumptions.

Here it is worth mentioning another related hardness result: Finding a chordless cycle through two target vertices is W[1]-complete [9]. Note that a chordless cycle is a 2-core, but Minimum 2-Core permits arbitrary 2-cores rather than only chordless cycles. (Cores are in general not even connected.) The Steiner Tree problem is known to be in FPT, with the number t of target vertices as the parameter. It can be solved in $O^*(3^t)$ time [4]. As recently shown [1], finding minimum d-cores without specified target vertices is $W[1]$-hard for any $d \geq 3$, with the size of the solution as parameter.

Our study is exploratory research in problem complexity, not derived from an immediate application. The use of cores and target (or "seed") vertices in [2] is very different from the Minimum d-Core problem, however one may use cores in similar inference tasks. Suppose we have a model of a protein interaction network with only a sparse set of confirmed interactions, and for some set T of proteins we know that they belong to the same functional group, moreover, in a functional group we expect each protein to interact with at least d others. Then it is sensible to ask what is a smallest possible vertex set including T, with this property.

2 Weighted Hypernode Hitting Sets with Two Hits

In this section we prove that MWHHS for $m = 2$ is in FPT with parameter k. First we outline our dynamic programming algorithm.

Hypernodes are listed in any order, and in each step of the algorithm we consider the next hypernode from list. We maintain a family F of *partial solutions*, where a partial solution is the set of hypernodes we have selected so far, along with their total weight. According to the specification of MWHHS, only the union of the selected hypernodes is relevant, hence we may sometimes consider a partial solution just as a set of nodes, without risk of confusion. A family F of partial solutions is called *promising* if at least one member of F can be extended to an optimal final solution, by adding a suitable subset of hypernodes from the remainder of the list. The algorithm starts with F containing only one partial solution, namely the empty set with weight 0. Trivially, this initial F is promising. As soon as all hypernodes have been considered, a promising F clearly contains some optimal solution, hence we can simply pick the best solution from F. Now we work out the details.

Remember that $V(1), \ldots, V(k)$ are the (disjoint!) hyperedges. We use the following scheme of denotations, for arbitrary letters. For any symbol X (an upper-case letter, perhaps with further symbols attached) that denotes a subset of nodes, we denote by $X(i)$ the set $X \cap V(i)$. If a set $X(i)$ contains only one node, we denote it by $x(i)$ (using the corresponding lower-case letter). Conversely, by mentioning a node $x(i)$ we implicitly mean that this is the only node of $X(i)$.

Consider any completed step, i.e., some prefix of the list of hypernodes has been read. The *signature* of a partial solution P is the vector s of numbers $s(i) = \min\{|P(i)|, 2\}$, for $i = 1, \ldots, k$. Any final solution extending P can be written as $P \cup Q$, where Q contains the nodes added to the partial solution P by later hypernodes. In the following discussion we will always split solutions in this way. Given a signature s, a *1-index* is an index i with $s(i) = 1$, similarly we speak of *0-indices* and *2-indices*. Let *seq* be any sequence of pairwise distinct indices, in particular, *seq* can be the empty sequence. If index j appears somewhere in *seq*, we denote by seq_j the prefix of *seq* before j.

Let T be the hypernode next in the list. We update F by adding all sets $P \cup \{T\}$, $P \in F$, to the old family F (and add the weight of T). Since F was promising before this extension, the new F is promising, too. The tricky part is to remove some partial solutions from F, in such a way that F remains promising, but the number of sets in F is kept below some threshold (which may only depend on parameter k) all the time.

Lemma 1 *Any promising family F of partial solutions has a promising subfamily with at most $e^2 2^k k!$ different partial solutions. (Here e denotes Euler's number.) Such a subfamily can be computed from F in a time polynomial in the size of F.*

Proof. First we only consider the partial solutions with any fixed signature s, therefore we conveniently suppress s in our notation. Let $G[]$ denote the family of members of F with signature s. Suppose $G[] \neq \emptyset$, otherwise nothing needs to be removed from $G[]$. Let $P[]$ be some partial solution in $G[]$ with minimum weight. Consider any optimal solution $P \cup Q$, $P \in G[]$, with the property that

$$Q(i) \setminus P[](i) \neq \emptyset \text{ for all 1-indices } i.$$

Then $P[] \cup Q$ is a valid solution, as we can easily see: For 0-indices i we have $|Q(i)| \geq 2$, and for 2-indices i we have $|P[](i)| \geq 2$. For 1-indices i, the displayed property guarantees $|P[](i) \cup Q(i)| \geq 2$. Furthermore, since $P[]$ has at most the weight of P, solution $P[] \cup Q$ is optimal as well. Hence, partial solutions $P \in G[]$ other than $P[]$ are only needed to form possible optimal solutions $P \cup Q$ where $Q(i) \subseteq P[](i)$ holds for some 1-index i. Note that the latter condition implies $q(i) = p[](i)$ and $p(i) \neq p[](i)$ for such a 1-index i and such a P.

This gives rise to the following inductive hypothesis, for any sequence *seq* of pairwise distinct 1-indices: Partial solutions $P \in G[seq]$ other than some minimum-weight partial solution $P[seq] \in G[seq]$ are only needed to form possible optimal solutions $P \cup Q$ where

$q(j) = p[seq_j](j)$ holds for all j in seq, and $p(i) \neq p[seq](i) = q(i)$ holds for some 1-index i not occuring in seq. (As we saw above, the hypothesis is true if seq is the empty sequence. Note that some conditions are vacuously true in this base case.)

Accordingly, for every 1-index i not occuring in seq we define $G[seq, i]$ to be the family of all partial solutions $P \in G[seq]$ with $p(i) \neq p[seq](i)$. Let $P[seq, i]$ be some minimum-weight partial solution in $G[seq, i]$. Now we prove the induction hypothesis for the sequence seq, i, similarly as in the base case.

Consider any optimal solution $P \cup Q$, $P \in G[seq, i]$, with the following properties:

- $q(j) = p[seq_j](j)$ for all j in seq,
- $q(i) = p[seq](i)$,
- $Q(j) \setminus P[seq, i](j) \neq \emptyset$ for all 1-indices j not occuring in seq, i.

Then $P[seq, i] \cup Q$ is a valid solution: For 0-indices j we have $|Q(j)| \geq 2$, and for 2-indices j we have $|P[seq, i](j)| \geq 2$. For 1-indices j not occuring in seq, i, the displayed properties obviously guarantee $|P[seq, i](j) \cup Q(j)| \geq 2$. For the particular index i, observe that $p[seq, i](i) \neq p[seq](i) = q(i)$, hence we have $|P[seq, i](i) \cup Q(i)| \geq 2$ as well. For 1-indices j in seq we also have $p[seq, i](j) \neq q(j)$, this because $q(j) = p[seq_j](j)$ holds, and $P[seq, i] \in G[seq, i] \subset G[seq_j, j]$ implies $p[seq, i](j) \neq p[seq_j](j)$ by the definition of $G[seq_j, j]$.

Furthermore, since $P[seq, i]$ has at most the weight of P, solution $P[seq, i] \cup Q$ is optimal among all solutions with the displayed properties. Consequently, partial solutions $P \in G[seq, i]$ other than $P[seq, i]$ are only needed to form possible optimal solutions $P \cup Q$ where $q(j) = p[seq_j](j)$ for all j in seq, and $q(i) = p[seq](i)$, but $Q(j) \subseteq P[seq, i](j)$ for some 1-index j not occuring in seq, i. The latter condition also implies $|Q(j)| = 1$, $q(j) = p[seq, i](j)$, and $p(j) \neq p[seq, i](j)$. This concludes the induction step.

Our inductive definition yields families $G[seq]$ and partial solutions $P[seq] \in G[seq]$ (unless $G[seq] = \emptyset$, in which case seq is not extended further). As soon as a sequence seq contains all 1-indices i, it suffices to keep only some optimal $P[seq]$ in $G[seq]$. Other $P \in G[seq]$ cannot be parts of optimal solutions $P \cap Q$ anymore, since no 1-indices j are left where $Q(j)$ could undesirably be contained in $P[seq](j)$.

Until now we considered any fixed signature s. By construction, the family of all $P[seq]$ for all sequences *seq and all signatures* s is a promising subfamily of F. Clearly, each $G[seq, i]$ and $P[seq, i]$ can be obtained from $G[seq]$ and $P[seq]$ in polynomial time (with respect to the size of F). It remains to bound the size of this promising subfamily. There exist $2^{k-t}\binom{k}{t}$ signatures with exactly t 1-indices, and for every such signature we can form $\sum_{i=0}^{t} \binom{t}{i}(t-i)! = \sum_{i=0}^{t} t!/i! < et!$ sequences of distinct 1-indices (where $t - i$ is the length). Finally observe:
$\sum_{t=0}^{k} 2^{k-t}\binom{k}{t} et! < e2^k \sum_{t=0}^{k} \binom{k}{k-t} t! < e^2 2^k k!$. □

From Lemma 1 we obtain:

Theorem 2 *MWHHS with multiplicity $m = 2$ can be solved in $O^*(2^k k!)$ time.*

Proof. To summarize the algorithm: We successively add the given hypernodes to the problem instance, in an arbitrary order, and maintain a promising family F of partial solutions, i.e., unions of selected hypernodes. Initially, F is the family with the empty set as the only member. For every new hypernode T we insert all sets $P \cup \{T\}$, $P \in F$, in F. Then we use the procedure in Lemma 1 to extract a promising subfamily of F with $O(2^k k!)$ sets. When all hypernodes are included, an optimal solution in F is also an optimal solution to the problem. All auxiliary computations are obviously polynomial. □

3 Hardness of Minimum Cores Including a Target Set

We turn to minimum d-cores with target vertices. Remember that MINIMUM 1-CORE is basically the HITTING SET problem and thus NP-complete. The following result is quite easy to obtain, however it does not directly follow from the case $d = 1$.

Theorem 3 MINIMUM d-CORE *is NP-complete for any constant* d.

Proof. Consider any $d \geq 2$. Given an instance of HITTING SET where, without loss of generality, the hyperedges cover the whole set of nodes, we construct a graph as follows. Every node of the hypergraph is represented by a vertex of the graph. Furthermore, every hyperedge e is represented by a target vertex, adjacent to those vertices representing the nodes of e. Finally, we attach to every vertex v, of both types, a $(d+1)$-clique of new target vertices, and insert an edge between v and $d-1$ vertices of this $(d+1)$-clique.

By construction, target vertices in the attached cliques have enough neighbors in any d-core which includes the target set. Since every target vertex which is not in an attached clique needs yet another neighbor in a d-core, we obtain a one-to-one correspondence between d-cores and hitting sets in the hypergraph. Note that every non-target vertex added to a d-core is in fact adjacent to $d-1$ vertices in its attached clique, and to another target vertex representing a hyperedge. □

Due to NP-completeness, it is natural to study the parameterized complexity of MINIMUM d-CORE. As mentioned earlier, case $d = 1$ is fixed-parameter tractable in the number t of target vertices, and for $d = 2$ we will show this in Section 4. For $d \geq 3$ we cannot get an analogous FPT result because of

Theorem 4 *The complexity of* MINIMUM 3-CORE *cannot be bounded by any function of* t *and a polynomial factor, unless* $P = NP$.

Proof. We reduce HITTING SET in polynomial time to MINIMUM 3-CORE with $t = 1$. Since HITTING SET is NP-complete, already this special case cannot be solved in polynomial time, unless $P = NP$.

For any given hypergraph $H = (V, F)$, i.e., an instance of HITTING SET, we construct the following graph with one target vertex. We make the target the root of a tree T. The root has three children, and every inner vertex of T gets exactly two children. We generate as many leaves as we have hyperedges in F, plus three special leaves. Each of the former leaves represents a hyperedge. We call them h-leaves. We add further vertices, each of which represents a node from V. We call them n-vertices. The h-leaf of every hyperedge $f \in F$ is connected by edges to exactly the n-vertices of the nodes in f. Finally, we connect all h-leaves also to one special leaf, and we connect every n-vertex to the two other special leaves.

Since inner vertices of T are not adjacent to vertices outside T, any 3-core C that includes the target vertex must obviously include the whole tree T, in particular all its leaves. Furthermore, every leaf needs two further neighbors in C. For the h-leaves, one of these required neighbors in C is the special leaf, and another neighbor in C must be an n-vertex. On the other hand, every n-vertex selected for C has enough neighbors in C: two special leaves, and at least one h-leaf. The special leaves get enough neighbors in C by construction. This way, minimum hitting sets in H correspond to minimum 3-cores in the graph. □

4 Minimum Two-Cores Including a Target Set

In this section we develop an FPT algorithm for MINIMUM 2-CORE, with the number t of target vertices as the parameter. We use some standard graph-theoretic notation: A subgraph *spanned* by an edge set consists of this edge set and all involved vertices (ignoring any additional edges between them!). The length of a path is the number of edges in this path.

Consider a graph $G = (V, E)$ with target set $T \subset V$. Clearly, we may assume that all vertices in G have degree at least 2. We reduce our graph as follows. Every vertex v gets *valency* $\max\{2 - |N(v) \cap T|, 0\}$, that is, the valency of v is the number of non-target neighbors that v demands in a 2-core C if we put v in C. Then, target vertices of valency 0 are removed. Edges between target vertices are removed as well. (Note that these removals do not affect the valencies of the remaining vertices.) For convenience we still use G and T to denote the reduced graph (with vertices labeled by their valencies) and the remaining target set, respectively.

For the moment we fix, simultaneously for every target vertex $v \in T$ with valency i, at least i neighbors of v that shall belong to C, and call these neighbors *ports*. Clearly, each port has valency 0 (if it is adjacent to several targets) or 1. Later we will discuss how we actually choose these ports, and this is the point where we will need the MWHHS algorithm for $m = 2$.

A path in G is said to be *regular* if all its inner vertices are non-targets and have valency 2. In particular, trivial paths of length 0 or 1 are regular. An u, v-path is a path from vertex u to vertex v. A subset F of edges is called *saturating* if, for every vertex v in the subgraph spanned by F, the number of edges from F incident to v is at least the valency of v. Since ports in C are fixed, we get straight from the definitions:

Lemma 5 *$C \supset T$ is a 2-core if and only if some saturating edge set spans exactly the vertices in $C \setminus T$ of valency 1 and 2.* □

Hence, in the following we can work with saturating edge sets rather than 2-cores, and take advantage of their special structure:

Lemma 6 *Let F be a saturating edge set that spans at least the set of ports of valency 1 and is minimal with this property. Then the subgraph spanned by F consists of connected components of the following types:*
Star*: a vertex called the* center *is connected to distinct ports by internally vertex-disjoint regular paths (in particular, a star might consist of a single path to only one port),*
Loop*: a regular path (possibly of length 0) with a port v at one end, and a cycle of vertices of valency 2 attached to the other end. We call this structure a* loop for vertex v.

Proof. In the following, all vertex degrees are understood with respect to F, and removing a vertex also means to remove all incident edges from F. Consider any connected component.

Let u_0 be a vertex of degree larger than 2. First we claim for every path $u_0, \ldots, u_k$, with edges in F, that all u_i, $0 < i < k$, have valency 2, and u_k has either valency 2 as well, or u_k is a port, or $u_k = u_0$ (the path is a cycle). We proceed by induction on k. For $k = 0$, the claim is vacuously true with $u_k = u_0$. Suppose that the claim holds for $u_0, \ldots, u_{k-1}$. Note that the degree of u_0 is larger than its valency. If also the the degree of u_k is larger than its valency, but $u_k \neq u_0$, then we can remove $u_1, \ldots, u_{k-1}$ (or remove edge u_0u_1 from F in the case $k = 1$), contradicting the minimality of F. Hence u_k cannot have valency 0. If u_k has valency and degree 1, then u_k must be a port, otherwise we can remove $u_1, \ldots, u_k$ from the component, which contradicts again the minimality of F. This proves the claim.

Due to the claim, if every path starting in u_0 ends in a port, the component is a star. The other case is that a cycle containing u_0 is in the component. Then, at most one further path (to some port) can start in u_0, otherwise the start edges of two of these paths would already provide two neighbors for u_0, and the cycle could be removed from F. Similarly, the component cannot contain more than one cycle: vertices of all further cycles (except u_0

itself) can be removed, contradicting the minimality of F. Hence the component is a loop in this case.

If no vertex u_0 with degree larger than 2 exists, the component is just a path or cycle. No two distinct vertices u, v of degree 2 but with smaller valency can exist, since otherwise we could remove the edge or subpath between u and v, leaving both u and v with at least one neighbor.

Thus, a cycle component has exactly one port and no other vertices of valency smaller than 2. That means, such a component is a loop, consisting of a cycle and a trivial path of length 0.

A path component cannot end with a subpath of vertices with valency 2, since we could remove a maximal such subpath. We conclude that a path component has two end-vertices of valency smaller than 2. Moreover, as we saw above, at most one inner vertex has valency smaller than 2. If such an inner vertex exists, say w, then both end-vertices must be ports, otherwise we could again remove a subpath on one side of w. Thus, we get a star consisting of two paths. If all inner vertices have valency 2, still at least one end-vertex is a port, and we get a star with one or two paths and an appropriately chosen center. (Note that, by the definition of star, in a regular path connecting two ports we can declare an arbitrary vertex the center.)

Now we have treated all cases. □

Next step is to observe that, for every port, we can efficiently compute several items specified in Lemma 6:

Lemma 7 *For every port u we can compute in polynomial time a shortest regular path from u to every vertex v, and a minimum-size loop for u (if one exists).*

Proof. Computing shortest paths (here restricted to paths with inner vertices of valency 2) is a standard task, thus we only have to prove the statement for loops. For every v, where either $v = u$ or v has valency 2, we compute a shortest regular u, v-path, and also a shortest cycle through v where all vertices (except v itself if $v = u$) have valency 2. Let us call it a *regular cycle* through v. For computing a shortest regular cycle through v we may check every neighbor w of v with valency 2 and determine a shortest regular w, v-path avoiding the edge vw.

We claim that a minimum-size loop for u consists of some shortest regular u, v-path and a shortest regular cycle through v, for some vertex v. To prove the claim, consider a minimum-size loop for u, and let v be the vertex where path and cycle of this loop intersect. If there exist several minimum-size loops for u, we choose one where the u, v-path is as short as possible. Assume that some shortest regular u, v-path and some shortest regular cycle through v intersect in further vertices other than v. Then, obviously, there exists either a smaller loop for u, or a loop with the same vertices and edges, where path and cycle intersect in exactly one vertex $v' \neq v$ at a smaller distance to u. Both cases contradict our choice of the loop. This establishes the claim for the specified vertex v.

Hence, in order to compute a minimum-size loop for u we only need to do the following: For all vertices v, combine some shortest regular u, v-path and some shortest regular cycle through v, and finally pick a combination with the minimum number of vertices. □

Now we construct in polynomial time an instance of MWHHS from the given graph and target set, based on Lemma 7.

Disjoint hyperedges from target vertices: A hyperedge is assigned to each target vertex u, and the nodes in the hyperedge of u are the vertices in $N(u)$. However, we "disjunctify" the neighborhoods of target vertices as follows. For any vertex v of G that is adjacent to several targets u, we put one copy of v in the hyperedge of every such u.

Moreover, for every target u of valency 1, we represent each vertex $v \in N(u)$ by *two* nodes (two copies of v) in the hyperedge of u.

Hypernodes from single vertices: For every vertex v of G being adjacent to more than one target, all nodes built from v (see above) form a hypernode of weight 1.

Hypernodes from stars: We consider every vertex c of G as the center of several possible stars. By the *regular distance* between two vertices of G we mean the length of a shortest regular path connecting them.

For every fixed vertex c, we do the following. In the neighborhood of every target vertex of valency 2, we determine two vertices with the smallest regular distances to c (ties are broken arbitrarily). Similarly, in the neighborhood of every target of valency 1, we determine one vertex with the smallest regular distance to c. These distinguished vertices will become ports in several stars with center c. We select some shortest regular path from c to each of the (at most $2t$) ports. Then we consider the (less than 4^t) unions of subsets of these selected regular paths starting in c. Among these unions we keep only the stars, i.e., unions of regular paths that are pairwise internally vertex-disjoint. Finally, for any such star we create a hypernode, consisting of the nodes corresponding to the ports. (In particular, for any port adjacent to a target u of valency 1, we include both corresponding nodes in the hypernode, so that it hits the hyperedge of u twice.) The weight of a hypernode is the number of vertices in the corresponding star in G.

Hypernodes from loops: For every vertex v of G such that a loop for v exists, we create a hypernode corresponding to some minimum-size loop for v. It consists of all nodes corresponding to v (see above), and its weight is the number of vertices in the minimum-size loop.

This finishes the reduction. We fix $m = 2$. The relationship between MWHHS and Minimum 2-Core is established by:

Lemma 8 *From any optimal solution to the MWHHS instance as constructed above, we can compute, in polynomial time, a minimum 2-core including T in graph G.*

Proof. By construction, every hypernode corresponds to: some star in G (with ports adjacent to targets), or some loop for one port (adjacent to some target in G), or a single vertex adjacent to more than one target. For convenience we refer to these single vertices as ports of valency 0. Recall that a port has valency 1 (0) if the port is neighbor of exactly one (at least two) target(s).

Let S be an optimal solution to our instance of MWHHS with $m = 2$. Then, the vertices in the corresponding stars, loops, and ports of valency 0 extend the target set T to a 2-core. This follows from the shape of stars and loops, and from the fact that S hits every hyperedge at least twice. By definition of hypernode weights, the weight of S is at least the total number of vertices in the stars and loops, plus the number of ports of valency 0 (in other words, the number of non-target vertices in the 2-core). We claim that the 2-core defined by S has already minimum cardinality.

Consider a 2-core $C \supseteq T$ with minimum cardinality, and assume for contradiction that C has fewer vertices than the 2-core we obtained from S. In the following we refer to the vertices of C being adjacent to any target vertices as ports.

Since C is a 2-core, every target is adjacent to at least the required number of ports and, by Lemma 5, some saturating edge set F spans the vertices of $C \setminus T$ of valency greater than 0. In particular, F spans at least the ports of valency 1. We may assume that F is minimal with this property, since otherwise a proper subset of F is saturating and spans the same ports of valency 1, and together with the ports of valency 0 this gives a 2-core no larger than C. Due to minimality, F has the structure reported in Lemma 6, that is, ports of valency 1 are connected by stars and loops which are pairwise disjoint.

Consider any loop component of F, say, a loop for port v. We may assume that this loop has minimum size among all possible loops for v, since otherwise we may replace it with a smaller loop for v and get a smaller 2-core, a contradiction. Hence, there exists a hypernode

for any loop component of F. Similarly, consider any star component of F, say with center c. Let v be any port in this star, and let u be the target adjacent to v. Note that v has valency 1 and is adjacent to no other target. If u had valency 2, and v is not one of the two closest ports v_1, v_2 in $N(u)$ selected in the reduction (i.e., those with smallest regular distances to c), we replace the path from c to v in the star with a shortest regular path from c to v_1 or v_2, provided that one of them is not yet in another star or loop. If both v_1 and v_2 are already occupied, u has already these two neighbors in C, and we can just remove the path from c to v. Thus, we either obtain a smaller 2-core (contradiction), or we need only those ports v_1, v_2 selected in the reduction. The reasoning is similar for a target u of valency 1: We only need to consider some port with minimum regular distance to c, and the resulting node was doubled in the reduction. In summary, after these replacements there exists a hypernode also for any star component in F.

Since, by assumption, C had already minimum size, F is still a minimal saturating edge set that spans the (possibly changed) ports of valency 1, thus it complies with the structure described in Lemma 6. But now the connected components of F and the ports of valency 0 in C correspond to hypernodes and form another solution S' to our MWHHS instance. Since the components are pairwise vertex-disjoint, the weight of S' equals the number of vertices spanned by F plus the number of ports of valency 0. This contradicts the minimality of the weight of S. □

We are ready to state our final result:

Theorem 9 Minimum 2-Core *with t targets can be solved in $O^*(2^t t!)$ time.*

Proof. This follows from Theorem 2 and Lemma 8, since in the reduction to MWHHS we needed only $O^*(4^t)$ time to construct the weighted hypernodes. □

5 Conclusions and Open Problems

We defined the Multiple Weighted Hypernode Hitting Set (MWHHS) problem where a minimum weight subset of "hypernodes" has to hit every hyperedge in at least m nodes, for some fixed m. Its complexity may find interest in logical inference and operations research (job assignment and scheduling). Our first result was that MWHHS for $m \leq 2$ and with the number k of hyperedges is in FPT, using some intricate dynamic programming. Our time bound is $O^*(2^k k!)$. It would be nice to simplify the algorithm and proof. It might also be possible to improve the $k!$ factor by using similarities between partial solutions with different index sequences (see the proof of Lemma 1). An more ambitious question is whether an exponential bound with a constant base can be accomplished. We also conjecture that the scheme can be extended to any constant m (using even more complex indices), which may lead to an FPT algorithm with another parameter m. However, this is not easy to see.

Then we used MWHHS with $m = 2$ as an auxiliary problem to solve another problem in the analysis of networks: finding a minimum 2-core that includes a prescribed set of t target vertices. It is related to known problems like Steiner tree and finding cycles through given vertices. We developed an FPT algorithm with time bound $O^*(2^t t!)$, using star-shaped subgraphs of potential solutions as hypernodes in MWHHS. The same open question as for MWHHS, regarding improved bounds and simplifications can be formulated also for Minimum 2-Core. Unfortunately, the analogous problem for 3-cores etc. is even unlikely to be in FPT, but there may be other natural parameterizations that make the problem tractable. We also propose a systematic effort to define degree-based notions of subgraphs with density requirements, and figure out the complexity of finding such subgraphs. This would enrich the arsenal of clustering paradigms in graphs. Various types of dense subgraphs are needed, e.g., to predict meaningful subgraphs in biological interaction networks, because apparently no type of cluster fits them all.

Acknowledgements

This work has been supported by the Swedish Research Council (Vetenskapsrådet), grant no. 2007-6437, "Combinatorial inference algorithms – parameterization and clustering". The author also thanks the anonymous referees for their helpful comments.

References

[1] Amini O, Sau I, Saurabh S (2008) Parameterized complexity of the smallest degree-constrained subgraph problem. In: 3rd Int Workshop on Parameterized and Exact Computation IWPEC 2008, LNCS 5018, pp 13-29

[2] Bader GD, Hogue CWV (2003) An automated method for finding molecular complexes in large protein interaction networks. BMC Bioinformatics 4:2

[3] Downey RG, Fellows MR (1999) Parameterized Complexity. Springer

[4] Dreyfus S, Wagner R (1971) The Steiner problem in graphs. Networks 1:195-207

[5] Eiter T, Makino K, Gottlob G (2008) Computational aspects of monotone dualization: A brief survey. Discrete Appl Math 156:2035-2049

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

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

[8] Fomin FV, Kratsch D, Woeginger GJ (2004) Exact (exponential) algorithms for the dominating set problem. In: 30th Int Workshop on Graph-Theoretic Concepts in Computer Science WG 2004, LNCS 3353, pp 245-256

[9] Haas R, Hoffmann M (2006) Chordless paths through three vertices. Theor Computer Science 351:360-371

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