Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries*

Dániel Marx†
School of Computer Science
Tel Aviv University
Tel Aviv, Israel
dmarx@cs.bme.hu

ABSTRACT

An important question in the study of constraint satisfaction problems (CSP) is understanding how the graph or hypergraph describing the incidence structure of the constraints influences the complexity of the problem. For binary CSP instances (i.e., where each constraint involves only two variables), the situation is well understood: the complexity of the problem essentially depends on the treewidth of the graph of the constraints [19, 24]. However, this is not the correct answer if constraints with unbounded number of variables are allowed, and in particular, for CSP instances arising from query evaluation problems in database theory. Formally, if $\mathcal{H}$ is a class of hypergraphs, then let CSP($\mathcal{H}$) be CSP restricted to instances whose hypergraph is in $\mathcal{H}$. Our goal is to characterize those classes of hypergraphs for which CSP($\mathcal{H}$) is polynomial-time solvable or fixed-parameter tractable, parameterized by the number of variables. In the applications related to database query evaluation, we usually assume that the number of variables is much smaller than the size of the instance, thus parameterization by the number of variables is a meaningful question.

The most general known property of $\mathcal{H}$ that makes CSP($\mathcal{H}$) polynomial-time solvable is bounded fractional hypertree width. Here we introduce a new hypergraph measure called *submodular width,* and show that bounded submodular width of $\mathcal{H}$ (which is a strictly more general property than bounded fractional hypertree width) implies that CSP($\mathcal{H}$) is fixed-parameter tractable. In a matching hardness result, we show that if $\mathcal{H}$ has unbounded submodular width, then CSP($\mathcal{H}$) is not fixed-parameter tractable (and hence not polynomial-time solvable), unless the Exponential Time Hypothesis (ETH) fails. The algorithmic result uses tree decompositions in a novel way: instead of using a single decomposition depending on the hypergraph, the instance is split into a set of instances (all on the same set of variables as the original instance), and then the new instances are solved by choosing a different tree decomposition for each of them. The reason why this strategy works is that the splitting can be done in such a way that the new instances are "uniform" with respect to the number extensions of partial solutions, and therefore the number of partial solutions can be described by a submodular function. For the hardness result, we prove via a series of combinatorial results that if a hypergraph H has large submodular width, then a 3SAT instance can be efficiently simulated by a CSP instance whose hypergraph is H. To prove these combinatorial results, we need to develop a theory of (multicommodity) flows on hypergraphs and vertex separators in the case when the function $b(S)$ defining the cost of separator S is submodular.

Categories and Subject Descriptors

F.2 [**Theory of Computing**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Design, Performance, Theory

Keywords

constraint satisfaction, conjunctive queries, submodular width, fixed-parameter tractability

1. INTRODUCTION

There is a long line of research devoted to identifying hypergraph properties that make the evaluation of conjunctive queries tractable (see e.g. [15, 31, 18, 19]). In the early 80s, it has been noted that acyclicity is one such property [5, 11, 32, 4]. Later, more general such properties were identified in the literature: for example, bounded query width [7], bounded hypertree width [15], and bounded fractional hypertree width [25, 20]. Our main contribution is giving a complete theoretical answer to this question: in a very precise technical sense, we characterize those hypergraph properties that imply tractability for the evaluation of a query. Efficient evaluation of queries is originally a question of database theory; however, the problem can be treated as a constraint satisfaction problem [23, 17, 31].

Constraint satisfaction. Constraint Satisfaction Problems (CSP) form a general framework that includes many standard algorithmic problems such as satisfiability, graph

†Research supported by ERC Advanced Grant DMMCA.
*The full version of the paper is available at `http://arxiv.org/abs/0911.0801`

STOC'10, June 5–8, 2010, Cambridge, Massachusetts, USA.

coloring, database queries, etc. [18, 12]. A CSP instance consists of a set V of variables, a domain D, and a set C of constraints, where each constraint is a relation on a subset of the variables. The task is to assign a value from D to each variable in such a way that every constraint is satisfied. It is easy to see that Boolean Conjunctive Query can be formulated as the problem of deciding if a CSP instance has a solution: the variables of the CSP instance correspond to the variables appearing in the query and the constraints correspond to the database relations. A distinctive feature of CSP instances obtained this way is that the number of variables is small (as queries are typically small), while the domain of the variables are large (as the database relations usually contain a large number of entries). This has to be contrasted with typical CSP problems from AI, such as 3-colorability and satisfiability, where the domain is small, but the number of variables is large. As our motivation is database-theoretic, in the rest of the paper the reader should keep in mind that we are envisioning scenarios where the number of variables is small and the domain is large.

A natural question is to investigate the effect of *structural restrictions* on the complexity of CSP; that is, what kind of restrictions on the structure induced by the constraints on the variables make the problem tractable. The *hypergraph* of a CSP instance is defined to be a hypergraph on the variables of the instance such that for each constraint $c \in C$ there is a hyperedge e_c containing exactly the variables appearing c. If the hypergraph of the CSP instance has simple structure, then the instance is easy to solve. It is well-known that a CSP instance I with hypergraph H can be solved in time $\|I\|^{O(\mathrm{tw}(H))}$ [14], where $\mathrm{tw}(H)$ denotes the treewidth of H and $\|I\|$ is the size of the representation of I.

Our goal is to characterize the "easy" and "hard" hypergraphs from the viewpoint of constraint satisfaction. However, formally speaking, CSP is polynomial-time solvable for every fixed hypergraph H: since H has a constant number k of vertices, every CSP instance with hypergraph H can be solved by trying all $\|I\|^k$ possible combinations on the k variables. It makes more sense to characterize those *classes* of hypergraphs where CSP is easy. For a class $\mathcal{H}$ of hypergraphs, let $\mathrm{CSP}(\mathcal{H})$ be the restriction of CSP to instances whose hypergraphs are in $\mathcal{H}$. For the characterization of the complexity of $\mathrm{CSP}(\mathcal{H})$, we can investigate two notions of tractability. $\mathrm{CSP}(\mathcal{H})$ is *polynomial-time solvable* if there is an algorithm solving every instance of $\mathrm{CSP}(\mathcal{H})$ in time $(\|I\|)^{O(1)}$, where $\|I\|$ is the length of the representation of I in the input. For example, if $\mathcal{H}$ is class of hypergraphs with bounded treewidth, then $\mathrm{CSP}(\mathcal{H})$ is polynomial-time solvable. The following notion interprets tractability in a less restrictive way: $\mathrm{CSP}(\mathcal{H})$ is *fixed-parameter tractable (FPT)* if there is an algorithm solving every instance I of $\mathrm{CSP}(\mathcal{H})$ in time $f(H)(\|I\|)^{O(1)}$, where f is an arbitrary function.

The case of bounded arities. If the constraints have bounded arity (i.e., the edge size in $\mathcal{H}$ is bounded by a constant r), then $\mathrm{CSP}(\mathcal{H})$ is well understood: bounded treewidth is the only polynomial-time solvable case.

THEOREM 1.1 ([19]). *If $\mathcal{H}$ is a recursively enumerable class of hypergraphs with bounded edge size, then (assuming* FPT $\neq$ W[1]*) the following are equivalent:*

1. *CSP($\mathcal{H}$) is polynomial-time solvable.*
2. *CSP($\mathcal{H}$) is fixed-parameter tractable.*
3. *$\mathcal{H}$ has bounded treewidth.*

Theorem 1.1 proves the surprising result that whenever $\mathrm{CSP}(\mathcal{H})$ is fixed-parameter tractable, it is polynomial-time solvable as well. The following sharpening of Theorem 1.1 shows that there is no algorithm whose running time is significantly better than the $\|I\|^{O(\mathrm{tw}(H))}$ bound of the treewidth based algorithm. The result is proved under the Exponential Time Hypothesis (ETH) [22], a somewhat stronger assumption than FPT $\neq$ W[1]: it is assumed that there is no $2^{o(n)}$ time algorithm for n-variable 3SAT.

THEOREM 1.2 ([24]). *If there is a function f and a recursively enumerable class $\mathcal{H}$ of hypergraphs with bounded edge size and unbounded treewidth such that CSP($\mathcal{H}$) can be solved in time $f(H)\|I\|^{o(\mathrm{tw}(H)/\log \mathrm{tw}(H))}$ for instances I with hypergraph $H \in \mathcal{H}$, then ETH fails.*

Unbounded arities. The situation is less understood in the unbounded arity case, i.e., when there is no bound on the maximum edge size in $\mathcal{H}$. First, the complexity in the unbounded-arity case depends on how the constraints are represented. In the bounded-arity case, if each constraint contains at most r variables (r being a fixed constant), then every reasonable representation of a constraint has size $|D|^{O(r)}$. Therefore, the size of the different representations can differ only by a polynomial factor. On the other hand, if there is no bound on the arity, then there can be exponential difference between the size of succinct representations (e.g., formulas [8]) and verbose representations (e.g., truth tables [26]). The running time of an algorithm is expressed as a function of the input size, hence the complexity of the problem can depend on how the input is represented: longer representation means that it is potentially easier to obtain a polynomial-time algorithm.

The most well-studied representation of constraints is listing all the tuples that satisfy the constraint. This representation is perfectly compatible with our database-theoretic motivation: the constraints are relations of the database, and a relation is physically stored as a table containing all the tuples in the relation. For this representation, there are classes $\mathcal{H}$ with unbounded treewidth such that CSP restricted to this class is polynomial-time solvable. A trivial example is the class $\mathcal{H}$ of all hypergraphs having only a single hyperedge of arbitrary size. There are other classes of hypergraphs with unbounded treewidth such that $\mathrm{CSP}(\mathcal{H})$ is solvable in polynomial time: for example, classes with bounded *(generalized) hypertree width* [16], bounded *fractional edge cover number* [20], and bounded *fractional hypertree width* [20, 25]. Thus treewidth is not the right measure for characterizing the complexity of the problem.

Our results. We introduce a new hypergraph width measure that we call *submodular width.* Small submodular width means that for every monotone submodular function b on the vertices of the hypergraph H, there is a tree decomposition where $b(B)$ is small for every bag B of the decomposition. (This definition makes sense only if we normalize the considered functions: for this reason, we require that $b(e) \leq 1$ for every edge e of H.) The main result of the paper is showing that bounded submodular width is the property that precisely characterizes the complexity of $\mathrm{CSP}(\mathcal{H})$:

THEOREM 1.3 (MAIN). *Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs. Assuming ETH, CSP($\mathcal{H}$) parameterized by the hypergraph H is fixed-parameter tractable if and only if $\mathcal{H}$ has bounded submodular width.*

Theorem 1.3 has an algorithmic side (algorithm for bounded submodular width) and a complexity side (hardness result for unbounded submodular width). Unlike previous width measures in the literature, where small value of the measure suggests a way of solving CSP($\mathcal{H}$) it is not at all clear how bounded submodular width is of any help. In particular, it is not obvious what submodular functions have to do with CSP instances. The main idea of our algorithm is that a CSP instance can be "split" into a small number of "uniform" CSP instances; for this purpose, we use a partitioning procedure inspired by a result of Alon et al. [3]. Conceptually, our algorithm goes beyond previous decomposition techniques in two ways. First, the tree decomposition that we use depends not only on the hypergraph, but on the actual constraint relations in the instance (we remark that this idea first appeared in [26] in a different context that does not directly apply to our problem). Second, we are not only decomposing the set of variables, but we also split the constraint relations. This way, we can apply different decompositions to different parts of the solution space.

The proof of the complexity side of Theorem 1.3 follows the same high-level strategy as the proof of Theorem 1.2 in [24]. In a nutshell, the argument of [24] is the following: if treewidth is large, then there is subset of vertices which is highly connected in the sense that the set does not have a small balanced separator; such a highly connected set implies that there is uniform concurrent flow (i.e., a compatible set of flows connecting every pair of vertices in the set); the paths in the flows can be used to embed the graph of a 3SAT formula; and finally this embedding can be used to reduce 3SAT to CSP. These arguments build heavily on well-known characterizations of treewidth and results from combinatorial optimization (such as the $O(\log k)$ integrality gap of sparsest cut). The proof of Theorem 1.3 follows this outline, but now no such well-known tools are available: we are dealing with hypergraphs and submodular functions in a way that was not explored before in the literature. One of the main difficulties of obtaining Theorem 1.3 is that we have to work in three different domains:

- **CSP instances.** As our goal is to investigate the existence of algorithms solving CSP, the most obvious domain is CSP instances. In light of previous results, we are especially interested in algorithms based on tree decompositions. For such algorithms, what matters is the existence of subsets of vertices such that restricting the instance to any of these subsets gives an instance with "small" number of solutions. In order to solve the instance, we would like to find a tree decomposition where every bag is such a small set.

- **Submodular functions.** Submodular width is defined in terms of submodular functions, thus submodular functions defined on hypergraphs is our second natural domain. We need to understand what large submodular width means, that is, what property of the submodular function and the hypergraph makes it impossible to obtain a tree decomposition where every bag has small value.

- **Flows and embeddings in hypergraphs.** In the hardness proof, our goal is to embed the graph of a 3SAT formula into a hypergraph. Thus we need to define an appropriate notion of embedding and study what guarantees the existence of embeddings with suitable properties. As in [24], we use the paths appearing in flows to construct embeddings. For our purposes, the right notion of flow is a collection of weighted paths where the total weight of the paths intersecting each hyperedge is at most 1. This notion of flows has not been studied in the literature before, thus we need to obtain basic tools and results on such flows.

A key question is how to find connections between these domains. As mentioned above and detailed in Section 3, we have a procedure that reduces a CSP instance into a set of uniform CSP instances, and the number of solutions on the different subsets of variables in a uniform CSP instance can be described by a submodular function. This method allows us to move from the domain of CSP instances to the domain of submodular functions. Section 4 is devoted to showing that if submodular width of a hypergraph is large, then there is a certain "highly connected" set in the hypergraph. Highly connected set is defined as a property of the hypergraph and has no longer anything to do with submodular functions. Thus this connection allows us to move from the domain of submodular functions to the study of hypergraphs. In Section 5, we show that a highly connected set in a hypergraph means that graphs can be efficiently embedded into the hypergraph. In particular, the graph of a 3SAT formula can be embedded into the hypergraph, which gives us (as shown in Section 6) a reduction from 3SAT to CSP($\mathcal{H}$). This connection allows us to move from the domain of embeddings back to the domain of CSP instances.

We can show that bounded submodular width is a strictly more general property than bounded fractional hypertree width, which was previously the most general property known to make CSP($\mathcal{H}$) FPT (see the full version of the paper). Thus Theorem 1.3 not only gives a complete characterization of the parameterized complexity of CSP($\mathcal{H}$), but its algorithmic side proves fixed-parameter tractability in a strictly more general case than what was known before.

2. PRELIMINARIES

Constraint satisfaction problems. We briefly recall the most important notions related to CSP. For more background, see e.g., [18, 12]. An instance of a *constraint satisfaction problem* is a triple (V, D, C), where: V is a set of variables, D is a domain of values, C is a set of constraints, $\{c_1, c_2, \ldots, c_q\}$. Each constraint $c_i \in C$ is a pair $\langle s_i, R_i \rangle$, where s_i is a tuple of variables of length m_i, called the *constraint scope,* and R_i is an m_i-ary relation over D, called the *constraint relation.*

For each constraint $\langle s_i, R_i \rangle$ the tuples of R_i indicate the allowed combinations of simultaneous values for the variables in s_i. The length m_i of the tuple s_i is called the *arity* of the constraint. A *solution* to a constraint satisfaction problem instance is a function f from the set of variables V to the domain of values D such that for each constraint $\langle s_i, R_i \rangle$ with $s_i = \langle v_{i_1}, v_{i_2}, \ldots, v_{i_m} \rangle$, the tuple $\langle f(v_{i_1}), f(v_{i_2}), \ldots, f(v_{i_m}) \rangle$ is a member of R_i. We say that an instance is *binary* if each constraint relation is binary, i.e., $m_i = 2$ for each constraint. It can be assumed that the instance does not contain two constraints $\langle s_i, R_i \rangle$, $\langle s_j, R_j \rangle$ with $s_i = s_j$, since in this case the two constraints can be replaced by the constraint $\langle s_i, R_i \cap R_j \rangle$. In the input, the relation in a constraint is represented by listing all the tuples of the constraint. We

denote by $\|I\|$ the size of the representation of the instance $I = (V, D, C)$.

Let $I = (V, D, C)$ be a CSP instance and let $V' \subseteq V$ be a nonempty subset of variables. The *projection* $\mathrm{pr}_{V'} I$ of I to V' is a CSP $I' = (V', D, C')$, where C' is defined the following way: For each constraint $c = \langle (v_1, \ldots, v_k), R \rangle$ having at least one variable in V', there is a corresponding constraint c' in C'. Suppose that $v_{i_1}, \ldots, v_{i_\ell}$ are the variables among $v_1, \ldots, v_k$ that are in V'. Then the constraint c' is defined as $\langle (v_{i_1}, \ldots, v_{i_\ell}), R' \rangle$, where the relation R' is the projection of R to the components $i_1, \ldots, i_\ell$, that is, R' contains an ℓ-tuple $(d'_1, \ldots, d'_\ell) \in D^\ell$ if and only if there is a k-tuple $(d_1, \ldots, d_k) \in R$ such that $d'_j = d_{i_j}$ for $1 \leq j \leq \ell$. Clearly, if f is a solution of I, then $f_{|V'}$ (f *restricted to* V') is a solution of $\mathrm{pr}_{V'} I$. For a subset $V' \subseteq V$, we denote by $\mathrm{sol}_I(V')$ the set of all solutions of $\mathrm{pr}_{V'} I$. If the instance I is clear from the context, we drop the subscript.

The *primal graph* (or *Gaifman graph*) of a CSP instance $I = (V, D, C)$ is a graph with vertex set V such that $u, v \in V$ are adjacent if and only if they appear together in the scope of some constraint. The *hypergraph* of a CSP instance $I = (V, D, C)$ is a hypergraph H with vertex set V, where $e \subseteq V$ is an edge if and only if there is a constraint whose scope is e. Let $\mathrm{CSP}(\mathcal{H})$ be the problem restricted to instances whose hypergraph is in $\mathcal{H}$.

Paths, separators, and flows in hypergraphs. A *path* P in hypergraph H is an ordered sequence $v_0, v_1, \ldots, v_r$ of vertices such that v_i and v_{i-1} are adjacent for every $1 \leq i < r$. We distinguish the endpoints of a path: vertex v_0 is the *first endpoint* of P and v_r is the *second endpoint* of P. A path is an $X - Y$ *path* if its first endpoint is in X and its second endpoint is in Y.

Let H be a hypergraph and $X, Y \subseteq V(H)$ be two (not necessarily disjoint) sets of vertices. An (X, Y)-*separator* is a set $S \subseteq V(H)$ of vertices such that every $X - Y$ path of H contains at least one vertex of S. In particular, this means that $X \cap Y \subseteq S$. An assignment $s : E(H) \to \mathbb{R}^+$ is a *fractional* (X, Y)-*separator* if every $X - Y$ path P is *covered* by s, that is, $\sum_{e \in E(H), e \cap P \neq \emptyset} s(e) \geq 1$. The *weight* of the fractional separator s is $\sum_{e \in E(H)} s(e)$.

Let H be a hypergraph and let $\mathcal{P}$ be the set of all paths in H. A *flow* of H is an assignment $f : \mathcal{P} \to \mathbb{R}^+$ such that $\sum_{P \in \mathcal{P}, P \cap e \neq \emptyset} f(P) \leq 1$ for every $e \in E(H)$. The *value* of the flow f is $\sum_{P \in \mathcal{P}} f(P)$. We say that a path P *appears* in flow f, or simply P is a *path of* f if $f(P) > 0$. For some $X, Y \subseteq V(H)$, an (X, Y)-*flow* is a flow f such that only $X - Y$ paths appear in f. A standard LP duality argument shows that the minimum weight of a fractional (X, Y)-separator is equal to the maximum value of an (X, Y)-flow. If f, f' are flows such that $f'(P) \leq f(P)$ for every path P, then f' is a *subflow* of f. The *sum* of the flows $f_1, \ldots, f_r$ is a mapping that assigns weight $\sum_{i=1}^r f_i(P)$ to each path P. If the sum of $f_1, \ldots, f_r$ is a flow, then we say that $f_1, \ldots, f_r$ are *compatible*.

Highly connected sets. An important step in understanding various width measures is showing that if the measure is large, then the (hyper)graph contains a highly connected set (in a certain sense). We define here the notion of highly connected that will be used in the paper. First, recall that a *fractional independent set* of a hypergraph H is a mapping $\mu : V(H) \to [0, 1]$ such that $\sum_{v \in e} \mu(v) \leq 1$ for every $e \in E(H)$. We extend functions on the vertices of H to subsets of vertices of H the natural way by setting $\mu(X) := \sum_{v \in X} \mu(v)$.

Let μ be a fractional independent set of hypergraph H and let $\lambda > 0$ be a constant. We say that a set $W \subseteq V(H)$ is (μ, λ)-*connected* if for any two disjoint sets $A, B \subseteq W$, the minimum weight of a fractional (A, B)-separator is at least $\lambda \cdot \min\{\mu(A), \mu(B)\}$. Note that if W is (μ, λ)-connected, then W is (μ, λ')-connected for every $\lambda' < \lambda$ and every $W' \subseteq W$ is also (μ, λ)-connected. Informally, if W is (μ, λ)-lambda connected for some fractional independent set μ such that $\mu(W)$ is "large", then we call W a highly connected set. For $\lambda > 0$, we denote by $\mathrm{con}_\lambda(H)$ the maximum of $\mu(W)$, taken over all (μ, λ)-connected set W of H. Throughout the paper, λ can be thought of as a sufficiently small universal constant, say, 0.001.

Embeddings. The hardness result presented in the paper and earlier hardness results for $\mathrm{CSP}(\mathcal{H})$ [19, 26, 24] are based on embedding some other problem (with a certain graph structure) in a CSP instance whose hypergraph is a member of $\mathcal{H}$. Thus we need appropriate notions of embedding a graph in a (hyper)graph. A crucial difference between the proof of Theorem 1.1 in [19] and the proof of Theorem 1.2 in [24] is that the former result is a based on finding a minor embedding of a grid, while the latter result uses an embedding where the images of distinct vertices are not necessarily disjoint, but can overlap in a controlled way. We define such embeddings the following way. We say that two sets of vertices $X, Y \subseteq V(H)$ *touch* if either $X \cap Y \neq \emptyset$, or there is an edge $e \in E(H)$ intersecting both X and Y. An *embedding* of graph G into hypergraph H is a mapping ψ that maps each vertex of H to a connected subset of $V(G)$ such that if u and v are adjacent in G, then $\psi(u)$ and $\psi(v)$ touch. The *depth* of a vertex $v \in V(H)$ in embedding ψ is $d_\psi(v) := |\{u \in V(G) \mid v \in \psi(u)\}|$, the number of vertices of G whose images contain v. The *vertex depth* of the embedding is $\max_{v \in V(H)} d_\psi(v)$. Because in our case we want to control the size of the constraint relations, we need a notion of depth that is sensitive to "what the edges see." We define *edge depth* of ψ to be $\max_{e \in E(H)} \sum_{v \in e} d_\psi(v)$. Equivalently, we can define edge depth as the maximum of $\sum_{v \in V(G)} |\psi(v) \cap e|$, taken over all edges of e of H.

Trivially, for any graph G and hypergraph H, there is an embedding of G into H having vertex depth and edge depth at most $|V(G)|$. If G has m edges and no isolated vertices, then $|V(G)|$ is at most $2m$. We are interested in how much we can gain compared to this trivial solution of depth $O(m)$. We define the *embedding power* $\mathrm{emb}(H)$ to be the maximum (supremum) value of α for which there is an integer m_α such that every graph G with $m \geq m_\alpha$ edges has an embedding into H with edge depth m/α.

Width parameters We follow the framework of width functions introduced by Adler [1]. A *tree decomposition* of a hypergraph H is a tuple $(T, (B_t)_{t \in V(T)})$, where T is a tree and $(B_t)_{t \in V(T)}$ is a family of subsets of $V(H)$ satisfying the following two conditions: (1) for each $e \in E(H)$ there is a node $t \in V(T)$ such that $e \subseteq B_t$, and (2) for each $v \in V(H)$ the set $\{t \in V(T) \mid v \in B_t\}$ is connected in T. The sets B_t are called the *bags* of the decomposition. Let $f : 2^{V(H)} \to \mathbb{R}^+$ be a function that assigns a nonnegative real number to each nonempty subset of vertices. The f-*width* of a tree-decomposition $(T, (B_t)_{t \in V(T)})$ is $\max\{f(B_t) \mid t \in V(T)\}$. The f-*width* of a hypergraph H is the minimum of the f-widths of all its tree decompositions. In other words,

f-width$(H) \leq w$ if and only if there is a tree decomposition of H where $f(B) \leq w$ for every bag B.

The main idea of tree decomposition based algorithms is that if we have a tree decomposition for instance I such that at most C assignments on B_t have to be considered for each bag B_t, then the problem can be solved by dynamic programming in time polynomial in C and $\|I\|$. The various width notions try to guarantee the existence of such decompositions. For example, the simplest such notion, treewidth, can be defined as $\mathrm{tw}(H) := s$-width(H), where $s(B) = |B| - 1$. Further width notions defined in the literature, such as generalized hypertree width and fractional hypertree width can also be conveniently defined using this setup, see [1].

We generalize the notion of f-width from a single function f to a class of functions $\mathcal{F}$. Let $\mathcal{F}$ be an arbitrary (possibly infinite) class of functions that assign nonnegative real numbers to nonempty subsets of vertices. The *$\mathcal{F}$-width* of a hypergraph H is $\mathcal{F}$-width$(H) := \sup\{f\text{-width}(H) \mid f \in \mathcal{F}\}$. Thus if $\mathcal{F}$-width$(H) \leq k$, then for every $f \in \mathcal{F}$, hypergraph H has a tree decomposition with f-width at most k. Note that this tree decomposition can be different for the different functions f. For normalization purposes, we consider only functions f on $V(H)$ that are *edge-dominated:* $f(e) \leq 1$ holds for every $e \in E(H)$. The main new definition of the paper is submodular width:

Definition 2.1. *A function $b : 2^{V(H)} \to \mathbb{R}^+$ is* submodular *if $b(X) + b(Y) \geq b(X \cap Y) + b(X \cup Y)$ holds for every $X, Y \subseteq V(H)$. Given a hypergraph H, let $\mathcal{F}$ contain the edge-dominated monotone submodular functions on $V(H)$. The* submodular width $\mathrm{subw}(H)$ *of hypergraph H is $\mathcal{F}$-width(H).*

3. FROM CSP INSTANCES TO SUBMODULAR FUNCTIONS

We prove the main algorithmic result in this section: CSP$(\mathcal{H})$ is FPT if $\mathcal{H}$ has bounded submodular width.

Theorem 3.1. *Let $\mathcal{H}$ be a class of hypergraphs such that $\mathrm{subw}(H) \leq c_0$ for every $H \in \mathcal{H}$. Then CSP$(\mathcal{H})$ can be solved in time $2^{2^{O(|V(H)|)}} \cdot \|I\|^{O(c_0)}$.*

The proof of Theorem 3.1 is based on two main ideas:

1. A CSP instance I can be decomposed into a bounded number of "uniform" CSP instances $I_1, \ldots, I_t$ (Lemma 3.9). Here uniform means that if $B \subseteq A$ are two sets of variables, then every solution of $\mathrm{pr}_B I_j$ has roughly the same number of extensions to $\mathrm{pr}_A I_j$.
2. If I is a uniform CSP instance, then (the logarithm of) the number of solutions on the different projections of I can be described by an edge-dominated submodular function (Lemma 3.10). Therefore, if the hypergraph H of I has bounded submodular width, then it follows that there is a tree decomposition where every bag has a small number of solutions.

In the implementation of the first idea (Lemma 3.9), we guarantee uniformity only to subsets of variables that are "small" in the following hereditary sense:

Definition 3.2. *For a CSP instance I and $M \geq 1$, a set $S \subseteq V$ is M-*small *if $|\mathrm{sol}_I(S')| \leq M$ for every $S' \subseteq S$.*

It is not difficult to find all the M-small sets, and every solution of the instances projected onto these sets:

Lemma 3.3. *Let $I = (V, D, C)$ be a CSP instance and $M \geq 1$ an integer. There is an algorithm with running time $2^{O(|V|)} \cdot poly(\|I\|, M)$ that finds the set $\mathcal{S}$ of all M-small sets $S \subseteq V$ and constructs $\mathrm{sol}_I(S)$ for each such $S \in \mathcal{S}$.*

The following definition gives the precise notion of uniformity that we use:

Definition 3.4. *Let $I = (V, D, C)$ be a CSP instance. For $B \subseteq A \subseteq V$ and an assignment $b : B \to D$, let $\mathrm{sol}_I(A|B = b) := \{a \in \mathrm{sol}_I(A) \mid a(x) = b(x)$ for every $x \in B\}$, the set of all extensions of b to a solution of $\mathrm{pr}_A I$. Let $\max_I(A|B) = \max_{b \in \mathrm{sol}_I(B)} |\mathrm{sol}_I(A|B = b)|$. We say that $A \subseteq V$ is c-*uniform *(for some integer c) if, for every $B \subseteq A$, $\max_I(A|B) \leq c|\mathrm{sol}_I(A)|/|\mathrm{sol}_I(B)|$. We define $\max_I(A|\emptyset) = |\mathrm{sol}_I(A)|$ and $\max_I(\emptyset|\emptyset) = 1$. An instance is (N, c, ϵ)-*uniform *if every N^c-small set is N^ϵ-uniform.*

Proposition 3.5. *For every $B \subseteq A \subseteq V$ and $C \subseteq V$, we have*

1. $\max(A|B) \geq |\mathrm{sol}(A)|/|\mathrm{sol}(B)|$,
2. $\max(A|B) \geq \max(A \cup C|B \cup C)$.

We want to avoid dealing with assignments $b \in \mathrm{sol}(B)$ that cannot be extended to a member of $\mathrm{sol}(A)$ for some $A \supseteq B$ (that is, $\mathrm{sol}(A|B = b) = \emptyset$). We require that there is no such unextendable b if A and B are M-small:

Definition 3.6. *A CSP instance is M-*consistent *if $\mathrm{sol}(B) = \mathrm{pr}_A \mathrm{sol}(A)$ for all M-small sets $B \subseteq A$. A CSP instance is* nontrivial *if $\mathrm{sol}(\{v\}) \neq \emptyset$ for any $v \in V$.*

Proposition 3.7. *If I is an M-consistent nontrivial CSP instance, then $\mathrm{sol}(S) \neq \emptyset$ for every M-small set S.*

We can achieve M-consistency by throwing away partial solutions that violate the requirements. Instance $I' = (V, D, C')$ is a *refinement* of $I = (V, D, C)$ if for every $\langle s, R\rangle \in C$, there is a $\langle s, R'\rangle \in C'$ such that $R' \subseteq R$.

Lemma 3.8. *Let $I = (V, D, C)$ be a CSP instance and $M \geq 1$ an integer. There is an algorithm with running time $2^{O(|V|)} \cdot poly(\|I\|, M)$ that produces an M-consistent CSP instance I' that is a refinement of I with $\mathrm{sol}(I) = \mathrm{sol}(I')$.*

Our algorithm for decomposing a CSP instance into uniform CSP instances is inspired by a combinatorial result of Alon et al. [3], which shows that, for every fixed n, an n-dimensional point set S can be partitioned into polylog$(|S|)$ $O(1)$-uniform classes. We follow the same proof idea: the instance is split into two instances if uniformity is violated somewhere, and we analyze the change of a weight function to bound the number of splits performed. However, the parameter setting is different in our proof: we want to partition into $f(|V|)$ classes, but we are satisfied with weaker uniformity. A minor difference is that we require uniformity only on N^c-small sets.

Lemma 3.9. *Let $I = (V, D, C)$ be a CSP instance and let N, c be integers and $\epsilon > 0$. There is an algorithm with running time $2^{2^{O(|V|)} \cdot c/\epsilon} \cdot \mathrm{poly}(\|I\|, N^c)$ that produces a set of (N, c, ϵ)-uniform N^c-consistent nontrivial instances $I_1, \ldots, I_t$ with $0 \leq t \leq 2^{2^{O(|V|)} \cdot c/\epsilon}$, all on the set V of variables, such that*

1. *every solution of I is a solution of exactly one I_i,*
2. *for every $1 \leq i \leq t$, instance I_i is a refinement of I.*

PROOF. The main step of the algorithm takes a CSP instance I and either makes it (N, c, ϵ)-uniform and N^c-consistent, or splits it into two instances I_{small}, I_{large}. By applying the main step recursively on I_{small} and I_{large}, we eventually arrive to a set of (N, c, ϵ)-uniform N^c-consistent instances. We will argue that the number of constructed instances is $2^{2^{O(|V|)} \cdot c/\epsilon}$.

In the main step, we first check if the instance is trivial; in this case we can stop with $t = 0$. Otherwise, we invoke the algorithm of Lemma 3.8 to obtain an N^c-consistent refinement of the instance, without losing any solution. Next we check if this N^c-consistent instance I is (N, c, ϵ)-uniform. This can be tested in time $2^{O(|V|)} \cdot \text{poly}(\|I\|, N^c)$ if we use Lemma 3.3 to find all the N^c-small sets and the corresponding sets of solutions. Suppose that N^c-small sets $B \subseteq A$ violate uniformity, that is, $\max(A|B) > M^\epsilon |\operatorname{sol}(A)|/|\operatorname{sol}(B)|$. Let $\operatorname{sol}_{\text{small}}(B)$ contain those tuples b for which $|\operatorname{sol}(A|B = b)| \leq \sqrt{M^\epsilon}|\operatorname{sol}(A)|/|\operatorname{sol}(B)|$ and let $\operatorname{sol}_{\text{large}}(B) = \operatorname{sol}(B) \setminus \operatorname{sol}_{\text{small}}(B)$. Note that

$$|\operatorname{sol}(A)| \geq |\operatorname{sol}_{\text{large}}(B)| \cdot (\sqrt{M^\epsilon}|\operatorname{sol}(A)|/|\operatorname{sol}(B)|)$$

(as every tuple $b \in \operatorname{sol}_{\text{large}}(B)$ has at least $\sqrt{M^\epsilon}|\operatorname{sol}(A)|/|\operatorname{sol}(B)|$ extensions to A), hence $|\operatorname{sol}_{\text{large}}(B)| \leq |\operatorname{sol}(B)|/\sqrt{M^\epsilon}$. Let instance I_{small} (resp., I_{large}) be obtained from I by adding the constraint $\langle B, \operatorname{sol}_{\text{small}}(B)\rangle$ (resp., $\langle B, \operatorname{sol}_{\text{large}}(B)\rangle$). Clearly, the set of solutions of I is the disjoint union of the sets of solutions of I_{small} and I_{large}. This completes the description of the main step.

It is clear that if the recursive procedure stops, then the instances at the leaves of the recursion satisfy the two requirements. We show that the height of the recursion tree can be bounded from above by a function $h(|V|, c, \epsilon)$ depending only on $|V|$, c, and ϵ; in particular, this shows that the recursive algorithm eventually stops and produces at most $2^{h(|V|,c,\epsilon)}$ instances.

Let us consider a path in the recursion tree starting at the root, and let I^1, I^2, ..., I^p be the corresponding N^c consistent instances. If a set S is N^c-small in I^j, then it is N^c-small in $I^{j'}$ for every $j' > j$: the main step cannot increase $|\operatorname{sol}(S)|$ for any S. Thus, with the exception of at most $2^{|V|}$ values of j, instances I^j and I^{j+1} have the same N^c-small sets. Let us consider a subpath I^x, ..., I^y such that all these instances have the same N^c-small sets. We show that the length of this subpath is at most $O(3^{|V|} \cdot c/\epsilon)$, hence $p = O(2^{|V|} \cdot 3^{|V|} \cdot c/\epsilon)$.

For the instance I^j, let us define the following weight:

$$W^j = \sum_{\substack{\emptyset \subseteq B \subseteq A \subseteq V \\ A,\, B \text{ are } N^c\text{-small}}} \log \max_{I^j}(A|B).$$

We bound the length of the subpath $I^x, \ldots, I^y$ by analyzing how this weight changes in I_{large} and I_{small} compared to I. Note that $0 \leq W^j \leq 3^{|V|} \log N^c = 3^{|V|} \cdot c \log N$: the sum consists of at most $3^{|V|}$ terms and (as A is N^c-small and the instance I^j is N^c consistent and nontrivial) $\max_{I^j}(A|B)$ is between 1 and N^c. We show that $W^{j+1} \leq W^j - (\epsilon/2)\log N$, which immediately implies that the length of the subpath is $O(3^{|V|} \cdot c/\epsilon)$. Let us inspect how W^{j+1} changes compared to W^j. Since I^j and I^{j+1} have the same N^c-small sets, no new term can appear in W^{j+1}. It is clear that $\max_{I^{i+1}}(A|B)$ cannot be greater than $\max_{I^i}(A|B)$ for any A, B. However, there is at least one term that strictly decreases. Suppose first that I^{j+1} was obtained from I^j by adding the constraint $\langle B, \operatorname{sol}_{\text{small}}(B)\rangle$. Then

$$\log \max_{I^{j+1}}(A|B) \leq \log \sqrt{N^\epsilon} \frac{|\operatorname{sol}_{I^j}(A)|}{|\operatorname{sol}_{I^j}(B)|}$$
$$\leq \log(\max_{I^j}(A|B)/\sqrt{N^\epsilon}) = \log \max_{I^j}(A|B) - (\epsilon/2)\log N.$$

On the other hand, if I^{j+1} was obtained by adding the constraint $\langle B, \operatorname{sol}_{\text{large}}(B)\rangle$, then

$$\log \max_{I^{j+1}}(B|\emptyset) = \log |\operatorname{sol}_{I^{j+1}}(B)|$$
$$\leq \log(|\operatorname{sol}_{I^j}(B)|/\sqrt{N^\epsilon}) = \log \max_{I^j}(B|\emptyset) - (\epsilon/2)\log N.$$

In both cases, we get that at least one term decreases by at least $(\epsilon/2)\log N$. □

Assume for a moment that we have a 1-uniform instance I with hypergraph H. Note that by Prop 3.5(1), this means that $\max(A|B) = |\operatorname{sol}(A)|/|\operatorname{sol}(B)|$. Suppose that every constraint contains at most N tuples and let us define the function $b(S) = \log_N |\operatorname{sol}(S)|$. For every edge $e \in E(H)$, there is a corresponding constraint, which has at most N tuples by the definition of N. Thus $|\operatorname{sol}(e)| \leq N$ and hence $b(e) \leq 1$ for every $e \in E(H)$, that is, b is edge dominated. The crucial observation of this section is that this function b is submodular:

$$\begin{aligned}
&b(X) + b(Y) \\
&= \log_N |\operatorname{sol}(X)| + \log_N \left(|\operatorname{sol}(X \cap Y)| \frac{|\operatorname{sol}(Y)|}{|\operatorname{sol}(X \cap Y)|} \right) \\
&= \log_N |\operatorname{sol}(X)| + \log_N \left(|\operatorname{sol}(X \cap Y)| \max(Y|X \cap Y) \right) \\
&\geq \log_N |\operatorname{sol}(X)| + \log_N \left(|\operatorname{sol}(X \cap Y)| \max(X \cup Y|X) \right) \\
&= \log_N \left(|\operatorname{sol}(X)| \frac{|\operatorname{sol}(X \cup Y)|}{|\operatorname{sol}(X)|} \right) + \log_N |\operatorname{sol}(X \cap Y)| \\
&= b(X \cap Y) + b(X \cup Y)
\end{aligned}$$

(the equalities follow from 1-uniformity; the inequality uses Prop. 3.5(2) with $A = Y$, $B = X \cap Y$, $C = X$). Therefore, if $\operatorname{subw}(H) \leq c$, then H has a tree decomposition where $b(B) \leq c$ and hence $|\operatorname{sol}(B)| \leq N^c$ for every bag B. This allows us to solve the problem by dynamic programming in time polynomial in N^c.

Lemma 3.9 guarantees some uniformity for the created instances, but not perfect 1-uniformity and only for the N^c-small sets. In Lemma 3.10, we define b in a slightly different way: we add small terms to correct errors arising from weaker uniformity and we truncate the function at large values (i.e., for sets that are not N^c-small).

LEMMA 3.10. *Let $I = (V, D, C)$ be a CSP instance with hypergraph H such that $|\operatorname{sol}(e)| \leq N$ for every $e \in E(H)$. If I is N^c-consistent and (N, c, ϵ^3)-uniform for some $c \geq 1$ and $\epsilon := 1/|V|$, then the following function b is an edge-dominated, monotone, submodular function on $V(H)$:*

$$b(S) := \begin{cases} (1-\epsilon)\log_N |\operatorname{sol}(S)| + 2\epsilon^2|S| - \epsilon^3|S|^2 & \text{if } S \text{ is } N^c\text{-small,} \\ (1-\epsilon)c + 2\epsilon^2|S| - \epsilon^3|S|^2 & \text{otherwise.} \end{cases}$$

If $\mathrm{subw}(H) \leq (1-\epsilon)c$, then there is a tree decomposition where every bag is N^c-small, and we can use this tree decomposition to find a solution. In fact, in this case N^c-consistency implies that every nontrivial instance has a solution. Putting together this observation with Lemmas 3.9 and 3.10, the proof of Theorem 3.1 follows.

4. FROM SUBMODULAR FUNCTIONS TO HIGHLY CONNECTED SETS

The aim of this section is to show that if a hypergraph H has large submodular width, then there is a large highly connected set in H. The main result is the following:

THEOREM 4.1. *For every sufficiently small constant $\lambda > 0$, the following holds. Let b be an edge-dominated monotone submodular function of H. If the b-width of H is greater than $\frac{3}{2}(w+1)$, then $\mathrm{con}_\lambda(W) \geq w$.*

For the proof of Theorem 4.1, we need to show that if there is no tree decomposition where $b(B)$ is small for every bag B, then a highly connected set exists. There is a standard recursive procedure that either builds a tree decomposition or finds a highly connected set (see e.g., [13, Section 11.2]). Simplifying somewhat, the main idea is that if a the graph can be decomposed into smaller graphs by splitting a certain set of vertices into two parts, then a tree decomposition for each part is constructed using the algorithm recursively, and the tree decompositions for the parts are joined in an appropriate way to obtain a tree decomposition for the original problem. On the other hand, if the set of vertices cannot be split, then we can conclude that it is highly connected. This high-level idea has been applied for various notions of tree decompositions [30, 28, 2, 29], and it turns out to be useful in our context as well. However, we need to overcome two major difficulties:

1. Highly connected set in our context is defined as not having certain *fractional separators* (i.e., weight assignments). However, if we want to build a tree decomposition in a recursive manner, we need *integer separators* (i.e., subsets of vertices) that decompose the hypergraph into smaller parts.
2. Measuring the sizes of sets with a submodular function b can lead to problems, since the size of the union of two sets can be much smaller the sum of the sizes of the two sets. We need the property that, roughly speaking, removing a "large" part from a set makes it "much smaller." For example, if A and B are components of $H \setminus S$, and both $b(A)$ and $b(B)$ are large, then we need the property that both of them are much smaller than $b(A \cup B)$. Adler [1, Section 4.2] investigates the relation between some notion of highly connected set and f-width, but assumes that f is additive: if A and B do not touch, then $f(A \cup B) = f(A) + f(B)$. However, for a submodular function b, it very well may be that $b(A) = b(B) = b(A \cup B)$.

To overcome the first difficulty, we have to understand what fractional separation really means. We show that if there is a fractional (A,B)-separator of weight w, then for *every* edge-dominated monotone submodular function b, there is an (A,B)-separator S with $b(S) = O(w)$. Note that this separator S can be different for different functions b, so we are not claiming that there is a single (A,B)-separator S that is small in every b. The converse is also true (we omit the proof), thus this gives a novel characterization of fractional separation, tight up to a constant factor. This result is the key idea that allows us to move from the domain of submodular functions to the domain of pure hypergraph properties: if there is no (A,B)-separator such that $b(S)$ is small, then we know that there is no small fractional (A,B)-separator, which is a property of the hypergraph H only and has no longer anything to do with the submodular function b.

To overcome the second difficulty, we introduce a transformation that turns a monotone submodular function b on $V(H)$ into a function b^* that encodes somehow the neighborhood structure of H as well. The new function b^* is no longer monotone and submodular, but is has a number of remarkable properties, for example, b^* remains edge dominated and $b^*(S) \geq b(S)$ for every set $S \subseteq V(H)$, implying that b^*-width is not smaller than b-width. The main idea is to prove Theorem 4.1 for b^*-width instead of b-width. Because of the way b^* encodes the neighborhoods, the second difficulty will disappear: for example, it will be true that $b^*(A \cup B) = b^*(A) + b^*(B)$ if there are no edges between A and B, that is, b^* is additive on disjoint components. Lemma 4.3 formulates (in a somewhat technical way) the exact property of b^* that we will need. It turns out that the result mentioned in the previous paragraph remains true with b replaced by b^*: if there is a fractional (A,B)-separator of weight w, then there is an (A,B)-separator S such that not only $b(S)$, but even $b^*(S)$ is $O(w)$.

The function b^*. We define the function b^* the following way. Let H be a hypergraph and let b be a monotone submodular function defined on $V(H)$. Let $S_{V(H)}$ be the set of all permutations of $V(H)$. For a permutation $\pi \in S_{V(H)}$, let $N^-_\pi(v)$ be the neighbors of v preceding v in the ordering π. For $\pi \in S_{V(H)}$ and $Z \subseteq V(H)$, we define

$$\partial b_{\pi,Z}(v) := b(v \cup (N^-_\pi(v) \cap Z)) - b(N^-_\pi(v) \cap Z).$$

In other words, $\partial b_{\pi,Z}(v)$ is the marginal value of v with respect to the set of its neighbors in Z preceding it. We abbreviate $\partial b_{\pi,V(H)}$ by ∂b_π. We extend the definition to subsets by $\partial b_{\pi,Z}(S) := \sum_{v \in S} \partial b_{\pi,Z}(v)$. Next, we define

$$b_\pi(Z) := \partial b_{\pi,Z}(Z) = \sum_{v \in Z} \partial b_{\pi,Z}(v) \qquad b^*(Z) := \min_{\pi \in S_{V(H)}} b_\pi(Z).$$

Thus $b_\pi(Z)$ is the sum of the marginal values with respect to a given ordering, while $b^*(Z)$ is the smallest possible sum taken over all possible orderings.

PROPOSITION 4.2. *Let H be a hypergraph and let b be a monotone submodular function defined on $V(H)$. For every $\pi \in S_{V(H)}$ and $Z \subseteq V(H)$ we have*

1. $b_\pi(Z) \geq b(Z)$,
2. $b^*(Z) \geq b(Z)$,
3. $b_\pi(Z) = b(Z)$ *if Z is a clique,*
4. $\partial b_{\pi,Z_1}(v) \leq \partial b_{\pi,Z_2}(v)$ *if* $Z_2 \subseteq Z_1$,
5. $\partial b_\pi(v) \leq \partial b_{\pi,Z}(v)$,
6. $b^*(X \cup Y) \leq b^*(X) + b^*(Y)$.

The following property of b^* allows us to avoid the second difficulty described at the beginning of Section 4.

LEMMA 4.3. *Let H be a hypergraph, let b be a monotone submodular function defined on $V(H)$ and let W be a set of*

vertices. Let π_W be an ordering of $V(H)$, and let $\mu(v) := \partial b_{\pi_W,W}(v)$ for $v \in W$ and $\mu(v) = 0$ otherwise. Let $A, B \subseteq W$ be two disjoint sets, and let S be an (A,B)-separator. If $b^*(S) < \mu(A), \mu(B)$, then $b^*((C \cap W) \cup S) < \mu(W)$ for every component C of $H \setminus S$.

Next we show that having a small fractional (A,B)-separator means that for every monotone edge-dominated submodular function b, there is an (A,B)-separator S such that $b^*(S)$ (and hence $b(S)$) is small. The proof is based on a standard trick that is often used for rounding fractional solutions for separation problems: we define a distance function and show by an averaging argument that cutting at some distance t gives a small separator. However, in our setting, we need significant new ideas to make this trick work: the main difficulty is that the cost function b is defined on *subsets* of vertices and is not a function defined by the cost of vertices. To overcome this problem, we use the function $\partial b_\pi(v)$ to assign a cost to every single vertex.

THEOREM 4.4. *Let H be a hypergraph, $X, Y \subseteq V(H)$ two sets of vertices, and $b : V(H) \to \mathbb{R}^+$ an edge-dominated monotone submodular function. Suppose that s is a fractional (X,Y)-separator of weight at most w. Then there is an (X,Y)-separator $S \subseteq V(H)$ with $b^*(S) = O(w)$.*

PROOF. Let us define $x(v) := \max\{1, \sum_{e \in E(H), v \in e} s(e)\}$. It is clear that if P is a path from X to Y, then $\sum_{v \in P} x(v) \geq 1$. We define the distance $d(v)$ to be the minimum of $\sum_{v' \in P} x(v')$, taken over all paths from X to v (this means that $d(v) > 0$ is possible for some $v \in X$). It is clear that $d(v) \geq 1$ for every $v \in Y$. Let us associate the closed interval $\iota(v) = [d(v) - x(v), d(v)]$ to each vertex v. If v is in X, then the left endpoint of $\iota(v)$ is 0, while if v is in Y, then the right endpoint of $\iota(v)$ is at least 1. The following claim is easy to see:

Claim 1: If u and v are adjacent, then $\iota(u) \cap \iota(v) \neq \emptyset$.

The *class* of a vertex $v \in V(H)$ is the largest integer $\kappa(v)$ such that $x(v) \leq 2^{-\kappa(v)}$, and we define $\kappa(v) := \infty$ if $x(v) = 0$. Recall that $x(v) \leq 1$, thus $\kappa(v)$ is nonnegative. The *offset* of a vertex v is the unique value $0 \leq \alpha < 2 \cdot 2^{-\kappa(v)}$ such that $d(v) = i(2 \cdot 2^{-\kappa(v)}) + \alpha$ for some integer i. Let us define an ordering $\pi = (v_1, \ldots, v_n)$ of $V(H)$ such that $\kappa(v)$ is nondecreasing, and among vertices having the same class, the offset is nondecreasing.

Let directed graph D be the orientation of the primal graph of H such that if v_i and v_j are adjacent and $i < j$, then there is a directed edge $\overrightarrow{v_i v_j}$ in D. If P is a directed path in D, then the *width* of P is the length of the interval $\bigcup_{v \in P} \iota(v)$ (note that by Claim 1, this union is indeed an interval). The following claim bounds the maximum possible width of a directed path:

Claim 2: If P is a directed path D starting at v, then the width of P is at most $16x(v)$.

To prove Claim 2, we first show that if every vertex of P has the same class $\kappa(v)$, then the width of P is at most $4 \cdot 2^{-\kappa(v)}$. Since the class is nondecreasing along the path, we can partition the path into subpaths such that every vertex in a subpath has the same class and the classes are distinct on the different subpaths. The width of P is at most the sum of the widths of the subpaths, which is at most $\sum_{i \geq \kappa(v)} 4 \cdot 2^{-i} = 8 \cdot 2^{-\kappa(v)} \leq 16x(v)$.

Suppose now that every vertex of P has the same class $\kappa(v)$ as the first vertex v and let $h := 2^{-\kappa(v)}$. As the offset is nondecreasing, path P can be partitioned into two parts: a subpath P_1 containing vertices with offset at least h, followed by a subpath P_2 containing vertices with offset less than h (one of P_1 and P_2 can be empty). We show that each of P_1 and P_2 has width at most $2h$, which implies that the width of P is at most $4h$. Observe that if $v \in P_1$ and $\iota(v)$ contains a point $i \cdot 2h$ for some integer i, then, considering $x(v) \leq h$ and the bounds on the offset of v, this is only possible if $\iota(v) = [i \cdot 2h, i \cdot 2h + h]$, i.e., $i \cdot 2h$ is the left endpoint of $\iota(v)$. Thus if $I_1 = \bigcup_{v \in P_1} \iota(v)$ contains $i \cdot 2h$, then it is the left endpoint of I_1. Therefore, I_1 can contain $i \cdot 2h$ for at most one value of i, which immediately implies that the length of I_1 is at most $2h$. We can argue similarly for P_2.

Claim 3: $\sum_{v \in V(H)} x(v)c(v) \leq w$ for $c(v) := \partial b_\pi(v)$.

Let us examine the contribution of an edge $e \in E(H)$ with value $s(e)$ to the sum. For every $v \in e$, edge e increases the value $x(v)$ by at most $s(e)$. Thus the total contribution of e is at most

$$s(e) \cdot \sum_{v \in e} c(v) = s(e) \cdot \sum_{v \in e} \partial b_\pi(v) \leq s(e) \cdot \sum_{v \in e} \partial b_{\pi,e}(v) = s(e) b_\pi(e) \leq s(e) b(e) \leq s(e),$$

where the first inequality follows Prop. 4.2(5); the second inequality follows form Prop. 4.2(3); the last inequality follows from the fact that b is edge dominated. Therefore, $\sum_{v \in V(H)} x(v)c(v) \leq \sum_{e \in E(H)} s(e) \leq w$, proving Claim 3.

We define $\mathcal{C}(S)$ to be the set of all vertices from which a vertex of S is reachable on a directed path in D.

Claim 4: For every $S \subseteq V(H)$, $\sum_{v \in \mathcal{C}(S)} c(v) = b_\pi(\mathcal{C}(S))$.

Observe that for any $v \in \mathcal{C}(S)$, every inneighbor of v is also in $\mathcal{C}(S)$, hence $N^-_\pi(v) \subseteq \mathcal{C}(S)$. Therefore, $\partial b_{\pi,\mathcal{C}(S)}(v) = \partial b_\pi(v) = c(v)$ and Claim 4 follows.

Now we are ready to prove the lemma. Let $S(t)$ be the set of all vertices $v \in V(H)$ for which $t \in \iota(v)$ and let $S_C(t) = \mathcal{C}(S(t))$. Observe that for every $0 \leq t \leq 1$, the set $S(t)$ (and hence $S_C(t)$) separates X from Y. We use an averaging argument to show that there is a $0 \leq t \leq 1$ for which $b_\pi(S_C(t))$ is $O(w)$. As $b^*(S_C(t)) \leq b_\pi(S_C(t))$, the set $S_C(t)$ satisfies the requirement of the lemma.

If we can show that $\int_0^1 b_\pi(S_C(t))dt = O(w)$, then the existence of the required t follows. Let $I_v(t) = 1$ if $v \in S_C(t)$ and let $I_v(t) = 0$ otherwise. If $I_v(t) = 1$, then there is a path P in D from v to $S(t)$. By Claim 2, the width of this path is at most $16x(v)$, thus $t \in [d(v) - 16x(v), d(v) + 15x(v)]$. Therefore, $\int_0^1 I_v(t)dt \leq 31x(v)$. Now we have

$$\int_0^1 b_\pi(S_C(t))dt = \int_0^1 \sum_{v \in S_C(t)} c(v)dt = \int_0^1 \sum_{v \in V(H)} c(v) I_v(t) dt = \sum_{v \in V(H)} c(v) \int_0^1 I_v(t)dt \leq 31 \sum_{v \in V(H)} x(v)c(v) \leq 31w$$

(we used Claim 4 in the first equality and Claim 3 in the last inequality). □

By Prop 4.2(2), the following lemma implies Theorem 4.1. The proof is a recursive procedure finding a tree decomposition, using Lemmas 4.3 and 4.4 to overcome the difficulties described at the beginning of the section.

LEMMA 4.5. *Let b be an edge-dominated monotone submodular function of H. If b^*-width$(H) > \frac{3}{2}(w+1)$, then $\mathrm{con}_\lambda(W) \geq w$ (for some universal constant λ).*

5. FROM HIGHLY CONNECTED SETS TO EMBEDDINGS

We show that the existence of highly connected sets implies that the hypergraph has large embedding power:

THEOREM 5.1. *For every sufficiently small $\lambda > 0$ and hypergraph H, there is a constant $m_{H,\lambda}$ such that every graph G with $m \geq m_{H,\lambda}$ edges has an embedding into H with edge depth $O(m/(\lambda^{\frac{3}{2}} \mathrm{con}_\lambda(H)^{\frac{1}{4}}))$.*

Our strategy is similar to [24]: we show that a highly connected set implies that a uniform concurrent flow exists; the paths appearing in the uniform concurrent flow can be used to embed the line graph of a complete graph; and every graph has an appropriate embedding in the line graph of a complete graph. To make this strategy work, we need generalizations of concurrent flows, multicuts, and multicommodity flows in our hypergraph setting and we need to obtain results that connect these concepts to highly connected sets. Let W be a set of vertices and let $(X_1, \ldots, X_k)$ be a partition of W. A *uniform concurrent flow of value ϵ on $(X_1, \ldots, X_k)$* is a compatible set of $\binom{k}{2}$ flows $F_{i,j}$ $(1 \leq i < j \leq k)$ where $F_{i,j}$ is an (X_i, X_j)-flow of value ϵ. We will need a uniform concurrent flow connecting a set of cliques, thus our first goal is to find a highly connected set that can be partitioned into k cliques in an appropriate way. (Proofs of this section appear in the full version of the paper.)

LEMMA 5.2. *Let H be a hypergraph and let $0 < \lambda < 1/16$ be a constant. Then there is fractional independent set μ, a $(\mu, \lambda/6)$-connected set W, and a partition $(K_1, \ldots, K_k)$ of W such that $k = \Omega(\lambda\sqrt{\mathrm{con}_\lambda(H)})$, and for every $1 \leq i \leq k$, K_i is a clique with $\mu(K_i) \geq 1/2$.*

If H is connected, then the maximum value of a uniform concurrent flow on $(X_1, \ldots, X_k)$ is at least $1/\binom{k}{2} = \Omega(k^{-2})$: if each of the $\binom{k}{2}$ flows has value $1/\binom{k}{2}$, then they are clearly compatible. The following lemma shows that if $X_1, \ldots, X_r$ are appropriate cliques of a (μ, λ)-connected set, then we can guarantee a better bound $\Omega(k^{-\frac{3}{2}})$.

LEMMA 5.3. *Let H be a hypergraph, μ a fractional independent set of H, and $W \subseteq V(H)$ a (μ, λ)-connected set of W for some $0 < \lambda < 1$. Let $(K_1, \ldots, K_k)$ be a partition of W such that K_i is a clique and $\mu(K_i) \geq 1/2$ for every $1 \leq i \leq k$. Then there is a uniform concurrent flow of value $\Omega(\lambda/k^{\frac{3}{2}})$ on $(K_1, \ldots, K_k)$.*

Intuitively, the intersection structure of the paths appearing in a uniform concurrent flow on cliques $K_1, \ldots, K_r$ is reminiscent of the edges of the complete graph on r vertices: if $\{i_1, j_1\} \cap \{i_2, j_2\} \neq \emptyset$, then every path of F_{i_1,j_1} touches every path of F_{i_2,j_2}. We use the following result from [24] (restated using the terminology of this paper), which shows that line graphs of cliques have good embedding properties. Let L_k be the line graph of the k-clique.

LEMMA 5.4. *For every $k > 1$ there is a constant $n_k > 0$ such that for every $G(V, E)$ with $|E| > n_k$ and no isolated vertices, the graph G has an embedding into L_k with vertex depth $O(|E|/k^2)$.*

The proof of Theorem 5.1 uses Lemmas 5.2 and 5.3 to find highly connected set and a uniform concurrent flow on it, and then uses Lemma 5.4 to embed the vertices of G into the paths appearing in the flows.

6. FROM EMBEDDINGS TO HARDNESS OF CSP

We prove the main hardness result in this section:

THEOREM 6.1. *Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs with unbounded submodular width. If there is an algorithm $\mathbb{A}$ and a function f such that $\mathbb{A}$ solves every instance I of CSP($\mathcal{H}$) with hypergraph $H \in \mathcal{H}$ in time $f(H) \cdot \|I\|^{o(\mathrm{subw}(H)^{\frac{1}{4}})}$, then the Exponential Time Hypothesis fails.*

In particular, Theorem 6.1 implies that CSP($\mathcal{H}$) for such a $\mathcal{H}$ is not fixed-parameter tractable:

COROLLARY 6.2. *If $\mathcal{H}$ is a recursively enumerable class of hypergraphs with unbounded submodular width, then CSP($\mathcal{H}$) is not fixed-parameter tractable, unless ETH fails.*

The Exponential Time Hypothesis (ETH) states that there is no $2^{o(n)}$ time algorithm for n-variable 3SAT. The Sparsification Lemma of Impagliazzo, Paturi, and Zane [22] shows that ETH is equivalent to the assumption that there is no algorithm for 3SAT whose running time is subexponential *in the number of clauses.* This result will be crucial for our hardness proof, as our reduction from 3SAT is sensitive to the number of clauses.

To prove Theorem 6.1, we show that a "too fast" algorithm $\mathbb{A}$ implies that a subexponential-time algorithm for 3SAT exists. We use the characterization of submodular width from Section 4 and the embedding results of Section 5 to reduce 3SAT to CSP($\mathcal{H}$) by embedding the incidence graph of a 3SAT formula into a hypergraph $H \in \mathcal{H}$. The basic idea of the proof is that if the 3SAT formula has m clauses and the edge depth of the embedding is m/r, then we can gain a factor r in the exponent of the running time. If submodular width is unbounded in $\mathcal{H}$, then we can make this gap r between the number of clauses and the edge depth arbitrary large, and hence the exponent can be arbitrarily smaller than the number of clauses, i.e., the algorithm is subexponential in the number of clauses.

The following simple lemma gives a transformation that turns a 3SAT instance into a binary CSP instance.

LEMMA 6.3 ([24]). *Given an instance of 3SAT with n variables and m clauses, it is possible to construct in polynomial time an equivalent CSP instance with $n + m$ variables, $3m$ binary constraints, and domain size 3.*

Next we show that an embedding from graph G to hypergraph H can be used to simulate a binary CSP instance I_1 having primal graph G by a CSP instance I_2 whose hypergraph is H. The size of the constraint relations of I_2 can grow very large: the edge depth of the embedding determines how large is this increase.

LEMMA 6.4. *Let $I_1 = (V_1, D_1, C_1)$ be a binary CSP instance with primal graph G and let ϕ be an embedding of G into a hypergraph H with edge depth q. Given I_1, H, and the embedding ϕ, it is possible to construct (in time polynomial in the size of the* output*) an equivalent CSP instance $I_2 = (V_2, D_2, C_2)$ with hypergraph H where the size of every constraint relation is at most $|D_1|^q$.*

With these tools at hand, we can prove Theorem 6.1 the following way. We show that if there is a class $\mathcal{H}$ of hypergraphs with unbounded submodular width and an algorithm $\mathbb{A}$ as in Theorem 6.1, then this algorithm can be used to solve 3SAT in subexponential time. The main ingredients are the embedding result of Theorem 5.1, and Lemmas 6.3 and 6.4 above on reduction to CSP. Furthermore, we need a way of choosing an appropriate hypergraph from the set $\mathcal{H}$. As discussed earlier, the larger the submodular width of the hypergraph is, the more we gain in the running time. However, we should not spend too much time on constructing the hypergraph and on finding an embedding. Therefore, we use the same technique as in [24]: we enumerate a certain number of hypergraphs and we try all of them simultaneously. The number of hypergraphs enumerated depends on the size of the 3SAT instance. This will be done in such a way that guarantees that we do not spend too much time on the enumeration, but eventually every hypergraph in $\mathcal{H}$ is considered for sufficiently large input sizes.

7. REFERENCES

[1] I. Adler. *Width functions for hypertree decompositions*. PhD thesis, Albert-Ludwigs-Universität Freiburg, 2006.

[2] I. Adler, G. Gottlob, and M. Grohe. Hypertree width and related hypergraph invariants. *European J. Combin.*, 28(8):2167–2181, 2007.

[3] N. Alon, I. Newman, A. Shen, G. Tardos, and N. Vereshchagin. Partitioning multi-dimensional sets in a small number of "uniform" parts. *European J. Combin.*, 28(1):134–144, 2007.

[4] C. Beeri, R. Fagin, D. Maier, A. O. Mendelzon, J. D. Ullman, and M. Yannakakis. Properties of acyclic database schemes. In *STOC*, pages 355–362, 1981.

[5] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. Assoc. Comput. Mach.*, 30(3):479–513, 1983.

[6] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[7] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoret. Comput. Sci.*, 239(2):211–229, 2000.

[8] H. Chen and M. Grohe. Constraint satisfaction problems with succinctly specified relations, 2006. Manuscript. Preliminary version in *Dagstuhl Seminar Proceedings 06401: Complexity of Constraints*.

[9] F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer. Some intersection theorems for ordered sets and graphs. *J. Combin. Theory Ser. A*, 43(1):23–37, 1986.

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

[11] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. Assoc. Comput. Mach.*, 30(3):514–550, 1983.

[12] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1999.

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

[14] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. of AAAI-90*, pages 4–9, Boston, MA, 1990.

[15] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64:579–627, 2002.

[16] G. Gottlob, F. Scarcello, and M. Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artificial Intelligence*, 138(1-2):55–86, 2002.

[17] G. Gottlob and S. Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *The Computer Journal*, 51(3):303–325, 2008.

[18] M. Grohe. The structure of tractable constraint satisfaction problems. In *MFCS'06*, pages 58–72, 2006.

[19] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1, 2007.

[20] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA'06*, pages 289–298, 2006.

[21] M. Grohe and D. Marx. On tree width, bramble size, and expansion. *J. Combin. Theory Ser. B*, 99(1):218–228, 2009.

[22] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

[23] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

[24] D. Marx. Can you beat treewidth? To appear in *Theory of Computing*.

[25] D. Marx. Approximating fractional hypetree width. In *SODA'09*, pages 902–911, 2009.

[26] D. Marx. Tractable structures for constraint satisfaction with truth tables. In *STACS'09*, pages 649–660, 2009.

[27] R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

[28] S. Oum. Approximating rank-width and clique-width quickly. In *WG'05*, pages 49–58, 2005.

[29] S. Oum and P. Seymour. Testing branch-width. *J. Combin. Theory Ser. B*, 97(3):385–393, 2007.

[30] S.-i. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

[31] F. Scarcello, G. Gottlob, and G. Greco. Uniform constraint satisfaction problems and database theory. In *Complexity of Constraints*, pages 156–195, 2008.

[32] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.