Parameterized Complexity of Maximum Happy Set and Densest k-Subgraph

Yosuke Mizutani

School of Computing, University of Utah, Salt Lake City, UT, USA

Blair D. Sullivan

School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

We present fixed-parameter tractable (FPT) algorithms for two problems, Maximum Happy Set (MaxHS) and Densest k-Subgraph (DkS) – also known as Maximum Edge Happy Set. Given a graph G and an integer k, MaxHS asks for a set S of k vertices such that the number of *happy vertices* with respect to S is maximized, where a vertex v is happy if v and all its neighbors are in S. We show that MaxHS can be solved in time $\mathcal{O}\left(2^{\mathsf{mw}} \cdot \mathsf{mw} \cdot k^2 \cdot |V(G)|\right)$ and $\mathcal{O}\left(8^{\mathsf{cw}} \cdot k^2 \cdot |V(G)|\right)$, where mw and cw denote the *modular-width* and the *clique-width* of G, respectively. This answers the open questions on fixed-parameter tractability posed in [1].

The DkS problem asks for a subgraph with k vertices maximizing the number of edges. If we define *happy edges* as the edges whose endpoints are in S, then DkS can be seen as an edge-variant of MaxHS. In this paper we show that DkS can be solved in time $f(\mathsf{nd}) \cdot |V(G)|^{\mathcal{O}(1)}$ and $\mathcal{O}\left(2^{\mathsf{cd}} \cdot k^2 \cdot |V(G)|\right)$, where nd and cd denote the *neighborhood diversity* and the *cluster deletion number* of G, respectively, and f is some computable function. This result implies that DkS is also fixed-parameter tractable by *twin cover number*.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases parameterized algorithms, maximum happy set, densest k-subgraph, modular-width, clique-width, neighborhood diversity, cluster deletion number, twin cover

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.23

Funding This work was supported in part by the Gordon & Betty Moore Foundation under award GBMF4560 to Blair D. Sullivan.

Acknowledgements We thank Arnab Banerjee, Oliver Flatt and Thanh Son Nguyen for their contributions to a course project that led to this research.

1 Introduction

In the study of large-scale networks, *communities* – cohesive subgraphs in a network – play an important role in understanding complex systems and appear in sociology, biology and computer science, etc. [16, 24]. For example, the concept of homophily in sociology explains the tendency for individuals to associate themselves with similar people [26]. Homophily is a fundamental law governing the structure of social networks, and finding groups of people sharing similar interests has many real-world applications [11].

People have attempted to frame this idea as a graph optimization problem, where a vertex represents a person and an edge corresponds to some relation in the social network. The notion of *happy* vertices was first introduced by Zhang and Li in terms of graph coloring [30], where each color represents an attribute of a person (possibly fixed). A vertex is *happy* if all of its neighbors share its color. The goal is to maximize the number of happy vertices by changing the color of unfixed vertices, thereby achieving the greatest social benefit.

17th International Symposium on Parameterized and Exact Computation (IPEC 2022).
Editors: Holger Dell and Jesper Nederlof; Article No. 23; pp. 23:1–23:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Later, Asahiro et al. introduced Maximum Happy Set (MaxHS) which defines that a vertex v is *happy* with respect to a *happy set* S if v and all of its neighbors are in S [1]. The MaxHS problem asks for a vertex set S of size k that maximizes the number of happy vertices. They also define its edge-variant, Maximum Edge Happy Set (MaxEHS) which maximizes the number of *happy edges*, an edge with both endpoints in the *happy set.* It is clear to see that MaxEHS is equivalent to choosing a vertex set S such that the number of edges in the induced subgraph on S is maximized. This problem is known as Densest k-Subgraph (DkS) in other literature. Both MaxHS and MaxEHS are NP-hard [1, 12], and we study their parameterized complexity throughout this paper.

1.1 Parameterized Complexity and Related Work

Graph problems are often studied with a variety of structural parameters in addition to natural parameters (size k of the happy set in our case). Specifically, we investigate the parameterized complexity with respect to *modular-width* (mw), *clique-width* (cw), *neighborhood diversity* (nd), *cluster deletion number* (cd), *twin cover number* (tc), *treewidth* (tw), and *vertex cover number* (vc), all of which we define in Section 2.3. Figure 1 illustrates the hierarchy of these parameters by inclusion; hardness results are implied along the arrows, and FPT[1] algorithms are implied in the reverse direction.

Asahiro et al. showed that MaxHS is W[1]-hard with respect to k by a parameterized reduction from the q-Clique problem [1]. They also presented FPT algorithms for MaxHS on parameters: clique-width plus k, neighborhood diversity, cluster deletion number (which implies FPT by twin cover number), and treewidth.

MaxEHS (DkS) has been extensively studied in different names (e.g. the k-Cluster problem [7], the Heaviest Unweighted Subgraph problem [22], and k-Cardinality Subgraph problem [5]). As for parameterized complexity, Cai showed the W[1]-hardness parameterized by k [6]. Bourgeois employed Moser's technique in [27] to show that MaxEHS can be solved in time $2^{\mathsf{tw}} \cdot n^{\mathcal{O}(1)}$ [3]. Broersma et al. proved that MaxEHS can be solved in time $k^{\mathcal{O}(\mathsf{cw})} \cdot n$, but it cannot be solved in time $2^{o(\mathsf{cw} \log k)} \cdot n^{\mathcal{O}(1)}$ unless the Exponential Time Hypothesis (ETH) fails [4]. To the best of our knowledge, the parameterized complexity by modular-width, neighborhood diversity, cluster deletion number and twin cover number remained open prior to our work. Figure 2 summarizes the known and established hardness results for MaxHS and MaxEHS.

1.2 Our Contributions

In this paper, we present four novel parameterized algorithms for MaxHS and MaxEHS. First, we shall provide a dynamic-programming algorithm that solves MaxHS in time $\mathcal{O}\left(2^{\mathsf{mw}} \cdot \mathsf{mw} \cdot k^2 \cdot |V|\right)$, answering the question posed by the authors of [1]. Second, we show that MaxHS is FPT by clique-width, giving an $\mathcal{O}\left(8^{\mathsf{cw}} \cdot k^2 \cdot |V|\right)$ algorithm, which removes the exponential term of k from the best known result, $\mathcal{O}\left(6^{\mathsf{cw}} \cdot k^{2(\mathsf{cw}+1)} \cdot |V|\right)$ [1]. While bounded modular-width implies bounded clique-width (Proposition 9), we give both algorithms because the one for modular-width has asymptotically faster running time.

Turning to MaxEHS, we prove it is FPT by neighborhood diversity, using an integer quadratic programming formulation. Lastly, we provide an FPT algorithm for MaxEHS parameterized by cluster deletion number with running time $\mathcal{O}\left(2^{\mathsf{cd}} \cdot k^2 \cdot |V|\right)$, which also

[1] An FPT (fixed-parameter tractable) algorithm solves the problem in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f.

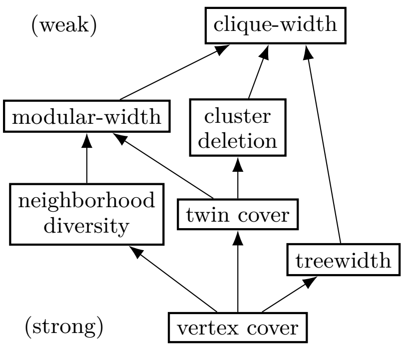

Figure 1 Hierarchy of relevant structural graph parameters. Arrows indicate generalizations.

Parameter	MaxHS	MaxEHS
Size k of happy set	W[1]-hard[1]	W[1]-hard[6]
Clique-width + k	FPT[1]	FPT[4]
Clique-width	FPT	W[1]-hard[4]
Modular-width	FPT	Open
Neighborhood diversity	FPT[1]	FPT
Cluster deletion number	FPT[1]	FPT
Twin cover number	FPT[1]	FPT
Treewidth	FPT[1]	FPT[3]
Vertex cover number	FPT[1]	FPT[3]

Figure 2 Known and established hardness results under select parameters for MaxHS and MaxEHS (as known as Densest k-Subgraph). New results from this paper in red.

implies the problem is FPT by twin cover number. These new results complete the previously-open parameterized complexities in Figure 2, except the one of DkS parameterized by modular-width.

Independent Work

Independently and simultaneously, Hanaka also showed the parameterized complexity of DkS by neighborhood diversity and cluster deletion number [21]. For neighborhood diversity, the complexity was implied by [25], as we discuss in sections 3.3 and 5.1. Further, the parameterized complexity by cluster deletion number was shown by an algorithm solving DkS in time $2^{\mathsf{bd}}\left((k^3 + \mathsf{bd})\,|V| + |E|\right)$, where bd denotes the block deletion number (note that it holds $\mathsf{cd} \leq \mathsf{bd}$).

2 Preliminaries

We use standard graph theory notation, following [10]. Given a graph $G = (V, E)$, we write $n(G) = |V|$ for the number of vertices and $m(G) = |E|$ for the number of edges. We use $N(v)$ and $N[v]$ to denote the open and closed neighborhoods of a vertex v, respectively, and for a vertex set $X \subseteq V$, $N[X]$ denotes the union of $N[x]$ for all $x \in X$. We write $\deg_G(v) = \deg(v)$ for the degree of a vertex v. We denote the induced subgraph of G on a set $X \subseteq V$ by $G[X]$. We say vertices u and v are *twins* if they have the same neighbors, i.e. $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. Further, they are called *true twins* if $uv \in E$.

2.1 Problem Definitions

Asahiro et al. first introduced the Maximum Happy Set problem in [1].

Maximum Happy Set (MaxHS)

Input: A graph $G = (V, E)$ and a positive integer k.

Problem: Find a subset $S \subseteq V$ of k vertices that maximizes the number of *happy* vertices v with $N[v] \subseteq S$.

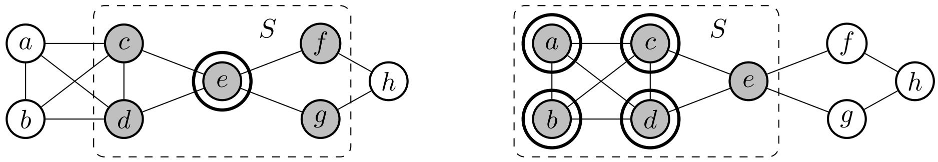

Figure 3 Given a graph above, if $k = 5$, choosing $S = \{c, d, e, f, g\}$ makes only one vertex (e) happy (left). On the other hand, $S = \{a, b, c, d, e\}$ is an optimal solution, making four vertices (a, b, c, d) happy (right).

Figure 3 illustrates an example instance with $k = 5$. Let us call a vertex that is not happy an *unhappy* vertex, and observe that the set of unhappy vertices is given by $N[V \setminus S]$, providing an alternative characterization of happy vertices.

▶ **Proposition 1.** *Given a graph $G = (V, E)$ and a happy set $S \subseteq V$, the set of happy vertices is given by $V \setminus N[V \setminus S]$.*

In addition, Asahiro et al. define an edge variant [1]:

Maximum Edge Happy Set (MaxEHS)

Input:	A graph $G = (V, E)$ and a positive integer k.
Problem:	Find a set $S \subseteq V$ of k vertices that maximizes the number of happy edges. An edge $uv \in E$ is *happy* if and only if $\{u, v\} \subseteq S$.

It is known that MaxEHS is identical to the Densest k-Subgraph problem (DkS), as the number of happy edges is equal to $m(G[S])$. Some literature (e.g. [14]) also phrases this problem as the dual of the Sparsest k-Subgraph problem.

2.2 Structural Parameters

We now define the structural graph parameters considered in this paper.

Treewidth. The most-studied structural parameter is treewidth, introduced by Robertson & Seymour in [28]. Treewidth measures how a graph resembles a tree and admits FPT algorithms for a number of NP-hard problems, such as Weighted Independent Set, Dominating Set, and Steiner Tree [9]. Treewidth is defined by the following notion of tree decomposition.

▶ **Definition 2** (treewidth [9])**.** *A tree decomposition of a graph G is a pair $(T, \{X_t\}_{t \in V(T)})$, where T is a tree and $X_t \subseteq V(G)$ is an assigned vertex set for every node t, such that the following three conditions hold:*

- $\bigcup_{t \in V(T)} X_t = V(G)$.
- *For every $uv \in E(G)$, there exists a node t such that $u, v \in X_t$.*
- *For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$ induces a connected subtree of T.*

*The width of tree decomposition is defined to be $\max_{t \in V(T)} |X_t| - 1$, and the **treewidth** of a graph G, denoted by* tw, *is the minimum possible width of a tree decomposition of G.*

Clique-width. Clique-width is a generalization of treewidth and can capture dense, but structured graphs. Intuitively, a graph with bounded clique-width k can be built from single vertices by joining structured parts, where vertices are associated by at most k labels such that those with the same label are indistinguishable in later steps.

▶ **Definition 3** (clique-width [8]). *For a positive integer w, a w-labeled graph is a graph whose vertices are labeled by integers in $\{1, \dots, w\}$. The **clique-width** of a graph G, denoted by* cw*, is the minimum w such that G can be constructed by repeated application of the following operations:*
- *(O1) Introduce $i(v)$: add a new vertex v with label $i \in \{1, \dots, w\}$.*
- *(O2) Union $G_1 \oplus G_2$: take a disjoint union of w-labeled graphs G_1 and G_2.*
- *(O3) Join $\eta(i, j)$: take two labels i and j, and then add an edge between every pair of vertices labeled by i and by j.*
- *(O4) Relabel $\rho(i, j)$: relabel the vertices of label i to label $j \in \{1, \dots, w\}$.*

This construction naturally defines a rooted binary tree, called a cw-expression tree G, where G is the root and each node corresponds to one of the above operations.

Neighborhood Diversity. Neighborhood diversity is a parameter introduced by Lampis [23], which measures the number of twin classes.

▶ **Definition 4** (neighborhood diversity [23]). *The **neighborhood diversity** of a graph $G = (V, E)$, denoted by* nd*, is the minimum number w such that V can be partitioned into w sets of twin vertices.*

By definition, each set of twins, called a *module*, is either a clique or an independent set.

Cluster Deletion Number. Cluster (vertex) deletion number is the distance to a cluster graph, which consists of disjoint cliques.

▶ **Definition 5** (cluster deletion number). *A vertex set X is called a cluster deletion set if $G[V \setminus X]$ is a cluster graph. The **cluster deletion number** of G, denoted by* cd*, is the size of the minimum cluster deletion set in G.*

Twin Cover Number. The notion of twin cover is introduced by Ganian [19] and offers a generalization of vertex cover number.

▶ **Definition 6** (twin cover number [19]). *A vertex set $X \subseteq V$ is a twin cover of $G = (V, E)$ if for every edge $uv \in E$ either (1) $u \in X$ or $v \in X$, or (2) u and v are true twins. The **twin cover number**, denoted by* tc*, is the size of the minimum twin cover of G.*

Modular-width. Modular-width is a parameter introduced by Gajarský et al. [17] to generalize simpler notions on dense graphs while avoiding the negative results brought by moving to the full generality of clique-width (e.g. many problems FPT for treewidth becomes W[1]-hard for clique-width [13, 14, 15]). Modular-width is defined using the standard concept of modular decomposition.

▶ **Definition 7** (modular-width [17]). *Any graph can be produced via a sequence of the following operations:*
- *(O1) Introduce: Create an isolated vertex.*
- *(O2) Union $G_1 \oplus G_2$: Create the disjoint union of two graphs G_1 and G_2.*
- *(O3) Join: Given two graphs G_1 and G_2, create the complete join G_3 of G_1 and G_2. That is, a graph G_3 with vertices $V(G_1) \cup V(G_2)$ and edges $E(G_1) \cup E(G_2) \cup \{(v, w) : v \in G_1, w \in G_2\}$.*
- *(O4) Substitute: Given a graph G with vertices $v_1, \dots, v_n$ and given graphs $G_1, \dots, G_n$, create the substitution of $G_1, \dots, G_n$ in G. The substitution is a graph $\mathcal{G}$ with vertex set $\bigcup_{1 \le i \le n} V(G_i)$ and edge set $\bigcup_{1 \le i \le n} E(G_i) \cup \{(v, w) : v \in G_i, w \in G_j, (v_i, v_j) \in E(G)\}$. Each graph G_i is substituted for a vertex v_i, and all edges between graphs corresponding to adjacent vertices in G are added.*

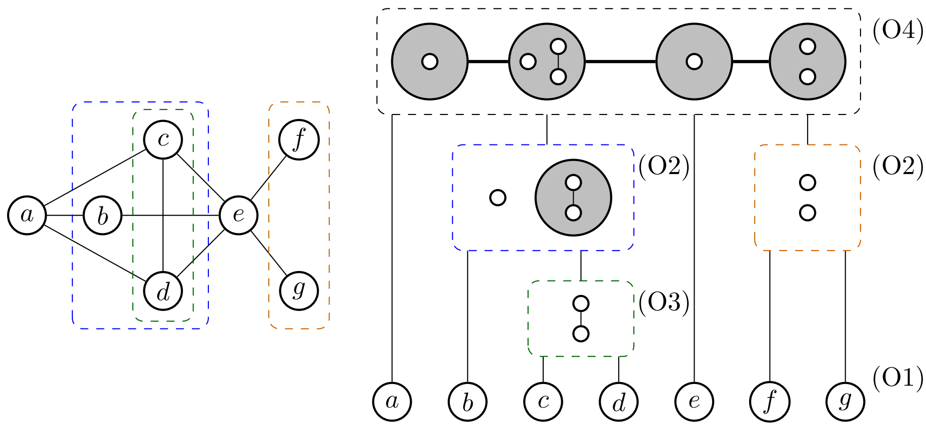

Figure 4 An example graph G (left) with modular-width 4. Modular decomposition of the same graph is shown at right. The parse-tree has G as the root, and its nodes correspond to operations (O1)-(O4). Notice that each node also represents a *module* – module members have the same neighbors outside the module.

*These operations, taken together in order to construct a graph, form a parse-tree of the graph. The width of a graph is the maximum size of the vertex set of G used in operation (O4) to construct the graph. The **modular-width**, denoted by* mw*, is the minimum width such that G can be obtained from some sequence of operations (O1)-(O4).*

Finding a parse-tree of a given graph, called a *modular decomposition*, can be done in linear-time [29]. See Figure 4 for an illustration of modular decomposition. Gajarský et al. also give FPT algorithms parameterized by modular-width for Partition into paths, Hamiltonian path, Hamiltonian cycle and Coloring, using bottom-up dynamic programming along the parse-tree.

Vertex Cover Number. The vertex cover number is the solution size of the classic Vertex Cover problem.

▶ **Definition 8.** *A vertex set $X \subseteq V$ is a vertex cover of $G = (V, E)$ if for every edge $uv \in E$ either $u \in X$ or $v \in X$. The **vertex cover number** of G, denoted by* vc*, is the size of the minimum vertex cover of G.*

2.2.1 Properties of Structural Parameters

Finally, we note the relationship among the parameters defined above, which establishes the hierarchy shown in Figure 1.

▶ **Proposition 9** ([1, 17]). *Let* $\mathsf{cw}, \mathsf{tw}, \mathsf{cd}, \mathsf{nd}, \mathsf{tc}, \mathsf{vc}, \mathsf{mw}$ *be the clique-width, tree-width, cluster deletion number, neighborhood diversity, twin cover number, vertex cover number, and modular-width of a graph G, respectively. Then the following inequalities hold[2]: (i)* $\mathsf{cw} \le 2^{\mathsf{tw}+1} + 1$*; (ii)* $\mathsf{tw} \le \mathsf{vc}$*; (iii)* $\mathsf{nd} \le 2^{\mathsf{vc}} + \mathsf{vc}$*; (iv)* $\mathsf{cw} \le 2^{\mathsf{cd}+3} - 1$*; (v)* $\mathsf{cd} \le \mathsf{tc} \le \mathsf{vc}$*; (vi)* $\mathsf{mw} \le \mathsf{nd}$*; (vii)* $\mathsf{mw} \le 2^{\mathsf{tc}} + \mathsf{tc}$*; and (viii)* $\mathsf{cw} \le \mathsf{mw} + 2$*.*

2 $\mathsf{cw} \le 2$ when $\mathsf{mw} = 0$; otherwise, $\mathsf{cw} \le \mathsf{mw} + 1$.

3 Background

Before describing our algorithms, we introduce some building blocks for our argument.

3.1 Entire Subgraphs

Structural parameters such as modular-width and clique-width entail the join operation in their underlying construction trees. When joining two subgraphs in MaxHS, it is important to distinguish whether all the vertices in the subgraph are included in the happy set. Formally, we introduce the notion of *entire subgraphs*.

▶ **Definition 10.** *Given a graph G and a happy set S, an* ***entire subgraph*** *with respect to S is a subgraph G' of G such that $V(G') \subseteq S$.*

By definition, the empty subgraph is always entire. The following lemma is directly derived from the definition of happy vertices.

▶ **Lemma 11.** *Let G be a complete join of subgraphs G_1 and G_2. For any happy set $S \subseteq V(G)$, $V(G_1)$ contains a happy vertex only if G_2 is entire with respect to S.*

Proof. If G_2 is not entire, there must exist $v \in V(G_2)$ such that $v \notin S$. Recall Proposition 1, and we have $N[V(G) \setminus S] \supseteq N(v) \supseteq V(G_1)$, which implies that any vertex in $V(G_1)$ cannot be happy. ◀

3.2 Knapsack Variant with Non-linear Values

The classic Knapsack problem has a number of variants, including 0-1 Knapsack [20] and Quadratic Knapsack [18]. In this paper we consider another variant, where the objective function is the sum of non-linear functions, but the function range is limited to integers. Specifically, each item has unit weight, but its value may vary depending on the number of copies of each type of item. We also require the weight sum to be exact and call this problem f-Knapsack, where f stands for function.

f-Knapsack

Input: Given a set of n items numbered from 1 to n, a weight capacity $W \in \mathbb{Z}_0^+$ and a value function $f_i : D_i \to \mathbb{Z}_0^+$, defined on a non-negative integral domain D_i for each item i.

Problem: For every $1 \leq i \leq n$, find the number $x_i \in D_i$ of instances of item i to include in the knapsack, maximizing $\sum_{i=1}^n f_i(x_i)$, subject to $\sum_{i=1}^n x_i = W$.

We show that this problem is solvable in polynomial-time.

▶ **Lemma 12.** *f-Knapsack can be solved in time $\mathcal{O}\left(nW^2\right)$.*

Proof. First, define the value $\phi[t, w]$ to be the maximum possible sum $\sum_{i=1}^t f_i(x_i)$, subject to $\sum_{i=1}^t x_i = w$ and $x_i \in D_i$ for every i. Then, perform bottom-up dynamic programming as follows.

- Initialize: $\phi[0, w] = \begin{cases} 0 & \text{if } w = 0, \\ -\infty & \text{if } w > 0 \text{ (meaning Infeasible)}. \end{cases}$
- Update: $\phi[t, w] = \max_{x_t \in D_t \wedge x_t \leq w} f_t(x_t) + \phi[t-1, w-x_t]$
- Result: $\phi[n, w]$ for $0 \leq w \leq W$ is the optimal value for weight w.

The base case ($\phi[0, w]$) represents the state where no item is in the knapsack, so both the objective and weight are 0; otherwise, infeasible. For the inductive step, any optimal solution $\phi[t, w]$ can be decomposed into $f_t(x_t) + \sum_{i=1}^{t-1} f_i(x_i)$ for some x_t, and the latter term ($\sum_{i=1}^{t-1} f_i(x_i)$) must equal $\phi[t-1, w-x_t]$ by definition. We consider all possible integers x_t, and thus the algorithm is correct.

Since $0 \le t \le n$, $0 \le w \le W$, and the update takes time $\mathcal{O}(W)$, the total running time is $\mathcal{O}(nW^2)$. By using the standard technique of backlinks, one can reconstruct the solution $\{x_i\}$ within the same asymptotic running time. ◀

The following result is a natural by-product of the algorithm above.

▶ **Corollary 13.** *Given an integer W, f-Knapsack for all weight capacities $0 \le w \le W$ can be solved in total time $\mathcal{O}(nW^2)$.*

3.3 Integer Quadratic Programming

For MaxEHS, we use the following known result that Integer Quadratic Programming is FPT by the number of variables and coefficients.

Integer Quadratic Programming (IQP)

Input: An $n \times n$ integer matrix Q, an $m \times n$ integer matrix A and an m-dimensional integer vector b.

Problem: Find a vector $x \in \mathbb{Z}^n$ minimizing $x^T Q x$, subject to $Ax \le b$.

▶ **Proposition 14** (Lokshtanov [25]). *There exists an algorithm that given an instance of* IQP*, runs in time $f(n, \alpha)L^{\mathcal{O}(1)}$, and outputs a vector $x \in \mathbb{Z}^n$. If the input IQP has a feasible solution then x is feasible, and if the input IQP is not unbounded, then x is an optimal solution. Here α denotes the largest absolute value of an entry of Q and A, and L is the total number of bits required to encode the input.*

It is convenient to have a linear term in the objective function. This can be achieved by introducing a new variable $\hat{x} = 1$ and adding $[0, q]$ as the corresponding row in Q [25].

▶ **Corollary 15.** *Proposition 14 holds if we generalize the objective function from $x^T Q x$ to $x^T Q x + q^T x$ for some n-dimensional integer vector q. Here α is the largest absolute value of an entry of Q, q and A.*

4 Algorithms for Maximum Happy Set

Now we describe our FPT algorithms for MaxHS with respect to modular-width and clique-width. At a high level, we employ a bottom-up dynamic programming (DP) approach on the parse-tree of a given graph, considering each node once. At each node, we use several techniques on precomputed results to update the DP table. For simplicity, our DP tables store the maximum number of happy vertices. Like other DP applications, a *certificate*, i.e. the actual happy set, can be found by using backlinks within the same asymptotic running time.

4.1 Parameterized by modular-width

We give an algorithm whose running time is singly-exponential in the modular-width, quadratic in k and linear in the graph size.

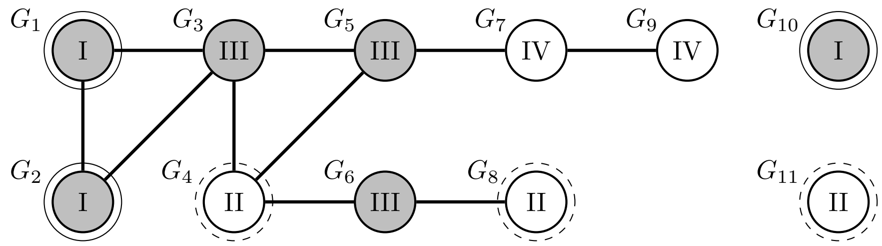

Figure 5 Four types of the subgraphs after applying operation (O4). Entire subgraphs (Type I and III) are shaded in gray. A subgraph becomes Type III or IV if it has a non-entire neighbor (e.g. G_3, G_9). In a Type I subgraph, all vertices are happy. Type II subgraphs may or may not admit a happy vertex. Type III and IV subgraphs cannot contain a happy vertex, as it is adjacent to a non-entire subgraph.

▶ **Theorem 16.** MaxHS *can be solved in time* $\mathcal{O}(2^{\mathsf{mw}} \cdot \mathsf{mw} \cdot k^2 \cdot |V(G)|)$*, where* mw *is the modular-width of the input graph* G*.*

Our algorithm follows the common framework seen in [17]. Given a graph G, a parse-tree with modular-width mw can be computed in linear-time [29]. The number of nodes in the parse-tree is linear in $|V(G)|$ [17]. Our algorithm traverses the parse-tree from the bottom, considering only operation (O4), as operations (O2)-(O3) can be replaced with a single operation (O4) with at most two arguments [17]. Further, we assume $2 \le \mathsf{mw} < k$ without loss of generality.

Each node in the parse-tree corresponds to an induced subgraph of G, which we write $\mathcal{G}$. We keep track of a table $\phi[\mathcal{G}, w]$, the maximum number of happy vertices for $\mathcal{G}$ with regard to a happy set of size w. We may assume $0 \le w \le k$ because we do not have to consider a happy set larger than size k. The entries of the DP table are initialized with $\phi[\mathcal{G}, w] = -\infty$. For the base case, a graph G_0 with a single vertex introduced by operation (O1), we set $\phi[G_0, 0] = 0$ and $\phi[G_0, 1] = 1$. The solution to the original problem is given by $\phi[G, k]$.

Our remaining task is to compute, given a graph substitution $\mathcal{G} = H(G_1, \ldots, G_n)$ ($n \le \mathsf{mw}$), the values of $\phi[\mathcal{G}, w]$ provided partial solutions $\phi[G_1, \cdot], \ldots, \phi[G_n, \cdot]$. We first choose a set of entire subgraphs from $G_1, \ldots, G_n$. Then, we identify the *subgraph type* for each G_i during a graph substitution.

▶ **Definition 17** (subgraph type). *Given a graph substitution* $H(G_1, \ldots, G_n)$*, where* $v_i \in V(H)$ *is substituted by* G_i*, and a happy set* S*, we categorize each substituted subgraph* G_i *into the following four types.*

- *Type I:* G_i *is entire and for every* j *such that* $v_j \in N_H(v_i)$*,* G_j *is entire.*
- *Type II:* G_i *is not entire and for every* j *such that* $v_j \in N_H(v_i)$*,* G_j *is entire.*
- *Type III:* G_i *is entire and not Type I.*
- *Type IV:* G_i *is not entire and not Type II.*

Intuitively, Type I and II subgraphs are surrounded by entire subgraphs in H, the *metagraph* to substitute, and Type I and III subgraphs are entire. A pictorial representation of this partition is presented in Figure 5. Observe that from Lemma 11, the subgraphs with Type III and IV cannot include any happy vertices. Further, Type II subgraphs are independent in H because their neighbors must be of Type III. This ensures that the choice of a happy set in Type II is independent of other subgraphs.

Lastly, we formulate an f-Knapsack instance as described in the following algorithm for updating the DP table on a single operation (O4).

▶ **Algorithm 1** (MaxHS-MW). *Given a graph substitution $\mathcal{G} = H(G_1, \ldots, G_n)$ and partial solutions $\phi[G_1, \cdot], \ldots, \phi[G_n, \cdot]$, consider all combinations of entire subgraphs from $G_1, \ldots, G_n$ and proceed the following steps.*

(Step 1) Identify subgraph types for $G_1, \ldots, G_n$. (Step 2) Formulate an f-Knapsack instance with capacity k and value functions f_i, based on the subgraph G_i's type as follows.

- *Type I* : $f_i(x) = |G_i|$, $\quad x = |G_i|$.
- *Type II* : $f_i(x) = \phi[G_i, x]$, $\quad 0 \le x < |G_i|$.
- *Type III* : $f_i(x) = 0$, $\quad x = |G_i|$.
- *Type IV* : $f_i(x) = 0$, $\quad 0 \le x < |G_i|$.

Then, update the DP table entries $\phi[\mathcal{G}, w]$ for $0 \le w \le k$ with the solution to f-Knapsack, if its value is greater than the current value.

We now prove that the runtime of this algorithm is FPT with respect to modular-width.

▶ **Lemma 18.** *Algorithm 1 correctly computes $\phi[\mathcal{G}, w]$ for every $0 \le w \le k$ in time $\mathcal{O}\left(2^{\mathsf{mw}} \cdot \mathsf{mw} \cdot k^2\right)$.*

Proof. First, the algorithm considers all possible sets of entire substituted subgraphs $(G_1, \ldots, G_n)$. The optimal solution must belong to one of them. It remains to prove the correctness of the f-Knapsack formulation in step 2. From Lemma 11, the subgraphs of Type III and IV cannot increase the number of happy vertices, so we set $f_i(x) = 0$. For Type I, the algorithm has no option but to include all of $V(G_i)$ in the happy set, and they are all happy.

The subgraphs of Type II are the only ones that use previous results, $\phi[G_i, \cdot]$. Since the new neighbors to G_i are required to be in the happy set, for any choice of the happy set in G_i, happy vertices remain happy, and unhappy vertices remain unhappy. Thus, we can directly use $\phi[G_i, \cdot]$; its choice does not affect other substituted subgraphs, as Type II subgraphs are independent in H. The domain of functions f_i is naturally determined by the definition of entire subgraphs.

Now, consider the running time of Algorithm 1. It considers 2^n possible combinations of entire subgraphs. Step 1 can be done by checking neighbors for each vertex in H, so the running time is $\mathcal{O}(|E(H)|) = \mathcal{O}\left(n^2\right)$. And step 2 takes time $\mathcal{O}\left(nk^2\right)$ from Corollary 13. The total running time is $\mathcal{O}\left(2^n(n^2 + nk^2)\right) = \mathcal{O}\left(2^{\mathsf{mw}} \cdot \mathsf{mw} \cdot k^2\right)$ as we assume $n \le \mathsf{mw} < k$. ◀

Proof of Theorem 16. It is trivial to see that the base case of the DP is valid, and the correctness of inductive steps is given by Lemma 18. We process each node of the parse-tree once, and it has $\mathcal{O}(|V(G)|)$ nodes [17]. Thus, the overall runtime is $\mathcal{O}(2^{\mathsf{mw}} \cdot \mathsf{mw} \cdot k^2 \cdot |V(G)|)$. ◀

4.2 Parameterized by clique-width

We provide an algorithm for MaxHS parameterized only by clique-width (cw), which no longer requires a combined parameter with solution size k as presented in [1].

▶ **Theorem 19.** *Given a* cw*-expression tree of a graph G with clique-width* cw*,* MaxHS *can be solved in time $\mathcal{O}(8^{\mathsf{cw}} \cdot k^2 \cdot |V(G)|)$.*

Here we assume that we are given a cw-expression tree, where each node t represents a *labeled* graph G_t. A labeled graph is a graph whose vertices are labeled by integers in $L = \{1, \ldots, \mathsf{cw}\}$. Every node must be one of the following: introduce node $i(v)$, union node $G_1 \oplus G_2$, relabel node $\rho(i, j)$, or join node $\eta(i, j)$. We write V_i for the set of vertices with label i.

Our algorithm traverses the cw-expression tree from the leaves and performs dynamic programming. For every node t, we keep track of the annotated partial solution $\phi[t, w, X, T]$, for every integer $0 \leq w \leq k$ and sets of labels $X, T \subseteq L$. We call X the *entire labels* and T the *target labels*. $\phi[t, w, X, T]$ is defined to be the maximum number of happy vertices having target labels T for G_t with respect to a happy set $S \subseteq V(G_t)$ of size w such that V_ℓ is entire to S if and only if $\ell \in X$. The entries of the DP table are initialized with $\phi[t, w, X, T] = -\infty$. The solution to the original graph G is computed by $\max_{X \subseteq L} \phi[r, k, X, L]$, where r is the root of the cw-expression tree. Now we claim the following recursive formula for each node type.

▶ **Lemma 20** (Formula for introduce nodes)**.** *Suppose t is an introduce node, where a vertex v with label i is introduced. Then, the following holds.*

$$\phi[t, w, X, T] = \begin{cases} 1 & \text{if } w = 1,\ X = L \text{ and } i \in T; \\ 0 & \text{if } w = 1,\ X = L \text{ and } i \notin T; \\ 0 & \text{if } w = 0 \text{ and } X = L \setminus \{i\}; \\ -\infty & \text{otherwise.} \end{cases}$$

Proof. First, notice that all labels but i are empty and thus entire. If we include v in the happy set, then we get $w = 1$ and $X = L$ (all labels are entire). The resulting value depends on the target labels. If i is a target label, i.e. $i \in T$, then v is a happy vertex having a target label, resulting in $\phi[t, w, X, T] = 1$. Otherwise, $\phi[t, w, X, T] = 0$. If $w = 0$, then label i cannot be entire, and the only feasible solution is $\phi[t, 0, L \setminus \{i\}, T] = 0$. ◀

▶ **Lemma 21** (Formula for union nodes)**.** *Suppose t is a union node with child nodes t_1 and t_2. Then, the following holds.*

$$\phi[t, w, X, T] = \max_{0 \leq \tilde{w} \leq w} \max_{\substack{X_1, X_2 \subseteq L \\ : X_1 \cap X_2 = X}} \phi[t_1, \tilde{w}, X_1, T] + \phi[t_2, w - \tilde{w}, X_2, T]$$

Proof. At a union node, since G_{t_1} and G_{t_2} are disjoint, any maximum happy set in G_t must be the disjoint union of some maximum happy set in G_{t_1} and that in G_{t_2} for the same target labels. We consider all possible combinations of partial solutions to G_{t_1} and G_{t_2}, so the optimality is preserved. Note that a label in G_t is entire if and only if it is entire in both G_{t_1} and G_{t_2}. ◀

▶ **Lemma 22** (Formula for relabel nodes)**.** *Suppose t is a relabel node with child node t', where label i in graph $G_{t'}$ is relabeled to j. Then, the following holds.*

$$\phi[t, w, X, T] = \begin{cases} -\infty & \text{if } i \notin X; \\ \phi[t', w, X, T'] & \text{if } i \in X \text{ and } j \in X; \\ \max_{Y \in \{\emptyset, \{i\}, \{j\}\}} \phi[t', w, X \setminus \{i\} \cup Y, T'] & \text{if } i \in X \text{ and } j \notin X, \end{cases}$$

where $T' = T \cup \{i\}$ if $j \in T$ and $T \setminus \{i\}$ otherwise.

Proof. At a relabel node $\rho(i, j)$, label i becomes empty, so it must be entire in G_t, leading to the first case. The variable T' converts the target labels in $G_{t'}$ to those in G_t. If label j is a target in G_t, then i and j must be targets in $G_{t'}$. Likewise, if label j is not a target in G_t, then neither i nor j should be targets in $G_{t'}$.

If label j is entire in G_t, then the maximum happy set must be the same as the one in $G_{t'}$ where both labels i and j are entire. If j is not entire in G_t, then we need to choose the best solution from the following: i is entire but j is not, j is entire but i is not, neither i nor j is entire. Because G_t and $G_{t'}$ have the same underlying graph, the optimal solution must be one of these. ◀

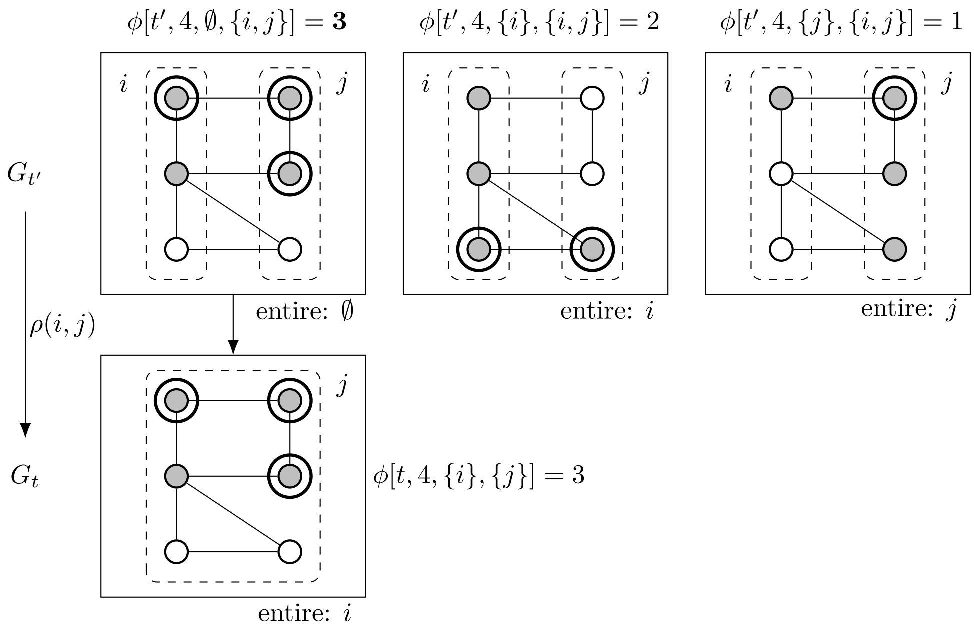

Figure 6 Visualization of the relabel operation $\rho(i,j)$ in the cw-expression tree of labels i, j. Figures show happy sets of size 4 (shaded in gray) maximizing the number of happy vertices (shown with double circles). After relabeling i to j, label i becomes empty and thus entire. Since label j in G_t corresponds to labels i and j in $G_{t'}$, to compute $\phi[t, 4, \{i\}, \{j\}]$, we need to look up three partial solutions $\phi[t', 4, \emptyset, T']$, $\phi[t', 4, \{i\}, T']$, and $\phi[t', 4, \{j\}, T']$, where $T' = \{i, j\}$, and keep the one with the largest value (the left one in this example).

Figure 6 illustrates the third case of the formula in Lemma 22.

▶ **Lemma 23** (Formula for join nodes). *Suppose t is a join node with the child node t', where labels i and j are joined. Then, the following holds.*

$$\phi[t, w, X, T] = \begin{cases} \phi[t', w, X, T] & \text{if } i \in X \text{ and } j \in X \\ \phi[t', w, X, T \setminus \{i\}] & \text{if } i \in X \text{ and } j \notin X \\ \phi[t', w, X, T \setminus \{j\}] & \text{if } i \notin X \text{ and } j \in X \\ \phi[t', w, X, T \setminus \{i,j\}] & \text{if } i \notin X \text{ and } j \notin X \end{cases}$$

Proof. At a join node $\eta(i,j)$, first observe that for any happy set, the vertices labeled other than i, j are unaffected; happy vertices remain happy. Further, if label j is not entire in G_t, then all vertices in V_i cannot be happy from Lemma 11. Thus, the maximum happy set in G_t is equivalent to the one in $G_{t'}$ such that label i is not a target label. The same argument applies to the other cases. ◀

Figure 7 illustrates the second case of the formula in Lemma 23. Lastly, we examine the running time of these computations.

▶ **Proposition 24.** *Given a cw-expression tree and its node t, and partial solutions $\phi[t', \cdot, \cdot, \cdot]$ for every child node t' of t, we can compute $\phi[t, w, X, T]$ for every w, X, T in time $\mathcal{O}\left(8^{\mathsf{cw}} \cdot k^2\right)$.*

Proof. It is clear to see that for fixed t, w, X, T, the formulae for introduce, relabel, and join nodes take $\mathcal{O}(1)$. If we compute $\phi[t, \cdot, \cdot, \cdot]$ for every w, X, T, the total running time is $\mathcal{O}\left((2^{\mathsf{cw}})^2 \cdot k\right)$ since w is bounded by k and there are 2^{cw} configurations for X and T.

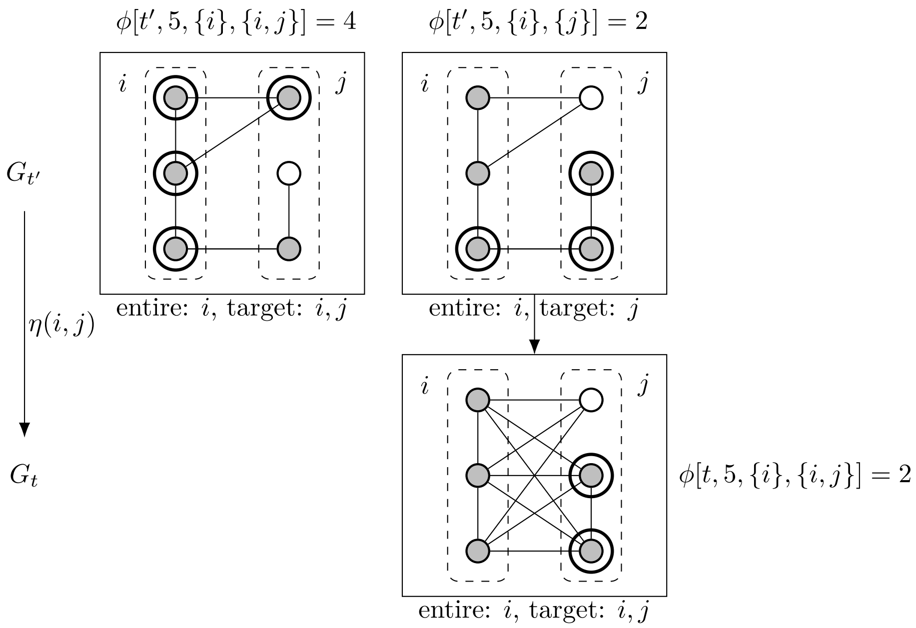

Figure 7 Visualization of the join operation $\eta(i, j)$ in the cw-expression tree of labels i, j. Figures show happy sets of size 5 (shaded in gray) maximizing the number of happy vertices (shown with double circles) for different target labels. We consider the case where label i is entire and j is not. The graph $G_{t'}$ admits 4 happy vertices if both i and j are target labels. However, this is no longer true after the join because label j is not entire. Instead, the optimal happy set for G_t can be found where label i is excluded from the target labels for $G_{t'}$, i.e. $\phi[t', 5, \{i\}, \{j\}]$, which admits 2 happy vertices with label j. Notice that we do not count the happy vertices with label i if it is not a target.

For the union node formula, observe that X can be determined by the choice of X_1 and X_2, so it is enough to consider all possible values for $w, T, \tilde{w}, X_1, X_2$, which results in the running time $\mathcal{O}\left((2^{\mathsf{cw}})^3 \cdot k^2\right)$, or $\mathcal{O}\left(8^{\mathsf{cw}} \cdot k^2\right)$. ◀

This completes the proof of Theorem 19, as we process each node of the cw-expression tree once, and it has $\mathcal{O}(|V(G)|)$ nodes.

5 Algorithms for Maximum Edge Happy Set

In addition to MaxHS, we also study its edge-variant MaxEHS. One difference from MaxHS is that when joining two subgraphs, we may increase the number of edges between those subgraphs, even if they are not entire. In other words, the number of edges between joining subgraphs depends on two variables, and quadratic programming naturally takes part in this setting. Here, we present FPT algorithms for two parameters – neighborhood diversity and cluster deletion number – to figure out the boundary between parameters that are FPT (e.g. treewidth) and W[1]-hard (e.g. clique-width) (see Figure 2).

5.1 Parameterized by neighborhood diversity

As shown in Figure 1, neighborhood diversity is a parameter specializing modular-width. To obtain a finer classification of structural parameters, we now show MaxEHS is FPT parameterized by neighborhood diversity.

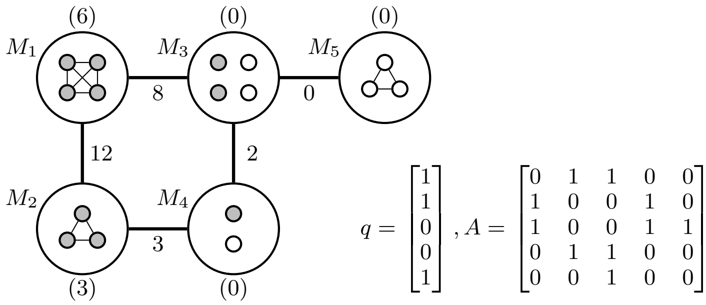

Figure 8 Example instance of MaxEHS with $\mathsf{nd} = 5$. The figure shows the quotient graph H of the given graph G on its modules $M_1, \ldots, M_5$. Every edge in H forms a biclique in G. The maximum edge happy set for $k = 10$ are shaded in gray. It also shows the number of internal edges for each module (e.g. (6) for M_1), and that of external edges between modules (e.g. 12 between M_1 and M_2). Vector q indicates if each module is a clique or an independent set (e.g. $q_1 = 1$ because M_1 is a clique), and A is the adjacency matrix of the quotient graph.

Let nd be the neighborhood diversity of the given graph G. We observe that any instance (G, k) of MaxEHS can be reduced to the instance of Integer Quadratic Programming (IQP) as follows.

▶ **Lemma 25.** MaxEHS *can be reduced to* IQP *with* $\mathcal{O}(\mathsf{nd})$ *variables and bounded coefficients in time* $\mathcal{O}(|V(G)| + |E(G)|)$.

We defer our proof to Appendix. Figure 8 exemplifies a quotient graph of a graph with $\mathsf{nd} = 5$, along with vector q and matrix A. The following is a direct result from Lemma 25 and Proposition 14.

▶ **Theorem 26.** MaxEHS *can be solved in time* $f(\mathsf{nd}) \cdot |V(G)|^{\mathcal{O}(1)}$, *where* nd *is the neighborhood diversity of the input graph* G *and* f *is a computable function.*

5.2 Parameterized by cluster deletion number

Finally, we present an FPT algorithm for MaxEHS parameterized by the cluster deletion number of the given graph.

▶ **Theorem 27.** *Given a graph* $G = (V, E)$ *and its cluster deletion set* X *of size* cd, MaxEHS *can be solved in time* $\mathcal{O}\left(2^{\mathsf{cd}} \cdot k^2 \cdot |V|\right)$.

Recall that by definition, $G[V \setminus X]$ is a set of disjoint cliques. Let $C_1, \ldots, C_p$ be the clusters appeared in $G[V \setminus X]$. Our algorithm first guesses part of the happy set S', defined as $S \cap X$, and performs f-Knapsack with p items.

▶ **Algorithm 2** (MaxEHS-CD). *Given a graph* $G = (V, E)$ *and its cluster deletion set* X, *consider all sets of* $S' \subseteq X$ *such that* $|S'| \leq k$ *and proceed the following steps.*

(Step 1) For each clique C_i, *sort its vertices in non-increasing order of the number of neighbors in* S'. *Let* $v_{i,1}, \ldots, v_{i,|C_i|}$ *be the ordered vertices in* C_i. *(Step 2) For each* $1 \leq i \leq p$, *construct a function* f_i *as follows:* $f_i(0) = 0$ *and for every* $1 \leq j \leq |C_i|$, $f_i(j) = f_i(j-1) + |N(v_{i,j}) \cap S'| + j - 1$. *(Step 3) Formulate an* f-Knapsack *instance with capacity* $k - |S'|$ *and value functions* f_i *for every* $1 \leq i \leq p$. *Then, obtain the solution* $\{x_i\}$ *with the exact capacity* $k - |S'|$ *if feasible. (Step 4) Construct* S *as follows: Initialize with* S' *and for each clique* C_i, *pick* x_i *vertices in order and include them in* S. *That is, update* $S \leftarrow S \cup \{v_{i,1}, \ldots, v_{i,x_i}\}$ *for every* $1 \leq i \leq p$. *Finally, return* S *that maximizes* $|E(G[S])|$.

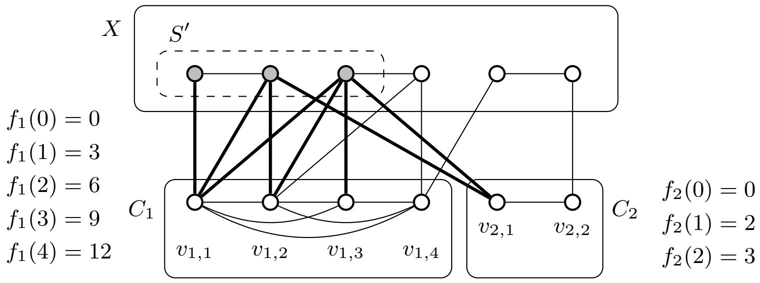

Figure 9 Visualization of MaxEHS-CD, given a graph $G = (V, E)$ with its cluster deletion set X and a fixed partial solution S' ($|S'| = 3$, shaded in gray). The graph after removing X, i.e. $G[V \setminus X]$, forms cliques C_1 and C_2. For each clique, vertices are sorted in decreasing order of the number of neighbors in S' (edges to S' in thicker lines). Functions f_1 and f_2 are constructed as described in the algorithm and used for f-Knapsack. For example, $f_1(3) = f_1(2) + 1 + (3 - 1)$ as vertex $v_{1,3}$ has one edge to S' and two edges to previously-added $v_{1,1}$ and $v_{1,2}$. If $k = 6$, then we pick $k - |S'| = 3$ vertices from C_1 and C_2. The optimal solution would be $\{v_{1,1}, v_{1,2}, v_{1,3}\}$ because $f_1(3) + f_2(0) = 9$ gives the maximum objective value in the f-Knapsack formulation.

Intuitively, we construct function f_i in a greedy manner. When we add a vertex v in clique C_i to the happy set S, it will increase the number of happy edges by the number of v's neighbors in S' and the number of the vertices in C_i that are already included in S. Therefore, it is always advantageous to pick a vertex having the most neighbors in S'. Figure 9 illustrates the key ideas of Algorithm 2. The following proposition completes the proof of Theorem 27.

▶ **Proposition 28.** *Given a graph $G = (V, E)$ and its cluster deletion set X of size* cd*, Algorithm 2 correctly finds the maximum edge happy set in time $\mathcal{O}\left(2^{\mathsf{cd}} \cdot k^2 \cdot |V|\right)$.*

Proof. The algorithm considers all possible sets of $S \cap X$, so the optimal solution should extend one of them. It is clear to see that when the f-Knapsack instance is feasible, S ends up with k vertices, since the sum of the obtained solution must be $k - |S'|$. The objective of the f-Knapsack is equivalent to $|E(G[S])| - |E(G[S'])|$, that is, the number of happy edges extended by the vertices in $V \setminus X$. Since S' has been fixed at this point, the optimal solution to f-Knapsack leads to that to MaxEHS. Lastly, the value function f_i is correct because for each clique C_i, the number of extended edges is given by $\binom{|S_i|}{2} + \sum_{v \in S_i} |N(v) \cap S'|$, where $S_i = S \cap C_i$ and $x_i = |S_i|$. This is maximized by choosing $|S_i|$ vertices that have the most neighbors in S', if we fix $|S_i|$, represented as x_i in f-Knapsack. This is algebraically consistent with the recursive form in step 2.

For the running time, the choice of S' adds the complexity of 2^{cd} to the entire algorithm. Having chosen S', vertex sorting (step 1) can be accomplished by checking the edges between S' and $V \setminus X$, so it takes only $\mathcal{O}(k \cdot |V|)$. The f-Knapsack (step 3) takes time $\mathcal{O}\left(pk^2\right) = \mathcal{O}\left(k^2 \cdot |V|\right)$ from Corollary 13, because there are p items and weights are bounded by k. Steps 2 and 4 do not exceed this asymptotic running time. The total runtime is $\mathcal{O}\left(2^{\mathsf{cd}} \cdot k^2 \cdot |V|\right)$. ◀

6 Conclusions & Future Work

We present four algorithms using a variety of techniques, two for Maximum Happy Set (MaxHS) and two for Maximum Edge Happy Set (MaxEHS). The first shows that MaxHS is FPT with respect to the modular-width parameter, which is stronger than clique-with but generalizes several parameters such as neighborhood diversity and twin cover number. We then give an FPT dynamic-programming algorithm for MaxHS parameterized by clique-width. This improves the best known complexity result of FPT when parameterized by clique-width plus k.

For MaxEHS, we prove that it is FPT by neighborhood diversity, using Integer Quadratic Programming. Lastly, we show an FPT algorithm parameterized by cluster deletion number, the distance to a cluster graph, which then implies FPT by twin cover number. These results have answered several open questions of [1] (Figure 2). While it is FPT, there cannot be a polynomial kernel with respect to nd and cd, due to the lower-bounds on Clique parameterized by vertex cover number, unless NP $\subseteq$ coNP/poly [2].

There are multiple potential directions for future research. As highlighted in Figure 2, the parameterized complexity of MaxEHS with respect to modular-width is still open. Another direction would be to find the lower bounds for known algorithms.

References

1 Yuichi Asahiro, Hiroshi Eto, Tesshu Hanaka, Guohui Lin, Eiji Miyano, and Ippei Terabaru. Parameterized algorithms for the happy set problem. *Discrete Applied Mathematics*, 304:32–44, 2021.

2 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

3 Nicolas Bourgeois, Aristotelis Giannakos, Giorgio Lucarelli, Ioannis Milis, and Vangelis Th. Paschos. Exact and approximation algorithms for densest k-subgraph. In *7th International Workshop on Algorithms and Computation (WALCOM 2013)*, volume 7748, pages 114–125, 2013.

4 Hajo Broersma, Petr A. Golovach, and Viresh Patel. Tight complexity bounds for fpt subgraph problems parameterized by the clique-width. *Theoretical Computer Science*, 485:69–84, 2013.

5 Maurizio Bruglieri, Matthias Ehrgott, Horst W. Hamacher, and Francesco Maffioli. An annotated bibliography of combinatorial optimization problems with fixed cardinality constraints. *Discrete Applied Mathematics*, 154(9):1344–1357, 2006.

6 Leizhen Cai. Parameterized Complexity of Cardinality Constrained Optimization Problems. *The Computer Journal*, 51(1):102–121, 2007.

7 D. G. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9(1):27–39, 1984.

8 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000.

9 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

10 Reinhard Diestel. Graph theory. fifth. vol. 173. *Graduate Texts in Mathematics. Springer, Berlin*, 2018.

11 David Easley, Jon Kleinberg, et al. Networks, crowds, and markets: Reasoning about a highly connected world. *Significance*, 9(1):43–44, 2012.

12 Uriel Feige and Michael Seltser. On the Densest K-Subgraph Problem. *Algorithmica*, 29:2001, 1997.

13 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Clique-width: On the Price of Generality. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 825–834. Society for Industrial and Applied Mathematics, 2009.

14 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized by clique-width. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, SODA '10, pages 493–502, USA, 2010. Society for Industrial and Applied Mathematics.

15 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of Clique-Width Parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.

16 Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

17 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation*, pages 163–176. Springer International Publishing, 2013.

18 G. Gallo, P. L. Hammer, and B. Simeone. Quadratic knapsack problems. In M. W. Padberg, editor, *Combinatorial Optimization*, Mathematical Programming Studies, pages 132–149. Springer, Berlin, Heidelberg, 1980.

19 Robert Ganian. Improving Vertex Cover as a Graph Parameter. *Discrete Mathematics & Theoretical Computer Science*, Vol. 17 no.2, 2015.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.

21 Tesshu Hanaka. Computing Densest k-Subgraph with Structural Parameters, 2022. `arXiv:2207.09803`.

22 G. Kortsarz and D. Peleg. On choosing a dense subgraph. *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, 1993.

23 Michael Lampis. Algorithmic Meta-theorems for Restrictions of Treewidth. *Algorithmica*, 64(1):19–37, 2012.

24 Angsheng Li and Pan Peng. Community Structures in Classical Network Models. *Internet Mathematics*, 7(2), 2011.

25 Daniel Lokshtanov. Parameterized integer quadratic programming: Variables and coefficients, 2015. `arXiv:1511.00310`.

26 Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

27 Hannes Moser. Exact algorithms for generalizations of vertex cover. *Institut für Informatik, Friedrich-Schiller-Universität Jena*, 2005.

28 Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms. Academic Press Inc.*, 7(3):309–322, 1986.

29 Marc Tedder, Derek Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming*, pages 634–645, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

30 Peng Zhang and Angsheng Li. Algorithmic aspects of homophyly of networks. *Theoretical Computer Science*, 593:117–131, 2015.

A Appendix

Here we present a proof of Lemma 25 on a reduction from MaxEHS to IQP.

Proof. First, we compute the set of twins (modules) $\mathcal{M} = M_1, \dots, M_{\mathsf{nd}}$ of G, and obtain the quotient graph H on the modules $\mathcal{M}$ in time $\mathcal{O}(|V(G)| + |E(G)|)$ [23]. Note that each module M_i is either a clique or an independent set. Let us define a vector $q \in \mathbb{Z}^{\mathsf{nd}}$ such that $q_i = 1$ if M_i is a clique, and $q_i = 0$ if M_i is an independent set. Further, let $A \in \mathbb{Z}^{\mathsf{nd} \times \mathsf{nd}}$ be the adjacency matrix of H where $A_{ij} = 1$ if $M_i M_j \in E(H)$, and $A_{ij} = 0$ otherwise.

We then formulate an IQP instance as follows:

- Variables: $x \in \mathbb{Z}^{\mathsf{nd}}$.
- Maximize: $f(x) = x^T(A + qq^T)x - q^T x$ (equivalently, minimize $-f(x)$).
- Subject to: $\sum_i x_i = k$ and $0 \le x_i \le |M_i|$ for every $1 \le i \le \mathsf{nd}$.

This formulation has nd variables, and its coefficients are either 0 or ± 1, thus bounded. After finding the optimal vector x, pick any x_i vertices from module M_i and include them in the happy set S. We claim that S maximizes the number of happy edges.

For any happy set S, the number of happy edges is given by the sum of the number of happy edges inside each module M_i, which we call *internal edges*, and the number of edges between each module pair M_i and M_j, or *external edges*. Let $x \in \mathbb{Z}^{\mathsf{nd}}$ be a vector such that $x_i = |S \cap M_i|$ for every i. Then, the number of internal edges of module M_i is $q_i \cdot \binom{x_i}{2}$, and the number of external edges between modules M_i and M_j is $A_{ij} \cdot x_i x_j$. The number of happy edges, i.e. $|E(G[S])|$, is given by: $h(x) = \left[\sum_{i=1}^{\mathsf{nd}} q_i \cdot \binom{x_i}{2}\right] + \left[\sum_{1 \leq i < j \leq \mathsf{nd}} A_{ij} \cdot x_i x_j\right]$. One can trivially verify $f(x) = 2h(x)$.

If the IQP instance is feasible, then we can find a happy set S of size k maximizing $h(x)$, which must be the optimal solution to MaxEHS. Otherwise, $\sum_i |M_i| = |V(G)| < k$, and MaxEHS is also infeasible. ◀