Correlating Theory and Practice in Finding Clubs and Plexes

Aleksander Figiel
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Tomohiro Koana
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

André Nichterlein
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Niklas Wünsche
Unaffiliated Researcher, Berlin, Germany

Abstract

For solving NP-hard problems there is often a huge gap between theoretical guarantees and observed running times on real-world instances. As a first step towards tackling this issue, we propose an approach to quantify the correlation between theoretical and observed running times.

We use two NP-hard problems related to finding large "cliquish" subgraphs in a given graph as demonstration of this measure. More precisely, we focus on finding maximum s-clubs and s-plexes, i. e., graphs of diameter s and graphs where each vertex is adjacent to all but s vertices. Preprocessing based on Turing kernelization is a standard tool to tackle these problems, especially on sparse graphs. We provide a parameterized analysis for the Turing kernelization and demonstrate their usefulness in practice. Moreover, we demonstrate that our measure indeed captures the correlation between these new theoretical and the observed running times.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Preprocessing, Turing kernelization, Pearson correlation coefficient

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.47

Related Version *Full Version*: `https://arxiv.org/abs/2212.07533`

Supplementary Material *Software*: `https://git.tu-berlin.de/afigiel/splex-sclub-correl`
archived at `swh:1:dir:12573624f6d53cfd4d7e9181b0cf1596585fb93d`

Funding *Aleksander Figiel*: Supported by the Deutsche Forschungsgemeinschaft (DFG) project MaMu (NI 369/19).
Tomohiro Koana: Supported by the Deutsche Forschungsgemeinschaft (DFG) project DiPa (NI 369/21).

1 Introduction

Highly engineered solvers often perform much better than the known theoretical results would suggest. This is especially true when dealing with NP-hard problems. Unless P = NP, no efficient (i. e. polynomial-time) algorithm exists that solves all input instances correctly. However, optimized implementations can often solve instances with millions of vertices, variables, etc. as demonstrated frequently at algorithm engineering conferences [4, 35]. Of course, these implementations are not polynomial-time algorithms for NP-hard problems – the real-world instances are simply not worst-case instances but have algorithmically exploitable structures. On the other hand, there are usually relatively small instances making these solvers struggle. So theoretical running time bounds can often not predict observed running times on the given data set. Obviously, a better correlation between theoretical results and empirical findings would be highly desirable.

A multivariate (i. e. parameterized) analysis of the algorithm allows for a more nuanced picture of running time bounds. In principle, it could provide us with a much better prediction for the running time. However, a comparison to the theoretical parameterized running time is rarely made in practice (although there are notable exceptions [39, 21]). This is probably due to the multitude of issues arising here; let us mention just a few: For example, most theoretical bounds are stated using the O-notation that hides constants. Matching these to observed running times (which depend also on the used hardware) is not straightforward. Moreover, optimized solvers often combine several tricks that work well in different cases. The effect of these tricks is often hard to analyze and the observed running times most likely depend on a large (possibly unknown) set of parameters.

In this work, we use the Pearson correlation coefficient to quantify the correlation between theoretical bounds and observed running times. Our approach allows us to compare which theoretical running-time bound fits "better" to the observed running times for a given data set. Hence, it is relatively easy (but tedious) to generate hypotheses for theoretical bounds that fit well to experimental observations. Although proving such bounds is a different story, our approach can be used to transform practical results into impulses for theory.

We exemplify our approach on the s-Club and s-Plex problems and show how for various solver variants different theoretical explanations can be used. To this end, we follow the approach of Walteros and Buchanan [39] who demonstrated by means of a multivariate analysis why Clique is often efficiently solvable in relatively sparse graphs.

1.1 Related work

Clique on Sparse Graphs. Clique is one of Karp's 21 NP-complete problems [18]. As such, it is well studied, both in theory and practice; see Wu and Hao [40] for a survey. The currently fastest exact algorithm has running time $O(1.20^n)$ [41], where n is the number of vertices. While 1.20 seems very small, for a graph with 400 vertices the number of steps has more than 30 *digits* which is still infeasibly large. In contrast, a maximum clique can be found in real-world graphs with millions of vertices [8, 14, 31, 38].

It is easy to see that any clique is contained in the neighborhood of each of its vertices. Thus, a very basic approach solving clique on a sparse graph $G = (V, E)$ is the following. Take a vertex v of minimum degree and find the largest clique in $N[v]$ (the closed neighborhood of v). Then, remove v and continue in the same fashion. In the end, output the largest found clique. The *degeneracy* d of a graph is the size of the largest neighborhood encountered in the above algorithm. Hence, the above algorithm can be implemented to run in $1.20^d \cdot n^{O(1)}$ time which is on large sparse graphs far better than the $O(1.20^n)$ bound. Many of the graphs considered by Walteros and Buchanan [39] have several hundred thousand vertices and can be solved in less than a minute (often less than a second). Yet, some of these graphs have a degeneracy of well above 400 (again resulting in an infeasibly large number of steps). To rectify this, Walteros and Buchanan [39] provide an algorithm running in $1.28^g n^{O(1)}$ time, where $g := d - k + 1$ is called the *core-gap* and k denotes the number of vertices in a maximum clique (see Section 3 for a more detailed explanation). Clearly g can be much smaller than d. In fact, Walteros and Buchanan [39] observe that all their relatively small but hard-to-solve instances have a large core-gap.

Clubs and Plexes. An s-club is a graph of diameter s. An s-plex is a graph with ℓ vertices where every vertex has degree at least $\ell - s$. While not required by definition, in this work we only consider *connected* s-plexes. The task in s-Club / s-Plex is to find the largest s-club / s-plex in a given graph. Both s-Club and s-Plex are NP-hard as they contain Clique as special case ($s = 1$). Both problems are well-studied in the literature, both from theoretical and practical perspective. For example, s-Plex is W[1]-hard with respect to the parameter solution size k for all $s \geq 1$ [19, 23]. In contrast, if $s > 1$, then s-Club is fixed-parameter tractable with respect to the solution size k [33, 9]. We refer to Komusiewicz [22] for a further overview on the parameterized complexity of these problems. Several algorithmic approaches (heuristics and exact algorithms) have been proposed and examined to find maximum-cardinality clubs [6, 5, 7, 9, 15, 28, 32, 25, 24] or plexes [3, 12, 11, 36, 29]. Variants of the above-mentioned Turing kernelization approach are employed in engineered solvers for finding clubs [15, 32] and plexes [3, 29].

1.2 Our results

We start by transferring the approach of Walteros and Buchanan [39] to s-Club and s-Plex. To this end, introducing a new graph parameter, we describe and analyze the Turing kernelization for both problems in Section 3. Moreover, we provide simple branching algorithms showing fixed-parameter tractability with respect to a gap parameter. In Section 4, we then analyze the impact of the Turing kernelization in computational experiments for $s \in \{2, 3\}$. To this end, we use ILP-formulations with and without Turing kernelization and basic lower bounds. For s-Club significant speedups are observed whereas for s-Plex the improvements are not as clear (though still a speedup factor of more than 2.5 is achieved on average). Finally, in Section 5 we use correlations (more precisely the Pearson correlation coefficient) to analyze how well our theoretical findings fit to our practically observed running times. While this measure makes no statement about the efficiency of the algorithms, we can observe that even with the use of black boxes such as ILP-solvers our theoretical findings are reflected in the experimental results, in particular for the s-Club problem.

2 Preliminaries

For an integer $a \in \mathbb{N}$, we denote by $[a]$ the set $\{1, \dots, a\}$. For a graph $G = (V, E)$, let $n := |V|$ and $m := |E|$ be the number of vertices and edges, respectively. Let $u, v \in V$ be two vertices of G. Let $\operatorname{dist}_G(u, v)$ denote the length of any shortest path between u and v. For $x \in \mathbb{N}$, let $N_{x,G}(v)$ be the xth neighborhood of v, i. e., the set of vertices u with $1 \leq \operatorname{dist}_G(u, v) \leq x$, $N_{x,G}[v] = \{v\} \cup N_{x,G}(v)$, and $\deg_{x,G}(v)$ be the size of its xth neighborhood, i. e., $\deg_{x,G}(v) = |N_{x,G}(v)|$. For a set $X \subseteq V$ of vertices, let $G[X]$ denote the subgraph induced by X. We drop the subscript $\cdot_x$ for $x = 1$. Also, we omit the subscript $\cdot_G$ when G is clear from context.

Clique relaxations. Let X be a set of vertices. If the vertices of X are pairwise adjacent, then we say that X is a *clique*. Let $s \in \mathbb{N}$ be an integer. We say that X is an *s-club* if the vertices of X have pairwise distance at most s, i. e., $\max_{u,v \in X} \operatorname{dist}_{G[X]}(u, v) \leq s$ and that X is an *s-plex* if $G[X]$ is connected and every vertex v in X has at most $s - 1$ vertices nonadjacent to v in $X \setminus \{v\}$, i. e., $\max_{v \in X} |X \setminus N(v)| \leq s$. (Note that a 1-club and a 1-plex are each a clique.) We sometimes abuse these terms to refer to the subgraph induced by an s-club or s-plex. The decision problems s-Club and s-Plex ask, given a graph G and an integer $k \in \mathbb{N}$, whether G contains an s-club and s-plex, respectively, of size at least k.

Degeneracy. We say that a graph $G = (V, E)$ is *d-degenerate* if for every subgraph G' of G, there exists a vertex with $\deg_{G'}(v) \leq d$. Equivalently, G is d-degenerate if there is an ordering of V in which every vertex has at most d neighbors that appear later in the ordering.

We say that such an ordering is a *degeneracy ordering* of G. The degeneracy d_G of G is the smallest number d such that G is d-degenerate. For a vertex $v \in V$ and an ordering σ of V, we denote by $Q^\sigma_G(v)$ (and $Q^\sigma_G[v]$) the set of vertices in $N_G(v)$ (and $N_G[v]$) that appear after v in σ. We also omit the superscript $\cdot^\sigma$ when it is clear.

Parameterized complexity. Here, we list several relevant notions from parameterized complexity. See e. g., Cygan et al. [13] for a more comprehensive exposition of parameterized complexity. A parameterized problem is *fixed-parameter tractable* or *FPT* for short if every instance (I, k) can be solved in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for some computable function f. Such an algorithm is called an *FPT algorithm.* It is widely believed that a parameterized problem is not FPT if it is $W[i]$*-hard* for $i \in \mathbb{N}$. One way to show fixed-parameter tractability is via the notion of *Turing kernel.* For $t \in \mathbb{N}$, a t*-oracle* for a parameterized problem is an oracle that solves any instance (I, k) in constant time, provided that $|I| + k \leq t$. We say that a parameterized problem admits a Turing kernel of size $f(k)$ if there is an algorithm with access to an $f(k)$-oracle that solves (I, k) in time $(|I| + k)^{\mathcal{O}(1)}$. It is straightforward to turn a Turing kernel into an FPT algorithm by simply replacing an $f(k)$-oracle with a brute-force algorithm. The brute-force algorithm runs in $f'(k)$ time for some computable function f', resulting in an $f'(k) \cdot (|I| + k)^{\mathcal{O}(1)}$-time algorithm.

3 Theory

In this section, we provide theoretical analysis of clique relaxations based on the notion of Turing kernels. We first describe in Section 3.1 the algorithm for Clique outlined by Walteros and Buchanan [39], which runs in $1.28^g n^{\mathcal{O}(1)}$ time for the gap $g := d - k + 1$. The algorithms have two components. The first component is the Turing kernel parameterized by the degeneracy d. In short, we show that Clique is polynomial-time solvable when we have access to $f(d)$-oracle (see Section 2). In practice, there is no such convenient oracle so we have to provide some algorithm – this is the second component. One way to substitute the oracle is to use a brute-force algorithm. Since every oracle call takes an input whose size is bounded by d, we already obtain an FPT algorithm parameterized by d. We can actually make a more refined analysis by considering the gap parameter $g = d - k + 1$. Essentially, we use an FPT algorithm parameterized by g rather than relying on brute force. Walteros and Buchanan [39] showed that many Clique instances that can be solved efficiently in practice indeed have small values of g.

We want to adapt this approach to clique relaxations, namely, s-Club and s-Plex. However, there is one issue: Under standard complexity assumptions, there is no FPT algorithm for s-Club or s-Plex. More precisely, s-Club is known to be NP-hard for $s = 2$ and $d = 6$ [16] and s-Plex is known to be W[1]-hard when parameterized by $d + s$ [20]. For this reason, we consider a broader notion of degeneracy, which we call x*-degeneracy* for $x \in \mathbb{N}$ (1-degeneracy coincides with the standard degeneracy). We give the formal definition in Section 3.2. With the notion of x-degeneracy at hand, we describe how to adapt the approach employed by Walteros and Buchanan [39] to s-Club and s-Plex in Section 3.3.

3.1 Algorithm for Clique

We subsequently sketch the algorithmic approach of Walteros and Buchanan [39] for Clique.

Turing kernel. The Clique problem admits a Turing kernel, in which every input to the oracle has at most $d+1$ vertices (thus size $\mathcal{O}(d^2)$) as follows: For an instance (G, k) of Clique, consider a degeneracy ordering of $\sigma = (v_1, \ldots, v_n)$ of G. We will assume that $k \leq d + 1$ since a d-degenerate graph has no clique of size $d+2$. Observe that for every clique C of G, we have $C \subseteq Q^\sigma[v]$, where $v \in C$ is the vertex that appears first in a degeneracy ordering σ. Thus, G has a clique of size k if and only if there exists a vertex $v \in V$ such that $G[Q^\sigma[v]]$ has a clique of size k. Since G is d-degenerate, $G[Q^\sigma[v]]$ has at most $d+1$ vertices and size $\mathcal{O}(d^2)$. This leads to a Turing kernel for the parameter d: Simply call the oracle for the instances $(Q^\sigma[v_1], k), \dots, (Q^\sigma[v_n], k)$ and return yes iff at least one oracle answer was yes.

Oracle algorithm. Every oracle can be replaced with a brute-force algorithm running in $\mathcal{O}(2^d d^2)$ time: Since $Q^\sigma[v]$ is of size at most $d+1$, there are $\mathcal{O}(2^d)$ subsets of $Q^\sigma[v]$ and for every subset, it takes $\mathcal{O}(d^2)$ time to check if every pair of vertices are adjacent. Thus, Clique can be solved in $\mathcal{O}(2^d \cdot d^2 n)$ time. In fact, we can refine the analysis for the oracle algorithm in terms of the *gap* parameter $d-k+1$: To that end, we solve the Deletion to Clique problem: Given a graph G and an integer ℓ, the task is to find a set of at most ℓ vertices whose deletion results in a clique. There is a simple $\mathcal{O}(2^\ell n^2)$-time algorithm for this problem: If all vertices are pairwise adjacent and $\ell \geq 0$, then we have a yes-instance at hand. Otherwise, there exist two nonadjacent vertices, say u and v. If $\ell = 0$, then we can conclude that there is no solution. If $\ell \geq 1$, then we recursively solve two instances $(G-u, \ell-1)$ and $(G-v, \ell-1)$. This algorithm runs in $\mathcal{O}(2^\ell \cdot n^2)$ time. Since we need to solve this problem on $G[Q^\sigma(v)]$ with $\ell := |Q^\sigma[v]| - k \leq d-k+1$, we have an $\mathcal{O}(2^{d-k+1} \cdot d^2 n)$-time algorithm for Clique. We remark that an instance (G, ℓ) of Deletion to Clique is equivalent to the Vertex Cover instance $(\overline{G}, \ell)$ where $\overline{G}$ is the complement graph of G. Thus, using a faster known FPT algorithm for Vertex Cover [10], we obtain an $\mathcal{O}^*_d(1.28^{d-k+1} n)$-time algorithm for Clique ($\mathcal{O}^*_d$ hides factors polynomial in d).

3.2 Extending degeneracy

As mentioned in the beginning of this section, we consider a broader notion of degeneracy that can defined in two ways.

▶ **Definition 3.1.** *Let G be a graph and $x \in \mathbb{N}$. The* x-degeneracy *of G is the smallest integer $d_x \in \mathbb{N}$ such that for each subgraph G' of G, there exists a vertex v with $|N_{x,G'}(v)| \leq d_x$.*

▶ **Definition 3.2.** *Let G be a graph and $x \in \mathbb{N}$. The* x-degeneracy *of G is the smallest integer d_x such that there is an ordering $\sigma = (v_1, \dots, v_n)$ of G such that for every $i \in [n]$, the xth neighborhood of v_i in $G[v_i, \dots, v_n]$ has size at most d_x. The ordering σ is called an* x-degeneracy ordering*. The set of vertices in $N_{x,G}[v]$ that appear after v in σ is $Q^\sigma_{x,G}[v]$.*

It is not difficult to show that these two definitions are equivalent. We remark that the notion of 2-degeneracy has been proposed by Trukhanov et al. [36] in the context of finding s-plexes. The x-degeneracy and an x-degeneracy ordering can be found in polynomial time:

▶ **Theorem 3.3.** *Given a graph G and an integer $x \in \mathbb{N}$, we can compute the x-degeneracy of G and an x-degeneracy ordering of G in $\mathcal{O}(n^2 m)$ time.*

Proof. Repeat the following until the graph is empty: for every vertex v, compute the xth neighborhood of v. Find a vertex whose xth neighborhood has the smallest size and delete it from the graph. The ordering in which vertices are deleted is an x-degeneracy ordering. The x-degeneracy is the maximum over all vertices of the xth neighborhood size when they are deleted. Note that we spend $\mathcal{O}(nm)$ time to compute the xth neighborhood of every vertex using e.g., BFS. Since we repeat this n times, the algorithm runs in the claimed time. ◀

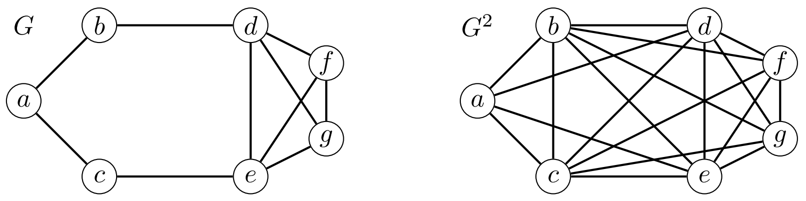

Figure 1 An example showing the difference between the 2-degeneracy of G (which is 5) and the degeneracy of G^2 (which is 6). The ordering $a, b, \dots, g$ is a 2-degeneracy ordering in G and a degeneracy ordering in G^2. Note that $\{b, c, \dots, g\}$ forms a clique in G^2 but not a 2-club in G.

We remark that a very similar graph parameter has been studied in the context of x-Club [25, 24]: the degeneracy of the x^{th} power graph G^x of G. The power graph G^x has the same vertices as G. Moreover, two vertices u and v are adjacent in G^x if and only if u and v have distance at most x in G. Note that every s-club in G is a clique in G^x. The converse, however, is not true; see Figure 1 for an example. While the degeneracy of G^x is in general larger than the x-degeneracy, it can be computed faster: in $O(nm)$ time [32]. In fact, the degeneracy of G^x has already been used in Turing kernelization for finding s-clubs [32]. Subsequently, we use the smaller x-degeneracy for stronger algorithmic results.

3.3 Algorithm for s-Club and s-Plex

Turing kernel. For s-Club, we adapt the Turing kernel for Clique as follows. For every s-club C of G, we have $C \subseteq Q_s^\sigma[v]$, where $v \in C$ is the first vertex of C in an s-degeneracy ordering σ. Thus, G has an s-club of size k if and only if there exists a vertex $v \in V$ such that $G[Q_s^\sigma[v]]$ has an s-club of size k. By the definition of x-degeneracy, we have $|Q_s^\sigma(v)| \leq d_s$. Thus, we have a Turing kernel in which every oracle call involves at most $d_s + 1$ vertices.

Oracle algorithm. Again, we can replace every oracle call with a brute-force algorithm. The input to every oracle call has at most $d_s + 1$ vertices and hence there are 2^{d_s+1} subsets. Moreover, for every subset, it takes $\mathcal{O}(d_s^3)$ time to determine whether the vertices have pairwise distance at most s, resulting in an algorithm running in $\mathcal{O}(2^{d_s} d_s^3)$ time. As in Section 3.1, we can also refine the algorithm substituting for the oracle using the parameter $d_s - k + 1$. To this end, we solve the Deletion to s-Club problem: Given a graph G and an integer ℓ, the task is to find a set of at most ℓ vertices whose deletion results in an s-club. There is a simple $\mathcal{O}(2^\ell n^3)$-time algorithm for this problem. If G has diameter at most s and $\ell \geq 0$, then we have a yes-instance at hand. Otherwise there exist two vertices, say u and v, with $\text{dist}_G(u, v) > s$. If $\ell = 0$, then we can conclude that there is no solution. If $\ell \geq 1$, then we recursively solve two instances $(G - u, \ell - 1)$ and $(G - v, \ell - 1)$. Since it takes $\mathcal{O}(n^3)$ time to compute all pairwise distances, this algorithm runs in $\mathcal{O}(2^\ell \cdot n^3)$ time. Since we need to solve this problem on $G[Q_s^\sigma[v]]$ with $\ell := |Q_s^\sigma[v]| - k \leq d_s - k + 1$, we obtain:

▶ **Theorem 3.4.** *Given the subgraph $G[Q_s^\sigma[v]]$ for every $v \in V$ for an s-degeneracy ordering σ, s-Club can be solved in $\mathcal{O}(2^{d_s - k} \cdot d_s^3 n)$ time.*

Turing kernel. For s-Plex, we will provide two adaptations. First, note that every s-plex is also an s-club (recall that we only consider connected s-plex). Thus the Turing kernel with the parameterization by d_s follows analogously. For another adaptation, we use the fact that any s-plex with at least $2s - 1$ vertices have diameter at most two, as observed by Seidman and Foster [34]: Suppose that two vertices u and v in an s-plex C have distance three in $G[C]$. Then, every vertex in C is nonadjacent to either u or v. Since for each of u and v, there are at most $s-1$ vertices nonadjacent to it, we have $|C| \leq 2s-2$. This leads to a Turing kernel with respect to the parameter d_2 when $k \geq 2s-1$. For every s-plex of size at least $2s-1$, we have $C \subseteq Q_2^\sigma[v]$, where $v \in C$ is the first vertex of C in a 2-degeneracy ordering σ. Thus, we have again a Turing kernel where every oracle call involves at most d_2+1 vertices. (Small s-plex of size at most $2s-2$ can be found in $O(n^{2s-1})$.)

Oracle algorithm. Again, we can replace every oracle call on h vertices with a brute-force algorithm. The input to every oracle call has at most $h+1$ vertices and hence there are 2^{h+1} subsets. Moreover, for every subset, it takes $\mathcal{O}(h^2)$ time to determine whether it is an s-plex, resulting an algorithm running in $\mathcal{O}(2^h h^2)$ time. As in Section 3.1, we can also refine the algorithm substituting for the oracle using the parameter $h-k+1$. To that end, we solve the DELETION TO s-PLEX problem: Given an n-vertex graph G and an integer ℓ, the task is to find a set of at most ℓ vertices whose deletion results in an s-plex. There is a simple $\mathcal{O}((s+1)^\ell n^2)$-time algorithm for this problem. If G is an s-plex and $\ell \geq 0$, then we have a yes-instance at hand. Otherwise there exist a vertex v and s vertices nonadjacent to v. If $\ell = 0$, then we can conclude that there is no solution. If $\ell \geq 1$, then we recursively solve $s+1$ instances $(G-v, \ell-1)$ and $(G-u, \ell-1)$ where u is one of s vertices nonadjacent to v. Since it takes $\mathcal{O}(n^2)$ time to check if the graph is an s-plex, this algorithm runs in $\mathcal{O}((s+1)^\ell \cdot n^2)$ time. Since we need to solve this problem on graphs with h vertices with $\ell := h-k$, we obtain the following:

▶ **Theorem 3.5.** *Given the subgraph $G[Q_s^\sigma[v]]$ for every $v \in V$ for an s-degeneracy ordering σ, s-PLEX can be solved in time $\mathcal{O}((s+1)^{d_s-k} \cdot d_s^2 n)$ and $\mathcal{O}(n^{2s-1} + (s+1)^{d_2-k} \cdot d_2^3 n)$.*

We remark that for very small s, the first term n^{2s-1} can be ignored in practice, because most instances contain an s-plex of size at least $2s-1$.

4 Experiments

In this section we present the results of our computational experiments for s-CLUB and s-PLEX for $s \in \{2,3\}$ on a large dataset of real-world graphs. We did not optimize every aspect of the implementations as our goal is to investigate the effect of Turing kernelization and the extent to which our theoretical findings are reflected in the running time (this is discussed in Section 5). We will see, that the Turing kernelization is quite beneficial for s-CLUB but for s-PLEX the situation is not as clear.

Setup & Dataset. All experiments were performed on a machine running Ubuntu 18.04 LTS, with an Intel Xeon® W-2125 CPU and 256GB of RAM. A maximum running time of 1 hour per instance was set. We used Gurobi 8.1 to solve ILP-formulations, limited to a single thread of execution. The program that was used to build the ILP models was implemented in C++ and compiled with g++ 7.5.

The static graphs from the Network Repository [30] were used for all experiments. Graphs for which at least one solver configuration timed out, ran out of memory, or completed in less than 0.05 seconds were omitted. The last case (less than 0.05 s) is to not deal with the effect of noise in the small running time measurements. The resulting dataset consists of 259 graphs, with 1033 vertices on average. The source code is available at `https://git.tu-berlin.de/afigiel/splex-sclub-correl`.

We remark that s-Club and s-Plex have been solved for small s on much larger graphs within minutes [15, 12, 11]. The reason we focus on smaller graphs is to have a meaningful multivariate analysis. More precisely, we want to see if the running time grows (as suggested by theory) with growing x-degeneracy and gap. Having running times for large graphs with small x-degeneracy and gap but not for large graphs with large x-degeneracy and gap would give misleading results in our analysis in Section 5.

Solvers. We used an ILP solver as oracle for s-Club and s-Plex in the Turing kernelization. The ILP formulations were taken from the literature: For s-Plex we used a straight-froward formulation with $\mathcal{O}(n)$ variables and constraints and $\mathcal{O}(n+m)$ non-zeroes[1] [27].

$$\begin{array}{ll}
\text{maximize:} & y \\
\text{subject to:} & x_v \in \{0,1\}, y \in \{0,\dots,n\} \\
 & y = \sum_{v \in V} x_v \\
\forall v \in V\colon & |V|(1-x_v) + \sum_{u \in N(v)} x_u \geq y - s
\end{array}$$

For 2-Club a simplified formulation by Bourjolly et al. [6] was used. It has $\mathcal{O}(n)$ variables, $\mathcal{O}(n^2)$ constraints, and $\mathcal{O}(n^3)$ non-zeroes.

$$\begin{array}{ll}
\text{maximize:} & \sum_{v \in V} x_v \\
\text{subject to:} & x_v \in \{0,1\} \\
\forall u, v \in V, \operatorname{dist}(u,v) > 2\colon & x_u + x_v \leq 1 \\
\forall u, v \in V, \operatorname{dist}(u,v) = 2\colon & x_u + x_v \leq 1 + \sum_{c \in N(u) \cap N(v)} x_c
\end{array}$$

For 3-Club the neighborhood formulation from Almeida and Carvalho [1, 2] was used, with $\mathcal{O}(n)$ binary variables, $\mathcal{O}(m)$ continuous variables, at most $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ constraints and non-zeroes, respectively[2].

$$\begin{array}{ll}
\text{maximize:} & \sum_{v \in V} x_v \\
\text{subject to:} & x_v \in \{0,1\} \\
\forall u, v \in V, \operatorname{dist}(u,v) > 3\colon & x_u + x_v \leq 1 \\
\forall u, v \in V, \operatorname{dist}(u,v) \in \{2,3\}\colon & x_u + x_v \leq 1 + \sum_{c \in N(u) \cap N(v)} x_c + \sum_{e \in E_{uv}} z_e \\
\forall e = \{a,b\} \in E\colon & z_e \leq x_a,\ z_e \leq x_b \\
\forall e \in E\colon & 0 \leq z_e \leq 1
\end{array}$$

[1] An s-plex of size $\ell > 2s-1$ is guaranteed to be connected and of diameter two [34]. As we only consider $s \in \{2,3\}$, we do not add constraints enforcing connectedness to the ILP. Unsurprisingly, all found subgraphs were still connected.

[2] For compact and general ILP formulations for s-Club ($s \geq 2$) we refer to Salemi and Buchanan [32] and Veremyev et al. [37].

Table 1 Average running times in seconds of various solver configurations.

	noTK	full	default	hint
2club	294.3	18.7	1.9	0.9
3club	641.8	296.0	81.4	41.7
2plex	26.4	63.8	10.1	10.2
3plex	39.6	260.6	81.4	77.6
3plex-2	39.6	63.4	12.4	12.0

where $E_{uv} = \{\{p, q\} \in E \mid p \in N(u) \setminus N(v), q \in N(v) \setminus N(u)\}$. We remark that different ILP-formulations for these problems are discussed in the literature [3, 27, 32]. However, analyzing them is beyond the scope of this work.

Solver variants. We tested several approaches using these ILP models to cover all concepts discussed in Section 3. To this end, we use four different solver configurations, namely `noTK`, `full`, `default` and `hint` (described below). We will refer to, for example, `2club_noTK` as the benchmark results of the `noTK` solver configuration on the 2-Club problem.

The `noTK` variant (no Turing kernel) simply built a single ILP model for the entire graph. All other variants use the Turing kernelization to some extent. The `full` variant makes only basic use of Turing kernelization, utilizing the 2/3-degeneracy as described in Section 3. There, each oracle call is solved via an ILP. For 2-Club and 2-Plex the Turing kernelization using 2-degeneracy is employed, for 3-Club and 3-Plex the one using 3-degeneracy. As there is only one connected 3-plex of diameter three (the P_4) which was never the largest 3-plex in our experiments we also used the 2-degeneracy based Turing kernelization for 3-Plex. We report the results of this variant under `3plex-2` (thus `3plex-2_noTK` is the same as `3plex_noTK`).

The `default` variant uses the Turing kernel approach in combination with a simple lower bound: It uses the maximum solution size of already solved ILPs as a lower bound on the global solution size by adding a constraint to the ILPs enforcing that the solution has to be larger than the current lower bound. Note that this lower bound does not appear in the algorithm descriptions in Section 3. The order in which the ILPs are solved can therefore have an impact on the overall running time. We used a fixed 2/3-degeneracy ordering and did not analyze the impact of the order. Instead, we remove this effect in the `hint` variant. There, we added a constraint to each ILP model in the Turing kernels which enforced that the solution size to the Turing kernel is at least the size of the optimal solution size for the entire graph. Thus, one can think of (heuristically) optimizing in the `default` variant the order in the Turing kernelization so that the oracle calls giving the largest results come first. Alternatively, this shows the maximum speedup possible by a "perfect" lower-bound heuristic. Note that in the `hint` variant at least one ILP model still has a feasible solution.

Results. In Table 1 we summarize the average running time of the different solver configurations on the four considered problems. The approach without Turing kernels is significantly slower for 2- and 3-Club but not so much for 2- and 3-Plex. This can also be seen in the detailed comparisons in Figure 2. Interestingly, the `noTK` variants are much faster in finding plexes than in finding clubs (more than 10 times larger average running time). However, the average running time of `2club_default` is five times smaller than that of `2plex_default`. Thus, on the one hand the ILP-formulation we use for finding 2/3-clubs may have a room for

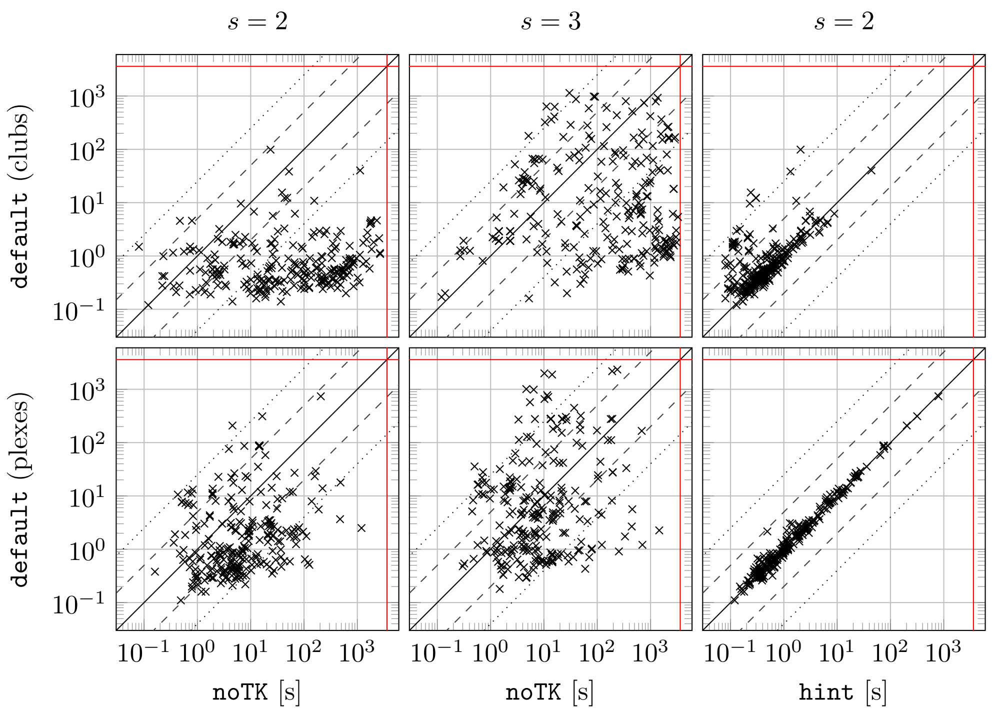

Figure 2 Running time comparison (in seconds) of different variants (top row for 2- and 3-Club, second row for 2- and 3-Plex). Each cross represents one instance with the x- and y-coordinates indicating the running time of the respective variant (in seconds): `default` and `noTK` resp. `hint`. Thus, a cross above (below) the solid diagonal indicates that the solver on the x-axis (y-axis) is faster on the corresponding instance. The diagonal lines mark factors of 1 (solid), 5 (dashed) and 25 (dotted). The solid horizontal red lines indicate the time limit (1 hour). For 2- and 3-Club a significant running time improvement is visible. For 2- and 3-Plex the picture is not so clear.

improvement. On the other hand, the Turing kernel approach works much better for clubs than plexes. The reason is most likely that the Turing kernels are built based on distance, which fits better with clubs than plexes. This can be seen in the gap-parameter: for 73 resp. 57 instances the gap-parameter was zero for 2-Club resp. 3-Club whereas for the plexes the gap-parameter was never zero.

Unsurprisingly, the `hint` variant is the fastest one. However, the `default` variant is not much slower than `hint` (see also right column of Figure 2), even though it does not receive the solution size as input, and instead uses the maximum solution of the previously solved ILPs to update the lower bound. Moreover, the `default` variant is considerably faster than the `full` variant. This shows the strength of providing lower bounds to the ILP-solver. Remarkably, for finding 2/3-clubs even the `full` variant brings a decent speedup compared to the `noTK` variant.

5 Correlation between Theory and Practice

We now analyze correlations between the theoretical running time bounds in Section 3 and the measured running times in Section 4. Since we have NP-hard problems, our working hypothesis is that the running time should depend exponentially on some parameter(s).

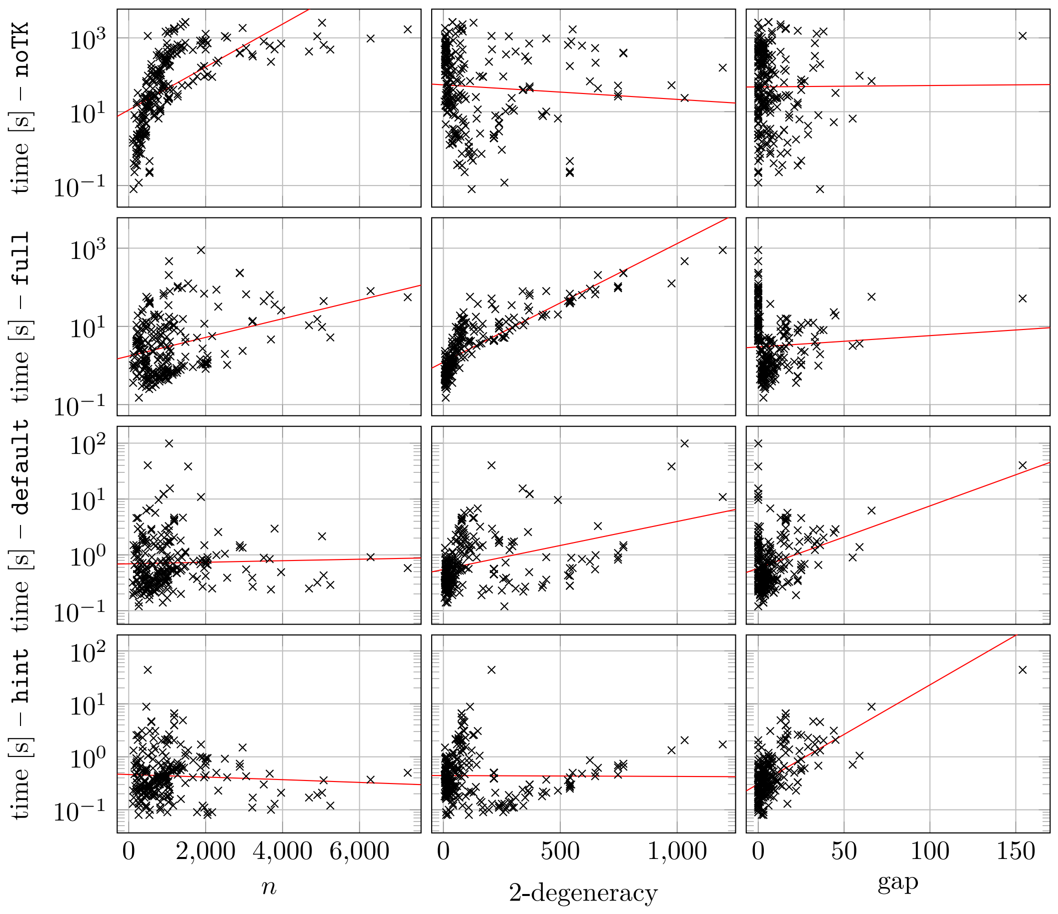

Figure 3 The running times of `2club_noTK`, `2club_full`, `2club_default`, and `2club_hint` plotted against the number n of vertices, the 2-degeneracy, and the gap of the input graph. The solid red lines are linear regressions best fitting to the data points (where the logarithm of the running times is taken).

Natural parameter candidates are the number n of vertices, the 2- resp. 3-degeneracy, and the gap parameter (suggested by our theoretical findings). Note that there is a multitude of other viable parameters[3]; however, studying them is beyond the scope of this work.

We studied five problems (counting `3plex` and `3plex-2` as two) with four different solvers of each problem. Thus, there are $5 \cdot 4 \cdot 3 = 60$ different (parameter, running time)-pairs to analyze. We illustrate the 12 pairs for `2club` in Figure 3 with the red lines depicting the exponential function of the form $\alpha^p \cdot \beta$ that best fit the data; these lines are computed via linear regression (logarithm of the running time versus parameter value p). Obviously, the suggested running time function on the bottom left (`2club_hint` with parameter n) is useless: our implementation will in general not become faster the larger the input gets. The red lines on the top left and bottom right seem far more sensible.

[3] See for example `https://manyu.pro/assets/parameter-hierarchy.pdf` or `https://www.graphclasses.org/` for dozens of graph parameters.

5.1 Method

Instead of "carefully looking" at each of the plots in Figure 3 and finding arguments for each one of them, we want an automated way of distinguishing useful from useless suggestions. To this end, we suggest using the Pearson correlation coefficient which we subsequently just call correlation coefficient. It is a standard measure of linear correlation between two sets of data. Simply put, given the (running time, parameter) data points the correlation coefficient computes a number between -1 and 1. If there is no correlation at all, then the coefficient is 0. With perfect correlation (i. e. the data points are on a straight line with positive slope) the coefficient is 1. With perfect negative correlation (i. e. the data points are on a straight line with negative slope) it is -1.

We report correlations between the logarithm of the running time and the parameter (in our case either the number of vertices, the generalized degeneracy, or the gap). Thus, values close to 1 represent good correlations (i.e. some exponential dependency between the running time and the parameter). However, note that a better correlation value does not imply a better running time. There are several specifics with our measure that we like to address before reporting the results.

Exponential Dependency. Note that our approach "ignores" the base of the exponential functions, in the following sense: A "perfect correlation" of 1 with, say the gap parameter g, implies that the measured running time t satisfies the following linear relation for some constant a and b: $\log t = a \cdot g + b \iff t = 2^{a \cdot g + b} = \beta \cdot \alpha^g$ where $\beta = 2^b$ and $\alpha = 2^a$ are again two constants. Of course, α has to be nowhere close to the bases of the exponential functions we describe in Section 3. Thus, a "perfect correlation" just indicates that there is *some* exponential dependency between the parameter and the running time.

Our justification here is that the measure needs to be somewhat imprecise: The measured running times can be greatly impacted by the configuration of the ILP-solver [17] or by the experimental setup [26]. Hence, hitting the theorized worst-case running time seems rather unlikely. Since we deal with NP-hard problems we expect (despite all heuristic improvements) some exponential function describing the running time. With our setting we can get suggestions for the base of the exponential function from experimental results.

Restricted Setting. We restrict ourselves to correlations between one parameter and the running time. While correlations between multiple parameters and the running time are possible, our theoretical results in Section 3 only suggest exponential dependencies between one parameter (2/3-degeneracy or gap) and the running time and not two parameters. Incorporating the polynomial factors of n in the running times of Section 3 is easily doable. However, in our analysis it changed the coefficients only marginally (by less than 5%, usually much less than 1%), probably due to the relatively small size of the graphs. As we are (for now) only interested in simple exponential dependencies, the Pearson correlation coefficient suffices as we can take the logarithm of all measured running times. Note that there are different correlation coefficients that can also measure non-linear correlations and might be better suited to other settings (e. g. when dealing with polynomial-time solvable problems).

Simplistic Analysis. We use a very simplistic statistical analysis. For example, we ignore confidence intervals, p-values, or similar issues. The reason being that any "good" correlation between a parameter (or a combination of parameters) and the measured running time is only an *indication* for such a correlation. In particular, if some "new" correlations are discovered with this method, then this only gives *suggestions*. Using our theoretical tools, we

Table 2 Tables summarizing the correlation of different graph parameters n (left table), d_2/d_3 (middle table), and the gap g (right table) with the logarithm of the measured running times of various solver configurations (def abbreviates default). In the middle the correlation with 2-degeneracy is shown for `2club, 2plex` and `3plex-2`, and with 3-degeneracy for `3club` and `3plex`).

	Correlation with n				Correlation with d_2/d_3				Correlation with gap			
	noTK	full	def	hint	noTK	full	def	hint	noTK	full	def	hint
2club	0.59	0.34	0.03	−0.06	−0.08	0.84	0.38	−0.01	0.01	0.06	0.35	0.61
3club	0.48	0.16	0.08	−0.22	−0.12	0.67	0.46	0.22	0.14	0.34	0.46	0.72
2plex	0.53	0.07	0.08	0.07	0.21	0.63	0.46	0.46	0.22	0.60	0.44	0.45
3plex	0.52	0.06	−0.02	−0.01	0.02	0.65	0.40	0.39	0.02	0.63	0.39	0.38
3plex-2	0.52	0.08	0.07	0.06	0.02	0.62	0.45	0.44	0.02	0.59	0.42	0.42

can prove (e. g. show a running time bound) or disprove (e. g. NP-hardness for a constant parameter value) such suggestions. This ability of theoretical verification allows us to ignore "safety"-features from statistical analysis.

5.2 Results

Table 2 summarizes the 60 correlation coefficients of three graph parameters with the logarithm of the measured running times.

Consider the first row corresponding to 2-Club in the three parts of Table 2. The first columns display the correlation with n which is best for the `noTK` variant. This well reflects our observations for the plots in the left column of Figure 3: The `default` and `hint` variant do not display any reasonable correlation with n, only `noTK` does to some extent. Similarly, in the right column of Figure 3 the correlations of the `default` and `hint` variant with the gap-parameter are quite decent, but not for the `noTK` variant. Moreover, in the plot for the `default` variant of the right column in Figure 3 there are a few instances that have a high running time despite a parameter value of zero. This is an argument against the suggested regression being a "good" explanation. Also, in the bottom right plot one can see that there are no such (drastic) outliers. Hence, the correlation with the `hint` variant with the gap is considerably "better" than with the slower `default` variant (despite only small differences in the average running time, see Table 1). This is also reflected in the corresponding correlation coefficients of 0.61 and 0.35 respectively (see two rightmost columns in Table 2) and, thus, supports the correlation coefficient as reasonable measure. All in all, the results for 2-Club indicate that the correlation coefficients reveal exponential dependencies between the running time and the considered parameter.

Correlations with number of vertices. The results for the other problems are somewhat similar to the ones for 2-Club. The correlation coefficient for the number of vertices is highest for all problems with the `noTK` configuration, whereas with the configurations based on Turing kernels it is significantly lower (or even negative). This is somehow expected, as all our ILP formulations use $\mathcal{O}(n)$ binary variables. State-of-the-art ILP solvers are highly complex ("a bag of tricks") and able to solve instances with millions of integer variables efficiently. Thus, the correlation of around 0.5 (for `noTK`) with the number of vertices is higher than for the other variants, but not the overall highest correlations (see second row and second column in Figure 3 for the plot corresponding to the highest correlation).

Correlations with generalized degeneracy. The Turing kernel approaches correlate in general better with the 2/3-degeneracy, notable exceptions are the `2club_hint` and `3club_hint` variants. As expected, across all problems the highest correlations with the 2/3-degeneracy are achieved by the `full` variants: The 2-degeneracy (3-degeneracy) is in our dataset on average more than five times (more than three times) smaller than n. Hence, the high correlations for the `noTK` variants with n translate to high correlations for the `full` variants with 2/3-degeneracy. For the `noTK` configuration there is barely any correlation with the 2/3-degeneracy. It thus seems that the ILP solver cannot exploit the 2/3-degeneracy – at least with the given ILP formulations.

Correlations for `default` and `hint` variants. As discussed in Section 4, the `default` and `hint` variants are considerably faster than the `full` variants due to having access to some (perfect) lower bound. As we use the black box of an ILP-solver we do not have theoretical running time bounds covering the effect of this lower bound. Also note that the running time differences between the `default` and `hint` variants are much smaller than the differences between other pairs of variants (see Table 1). However, we can observe significant differences in the correlations of the `default` and `hint` variants for 2-Club and 3-Club with respect to the generalized degeneracy and the gap parameter. Moreover, the correlation coefficients support some speculations: The correlations in the middle table of Table 2 suggest that this running time improvement is not (so much) correlated to the 2/3-degeneracy but to another parameter. For finding clubs the gap-parameter is a good explanation: `2club_hint` and `3club_hint` have high correlations with the gap parameter; this can also be seen in the bottom right 2×2 plot subgrid of Figure 3. Thus, with a better lower bound computation (i. e., some actual heuristic) we suspect the correlation of the `default` variant with the degeneracy to decrease and increase with the gap parameter.

For plexes this argumentation does not hold. There is rarely any difference in the correlation coefficients of the `default` and `hint` variant with the 2/3-degeneracy and the gap-parameter. The reason is simple: while the gap is considerably smaller than the 2/3-degeneracy for 2/3-Club (the clubs are rather large), this is not the case for 2/3-Plex (the plexes are quite small), see Figure 4 (in the appendix). Thus, for 2/3-Plex the correlations differ only marginally between the `hint` and `default` variants. Moreover, this explains very well why despite `2club_noTK` being quite slow compared to `2plex_noTK` the variant `2club_hint` is much faster than `2plex_hint`: The average gap for our 2-club instances is 7.9, hence the exponential running time dependency on the gap is manageable. For 3-club instances, the average gap is 18.6 which, apparently, is one of the reasons why the `3club` variants are much slower than the `2club` variants.

6 Conclusion

We provided theoretical bounds for algorithms solving s-Club and s-Plex and experimentally tested the employed Turing kernelization for $s \in \{2, 3\}$. We found that the Turing kernel approach improves the running time significantly more for clubs than plexes. We believe that this is due to the fact that the x-degeneracy is defined based on distance. This suggests the need for exploring notions more suitable for finding plexes.

We also discussed the correlation between the observed running times and the theoretical bounds. Yet, there is still a large gap between theory and practice: for example, the bases of the exponential function obtained by regression are all below 1.1 – much smaller than current theoretical results suggest. We are confident that our approach based on correlation coefficients can help to close this gap. The approach is easy to employ and quite flexible.

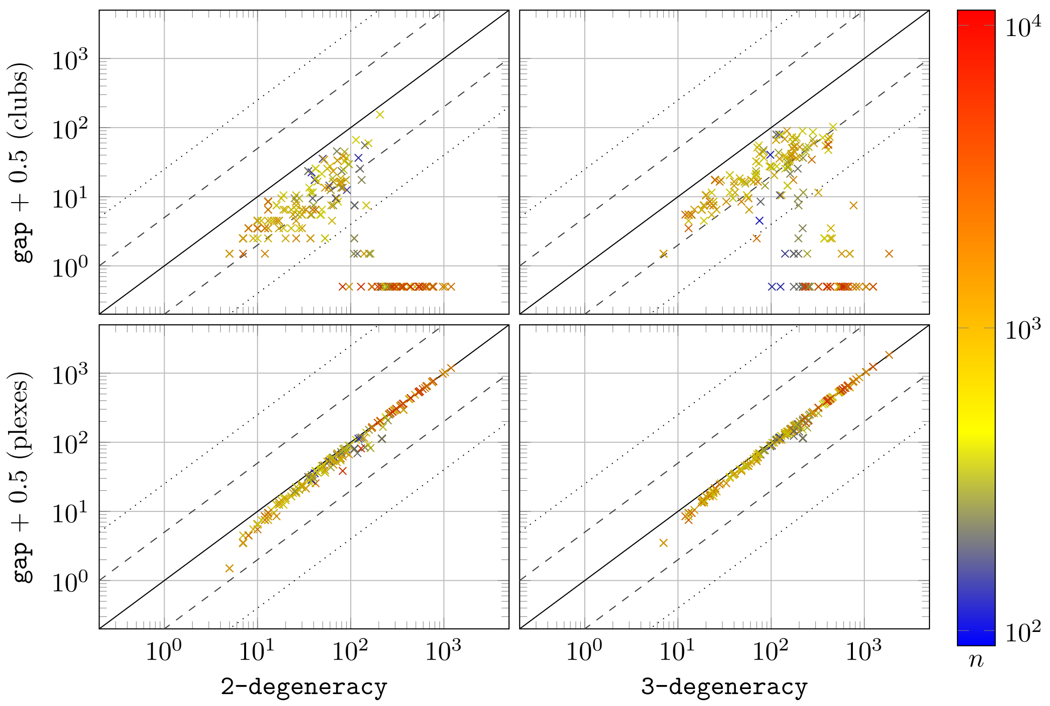

Figure 4 Relation between 2-degeneracy, gap, and number of vertices. We added 0.5 to the gap as we use log scale and several instances have a gap of zero (for `2club` and `3club`). The diagonal lines mark factors of 1 (solid), 5 (dashed) and 25 (dotted).

Finally, we discuss some directions for future work:

- Checking whether the running times correlate with multiple parameters is an easy (but tedious) extension. The whole process should allow for relatively easy automation. An automated tool could generate a list of likely correlations from experimental results. These can then be analyzed theoretically with the parameterized complexity framework. This way, practice could give more impulses for theory.
- Our approach is not limited to analyzing running times. Other objectives could be the size of preprocessed instances (using the kernelization framework from parameterized algorithmics) or approximation factors of heuristics or approximation algorithms.
- While we use worst-case analysis, average case analysis or smoothed analysis are further natural candidates.
- Improving our general approach: For example, how to incorporate timeouts? Or are different correlation coefficients better suited?

References

1 Maria Teresa Almeida and Filipa D. Carvalho. The k-club problem: new results for k= 3. *CIO - Centro de Investigação Operacional*, CIO - Working Paper 3/2008, 2008.

2 Maria Teresa Almeida and Filipa D. Carvalho. An analytical comparison of the lp relaxations of integer models for the k-club problem. *European Journal of Operational Research*, 232(3):489–498, 2014. doi:10.1016/j.ejor.2013.08.004.

3 Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011. doi:10.1287/opre.1100.0851.

4 Thomas Bläsius, Tobias Friedrich, David Stangl, and Christopher Weyand. An efficient branch-and-bound solver for hitting set. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX '22)*, pages 209–220. SIAM, 2022. doi:10.1137/1.9781611977042.17.

5 Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. Heuristics for finding k-clubs in an undirected graph. *Computers & Operations Research*, 27(6):559–569, 2000.

6 Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research*, 138(1):21–28, 2002. doi:10.1016/S0377-2217(01)00133-3.

7 Austin Buchanan and Hosseinali Salemi. Parsimonious formulations of low-diameter clusters. *Optimization Online Eprints*, 2017.

8 Lijun Chang. Efficient maximum clique computation over large sparse graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019)*, pages 529–538. ACM, 2019. doi:10.1145/3292500.3330986.

9 Maw-Shang Chang, Ling-Ju Hung, Chih-Ren Lin, and Ping-Chen Su. Finding large k-clubs in undirected graphs. *Computing*, 95(9):739–758, 2013. doi:10.1007/s00607-012-0263-3.

10 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

11 Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. Cliques are too strict for representing communities: Finding large k-plexes in real networks. In *Proceedings of the 26th Italian Symposium on Advanced Database Systems*, volume 2161 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: http://ceur-ws.org/Vol-2161/paper41.pdf.

12 Alessio Conte, Tiziano De Matteis, De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. D2K: scalable community detection in massive networks via small-diameter k-plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*, pages 1272–1281. ACM, 2018. doi:10.1145/3219819.3220093.

13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

14 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM J. Exp. Algorithmics*, 18, 2013. doi:10.1145/2543629.

15 Sepp Hartung, Christian Komusiewicz, and André Nichterlein. Parameterized algorithmics and computational experiments for finding 2-clubs. *Journal of Graph Algorithms and Applications*, 19(1):155–190, 2015. doi:10.7155/jgaa.00352.

16 Sepp Hartung, Christian Komusiewicz, André Nichterlein, and Ondrej Suchý. On structural parameterizations for the 2-club problem. *Discrete Applied Mathematics*, 185:79–92, 2015. doi:10.1016/j.dam.2014.11.026.

17 Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010)*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer, 2010. doi:10.1007/978-3-642-13520-0_23.

18 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

19 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.

20 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Computing dense and sparse subgraphs of weakly closed graphs. In *Procedings of the 31st International Symposium on Algorithms and Computation (ISAAC 2020)*, pages 20:1–20:17, 2020. doi:10.4230/LIPIcs.ISAAC.2020.20.

21 Tomohiro Koana, Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. *ACM Journal of Experimental Algorithmics*, 26:1.3:1–1.3:30, 2021. doi:10.1145/3439801.

22 Christian Komusiewicz. Multivariate algorithmics for finding cohesive subnetworks. *Algorithms*, 9(1):21, 2016. doi:10.3390/a9010021.

23 Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38-40):3640–3654, 2009. doi:10.1016/j.tcs.2009.04.021.

24 Yajun Lu, Esmaeel Moradi, and Balabhaskar Balasundaram. Correction to: Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts. *Optimization Letters*, 12(8):1959–1969, 2018. doi:10.1007/s11590-018-1273-7.

25 Esmaeel Moradi and Balabhaskar Balasundaram. Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts. *Optimization Letters*, 12(8):1947–1957, 2018. doi:10.1007/s11590-015-0971-7.

26 Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. Producing wrong data without doing anything obviously wrong! In Mary Lou Soffa and Mary Jane Irwin, editors, *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS 2009)*, pages 265–276. ACM, 2009. doi:10.1145/1508244.1508275.

27 Mohammad Javad Naderi, Austin Buchanan, and Jose L. Walteros. Worst-case analysis of clique mips. *Mathematical Programming*, 195(1):517–551, 2022. doi:10.1007/s10107-021-01706-2.

28 F. Mahdavi Pajouh and B. Balasundaram. On inclusionwise maximal and maximum cardinality k-clubs in graphs. *Discrete Optimization*, 9:84–97, 2012.

29 Foad Mahdavi Pajouh, Esmaeel Moradi, and Balabhaskar Balasundaram. Detecting large risk-averse 2-clubs in graphs with random edge failures. *Annals of Operations Research*, 249(1-2):55–73, 2017. doi:10.1007/s10479-016-2279-0.

30 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. accessed 01.01.2022. URL: https://networkrepository.com.

31 Ryan A. Rossi, David F. Gleich, and Assefaw Hadish Gebremedhin. Parallel maximum clique algorithms with applications to network analysis. *SIAM J. Sci. Comput.*, 37(5), 2015. doi:10.1137/14100018X.

32 Hosseinali Salemi and Austin Buchanan. Parsimonious formulations for low-diameter clusters. *Mathematical Programming Computation*, 12(3):493–528, 2020. doi:10.1007/s12532-020-00175-6.

33 Alexander Schäfer, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Parameterized computational complexity of finding small-diameter subgraphs. *Optimization Letters*, 6(5):883–891, 2012. doi:10.1007/s11590-011-0311-5.

34 Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6(1):139–154, 1978.

35 Darren Strash and Louise Thompson. Effective data reduction for the vertex clique cover problem. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX '22)*, pages 41–53. SIAM, 2022. doi:10.1137/1.9781611977042.4.

36 Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications*, 56(1):113–130, 2013. doi:10.1007/s10589-013-9548-5.

37 Alexander Veremyev, Oleg A. Prokopyev, and Eduardo L. Pasiliao. Critical nodes for distance-based connectivity and related problems in graphs. *Networks*, 66, 2015. doi:10.1002/net.21622.

38 Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS J. Comput.*, 27(1):164–177, 2015. doi:10.1287/ijoc.2014.0618.

39 Jose L. Walteros and Austin Buchanan. Why is maximum clique often easy in practice? *Operations Research*, 68(6):1866–1895, 2020. doi:10.1287/opre.2019.1970.

40 Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015. doi:10.1016/j.ejor.2014.09.064.

41 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.