Algorithms and Experiments for Finding Robust 2-Clubs

Masterarbeit
zur Erlangung des akademischen Grades
Master of Science im Studiengang Informatik

Technische Universität Berlin
Fakultät IV - Elektrotechnik und Informatik

eingereicht von Marten Picker
geboren am 16.05.1985 in Leipzig

Betreuer: Prof. Dr. Rolf Niedermeier,
Dr. Sepp Hartung,
Dr. Christian Komusiewicz,
Dr. André Nichterlein

Berlin, 13. August 2015

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Ort, Datum	Unterschrift

Zusammenfassung

Das NP-vollständige 2-Club ist ein gründlich untersuchtes Berechnungsproblem mit vielfältigen Anwendungen, zum Beispiel in der Analyse sozialer Netzwerke. Es fragt nach dem Subgraphen mit Durchmesser zwei mit der größten Anzahl Knoten in einem gegebenen Graphen. Es wurde als ein Modell für kohäsive Untergruppen in sozialen Netzwerken, d.h. Gruppen von Akteuren welche in besonderem Ausmaße interagieren, vorgeschlagen und untersucht. Obwohl deutliche Fortschritte bei der komplexitätstheoretischen Klassifizierung des Problems und der Entwicklung effizienter Algorithmen für das Problem gemacht wurden, stellt sich heraus, dass in der Praxis für viele Graphen die Lösung ein uninteressanter und wenig aussagekräftiger Subgraph ist.

Diese Arbeit untersucht daher eine Verallgemeinerung namens $(2,t)$-Club unter komplexitätstheoretischen und algorithmischen Gesichtspunkten, welche sinnvollere Ergebnisse liefert. Beim $(2,t)$-Club-Problem fragen wir nicht nur nach einem Subgraphen mit Durchmesser zwei, sondern verlangen ferner, dass alle Paare von nicht-adjazenten Knoten mindestens t gemeinsame Nachbarn haben. Dies vermeidet, dass der Status einer Lösung durch einen einzigen Akteur dominiert wird, was zu robusteren Graphen vom Durchmesser zwei als Lösungen führt.

Für dieses generalisierte Problem wird eine Vielzahl von theoretischen Ergebnissen präsentiert, sowie ein Algorithmus welcher das Problem schnell auf realen Graphen lösen kann. Wir zeigen die NP-Vollständigkeit des Problems auf allgemeinen Graphen, sowie Ergebnisse für verschiedene spezielle Graphklassen und strukturelle Parameter. Viele der Ergebnisse haben Bezug zu bekannten 2-Club-Ergebnissen, was die Gemeinsamkeiten beider Probleme unterstreicht. Ferner präsentieren wir experimentelle Ergebnisse bezüglich des Laufzeitverhaltens unseres Algorithmus sowie der Struktur gefundener Lösungen für $(2,t)$-Club auf realen Graphen.

Unsere Arbeit zeigt, dass $(2,t)$-Club sowohl eine sinnvolle Erweiterung des originalen Modells ist, als auch auf vielen Graphen in der Praxis gut gelöst werden kann.

Die Implementierung unseres Algorithmus, welche diese Arbeit begleitet, kann von der folgenden Adresse heruntergeladen werden:
http://fpt.akt.tu-berlin.de/two_t_club/

Abstract

The NP-complete 2-Club problem is a well-studied computational problem with various applications, for example social network analysis. It asks to find the subgraph of diameter two with the largest number of vertices in a given graph. It has been proposed and studied as a model for cohesive subgroups in social networks, i.e. groups of actors interacting to such an extent as to suggest a special relation to each other. While significant progress has been made at classifying the complexity of 2-Club and presenting efficient algorithms to solve it, on most real world graphs the model only produces uninteresting and uninformative solutions.

This work sets out to analyze a generalization of the problem, called $(2,t)$-Club, from a complexity-theoretic and algorithmic viewpoint, which yields more useful solutions. In the $(2,t)$-Club problem we not only require the solution to have diameter two, but all non-adjacent pairs of vertices of the solution to have at least t common neighbors. This avoids that one actor alone determines the status of the entire solution, which leads to a more robust kind of diameter-two graphs as solutions.

For this generalized problem a number of theoretical results are presented, as well as a fast algorithm for solving the problem on real world graphs. We show NP-completeness for the problem on general graphs and provide results for various special graph classes as well as for structural parameterization using the parameterized complexity framework. Many of the results tie to earlier results regarding the 2-Club problem, outlining common links between both problems. We also provide experimental results regarding the performance of our algorithm as well as the structure of solutions for $(2,t)$-Club.

Our work shows that $(2,t)$-Club is both a useful extension of the original model as well as a tractable problem for many graphs in practice.

The implementation of our algorithm for solving $(2,t)$-Club accompanying this work can be downloaded from:
http://fpt.akt.tu-berlin.de/two_t_club/

Contents

Chapter 1

Introduction

This work deals with (2,t)-clubs or *robust 2-clubs*, a generalization of the existing notion of 2-clubs, which aims to provide a better model for cohesive subgroups in social networks and other graphs. In particular, we will explore the computational complexity of the problem of finding the largest such structure (with respect to number of vertices) in a graph, which we call the (2,t)-Club problem. In this introductory section we motivate our work, give a first glimpse at the problem at hand, discuss related work, and recall a number of definitions and concepts from graph theory and computational complexity theory used throughout this work.

1.1 Motivation

Among the variety of applications of graph theory to real world problems social network analysis is of particular interest today. One important task in social network analysis is to find large cohesive subgroups, where each actor has a close relationship to the other actors in the subgroup. There is a number of different models for cohesive subgroups in graphs, including cliques, s-cliques and s-plexes, which we will briefly review in the following section. One model are the so-called s-clubs, subgraphs which have diameter at most s. The problem of finding maximum s-clubs in a graph has been studied in a variety of papers and its computational complexity is by now relatively well-understood. However, as noted by Hartung et al. [Har+15b] s-clubs and in particular 2-clubs can exhibit a certain lack of robustness, which makes their usefulness as a model for cohesive subgroups in social networks questionable. To be more precise, by the definition of s-clubs, it is enough for any two vertices in a s-club to be connected by *just one* path of length at most s. This can lead to s-clubs just being star-like structures in a graph, where all vertices are connected to each other via a central hub vertex. For instance a maximum degree vertex of the input graph and all of its neighbors form a 2-club. Indeed, Hartung et al. [Har+15b] found that

on the majority of real-world instances considered by them, the maximum 2-club is just the maximum degree vertex and its neighbors. This calls the meaningfulness of such "communities" in question. One attempt at fixing this deficiency, considering only edges of sufficient weight[1], did not yield better results. Another approach to fix the deficiency of maximum 2-clubs on real world graphs is to mandate every pair of non-adjacent vertices in a 2-club to have multiple neighbors, instead of just one as would be required by standard 2-clubs (due to the diameter two condition). Therefore this work deals with this variant of 2-clubs, called $(2,t)$-clubs, which are more robust by definition: In a standard 2-club, the existence of a single dominating vertex, providing a length-two path for all pairs of other vertices, can decide about the status of being a 2-club alone, regardless of the connections between the remaining vertices. And removing this central "hub vertex" might leave us with any arbitrary graph, which might even be far from being connected. In contrast, by demanding at least t common neighbors for all pairs of non-adjacent vertices, there are at least t vertices that must be removed to destroy the diameter-two property. Therefore, $(2,t)$-clubs (or *robust 2-clubs*) seem to provide a better model for cohesive subgroups in graphs, less prone to producing uninformative solutions on real world data.

1.2 s-Club and Other Clique Relaxations

Before we introduce $(2,t)$-clubs and the associated $(2,t)$-Club problem, we want to give a brief overview over various models which have been proposed to find cohesive subgroups in graphs. The most immediate model for a cohesive subgroup is a *clique*, i.e. a subset of vertices where all vertices are adjacent to each other. However, often this model seems too restrictive and one would identify significantly less strongly related groups as cohesive subgroups. Therefore a number of clique relaxation models have been proposed to provide for less strict but still meaningful definitions of a cohesive subgroup. Pattillo et al. [Pat+13] give a concise overview over such clique relaxation models. A pioneering work in applying graph-theoretic approaches to describe "sociometric cliques" (i.e. cohesive subgroups in a social network) was given by Alba [Alb73]. Mokken [Mok79] later built upon and refined this work and gave the concept called *sociometric cliques of diameter s* by Alba [Alb73] its current name, s-clubs.[2] We briefly list some of the clique relaxation models here, before we return a little more in detail to s-clubs. For details on the following clique relaxation models we again refer

[1] Amongst others Hartung et al. worked with the DBLP coauthor graph, where edges represent co-authorship of two authors and restricted the graph to co-authorships, where the two authors have authored at least $i > 1$ papers together.

[2] The name of the parameter s sometimes differs, e.g. s-clubs are also known as k-clubs. As the letter k usually holds a special meaning in parameterized complexity (for the parameter solution size), we prefer to call the parameter s to avoid confusion.

to Pattillo et al. [Pat+13].

s-clubs are graphs of diameter at most s. As cliques are the graphs of diameter one, s-clubs are a distance-based relaxation of cliques.

s-cores are graphs of minimum degree s. As a model s-cores have the advantage of being easily computable in polynomial time, but can consist of multiple dense clusters which are only loosely connected to each other.

γ-quasi-cliques are graphs of edge density at least γ.

s-plexes are graphs where every vertex has at most s other vertices to which it is *not* adjacent.

s-cliques (not to be confused with cliques of size s) are vertex subsets of a graph G where all vertices are within distance s to each other in G. This definition has some similarities to s-clubs, however, s-cliques are not solely defined by the subgraph they form, but also their relationship to the other vertices in the original input graph (which might provide the short paths between the vertices in the s-clique).

An advantage of s-clubs is that the model scales relatively well for real world social networks of varying size. Where edge density (as used in the definition of quasi-cliques) can be expected to drop considerably when increasing the size of a subgraph in a social network and degrees of vertices (as used in the definition of s-cores and s-plexes) can vary wildly, the famous observation by Milgram [Mil67] regarding the small world phenomenon shows that social networks usually have rather small diameter regardless of size. This usually also means that 2-clubs and 3-clubs tend to be the only s-clubs of interest in analyzing social networks, as already $s = 4$ can easily include major portions of the input graph. Furthermore, s-clubs are by definition connected, which seems like a desirable property for cohesive subgroups. Therefore, s-clubs seem to provide the most reliable model for cohesive subgroups in social networks. The possible deficiency of s-clubs is, however, that by definition nothing precludes a single vertex from providing the required short path for all pairs of other vertices in the s-club. As already mentioned, this usually is the case when looking for the largest 2-club in a (sparse) real world graph like social networks. Insofar 2-clubs (and s-clubs in general) can still lack in terms of cohesiveness or robustness and one might want to consider a variant of the problem dealing with this deficiency. This is where $(2,t)$-clubs come in.

1.3 The (2,t)-Club Problem

If we look at the definition of s-clubs, we easily notice that for $s = 2$ the problem of finding the largest diameter-two subgraph can also be stated as looking for the largest subgraph where any pair of non-adjacent vertices has at least one common neighbor. This directly leads to a possible generalization of 2-clubs, if we require t, instead of just one, common neighbors for all pairs of non-adjacent vertices. By doing so we can state the problem (2,t)-Club of finding the (2,t)-club with the most vertices in a graph as follows:

(2,t)-Club
Input: A graph $G = (V, E)$ and positive integers t and k.
Question: Is there a subset $S \subseteq V$ with $|S| \geq k$, such that in the induced subgraph $G[S]$ every pair of vertices is either adjacent or shares at least t common neighbors?

We stated the problem in its decision version, but the optimization version can be derived easily from binary search over k. (2,1)-Club is just the same as 2-Club and it is apparent that every (2,t)-club is also a (2,t')-club for every $t' \leq t$. Often we will just assume t to be a constant, but we will also consider the problem for arbitrary t where this would make a difference.

One could of course also formulate a more general (s,t)-Club problem, where we require each pair of non-adjacent vertices to have at least t vertex-disjoint paths of length at most s between them. In this work we will focus on the most basic variant of robust s-clubs, that is, robust 2-clubs or (2,t)-clubs.

1.4 Related Work

The generalization of 2-Club to (2,t)-Club, as presented in this work, is not entirely new: Veremyev and Boginski [VB12] have studied a very similar generalization which they called *R-robust s-clubs*, including the special case of *R-robust 2-clubs*, where R plays the same role as the parameter t in our (2,t)-Club problem. In contrast to our definition, they however require all vertices in a R-robust 2-club to have at least R common neighbors. This is motivated by their use-case of finding networks with robustness against external attacks: R-robust s-clubs are robust both in terms of vertex deletions as well as edge deletions, meaning that the removal of any vertex or edge will still leave us with a $(R-1)$-robust s-club. Our model is only robust with regard to vertex deletions, as an edge deletion might increase the diameter of a (2,t)-club by one and does not guarantee a (2,$t-1$)-club. However, in our use-case of finding cohesive subgroups in social networks the

definition of R-robust 2-clubs seems to restrictive and unnecessary. Veremyev and Boginski [VB12] have shown a number of robustness properties for R-robust s-clubs and introduced an integer linear programming formulation for the problem. (2,t)-clubs also have a strong connection to strongly regular graphs, insofar that every strongly regular graph with parameters (v, k, λ, μ) is a (2,μ)-club.

This work rests strongly on previous work by other authors, as many results for 2-Club transfer rather easily to (2,t)-Club. In particular, the work of Hartung et al. [Har+15a] provides many of these results. The algorithm for solving the (2,t)-Club problem, which we have implemented as part of this work, is itself a generalization and refinement of the algorithm provided by Hartung et al. [Har+15b]. Some of our approaches of studying the (2,t)-Club problem have also been followed by other authors for the 2-Club or s-Club problem. In particular, Golovach et al. [Gol+14] have studied s-Club on various special graph classes, an approach to which we also dedicate significant portions of this work. Another work which pointed us to some of our results is the thesis by Schäfer [Sch09].

As our problem is a generalization of 2-Club, we conclude this section with the following overview over known results for 2-Club and s-Club.

Known results regarding 2-Club and s-Club

Bourjolly et al. [Bou+02] have shown s-Club to be NP-complete. Asahiro et al. [Asa+10] have shown that s-Club cannot be approximated in polynomial time within a factor of $O(n^{1/2-\epsilon})$ for any $\epsilon > 0$. They could however provide a factor $O(n^{1/2})$ approximation for even s and a factor $O(n^{2/3})$ approximation for odd s. Various authors have studied s-Club on special graph classes. Schäfer [Sch09] has shown that s-Club is polynomial-time solvable on trees, interval graphs and planar graphs, while 2-Club is polynomial-time solvable on bipartite graphs. Asahiro et al. [Asa+10] have shown NP-hardness for split graphs and polynomial-time solvability for odd s on chordal graphs. Golovach et al. [Gol+14] have proven polynomial-time solvability on chordal bipartite, strongly chordal, AT-free and distance-hereditary graphs. For 4-chordal graphs they could show NP-hardness, while on weakly chordal graphs the problem is polynomial-time solvable for odd s only.

Hartung et al. [Har+15a] have provided numerous results for structural parameterization of 2-Club: Fixed-parameter tractability for distance to cluster and co-cluster graphs, W[1]-hardness for h-index and NP-hardness for constant clique cover number, domination number, degeneracy, average degree and distance to bipartite graphs. In another work Hartung et al. [Har+15b] have shown fixed-parameter tractability with regard to size of a vertex cover, size of a cluster editing set and size of a feedback edge set. Schäfer [Sch09] has shown fixed-parameter tractability for treewidth and Schäfer et al. [Sch+12] have shown fixed-parameter tractability for maxi-

mum degree and distance to 2-club.

Numerous algorithms have been proposed and tested for solving 2-Club and s-Club, including [Bou+02; Cha+13; Har+15b; SB13; Wot14]. Integer linear programming approaches have been studied by Bourjolly et al. [Bou+02], Balasundaram et al. [Bal+05] and Carvalho and Almeida [CA11].

1.5 Preliminaries

In this section we introduce some basic concepts from graph theory and parameterized complexity theory used throughout this work. These introductions will be brief and mainly serve the purpose to help the reader recall important definitions and concepts and avoid confusion over use of notation and names. Particularly regarding graph-theoretic concepts we assume a basic familiarity of the reader with the terms and concepts used in this work. There are many introductory texts on graph theory, for example Diestel [Die12]. More detailed and formal introductions into the topic of parameterized complexity are given in the monographs by Niedermeier [Nie06], Downey and Fellows [DF13] and Flum and Grohe [FG06].

We assume that the reader is familiar with basic concepts from computational complexity theory, especially the use of *polynomial-time reductions* to prove NP-hardness and *Landau notation* (also commonly known as *Big O notation*). As a less common variant of the standard Landau notation, we also use the O^*-notation which also hides factors polynomial in the input size, e.g. $(2^n \cdot n^2) \in O^*(2^n)$. We assume familiarity with basic algorithmic techniques as for example the use of search trees[3] to solve hard computational problems.

1.5.1 Graph-Theoretic Preliminaries

We only consider finite simple graphs, i.e. undirected graphs without multiple edges or loops. A graph $G = (V, E)$ is defined by its set of *vertices* V and its set of *edges* E which is a subset of the set of all unordered pairs of elements of V. Vertices are also called *nodes* in some contexts. If we are given a graph G without explicitly named vertex set or edge set, $V(G)$ refers to the vertex set of G and $E(G)$ to its edge set. The two vertices connected by an edge are called its *endpoints*. Two vertices connected by an edge are called *adjacent* to each other, or also: neighbors. An edge and an endpoint of this edge are called *incident* to each other. We also call two edges which share an endpoint incident to each other. The *degree* of a vertex is the number of other vertices adjacent to it. A (simple) *path* is a sequence

[3]Not to be confused with the data structure used to store values sorted of the same name. When we talk about search trees in this work we *always* mean the tree associated with a branching algorithm exploring some search space for the problem at hand.

of vertices such that no vertex occurs twice and any two successive vertices in the path are adjacent. The *length of a path* is the number of edges along it. A (simple) *cycle* is a path where the first and last (but no other) vertices are the same. A graph without cycles is called *acyclic*. The *distance* $d(u, v)$ between two vertices u and v is defined as the length of a shortest path between these two vertices. If there is a path between two vertices those vertices are said to be *connected*, and *disconnected* otherwise. A *connected graph* has no pairs of disconnected vertices. A *connected component* of a graph is a maximal set of vertices which are pairwise connected. Unless stated otherwise, we assume that graphs are connected, as (2,t)-Club can be solved on disconnected graphs by solving the problem for each connected component. The *diameter* of a graph is the length of the longest shortest path in the graph, i.e. the maximum distance between any two vertices in the graph. The *open d-neighborhood* $N_d(v)$ of a vertex v is the set of all vertices within distance d of v except v itself. The *closed d-neighborhood* $N_d[v]$ is defined as $N_d[v] := N_d(v) \cup \{v\}$. We use n and m throughout this work to denote the cardinality of V and E, that is, $n := |V|$ and $m := |E|$, particularly in the context of running time bounds.

A *subgraph* (V', E') of a graph $G = (V, E)$ is a graph with $V' \subseteq V$ and $E' \subseteq E$ such that all edges in E' are between vertices in V', i.e. a graph derived from G by deleting vertices and edges. An *induced subgraph* (V', E') is a subgraph where for any pair of vertices $v, w \in E'$ it holds that $\{v, w\} \in E' \iff \{v, w\} \in E$, that is an induced subgraph retains all edges between its vertices present in the original graph. For a graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices $G[S]$ denotes the subgraph induced by the vertices in S, that is the unique induced subgraph of G with vertex set S. A *graph class* is a complete (and usually not finite) set of graphs which are defined by some common property or characteristic, for example the class of *forests* which are defined by containing no cycles. Some basic graph classes used in this work are:

- Trees, which are connected forests.
- Complete graphs, which are graphs where all edges are present. The unique complete graph on n vertices is called K_n.
- Bipartite graphs, which is the class of 2-colorable graphs, that is graphs whose vertex set can be partitioned into two sets U and V such that every edge is between one endpoint in U and one endpoint in V.
- Complete bipartite graphs, which are bipartite graphs where all edges between U and V are present. The unique complete bipartite graph with n vertices on one side of the partition and k on the other is called $K_{n,k}$.

For the definition of other graph classes used in this work we refer the reader to Brandstädt et al. [Bra+99] or to www.graphclasses.org. A graph has *distance d to a graph class C* if by deleting at most d vertices it can be transformed into a graph from class C.

We call any subset $C \subseteq V$ of vertices of a graph a *clique*, if all vertices from C are adjacent to each other, that is C induces a complete subgraph. Note that we do not require the set C to be maximal in order to call it a clique. Likewise, a *biclique* is a subset of vertices inducing a (not necessarily maximal) complete bipartite subgraph.

P_i denotes the path on i vertices, i.e. of length $i-1$. C_i the cycle with i vertices, which we also call i-cycle. The complement $\overline{G} = (V, E')$ of a graph $G = (V, E)$ is defined by $\forall v, w \in V \,.\, \{v, w\} \in E' \iff \{v, w\} \notin E$. The vertex connectivity of a graph is the minimum number of vertices that has to be removed to disconnect the graph. Edge connectivity is defined the same way over edges. The edge density of a graph is equal to $m/\binom{n}{2}$, i.e. the fraction of edges present compared to the maximum possible number of edges in a graph of n vertices.

1.5.2 Parameterized Complexity

In this subsection we give a brief overview over the main ideas of parameterized complexity theory. We avoid the usual more formal definitions in favor of an intuitive approach which should be sufficient for understanding the following work. For details we again refer to the monographs [DF13; FG06; Nie06].

Classic complexity theory classifies computational problems according to their inherent complexity based on the single parameter input size n. While input size is arguably the most natural and useful parameter when analyzing computational problems, considering additional parameters can lead to a much more fine-grained analysis of complexity. A parameterized problem in parameterized complexity now simply is a computational problem where we consider at least one other parameter besides input size. A parameter can be any meaningful measure or number describing the input instance, for graph problems we might for instance consider diameter, maximum degree, treewidth, the size of a minimum vertex cover of the graph and many others. The most common parameter is the size k of the sought solution, e.g. the allowed number of vertices in a solution for Vertex Cover.[4] Parameterization does not change the nature of the problem itself, but might change our outlook on it, for example when devising algorithms to solve the problem. For example, there are many computational problems known to be

[4] To elaborate, we consider the decision version of Vertex Cover. That is, we do not ask for a minimum vertex cover of the given graph but an answer to the question whether the graph contains some vertex cover of size at most k.

NP-hard which behave very well if a certain parameter is kept small, even on very large inputs. While NP-hard problems are suspected to always require super-polynomial time with respect to input size n only, considering some other parameter in addition to input size often results in running times only super-polynomial in the parameter but polynomial (maybe even linear) in input size. If this parameter is then fixed, this results in polynomial running time with respect to input size, where the degree of the polynomial *does not* depend on k.[5]

For example, Vertex Cover parameterized by the maximum size k of a solution provides a simple argument to see that it can be solved in time $2^k \cdot n^{O(1)}$: For every edge $\{u, v\}$ in the input graph we have to take either u or v into the vertex cover. Thus, if we repeatedly branch over the endpoints of uncovered edges we arrive at a binary search tree with depth at most k as no solution is allowed to contain more than k vertices.

A parameterized problem which is solvable in time polynomial in the input size but superpolynomial in some other parameter(s) is called *fixed-parameter tractable* with regard to these parameter(s). One and the same computational problem can turn out fixed-parameter tractable for some parameters but fixed-parameter intractable for others. We will now formalize the notion of fixed-parameter tractability a bit.

Definition 1.1. A problem is called fixed-parameter tractable with regard to some parameter k if it can be solved in running time $f(k) \cdot poly(n)$, where f is a computable function depending solely on k and $poly(n)$ a polynomial in the input size n.[6] A parameterized problem is in the complexity class FPT[7] if it is fixed-parameter tractable with regard to its parameter(s).

The basic classes of presumed fixed-parameter intractability are W[1] and W[2], with W[1]⊆W[2]. The exact definition of the hierarchy W[t] of classes these two are from is neither necessary nor especially enlightening for the purposes of this work. The important fact is that it is presumed that FPT ≠ W[1], i.e. W[1]-hard problems are not fixed-parameter tractable. FPT and W[1] therefore have a similar relation to each other with regard to fixed-parameter tractability as P and NP have with regard to polynomial-time solvability in classic complexity theory. In fact the basic proof technique for fixed-parameter intractability, parameterized reductions or *fpt-reductions*, are very similar to polynomial-time reductions used in NP-hardness proofs. To prove the fixed-parameter intractability of some parameterized problem

[5] Although the constants hidden in the resulting running time might be very large.

[6] Given multiple parameters $k_1, ..., k_i$, f may depend on all of these but only these.

[7] We will frequently use "FPT" just as a shorthand for "fixed-parameter tractable", without meaning the complexity class per se.

A with regard to some parameter k we reduce some parameterized problem B known to be W[1]-hard with regard to some parameter k' to A. The reduction must be computable in time $f(k') \cdot poly(n')$, where n' is the input size for problem B, and for every generated instance of A the parameter k must be bounded by some computable function $g(k')$ of the parameter k' of the original instance of B. We of course also require the notion of correctness known from classic complexity theory from the reduction, i.e. that the reduction maps every instance to an equivalent instance. Fixed-parameter tractability with regard to some parameter k can be proven by describing an algorithm with running time $f(k) \cdot poly(n)$ for the problem. There are various common approaches to arrive at such an algorithm, including kernelization, bounded search trees and dynamic programming.

To summarize, the theory of parameterized complexity is a tool for fine-grained complexity analysis of computational problems. Its practical application is to devise algorithms for NP-hard problems where the superpolynomial (usually exponential) growth of the running time is restricted to the parameter. Given small enough values of the parameter in question the problem then might turn out tractable despite being NP-hard. One might liken the approach of parameterization to other approaches trying to identify easy special cases of NP-hard problems, like restricting NP-hard graph problems to special graph classes.

1.6 Overview

In this section we give a short overview over the structure of the remaining work by giving short descriptions of the contents of the following chapters.

In Chapter 2 we will have a first more detailed look at the $(2,t)$-Club problem, discussing some basic structural properties of $(2,t)$-clubs, showing NP-completeness and considering the basic parameterizations by solution size k and the dual parameter $n - k$. We will prove W[1]-containment for the problem with regard to k and provide a simple algorithm for the dual parameter $n - k$, which we prove to be asymptotically optimal under the assumption of a reasonable complexity-theoretic hypothesis.

In Chapter 3 we will study the problem on special graph classes in light of its NP-completeness on general graphs. We will show polynomial-time solvability on interval graphs, cographs, co-cluster graphs, threshold graphs, planar graphs and bipartite graphs, although the latter case is only polynomial-time solvable for fixed t. For all of these graph classes except planar graphs we provide enough detail to make readily implementable algorithms with

concrete running time upper bounds. Additionally we will prove polynomial-time solvability for $t = 2$ on co-bipartite graphs and fixed-parameter tractability with regard to solution size k on split graphs.

In Chapter 4 we give some results for structural parameterization of $(2,t)$-Club, which together with results transferring from 2-Club (as obtained by Hartung et al. [Har+15a]) classify the parameterized complexity of $(2,t)$-Club for a wide range of parameters. We also introduce a (to our best knowledge) new parameter derived from the *Hirsch index* which is of interest in the context of $(2,t)$-Club and 2-Club, based on algorithmic considerations to exploit the fact that all $(2,t)$-clubs have diameter at most two.

In Chapter 5 we present an overview of an algorithm for $(2,t)$-Club we have implemented. We then proceed with various experiments showing that the implemented algorithm has good performance on real world graph instances and that the more general $(2,t)$-Club problem likely provides better results in the context of identifying cohesive subgroups in real world social networks.

In Chapter 6 we will give an outlook and conclusion, noting open problems not adressed by this work and giving pointers to possible further research. We will return briefly to the more general (s,t)-Club and also discuss some other variants of the problem which might be of interest.

In an Appendix we present a list of further experimental data.

Chapter 2

The (2,t)-Club Problem

In this chapter we study the (2,t)-Club problem by first looking at structural properties of (2,t)-clubs before giving an easy reduction from 2-Club which proves the NP-hardness of the problem. We will further study the parameterized complexity of (2,t)-Club when parameterizing it by the solution size k and when parameterizing it by the dual parameter $n - k$. For the parameter k we were not able to decide whether (2,t)-Club is FPT or W[1]-hard, but we can show W[1]-containment with regard to k. For the dual parameter $n - k$ a simple search tree algorithm can achieve a running time of $O^*(2^k)$, which can be shown to be asymptotically optimal under a reasonable complexity theoretic assumption. As a reminder, the (2,t)-Club problem can be stated as follows:

(2,t)-Club
Input: A graph $G = (V, E)$ and positive integers t and k.
Question: Is there a subset $S \subseteq V$ with $|S| \geq k$, such that in the induced subgraph $G[S]$ every pair of vertices is either adjacent or has at least t common neighbors?

For the remainder of this work we call two vertices v and w *agreeable* (with regard to (2,t)-Club), if and only if v and w are adjacent or have at least t common neighbors. In many cases we will consider t to be a constant instead of an input parameter. A graph G is a (2,t)-club, if every pair of vertices in G is agreeable. By '(2,t)-club' we refer interchangeably to graphs which are (2,t)-clubs as well as vertex subsets of a graph which induce a (2,t)-club. It is important to note, that two vertices v and w being agreeable, with regard to (2,t)-Club, in a graph G does not necessarily mean that there is some (2,t)-club S in G containing both v and w. For example, two non-adjacent vertices v and w might have t or more common neighbors, making them agreeable by our definition, but there is no subset of these neighbors of size at least t, where all pairs of vertices are agreeable to each other.

2.1 Structural Properties of (2,t)-Clubs

Before studying the (2,t)-Club problem itself, we derive some structural properties of (2,t)-clubs. As noted before in Section 1.3, every (2,t)-club with $t \geq t'$ is also a (2,t')-club by definition. Therefore, we will prove some properties for (2,2)-clubs, but they hold for all $t \geq 2$. We will start with three simple observations concerning small (2,t)-clubs and those with low degree vertices.

Observation 2.1. *A (2,t)-club S with at most $t+1$ vertices is a clique.*

Proof. For any pair of vertices of S, there are only $t-1$ further vertices in S which can be potential common neighbors for both. Therefore two vertices in S can only be agreeable if they are adjacent. Thus, S is a clique. □

Observation 2.2. *A (2,t)-club S containing a vertex of degree less than t is a clique.*

Proof. Consider a vertex of degree less than t in S. As S is a (2,t)-club, v must be agreeable to all other vertices in S. But v has less than t neighbors, therefore v can only be agreeable to vertices with which it is adjacent. Consequently, v must be adjacent to all other vertices in S and therefore S contains at most t vertices. From Observation 2.1 it follows that S must be a clique. □

Definition 2.3. We call a graph G with n vertices, which can be turned into a $K_{t,(n-t)}$ (a complete bipartite graph with t vertices on one side and $n-t$ vertices on the other) by edge deletions, a **t-diamond**.

Observation 2.4. *A (2,t)-club S containing a vertex v of degree exactly t is a t-diamond.*

Proof. If there is a vertex v of degree exactly t, all vertices in $S \backslash N(v)$ must be adjacent to all vertices from $N(v)$ as any vertex w which is not adjacent to v must be adjacent to all of $N(v)$ in order to be agreeable to v. Therefore, S is a t-diamond. □

The next observation deals with the connectivity of (2,t)-clubs.

Observation 2.5. *Every (2,t)-club has vertex-connectivity at least t and consequently edge-connectivity at least t.*

Proof. A graph is disconnected if for some pair of vertices in it there exists no path between the two vertices. After deleting vertices from a graph G only pairs of vertices which are not adjacent in G can have no path between them. As every pair of non-adjacent vertices in a (2,t)-club has at least t common neighbors, we have to delete at least t vertices from a (2,t)-club in order to disconnect it. Edge-connectivity of a graph is always lower-bounded by its vertex-connectivity. □

These bounds on connectivity are sharp. The $K_{t,x}$ for any $x \geq t$ is a (2,t)-club and can be disconnected by either deleting the t vertices on the side with cardinality t or removing all t edges incident to a vertex of the side with cardinality x.

Observation 2.6. *Every vertex and every edge of a (2,2)-club of size at least four is part of at least one 4-cycle. Furthermore, any two vertices v and w in a (2,2)-club of size at least four are part of some common 4-cycle, which implies that every vertex in a (2,2)-club of size $k \geq 4$ lies on at least $\frac{k-1}{3}$ different 4-cycles.*

Below, We will prove the generalization of this observation for (2,t)-clubs in Observation 2.8. The observation implies that there are no non-trivial acyclic (2,2)-clubs, making the (2,2)-Club problem trivial on acyclic graphs, as any edge constitutes a maximum size (2,2)-club.

Observation 2.7. *For any induced P_3 with endpoints u and v in a 2,2-club S of size ≥ 3, there is some vertex w such that $\{u, w\} \in E(G[S])$ and $\{v, w\} \in E(G[S])$.*

Proof. The endpoints of an induced P_3 are not adjacent, therefore they must have at least two common neighbors and the observation follows. □

These results also hold for (2,t)-clubs with $t > 2$, as (2,t)-clubs for $t > 2$ are also (2,2)-clubs, but the first observation can be strengthened for those.

Observation 2.8. *Every vertex and every edge of a (2,t)-club of size at least $t + 2$ is part of at least one $K_{2,t}$. Furthermore, any two vertices v and w in a (2,t)-club of size at least $t + 2$ are part of some common $K_{2,t}$, which implies that every vertex in a (2,t)-club of size $k \geq t + 2$ is part of at least $\frac{k-1}{t+1}$ different $K_{2,t}$.*

Proof. Non-adjacent vertices are required to have t common neighbors, so the observation that they are part of a common $K_{2,t}$ holds trivially for them. For adjacent vertices v and w, pick any vertex u which is adjacent to exactly one of them. Assume without loss of generality that vertex u is adjacent to v, but not to w. Now u and w must have at least t common neighbors and their common neighbors include v, therefore the observation holds. If no such vertex u exists, v and w must have at least t common neighbors, as any vertex x not adjacent to v and w needs at least t common neighbors with both of them. If no such vertex x exists, v and w are both adjacent to all remaining $k - 2 \geq t$ vertices. So the observation holds for adjacent vertices and therefore any edge in a (2,t)-club is also part of at least one $K_{2,t}$. For the last part of the observation, note that any $K_{2,t}$ contains exactly $t + 2$ vertices. Furthermore we have just shown that every vertex v in a (2,t)-club S is part of a common $K_{2,t}$ with each vertex in $S \backslash \{v\}$. As there are $k - 1$

other vertices in $S \backslash \{v\}$ and any given $K_{2,t}$ can contain at most $t+1$ vertices besides the vertex v we are considering, every vertex of a (2,t)-club is part of at least $\frac{k-1}{t+1}$ different $K_{2,t}$. □

(2,t)-clubs cannot be characterized by forbidden subgraphs or forbidden induced subgraphs, which can be easily seen by observing that any graph G can be extended to a (2,t)-club by introducing t new vertices, which are all adjacent to all original vertices.

2.2 NP-Hardness of (2,t)-Club

As a first step towards examining the computational complexity of the (2,t)-Club problem, we will now show that it is NP-complete.

Theorem 2.9. *(2,t)-Club is NP-complete.*

Proof. Whether a subset $S \subseteq V$ of vertices of a graph $G = (V, E)$ constitutes a (2,t)-club can be easily checked in polynomial time by checking for every one of the at most $O(n^2)$ vertex pairs in S, whether they are adjacent or have at least t common neighbors. This requires no more than $O(\Delta^2)$ time per vertex pair, where Δ is the maximum degree of the graph. Therefore, (2,t)-Club is in NP.

The NP-hardness of (2,t)-Club for any t can be shown via a simple reduction from the 2-Club problem, which was shown to be NP-complete by Bourjolly et al. [Bou+02]. For any 2-Club instance (G, k) we construct an equivalent (2,t)-Club instance $(G', k + t - 1)$, extending $G = (V, E)$ to $G' = (V', E')$ by adding $t - 1$ vertices $v_1^*, ..., v_{t-1}^*$ which are adjacent to all original vertices of G and to each other. This can obviously be done in polynomial time, as adding the new vertices and the edges between them requires constant time for constant t, while adding the required edges to the original vertices requires $O(n)$ time. Now G has a 2-club of size k if and only if G' has a (2,t)-club of size $k+t-1$. Any 2-club S in G corresponds to a (2,t)-club $S \cup V^*$ in G', where $V^* = V' \setminus V$, as the vertices in V^* introduce $t - 1$ paths of length 2 for every vertex pair in S and every non-adjacent vertex pair in S already has at least one length-2 path in S due to S being a 2-club. Furthermore every maximal (2,t)-club in G' must contain all of V^*, because the vertices in V^* are adjacent and thus agreeable to all vertices of G', and any (2,t)-club S containing all of V^* must fulfill the property that $S \setminus V^*$ is a 2-club, because every pair of non-adjacent vertices in $S \setminus V^*$ still requires one more path of length two (beyond the $t - 1$ provided by V^*) between them. Therefore we have a polynomial-time reduction from 2-Club to (2,t)-Club and (2,t)-Club is NP-hard. □

Apart from being simple the above reduction has at least one other major advantage: Many structural graph parameters remain unchanged or

mostly unchanged (up to some additive constant) by the transformation. Therefore many hardness results for structural parameterization on 2-Club, as for instance those presented by Hartung et al. [Har+15a], can be easily transferred to $(2,t)$-Club:

Corollary 2.10. *$(2,t)$-Club is still NP-hard on graphs ...*

1. *... with diameter 2 and domination number 1.*
2. *... with clique cover number 3.*
3. *... with distance t to bipartite graphs.*
4. *... with average degree at most α, for any constant $\alpha > 2t$, which are connected.*
5. *... with degeneracy $5 + t$.*

Furthermore $(2,t)$-Club is W[1]-hard when parameterized by the h-index of the input graph.

Proof. All of these follow from the above reduction from 2-Club to $(2,t)$-Club and the hardness results for structural parameterization of 2-Club by Hartung et al. [Har+15a], where all following claims about 2-Club were proven. For (1), note that all graphs G' constructed by the reduction have at least one vertex adjacent to all other vertices and thus diameter 2 and domination number 1. For (2), observe that the reduction does not change the clique cover number of the original graph as all of the vertices in V^* can be joined with any clique in G but do not join any two different cliques of G to a larger clique. 2-Club is NP-hard on graphs with clique cover number 3, therefore $(2,t)$-Club is too. Regarding (3), note that adding $t-1$ vertices to any graph can increase the distance (measured in vertex deletions) to any given graph class by at most $t-1$. 2-Club is NP-hard on graphs with distance 1 to bipartite graphs, therefore $(2,t)$-Club is NP-hard on graphs with distance t to bipartite graphs. Regarding (4), 2-Club is NP-hard on connected graphs with average degree at most α, where α is any constant > 2. Note that the reduction introduces exactly $n(t-1) + \frac{(t-1)(t-2)}{2}$ new edges, therefore the degree sum increases by $2n(t-1) + (t-1)(t-2)$. The second term vanishes when n approaches ∞, so the average degree increases by $2t-2$ because of the reduction and $(2,t)$-Club must be NP-hard on connected graphs with average degree α for any constant $\alpha > 2t$. Note that this bound is only what follows directly from the reduction, we will show later that $(2,t)$-Club is already NP-hard for even smaller average degree. For (5) Hartung et al. have shown that 2-Club is NP-hard on graphs with degeneracy six. Adding $t-1$ vertices seeing all other vertices increases degeneracy by at most $t-1$, therefore $(2,t)$-Club must be NP-hard on graphs with degeneracy $5+t$. For the last claim observe that the h-index of

a graph can increase by at most $t-1$ when adding $t-1$ vertices. 2-Club parameterized by h-index is W[1]-hard, therefore (2,t)-Club is too for any constant t. □

In Chapter 4, we will briefly return to these results and discuss whether they are tight. As a last remark on the reduction from 2-Club to (2,t)-Club, note that if we have d vertices in a graph G which are all adjacent to all vertices in G, then the (2,t)-Club problem for a given constant t on G can be reduced to the (2,t-d)-Club problem on G', where G' is G minus the d aforementioned vertices. If d equals or exceeds t, G becomes trivially a (2,t)-club.

2.3 (2,t)-Club Parameterized by Solution Size

The most natural and common parameter for parameterizing a computational problem is the size k of a solution. For arbitrary t it is easy to see that (2,t)-Club must be W[1]-hard parameterized by k because Clique is W[1]-hard parameterized by k and both problems are equivalent for $t \geq n-1$. For if we set $t \geq n-1$, then solutions can only be cliques as there are not enough potential common neighbors for non-adjacent vertices. And likewise every clique is a (2,t)-club, therefore (2,t)-Club and Clique are equivalent for $t \geq n-1$. Consequently, we only need to consider t being a constant. Unfortunately, we do not know whether (2,t)-Club is FPT or W[1]-hard with regard to k in this case, but we can show W[1]-containment with regard to k. To show this we use the following characterization of W[1] due to Chen et al. [Che+05] in the formulation of Guo et al. [Guo+07].

Lemma 2.11. *(cited from Guo et al. [Guo+07]) For a parameterized problem P, we have $P \in$ W[1] if and only if there exist a computable function f, a polynomial $p(n)$, and a nondeterministic RAM program deciding P such that for every run of the program on an instance (x, k) (where $|x| = n$),*

1. *it performs at most $f(k) \cdot p(n)$ steps;*
2. *at most the first $f(k) \cdot p(n)$ registers are used;*
3. *at every point of the computation, no register contains numbers strictly greater than $f(k) \cdot p(n)$;*
4. *all nondeterministic steps are among the last $f(k)$ steps.*

Theorem 2.12. *(2,t)-*Club *parameterized by the solution size k is contained in W[1].*

Proof. It can be easily seen, that a nondeterministic RAM program solving (2,t)-Club parameterized by k exists, which obeys the restrictions of the above lemma. We require a data structure which allows us to check any two given vertices for adjacency in $O(1)$ time, for example an adjacency matrix. If the input graph is not given in such a data structure, we must first convert it into such a data structure in deterministic polynomial time. A polynomial number of additional deterministic steps at the beginning of the program does not violate any of the restrictions of Lemma 2.11 if the restrictions are obeyed otherwise. The program simply uses nondeterminism to guess a size k solution and then checks whether the guessed solution has the (2,t)-club property by testing each pair of vertices in the solution for agreeability. Checking a guessed solution for the (2,t)-club property requires only $f(k)$ time for some computable function f, as opposed to for instance the W[2]-hard Dominating Set problem checking the solution does not require to consider vertices outside of the guessed solution. Therefore the number of steps of the algorithm is bounded by $k + f(k)$. We only need to store which k vertices we have guessed, which pair of vertices of the solution we currently check for agreeability and up to k neighbors when checking a pair, which means that we only require a number of registers linear in k. As the only information we store are vertex IDs of the original graph, the numbers contained in registers are bounded by n. Therefore the proposed program obeys all restrictions of the above lemma and (2,t)-Club parameterized by k is contained in W[1]. □

2.4 (2,t)-Club Parameterized by the Dual Parameter n-k

Another common parameter for computational problems is the dual parameter $n - k$, which is, in the case of vertex deletion problems, the number of vertices which have to be deleted in order to obtain a solution. For (2,t)-Club parameterized by the dual parameter $n - k$ there is a simple search tree algorithm running in $O^*(2^{n-k})$ time, which can be shown to be asymptotically optimal under a reasonable complexity-theoretic assumption. (2,t)-Club parameterized by $n-k$ can also be viewed as the following vertex deletion problem:

(2,t)-Club Vertex Deletion
Input: A graph $G = (V, E)$ and positive integers t and k.
Question: Is there a subset $D \subseteq V$ with $|D| \leq k$ such that the induced subgraph $G[V \setminus D]$ is a (2,t)-club?

The parameter k of the $(2,t)$-Club Vertex Deletion problem corresponds to the dual parameter $n - k$ of the $(2,t)$-Club problem. We will now present a simple algorithm solving $(2,t)$-Club Vertex Deletion in time $O^*(2^k)$. The O^* notation hides factors which are polynomial in n, but the polynomial factor is reasonable for the given algorithm for $(2,t)$-Club Vertex Deletion.

Theorem 2.13. *$(2,t)$-Club Vertex Deletion can be solved in time $O^*(2^k)$, where k is the number of allowed vertex deletions.*

Proof. The algorithm works as follows: Start with the input graph G and the input parameter $k \geq 0$. If at any point G becomes a $(2,t)$-club, terminate the algorithm and return *yes*. If G is currently no $(2,t)$-club and $k > 0$, find two vertices v, w in G, which are not agreeable to each other, that is, v and w which are neither adjacent nor have at least t common neighbors. Now branch over the cases of deleting either v or w, that is proceed with $G - v$ and k reduced by one in the one branch and $G - w$ and k reduced by one in the other. The current value of k keeps track of the remaining allowed deletions. If k becomes zero and G is not a $(2,t)$-club yet, abort the current branch. This procedure leads to a search tree with at most 2^k leaves. We need to keep track of agreeability between all vertex pairs to select vertices for branching in each step (and also to check whether G already is a $(2,t)$-club). As vertex deletions can change agreeability between vertices, we need to update the agreeability information after each step. The particulars of testing G for being a $(2,t)$-club and updating agreeability determine the hidden polynomial factor in the running time. □

We will now prove that the given algorithm is asymptotically optimal. We do so by showing that there can be no algorithm with running time $O^*((2-\epsilon)^k)$ for any $\epsilon > 0$ for $(2,t)$-Club Vertex Deletion, unless the Strong Exponential Time Hypothesis (SETH) fails. The SETH conjectures that Cnf-Sat, the satisfiability problem for boolean formulas in conjunctive normal form, cannot be solved in time $O^*((2-\epsilon)^n)$ for any $\epsilon > 0$, where n is the number of variables in the formula. The proof we employ is a slight modification of the proof given by Hartung et al. [Har+15b] for showing that the s-Club Vertex Deletion problem admits no such algorithm.

Theorem 2.14. *The Strong Exponential Time Hypothesis (SETH) implies that there is no $O^*((2-\epsilon)^k)$ algorithm for any $\epsilon > 0$ for $(2,t)$-Club Vertex Deletion.*

Proof. We will present a construction transforming a CNF formula F with N variables in polynomial time into a graph G_F, which can be turned into a $(2,t)$-club by N vertex deletions if and only if F is satisfiable. Therefore an $O^*((2-\epsilon)^k)$ algorithm for $(2,t)$-Club Vertex Deletion would disprove the SETH. Without loss of generality we assume that no clause contains

the positive and negative literal of the same variable, as such tautological clauses can be simply omitted.

The construction works as follows: For each variable x of the formula introduce the *literal vertices* v_x and $v_{\bar{x}}$ corresponding to the positive and negative literal of that variable. For each clause C of the formula introduce the *clause vertices* v_{C_α} and v_{C_β} and $t-1$ *clause anchor vertices* v_{C_1} through $v_{C_{t-1}}$. For each clause C make both clause vertices v_{C_α} and v_{C_β} adjacent to all clause anchor vertices of the same clause and to all literal vertices corresponding to the literals occurring in C.

For every pair of vertices v and w from the set of all literal vertices and all clause vertices such that v and w are not the two literal vertices belonging to the same variable x and such that v and w are not the two clause vertices belonging to the same clause C, introduce t *path vertices* p_{vw_1} through p_{vw_t} and make them all adjacent to v and w. Finally make the set of vertices consisting of all path vertices and all clause anchor vertices a clique. The number of vertices of the constructed graph G_F is polynomial in the length of the formula F (measured in terms of literal occurences) and for each pair of vertices in G_F we can easily determine in polynomial time whether the two vertices should be adjacent or not according to above construction. Therefore the described transformation is polynomial-time.

Observe that any vertex v_x is not agreeable to the vertex $v_{\bar{x}}$ as they are not adjacent and have no common neighbors. Therefore, to obtain a $(2,t)$-club from G_F by vertex deletion we must delete at least one literal vertex for every variable, requiring all N vertex deletions. We now claim that deleting N literal vertices from G_F (each one for every variable of F) results in a $(2,t)$-club if and only if the remaining literal vertices represent a satisfying assignment for F. Note that path vertices and clause anchor vertices are agreeable to all vertices. These vertices themselves form a clique and every path vertex and every clause anchor vertex has at least t common neighbors with every literal vertex and every clause vertex, as every literal vertex and every clause vertex is adjacent to at least t path vertices. The remaining literal vertices (after deleting one literal vertex for each variable) are also agreeable to each other, as for every variable only one literal vertex remains and two literal vertices of different variables share t common neighbors among the path vertices by construction. Furthermore, the remaining literal vertices are also agreeable to all clause vertices due to t shared path vertex neighbors. Last any two clause vertices belonging to different clauses are agreeable due to t shared path vertex neighbors. The only candidates for non-agreeable vertices are therefore two clause vertices v_{C_α} and v_{C_β} belonging to the same clause C. Each such pair v_{C_α} and v_{C_β} has the $t-1$ clause anchor vertices v_{C_1} through $v_{C_{t-1}}$ as common neighbors and no common neighbors among the path vertices and the remaining clause vertices. The only candidates for the missing common neighbor of v_{C_α} and v_{C_β} are the literal vertices corresponding to the literals occurring in C.

Therefore, for all clauses at least one literal vertex corresponding to a literal in that clause must remain in order for the remaining graph to be a (2,t)-club. This is the case if and only if the remaining literal vertices represent a satisfying assignment of F. Therefore the constructed graph G_F can be turned into a (2,t)-club by N vertex deletions, if and only if F has a satisfying assignment. This polynomial-time reduction from Cnf-Sat to (2,t)-Club Vertex Deletion, where the allowed number k of deletions equals the number of variables N, therefore makes the SETH apply to (2,t)-Club Vertex Deletion. There can be no $O^*((2-\epsilon)^k)$ algorithm for (2,t)-Club Vertex Deletion, unless the SETH fails. □

As a final note, a simple observation due to Chang et al. [Cha+13] shows that the algorithm described in Theorem 2.13 minus dual parameterization (i.e. not aborting a branch after k vertices have been deleted, but trying to find the largest solution regardless of size) and any other algorithm for (2,t)-Club based on a search tree can achieve a worst case running time of $O^*(\alpha^n)$, where $\alpha \approx 1.618$ is the golden ratio.

Corollary 2.15. *(2,t)-Club can be solved in time $O^*(1.618^n)$.*

Proof. Consider the search tree algorithm of Theorem 2.13 (minus dual parameterization). Whenever we branch over a pair of non-agreeable vertices v and w, we can delete v in one branch and delete w and fix v (as contained in the sought solution) in the other branch, instead of just deleting w in the other branch. The reasoning is that the first branch already explores all possible solutions where v is not present, therefore deleting v at any future point in the second branch makes no sense. This observation leads to a branching vector of $(1, 2)$, which evaluates to the golden ratio α. □

Chapter 3

(2,t)-Club on Special Graph Classes

In this section we study the complexity of the (2,t)-Club problem on special graph classes. As the problem turns out NP-hard on general graphs, it is of interest to see on which graph classes it can be solved in polynomial time. An overview over a wide variety of common graph classes can be found in the book by Brandstädt et al. [Bra+99] or on the webpage *Information System on Graph Classes and their Inclusions* (http://www.graphclasses.org). For the s-Club problem Golovach et al. [Gol+14] and previously Schäfer [Sch09] studied the complexity on various graph classes. In particular we can generalize the polynomial-time solvability results of Schäfer [Sch09] for bipartite graphs, planar graphs and interval graphs to (2,t)-Club.

Figure 3.1 shows a hierarchy of graph classes and the complexity of the (2,t)-Club problem on these classes. In the following section we study the complexity of (2,t)-Club on each of these, with the exception of AT-free graphs, strongly chordal graphs, chordal bipartite graphs and distance-hereditary graphs, which we have included in the figure as they are related to the other classes and pose interesting open questions. The classification for chordal and weakly chordal graphs follows directly from our results for split graphs. In this chapter we prove polynomial-time solvability for interval graphs, cographs, co-cluster graphs, threshold graphs and planar graphs for arbitrary t and polynomial-time solvability for bipartite graphs for constant t. On split graphs (2,t)-Club turns out to be still NP-hard, but we can show fixed-parameter tractability with regard to solution size k for these graphs. Not shown in Figure 3.1 are co-bipartite graphs, for which we could only obtain partial results, namely polynomial-time solvability for $t \leq 2$. The NP-hard classification for (2,t)-Club on (2,τ)-clubs with $\tau < t$ can be easily seen to follow from our reduction in Theorem 2.9. Take an instance of (2,$t-\tau$)-Club on any graph, add τ dominating vertices as in the reduction, and you have an equivalent (2,t)-Club instance on a (2,τ)-club.

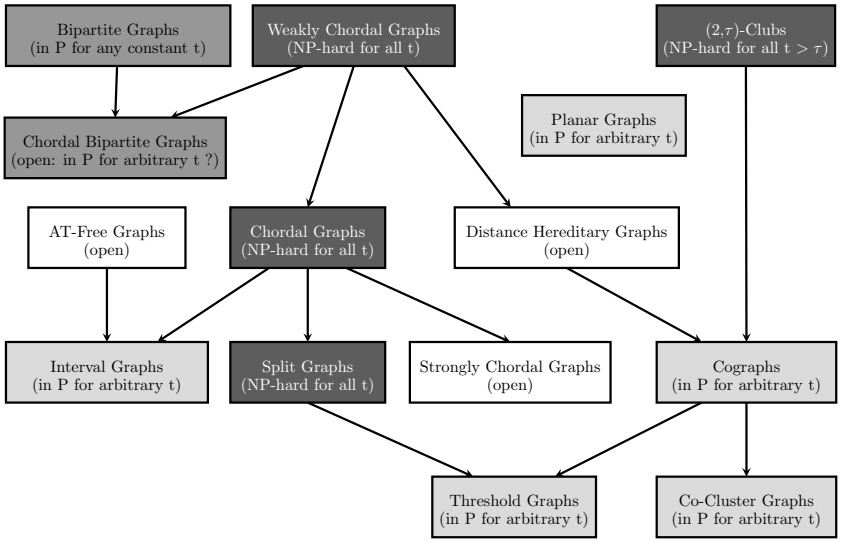

Figure 3.1: Diagram of various graph classes, showing their relationships to each other (a directed arrow means that the lower class is completely contained in the higher one) and the hardness of the $(2,t)$-Club problem on these classes. Superclasses inherit NP-hardness results from their subclasses, subclasses inherit polynomial-time solvability results from their superclasses.

3.1 Interval Graphs

In this section we will show that the $(2,t)$-Club problem is polynomial-time solvable on interval graphs. Interval graphs are graphs which are derived from a collection of intervals on the real line such that every interval is identified with exactly one vertex of the graph and two vertices are adjacent if their respective intervals overlap. Recognizing an interval graph and finding an interval representation for it can be done in time $O(n+m)$ time, as first shown by Booth and Lueker [BL76]. If we look at the interval representation of an interval graph, every point on the real line corresponds to a subset of the vertices of the graph, which forms a (not necessarily maximal and possibly empty) clique. If we view the beginnings and ends of intervals as events, then some point corresponds to a maximal clique if and only if the last event before the point was the beginning of an interval and the first event after the point is the end of an interval. Every maximal clique of an interval graph is identified by at least one point on the real line. Two points on the real line with no events in between them of course identify the same clique. Therefore an interval graph has at most n distinct maximal cliques, which can be easily found by scanning the interval representation from left to right. To prove polynomial running time for $(2,t)$-Club on interval graphs, we first prove the following lemma.

Lemma 3.1. *Every maximal (2,t)-club S of an interval graph G contains a maximal clique C_M of G. Furthermore any vertex v belonging to $S \backslash C_M$ is adjacent to at least t vertices from C_M.*

Proof. Take any two vertices v and w from a maximal (2,t)-club S of G which are not adjacent. If no such vertex pair exists, then S is a clique and because S is a maximal (2,t)-club of G, it then is a maximal clique of G and the claim follows. Otherwise, a maximal (2,t)-club contains two intervals I_v and I_w which do not overlap. As S is a (2,t)-club, there must be at least t intervals spanning the gap between I_v and I_w.[1] Let C be the vertex set corresponding to the intervals spanning I_v and I_w and let I_C be the corresponding interval set. Obviously C is a (not necessarily maximal) clique of G, as all intervals from I_C overlap with each other in the gap between I_v and I_w.

Now there can be no vertex x in $S \backslash C$ which is not adjacent to at least t vertices from C, because an interval I_x overlapping with $k < t$ intervals from I_C can only do so by overlapping with the k intervals of I_C which start first or by overlapping with the k intervals of C which end last. Consequently, I can overlap with at most one of I_v and I_w and can not overlap with the gap between I_v and I_w (as then it would overlap *all* intervals from I_C by definition). Now x has $k < t$ common neighbors from C with one vertex from v and w with which it is not adjacent. But there can be no further common neighbors outside of C, because the interval of every common neighbor would need to span the gap between I_v and I_w and therefore its vertex would be in C by definition.

We have now shown that every (2,t)-club S of an interval graph G contains a clique C such that each vertex from $S \backslash C$ is adjacent to at least t vertices from C. This is the second part of our claim, we still need to show that C is either a maximal clique of G or that C is contained in some maximal clique C_M of G such that $C_M \subseteq S$. If the second part of our claim already holds for C, then it also holds for $C_M \supset C$.

Proving the first part of the claim is easy now. Either C is a maximal clique of G or there is some maximal clique C_M of G with $C_M \supset C$. As every vertex v from S must be in C or adjacent to at least t vertices from C, every vertex v from S is agreeable to all vertices from $C_M \backslash C$. If $v \in C$ it is adjacent to all vertices in C_M, if $v \notin C$ it has t common neighbors in C with all vertices from $C_M \backslash C$. Therefore if S is a maximal (2,t)-club containing C it must also contain C_M. □

Using this lemma we can provide a polynomial-time algorithm for (2,t)-Club on interval graphs.

[1] For two intervals I_a and I_b, where I_a ends before I_b starts, we say an interval I spans the gap between them, if I starts no later than I_a ends and ends no earlier than I_b starts.

Theorem 3.2. *(2,t)-*Club *can be solved in time $O(n^2)$ on interval graphs.*

Proof. We are working on an interval representation of the input interval graph. If none is given, we can compute it in time $O(n+m)$ as shown by Booth and Lueker [BL76]. From the interval representation we can obtain a sorted list of events (by occurrence on the real line), where each event marks the start or end of an interval. To avoid complications in our argument we assume that in the interval representation no two events happen at the same time. It is easy to see that every interval graph has such an interval representation.[2]

Given this representation of G we can easily identify all $O(n)$ maximal cliques of G in linear time by scanning the list of events from beginning to end. Whenever an end event e occurs after a start event with no events in between, the set of vertices whose intervals do not start later than e and do not end earlier than e, that is all vertices containing the point of e, form a maximal clique C of G. The maximum $(2,t)$-club of G must contain some maximal clique of G. Therefore we just need to construct for each maximal clique C of G a maximum $(2,t)$-club containing C and take the largest of these at most n $(2,t)$-clubs. We will now show that we can find the maximum $(2,t)$-club containing some maximum clique C of G in $O(n)$ time, resulting in a $O(n^2)$ overall running time.

As observed above, a $(2,t)$-club S of G containing C can only contain vertices of $V(G)\backslash C$ which are adjacent to at least t vertices from C. Identifying the set of vertices $A \subseteq V(G)\backslash C$ adjacent to at least t vertices from C can easily be done in linear time $O(n)$ by scanning the interval representation of G from beginning to end. A vertex a belongs to A, if and only if its end event occurs after the first t start events for vertices from C *and* its start event occurs before the last t end events for vertices from C.

Now note that the vertices in A can be grouped into at most $2d$ types, where $d = |C| - t$. These are the types S_i with $t \leq i < |C|$ of vertices which are only adjacent to the i vertices from C whose intervals start first and the types E_i with $t \leq i < |C|$ of vertices which are only adjacent to the i vertices from C whose intervals end last. Note that $S_{|C|}$ and $E_{|C|}$ can be omitted because C is a maximal clique of G, so there can be no vertices in $V\backslash C$ adjacent to all vertices of C. As $|C| \leq n$ there are only $O(n)$ types. If we can add one vertex of a type to the solution S we can add all vertices of that type to the solution. Furthermore, if we can add type S_i to the solution, we can add all types S_j to the solution where $j > i$. And if we can add type E_i to the solution, we can add all types E_j to the solution where

[2] If multiple events happen at the same time and are directly preceded by an event at point a and directly followed by an event at point b, then spread these events out over the open interval (a, b) such that all start events occur before all end events. The interval graph of the resulting interval representation is the same as the interval graph of the original collection of intervals.

$j > i$. For every s with $t \leq s < |C|$ there is a unique e with $t \leq e < |C|$ and $|E_e| > 0$, such that if we add all types S_i with $i \geq s$ to the solution, we can add all types E_i with $i \geq e$ to the solution but no more. Therefore, we only have to consider the $O(n)$ possible choices for s to find the largest $(2,t)$-club containing C. As for every two values s_1 and s_2 of s with $s_1 < s_2$ it holds that $e_1 \geq e_2$ for the values e_1 and e_2 of e which correspond respectively to s_1 and s_2, we only need $O(n)$ time to find the corresponding values of e *for all* possible choices of s. If we decrease s, then e can only increase or stay the same, and vice versa. Therefore we can find the maximum $(2,t)$-club of an interval graph G containing some maximal clique C of G in $O(n)$ time. As there are at most n maximal cliques in an interval graph, finding the maximum $(2,t)$-club of an interval graph requires $O(n^2)$ time. □

3.2 Cographs

In this section we will provide an algorithm solving $(2,t)$-Club on the class of cographs in time $O(nt^2)$ given a cotree representation of the input graph. If a cotree representation is not given, additional running time $O(n + m)$ is required to recognize the cograph as such and construct a cotree representation, which brings the overall running time to $O(nt^2 + m)$. The same is true if one is not satisfied with the vertex set of a largest $(2,t)$-club as solution, but requires a copy of the solution subgraph (including edges) as return value.

There are various different characterizations for cographs. For instance, cographs are the graphs where every connected component has diameter 2 and is distance-hereditary. The characterization which lends itself best to devising algorithms for cographs is that every cograph can be represented by a cotree. A cotree T_G for some cograph G is defined as follows: Each vertex of G is a leaf in T_G. The inner nodes of T_G represent subgraphs of G and there are two types of inner nodes: 0-nodes (or *union nodes*) and 1-nodes (or *join nodes*). A union node corresponds to the disjoint union of the subgraphs represented by its children. A join node corresponds to the *join* of the subgraphs represented by its children. The *join* of two or more graphs is the disjoint union of the original graphs retaining all edges inside the original graphs and finally adding edges between all pairs of vertices which originally belonged to different graphs. The *join* can also be obtained by forming the disjoint union of the complements of the original graphs and then complementing the result again. One can recognize a cograph and construct a cotree representation of it in time $O(n + m)$; refer to the paper by Habib and Paul [HP05] for a description of the algorithm. With the cotree representation of a cograph many graph problems can be easily solved using a bottom-up dynamic programming approach. We will now

give a relatively simple algorithm for (2,t)-Club on cographs utilizing the cotree representation. While inner nodes of cotrees can have any number of children, we will restrict ourselves, for the sake of simplicity, to binary cotrees. It can be easily seen, that for every cotree we can construct an equivalent binary cotree by replacing any inner node of type T and with c children by a cascade of $c-1$ binary nodes of type T, where one child of all but the last binary T-node in the cascade is the next T-node in the cascade and the remaining children are the children of the original higher degree T-node. The ordering of the cascade does not matter. If we are given a non-binary cotree we can transform it into an equivalent binary cotree in $O(n)$ time.

Theorem 3.3. *Given a (binary) cotree representation of a cograph G, (2,t)-*Club *can be solved in time $O(nt^2)$.*

Proof. We employ a dynamic programming approach where we store for each inner node of the cotree a table of at most $t+1$ values, denoting for each value τ from 0 to t the size of the largest (2,τ)-club of the subgraph represented by the node. For ease of notation we also consider (2,0)-clubs here; the largest (2,0)-club of any graph is by definition just the whole graph itself as every graph is a (2,0)-club. This information for each node is sufficient to compute the size of a largest (2,t)-club of the input cograph. To return a vertex set forming a maximum (2,t)-club we will need to store additional information for each value τ in the table. We will introduce this additional information while describing the idea of the algorithm.

We fill in the tables bottom-up beginning with the leaves until we are able to fill in the root table and thus solved the problem. For the leaves the vertex represented by the leaf is trivially the largest (2,τ)-club of the subgraph for any value of τ. For union nodes, the largest (2,τ)-club for any $\tau > 0$ of the subgraph is simply the maximum cardinality (2,τ)-club among the largest (2,τ)-clubs of the children of the union node. In order to reconstruct the vertex set of a largest (2,τ)-club of the input cograph later on, we need to store additionally to the cardinality of a maximum (2,τ)-club for each value $\tau > 0$ the child containing said maximum (2,τ)-club.

The non-trivial part of the algorithm is filling in the tables for join nodes. As a first observation, if both children of a join node have at least τ vertices in the subgraphs represented by them, the subgraph represented by the join node is trivially a (2,τ)-club, as only vertices belonging to the same original subgraph can be non-adjacent after the join and have at least all vertices of the other original subgraph as common neighbors. If one subgraph has only $c < \tau$ vertices, the set of vertices taken from the other subgraph must at least form a (2,$\tau - c$)-club in order to be extendable to a (2,τ)-club by adding the vertices of the first subgraph. Therefore, whenever the cardinality c of one side is too low, we must restrict the vertices of the other side to a (2,$\tau - c$)-club. Restricting the other side to a (2,$\tau - c$)-club might in turn make its

cardinality c' too low and we have to restrict the first side to a $(2,\tau-c')$-club, which in turn might shrink the cardinality of the first side even further and require a more restrictive club on the other side, and so on. The algorithm for a join node therefore initializes with values $t_L = t_R = 0$, which means that first we try to add the largest (2,0)-club of each side, that is all vertices. For given t_L and t_R we can retrieve c_L and c_R, the cardinality of the largest $(2,t_L)$-club respectively $(2,t_R)$-club of the left respectively right subgraph, from the tables already computed by the bottom-up process. Now, while $t_L + c_R < \tau$ or $t_R + c_L < \tau$ we must adjust the values of t_L and t_R (and consequently update c_L and c_R) until both $t_L + c_R \geq \tau$ and $t_R + c_L \geq \tau$. If we only adjust t_L and t_R by the difference of $t_L + c_R$ respectively $t_R + c_L$ to τ, we are guaranteed that as soon as both sums are at least τ, the current value of $c_L + c_R$ is the size of the largest $(2,\tau)$-club of the subgraph represented by the join node. To later reconstruct a vertex set of a maximum size $(2,t)$-club of the entire graph, the table also needs to store for each τ the values t_L and t_R which provide a largest $(2,\tau)$-club of the join node.

The binary cotree of a given cograph has $O(n)$ nodes. For leaves and union nodes we need $O(1)$ time to compute the table entry for each τ. For join nodes we need to perform the adjustment algorithm for t_L and t_R for each table entry. As we only increase t_L and t_R and both t_L and t_R are bounded by 0 and t, we only need $O(t)$ time for each run of the adjustment process. As each vertex of the cotree holds as most t table entries the overall running time of the algorithm is in $O(nt^2)$. □

As a side note, in an actual implementation of the algorithm one should favor a top-down approach over a bottom-up approach, such that a table entry for a node is only computed the first time it is requested by the parent node. It is likely to happen that many of the overall $O(nt)$ table entries are never required to compute the largest $(2,t)$-club for the entire graph. For instance we never require the entries for $\tau \geq t'$ of the children if we want to compute the largest (2,t')-club of a join node. Therefore any node which has d join nodes on its path to the root never needs to store table entries for values larger than $t - d$. And if for any node we never have to compute values for $\tau > t'$ we never have to compute values for $\tau > t'$ for any node in the subtree of which the considered node is the root.

3.3 Co-Cluster Graphs and Threshold Graphs

In this section we will provide linear time algorithms for (2,t)-Club on co-cluster graphs and threshold graphs. While we have already shown $O(nt^2)$ respectively $O(nt^2 + m)$ running time for the class of cographs, which contains the classes of co-cluster graphs and threshold graphs, we can provide very simple $O(n)$ or $O(n + m)$ algorithms[3] for these types of graphs which can make a difference for higher values of t.

(2,t)-Club is trivial on cluster graphs, i.e. graphs which are the disjoint union of complete graphs. The largest cluster of a cluster graph is also always the largest (2,t)-club irregardless of the value of t. On co-cluster graphs, which are derived from complementing a cluster graph, the problem is also easily solvable but this requires a few observations.

Theorem 3.4. *(2,t)-Club is solvable in time $O(n)$ on co-cluster graphs, given a co-cluster representation of the graph identifying each co-cluster and the vertices belonging to it.*

Proof. In a co-cluster graph two vertices belonging to the same co-cluster are never adjacent and those belonging to different co-clusters are always adjacent. Any two vertices of the same co-cluster have all vertices belonging to different co-clusters as common neighbors. Therefore, if no co-cluster contains more than $n - t$ vertices, the entire graph is trivially a (2,t)-club. If some co-cluster contains more than $n - t$ vertices, we can take only one vertex of this co-cluster into any (2,t)-club. By symmetry (all vertices of a co-cluster are adjacent to all vertices outside of it), it does not matter which vertex we take. Note that after removing all but one vertex from this large cluster, the entire graph has at most t vertices left and no two non-adjacent vertices can be in any (2,t)-club together. Therefore, if some co-cluster has more than $n - t$ vertices, a largest (2,t)-club of the graph is a clique containing exactly one vertex from each co-cluster. Checking whether any co-cluster has more than $n - t$ vertices can obviously be done in overall time $O(n)$ given a co-cluster representation of the graph. Likewise, either reporting that the whole graph is the solution or returning exactly one vertex from every co-cluster as solution can be done in time $O(n)$. Therefore, (2,t)-Club is solvable in time $O(n)$ on co-cluster graphs given a co-cluster representation. □

If the input graph is not given as co-cluster representation, additional running time $O(n + m)$ applies for recognizing the input as a co-cluster graph and identifying the co-clusters, leading to an overall running time of $O(n + m)$. Demanding a copy of the graph when the whole graph is the

[3]Depending on whether we are given a nice representation of these graphs or have to construct one first.

solution also requires $O(n + m)$ time for cloning the graph data structure. Co-cluster graph recognition and co-cluster identification can be done in $O(n + m)$ time using the following method: Consider the vertices of the graph in any order. If the currently considered vertex is the first or adjacent to all vertices already considered, add it to a new co-cluster. If the considered vertex is not adjacent to some vertices already considered, it must belong to the same co-cluster as these vertices. If any two of these non-adjacent vertices are marked for different co-clusters the input graph is no co-cluster graph. Note that if the input graph is not given in a sparse representation like adjacency lists, but as an adjacency matrix, the time for recognition and thus the overall running time increases to $O(n^2)$.

Next we provide a linear-time algorithm for $(2,t)$-Club on threshold graphs. Threshold graphs are graphs where each vertex can be assigned a weight, such that there is an edge between two given vertices if and only if the sum of the weights of both vertices exceeds some threshold value (which is the same for the entire graph and all pairs of vertices). All threshold graphs can be constructed by a sequence of two operations:

1. Adding a new isolated vertex.
2. Adding a new vertex adjacent to all prior vertices.

Therefore threshold graphs can be represented by a length-n sequence of binary digits, where each digit is identified with one vertex of the graph, where 0 denotes adding an isolated vertex and 1 denotes adding a dominating vertex. Given such a sequence representation of a threshold graph, finding a set of vertices constituting a maximum $(2,t)$-club is easy.

Theorem 3.5. *$(2,t)$-Club is solvable in time $O(n)$ on threshold graphs, given a sequence representation of the graph.*

Proof. The following simple rule finds a vertex set constituting a maximum $(2,t)$-club: Delete all 0-vertices which are succeeded by less than t 1-vertices, with the sole exception of a leading 0-vertex (a 0 which is the first digit in the sequence). Applying this rule can obviously be done in $O(n)$ time by scanning the sequence backwards, counting the number of 1-vertices encountered so far and deleting all encountered 0-vertices while the counter is smaller than t. We always leave the first digit untouched. The correctness of the rule follows from a few simple observations: First, all pairs of 1-vertices are agreeable to each other, because the latter vertex of a pair is always made adjacent to the former when adding it. Second, a 0-vertex is agreeable to all 1-vertices if and only if it is succeeded by at least t 1-vertices or if it precedes all 1-vertices. Two 0-vertices are agreeable if and only if both are succeeded by at least t 1-vertices. Therefore we can always add all 1-vertices and a leading 0-vertex as well as all 0-vertices succeeded by

at least t 1-vertices. Adding a 0-vertex which neither has at least t succeeding 1-vertices nor is the first vertex in the sequence never helps, as then we cannot add any vertices preceding it while gaining no additional vertices succeeding it. □

Again, the running time increases to $O(n + m)$ if we are not satisfied with the set of vertices constituting the maximum (2,t)-club, but require a copy of the entire subgraph including all edges between vertices of the solution. Also, if we are not given the sequence representation described above, we also require $O(n + m)$ time constructing it, or $O(n^2)$ time if the graph is described by a complete adjacency matrix instead of adjacency lists or other sparse descriptions. Finding the sequence representation of a threshold graph (and recognizing it as such) can be done as follows:

For the input graph G determine the degree of each vertex, which can be done in time $O(n+m)$ for adjacency lists as every edge is observed only twice, but requires $O(n^2)$ time if the input is described by an adjacency matrix. Afterwards sort the vertices into buckets according to their degree, which requires $O(n)$ time. Initialize $N = 0$, $D = 0$ and $s = \epsilon$, where N is the number of vertices we have already processed, D the number of dominating vertices (that is, 1-vertices) found so far and s is the sequence representation we want to build. We construct the sequence representation backwards by repeatedly looking for dominating or isolated vertices in the remaining graph and (implicitly) removing them. The algorithm finishes when $N = n$ with the sequence representation stored in s. Until then at any step we do:

(1) Check whether there is some vertex of degree $n - N - 1$ by looking at the respective bucket. Such a vertex would be dominating in the remaining graph after we have removed the first N vertices. If there is such a vertex, remove it from the bucket, increment N and D by one, append $'1'$ to the beginning of s and proceed with the next iteration.

(2) If (1) did not apply, check for a vertex of degree D by looking at the respective bucket. Such a vertex would be isolated in the remaining graph after we have removed D previous 1-vertices. If there is such a vertex, remove it from the bucket, increment N by one, append $'0'$ to the beginning of s and proceed with the next iteration.

(3) If neither (1) or (2) applies at any step, the input graph was no threshold graph.

We can remember which vertex of the graph we associated with each digit in the sequence to later identify a set of vertices of the graph which forms a maximum (2,t)-club after running our (2,t)-Club algorithm for threshold graphs. Each iteration takes only constant time and overall there are $O(n)$ iterations. Consequently, generating the sequence representation

of a threshold graph can be done in time $O(n + m)$ respectively $O(n^2)$ depending on the format of the input.

3.4 Planar Graphs

In this section we will prove that (2,t)-Club is solvable in polynomial time on planar graphs. We do so by proving that there is only a finite number of planar (2,3)-clubs before arguing that (2,2)-Club can be solved in polynomial time on planar graphs, too. In the following proof we will use + to denote a *graph join*, which we already encountered when discussing cographs and which is defined as $G + H = \overline{\overline{G} \uplus \overline{H}}$, where $\uplus$ is the disjoint union of two graphs.

Theorem 3.6. *There are only eight different planar (2,3)-clubs, namely the complete graphs K_1 through K_4, the $P_3 + \overline{K_2}$, the $C_3 + \overline{K_2}$, the $C_4 + \overline{K_2}$ and the $C_5 + \overline{K_2}$.*

Proof. Given a planar (2,3)-club S, we consider a vertex v of degree at most 5 in it, which according to Euler's formula must exist.[4] We assume that v has minimum degree among all vertices in S. Obviously, if $\deg(v) < 3$, then S must be a complete graph. We now consider the three remaining cases for $\deg(v)$. We use Kuratowski's theorem, which states that no planar graph can contain a $K_{3,3}$ or K_5 minor.[5] We know that every complete graph is a (2,3)-club and that the largest complete subgraph in a planar graph can be a K_4, we therefore restrict our attention in the following parts to non-complete (2,3)-clubs.

Case deg(v) = 3: Any vertex not adjacent to v must be adjacent to all vertices of $N_1(v)$. Consequently, there can be only one vertex not adjacent to v, otherwise we get a $K_{3,3}$ minor. If there is no vertex not adjacent to v, S is obviously the K_4. If there is one vertex u not adjacent to v, the vertices in $N_1(v)$ now all have two common neighbors, v and u, and must still induce a 2-club for S to be a (2,3)-club. The only 2-clubs on three vertices are the C_3 and the P_3. Therefore the case $\deg(v) = 3$ can lead to only two planar (2,3)-clubs, which are not complete: the $C_3 + \overline{K_2}$ and the $P_3 + \overline{K_2}$.

Case deg(v) = 4: First, let us assume that there are no vertices at distance 2 to v. Then S has some vertex of degree at most 3, otherwise we get the K_5, which is not planar. This contradicts the choice of v as minimum-degree vertex of S. Next, let us assume that there is some vertex u at distance 2 to v, which is adjacent to all of $N_1(v)$. Now there can be no further vertices at distance 2 to v (which each require three neighbors

[4] Euler's formula states that every (simple and connected) planar graph G obeys $|E(G)| \leq 3|V(G)| - 6$. Therefore, the average degree in a planar graph is always strictly less than 6.

[5] For details on Kuratowski's theorem see for example Diestel [Die12].

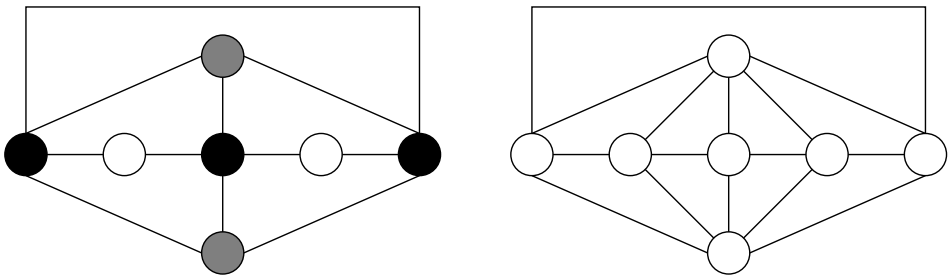

Figure 3.2: On the left side of the picture we see the situation for $\deg(v) = 4$ and two vertices at distance 2 to v, which both have three neighbors among $N_1(v)$, but not the same three. On the right side we see the unique (2,3)-club resulting from this situation.

among $N_1(v)$) or we get a $K_{3,3}$ minor. Now all vertices from $N_1(v)$ have v and u as common neighbors. Therefore $N_1(v)$ must still induce a 2-club. However, this 2-club can not have a vertex of degree 3, otherwise this vertex would cause a $K_{3,3}$ minor together with v and u, which now are all adjacent to the remaining vertices in $N_1(v)$. The only 2-club on four vertices without vertices of degree 3 is the C_4. Therefore, the only possible planar (2,3)-club in this case is the $C_4 + \overline{K_2}$. Finally, let us assume that vertices at distance 2 to v have exactly three neighbors in $N_1(v)$. No two of these vertices may have the same three neighbors in $N_1(v)$ or we get a $K_{3,3}$ minor. But then the vertices at distance 2 to v must form a 2-club, as every pair of them has exactly two common neighbors among $N_1(v)$ and either needs a further neighbor (which can only be from the set of vertices at distance 2 to v) or must be adjacent. This in turn means that there can be at most two vertices at distance 2 to v: If there were three such vertices, we contract two of them which are adjacent and the contracted vertex then has three common neighbors with both the remaining vertex at distance 2 to v and v itself, which leads to a $K_{3,3}$ minor. Therefore, we are left with two remaining cases, either one or two vertices at distance 2 to v, which have exactly three neighbors among $N_1(v)$. The first case is not possible, as the vertex from $N_1(v)$, which is not adjacent to the vertex at distance 2, then needs to be adjacent to the other three vertices from $N_1(v)$, just like v and the distant vertex, leading to a $K_{3,3}$ minor. The other case is illustrated in Figure 3.2. The black vertex in the center is v with four neighbors. There are two vertices (both also black) at distance 2 to it, each having three neighbors among $N_1(v)$, but not the same three neighbors. As the vertices at distance 2 to v must form a 2-club, as remarked before, the two black vertices at distance 2 are adjacent. Every pair of vertices, where at least one vertex is white and no vertex is black, now needs either one additional common neighbor or adjacency, in order for the graph to be a (2,3)-club. We cannot add edges to the black vertices, as we are assuming $\deg(v) = 4$ and that

both vertices at distance 2 to v have exactly three neighbors in $N_1(v)$. We cannot make the gray vertices adjacent, as then we could contract the white vertices with the center and arrive at a K_5. We cannot make the white vertices adjacent, as then we could contract the white vertices and arrive at a $K_{3,3}$ minor with the black vertices on one side and the contracted vertex and the gray vertices on the other. Therefore, we are only allowed edges between a white vertex and a gray vertex, and we need all four of them in order to get a (2,3)-club. The unique resulting graph is the $C_5 + \overline{K_2}$.

Case deg(v) = 5: Finally, we consider the case of $\deg(v) = 5$. Note, that any planar (2,3)-club with a vertex u with $\deg(u) < 5$ has been covered in the previous cases. Therefore, the remaining question is: Are there any planar (2,3)-clubs with minimum degree 5? Euler's formula states that $|E(G)| \leq 3|V(G)| - 6$ for any planar graph G. For a graph G with minimum degree 5 we have $|E(G)| \geq 2.5 \cdot |V(G)|$ due to the handshaking lemma. Combining both we get $2.5 \cdot |V(G)| \leq 3|V(G)| - 6$ and $12 \leq |V(G)|$. Consequently, S must have at least 12 vertices in this case and, because of $\deg(v) = 5$, at least six vertices at distance 2 to v. Let F be the set of vertices at distance 2 to v. Now at most one vertex from F can have four neighbors in $N_1(v)$: Two vertices from F with four neighbors in $N_1(v)$ would overlap in three vertices and form a $K_{3,3}$ minor together with v. However every vertex from F needs three neighbors in $N_1(v)$ and no two vertices from F may share the same three neighbors. Now consider a vertex w with three neighbors in $N_1(v)$. As the minimum degree of S is supposed to be 5, w must have at least two further neighbors in F. First, let us consider the case that there is a vertex u in F with four neighbours in $N_1(v)$. If we now contract w with one of its neighbors in F, which is not u, we then have two vertices in F with each four neighbors in $N_1(v)$ and consequently a $K_{3,3}$ minor formed by u, v and the contracted vertex. Secondly, let us assume there was no vertex in F with four neighbors in $N_1(v)$. The graph induced by F then must have minimum degree 2 due to our assumption that S has minimum degree 5. In any graph with six vertices and minimum degree 2 there are two edges which are not incident. For each of these two edges, contract the endpoints and we again arrive at two vertices in F ,with each four neighbors in $N_1(v)$. Therefore, there does not exist any planar (2,3)-club with minimum degree 5. □

This result leads to a linear time algorithm for (2,t)-Club on planar graphs for $t > 2$.

Corollary 3.7. *(2,t)-*Club *can be solved in linear time on planar graphs for $t > 2$.*

Proof. We can easily check the input graph in linear time for the presence of the subgraphs mentioned in Theorem 3.6, as Subgraph Isomorphism is solvable in linear time on planar graphs, given a fixed subgraph H to

look for. An algorithm for planar subgraph isomorphism running in time $|H|^{O(|H|)} \cdot n$, where $|H|$ is the size of the sought subgraph, was given by Eppstein [Epp99]. Dorn [Dor10] later gave an improved algorithm with running time $2^{O(|H|)} \cdot n$. □

Now that we have shown that (2,t)-Club can be solved in linear time for $t > 2$ on planar graphs, we consider (2,2)-Club for planar graphs. The set of planar (2,2)-clubs is not finite, as every $K_{2,t}$ with $t > 1$ is both planar and a (2,2)-club. However, we suspect that the following conjecture holds, recalling our earlier definition of t-diamonds from Definition 2.3.

Conjecture 3.8. *There is only a finite number of planar (2,2)-clubs which are not 2-diamonds.*

The motivation why this is important, if true, is simple. Checking a graph for the largest 2-diamond can be done easily in polynomial time. If the number of planar (2,2)-clubs which are not 2-diamonds is finite, we can afterwards look for these subgraphs in linear time using a subgraph isomorphism algorithm for planar graphs.

As we do not know whether the conjecture holds, we use another argument to show that (2,2)-Club is polynomial-time solvable on planar graphs. This argument has been used before in the work by Schäfer [Sch09] to show that s-Club is polynomial-time solvable on planar graphs.

Theorem 3.9. *(2,t)-Club is solvable in time $O(n^2+nm)$ on planar graphs.*

Proof. First note, that any (2,t)-club S in a graph G containing some vertex v must be fully contained within the closed 2-neighborhood of v. Therefore, it is sufficient to check the closed 2-neighborhood of each vertex in G for the largest (2,t)-club it contains and then return the largest of these (2,t)-clubs. If the check for every single vertex can be done in polynomial time, then checking the entire graph can be done in polynomial time. More precisely, if the check for any single vertex can be done in time $T(n, m)$, then checking the entire graph can be done in time $n \cdot T(n, m) + O(n(n+m))$, as identifying and extracting the closed 2-neighborhood for every vertex can be done in time $O(n(n + m))$. To identify the 2-neighborhoods one can use the $o(nm)$ all-pairs shortest path algorithm for undirected and unweighted graphs by Chan [Cha12].

The closed 2-neighborhood of any vertex has at most diameter 4. It is known that planar graphs with bounded diameter d have treewidth $O(d)$ and that a tree decomposition with this treewidth can be found in time $O(d \cdot n)$. This result was first implicitly given by Baker [Bak94]; see for instance Eppstein [Epp00] for an explicit mentioning of it.

In Theorem 4.4 we will show that (2,t)-Club can be solved in time $f(w) \cdot O(n)$ on graphs with treewidth w. Consequently, (2,t)-Club is solvable in

time $O(n)$ on planar graphs with diameter 2 and using our initial observation therefore solvable in time $O(n^2 + nm)$ on general planar graphs. □

Note that our proof for fixed-parameter tractability of (2,t)-Club with regard to treewidth relies on Courcelle's theorem (see Section 4.3 for details), which makes Theorem 3.9 a classification result only, as the constant hidden in the O-notation is likely to be infeasibly large.

3.5 Bipartite Graphs

In this section we provide a proof that (2,t)-Club for constant t is polynomial-time solvable on bipartite graphs. Schäfer [Sch09] observed that a subgraph S of a bipartite graph B is a 2-club if and only if S is a biclique. The simple argument is that two vertices belonging to different sides of a bipartite graph cannot have even distance to each other, so if their distance is supposed to be at most 2 they must be adjacent. As all (2,t)-clubs are also 2-clubs, every (2,t)-club in a bipartite graph must also be a biclique. But in contrast to 2-clubs each side of the biclique must have cardinality at least t or both sides must have cardinality 1, as every pair of non-adjacent vertices requires at least t common neighbors.

The problem of finding a maximum vertex biclique in a bipartite graph (without cardinality restrictions for the individual sides) can be solved in polynomial time via matching. This is a known result since at least Garey and Johnson [GJ79]. But as we were unable to find references[6] explaining and proving the whole procedure and especially stating concrete running time bounds, we will for the sake of completeness reproduce the procedure to achieve this here (without any claim to originality):

Given a bipartite input graph $G = (U \cup V, E)$ consider the *bipartite complement* $G' = (U \cup V, E')$ of G, where $\{u, v\} \in E'$ if and only if $u \in U \wedge v \in V \wedge \{u, v\} \notin E$. That is, all edges between the two sets U and V are complemented, but U and V themselves remain independent sets. There now is a one-to-one correspondence between bicliques in G and independent sets in G' and a maximum biclique of G is a maximum independent set of G' and vice versa. To find a maximum independent set in G' we can compute a minimum vertex cover of G', all vertices not belonging to some given minimum vertex cover of a graph constitute a maximum independent set of the graph. König's theorem (see Bondy and Murty [BM76], Theorem 5.3, p. 74) states that in any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover. This implies that for any given maximum matching there exists some minimum

[6] Hochbaum [Hoc98] discusses the more general biclique problem for graphs with weighted vertices, but the running time bounds for this more general problem are worse.

vertex cover which contains exactly one endpoint from each edge of the matching, as every edge of the matching must be covered, but by covering both endpoints of a matched edge we would then exceed the allowed number of vertices in the cover.

Furthermore, any vertex not incident to any edges of the matching cannot be in this minimal vertex cover, as we need all vertices to cover endpoints of matching edges. Therefore, we can first rule out all vertices not contained in the matching. We then must take all vertices adjacent to the vertices just ruled out into the vertex cover. These added vertices must by definition of maximum matchings be all part of the maximum matching. As we can only take one endpoint per matched edge, we can then rule out all vertices connected to a vertex just added via an edge which is contained in the matching. With these newly ruled out vertices we proceed as before, taking their neighbors into the vertex cover and then in turn ruling out the neighbors via matched edges of these vertices, and so on. This process repeats until no new vertices are visited. As we never visit any vertex twice and thus observe every edge at most twice, the running time of this algorithm is in $O(n+m)$. As we complement the input graph before running this algorithm, we must replace this bound with $O(n^2)$, as the running time does not depend on the number of edges of the original graph but the number of edges of the complement. The algorithm obviously makes only correct choices when deciding whether to put a vertex into the cover or not. However, the above algorithm might not decide for every matched edge which endpoint to put in the vertex cover, because there might be matched edges which are not connected to any unmatched vertices. If this happens, we can simply pick any one side of the bipartition and take all endpoints of undecided matching edges which are on this side into the vertex cover.

Why? Suppose there is some uncovered edge after performing above algorithm and then adding all endpoints of undecided matching edges of one side of the bipartition into the cover. The uncovered edge cannot belong to the maximum matching, as we put one endpoint of every edge of the matching into the cover. But both endpoints of the uncovered edge must also be endpoints of matched edges, otherwise our above algorithm would have ruled out the one which is not and taken the other into the cover. If both matched edges incident to the uncovered edge were undecided, the edge would now be covered as its two endpoints are in different parts of the bipartition and picking either side for the undecided edges would put one endpoint of the uncovered edge into the cover. If exactly one matched edge incident to the uncovered edge has been decided, it must have decided on the endpoint not belonging to the uncovered edge (as it is still uncovered). But then above algorithm would have ruled out the endpoint of the matched edge belonging to the uncovered edge and consequently would have ruled the other endpoint of the uncovered edge as part of the vertex cover. This leaves the case that both incident matched edges have been decided, but

neither has been decided on an endpoint of the uncovered edge. This cannot be, as now we cannot arrive at a vertex cover which contains only one vertex of every matched edge, but we have argued that our above algorithm does not make mistakes when deciding which vertices to put into the vertex cover and König's theorem guarantees us that there is some minimum vertex cover containing exactly one endpoint of every edge of any given maximum matching of the graph. Thus, picking all endpoints of uncovered matched edges belonging to one and the same side of the bipartition is correct and leads to a minimum vertex cover after the above $O(n^2)$ algorithm.

Therefore, by putting all these steps together we can use a bipartite matching algorithm like the Hopcroft-Karp algorithm (see Hopcroft and Karp [HK73]) to find the maximum vertex biclique of a bipartite graph. The Hopcroft-Karp algorithm runs in time $O(m \cdot \sqrt{n})$, but as we complement the input graph before running the Hopcroft-Karp algorithm we must replace this bound with $O(n^{2.5})$. All the steps transforming Maximum Vertex Biclique on bipartite graphs to a bipartite matching problem can also be done within this time. Consequently, finding a maximum vertex biclique of a bipartite graph can be done in $O(n^{2.5})$ time.

We use this result to extend it to an $O(n^{2t+0.5})$ algorithm for solving (2,t)-Club on bipartite graphs, which proves polynomial-time solvability of (2,t)-Club on bipartite graphs for any constant t. Note, that this is mostly a classification result, as the algorithm is infeasible for larger values of t.

Theorem 3.10. *(2,t)-Club can be solved in time $O(n^{2t+0.5})$ on bipartite graphs for some constant t.*

Proof. As remarked before, on bipartite graphs finding a maximum size (2,t)-club is equivalent to finding a maximum size biclique with at least t vertices on either side of the partition. For each subset S of each $t-1$ vertices from either side of the bipartition, check whether the induced subgraph $G[S]$ is a complete bipartite graph. If it is not, proceed with the next subset. If it is, construct the graph G' which contains no vertices from S and only those vertices from $V \setminus S$ which are common neighbors of all the $t-1$ vertices of S which belong to the other side of the bipartition. The edges between the vertices of G' are the same as in G. Now find a maximum size biclique of G' via the Hopcroft-Karp algorithm. If B is such a maximum size biclique of G', then $B \cup S$ is a maximum size biclique among all bicliques of G which contain all of S. By generating all sets S which are bicliques with $t-1$ vertices on either side, we are guaranteed to find the maximum size biclique on G which has at least t vertices on either side. If no such biclique exists, any edge of G is a maximum size (2,t)-club. Overall we generate $O(n^{2t-2})$ subsets and perform the $O(n^{2.5})$ Hopcroft-Karp algorithm on each of them.

Checking whether a given subset is complete bipartite requires $O(n \cdot t)$ time, which is dominated by the running time of Hopcroft-Karp as we consider t to be a constant. Therefore the algorithm has a running time of $O(n^{2t+0.5})$. □

$(2,t)$-Club for arbitrary unbounded t is NP-hard on bipartite graphs, because it would solve the problem Balanced Complete Bipartite Subgraph as a by-product. This problem asks whether a bipartite graph contains a $K_{t,t}$ for some integer t as an induced subgraph. It was shown by Garey and Johnson to be NP-complete, see Garey and Johnson [GJ79], problem [GT24].

3.6 Co-Bipartite Graphs

In this section we study the complexity of $(2,t)$-Club on co-bipartite graphs, complements of bipartite graphs. Co-bipartite graphs can alternatively be characterized as graphs with clique cover number two. One can easily recognize co-bipartite graphs and identify the two cliques in polynomial time, as 2-colorings can be computed in linear time. The decomposition of a co-bipartite graph into two cliques is only ambiguous if the graph contains any dominating vertices, that is vertices adjacent to all other vertices of the graph. However, as remarked before at the end of Section 2.2, if the graph G of an instance of the $(2,t)$-Club problem contains d dominating vertices, we can replace it by the equivalent $(2,t-d)$-Club instance on the graph G', where all dominating vertices have been removed. Therefore, we can assume in the following that the decomposition into two cliques is always unique and results in two disjoint maximal cliques. The decomposition into two cliques C_1 and C_2 can be computed by initially labelling one arbitrary vertex of the graph as part of C_1. As long as some vertices remain unlabelled pick any labelled vertex with unlabelled non-neighbors and label all unlabelled non-neighbors with the other clique label. This obviously can be done in $O(n^2)$ time. Now we prove that $(2,2)$-Club is polynomial-time solvable on co-bipartite graphs.

Theorem 3.11. *(2,2)-Club can be solved in $O(n^2)$ time on co-bipartite graphs.*

Proof. Given the decomposition of the co-bipartite input graph G into two maximal cliques C_1 and C_2, we label the vertices in both cliques as follows. Vertices with two or more neighbors in the other clique are called *multiple link vertices*, those with exactly one neighbor in the other clique are called *single link vertices*, all others are *remote vertices*. Now two vertices v and w are agreeable if and only if:

1. They are in the same clique
2. or neither vertex is remote
3. or one of them is a multiple link vertex.

Case 1 is obvious. If they belong to different cliques and neither v nor w are remote, both have at least one neighbor w' or v', respectively, in the other clique. If $v = v'$ or $w = w'$, then v and w are adjacent. Otherwise $vv'w$ and $vw'w$ are two length-2 paths connecting them. Lastly, a multiple link vertex is agreeable to all other vertices, as any vertex in the other clique can be reached via at least two length-2 paths due to the multiple links into that clique. Two remote vertices from different cliques are not agreeable, as all paths between them must pass at least one link vertex from each clique and are thus of length ≥ 3. A remote vertex is also not agreeable to any single link vertex from the other clique, as there is only one path of length two between them. We are therefore left with only three possibilities for a maximum (2,2)-club:

1. all vertices from C_1 and all multiple link vertices from C_2
2. all vertices from C_2 and all multiple link vertices from C_1
3. or all multiple and single link vertices from both cliques

The first is a (2,2)-club, because it only contains those vertices of C_2 which are multiple link vertices and thus agreeable to all vertices, and cannot be extended unless C_1 has no remote vertices (in which case the optimal solution is covered by the other cases). The same argument holds for case two. In the last case we have a (2,2)-club, because no remote vertices are in it, and it is maximal, because we cannot add any remote vertex, unless one clique has no single link vertices (which is covered by cases 1 and 2). So none of these (2,2)-clubs can be extended and all other (2,2)-clubs can be extended to one of these.

The running time of the algorithm is in $O(n^2)$. Computing a clique decomposition (if none is given) can be done in $O(n^2)$ time. Labelling all vertices with the appropriate vertex type (remote, single link, multiple link) can be done in $O(n^2)$ time by counting the neighbors in the other clique for each vertex. Determining the vertex sets of the three candidate graphs and comparing their cardinality can be done in $O(n)$. □

Solving 2-Club on co-bipartite graphs is easy: A pair of vertices is not agreeable if and only if both vertices are remote and belong to different cliques. Therefore, the only two candidates for maximum 2-clubs on co-bipartite graphs are the graphs derived from deleting all remote vertices of one of the two cliques. For the general $(2,t)$-Club problem, we define the

link number of a vertex to be the number of neighbors it has in the other clique. Now, two vertices are not agreeable if and only if they belong to different cliques and the sum of their link numbers is smaller than t. For $t \leq 2$ two vertices being agreeable always means that there was also some maximal $(2,t)$-club containing both vertices. Even more, it means that for any two agreeable vertices, if the vertex with the lower link number was in some $(2,t)$-club the other vertex could be added, too. The problem for $t \geq 3$ now is that having link number sum $\geq t$ is only a necessary but not a sufficient condition for two given vertices v and w to be in some $(2,t)$-club together. We might not be able to put t of the common neighbors of v and w into a $(2,t)$-club together, because no t of the common neighbors are all mutually agreeable. For instance consider the case $t = 3$ and a vertex v with link number 2 in one clique and a vertex w with link number 1 in the other. The vertices v and w are agreeable, but their three common neighbors might not be part of any (2,3)-club together, for instance because they are all vertices with link number 1 and two vertices with link number 1 from different cliques cannot be in a (2,3)-club together unless adjacent, which cannot be the case here.

So we see that $(2,t)$-Club on co-bipartite graphs seems to be more problematic when $t \geq 3$ and we leave it as an open question here which running time can be achieved for higher values of t.

3.7 Split Graphs

In this section we will show NP-hardness for $(2,t)$-Club on split graphs, which also implies NP-hardness for larger graph classes like chordal graphs, which contain the class of split graphs. However, despite this negative result, we will also show, that $(2,t)$-Club is fixed parameter tractable with regard to the solution size k on split graphs.

Theorem 3.12. *$(2,t)$-Club remains NP-hard on split graphs.*

Proof. Consider the following simple reduction from Clique to $(2,t)$-Club: Make the original vertex set independent and introduce t new vertices for every edge of the original graph and make them adjacent to the original two vertices incident to this edge. Make all of the introduced edge vertices a clique. The reduction is correct, as only those original vertices have t common neighbors which were adjacent in the original graph and all introduced edge vertices are adjacent to each other and have at least t common neighbors with all original vertices, so any size k clique in the original graph corresponds to a size $k+mt$ $(2,t)$-club in the new graph, where m is the number of edges in the original graph. Now notice that the graph constructed by the reduction is a split graph. Therefore $(2,t)$-Club is NP-hard on split graphs. □

Theorem 3.13. *(2,t)-*Club *is fixed parameter tractable with regard to solution size k on split graphs and can be solved in time $2^{2^k} \cdot poly(|G|)$.*

Proof. We consider a partition of the split graph into a clique and an independent set such that the clique is maximal. Any maximum size $(2,t)$-club of a split graph always contains the entire clique. If the clique has size at least k, the instance is trivial and we can simply return the clique as solution. Otherwise, if the clique has size $g < k$, we can now partition the vertices in the independent set into $2^g < 2^k$ types according to their neighborhood in the clique. If we can take one vertex of a type, we can take all vertices of that type. Therefore we must only decide which types we add to the clique to obtain a maximum size $(2,t)$-club, deciding for each of the $O(2^k)$ types whether to add it or not, and running time $2^{2^k} \cdot poly(|G|)$ follows. □

Chapter 4

Structural Parameterization for (2,t)-Club

In this section we will have a brief look at structural parameterization for (2,t)-Club. In fact, with the results easily transferring from the structural parameterization results by Hartung et al. [Har+15a] for 2-Club as seen in Corollary 2.10 and a positive fixed-parameter tractability result for treewidth in Section 4.3 we can already classify a rather wide variety of structural graph parameters as either FPT, W[1]-hard or already NP-hard for constant parameter values.

Furthermore we will introduce a (to our knowledge) new parameter called h_2-index in Section 4.2 which is tied to practical algorithmic considerations concerning the (2,t)-Club problem and provides better running time bounds for the majority of real world graphs. We also propose an even stronger parameter, which we call N_2-degeneracy, whose practical use we did not evaluate as it is both harder to compute and requires additional (polynomial) preprocessing work by an algorithm to realize a better running time bound based on it.

Figure 4.1 shows a hierarchy of parameters, where we show the relation of parameters to each other and the classification results we could obtain. The results with NP-hardness for constant parameter value as well as W[1]-hardness for h-index all followed easily from the results of Hartung et al. [Har+15a] as we have seen in Corollary 2.10. The FPT result for distance to (2,t)-club has been proven in Theorem 2.13, where we have shown that (2,t)-Club Vertex Deletion can be solved in time $O^*(2^k)$. From this fixed-parameter tractability for distance to clique follows immediately as this parameter is lower-bounded by distance to (2,t)-club. In Section 4.1 we will show fixed-parameter tractability for the maximum degree and in section Section 4.3 we show fixed-parameter tractability for treewidth which immediately leads to many parameters which are lower-bounded by treewidth to be FPT, too. A number of parameters of the form 'distance to graph class'

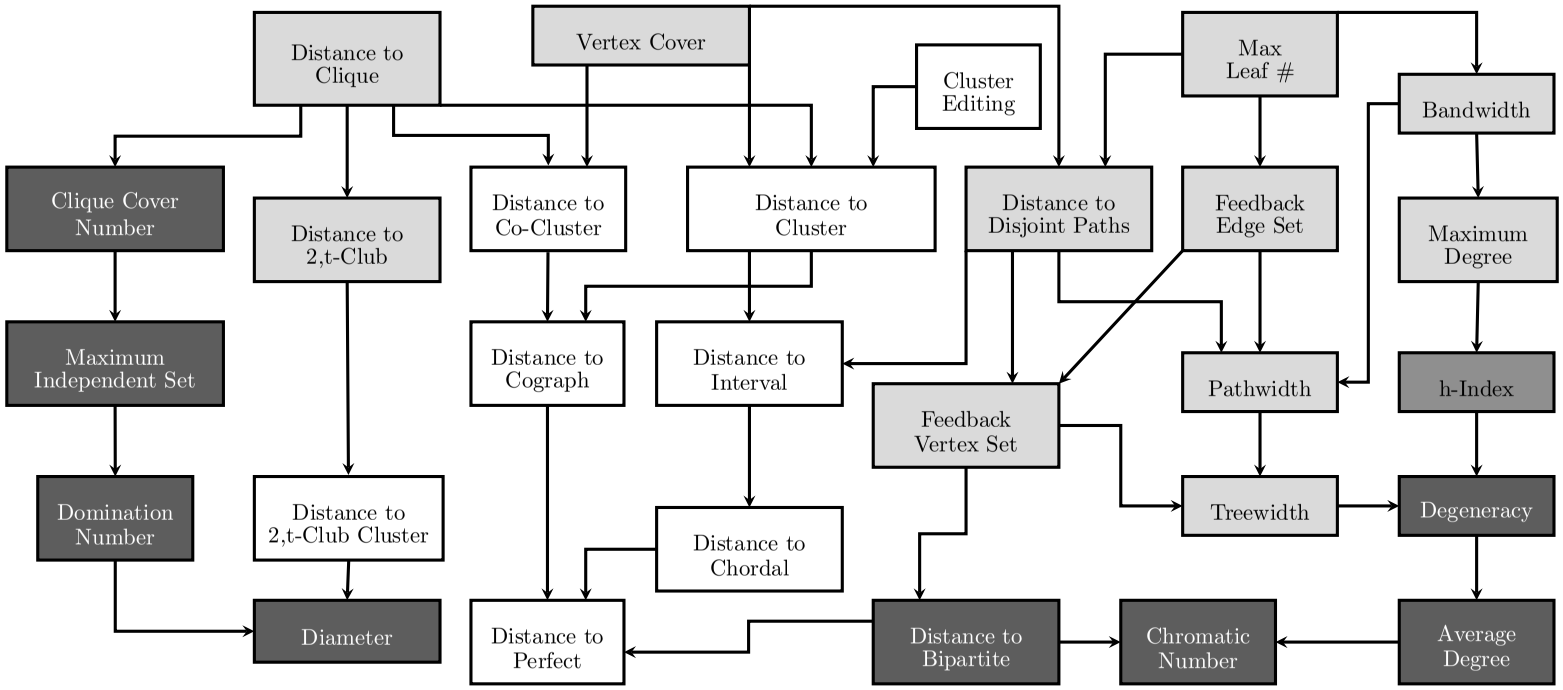

Figure 4.1: Diagram of graph parameters, showing their relationship to each other. An arrow from one parameter to another means that the higher up parameter is lower-bounded by the parameter further down. In darkest gray we have the parameters for which the problem is already NP-hard for constant parameter values, in medium gray for which it is W[1]-hard and in light gray for which it is FPT. The parameters for which we have no results so far are shown in white.

and the parameter 'size of cluster editing set' remain open. Certainly there are further parameters which could be of interest to the problem not shown in the figure.

First let us briefly return to the parameters discussed in Corollary 2.10. We have shown that for various parameters $(2,t)$-Club is already NP-hard for constant parameter values, but we have not yet discussed whether those bounds are strict. We have shown that $(2,t)$-Club is NP-hard on graphs with both diameter 2 and domination number 1 and it should be obvious that these bounds are tight. We have also seen that $(2,t)$-Club is NP-hard on graphs with clique cover number 3. Graphs with clique cover number 2 are the same as co-bipartite graphs discussed in Section 3.6. We have therefore shown that $(2,t)$-Club is polynomial-time solvable on graphs with clique cover number 2 for $t = 2$. Regarding higher t the question remains open. We have further seen in Corollary 2.10 that $(2,t)$-Club is NP-hard on graphs with distance t to bipartite graphs and we do not know whether this bound is tight or not. Likewise we do not know whether the bound $5 + t$ for degeneracy is tight, but we do know that the bound must be at least t. This follows from Observation 2.2 which implies that the only $(2,t)$-clubs in a graph with degeneracy less than t can be cliques, but Clique is FPT with regard to degeneracy. Last, we shown that $(2,t)$-Club is NP-hard on connected graphs when the average degree exceeds $2t$. Using the same argument used in Hartung et al. [Har+15a], adding long paths to the graph, this bound can be pushed down to 2.

4.1 Maximum Degree

The maximum degree Δ is a parameter for which it is easy to show fixed-parameter tractability, due to the fact that every $(2,t)$-club is fully contained in the closed 2-neighborhood of each of its vertices in the original graph, which itself stems from the fact that all $(2,t)$-clubs have diameter at most two. Nonetheless, maximum degree is an interesting parameter as we shall see.

Theorem 4.1. *$(2,t)$-Club can be solved in time $O^*(1.618^{\Delta^2})$ on graphs with maximum degree Δ.*

Proof. As already pointed out, every $(2,t)$-club has diameter at most two and is therefore fully contained in the closed 2-neighborhood of each of its vertices in the original input graph. To find the largest $(2,t)$-club in a graph G it is now sufficient to search each of the n closed 2-neighborhoods of G for a largest $(2,t)$-club and take the maximum among all of these $(2,t)$-clubs. In a graph with maximum degree Δ the size of the closed 2-neighborhood of any vertex v is upper-bounded by $\Delta^2 + 1$ as v can have at most Δ neighbors which each can have at most $\Delta - 1$ other neighbors besides v. Further,

as we have seen in Corollary 2.15, we can solve (2,t)-club on any graph in time $O^*(1.618^n)$ where n is the size of the graph. Therefore, the largest (2,t)-club contained in any one closed 2-neighborhood of G can be found in time $O^*(1.618^{\Delta^2})$. As there are n such neighborhoods in a graph of size n and finding the maximum among the n different largest (2,t)-clubs for each 2-neighborhhood can be done in time $O(n)$, the claim follows. □

While this result is simple it has important practical implications. It allows us to decompose larger (and relatively sparse) graphs into a linear number of smaller graphs whose size is bounded by Δ^2. These smaller individual graphs are called *Turing kernels* and the process of reducing the problem such *Turing kernelization*. We use this result in our own implementation of a (2,t)-Club algorithm described in Section 5.1, see there for further details on Turing kernelization.

4.2 A New Parameter for the (2,t)-Club Problem: h_2-Index

We can siginificantly strengthen the result just obtained for many real world graphs by introducing a new parameter derived from a distance-based generalization of the *h-index (Hirsch index)*. We recall that the h-index h of a graph G is the largest natural number h such that G contains at least h vertices of degree at least h. The degree of a vertex is the same as the size of its open 1-neighborhood. We now adapt the definition to closed 2-neighborhoods to arrive at a parameter relevant for the (2,t)-Club problem.

Definition 4.2. We call the largest h for which a graph G contains at least h vertices with a closed 2-neighborhood of size at least h the h_2-index h of this graph G.

Defining this index over either the closed or open 2-neighborhood does not make much difference as the parameter would change by at most one. Defining over the open neighborhood would yield a more direct generalization of the h-index. We still went with the closed neighborhood as this definition is closer to our problem. For this parameter we can show the following result:

Theorem 4.3. *(2,t)-Club can be solved in time $O^*(1.618^h)$ on graphs with h_2-index h.*

Proof. When decomposing the input graph into n kernels consisting of the closed 2-neighborhoods of the vertices of the input graph (as described above) we can remove a vertex v from all subsequent kernels after solving the kernel for vertex v. This is because any (2,t)-club containing v must be fully contained in the kernel for the vertex v, thus we cannot find any

larger (2,t)-clubs containing v after we have solved the kernel for v. Additionally we can sort the kernels by their size beforehand and then solve them in order of ascending size. Then at most h kernels have size larger than h by definition of the h_2-index, namely the last h kernels in the list of sorted kernels. However, if we remove every vertex whose kernel we have solved from subsequent kernels, these kernels can have size at most h once we come around to solve them. Therefore, the size of the largest kernel we have to consider can be at most h, replacing the weaker bound Δ^2 from Theorem 4.1 and leading to the claimed running time. □

We point to the real world test instances listed in Figure 5.3 in Section 5.3.5 to motivate the fact that on real world graphs the h_2-index often provides a much better bound than the square Δ^2 of the maximum degree. This bound is also a considerable improvement over the $O^*(1.618^n)$ bound obtained by Chang et al. [Cha+13] for all graphs where the h_2-index is considerably smaller than n, which should apply for many real world social networks and other graphs.

We could obtain the same running time result for an even stronger parameter based on a distance-based generalization of *degeneracy*. We could define the *N_2-degeneracy (two-neighborhood-degeneracy)* d of a graph G as the smallest number d such that every subgraph of G contains a vertex with a closed 2-neighborhood of size at most d. Then we could solve (2,t)-Club in time $O^*(1.618^d)$ if we make sure that the list of kernels remains sorted by size at all times, that is if we update kernel sizes and re-sort the list accordingly after each deletion of a vertex. It is easy to see that in this case the largest kernel we ever encounter has size at most d. While N_2-degeneracy should obviously be computable in polynomial time, we suspect that the running time overhead for keeping the kernel list sorted at all times might be too large for many practical purposes despite being polynomial. Note that we would have to update kernel sizes after each vertex deletion, that is $O(n)$ times, and we suspect that each update requires to re-evaluate the closed 2-neighborhood of every vertex adjacent to the deleted vertex (kernels of vertices at distance 2 to the deleted vertex simply decrease by one in size). This suggests a running time which is worse by a factor $O(\Delta)$ than determining kernel sizes just once, neglecting the fact that we also have to re-sort the kernel list after every update (instead of just once). Therefore, we went with the h_2-index as a weaker but more easily computable and probably more widely applicable parameter.

Further generalizations of h_2-index and N_2-degeneracy to h_s-index and N_s-degeneracy are also of interest for the s-Club problem and the more general (s,t)-Club problem. That is, these problems are fixed-parameter tractable with regard to h_s-index or N_s-degeneracy with the same running time bound as above (using the exact same Turing kernelization argument).

4.3 Treewidth

In this section we will prove that (2,t)-Club is fixed-parameter tractable when parameterized by the treewidth w of the input graph. To show this we use Courcelle's famous theorem (see Courcelle [Cou90]), which states that every graph property which can be expressed as a formula Φ in MSOL (monadic second order logic) is decidable in time $f(|\Phi|, w) \cdot O(n)$, where $|\Phi|$ is the length of the formula Φ, on graphs with treewidth w. To be precise, we need an extension of MSOL called LinEMSOL (linear extended monadic second order logic) by Courcelle et al. [Cou+00], which was studied by Arnborg et al. [Arn+91] and for which Courcelle's theorem also holds.

Theorem 4.4. *(2,t)-*Club *can be solved in time $f(w) \cdot O(n)$ on graphs with treewidth w.*

Proof. We need to find a MSOL formulation of (2,t)-Club to apply Courcelle's theorem to the problem. As Courcelle's theorem only shows that there exists an $f(|\Phi|, w) \cdot O(n)$ algorithm to decide a graph property expressed by a MSOL formula Φ, we also need to ensure that the length of our MSOL formula Φ is constant or only depends on the parameter w in order to obtain a $f(w) \cdot O(n)$ algorithm. As a shorthand, we define the following predicate $card_k(S)$ which is true if and only if a set S has at least cardinality k:

$$card_k(S) \;:=\; \exists x_1, x_2, ..., x_k \in V \;.\; (\forall i \in [k] \;.\; x_i \in S) \;\wedge\; (\forall i, j \in [k] \;.\; i \neq j \implies x_i \neq x_j) \quad (4.1)$$

Here $[k]$ is the set of the first k positive integers and the universally quantified parts of the definition are just for notational convenience.

For a legal formulation in MSOL we have to write out these parts as $x_1 \in S \;\wedge\; x_2 \in S \;\wedge\; ... \;\wedge\; x_k \in S$ respectively $x_1 \neq x_2 \;\wedge\; x_1 \neq x_3 \;\wedge\; ... \;\wedge\; x_{k-1} \neq x_k$, resulting in a fomula length of $O(k^2)$ for the predicate $card_k(S)$.

Using this cardinality predicate we can now easily express the property that a graph contains a subgraph S which is a (2,t)-club in monadic second order logic:

$$\exists S \;.\; \forall v, w \in V \;.\; v \in S \wedge w \in S \implies (adj(v, w) \;\vee\; \exists N \subseteq V \;.\; card_t(N) \wedge \forall n \in N \;.\; adj(v, n) \wedge adj(w, n)) \quad (4.2)$$

This formulation has two problems. First, by using the predicate $card_t(N)$ its length depends quadratically on t (the remaining parts of the formula have constant length). Secondly, asking whether a graph contains *some* (2,t)-club as a subgraph is not what we were after and always trivially true due to the null graph (empty graph on zero vertices) which is trivially a (2,t)-club.

To address the first problem, recall Observation 2.2, which stated that every (2,t)-club which is not a clique must have minimum degree t. Therefore, a (2,t)-club which is not a clique can only occur in graphs with degeneracy at least t. Furthermore, a clique is a (2,t)-club for any value of t. Consequently, on a graph with degeneracy at most d all problem instances with $t > d$ are equivalent to each other as the solution can only be a clique. The degeneracy of a graph is upper-bounded by its treewidth. In all problem instances where t exceeds the treewidth w of the input graph we can set t to $w + 1$ without changing the problem. Alternatively, if $t > w$ we can just note that then (2,t)-Club is equivalent to Clique, which is known to be fixed-parameter tractable with regard to treewidth. Therefore, we only need to consider problem instances where $t \leq w$ and the length of our formula is consequently in $O(w^2)$.

Regarding the second problem, we need to state that we are looking for a subgraph S of cardinality at least k in order to arrive at a MSOL formula equivalent to the (2,t)-Club problem. If we would use the cardinality predicate $card_k(S)$ as defined before, the length of our formula would now depend quadratically on k and Courcelle's theorem would only provide us with fixed-parameter tractability with regard to w *and* k. Arnborg et al. [Arn+91] have shown that the results of Courcelle's theorem extend to something which they call *linear EMS extremum problems* [1], which allows us to assign integer values to the elements of V (in the case of graph problems like (2,t)-Club) and to demand for free set variables to be optimized (maximized or minimized) with regard to the sum of values of their elements. Therefore, the following formula which is equivalent to the optimization version of (2,t)-Club describes a linear EMS extremum problem:

$$\begin{aligned} &\max |S|,\ s.t.\ \forall v, w \in V\ .\ v \in S \wedge w \in S\ \Longrightarrow \\ &(adj(v, w)\ \vee\ \exists N \subseteq V\ .\ card_t(N) \wedge \forall n \in N\ .\ adj(v, n) \wedge adj(w, n)) \end{aligned} \tag{4.3}$$

Thus we have shown that (2,t)-Club is fixed-parameter tractable with regard to treewidth and can be solved in time $f(w) \cdot O(n)$ on graphs with treewidth w. □

It is important to note that this is a classification result only, as $f(w)$ is a non-elementary function (unless P = NP) and will be too large to make the algorithm practically feasible.

[1] called LinEMSOL(τ) optimization problems in Courcelle et al. [Cou+00]

Chapter 5

Implemented Algorithm and Experimental Results

In this chapter we provide a basic description of an algorithm we have implemented for solving (2,t)-Club and provide experimental results for solving the problem using our algorithm. We will show that our algorithm provides reasonably good performance and is state-of-the-art for solving the 2-Club problem on large real world graphs which yield easily to data reduction. We will further showcase the results for various real world graphs and social networks, discussing whether (2,t)-Club might provide a better model for cohesive subgroups than 2-Club on these graphs. The Java source code of our algorithm can be downloaded from http://fpt.akt.tu-berlin.de/two_t_club/.

5.1 Implemented Search Tree Algorithm for Finding (2,t)-Clubs

In this section we describe the algorithm we implemented for solving (2,t)-Club instances. The algorithm is based on a search tree method similar to those of Bourjolly et al. [Bou+02] combined with data reduction rules similar to those of Hartung et al. [Har+15b]. Additionally our algorithm has data structures for maintaining agreeability information for all vertices and utilizes Turing kernelization to handle larger graphs. The basic idea of Turing kernelization is to reduce the original problem to a polynomial (in the input size) number of kernels, which are bound in size by some function of some parameter of the input and whose results can be easily combined into a result for the original problem. In our case we utilize our observations from Section 4.1 to decompose the input graph into n closed 2-neighborhoods, one for each vertex of the original graph. For a detailed description of Turing kernelization refer to the article by Binkele-Raible et al. [BR+12] or the book of Downey and Fellows [DF13]. We begin with the description of the basic underlying search tree method.

5.1.1 The Search Tree Method and Turing Kernelization

Most algorithms for the 2-Club problem utilize the following search tree method which was introduced by Bourjolly et al. [Bou+02]: While the input graph G is not a 2-club, find a vertex v such that $|N_2(v)|$ (the size of its 2-neighborhood) is minimum among all vertices. Then branch into the cases of either deleting v or marking it (to be contained in the sought 2-club). When marking a vertex v all other vertices not contained in its 2-neighborhood $N_2(v)$ will be deleted. During branching we maintain the size ℓ of the largest 2-club found so far as a lower bound and abort a branch if the size of the graph is ℓ or less. For 2-Club the lower bound can be initialized with the maximum degree of the graph plus one, as the closed 1-neighborhood of any vertex is a 2-club. If at any point the closed 2-neighborhood contains the whole graph for all vertices, we have found a 2-club. As observed by Chang et al. [Cha+13], marking a vertex always forces at least one vertex deletion, leading to a branching vector of $(1, 2)$ in the worst case and $O^*(\alpha^n)$ running time, where $\alpha = 1.618$ is the golden ratio.

This search tree algorithm can be modified as follows to solve $(2,t)$-Club instances: Let the agreement set $N_2^t(v)$ for v denote the set of all vertices adjacent to v or sharing at least t common neighbors with v. While the input graph G is not a $(2,t)$-club, find a vertex v for which $|N_2^t(v)|$ is minimum. Then branch into either deleting v or marking v. When marking v, all other vertices not contained in $N_2^t(v)$ will be deleted. If at any point $N_2^t(v) = V \backslash v$ for all vertices v, we have found a $(2,t)$-club. Again we maintain the size of the best solution found so far as a lower bound for the size of solutions we are looking for in order to abort branches where the graph becomes too small to yield a better solution. The upper bound $O^*(1.618^n)$ on the running time as observed by Chang et al. [Cha+13] also applies for the $(2,t)$-Club search tree, as we remarked earlier in Corollary 2.15.

To make our algorithm feasible on large graphs with many vertices we utilize Turing kernelization based on our observations from Section 4.1. For each vertex v of the original graph, we construct a Turing kernel consisting of the closed 2-neighborhood $N_2[v]$ of that vertex, as every $(2,t)$-club is fully contained in the closed 2-neighborhood of some vertex due to $(2,t)$-clubs having diameter ≤ 2. We then run the search tree method described above on each of the generated Turing kernels. For each Turing kernel we initially mark the vertex v from which we derived the kernel, because if v were not contained in an optimal solution, we had no need to check the Turing kernel for v. Additionally, after solving the Turing kernel for v we remove v from the graph and all subsequent Turing kernels, because if v were part of an optimal solution, we would have found this solution on the kernel for v. Also, we discard any Turing kernel not larger than the current lower bound. The

largest (2,t)-club of the input graph is then simply the maximum among the largest (2,t)-clubs of each individual kernel.

As for an initial lower bound, we use maximum degree plus one for $t = 1$ as described above. For $t \geq 2$ we use no initial lower bound, that is we set it to zero. This is because we know of no lower bounds which are easy to compute and sufficiently strong, and our algorithm usually quickly finds some decent sized solution by itself and spends most of its time improving over near-optimal solutions or proving the found solution to be optimal. For $t = 1$ the maximum degree offers a cheap and quite strong bound (the size of the maximum 2-club is bounded by the square of the maximum degree). The analogon to the maximum degree lower bound for $t \geq 2$ would be the subset of t vertices with the largest common neighborhood (of at least t vertices). Finding this subset is already a computationally hard problem by itself for larger t. Moreover, experiments by us on generating (2,2)-clubs showed, that already for $t = 2$ the largest common neighborhood of two vertices in a (2,2)-club of 5000 vertices can be as small as 30, providing a very weak lower bound. In a further experiment we were able to construct a (2,2)-club with 3279 vertices where no two vertices have more than 16 common neighbors.[1]

If one does know the largest clique of the input graph (or any clique), this clique can be used as a lower bound. However, Clique itself is NP-hard and provides no quality guarantee for lower-bounding (2,t)-Club, as there can be arbitrarily large triangle-free (2,t)-clubs.

[1]Those experiments suggested that the possible maximum size of a (2,2)-club with all common neighborhoods (of any two vertices) bounded in size by b seems to depend at least exponentially on b. More specifically, for each increase of b by one we were able to find a (2,2)-club about 1.5 times as large as the largest one found for the bound $b-1$. The results were as follows, where b denotes the bound on the size of common neighborhoods and k the size of the largest (2,2)-club found obeying this bound among s (2,2)-clubs sampled:

b	s	k	b	s	k	b	s	k
2	3000	16	7	1000	90	12	100	678
3	3000	16	8	300	133	13	30	1048
4	3000	26	9	300	191	14	10	1533
5	1000	38	10	300	294			
6	1000	62	11	100	440	16	3	3279

We did not sample any graphs for $b = 15$ (no specific reason). The graph for $b = 2$ and $b = 3$ is the Clebsch graph, also known as Greenwood-Gleason graph. As a final side note, the Berlekamp-van Lint-Seidel graph (see e.g. http://mathworld.wolfram.com/Berlekamp-vanLint-SeidelGraph.html) is a strongly regular graph with 243 vertices and parameters $\lambda = 1$ and $\mu = 2$, therefore it is a (2,2)-club of size 243 where no two vertices have more than 2 common neighbors!

Because of the absence of an initial lower bound, we check the individual kernels of the graph in order of increasing size, which seems to be a good idea even in the presence of a good initial lower bound. The idea is to keep the gap between the current solution size lower bound and size of the currently considered kernel as small as possible for all kernels considered, as this gap constitutes an upper bound on the depth of the search tree for any given kernel. To reduce overhead from re-evaluating the size of each kernel after every vertex deletion, we sort the kernels of the graph once at the beginning of the algorithm by the size of the 2-neighborhood of the respective vertices and stick to this order throughout the run of the algorithm, even though some kernels might become smaller than some of their predecessors due to vertex deletions. Due to using Turing kernelization and sorting the kernels our $O^*(1.618^h)$ bound on the running time for the h_2-index h of the input graph as described in Definition 4.2 and Theorem 4.3 applies. Investigating whether or when it might be worthwhile in practice to keep the list of kernel sizes sorted and up-to-date at all times (and thus achieving a running time $O^*(1.618^d)$ for N_2-degeneracy d as discussed in Section 4.2) might be an interesting direction for future work.

5.1.2 Data Structures

To further improve the search tree method, we apply various data reduction rules to shrink the original graph, as well as every Turing kernel both initially and after each deletion and marking performed. To perform the data reduction rules quickly the following information about the current kernel is held up-to-date at all times:

Agreeability matrix A matrix storing for each pair of vertices in the current kernel their agreeability, that is their number of common neighbors. Agreeability information is only relevant for non-adjacent vertices. For $t = 1$ a value of 0 for non-adjacent vertices is equivalent to having distance at least 3, while a positive value is equivalent to having distance at most 2.

Conflict set A set of vertex pairs containing all pairs of vertices which are not adjacent and do not share at least t common neighbors in the current kernel. Therefore, for each conflict (v, w) at least one of the two vertices v or w must be deleted in order to obtain a (2,t)-club. Although the agreeability matrix implicitly contains all conflicts (two vertices are in conflict if they are not adjacent and their value in the matrix is $< t$), we store conflicts explicitly in order to avoid scanning the entire matrix when the set of all conflicts is required. For $t = 1$ a conflict means that the respective vertices have distance ≥ 3 to each other.

Agreements vector A vector containing for each vertex v of the current kernel the number of its agreements, that is the number of vertices *not* being in conflict with v. The vector therefore contains for each vertex the size of its agreement set as described above for the search tree method. Among other things this vector allows a fast (linear in current kernel size) look-up of the vertex of minimum agreement, which is used as the next branching candidate. For $t = 1$ the entries in the agreements vector correspond to the size of the 2-neighborhood of the respective vertex.

Initialization of these data structures requires $O(m\Delta + n^2)$ time, where n is the size of the kernel, m the number of edges in the kernel and Δ the maximum degree of the kernel. We first initialize the agreeability matrix with all zero values and then for each vertex v of the kernel increase the entry of each pair of neighbors of v by one. Afterwards a single pass over the agreeability matrix suffices to count the number of agreements for each vertex and identifying every initial conflict. The initialization cost can be quite large for dense kernels with many vertices, so we apply all reduction rules we can use without this information first and discard the kernel before initialization if it becomes too small. To keep the information up-to-date at all times, for every vertex v we delete we must

1. Lower the agreements of all vertices adjacent to v by one, which can be done in $O(d)$ time, where d is the degree of v.

2. Lower the agreements of all vertices not adjacent to and agreeable to v by one, which can be done in $O(n)$ time, where n is the current size of the kernel.

3. Reduce agreeability for all pairs of neighbors of v by one, which can be done in $O(d^2)$ time, and if any non-adjacent pair $\{u, w\}$ of neighbors of v then has agreeability exactly $t - 1$ we must lower the agreements for u and w by one and introduce a conflict for them.

This leads to a $O(n + d^2)$ overhead for every delete operation to keep information up-to-date, which is quite significant but enables us to perform our data reduction rules quickly and extensively whenever they apply and as soon as they apply.

5.1.3 Data Reduction Rules

The data reduction rules used by us are mostly generalizations of the data reduction rules for 2-Club used by [Har+15b] and are as follows:

R1: Marked Conflict Rule If at any time two vertices are both marked and in conflict, then abort the branch.

R2: Low Degree Rule Remove vertices of degree $< t$. If such a vertex was marked, then abort the branch. For $t = 1$ delete all vertices of degree one. (This reduction rule is not universally correct, please note our explanations of the rules below.)

R3: Conflict Resolution Remove all vertices which are in conflict with any marked vertices.

R4: Low Agreement Rule Remove vertices whose number of agreements is lower than the current lower bound. If this rule leads to the deletion of some marked vertex, abort the branch.

R5: Vertex Cover Rule Let $G = (V, E)$ be the current state of the kernel. Consider the conflict graph $G_C = (V, E')$, where $\{v, w\} \in E'$ if and only if v and w are in conflict. Now, the size of a minimum vertex cover of G_C is a lower bound on the number of vertices in G that still must be deleted in order to obtain a $(2,t)$-club. Use the simple and fast 2-approximation for Vertex Cover, where we repeatedly put both endpoints of some uncovered edge into the cover until all edges are covered. If the current size of G minus half the size of the obtained vertex cover does not exceed our current lower bound, we abort this branch.

R6: No Choice Rule If at any time two non-adjacent marked vertices have exactly t common neighbors, mark all of their neighbors which are not marked yet. If this is not possible, abort the branch.

The correctness of R1 and R3 is obvious, as a $(2,t)$-club by definition cannot contain vertices in conflict. Regarding R2 two important points must be made: First, deleting degree one vertices for $t = 1$ is only correct when initializing the algorithm with the closed 1-neighborhood of the maximum degree vertex as initial solution. A 2-club containing a degree one vertex can only consist of the closed 1-neighborhood of the vertex adjacent to it. Therefore all solutions better than the one obtained from the closed 1-neighborhood of the maximum degree vertex cannot contain degree one vertices. Second, deleting vertices of degree $< t$ is only correct to the extent that only "trivial" solutions for $(2,t)$-Club contain vertices of degree $< t$. Obviously when a vertex has degree $< t$ it cannot be agreeable to any

non-adjacent vertex as it cannot have t common neighbors with it. However cliques of size at most t are by definition (2,t)-clubs and have vertices of degree $< t$. Therefore when our algorithm finds no solution, it means that the solution is some clique of size $\leq t$. The omission of small cliques as solution is motivated by the fact that R2 is a powerful reduction rule, especially for larger t, on most graphs. Secondly, small cliques might be uninteresting solutions in many settings and one can always run a clique algorithm on the graph afterwards, if one is interested in the largest clique of the graph.[2] We can use R2 extensively both on the whole graph as well as every kernel individually, both initially and whenever the degree of a vertex falls below t due to vertex deletions.

The correctness of R4 stems from the fact that a (2,t)-club containing some vertex v can only contain vertices which are agreeable with v, so if v currently has less agreements than the current lower bound, all (2,t)-clubs containing v contain no more vertices than the current lower bound.

R5 is another strong reduction rule, but with significant cost. We therefore use the following observations to reduce the number of necessary applications of the Vertex Cover Rule (by roughly one third on average): Whenever we have created no new conflicts since the last application of the VCR, the last result of the VCR is still a valid upper bound for the required number of vertex deletions. If this last result is not enough to shrink the kernel to or below the lower bound, there is no need to check the VCR again. If it is enough we need to recheck the VCR as the number of conflicts might have decreased due to vertex deletions. If we have instead created X new conflicts since the last application of the VCR, a new application of the VCR can at most report X more required deletions over the last result of the VCR. So if the last result plus X does not shrink the kernel enough, there is no need to check the VCR again. Again, if the last result plus X does shrink the kernel enough, we need to check the VCR again as deletions might have resolved conflicts and X additional required deletions is likely to be an overestimation. To further reduce the number of VCR applications, we remember the last vertex cover of the conflict graph and only count those conflicts as new which are not covered by this last vertex cover.

It should be noted again that we use all our data reduction rules as soon as they apply, that is especially during delete and mark operations on the kernel we immediately collect all vertices which are subject to data reduction due to the deletion and marking and perform recursive markings and deletions

[2] That is exactly what our implementation of the algorithm does and we found out that a Clique algorithm usually runs much faster than our (2,t)-Club algorithm on the real world graphs considered.

according to the reduction rules immediately afterwards. This spares us from scanning over all vertices for data reduction in each branch after the initial data reduction pass on the whole graph and each kernel. The only reduction rule applied directly on search tree branches is R5, all others are applied once when initializing the kernel and otherwise only during delete and mark operations as they become applicable.

5.2 ILP formulation of the (2,t)-Club problem

Integer linear programming is a powerful tool which enables us to formulate a wide variety of combinatorial optimization problems in a concise form known as integer linear program (ILP). There exist several highly developed solvers for ILPs which can quickly find solutions for instances of a computational problem expressible as an ILP. In developing an algorithm for a problem, comparison with the output of the ILP solver can provide a benchmark for the performance of the own algorithm. Also, comparison of solutions reported by the ILP solver and the own algorithm might help to spot incorrect behaviour of the own algorithm leading to wrong solutions. As (2,t)-Club is a new problem with no prior algorithmic implementations, save for an ILP for a very similar problem by Veremyev and Boginski [VB12], we want to utilize a state-of-the-art ILP solver (Gurobi) as benchmark for our algorithm on instances with $t \geq 2$. To this end we now develop an ILP formulation for the (2,t)-Club problem, which we will later use in our performance tests in Section 5.3.4. Given a graph $G = (V, E)$ and an integer t the (2,t)-Club problem can be solved with the following integer linear program:

Introduce a 0/1-variable v_i for each vertex i of the graph G. Setting $v_i = 1$ indicates $i \in S$, where S is the solution set of vertices, i.e. the set of vertices constituting the maximum (2,t)-club. For each non-adjacent pair of vertices u and w introduce the constraint $t \cdot v_u + t \cdot v_w - \sum_{i \in N(u) \cap N(w)} v_i \leq t$, which forces us to either take at most one of the two vertices u and w into the solution or add at least t of their common neighbors, too. Now maximize $\sum_{i \in V} v_i$ subject to these constraints.

This formulation is a generalization to (2,t)-Club of the ILP formulation for the 2-Club problem as it can be found for instance in Balasundaram et al. [Bal+05] or Carvalho and Almeida [CA11].

5.3 Experimental Results

Now that we have described the basic principles behind our algorithm, we want to present a number of experimental results regarding the performance of our algorithm and the kind of solutions the problem yields for real world graphs representing social networks. We will start by describing the experimental setting and the test instances we used to perform our experiments.

5.3.1 Experimental Setting

All algorithms that we ran for our experiments were executed on a notebook with Intel Core i5-4200U 1.6GHz processor and 8 GB RAM running the Ubuntu 14.04 operating system, save for the VNS algorithm by Shahinpour and Butenko [SB13] for which we only cite the results. Our algorithm was implemented in Java and we ran both it and the algorithm by Hartung et al. [Har+15b] under the OpenJDK runtime environment version 1.7.0_65. We instructed the JVM (Java Virtual Machine) to allow both algorithms to allocate up to 6 GB of memory. We compiled the algorithm by Chang et al. [Cha+13] with *gcc* version 4.8.2. Finally, we used the Java API of Gurobi version 6.0 to solve ILP formulations of (2,t)-Club instances.

5.3.2 Test Instances

We tested our implementation of the described (2,t)-Club algorithm on random instances as well as real world instances from the DIMACS implementation challenges and the Stanford Network Analysis Project (SNAP). To generate random instances we use three different random graph generation schemes: The extension to the basic Erdős-Rényi model proposed by Gendreau et al. [Gen+93], the Barabási-Albert model introduced by Barabási and Albert [BA99] and a simple scheme for generating random (nearly) regular graphs.

Gendreau model. In the basic Erdős-Rényi model one specifies a number of vertices n and an edge density p and the generation method then includes each possible edge between the n vertices with probability p each. In the extended scheme proposed by Gendreau et al. one specifies two density parameters a and b with $a \leq b$ instead and the generation method then assigns each vertex a density value picked uniformly at random from the interval $[a, b]$. The probability for each possible edge to be present is then the arithmetic mean of the density values of its both endpoints. For $a = b$ the Gendreau model is equivalent to the Erdős-Rényi model with $p = a = b$. For a given size n of the graph and parameter t of the (2,t)-Club problem experiments show that there is a critical density p for which the instances become hardest. For $t = 1$ and a graph size between about

100 and 150 vertices, Bourjolly et al. [Bou+02] identified this critical density to be around 0.15. Therefore, previous algorithms for the 2-Club problem have been experimentally tested predominantly for graphs generated with the Gendreau method where $\frac{a+b}{2} = 0.15$ for various intervals ranging from $[0, 0.3]$ to $[0.15, 0.15]$. Instances generated using this model turn out to be hardest when the density is fixed, that is $a = b$.

Barabási-Albert model. The Erdős-Rényi model generates graphs where the vertex degrees are very close to $p \cdot (n-1)$ and significant deviations from this expected degree are unlikely. Even in the Gendreau model with $a = 0$ the expected degree of a density 0 vertex only differs by factor 3 from the expected degree of a maximum density vertex and significant deviations from expected degree are unlikely. In contrast, many real world graphs, in particular social networks, have a wide range of vertex degrees including some high degree vertices and many low degree vertices. Social networks are often observed to follow a power law of degree distribution, where the probability of a randomly picked vertex having a given degree d is roughly proportional to $d^{-\gamma}$ for some parameter γ usually between 2 and 3 for many networks. Such networks are commonly called *scale-free networks*, see e.g. Barabási and Albert [BA99] for details.

Barabási and Albert [BA99] proposed a random graph model which produces random graphs exhibiting such degree distributions by modelling *growth* and *preferential attachment*, the two key features for scale-free networks identified by the authors. Graphs generated by the Barabási-Albert model can therefore be expected to be considerably closer to the shape of real world social networks than those generated by the Gendreau model. The Barabási-Albert model generates graphs as follows: Starting from some connected graph on k vertices we iteratively add one vertex to the graph (therefore: *growth*), connecting it to d vertices already present, where the probability of each previous vertex to be picked as a neighbor of the new vertex is proportional to its current degree (therefore: *preferential attachment*). We proceed adding vertices until the desired number n of vertices of the graph has been reached. In our experiments we use size k-cycles as seed for our graphs and then add $n - k$ vertices each introducing d new edges between the new vertex and the old ones. The use of cycles as seed graphs is motivated by the fact that we want to avoid forming overly dense clusters of vertices surrounded by sparse protuberances, which might make the instances nearly trivial.

Random regular model. As a third kind of random instances we generate random (nearly) regular graphs, using the following method: Given a number n of vertices and a desired degree d we start out from an edgeless graph on n vertices. Now we repeatedly pick a vertex of minimum degree

and pick a second vertex uniformly at random from the set of vertices not adjacent to the first vertex and having smallest degree (among all vertices not adjacent to the first one). We then introduce an edge for the picked pair and repeat until we have generated $\frac{n \cdot d}{2}$ edges. Graphs generated using this model are not guaranteed to be regular, as at the end of the generation algorithm we might be left with some vertices of degree $d-1$ which are already adjacent to each other, forcing us to make them adjacent to vertices already having degree d, thus leading to some vertices having degree $d-1$ or $d+1$. However the probability that more than one pair of vertices violates the desired degree is small, if $\frac{n}{d}$ is sufficiently large (like about 5 in our experimental instances). The probability of the generated graph being regular can be roughly estimated at about $1 - \frac{d}{n}$. Of course, either n or d is required to be even for the generation method to work. Like for the Erdős-Rényi model where we have a critical density p for given n and t, the random regular model has a critical degree d for given n and t, where the instances are hardest. For a given size n of the graph random regular instances at critical degree d turn out to be significantly harder than Gendreau/Erdős-Rényi instances at critical density p, which is our motivation to include the random regular model besides the traditionally used Gendreau model in our experiments. There are more sophisticated methods for generating random regular graphs, especially ones guaranteeing regularity, see for example Wormald [Wor99]. However, for our purposes the described scheme for generating nearly regular graphs is sufficient and conveniently easy to implement.

Real world instances Finally we tested our algorithm on a number of real world graphs taken from:

- The *10th DIMACS Implementation Challenge*, see http://www.cc.gatech.edu/dimacs10/
- including graphs from Chris Walshaw's *Graph Partitioning Archive*, see: http://staffweb.cms.gre.ac.uk/~wc06/partition/
- as well as graphs from the *Stanford Network Analysis Project (SNAP)*, see: http://snap.stanford.edu/

From SNAP and the 'Co-author and Citation Networks' category of the 10th DIMACS challenge we obtained real world social network graphs to analyze the maximum (2,t)-clubs found on these graphs for various t. From the 'Clustering' category of the 10th DIMACS challenge and Chris Walshaw's Graph Partitioning Archive we obtained a collection of graphs which have been repeatedly used for testing implementations for 2-Club and related problems in the past. Last but not least, we have used a graph derived by Hartung et al. [Har+15b] from citation data, which they have used in their experiments.

5.3.3 Comparison with Existing 2-Club Implementations

We used our algorithm and the algorithm by Hartung et al. [Har+15b] to solve the 2-Club problem on random as well as real world instances. Our findings show that the Hartung algorithm outperforms our algorithm on most random instances by a factor usually around 2 to 3 on average, while our algorithm consistently outperforms the Hartung algorithm on real world instances, often by multiple orders of magnitude. Presumably our more extensive data reduction approach works well on real world instances, where much of the work is usually data reduction instead of search, while the overhead introduced by the more extensive data reduction, especially the cost per operation to keep agreement information up-to-date, outweighs the benefit on dense random instances where data reduction usually yields significantly less improvement and heavy searching is required.

Some remarks concerning the following parts and figures: The algorithm implemented by us is named RTC (short for *RobustTwoClub*) in the following; Δ denotes the maximum degree of the input instance. Both our algorithm and the algorithm by Hartung et al. [Har+15b] can run with or without Turing kernelization as described above and we usually disabled it where not required as particularly our algorithm can slow down significantly when kernelization is activated where not needed. For our algorithm we use the terms *kernel* and *hard kernel* in the following way: A *kernel* is a kernel derived from Turing kernelization which cannot be dismissed due to too small size. A *hard kernel* is a *kernel* which does not yield to applying data reduction rules alone but requires some run of the search tree method to branch over possible solutions. The count for kernels does include hard kernels.

Real World Instances

First we compare both algorithms on real world instances from the 10th DIMACS implementation challenge, the results can be seen in Figure 5.1. The first half of the table includes a set of instances that has been popular for testing 2-Club instances in the past. Amongst others Shahinpour and Butenko [SB13], Hartung et al. [Har+15b] and Wotzlaw [Wot14] have tested their implementation on this collection of instances or a subset of it. McCreesh and Prosser [MP14] also tested most of these instances for their implementation of a 2-Clique algorithm. Compared to Shahinpour and Butenko [SB13] we left out one instance from the test set which had an other input format than the remaining graphs and could not be interpreted correctly by the Hartung algorithm. The last four instances in the second part of the table are three further graphs from the 10th DIMACS implemen-

Instances					**RTC**			**Hartung**		**Chang**	**VNS***
name	n	m	Δ	2-club size	time	kernels	hard kernels	time	kernels	time	time
karate	34	78	17	18	0.00	0	0	0.00	0	0.00	1.11
dolphins	62	159	12	13	0.00	14	0	0.00	1	0.00	1.544
polbooks	105	441	25	28	0.00	18	2	0.04	12	0.01	12.324
adjnoun	112	425	49	50	0.00	0	0	0.00	0	0.01	70.370
football	115	613	12	16	0.03	79	9	1.23	67	0.06	5.710
jazz	198	2742	100	103	0.42	29	1	0.59	1	0.12	2012.920
celegans_metabolic	453	2025	237	238	0.02	0	0	0.14	0	0.02	3603.315
email	1133	5451	71	72	0.23	305	0	113.51	295	3.49	273.983
polblogs	1490	16715	351	352	3.56	191	0	70.79	48	23.61	3872.459
netscience	1589	2742	34	35	0.01	0	0	0.04	0	0.08	70.450
add20	2395	7462	123	124	0.07	0	0	0.64	0	0.46	1957.428
data	2851	15093	17	18	0.17	935	25	—	—	10.11	260.785
uk	4824	6837	3	5	0.04	1060	2	425.09	1052	23.23	392.106
power	4941	6594	19	20	0.03	0	0	0.20	0	0.74	480.024
add32	4960	9462	31	32	0.05	0	0	0.43	0	0.78	567.450
hep-th	8361	15751	50	51	0.06	0	0	0.72	0	4.57	1825.316
whitaker3	9800	28989	8	9	0.18	3889	0	311.70	2061	—	2320.600
crack	10240	30380	9	10	0.37	3916	75	430.31	2745	—	2452.326
PGPgiantcompo	10680	24316	205	206	0.19	0	0	1.02	0	4.33	5183.705
cs4	22499	43858	4	6	0.33	11721	6	697.51	6601	—	11371.810
citationCiteseer	268495	1156647	1318	1319	25.85	0	0	61.55	0		
coAuthorsCiteseer	227320	814134	1372	1373	12.07	0	0	36.29	0		
coAuthorsDBLP	299067	977676	336	337	17.52	0	0	69.33	0		
graph_thres_01	715633	2511988	804	805	92.13	0	0	240.54	0		

Figure 5.1: Comparison of running times (in seconds) for the different algorithms for various graphs from the DIMACS implementation challenges. *The results for VNS were taken from Shahinpour and Butenko [SB13].

tation challenge and a special graph[3] used by Hartung et al. [Har+15b] for their experiments. For all instances we also ran the algorithm by Chang et al. [Cha+13], which provided good results for random instances but hasn't been extensively tested on real world instances so far. We had to convert the instances into a format readable by the Chang algorithm first, but this should have no bearing on the results. Finally, we cite the results by Shahinpour and Butenko [SB13] (for their algorithm called VNS). Their experimental setup can be found in their paper. It is important to note that their algorithm did not guarantee optimal solutions, but did find one in all instances except 'cs4'. We used Turing kernelization for both our algorithm and the Hartung algorithm, even though most of the smaller graphs do not require it. Where Figure 5.1 shows no results for the Hartung algorithm or the Chang algorithm, the respective algorithm ran out of memory and had to be aborted.

From the results in Figure 5.1 we can draw a number of conclusions. First, our algorithm seems to be the state-of-the-art algorithm when it comes to solving 2-Club on sparse real world instances, although the Chang algorithm provides reasonably good running times across the board, too. The Hartung algorithm runs fast on most instances, but seems to have significant trouble with some of the instances. The major obstacle for the Chang algorithm for solving larger real world graphs seems to be the lack of Turing kernelization, which could be easily amended.

Secondly, we observe that for most graphs tested, the largest 2-club has size of the maximum degree plus one. This both makes the solutions quite uninteresting from the viewpoint of finding cohesive subgroups in graphs as mentioned earlier and makes our fast running time results less spectacular than they might first appear as we only have to verify a trivially found solution in most instances. The last point is clearly seen by observing that RTC reports nearly no hard kernels for the considered problem instances, therefore nearly all instances could be solved using polynomial-time data reduction alone. We conclude that these instances are of questionable use for benchmarking 2-Club implementations and that 2-Club is a rather pointless model (at least in its maximization variant) for cohesive subgroups on most social networks and other real world graphs.

We compared our algorithm with the Hartung algorithm on further instances from Chris Walshaw's Graph Partitioning Archive. Refer to Figure A.1 in the appendix. The results for these instances both confirm that 2-Club is rather pointless on sparse real world instances (producing uninteresting trivial solutions) and the advantage of our algorithm at recognizing this fact reasonably fast.

[3] see Hartung et al. [Har+15b] for details

Random Instances

We further tested our algorithm and the Hartung algorithm on randomly generated instances. We omitted comparison with other algorithms as the Hartung algorithm seems to be state-of-the-art for solving 2-Club on random graphs generated by the Gendreau model. A comparison with the also fast algorithm by Chang et al. [Cha+13] has already been done by Hartung et al. [Har+15b]. Our findings have shown that the Hartung algorithm outperforms our algorithm on hard random instances nearly always, not just on average but for nearly every instance. The running time factor between both algorithms is, however, only about 2 to 3 on average and on the vast majority of individual instances, meaning the relative running times fluctuate little. This is probably due to the fact that for 2-Club our algorithm is basically just a variant of the Hartung algorithm, using to a large degree essentially the same strategies and data reduction rules and mostly differing in how extensively data reduction (and thus at how much cost in additional polynomial overhead) is applied.

We generated instances for each of the three models described in Section 5.3.2. The results for each random graph model can be found in the appendix: Figure A.2 contains the results for the Gendreau model, Figure A.3 the results for the random regular model and Figure A.4 the results for the Barabási-Albert model. We ran both algorithms without utilizing Turing kernelization as it wasn't needed and the generated random instances are both too small and dense to make Turing kernelization yield any benefit.

For the Gendreau model we went with average density 0.15 which has been identified by Bourjolly et al. [Bou+02] as a particularly hard density for 2-Club for graphs of size about 100 - 150. We tested both algorithms for different density variances from [0.00, 0.30] to [0.15, 0.15] and different graph sizes for each pair of density parameters. For each combination of density parameters and graph size we generated 100 test instances and averaged all values over them. We note average edges, average maximum degree and average 2-club size for all combinations and give average running times as well as minimum and maximum running time for both algorithms. We also note the average number of search tree branches reported by our algorithm and the average number of recursions reported by the Hartung algorithm.

From the results in Figure A.2 we first see that the 2-Club problem gets harder when the variance of density decreases. We further see that the Hartung algorithm quite constantly is a little more than twice as fast as our algorithm, both in average as well as maximum running time. Furthermore, the reported branches and recursions are roughly the same for both algorithms. Although we are not entirely sure that the recursions reported by the Hartung algorithm are defined and computed the same way as the number of branches in our algorithm, the fact that our algorithm nearly

always reports less branches might suggest that a little more data reduction takes place. However, the additional computational overhead for applying these reductions seems to outweigh the benefit on these random instances.

For the random regular model we tried to generate nearly regular graphs for various combinations of graph size and desired degree. Again we generated 100 instances for each combination of parameters and averaged the results over them. For the random regular model the number of edges is fixed for a given graph size and desired degree, however the maximum degree fluctuates slightly as our method of generating random regular graphs is not entirely perfect as described in Section 5.3.2.

From the results in Figure A.3 we see that the random regular model seems to produce significantly harder instances than the Gendreau model for any given graph size. Especially for the larger graph sizes we also see that there is such a thing as a critical degree for the problem on a given graph size, for which it is hardest. The performance results for this different model are consistent with our results for the Gendreau model: The Hartung algorithm is constantly about twice as fast, both on average and in maximum running time. On the other hand, our algorithm reports less branches on average for all combinations of graph size and degree. This reinforces our impression that our algorithm performs more effective data reduction, whose gain is outweighed by the required overhead.[4]

Last we tested both algorithms for the Barabási-Albert model, where we repeatedly add a new vertex to a fixed number[5] of previous vertices with probability proportional to the current degree of the individual previous vertices. The idea behind the Barabási-Albert model was to generate random instances which are reasonably alike to real world social networks. Therefore we wanted to generate larger and sparser graphs with a wide spread of vertex degrees. Our results in Figure A.4 for the Barabási-Albert model tell us more about the 2-Club problem itself than about comparing both algorithms. We quickly found that we could only achieve reasonable running times for larger graph sizes if the degree (which in this context means the degree of a newly added vertex as we add it) was kept relatively small. However, when we kept the degree small the maximum 2-club *always* turned

[4]The relatively small factor of about 2 could also be due to many other implementation details. But as we believe our algorithm to be relatively optimized with regard to minor implementation details, attributing the difference in performance to data reduction overhead seems to be a reasonable assumption. Especially as our algorithm is actually faster than the Hartung algorithm when both algorithms have to perform only data reduction, see those results in Figure 5.1 and Figure A.1 where both algorithms report zero kernels. Note that when we talk about aditional data reduction overhead we mean data reduction overhead per node in the search tree during the branching phase of the algorithms here, not overhead due to initial data reduction before starting branching.

[5]denoted as 'degree' in Figure A.4

out to be just the maximum degree vertex and its neighbors, as can be seen in the figure. So we have yet more evidence that with 2-Club we are stuck between a rock and a hard place for larger graphs. Either the graph is (roughly speaking) far too dense to solve the problem in reasonable time, or it is far too sparse to produce anything but trivial solutions as largest 2-club, i.e. the maximum degree vertex and its neighbors. As a side note, for larger n (while keeping the degree small) our algorithm seems to overtake the Hartung algorithm, presumably because these much sparser graphs allow for more data reduction and the portion of running time spent on data reduction becomes larger.

Concluding Thoughts on the Comparison of the Algorithms

In conclusion we can say that our algorithm provides a state-of-the-art solution for the 2-Club problem on large instances which yield well to mere data reduction, while retaining a certain competitiveness on hard random instances. The more extensive data reduction which allows us to solve real world instances substantially faster also introduces an overhead which makes our algorithm a good bit slower on random instances, but without such drastic differences as observed on the real world graphs. Sepp Hartung remarked that the algorithm by Hartung et al. [Har+15b] was especially geared towards solving random instances generated by the Gendreau model, while our algorithm was designed from the outset with large real world graphs in mind. The results given above seem to reflect these different design goals. As our algorithm was meant primarily as an algorithm for the more general (2,t)-Club problem and less as a new algorithm for solving 2-Club, the question arises whether our seemingly good performance for the case $t = 1$ transfers to the general case. To this end, in the following section we compare our algorithm to solving the ILPs for (2,t)-Club obtained from the formulation in Section 5.2 using the popular state-of-the-art ILP solver Gurobi.

5.3.4 Comparison with ILP for Random (2,t)-Club Instances

In absence of existing algorithms for (2,t)-Club we used the ILP formulation described in Section 5.2 and implemented it through the Java API of Gurobi version 6.0 to benchmark the performance of our algorithm for $t \geq 2$. We used the Gendreau random graph model to generate small test instances and ran both our algorithm and Gurobi on them. For both graph sizes n tested (70 and 85) we generated 100 instances for each value of t from 1, 2, 4 and 8, as well as 100 instances for $n = 70$ and $t = 16$. For each combination of n and t we picked density values that generated reasonably hard instances, which usually occurs somewhere around where the average solution size is about half the graph size. We still kept the den-

Instances						**RTC**			**ILP**		
n	t	density	avg. m	avg. Δ	avg. 2-club size	T_{avg}	T_{min}	T_{max}	T_{avg}	T_{min}	T_{max}
70	1	[0.125, 0.25]	451.80	21.70	32.88	0.13	0.03	0.40	0.66	0.06	1.28
	2	[0.2, 0.3]	599.27	26.20	34.40	0.22	0.05	0.85	1.12	0.09	5.11
	4	[0.25, 0.4]	784.98	32.62	34.33	0.46	0.05	2.06	2.89	0.10	21.03
	8	[0.4, 0.5]	1084.06	41.01	47.59	0.76	0.01	4.07	4.34	0.05	29.53
	16	[0.55, 0.6]	1392.04	49.54	25.08	0.96	0.02	3.34	5.95	0.06	17.87
85	1	[0.125, 0.225]	627.90	24.39	36.52	0.61	0.14	1.52	1.77	0.70	5.92
	2	[0.2, 0.28]	860.44	30.38	44.92	1.11	0.07	2.96	4.13	0.12	21.98
	4	[0.26, 0.34]	1072.81	36.54	34.35	3.06	0.15	13.99	31.22	0.88	127.99
	8	[0.38, 0.44]	1464.38	45.64	40.50	5.83	0.03	26.51	33.78	0.08	176.12

Figure 5.2: Running time comparisons for our algorithm and the ILP formulation for (2,t)-Club for various t on graphs generated using the Gendreau model. For each combination of n and t we determined density parameters which gave appropriately hard and mostly non-trivial instances and averaged running times (in seconds) for both approaches over 100 generated instances.

sity parameters variable (as opposed to the basic Erdős-Rényi model) as we wanted to generate instances of varying hardness. Towards higher t the size of the solution becomes increasingly sensitive to the average density of the graph with small increases in density often resulting in change from nearly all solutions being small cliques to nearly all solutions containing the entire graph. Therefore, we decreased the density intervals for higher t to have less trivial instances in these cases. Figure 5.2 shows the results of our tests. We can see that our algorithm has a clear edge over the ILP solver on average. One should take the running time differences for varying parameters and especially the density parameters themselves with a grain of salt and not interpret too much into them, as we determined the density parameters experimentally and can not guarantee that we picked comparably hard parameters for each combination of n and t. Although we see that both approaches need increasingly more time the higher t gets, this can be to a large part due to the increasing density of the tested graphs (for increasing t). Therefore, we shouldn't necessarily read these results as indication that $(2,t)$-Club becomes essentially harder for increasing t.

We omit a comparison on real world graphs between our algorithm and the ILP formulation for the following reason. On most real world instances tested by us the process of finding the largest $(2,t)$-club often boils down to mere data reduction running in polynomial time. Furthermore, for a fair comparison both approaches should be tested using Turing kernelization for larger graphs, further introducing significant portions of the overall running time which are polynomial and would be identical for both approaches. We therefore would expect very similar running times for both approaches on instances where the problem becomes solvable by mere data reduction, similar to the results obtained by Hartung et al. [Har+15a] who obtained nearly the same running times for their algorithm and the ILP formulation for 2-Club on real world graphs.

5.3.5 The Search Tree Algorithm on Real World Instances

Next we want to provide some of our results on solving $(2,t)$-Club on various real world instances, this time not only from the viewpoint of algorithm performance, but also from the viewpoint of structure of solutions and behaviour of real world data for the problem. To this end we run our $(2,t)$-Club algorithm on six real world graphs derived from social networks, for all values of t.[6] Before we give a short overview over the graphs tested, we want to consider a few aspects we want to analyze in our experiments:

[6] For every graph there is a largest t for which the solution is not a clique. Once we have found a clique as solution for some t, we know that all solutions for $t' > t$ are also this clique.

First, the main motivation behind the definition of $(2,t)$-clubs was to avoid the uninformative solutions of 2-Club lacking in robustness. We therefore need to test whether $(2,t)$-Club provides better results. Of course, $(2,t)$-clubs are more robust by definition. Where one single dominating vertex can determine the solution for 2-Club and disconnect it when removed, for $(2,t)$-Club at least t vertices are required to determine a solution and only deleting at least t vertices from a solution might disconnect it. Still, although better than the usual solutions for 2-Club, we probably do not want these solutions where only t vertices determine the status of the solution, especially if t is comparably small. Such solutions would have a set of t vertices which are adjacent to all vertices not belonging to this set. We have given such graphs a name before in Definition 2.3: t-diamonds. Those t-diamonds are the generalisation of the star-like structures we often observe as solutions for 2-Club. We therefore check in the following experiments whether an obtained solution for $(2,t)$-Club is a t-diamond. We go even further and determine the *diamond number* d for each solution: The diamond number of a $(2,t)$-Club solution S is the largest $d \leq t$ such that S can be transformed into a $K_{d,n-d}$ by edge deletions. To avoid ambiguity the diamond number is bounded by $\frac{n}{2}$, where n means the size of the solution here. The restriction to $d \leq t$ is motivated by the fact that we check each solution for the largest t for which it is a $(2,t)$-club, so the diamond number cannot exceed t (otherwise the solution would be a $(2,t')$-club for some larger t'). We can consider a solution to be more meaningful if the diamond number does not equal t. Regarding the usefulness of the diamond number as an indicator for solution quality in practice, note that determining the diamond number of a graph is a hard computational problem in itself, which can make checking the solution for its diamond number take longer than finding the solution in the first place (at least for larger t). In our experiments we were always able to find the diamond number in reasonable time, but we did not include running times for these checks in the figures.

As another point of consideration, we did mention that we check the graphs for solutions for all values of t. It then would be interesting to see how the different solutions for different values of t, especially subsequent ones, would relate to each other. That is, are the solutions for larger values of t subsets of solutions for smaller values of t or do they at least overlap significantly? We therefore will check for each solution for some value of t the number of vertices that also belonged to the previous solution and the number of vertices which didn't. While this only gives a rough indication for the relation of solutions for different values of t to each other, it nonetheless provides us with some interesting results.

Finally, as we have seen that our algorithm reported nearly no hard kernels for solving 2-Club on real world graphs, we want to have a more detailed look at the running time behavior this time. In particular we want to know whether the more general $(2,t)$-Club normally requires heavy search-

	n	m	Δ	d	h	h_2	ω
coAuthorsCiteseer	227,320	814,134	1,372	7	114	1,372	87
coAuthorsDBLP	299,067	977,676	336	7	132	1,126	115
citationCiteseer	268,495	1,156,647	1,318	9	209	1,928	13
facebook_combined	4,039	88,234	1,045	44	164	1,045	69
cit-HepTh	27,770	352,285	2,468	25	192	3,120	23
soc-Slashdot0902	82,168	504,230	2,552	12	275	5,116	27

Figure 5.3: Table of instances used for testing the algorithm on real world data.

ing on real world graphs or if the polynomial-time data reduction does most of the work. Therefore, we will list the amount of time spent on the search tree phase separately in addition to the overall running time. Furthermore, we will not only list the number of branches in the search tree for every instance, but also the number of operations (marking and deleting vertices) during the search tree phase. The quotient of overall operations divided by branches should give us some indication regarding the amount of data reduction happening during the search tree phase, as every operation after the first for a search tree branch is due to data reduction recursively marking and deleting vertices.

The instances we use for our experiments on real world data are given in Figure 5.3. The first three instances are from the 10th DIMACS Implementation Challenge and the last three from the Stanford Network Analysis Project. Refer to the weblinks given before on page 60 if interested in details about the context of these graphs. For each instance we have given the number of vertices n and number of edges m, the maximum degree Δ, the average degree d rounded to integers, the h-index, the h_2-index (as discussed in Section 4.2) and the clique number ω. The first three instances are a little bit larger, but also sparser, and we especially see that their h_2-index (providing a rough measure for the hardness of the instances) is rather small for their size compared to the latter three instances. As we already underwent the trouble of determining the clique number for each graph, we use it as a lower bound for our search of the largest $(2,t)$-clubs for each t.

We will only discuss the results for the graph 'soc-Slashdot0902' in more detail here, as it seems to be the most interesting instance and our observations regarding it can be easily compared to the remaining graphs. We present the results for this graph in Figure 5.4, the results for the remaining graphs can be found in Figures A.5 to A.11 in the appendix. All these figures have the same following structure in common: We list results for all values of t for each graph, up to the first t for which the solution is a clique. If we omit some value of t in a figure, as it happens in all figures but Figure 5.4, it means that the solution for this value of t is the same as the solution for the next listed value of t. We provide for each solution the

number n of vertices in it, the number m of edges in it, its edge density, the diamond number d and the number Δ_s of vertices in this solution that were not part of the solution for the previous value of t. The number of vertices in which a solution overlaps with the previous solution is therefore $n - \Delta_s$. We list the time in seconds we needed to solve the problem for a given value of t on the graph, as well as the amount of this time spent on the search tree phase (named 'time$_{ST}$' in the figures). We further list the number of kernels K and number of hard kernels K_{hard}, as well as the size n_{hard} of the largest hard kernel. Finally, we give the amount of search tree branches (i.e. nodes) and operations (ops) required by the search tree phase across all hard kernels. If the diamond number equals t, we mark the number bold, as in this case the solution is less informative, as discussed.

Analyzing the solutions in Figure 5.4 for the graph 'soc-Slashdot0902', we first see that the solutions quickly decrease in size for the first few values of t, after which the solution sizes only decrease slowly until hitting a clique at $t = 37$. We also see that the first three solutions are t-diamonds, but all for $t \geq 4$ are not. However, for $t = 4$ the solution is already much smaller than the largest 2-club. Another interesting aspect is the sudden jump in density at $t = 4$, as well as the fact that up to $t = 4$ the solutions change significantly over the previous solution (as observed by Δ_s), while afterwards the solutions all seem to remain mostly in the same region of the graph. Regarding running times we see that for most values of t the running time is clearly dominated by the polynomial-time data reduction, only for $t = 3$ the search tree phase clearly dominates the running time, while for the few values following it we have at least some substantial search tree activity. We see that even in the case $t = 3$ the fraction of kernels which are hard is relatively small and the size of the largest hardest kernel is a good bit below the h_2-index given in Figure 5.3. A very interesting observation regarding the performance of the algorithm is the relatively low number of branches required. In the case $t = 3$ we only have 53334 branches across 431 hard kernels, which is only moderately above 100 for each kernel, despite the largest hard kernel having size 1058. On the other hand we take nearly one hour to process these 53334 branches, which means we merely process about 17 branches per second. This seems slow but is due to the heavy data reduction going on during the search tree phase. Across 53334 branches we perform over 9 million operations, which means that on average we recursively mark or delete over 170 additional vertices for each branch we make. So we see that even during the search tree phase data reduction seems to make major contributions to solving the problem.

Regarding the remaining tested instances, it is noteworthy that 'soc-Slashdot0902' is the only one with significant running times of the search tree phase. On many real world graphs (2,t)-Club still seems to mostly yield to mere polynomial-time data reduction. However, especially the easier and particularly sparse instances from the DIMACS challenges seem to produce

t	n	m	density	Δ_s	d	time	time$_{ST}$	K	K_{hard}	n_{hard}	branches	ops
1	2553	11069	0.00340	—	**1**	8.1860	0.0000	0	0	0	0	0
2	1449	6181	0.00589	1244	**2**	468.1950	0.0099	39055	28	1449	28	81
3	328	1907	0.03556	290	**3**	3921.6160	3111.2478	36997	431	1058	53334	9376216
4	272	11429	0.31010	251	0	934.3680	340.9723	29998	126	767	16111	1371953
5	252	10638	0.33637	4	0	536.8210	36.1618	25503	53	481	6165	201701
6	238	10081	0.35744	8	0	454.5300	13.0771	22105	41	359	2684	81988
7	227	9636	0.37566	6	0	415.5990	7.6714	19428	34	328	1677	47042
8	218	9328	0.39437	4	0	383.5250	4.4924	17262	24	305	1208	27804
9	210	8928	0.40684	5	0	363.1970	2.7307	15598	23	284	720	16771
10	199	8497	0.43130	3	0	346.6280	3.7971	14122	26	270	1424	21465
11	191	8140	0.44861	3	0	322.3770	2.8339	12813	25	258	912	15649
12	182	7675	0.46597	5	0	303.9440	3.5022	11702	26	244	1089	18399
13	174	7304	0.48528	3	0	291.1810	4.4489	10764	28	242	1444	22440
14	168	7058	0.50314	6	0	273.1800	4.4051	9901	26	233	1337	20353
15	162	6781	0.51998	4	0	252.0720	2.7996	9175	26	224	770	13803
16	156	6436	0.53234	6	0	239.8690	2.4389	8501	21	213	686	11977
17	152	6242	0.54392	0	0	223.1350	1.6678	7917	20	205	447	8407
18	147	6028	0.56174	2	0	208.0750	1.2275	7386	18	194	378	6744
19	142	5746	0.57397	6	0	191.8770	1.0134	6850	18	186	413	5949
20	139	5595	0.58336	1	0	180.6770	0.5849	6344	15	179	284	3843
21	132	5255	0.60780	2	0	167.0680	0.9996	5907	18	175	471	5962
22	128	5049	0.62119	2	0	154.1870	0.7995	5486	16	175	349	4640
23	125	4866	0.62787	2	0	144.9800	0.4839	5091	15	163	247	3277
24	121	4685	0.64532	4	0	133.5180	0.3860	4727	15	153	206	2752
25	119	4569	0.65076	4	0	120.3110	0.3154	4313	16	144	224	2644
26	115	4350	0.66362	1	0	112.6120	0.3783	3981	18	143	242	2964
27	109	4027	0.68417	2	0	102.3960	0.4889	3670	18	143	260	3505
28	104	3743	0.69884	2	0	92.1760	0.5805	3338	18	141	315	3628
29	100	3547	0.71657	2	1	81.6320	0.4864	3047	13	139	227	3010
30	97	3375	0.72487	3	1	73.5930	0.4085	2743	9	133	238	2847
31	92	3114	0.74391	4	5	66.9230	0.5240	2478	10	129	249	3323
32	85	2755	0.77171	4	4	58.1980	1.0087	2221	15	134	416	5728
33	82	2607	0.78500	3	4	50.9250	0.7908	1974	14	130	293	4629
34	80	2508	0.79367	1	4	45.3040	0.6121	1745	12	126	264	4093
35	74	2200	0.81451	3	4	39.3550	1.1160	1524	15	125	400	7143
36	50	1126	0.91918	4	10	41.2080	6.0915	1324	39	136	2175	38116
37	27	351	1.00000									

Figure 5.4: Results for the graph 'soc-Slashdot0902'.

t-diamonds as solutions even for higher values of t. An interesting behavior that can be seen on some of these other graphs is that sometimes the solutions for a range of values of t all are in the same region of the graph, just to jump to an entirely different region and stay there for another range of values. See the results for the graph 'facebook_combined' beginning in Figure A.8. The solutions for t from 3 to 11 are all roughly in the same region, as are the results for all $t \geq 12$, but this region switches completely when going from $t = 11$ to $t = 12$. Another common theme across all instances is that already the largest (2,2)-club seems to be considerably smaller than the largest 2-club most of the time.

In concluding the experiments, we hope that we have shown that (2,t)-Club is both a meaningful and tractable problem for most real world social network data. In particular we have seen that for many sparse real world graphs the problem often runs in effectively polynomial time and that across the spectrum of values of t there tend to be always some and often many solutions which avoid the deficiencies of the basic 2-Club model for real world social networks. Which value of t will provide a particularly good solution for a given graphs is, however, hard to predict. One might therefore in practice favor the same approach we used in these experiments and determine the entire spectrum of solutions for all values of t.

Chapter 6

Outlook and Conclusion

We conclude this work by summarizing the results we have obtained and noting open questions and pointers for further research.

6.1 Conclusion

In this work we have presented a generalization of the well-known 2-Club problem with the aim of providing a better model for cohesive subgroups in social network analysis. Just like the original 2-Club problem the $(2,t)$-Club problem is NP-complete, so we focussed our theoretical discussions on finding easy special cases of the problem by considering special graph classes and problem parameterization. We have shown polynomial-time solvability for a variety of graph classes and introduced a new parameter called h_2-index with relevance to the $(2,t)$-Club problem.

On the practical side we have provided and implemented an algorithm based on earlier work by Hartung et al. [Har+15b] which turns out to be reasonably fast, especially on larger real world graphs where previous algorithms for 2-Club required much more time. While we did not have any existing algorithms for the more general $(2,t)$-Club problem to compare our algorithm against, our results on real world graphs and the performance comparison with ILP formulations show reasonable running times for the general $(2,t)$-Club problem, too. The solutions for $(2,t)$-Club turn out to be usually less trivial than the solutions for 2-Club on real world graphs, motivating the usefulness of $(2,t)$-Club over 2-Club for social network analysis.

In summary, the worst case complexity of $(2,t)$-Club seems to be mostly in line with the complexity of the basic 2-Club problem. A notable exception seems to be on bipartite graphs where the problem gets harder with increasing t. Another possible exception could lie with its complexity parameterized by solution size, for which we could not rule out W[1]-hardness. On many real world graphs 2-Club turns out significantly easier in practice

due to existence of a trivial solution which can be found easily and only has to be verified.

6.2 Open Questions

Our work leaves several questions regarding $(2,t)$-Club open. The most important open question seems to regard the status of $(2,t)$-Club for constant t when parameterized by the solution size k. We were able to show W[1]-containment, but the question whether the problem is truly W[1]-complete for constant t or fixed-parameter tractable with regard to k remains.

We have also considered the $(2,t)$-Club problem on various graph classes, but could not resolve the status of some classes which might be of interest, like AT-free graphs or strongly chordal graphs. These classes would be particularly interesting in comparison with the results of Golovach et al. [Gol+14] to further underline differences between 2-Club and the more general $(2,t)$-Club problem. Regarding co-bipartite graphs, we could only show polynomial-time solvability for $t = 2$. How does $(2,t)$-Club behave on co-bipartite graphs for larger t? Does it show a similar behaviour to the problem on bipartite graphs, where the degree of the polynomial depends on t? Further, can any practical algorithm be designed for $(2,t)$-Club with $t \leq 2$ on planar graphs? Our proof for polynomial-time solvability depends on Courcelle's theorem which does not directly lead to an algorithm fast in practice. Last, are there better strategies for a FPT-algorithm for $(2,t)$-Club parameterized by solution size k on split graphs, i.e. algorithms whose running time is not doubly exponential in k?

We could classify the parameterized complexity of $(2,t)$-Club for a variety of parameters transferring results by Hartung et al. [Har+15a] and proving fixed-parameter tractability for treewidth as well as some other parameters. However, regarding practical applications only the considerations for maximum degree (and the newly introduced parameter h_2-index) yield algorithmic results in the form of Turing kernelization. It would be interesting to find practical FPT-algorithms for some of the parameters which we only show to be FPT indirectly through treewidth. Further we have left a number of interesting parameters as open problems, including many of those with distance to a particular graph class. Last it would be of interest whether our brief thoughts on the parameter N_2-degeneracy have practical applications, i.e. whether constantly sorting the list of Turing kernels at all times in an algorithmic implementation is feasible and useful.

Another pointer for further research will be given in the following subsection where we will discuss the generalization of $(2,t)$-Club to higher diameter than two.

6.3 The (s,t)-Club Problem and Other Variants of the Problem

In this work we more or less thoroughly considered the (2,t)-Club problem as an generalization of the 2-Club problem. However, in the literature one usually considers the more general s-Club problem where one asks for the largest subgraph of diameter s instead of just diameter two. Bringing both (2,t)-Club and s-Club together leads to the more general (s,t)-Club problem which we would state as follows:

(s,t)-Club
Input: A graph $G = (V, E)$ and positive integers s, t and k.
Question: Is there a subset $S \subseteq V$ with $|S| \geq k$, such that in the induced subgraph $G[S]$ every pair of vertices is either adjacent or connected by at least t vertex-disjoint paths of length at most s?

While we have mentioned the more general (s,t)-Club problem in the introduction, we did not explore it any further. This is in part because we feel that varying the parameter s requires different algorithmic approaches and separate analysis, going beyond the scope of this work which focussed on the basic special case of $s = 2$. Golovach et al. [Gol+14] have shown that the s-Club problem varies in complexity between polynomial and NP-hard on some graph classes depending on whether s is odd or even, which supports our assumption that altering s might shake up things significantly. In contrast our findings in this work give the impression that altering t does not significantly alter the inherent complexity of the problem in most cases, with the exception of the gap between $t = 1$ and $t \geq 2$. In fact we would be surprised if the problem would show drastically different behaviour for some $t \geq 2$ when going from t to $t + 1$.

However, the more general (s,t)-Club might very well be of practical interest, as s-Club is often noted for being of interest for at least $s = 3$ on real world instances, too. Going beyond $s = 3$ for s-Club is usually of less interest as most real world social networks exhibit the small world phenomenon as first noted by Milgram [Mil67]. With the additional restrictions imposed by the parameter t, however, the more general (s,t)-Club problem might be of interest for larger values of s, if t is sufficiently large at the same time.

We note again that Veremyev and Boginski [VB12] have already introduced a problem very similar to (s,t)-Club, which they called R-robust s-Club, where R plays the same role as our t. The authors however require that *all* pairs of vertices (including adjacent ones) must have the required number of vertex-disjoint paths, which makes sense from their use-case of

networks with robustness against external attacks, but seems too restrictive for our use-case of identifying cohesive subgroups in social networks. Nonetheless, R-ROBUST s-CLUB is a variant of (s,t)-CLUB for which Veremyev and Boginski [VB12] have argued notable applications of interest. It would also be interesting to see how the two similar problems compare regarding their inherent computational complexity. As a side note, the noticeable gap in complexity between the very compact ILP formulation for R-ROBUST 2-CLUB and the rather intricate ILP formulation for R-ROBUST s-CLUB (which only considers a considerably relaxed version of their initial definition), as both given by Veremyev and Boginski [VB12], further supports our feeling that (s,t)-CLUB is a considerably more intricate problem than $(2,t)$-CLUB.

As a further variant of the problem one could consider $(2,t)$-CLIQUE or (s,t)-CLIQUE, where we do not require the connections between two non-adjacent vertices of the solution to be part of the solution, i.e. we only demand from each pair of non-adjacent vertices in the solution to have the required number of common neighbors or paths between them *in the original input graph* and not necessarily the solution graph. These variants therefore have the same relation to the original problem as s-CLIQUE has to s-CLUB, hence the naming. The use of this problem variant for finding cohesive subgroups in social networks might be questionable as the resulting solutions might not be connected. One might try to amend this by requiring at least one neighbor or path to be part of the solution, therefore ensuring connectedness.

Last but not least, we have only considered the problem of finding the largest subgraph with the desired properties so far. But in practice the largest subgraph might not always be the most interesting, which is especially apparent in the case of 2-CLUB where nearly all solutions on real world graphs are trivially the 1-neighborhood of the maximum degree vertex. Instead we might want to consider the problem of enumerating all subgraphs (or all subgraphs having a certain minimum size) with the desired properties. Of course, for dense graphs the number of solutions might explode quickly, making the problem infeasible for all but small graphs. But if the graph is sufficiently sparse (with occasional denser clusters) and the minimum size requirement sufficiently high, the number of solutions to find might be reasonable even for larger graphs. Nonetheless, the scenario that there are many solutions organized around the same cluster and differing only in a few vertices seems likely. Hence, if exploring the problem of enumerating (s,t)-clubs one might want to analyze the relation and position of the individual (s,t)-clubs of a graph to each other.

Literature

[Alb73] R. D. Alba. "A graph-theoretic definition of a sociometric clique". In: *Journal of Mathematical Sociology* 3.1 (1973), pp. 113–126 (cit. on p. 2).

[Arn+91] S. Arnborg, J. Lagergren, and D. Seese. "Easy problems for tree-decomposable graphs". In: *Journal of Algorithms* 12.2 (1991), pp. 308–340 (cit. on pp. 48, 49).

[Asa+10] Y. Asahiro, E. Miyano, and K. Samizo. "Approximating maximum diameter-bounded subgraphs". In: *LATIN 2010: Theoretical Informatics*. Springer, 2010, pp. 615–626 (cit. on p. 5).

[BA99] A.-L. Barabási and R. Albert. "Emergence of scaling in random networks". In: *Science* 286 (1999), pp. 509–512 (cit. on pp. 58, 59).

[Bak94] B. S. Baker. "Approximation algorithms for NP-complete problems on planar graphs". In: *Journal of the ACM (JACM)* 41.1 (1994), pp. 153–180 (cit. on p. 35).

[Bal+05] B. Balasundaram, S. Butenko, and S. Trukhanov. "Novel approaches for analyzing biological networks". In: *Journal of Combinatorial Optimization* 10.1 (2005), pp. 23–39 (cit. on pp. 6, 57).

[BL76] K. S. Booth and G. S. Lueker. "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms". In: *Journal of Computer and System Sciences* 13.3 (1976), pp. 335–379 (cit. on pp. 23, 25).

[BM76] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*. North-Holland, 1976 (cit. on p. 36).

[Bou+02] J.-M. Bourjolly, G. Laporte, and G. Pesant. "An exact algorithm for the maximum k-club problem in an undirected graph". In: *European Journal of Operational Research* 138.1 (2002), pp. 21–28 (cit. on pp. 5, 6, 15, 50, 51, 59, 64).

[BR+12] D. Binkele-Raible, H. Fernau, F. V. Fomin, D. Lokshtanov, S. Saurabh, and Y. Villanger. "Kernel(s) for problems with no kernel: On out-trees with many leaves". In: *ACM Transactions on Algorithms (TALG)* 8.4 (2012), article number 38 (cit. on p. 50).

[Bra+99] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999 (cit. on pp. 8, 22).

[CA11] F. D. Carvalho and M. T. Almeida. "Upper bounds and heuristics for the 2-club problem". In: *European Journal of Operational Research* 210.3 (2011), pp. 489–494 (cit. on pp. 6, 57).

[Cha+13] M.-S. Chang, L.-J. Hung, C.-R. Lin, and P.-C. Su. "Finding large k-clubs in undirected graphs". In: *Computing* 95.9 (2013), pp. 739–758 (cit. on pp. 6, 21, 47, 51, 58, 63, 64).

[Cha12] T. M. Chan. "All-pairs shortest paths for unweighted undirected graphs in o(mn) time". In: *ACM Transactions on Algorithms (TALG)* 8.4 (2012), article number 34 (cit. on p. 35).

[Che+05] Y. Chen, J. Flum, and M. Grohe. "Machine-based methods in parameterized complexity theory". In: *Theoretical Computer Science* 339 (2005), pp. 167–199 (cit. on p. 17).

[Cou+00] B. Courcelle, J. A. Makowsky, and U. Rotics. "Linear time solvable optimization problems on graphs of bounded clique-width". In: *Theory of Computing Systems* 33.2 (2000), pp. 125–150 (cit. on pp. 48, 49).

[Cou90] B. Courcelle. "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs". In: *Information and computation* 85.1 (1990), pp. 12–75 (cit. on p. 48).

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013 (cit. on pp. 6, 8, 50).

[Die12] R. Diestel. *Graph Theory, 4th Edition*. Vol. 173. Graduate texts in mathematics. Springer, 2012 (cit. on pp. 6, 32).

[Dor10] F. Dorn. "Planar subgraph isomorphism revisited". In: *Proc. of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*. 2010, pp. 263–274 (cit. on p. 35).

[Epp00] D. Eppstein. "Diameter and treewidth in minor-closed graph families". In: *Algorithmica* 27.3-4 (2000), pp. 275–291 (cit. on p. 35).

[Epp99] D. Eppstein. “Subgraph isomorphism in planar graphs and related problems”. In: *Journal of Graph Algorithms and Applications* 3.3 (1999), pp. 1–27 (cit. on p. 35).

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006 (cit. on pp. 6, 8).

[Gen+93] M. Gendreau, P. Soriano, and L. Salvail. “Solving the maximum clique problem using a tabu search approach”. In: *Annals of Operations Research* 41.4 (1993), pp. 385–403 (cit. on p. 58).

[GJ79] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to NP-completeness*. W.H. Freeman, New York, 1979 (cit. on pp. 36, 39).

[Gol+14] P. A. Golovach, P. Heggernes, D. Kratsch, and A. Rafiey. “Finding clubs in graph classes”. In: *Discrete Applied Mathematics* 174 (2014), pp. 57–65 (cit. on pp. 5, 22, 75, 76).

[Guo+07] J. Guo, R. Niedermeier, and S. Wernicke. “Parameterized complexity of vertex cover variants”. In: *Theory of Computing Systems* 41 (2007), pp. 501–520 (cit. on p. 17).

[Har+15a] S. Hartung, C. Komusiewicz, A. Nichterlein, and O. Suchý. “On structural parameterizations for the 2-club problem”. In: *Discrete Applied Mathematics* 185 (2015), pp. 79–92 (cit. on pp. 5, 11, 16, 43, 45, 68, 75).

[Har+15b] S. Hartung, C. Komusiewicz, and A. Nichterlein. “Parameterized algorithmics and computational experiments for finding 2-Clubs”. In: vol. 19. 1. 2015, pp. 155–190 (cit. on pp. 1, 5, 6, 19, 50, 55, 58, 60, 61, 63, 64, 66, 74).

[HK73] J. E. Hopcroft and R. M. Karp. “An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs”. In: *SIAM Journal on Computing* 2.4 (1973), pp. 225–231 (cit. on p. 38).

[Hoc98] D. S. Hochbaum. “Approximating clique and biclique problems”. In: *Journal of Algorithms* 29.1 (1998), pp. 174–200 (cit. on p. 36).

[HP05] M. Habib and C. Paul. “A simple linear time algorithm for cograph recognition”. In: *Discrete Applied Mathematics* 145.2 (2005), pp. 183–197 (cit. on p. 26).

[Mil67] S. Milgram. “The small world problem”. In: *Psychology Today* 2.1 (1967), pp. 60–67 (cit. on pp. 3, 76).

[Mok79] R. J. Mokken. “Cliques, clubs and clans”. In: *Quality & Quantity* 13.2 (1979), pp. 161–173 (cit. on p. 2).

[MP14] C. McCreesh and P. Prosser. “Finding maximum k-cliques faster using lazy global domination”. In: *arXiv preprint arXiv:1408.6485* (2014) (cit. on p. 61).

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Vol. 31. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006 (cit. on pp. 6, 8).

[Pat+13] J. Pattillo, N. Youssef, and S. Butenko. “On clique relaxation models in network analysis”. In: *European Journal of Operational Research* 226.1 (2013), pp. 9–18 (cit. on pp. 2, 3).

[SB13] S. Shahinpour and S. Butenko. “Algorithms for the maximum k-club problem in graphs”. In: *Journal of Combinatorial Optimization* 26.3 (2013), pp. 520–554 (cit. on pp. 6, 58, 61–63).

[Sch+12] A. Schäfer, C. Komusiewicz, H. Moser, and R. Niedermeier. “Parameterized computational complexity of finding small-diameter subgraphs”. In: *Optimization Letters* 6.5 (2012), pp. 883–891 (cit. on p. 5).

[Sch09] A. Schäfer. “Exact algorithms for s-club finding and related problems”. Diplomarbeit. Friedrich-Schiller-Universität Jena, 2009 (cit. on pp. 5, 22, 35, 36).

[VB12] A. Veremyev and V. Boginski. “Identifying large robust network clusters via new compact formulations of maximum k-club problems”. In: *European Journal of Operational Research* 218.2 (2012), pp. 316–326 (cit. on pp. 4, 5, 57, 76, 77).

[Wor99] N. C. Wormald. “Models of random regular graphs”. In: *Surveys in Combinatorics*. London Mathematical Society Lecture Note Series 267 (1999), pp. 239–298 (cit. on p. 60).

[Wot14] A. Wotzlaw. “On Solving the Maximum k-club Problem”. In: *arXiv preprint arXiv:1403.5111* (2014) (cit. on pp. 6, 61).

Appendix

This appendix contains various additional tables of results from our experiments discussed in Chapter 5. The following is a list of the figures contained in this appendix for easy reference.

Figure A.1 :
Comparison of our RTC algorithm and the Hartung algorithm for various larger real world instances from Chris Walshaw's Graph Partitioning Archive.

Figure A.2 :
Comparison of our RTC algorithm and the Hartung algorithm on random graphs generated by the Gendreau model.

Figure A.3 :
Comparison of our RTC algorithm and the Hartung algorithm on random graphs generated by the random regular model.

Figure A.4 :
Comparison of our RTC algorithm and the Hartung algorithm on random graphs generated by the Barabási-Albert model.

Figure A.5 :
Results for the graph 'coAuthorsCiteseer'.

Figure A.6 :
Results for the graph 'coAuthorsDBLP'.

Figure A.7 :
Results for the graph 'citationCiteseer'.

Figure A.8 - Figure A.10 :
Results for the graph 'facebook_combined'.

Figure A.11 :
Results for the graph 'cit-HepTH'.

Instances					**RTC**			**Hartung**	
name	n	m	Δ	2-club size	time	kernels	hard kernels	time	kernels
3elt	4720	13722	9	10	0.07	237	0	11.31	90
4elt	15606	45878	10	11	0.16	26	0	40.56	266
bcsstk33	8738	291583	140	141	22.52	3967	0	—	—
cti	16840	48232	6	7	1.52	13573	996	—	—
fe_4elt2	11143	32818	12	13	0.08	0	0	2.03	0
fe_body	45087	163734	28	29	0.65	0	0	9.47	0
fe_pwt	36519	144794	15	16	0.59	389	0	658.08	5016
fe_sphere	16386	49152	6	7	0.93	9444	1328	—	—
finan512	74752	261120	54	56	2.90	6849	1	2443.27	5567
memplus	17758	54196	573	574	1.06	0	0	22.74	0
t60k	60005	89440	3	5	0.35	17331	1	1054.36	12289
vibrobox	12328	165250	120	121	15.25	2509	1	—	—
wing	62032	121544	4	6	1.26	31583	6	1618.87	18577
wing_nodal	10937	75488	28	29	2.13	5599	0	—	—

Figure A.1: Running time comparisons for graphs from Chris Walshaw's Graph Partitioning Archive.

Instances					**RTC**				**Hartung**			
density	n	avg. m	avg. Δ	avg. 2-club size	T_{avg}	T_{min}	T_{max}	branches	T_{avg}	T_{min}	T_{max}	recursions
$[0.00, 0.30]$	100	741.29	29.00	41.71	0.22	0.03	0.73	1982	0.12	0.02	0.79	2175
	120	1067.44	34.36	55.17	0.74	0.03	3.21	5671	0.36	0.03	1.57	6222
	130	1250.90	36.76	62.40	1.35	0.03	12.76	9732	0.64	0.03	4.62	10315
	140	1454.27	39.39	70.68	2.25	0.05	30.07	15268	1.03	0.04	12.66	16214
	150	1680.97	41.47	80.95	2.46	0.04	22.52	15620	1.16	0.03	8.91	15583
$[0.05, 0.25]$	100	746.18	26.66	35.50	0.61	0.11	2.10	5032	0.28	0.06	1.12	6004
	120	1080.29	31.82	47.93	2.92	0.28	10.87	20997	1.28	0.15	4.73	22727
	130	1258.56	34.13	53.27	6.40	0.47	31.31	40947	2.90	0.18	14.12	46646
	140	1454.59	36.47	60.49	14.44	0.50	84.19	81110	6.26	0.21	32.65	85140
$[0.10, 0.20]$	100	741.89	24.71	31.48	1.21	0.39	3.04	8029	0.58	0.19	1.39	10260
	120	1075.20	29.64	40.11	10.52	2.66	33.87	58984	4.47	1.15	14.50	68280
	130	1262.24	31.58	45.68	23.66	5.69	81.11	136241	10.46	2.29	37.35	157180
$[0.15, 0.15]$	100	741.90	24.43	30.09	1.23	0.29	3.06	9030	0.58	0.12	1.44	11312
	120	1076.97	28.35	38.10	11.41	2.69	27.40	69642	4.85	1.31	12.54	82715
	130	1259.74	30.61	42.36	33.53	5.98	112.32	179421	15.04	3.29	47.80	218988

Figure A.2: Running time comparisons for randomly generated graphs using the Gendreau model.

Instances					**RTC**				**Hartung**			
vertices	degree	edges	avg. Δ	avg. 2-club size	T_{avg}	T_{min}	T_{max}	branches	T_{avg}	T_{min}	T_{max}	recursions
70	13	455	13.16	22.65	0.52	0.18	2.18	5351	0.26	0.09	0.85	7315
	14	490	14.23	27.85	0.78	0.08	2.79	8917	0.39	0.06	1.06	11189
	15	525	15.33	35.19	0.73	0.12	1.96	10035	0.39	0.11	1.08	11741
	16	560	16.21	46.94	0.28	0.03	2.36	4470	0.16	0.00	1.49	4903
80	14	560	14.26	23.02	1.32	0.63	3.54	11902	0.66	0.26	1.93	16270
	15	600	15.21	27.82	2.28	0.88	4.19	21160	1.09	0.37	2.00	26692
	16	640	16.25	34.44	3.23	1.18	6.48	31674	1.61	0.43	3.70	39069
	17	680	17.31	44.66	2.37	0.30	5.53	28373	1.28	0.21	3.05	33756
90	16	720	16.17	28.34	6.32	3.09	9.47	48100	3.05	1.25	4.44	65369
	17	765	17.18	34.15	11.71	4.92	24.68	91234	5.67	2.35	13.44	115518
	18	810	18.28	43.38	15.29	5.17	32.86	133479	7.11	2.03	13.49	151179
	19	855	19.30	59.93	3.53	0.10	15.99	43730	2.25	0.06	12.28	54485
100	17	850	17.34	28.61	16.36	8.39	26.18	105849	7.72	3.20	13.35	147519
	18	900	18.21	34.15	36.20	13.97	61.14	242391	16.86	6.45	30.31	319217
	19	950	19.27	42.34	62.76	24.99	151.16	427750	27.53	5.00	59.37	505930
	20	1000	20.29	55.94	39.70	5.40	112.25	337178	18.88	3.62	50.75	378765

Figure A.3: Running time comparisons for randomly generated graphs using the random regular model.

Instances						**RTC**			**Hartung**		
vertices	seed size	degree	edges	avg. Δ	avg. 2-club size	T_{avg}	T_{min}	T_{max}	T_{avg}	T_{min}	T_{max}
200	20	10	1820	65.80	67.00	1.67	0.16	6.20	0.71	0.09	2.49
		15	2720	87.26	110.79	6.16	0.54	25.35	2.67	0.32	8.73
		20	3620	103.89	161.68	0.27	0.04	5.81	0.13	0.02	1.84
400	40	10	3640	84.52	85.52	2.80	1.68	9.67	2.16	1.25	7.26
600	60	10	5460	92.03	93.03	8.16	6.04	11.19	8.18	6.97	11.46
800	80	10	7280	100.40	101.40	21.36	13.46	27.46	24.86	19.75	32.00
1000	100	10	9100	105.70	106.70	46.30	35.52	51.27	58.67	53.91	64.51

Figure A.4: Running time comparisons for graphs using the Barabási-Albert model with a cycle as seed graph.

t	n	m	density	Δ_s	d	time	time$_{ST}$	K	K_{hard}	n_{hard}	branches	ops
1	1373	3695	0.00392	—	**1**	10.2280	0.0000	0	0	0	0	0
2	483	2394	0.02057	482	**2**	8.1980	0.0053	2208	9	483	16	74
3	387	2000	0.02678	387	**3**	5.9820	0.0014	1896	5	387	5	9
4	386	1996	0.02686	0	**4**	3.6810	0.0007	1229	3	386	3	3
5	159	1186	0.09442	159	**6**	2.3770	0.0003	1158	2	159	2	2
7	139	1048	0.10927	0	**7**	1.9900	0.0003	979	2	139	2	2
8	113	4012	0.63401	113	0	2.5790	0.0032	2105	3	120	10	51
9	106	3847	0.69128	6	8	2.4520	0.0034	2313	2	120	16	36
12	103	4071	0.77499	103	**13**	2.0830	0.0017	1922	2	110	11	24
14	101	3747	0.74198	101	8	1.8140	0.0011	1652	1	104	4	16
15	100	3725	0.75253	0	11	1.6830	0.0007	1534	1	102	3	14
19	98	3628	0.76331	0	11	1.3220	0.0007	1108	1	100	3	8
20	96	3571	0.78311	2	11	1.2320	0.0008	1052	1	100	5	12
21	94	3496	0.79982	0	14	1.1150	0.0010	988	1	100	4	14
23	92	3399	0.81199	0	14	1.2480	0.0008	846	1	92	1	11
24	91	3361	0.82076	0	16	1.0670	0.0008	778	1	91	1	5
26	87	3741	1.00000									

Figure A.5: Results for the graph 'coAuthorsCiteseer'.

t	n	m	density	Δ_s	d	time	time$_{ST}$	K	K_{hard}	n_{hard}	branches	ops
1	337	6915	0.12214	—	**1**	15.6640	0.0000	0	0	0	0	0
2	216	6252	0.26925	0	**2**	19.0620	0.0103	4137	9	216	15	122
3	160	7940	0.62421	158	1	12.2300	0.0043	6196	5	161	7	33
4	155	7895	0.66150	0	**4**	9.4360	0.0023	5215	3	155	3	5
5	152	7874	0.68613	0	**9**	7.8970	0.0014	7027	2	152	2	3
10	146	7747	0.73188	0	**11**	2.1720	0.0021	777	3	146	3	5
12	145	7720	0.73946	0	**23**	0.6730	0.0026	188	3	145	3	4
24	128	6989	0.85987	0	**24**	0.3470	0.0018	36	2	128	2	13
25	120	6737	0.94356	0	**25**	0.2830	0.0022	44	2	120	2	18
26	118	6682	0.96798	0	**27**	0.1840	0.0013	33	2	118	2	9
28	117	6654	0.98055	0	**34**	0.1900	0.0012	28	1	117	1	7
35	116	6620	0.99250	0	**58**	0.1610	0.0007	18	1	116	1	1
66	115	6555	1.00000									

Figure A.6: Results for the graph 'coAuthorsDBLP'.

t	n	m	density	Δ_s	d	time	time$_{ST}$	K	K_{hard}	n_{hard}	branches	ops
1	1319	3758	0.00432	—	**1**	17.7850	0.0000	0	0	0	0	0
2	268	1409	0.03938	268	**2**	63.2400	0.0059	83591	31	268	33	389
3	120	493	0.06905	120	**3**	78.5130	0.0016	139791	11	120	15	1027
4	57	548	0.34336	57	0	66.4280	0.0124	112294	11	97	77	3393
5	45	450	0.45455	4	0	51.6880	0.0133	82175	10	65	85	1822
6	39	378	0.51012	0	0	38.1640	0.0009	57252	9	45	23	642
7	37	264	0.39640	37	**7**	27.5390	0.0003	37924	4	37	8	275
8	33	283	0.53598	33	**8**	18.5630	0.0003	23701	5	34	9	204
9	31	255	0.54839	0	**9**	11.2540	0.0001	13671	2	32	3	108
10	27	227	0.64672	27	**10**	6.5480	0.0001	7068	3	27	3	81
11	24	197	0.71377	24	**12**	3.6080	0.0000	3214	1	24	1	72
13	13	78	1.00000									

Figure A.7: Results for the graph 'citationCiteseer'.

t	n	m	density	Δ_s	d	time	time$_{ST}$	K	K_{hard}	n_{hard}	branches	ops
1	1046	27795	0.05086	—	**1**	0.5960	0.0000	0	0	0	0	0
2	300	15729	0.35070	297	1	30.6050	9.0692	2638	31	431	1351	27351
3	274	14481	0.38718	274	1	18.9950	0.7682	2591	18	377	506	8993
4	259	13740	0.41124	0	1	16.3760	0.6185	2468	15	332	502	5761
5	251	13395	0.42693	5	1	14.2670	0.5427	2301	14	301	327	4280
6	244	13099	0.44185	3	1	13.3020	0.8140	2155	10	289	345	4707
7	240	12819	0.44697	7	1	12.1660	0.6482	1987	12	281	307	3690
8	234	12567	0.46099	5	1	11.9280	0.7844	1850	11	276	309	3936
9	226	12125	0.47689	7	1	11.8110	1.1148	1710	9	274	411	4977
10	222	11874	0.48404	3	1	11.0270	0.9070	1569	11	268	274	4483
11	220	11793	0.48954	5	1	10.3590	0.5442	1450	6	262	167	3117
12	213	16322	0.72292	213	1	10.0650	0.5770	1322	6	262	130	3006
13	213	16310	0.72238	1	1	8.4180	0.0476	1215	2	248	35	1163
14	212	16273	0.72758	0	1	7.7170	0.0051	1108	1	229	18	865
15	211	16226	0.73239	0	1	7.3720	0.0054	1014	1	229	19	741
20	210	16159	0.73634	0	1	5.5140	0.0049	724	1	222	13	70
21	209	16100	0.74071	0	1	4.9290	0.0051	674	1	220	12	67
22	208	16031	0.74466	0	1	5.0140	0.0072	630	1	220	13	72
24	208	16025	0.74438	1	1	4.3430	0.0061	570	1	218	11	56
25	207	15970	0.74903	0	1	4.0870	0.0057	542	1	218	12	62
26	206	15886	0.75236	1	1	3.6790	0.0109	484	1	216	15	71
27	205	15822	0.75667	1	1	3.3880	0.0129	434	1	216	16	77
28	204	15754	0.76084	0	1	3.4940	0.0063	408	1	216	13	63
29	203	15686	0.76506	2	1	3.3410	0.0062	387	1	214	12	67
33	202	15591	0.76799	0	1	2.6590	0.0066	288	1	214	13	44
34	202	15614	0.76912	1	1	2.2880	0.0062	262	1	213	12	42
35	201	15511	0.77169	0	1	2.3850	0.0069	246	1	213	13	53
36	201	15532	0.77274	1	1	2.2250	0.0080	228	1	213	13	42

Figure A.8: Results for the graph 'facebook_combined' (part one).

t	n	m	density	Δ_s	d	time	time$_{ST}$	K	K_{hard}	n_{hard}	branches	ops
37	200	15457	0.77673	0	1	2.1390	0.0071	222	1	213	14	45
40	199	15374	0.78037	0	1	1.9780	0.0081	204	1	212	14	41
43	197	15204	0.78753	0	2	2.0130	0.0112	189	1	212	16	44
44	196	15113	0.79084	0	2	1.8630	0.0095	188	1	212	17	57
45	196	15106	0.79048	3	1	1.8060	0.0091	184	1	212	17	54
46	195	15025	0.79434	1	2	1.7980	0.0129	178	1	211	19	60
47	193	14834	0.80063	1	2	1.7630	0.0116	174	1	211	20	79
48	192	14745	0.80416	1	2	1.7260	0.0150	171	1	211	20	88
49	191	14654	0.80761	1	2	1.8530	0.0174	164	1	211	21	88
50	190	14551	0.81041	2	2	1.7000	0.0238	159	1	211	26	108
51	189	14471	0.81453	3	2	1.7040	0.0194	157	1	211	22	109
53	189	14467	0.81431	1	2	1.6030	0.0157	146	1	210	24	75
54	188	14369	0.81744	0	2	1.6030	0.0147	140	1	210	23	75
55	186	14168	0.82348	0	2	1.6130	0.0139	134	1	208	23	115
56	185	14067	0.82650	1	2	1.6400	0.0186	129	1	208	26	127
57	185	14091	0.82791	3	2	1.7370	0.0182	127	1	208	26	107
58	183	13885	0.83378	1	3	1.7900	0.0232	121	1	207	25	113
59	183	13882	0.83360	2	3	1.8610	0.0195	115	1	206	24	108
60	183	13885	0.83378	2	3	1.6980	0.0238	113	1	206	26	104
63	182	13776	0.83638	0	3	1.6200	0.0203	100	1	206	25	85
64	180	13575	0.84264	0	3	1.6360	0.0235	96	1	206	27	131
66	180	13572	0.84246	1	3	1.5520	0.0209	86	1	205	26	105
67	178	13363	0.84828	0	3	1.4670	0.0233	82	1	205	28	129
68	177	13258	0.85118	0	3	1.5360	0.0235	80	1	205	29	135
69	177	13256	0.85105	1	3	1.2440	0.0170	79	1	203	27	125
70	176	13143	0.85344	1	3	1.1870	0.0183	73	1	203	28	135

Figure A.9: Results for the graph 'facebook_combined' (part two).

t	n	m	density	Δ_s	d	time	time$_{ST}$	K	K_{hard}	n_{hard}	branches	ops
71	174	12916	0.85815	0	3	0.8540	0.0230	27	1	203	32	151
72	173	12802	0.86047	0	4	0.8170	0.0240	27	1	202	30	158
73	173	12811	0.86107	1	3	0.8050	0.0188	27	1	201	29	137
74	170	12471	0.86815	0	3	0.8810	0.0359	29	1	200	32	216
75	170	12482	0.86892	1	4	0.8450	0.0247	29	1	200	31	184
76	170	12474	0.86836	2	4	0.8460	0.0234	29	1	200	31	152
77	169	12365	0.87102	0	4	0.8030	0.0250	29	1	200	32	180
78	168	12266	0.87439	3	4	0.8120	0.0259	28	1	200	33	190
79	166	12038	0.87901	1	4	0.9110	0.0307	27	1	200	36	223
80	166	12039	0.87908	2	4	0.8740	0.0283	26	1	200	35	209
81	165	11926	0.88145	0	5	0.9830	0.0394	27	1	200	36	218
82	164	11812	0.88373	0	5	1.2620	0.0295	26	1	200	37	219
83	161	11479	0.89123	3	6	0.9170	0.0928	27	2	200	61	332
84	160	11362	0.89324	0	6	0.8750	0.0701	27	2	197	46	295
85	158	11136	0.89785	2	7	0.9660	0.1365	28	3	197	65	415
86	156	10903	0.90182	3	8	1.0260	0.1950	29	3	197	61	527
87	154	10676	0.90620	1	9	1.1010	0.2268	28	4	196	69	577
88	153	10566	0.90867	2	7	1.0700	0.2293	28	3	195	61	591
89	150	10225	0.91499	1	7	1.2640	0.4414	30	4	191	92	908
90	149	10112	0.91711	1	7	1.2020	0.3889	29	4	191	83	812
91	149	10117	0.91756	1	8	1.0980	0.2782	28	4	191	66	645
92	148	10002	0.91947	2	9	1.0220	0.2304	28	4	189	64	575
93	144	9536	0.92618	2	9	1.3290	0.5114	29	4	187	96	1003
94	143	9429	0.92869	1	9	1.2330	0.4315	28	4	187	81	881
95	140	9084	0.93361	2	9	1.4320	0.6267	28	4	186	101	1198
96	139	8988	0.93713	4	10	1.2780	0.5016	27	4	185	90	1019
97	139	8988	0.93713	1	11	1.0190	0.2771	26	3	184	65	674
98	131	8084	0.94938	0	15	2.1110	1.3637	25	6	183	187	2323
99	128	7750	0.95349	0	16	2.9980	2.2769	24	6	182	282	3893
100	69	2346	1.00000									

Figure A.10: Results for the graph 'facebook_combined' (part three).

t	n	m	density	Δ_s	d	time	time$_{ST}$	K	K_{hard}	n_{hard}	branches	ops
1	2469	35995	0.01181	—	**1**	3.1930	0.0000	0	0	0	0	0
2	1671	24639	0.01766	1	**2**	161.4300	0.5409	16306	38	1671	685	9059
3	1513	21797	0.01906	0	**3**	157.9630	2.7496	18435	47	1513	2078	47285
4	713	9263	0.03649	0	**4**	206.1540	6.2105	19979	57	715	3021	110407
5	202	2262	0.11142	0	**5**	205.7100	9.9777	19128	80	309	3149	160137
6	117	1267	0.18671	48	5	195.8240	17.0084	17595	158	279	5911	241086
7	97	2196	0.47165	92	1	167.7180	8.3952	16030	109	225	3438	126852
8	90	1993	0.49763	90	0	150.5420	3.5204	14519	69	197	1532	54524
9	83	1794	0.52718	6	0	136.0890	2.0384	13185	54	167	1282	33019
10	75	1556	0.56072	7	0	130.0390	1.7438	12011	61	151	1210	27129
11	72	1730	0.67684	72	1	120.2380	0.5422	11040	34	126	532	11686
12	68	1411	0.61940	68	0	112.0680	0.3297	10160	25	123	299	7618
13	66	1574	0.73380	66	0	104.3430	0.1352	9402	18	113	145	4148
14	64	1521	0.75446	0	0	96.2710	0.0623	8725	18	104	98	2558
15	63	1497	0.76651	63	1	89.9750	0.0135	8075	11	77	67	1481
16	62	1471	0.77790	1	1	82.2570	0.0089	7438	13	73	61	1043
18	60	1406	0.79435	60	1	66.8960	0.0056	6285	8	68	37	530
20	59	1376	0.80421	0	1	53.9380	0.0249	5147	8	65	81	709
22	58	1341	0.81125	0	1	40.7400	0.0315	4127	8	64	79	780
23	57	1308	0.81955	0	1	35.5040	0.0121	3614	5	64	36	370
24	55	1235	0.83165	1	1	30.1020	0.0058	3130	6	64	21	195
25	55	1233	0.83030	1	1	24.8040	0.0038	2659	5	60	19	107
26	52	1128	0.85068	1	1	20.6690	0.0018	2216	3	58	14	77
27	52	1146	0.86425	52	3	15.9890	0.0017	1812	3	56	11	66
28	51	1108	0.86902	0	3	12.1920	0.0021	1422	3	56	16	71
29	50	1072	0.87510	1	3	8.1000	0.0202	1031	4	58	43	361
30	45	894	0.90303	1	5	5.7370	0.0828	729	12	61	127	1249
31	23	253	1.00000									

Figure A.11: Results for the graph 'cit-HepTh'.