



Exact Algorithms for s -Club Finding and Related Problems

DIPLOMARBEIT
zur Erlangung des akademischen Grades
Diplom-Informatiker

Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik

eingereicht von Alexander Schäfer
geb. am 19.07.1983

Betreuer: Dipl.-Bioinf. Christian Komusiewicz,
Dipl.-Inf. Hannes Moser,
Prof. Dr. Rolf Niedermeier

Jena, 8. Dezember 2009

Zusammenfassung

Das CLIQUE Problem ist eines der am besten untersuchten Probleme in der Informatik. Wohingegen nur wenige Untersuchungen existieren, die sich mit der wichtigen Verallgemeinerung von CLIQUE, dem s -CLUB Problem, beschäftigen. Im besonderen gibt es keine intensiven Untersuchungen bezüglich der parametrisierten Komplexität dieses Problems.

In dieser Arbeit wird gezeigt, dass s -CLUB festparameter-handhabbar bezüglich der Anzahl der Knoten in der Lösung ist. Im Bezug auf Polynomialzeit-Datenreduktionen, wird gezeigt, dass es keinen many-to-one Kern polynomieller Größe für s -CLUB geben kann. Im Gegensatz dazu wird ein Turing Kern angegeben, dessen Größe kubisch in der Knotenanzahl der gesuchten Lösung ist. In diesem Kontext wird auch eine interessante Verbindung zur Approximation einer Lösung für s -CLUB gezeigt sowie ein kombinierter Algorithmus angegeben, der diese Verbindung ausnutzt. Ferner wird die Komplexität von s -CLUB auf beschränkten Graphklassen untersucht.

Um effiziente Festparameter-Algorithmen zu erhalten, ist es oft nützlich, die Parametrisierung zu ändern. Daher wird auch s -CLUB mit einer dualen Parametrisierung untersucht, welches als das s -CLUB VERTEX DELETION Problem definiert wird. Es wird gezeigt, dass dieses Problem festparameter-handhabbar bezüglich der Anzahl der Knoten in der Lösung ist.

Ebenfalls wird das s -CLUB CLUSTER VERTEX DELETION Problem eingeführt, welches eine Generalisierung des CLUSTER VERTEX DELETION Problems ist. Für dieses Problem wird NP-Vollständigkeit und Festparameter-Handhabbarkeit bezüglich des kombinierten Parameters Anzahl der Knoten in der Lösung und s gezeigt.

Abstract

The `CLIQUE` problem is one of the best-studied problems in computer science. However, there exist only few studies concerning the important `CLIQUE` generalization, called the `s-CLUB` problem. In particular there have been no intensive investigations with respect to the parameterized complexity of this problem.

In this thesis we show that `s-CLUB` is fixed-parameter tractable with respect to the number of vertices in the solution. In terms of polynomial time data reduction, we show that `s-CLUB` does not admit a polynomial many-to-one kernel. In contrast to that we give a cubic-vertex Turing kernel. In this context we also show an interesting connection to the approximation of a solution for `s-CLUB`, and give a combined algorithm to exploit this connection. Furthermore we study the complexity of `s-CLUB` on some restricted graph classes.

In order to obtain efficient fixed-parameter algorithm, it is often useful to change the parameterization. Therefore, we analyze `s-CLUB` with a dual parameterization, which we define as the `s-CLUB VERTEX DELETION` problem. We show that this problem is fixed-parameter tractable with respect to the number of vertices in the solution.

We also introduce the `s-CLUB CLUSTER VERTEX DELETION` problem, which is a generalization of the `CLUSTER VERTEX DELETION` problem. For this problem we show NP-completeness and fixed-parameter tractability with respect to the combined parameter number of vertices in the solution and s .

Contents

1	Introduction	1
1.1	Motivation and Application	1
1.2	Known Results	3
1.3	Overview	3
2	Basic Notations and Definitions	5
2.1	Basic Notation in Graph Theory	5
2.2	Parameterized Complexity	6
2.3	Search Trees of Bounded Size	7
2.4	Data Reduction	8
2.4.1	Many-to-One Kernelization	8
2.4.2	Turing Kernelization	9
3	s-Club and Related Problems	11
3.1	Basic Observations Concerning s -Clubs	12
3.1.1	s -Club Definitions	12
3.1.2	The Non-Hereditariness of s -Club	12
3.1.3	Monadic Second Order Logic Formulation of s -Club	12
3.1.4	Integer Linear Program for s -Club	13
3.1.5	Dual Parameterization for s -Club	14
3.2	s -Club and Other Clique Relaxations	14
3.3	s -Club and Other Bounded Diameter Problems	17
4	s-Club on General Graphs	19
4.1	A Problem Kernel for the s -Club Problem	19
4.1.1	A Lower Bound for Many-to-One Kernels for s -Club	19
4.1.2	A Turing Kernel for s -Club	21
4.2	A Branching Algorithm for s -Club	26
5	s-Club on Special Graph Classes	29
5.1	Trees	29
5.2	Interval Graphs	34
5.3	2-Club on Bipartite Graphs	37

5.4	Graph Classes with bounded Clique- or Treewidth	38
5.5	Planar Graphs	38
6	Vertex Deletion Problems	41
6.1	Obtaining an s -Club	41
6.1.1	s -Club Vertex Deletion is Fixed-Parameter Tractable	42
6.1.2	Data Reduction	44
6.2	Obtaining an s -Club Cluster Graph	45
6.2.1	s -Club Cluster Vertex Deletion is NP-complete	46
6.2.2	s -Club Cluster Vertex Deletion is FPT	49
6.2.3	s -Club Cluster Vertex Deletion on Trees	50
7	Outlook and Conclusion	53
7.1	Conclusion	53
7.2	Outlook	55
7.3	Acknowledgements	56
	Bibliography	57
	Selbstständigkeitserklärung	65

1 Introduction

The s -club concept was defined in the context of social science by Alba [Alb73]. Consider people in a social community and the relation “knowing each other”. Then, an s -club is a subgroup of persons that know each other directly, or over at most $s - 1$ persons which are in this subgroup as well. In general, people could be any set of objects and “knowing each other” could be an arbitrary relation. This work investigates the algorithmic complexity of finding s -clubs in large structures.

1.1 Motivation and Application

Although the term s -club was defined in 1973 [Alb73], the interest in these structures goes back to at least the 1960’s. Especially, there has been much interest in constructing s -clubs [HS60, ER62, Dam73, EFH80, II81, Del85, FHS95], often because one can guarantee some sort of efficiency in them. In this thesis the focus will be on already given networks in which we want to find “efficient” subnetworks. The task of finding an s -club of a particular size in a given structure is called the s -CLUB problem.

The s -CLUB problem is a generalization of the famous CLIQUE problem, since for $s = 1$ these problems are equivalent. There are mainly two arguments for analyzing the s -CLUB problem:

1. From a practical standpoint, CLIQUE may be too restrictive for the given problem. For example take the task of identifying structures in protein interaction networks. A protein interaction network is a set of proteins and the relation “known to interact”. But protein-protein interactions involve not only the direct-contact association of protein molecules but also longer range interactions [TS76]. Thus an interesting structure could interact over a distance of s interactions. This may not be revealed by identifying a clique, but certainly does by an s -club. In general the s -club structure is useful if the interaction not happen directly, but through at most s interactions.

2. From a parameterized complexity standpoint, the CLIQUE problem is not believed to be (fixed-parameter) tractable with respect to its solution size. More precisely it is W[1]-hard [DF99]. In contrast, the s -CLUB problem, with $s \geq 2$, is (fixed-parameter) tractable with respect to the parameters solution size and s [Kom07]. It is interesting to work out the difference in the parameterized complexity of CLIQUE and s -CLUB in more detail.

Based on these arguments this work focuses on the s -CLUB problem. In the following, the first argument will be additionally supported by giving possible or already known applications.

The s -CLUB problem found already consideration in the following real-world applications. Memon and Larsen [ML06] applied s -CLUB for the analysis of terror networks. In these networks the elements are terrorists and the relation is “known to work together”. We already mentioned protein-protein interaction networks. Pasupuleti [Pas08] used s -CLUB to analyze the structure of these networks. In the following we discuss possible applications in the context of networks with a specific property.

For the last decade the major topic in applied network science has been “small world” networks, mainly due to a paper by Watts and Strogatz [WS98], which has been cited more than 9000 times until today. This paper showed evidence that in many real-world networks the average path length is very short with respect to the number of elements in it. The authors called these networks small world networks, while referring to a famous experiment from Stanley Milgrim [TM69]. An interesting example for such a small world network is a network of mathematicians called the “Erdős Network” [Gro09]. In this network the Erdős number zero is assigned to Paul Erdős, who was a very productive mathematician, and the Erdős number one to researchers who published with him. The co-authors of people with Erdős number one have the Erdős number two, and so on, building one of the best studied small-world networks. There are many other networks which have been found to show small world properties. Examples include road maps [JC04], food webs [MS02], airplane passenger traffic [ASBS00], metabolite processing networks [FW00], neural networks [SZ04], mobile call graphs [NGD⁺06], tennis players and their matches [Sit07] and ownership links among German companies [KW01].

In general, a graph is considered a small world network if its average local clustering coefficient is significantly higher than a random graph constructed on the same vertex set, and if the graph has a short average path length. These are more or less global measurements. The s -CLUB problem directly identifies interesting local structures or “sub small worlds” of scalable diameter, on which further analysis can be performed. Therefore, solving the s -CLUB problem could be an

initial step of a detailed exact analysis of local structures in small world networks. Therefore, efficiently solving the s -CLUB problem is useful in many applications or could even form new ones.

1.2 Known Results

To our knowledge the first approach from computer science to the s -CLUB problem was made by Bourjolly et al. [BLP00]. They gave a heuristic algorithm that determines the vertex with maximum degree and uses it as the center of a star graph of maximum size. Butenko and Prokopyev [BP07] have shown that, unless $P = NP$, one cannot design a polynomial time algorithm for the s -CLUB problem, giving a larger solution than the trivial approach of picking a vertex with maximum degree and all its neighbors as the solution set. As a consequence, with the approach by Bourjolly et al. [BLP00] the heuristic approaches for the s -CLUB problem already seem to have reached their limits. To our knowledge the first proof of NP-completeness for the s -CLUB problem was given by Bourjolly et al. [BLP02]. In this work also a formulation of the s -CLUB problem as integer linear program (ILP) was given. Butenko et al. [BBT05] gave similar results, they also evaluated their integer linear program on biological data. Both publications gave no worst-case running time. In the following we use the variable k for the solution size of the s -CLUB problem. In approximation theory, Marincek and Mohar [MM02] showed that there is no better polynomial time approximation possible than $k^{1/3-\epsilon}$, for any $\epsilon > 0$, unless $P=NP$. The first fixed-parameter approach to the s -CLUB problem was made by Komusiewicz [Kom07]. He classified s -CLUB to be fixed-parameter tractable with respect to the combined parameter (k,s) .

1.3 Overview

The remaining part of this thesis is structured as follows. Chapter 2 is an introduction to several notions we use in this thesis. Section 2.1 gives a short introduction to some basic notation from graph theory. In Section 2.2 the most important definitions from parameterized complexity theory are introduced. We introduce search trees of bounded depth in Section 2.3 and give a brief introduction to polynomial time data reduction in Section 2.4, including the concepts of many-to-one kernelization (Section 2.4.1) and Turing kernelization (Section 2.4.2).

Chapter 3 gives the definition of the s -CLUB problem and the corresponding solution set s -club. Then, in Section 3.1 an overview of interesting aspects of s -clubs and the s -CLUB problem is given. Section 3.2 describes two concepts which are clique relaxations similar as the s -club concept is. Furthermore, Section 3.3 mentions two problems which are concerned with graphs of bounded diameter like the s -CLUB problem.

Chapter 4 presents our results for s -CLUB on general graphs. In Section 4.1 we give two results on problem kernelization. In detail we show that s -CLUB cannot have a polynomial-size many-to-one kernel in Section 4.1.1 and give a k^3 -vertex Turing kernel in Section 4.1.2. A fixed-parameter algorithm for s -CLUB on general graphs is given in Section 4.2.

Chapter 5 presents our results for s -CLUB on special graph classes. Section 5.1 gives an algorithm to solve s -CLUB on trees in $O(n \cdot s^2)$ time. Section 5.2 gives an algorithm to solve s -CLUB on interval graphs in $O(n^2)$ time. Section 5.3 shows how to solve 2-CLUB on bipartite graphs in $O(n^5)$ time. Section 5.4 classifies s -CLUB to be polynomial time solvable on graph classes with bounded clique- or treewidth. Section 5.5 classifies s -CLUB to be fixed-parameter tractable with respect to parameter s on planar graphs.

Chapter 6 introduces two vertex deletion problems which are closely related to s -CLUB. In Section 6.1 we analyze the s -CLUB VERTEX DELETION problem, which is a dual parameterization of s -CLUB. More precisely, we will show that s -CLUB VERTEX DELETION is fixed-parameter tractable with respect to its solution size in Section 6.1.1 and give data reduction rules for this problem in Section 6.1.2. In Section 6.2 we analyze the s -CLUB CLUSTER VERTEX DELETION problem, which asks to delete vertices in the input graph such that in the remaining graph every connected component is an s -club. In Section 6.2.1 we show that s -CLUB CLUSTER VERTEX DELETION is NP-complete. In Section 6.2.2 we show that this problem is fixed-parameter tractable with respect to the combined parameter solution size and s . Section 6.2.3 shows that s -CLUB CLUSTER VERTEX DELETION is solvable on trees in $O(n \cdot s)$ time.

Finally, Chapter 7 gives some conclusions from this thesis in Section 7.1 and an outlook for future research in Section 7.2.

2 Basic Notations and Definitions

This chapter summarizes some basic notations used throughout this work and provides a brief introduction to parameterized complexity theory.

2.1 Basic Notation in Graph Theory

This section provides the reader with basic terms and definitions used throughout this thesis. For a general introduction to graph theory, see [Die05].

In this work we only consider undirected graphs. An undirected *graph* is defined as a tuple (V, E) , where V is a finite set of *vertices* and E is a set of *edges*, which are unordered tuples of vertices, that is, $E \subseteq \{\{u, v\} \mid u, v \in V\}$. By $V(G)$ we denote the set of vertices in G and by $E(G)$ is the set of edges in G . Throughout this work, let $n := |V|$ and $m := |E|$. A vertex v is called *incident* edge e , if $v \in e$. Two vertices v, w are called *adjacent* if there exists an edge e such that $v, w \in e$.

A *path* P_i in a graph G is a sequence of i vertices (p_1, \dots, p_i) such that for all $1 \leq j \leq i$ $\{p_{j-1}, p_j\} \in E$. The length of a path is the number of edges traversed. Two vertices in this sequence are called *connected*. If $p_1 = p_n$ the sequence is called a *cycle*. A *connected graph* is a graph $G = (V, E)$ such that every pair $u, v \in V$ is connected. The *distance* $d(u, v)$ between two vertices u and v of a graph is the minimum length among all paths connecting them. If no such path exists, then the distance is set equal to infinity. The *diameter* of a graph is the length of the longest shortest path between any two graph vertices (u, v) of a graph and will be denoted as $\text{diam}(G)$.

The *degree* $\text{deg}(u)$ of a vertex u is the number of edges incident to u . The (open) *i -neighborhood* of v is the set of all vertices that lie at distance at most i from v : $N_i(v) = \{u \mid 1 \leq d(u, v) \leq i\}$. The *closed i -neighborhood* of v is $N_i[v] = N_i(v) \cup \{v\}$. The *exact i -neighborhood* of v is the set of all vertices that lie at distance exact i from v : $N_i^{\circ}(v) = \{u \mid d(u, v) = i\}$.

For a vertex set S , we use $G[S]$ to denote the *subgraph* of G induced by S , with vertex set S and edge set $E' = \{\{u, v\} \in E(G) \mid u, v \in S\}$. A graph property \mathcal{P} is *hereditary* if whenever a graph G obeys \mathcal{P} , then all induced subgraphs of G obey \mathcal{P} also. Let $G = (V, E)$ and $A \subseteq V$, then we define $G - A := G[V \setminus A]$.

A *tree* $T = (V, E)$ is a connected graph without cycles. A tree is called a *rooted tree* if one vertex has been designated the *root*, in which case the edges have a natural orientation, towards or away from the root. A vertex v with $\deg(v) = 1$ is a *leaf*. A *lowest leaf* is a vertex with the maximum distance to the root. The *father* of a vertex v is the vertex that is adjacent to v and lies on the path from v to the root. The *children* of a vertex v are the vertices that are adjacent to v and are not the father of v and will be denoted as $\text{child}(v)$. The *subtree* T_v is induced by all the vertices connected to v via paths that do not contain the father of v . The *height* of a vertex v is the distance to the lowest leaf in the induced subtree T_v and will be denoted as $h(v)$. The *level* of a vertex v is distance from a lowest leaf to the root, minus the actual distance of v to the root and will be denoted as $l(v)$.

2.2 Parameterized Complexity

Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems. One dimension is the input size n (as in classical complexity theory), and the other one is the parameter k (in this work the size of the solution).

We begin with the basic definitions of parameterized complexity theory given by Downey and Fellows [DF99]:

Definition 2.1. A parameterized problem is a language $L \subseteq \Sigma^* \times \Sigma^*$, where Σ is a finite alphabet. The second component is called the parameter of the problem.

Throughout this work the parameter is a nonnegative integer. Thus, we will usually write $L \subseteq \Sigma^* \times \mathbb{N}$ instead of $L \subseteq \Sigma^* \times \Sigma^*$. For $(x, k) \in L$, the two dimensions of the parameterized complexity analysis are then the input size $n := |(x, k)|$ and the parameter k . In this work we also need to know what the *unparameterized version* \hat{L} of L is. Therefore we define \hat{L} by $\hat{L} = \{x\#1^k \mid (x, k) \in L\}$ where $\#$ is the blank letter and 1 is any letter from Σ .

Definition 2.2. A parameterized problem L is fixed-parameter tractable if there is an algorithm that decides in $f(k) \cdot n^{O(1)}$ time whether $(x, k) \in L$, where f is an arbitrary computable function depending only on k . The complexity class that contains the fixed-parameter tractable problems is called FPT.

Note that the running time of a fixed-parameter algorithm is polynomial in the instance size and exponential only in the parameter k . Hence, if k is fixed at a small value, a problem in FPT can still be considered “tractable” despite its traditional classification as “intractable”.

2.3 Search Trees of Bounded Size

In this work we will use the concept of search trees of bounded size to give fixed-parameter algorithms. A search tree is a systematic exhaustive search, organized in a tree-like fashion. The basic idea is to find in polynomial time a small part of the input such that at least one element of that part has to be in an optimal solution. We then branch into several cases of choosing an element of the small part to be in the solution, and then proceed recursively until a solution is found. A search tree corresponds to the recursive calls of such an algorithm. If we can bound the number of cases in each search tree node as well as the height of the tree, then we obtain a fixed-parameter algorithm.

Search tree algorithms work in a recursive manner. The number of recursion calls is the number of nodes in the according tree. This number is governed by linear recurrences with constant coefficients. These can be solved by standard mathematical methods [Nie06]. If the algorithm solves a problem of size n and calls itself recursively for problems of sizes $n - d_1, \dots, n - d_i$, then (d_1, \dots, d_i) is called the *branching vector* of this recursion. It corresponds to the recurrence $T_n = T_{n-d_1} + \dots + T_{n-d_i}$ for the asymptotic size T_n of the overall search tree.

In order to give an instance for a search tree of bounded size, we start by introducing the parameterized VERTEX COVER problem, which is defined as follows:

VERTEX COVER

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Question: Is there vertex set $C \subseteq V$ of size at most k , such that each edge of G is incident with at least one element of C ?

For parameterized VERTEX COVER, where the size of a vertex cover is bounded by a parameter k , a small part of the input that contains at least one element of an optimal solution is a single edge in the graph, because we know that each edge must be covered by at least one of its endpoints. Thus, we select an arbitrary edge $e = \{u, v\}$ of the graph, and branch into the two subcases of adding u or v to the vertex cover. At least one of the two assumptions is correct. Then, if we

decided that for example u is in the vertex cover, then we can delete u and all incident edges from the graph and proceed with the remaining instance. In each branching step, one vertex is deleted from the graph. After at most k recursive steps we either obtain an instance without edges, and we have found a solution, or the remaining instance has still some edges, which means that the corresponding path from the search tree root to the leaf cannot lead to a solution of size at most k . The branching vector corresponding to this recursion is $(1, 1)$. This leads to a search tree of size $O(2^k)$. The selection of an edge and the branching can be done in $O(n)$ time and therefore we obtain a fixed-parameter algorithm with running time $O(2^k \cdot n)$.

2.4 Data Reduction

The idea of data reduction is the elimination of polynomial time “solvable” parts of the input data, to obtain “hard” cores. Then time intensive algorithms are only applied to this cores. Data reduction is also useful for other techniques in the area of algorithms for solving hard problems (such as approximation, heuristics or ILP formulations).

2.4.1 Many-to-One Kernelization

We now give the original definition of kernelization as it was introduced by Downey and Fellows [DF99]. This definition of kernelization reduces the input instance to exactly one kernel and is defined as follows:

Definition 2.3. *Let L be a parameterized problem. A reduction has a problem kernel or kernelization is a transformation of an instance (x, k) to an instance (x', k') , such that:*

1. $(x, k) \in L \Leftrightarrow (x', k') \in L$
2. $|x'| \leq g(k)$ for some arbitrary computable function g depending only on k
3. $k' \leq k$, and
4. the transformation runs in polynomial time.

The size of a reduced instance does only depend on the size of the parameter k , and not on the original input n anymore. In this work we will often call this kernelization a many-to-one kernelization.

2.4.2 Turing Kernelization

As recent work has shown, the traditional kernelization is sometimes too restrictive. This is also the case for problems we analyze in this work. Therefore we also give the definition of Turing kernelization, as recently introduced by Lokshatnov [Lok09]. In order to define Turing kernels, we first define an t -oracle:

Definition 2.4. [Lok09] *A t -oracle for a parameterized problem Π is an oracle that takes as input (I, k) with $|I| \leq k$, $k \leq t$ and decides $(I, k) \in \Pi$ in constant time.*

Definition 2.5. [Lok09] *A parameterized problem Π is said to have a $g(k)$ -sized Turing kernel if there is an algorithm which, given an input (I, k) together with a $g(k)$ -oracle for Π , decides whether $(I, k) \in \Pi$ in time polynomial in $|I|$ and k .*

Note that a many-to-one kernel is equivalent to a Turing kernel where the kernelization algorithm is allowed to make only one oracle call and must return the same answer as the oracle. Since the definition of Turing kernelization is more general than it will be necessary in this thesis, we illustrate how the idea will be used. The Turing kernelization algorithm reduces the input instance (I, k) to $|I|$ independent kernels of size at most $g(k)$. In more detail, the algorithm outputs instances $(Z_1, k'), \dots, (Z_{|I|}, k')$ such that (I, k) is a *yes* instance if and only if (Z_i, k') is a *yes* instance for some i , and if $|Z_i| \leq g(k)$ for every i .

3 s -Club and Related Problems

This chapter starts by giving a definition of s -clubs and the corresponding s -CLUB problem. Then, in Section 3.1 an overview of interesting aspects of s -clubs and the s -CLUB problem is given. Section 3.2 introduces two further concepts which are clique relaxations like the s -club concept. Furthermore, Section 3.3 introduces two problems which are concerned with graphs of bounded diameter like the s -CLUB problem is.

In this work we call a vertex set an s -club, if it fulfills the following definition:

Definition 3.1. *Let G be a graph with $G = (V, E)$, then an s -club is a subset of vertices $S \subseteq V$ such that the diameter of $G[S]$ is at most s .*

The task of finding an s -club is called the s -CLUB problem. More precisely, this problem is defined as follows:

s -CLUB

Input: An undirected graph $G = (V, E)$ with integers $s, k \geq 2$.

Question: Is there a set of vertices $S \subseteq V$ of size at least k such that $G[S]$ has diameter at most s ?

The s -CLUB problem is NP-complete [BLP02, BBT05], and this remains true even in graphs of diameter $s + 1$ [BP07]. With $s = 1$ the s -CLUB problem would be equivalent to the CLIQUE problem, which is known to be W[1]-hard with respect to its solution size [DF99]. W[1]-hard means that the CLIQUE problem is not believed to be fixed-parameter tractable. Therefore, many of our results will not work for $s = 1$ and we define the s -CLUB problem with $s \geq 2$.

3.1 Basic Observations Concerning s -Clubs

In this section we give a detailed overview one interesting aspects of s -clubs and the s -CLUB problem.

3.1.1 s -Club Definitions

Originally [Alb73], s -clubs were defined by demanding additionally to Definition 3.1 the maximality with respect to inclusion. Consider for example, the graph in Figure 3.1(a). By the original definition $\{a, b, c\}$ is not a 2-club since it is not maximal, in contrast $\{a, b, c, d, e\}$ is a 2-club. Balasundaram et al. [BBT05] discussed that, assuming the original definition [Alb73], checking whether a given subset is an s -club would be a non-trivial problem itself. Take again the graph given in Figure 3.1(a). The subgraph $G[S]$ with $S = \{a, b, c\}$ has diameter two. Adding one of the vertices e or d to S would increase the diameter of the induced subgraph; however, if both vertices are added, the diameter is two. In this work we follow this argumentation of Balasundaram et al. and use Definition 3.1.

3.1.2 The Non-Hereditariness of s -Club

A graph has the s -club property if it has diameter at most s . Now we show that the s -club property is non-hereditary by a simple example illustrated in Figure 3.1. The subgraph $G[\{a, b, c, e\}]$ given in Figure 3.1(b) has diameter 3 whereas the original graph in Figure 3.1(a) has diameter two. Thus $G[\{a, b, c, d, e\}]$ is a 2-club, whereas its subgraph $G[\{a, b, c, e\}]$ is not a 2-club. Hence, s -club is a non-hereditary graph property. This is one of the difficulties throughout this work, since it is unclear whether standard techniques work with respect to non-hereditary properties.

3.1.3 Monadic Second Order Logic Formulation of s -Club

Monadic second order logic (MSO) is an important tool in computational complexity analysis. For problems which are expressible in monadic second order logic, one can quickly classify the complexity of this problem on graphs with bounded clique- or treewidth [CO00, CMR01]. For more details see the books by Flum and Grohe [FG06] or by Niedermeier [Nie06]. We will use the following MSO formulation of s -CLUB in Section 5.4 and Section 5.5.

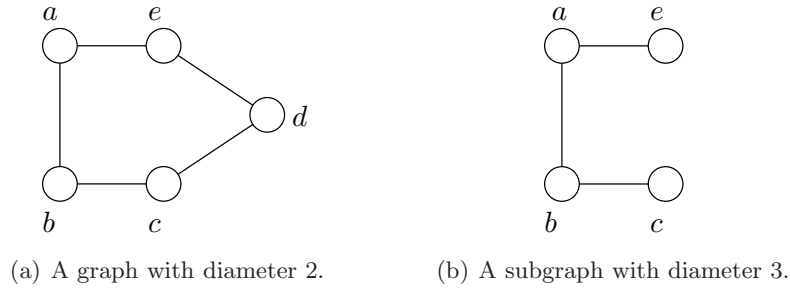


Figure 3.1: The subgraph $G[\{a, b, c, e\}]$ in (b) has a higher diameter than the original graph G in (a).

The formulation in Equation 3.1 translates the definition of s -clubs into MSO. To do so, the formulation uses the terms $S(u)$ and $E(u, v)$, where $S(u)$ is equivalent to “ $u \in S$ ” and $E(u, v)$ is equivalent to “ u and v are adjacent”. Using this, the maximization version of the s -CLUB problem can be expressed in Monadic Second Order Logic as follows:

$$\begin{aligned}
 \max S \forall u, v (S(u) \wedge S(v) \rightarrow & [E(u, v) \\
 & \vee \exists x_1 (S(x_1) \wedge E(u, x_1) \wedge E(x_1, v)) \\
 & \vee \exists x_1, x_2 (S(x_1) \wedge S(x_2) \wedge E(u, x_1) \wedge E(x_1, x_2) \wedge E(x_2, v)) \\
 & \vdots \\
 & \vee \exists x_1, x_2, \dots, x_s (S(x_1) \wedge S(x_2) \wedge \dots \wedge S(x_s) \wedge E(u, x_1) \wedge E(x_1, x_2) \wedge \dots \wedge E(x_s, v))] \\
 & (3.1)
 \end{aligned}$$

3.1.4 Integer Linear Program for s -Club

A general way to solve combinatorial problems is to translate them into optimization problems and utilize linear programming techniques. An *integer linear program* (ILP) is a linear program where the values of the decision variables are restricted to integers. For more details on integer linear programs see the book by Schrijver [Sch98].

For the s -CLUB problem Bourjolly et al. [BLP02] gave the following formulation as an integer linear program:

$$\text{maximize } \sum_{i \in V} x_i \tag{3.2}$$

subject to:

$$\sum_{i \in C_{ij}^s} y_t \geq x_i + x_j + 1 \quad \forall \{i, j\} \notin E \quad C_{ij}^s \neq \emptyset \quad (3.3)$$

$$y_t \leq x_r \quad \forall t \in C \quad \forall r \in V_t \quad (3.4)$$

$$x_i + x_j \leq 1 \quad \forall \{i, j\} \notin E \quad C_{ij}^s \neq \emptyset \quad (3.5)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (3.6)$$

$$x_j \in \{0, 1\} \quad \forall j \in V \quad (3.7)$$

where C_{ij}^s is the set of all paths of length at most s between vertices i and j in G and where y_t is an auxiliary binary variable associated with every path $P \in \cup_{i,j \in V} C_{ij}^s$. This formulation ensures that if there are two vertices i and j in a s -club, then all the vertices on the paths with length at most s between them are included in this s -club as well. Although we will not utilize the ILP formulation of s -CLUB in this thesis, it may be a starting point for further investigations or practical implementations.

3.1.5 Dual Parameterization for s -Club

The idea of fixed-parameter algorithms is to confine the exponential running time to a small parameter. For the s -CLUB problem one possible parameter is the number k of vertices in the solution. If one wants to identify an s -club which has almost as many vertices as the whole graph, this will not be efficient any more. The idea of s -CLUB VERTEX DELETION is to find the complement set of an s -club. The size of this set is $n - k$, and if k is close to n , then under the assumption of an fixed-parameter algorithm for s -CLUB VERTEX DELETION we will find the s -club again efficiently. In Section 6.1 the s -CLUB VERTEX DELETION problem is discussed in more detail.

3.2 s -Club and Other Clique Relaxations

This section introduces two concepts which are clique relaxations, like the s -club concept, and their corresponding decision problems. For $s = 1$, these relaxations are identical to the clique concept. An overview on these relaxations and clique is given in Figure 3.2.

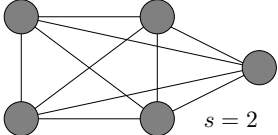
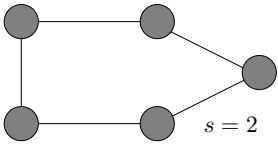
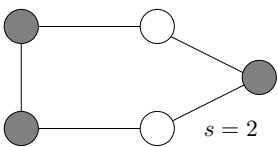
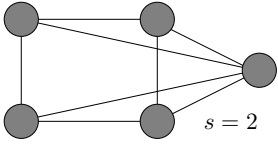
Name	Definition	Example
clique	$S \subseteq V$ is a clique if and only if $G[S]$ is a complete graph	
<i>s</i> -club	$S \subseteq V$ is an <i>s</i> -club if and only if $G[S]$ has diameter at most <i>s</i>	
<i>s</i> -clique	$S \subseteq V$ is an <i>s</i> -clique if and only if $\forall u, v \in S : d(u, v) \leq s$	
<i>s</i> -plex	$S \subseteq V$ is an <i>s</i> -plex if and only if $\min_{v \in S} \deg_{G[S]}(v) \geq S - s$	

Figure 3.2: An overview over clique and three clique relaxations. The vertices in the respective set S are marked grey.

An *s*-clique is defined in Figure 3.2. The *s*-clique concept, like the *s*-club concept, relaxes over the maximal distance between each of the vertices in the set. The *s*-CLIQUE problem is defined as follows:

s-CLIQUE

Input: An undirected graph $G = (V, E)$ and positive integers k, s .

Question: Is there a set of vertices $S \subseteq V$ of size at least k such that for any two vertices $u, v \in S$, $d(u, v) \leq s$?

The *s*-CLIQUE problem is NP-complete [BLP02] and fixed-parameter tractable with respect to the combined parameter (k, s) [Kom07]. In the following we describe the difference between an *s*-clique and an *s*-club. Every *s*-club is an *s*-clique but not vice versa. Take for example the graph G given in Figure 3.3. The set $A = \{a, b, d, f, g\}$ is a 2-clique but not a 2-club, since for example the vertices f and d are connected via a length-2 path in G but not in $G[A]$. Hence,

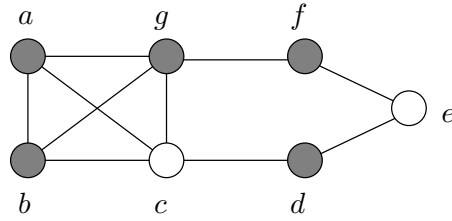


Figure 3.3: The grey marked set $A = \{a, b, d, f, g\}$ is a 2-clique but not a 2-club.

s -club is in this sense a more self-contained structure than s -clique. However, the results for the s -CLUB problem given in this work are easily transferable to the s -CLIQUE problem, by working with the distance in the input graph instead of the distance in the induced subgraph.

In the following we motivate the s -plex concept and present its definition. Seidman and Foster [SF78] observed that s -cliques, and by that also s -clubs, are often not robust. Seidman and Foster measured the robustness of a subgraph in terms of “the degree of which the structure is vulnerable to the removal of any given individual”. Consider a 2-club S of size k , which is a star. The induced subgraph $G[S]$ is according to Seidman and Forster not robust, since we could remove the vertex v with $\deg(v) = k - 1$ and remain with $k - 1$ unconnected components. To overcome this problem Seidman and Foster introduced the s -plex concept. An s -plex is defined in Figure 3.2. In contrast to the s -club concept, the s -plex concept relaxes over the degree of the vertices in the defined set. Using the definition of s -plex from Figure 3.2, the s -PLEX problem is defined as follows:

s -PLEX

Input: An undirected graph $G = (V, E)$ and positive integers k, s .

Question: Is there an s -plex $S \subseteq V$ of size at least k ?

The s -PLEX problem was shown to be NP-complete [BBHS09] and $W[1]$ -hard with respect to the combined parameter (s, k) [KHMN09]. For this problem there exist several further results [GKNU09, MNS09, Pei09].

In conclusion, s -club is a self-contained, but not necessary robust structure. This should be considered in real-world applications.

3.3 *s*-Club and Other Bounded Diameter Problems

This section introduces two problems which are also concerned with graphs of bounded diameter. The first one is a problem in discrete mathematics. The second one is a problem in algorithmic complexity, which unfortunately turns out to be W[2]-hard.

In the context of graphs with bounded diameter, one of the best studied problems is the DEGREE/DIAMETER problem, which is defined as follows:

DEGREE/DIAMETER

Input: A diameter D , a maximum degree δ , a number of vertices n .

Question: Can we construct a graph G of diameter D , with n vertices, which each have degree at most δ ?

An application for this problem is in the design of networks. In the design process there are a number of features that must be taken into account. The most common ones are limitations on the vertex degrees and on the diameter.

However, the number of vertices in a graph with degree δ and diameter D is upper-bounded by the *Moore bound* [HS60]:

$$\delta + (\delta - 1)\delta + \dots + \delta(\delta - 1)^{D-1} = \frac{\delta(\delta - 1)^D - 2}{\delta - 2} = N(\delta, D) \quad (3.8)$$

Research activities related to the DEGREE/DIAMETER problem fall into two main streams. On the one hand, there are proofs of non-existence of graphs of order close to the general Moore bound [Dam73, Ple74]. On the other hand, there is much activity in the construction of large graphs [HS60, II81, Del85, FHS95], providing better lower-bounds. For more on the DEGREE/DIAMETER Problem see the survey papers by Chung [Chu87] or by Miller and Siran [MS05].

The DEGREE/DIAMETER problem is interested in the construction of large graphs of bounded diameter. In this thesis the focus will be on already given graphs in which we want to find large graphs of bounded diameter.

Another interesting problem is the MAXIMUM DIAMETER EDGE ADDITION problem, which is defined as follows:

MAXIMUM DIAMETER EDGE ADDITION

Input: An undirected, connected graph $G = (V, E)$ and positive integers D, k .

Question: Can we obtain a supergraph G' of G by adding k edges to G , such that G' has diameter of at most D ?

The problem is also known as GRAPH DIAMETER- D AUGMENTATION problem. An application for this problem is the maintenance of networks. Think of an inefficient network in terms of long pathways between the elements. One may ask the question: Can we make the network efficient again, in terms of a longest pathway with length at most D , by adding at most k new connections to the network? This is the MAXIMUM DIAMETER EDGE ADDITION problem in the graph corresponding to the network.

The MAXIMUM DIAMETER EDGE ADDITION problem was first defined and analyzed by Schoone et al. [SBL87], who could show that this problem is NP-complete for $D \geq 3$. Li et al. [LMSL92] strengthened the result by showing that MAXIMUM DIAMETER EDGE ADDITION is NP-complete even for $D = 2$. Recently, Nastos and Gao [NG09] have shown that this problem is also W[2]-hard for $D \geq 2$. This means that this problem is presumably not fixed-parameter tractable.

4 s -Club on General Graphs

In this chapter we investigate the s -CLUB problem on general graphs. Section 4.1 analyzes two different kernelization techniques for s -CLUB. Section 4.2 gives a branching algorithm to solve s -CLUB on general graphs.

4.1 A Problem Kernel for the s -Club Problem

Section 4.1.1 shows that the s -CLUB problem does not admit a polynomial many-to-one kernel unless there is an unlikely collapse of complexity classes. In contrast, Section 4.1.2 shows that there is a k^3 -vertex Turing kernel for s -CLUB.

4.1.1 A Lower Bound for Many-to-One Kernels for the s -Club Problem

Using a lower bounds engine from Bodlaender et al. [BDFH09], we show that s -CLUB does not admit a polynomial size kernel, unless the polynomial hierarchy collapses to the third level. This kernelization lower bound engine [BDFH09] was already used to show that a variety of problems [BTY09, DLS09, FFL⁺09, KW09] do not admit polynomial kernels under this assumption. In order to use the engine the terms *composition algorithm* and *compositional parameterized problem* must be defined:

Definition 4.1. [BDFH09] A composition algorithm for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that receives as input a sequence $((x_1, k), \dots, (x_t, k))$, with $(x_i, k) \in \Sigma^* \times \mathbb{N}^+$ for all $1 \leq i \leq t$, uses time polynomial in $\sum_{i=1}^t |x_i| + k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}^+$ with:

1. $(y, k') \in L \Leftrightarrow (x_i, k) \in L$ for some $1 \leq i \leq t$ and
2. k' is polynomial in k .

A parameterized problem is compositional if there exists a composition algorithm for it.

If we can show that the s -CLUB problem is compositional, then we can use the following lemma:

Lemma 4.1. [BDFH09, FS08] *Let L be a compositional parameterized problem whose unparameterized version \hat{L} is NP-complete. If L has a polynomial kernel, then the polynomial hierarchy collapses to the third level.*

In order to show that the s -CLUB problem is compositional, we characterize a compositional parameterized graph problem by the following lemma:

Lemma 4.2. [BDFH09] *Let L be a parameterized graph problem such that for any pair of graphs G_1 and G_2 , and any integer $k \in \mathbb{N}$, we have $((G_1, k) \in L \vee (G_2, k) \in L) \Leftrightarrow (G_1 \uplus G_2, k) \in L$, where $G_1 \uplus G_2$ is the disjoint union of G_1 and G_2 . Then L is compositional.*

Now we utilize Lemma 4.2 to show that the s -CLUB problem is compositional even if parameterized by k .

Lemma 4.3. *The s -CLUB problem is compositional.*

Proof. We have to show that (G_1, k) or (G_2, k) is a yes-instance if and only if $(G_1 \uplus G_2, k)$ is a yes-instance.

“ \Rightarrow ”: Assume that (G_1, k) or (G_2, k) is a yes-instance. In a disjoint union of G_1 and G_2 , the graphs G_1 and G_2 will exist as subgraphs in $(G_1 \uplus G_2, k)$. Since s -CLUB is defined over subgraphs and (G_1, k) or (G_2, k) is a yes-instance, the following statement holds: if (G_1, k) or (G_2, k) is a yes-instance, $(G_1 \uplus G_2, k)$ is also a yes-instance.

“ \Leftarrow ”: Assume that $(G_1 \uplus G_2, k)$ is a yes-instance. Because of the properties of the disjoint union, the subgraphs G_1 and G_2 are not connected in $(G_1 \uplus G_2, k)$. Since the diameter of unconnected components is infinite, subgraphs containing vertices from G_1 and G_2 are of no interest and at least one of the subgraphs (G_1, k) or (G_2, k) has to form a yes-instance in $(G_1 \uplus G_2, k)$. Hence, if $(G_1 \uplus G_2, k)$ is a yes-instance, then (G_1, k) or (G_2, k) is a yes-instance. \square

By using the facts that s -CLUB is compositional, that its unparameterized counterpart is NP-complete [BLP02], and by applying Lemma 4.1, the following statement holds.

Theorem 4.1. *Parameterized s -CLUB cannot have a polynomial kernel, unless the polynomial hierarchy collapses to the third level.*

4.1.2 A Turing Kernel for the s -Club Problem

In Section 4.1.1 we observed that the s -CLUB problem does not admit a polynomial kernel in a many-to-one kernelization. Fernau et al. [FFL⁺09] recently showed an analogous result for the k -LEAF OUT-BRANCHING and k -LEAF OUT-TREE problem. In contrast to this result they also gave a Turing kernelization for these two problems. In the following we will show an analogous result for the s -CLUB problem.

To begin with, a simple observation for the s -CLUB problem can be made:

Observation 4.1. *If a vertex v is part of a solution S for the s -CLUB problem, then only vertices in the s -neighborhood of v can also be part of S .*

Observation 4.1 follows from the fact that all other vertices have distance greater than s to the vertex v . Next, a reduction rule is introduced to shrink the number of vertices in the s -neighborhood of a vertex:

Reduction Rule 1. *If there exists a vertex $v \in V$ with $|N_{\lfloor s/2 \rfloor}(v)| \geq k - 1$, then set $S = N_{\lfloor s/2 \rfloor}[v]$ and return “yes”.*

Lemma 4.4. *Reduction Rule 1 is correct and can be performed in $O(n(n + m))$ time.*

Proof. To show that Reduction Rule 1 is correct, we show that $G[S]$ is an s -club, that is, the pairwise distance of any two vertices $x, y \in N_{\lfloor s/2 \rfloor}[v]$ is at most s . Since $x, y \in N_{\lfloor s/2 \rfloor}[v]$ and by the definition of $N_{\lfloor s/2 \rfloor}[v]$, the distance from x to v is at most $\lfloor \frac{s}{2} \rfloor$ and the distance from v to y is at most $\lfloor \frac{s}{2} \rfloor$. Consequently, the distance between x and y is at most s . Thus, Reduction Rule 1 is correct. Computing the set $N_{\lfloor s/2 \rfloor}[v]$ can be done in $O(n + m)$ time for one vertex by breadth-first search. Checking whether Reduction Rule 1 can be applied to any vertex thus takes $O(n(n + m))$ total time. \square

By Section 4.1.1 we know Reduction Rule 1 is of no help to find a polynomial “classic” kernel. Therefore, we describe in the following how to obtain a Turing kernel. Recall that a t -oracle for s -CLUB is an oracle that takes as input (I, k) and decides $(I, k) \in s$ -CLUB in constant time. As input (I, k) we provide the t -oracle with n s -neighborhoods, one for each vertex from the input graph, and the parameter k . Due to Observation 4.1 it is obvious that there exists an s -club of size k in G if and only if there exists an s -club of size k in one of these s -neighborhoods. Hence, the Turing kernel is correct. To form a Turing kernel, the size of each s -neighborhood has to be bounded by some function $g(k)$. Therefore,

we assume in the following that Reduction Rule 1 has been applied for each vertex in G , and show that this upper-bounds the s -neighborhood of each vertex $v \in V(G)$. For this we will distinguish between two cases depending on s . First, we will show that s -CLUB admits a k^2 -vertex Turing kernel when s is even, then we will show that s -CLUB admits a k^3 -vertex Turing kernel when s is odd.

Recall that the exact i -neighborhood of a vertex v is defined as: $N_i^\circ(v) = \{u \mid d(u, v) = i\}$. Using $N_i^\circ(v)$, we observe, that $N_s[v]$ is expressible by the union of two sets:

Claim 4.1. *When s is even, $N_s[v] = N_{s/2}[v] \cup N_{s/2}(N_{s/2}^\circ(v))$.*

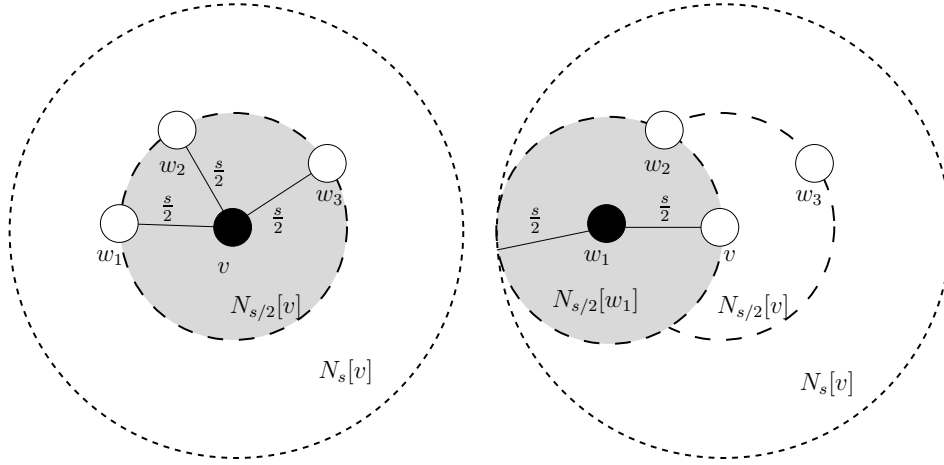
Proof. We now show that Claim 4.1 is correct: Any vertex that has distance of at most $\frac{s}{2}$ from v is in $N_{s/2}[v]$, this follows from the definition of the $\frac{s}{2}$ -neighborhood. Any vertex x with $\frac{s}{2} \leq d(v, x) \leq s$ has distance of at most $\frac{s}{2}$ to at least one vertex in $N_{s/2}^\circ(v)$ and thus belongs to $N_{s/2}(N_{s/2}^\circ(v))$. \square

Now we utilize Claim 4.1 for the following theorem.

Theorem 4.2. *For even s , s -CLUB admits a k^2 -vertex Turing kernel, which can be computed in $O(n(n + m))$ time.*

Proof. To show the correctness, we bound the size of the set $N_s[v]$, by bounding the sizes of $N_{s/2}[v]$ and $N_{s/2}(N_{s/2}^\circ(v))$. With Reduction Rule 1 applied for vertex v , as illustrated in Figure 4.1(a), $N_{s/2}[v]$ contains at most $k - 1$ vertices. Thereby also the size of the set of vertices $N_{s/2}^\circ(v)$ is upper-bounded by $k - 2$. With Reduction Rule 1 applied to each vertex $w \in N_{s/2}^\circ(v)$, as illustrated in Figure 4.1(b), the size $|N_{s/2}(w)|$ of the $\frac{s}{2}$ -neighborhood for each w is upper-bounded by $k - 1$. Thus the size of $N_{s/2}(N_{s/2}^\circ(v))$ is at most $k^2 - 2k + 1$. Clearly, the size of $N_{s/2}[v] \cup N_{s/2}(N_{s/2}^\circ(v))$ is at most $k - 1 + k^2 - 2k + 1 \leq k^2$ and by Claim 4.1 we know that $|N_s[v]| \leq k^2$. By Lemma 4.4 we know that applying Reduction Rule 1 to the input graph G takes at most $O(n(n + m))$ time. This is sufficient, since reducing the input graph G reduces also the s -neighborhood of each vertex in G . \square

If s is odd, then the kernelization is different. In the case of even s we can bound the s -neighborhood of v by restricting the number of vertices of two subsets, for odd s we bound three subsets. In the following, we show that for any vertex $v \in V$ there exists a k^3 -vertex Turing kernel for the s -CLUB problem with odd s . Because s is odd, $\lfloor \frac{s}{2} \rfloor = \frac{s-1}{2}$. For the ease of presentation, $N_{(s-1)/2}$ is now being used instead of $N_{\lfloor s/2 \rfloor}$. Using the $\frac{s-1}{2}$ -neighborhood, we can express $N_s[v]$ by the union of three sets:



(a) Reduction Rule 1 on v bounds the $\frac{s}{2}$ -neighborhood of v to at most $k-1$ vertices.

(b) Reduction Rule 1 on w_1 bounds the $\frac{s}{2}$ -neighborhood of w_1 to at most $k-1$ vertices.

Figure 4.1: Turing kernelization via Reduction Rule 1 for the s -CLUB problem, with even s .

Claim 4.2. $N_s[v] \subseteq N_{(s-1)/2}[v] \cup N_{(s-1)/2}(N_{(s-1)/2}^\circ(v)) \cup N_{(s-1)/2}(N_{s-1}^\circ(v))$.

Proof. Any vertex which has distance of at most $\frac{s-1}{2}$ from v is in $N_{(s-1)/2}[v]$, this follows from the definition of the $\frac{s-1}{2}$ -neighborhood. Any vertex y with $\frac{s-1}{2} \leq d(v, y) \leq s-1$ has distance of at most $\frac{s-1}{2}$ to at least one vertex in $N_{(s-1)/2}^\circ(v)$ and thus belongs to $N_{(s-1)/2}(N_{(s-1)/2}^\circ(v))$. Any vertex z with $s-1 \leq d(v, z) \leq 3\frac{s-1}{2}$ has distance of at most $\frac{s-1}{2}$ to at least one vertex in $N_{s-1}^\circ(v)$ and thus belongs to $N_{(s-1)/2}(N_{s-1}^\circ(v))$. Since $3\frac{s-1}{2} = 1.5s - 1.5 \geq s$ for $s \geq 3$, Claim 4.2 holds. \square

No we utilize Claim 4.2 for the following theorem.

Theorem 4.3. For odd s , s -CLUB admits a k^3 -vertex Turing kernel, which can be computed in $O(n(n+m))$ time.

Proof. To show correctness, we bound the size of the set $N_s[v]$, by upper-bounding the number of vertices in each of the three sets $N_{(s-1)/2}[v]$, $N_{(s-1)/2}(N_{(s-1)/2}^\circ(v))$ and $N_{(s-1)/2}(N_{s-1}^\circ(v))$. Reduction Rule 1 applied for vertex v secures that the closed $\frac{s-1}{2}$ -neighborhood of v consists of at most $k-1$ vertices. By that also the size of $N_{(s-1)/2}^\circ(v)$ is bounded to at most $k-2$. Reduction Rule 1 applied for each

vertex $w \in N_{(s-1)/2}^\circ(v)$ bounds for each w the size of the $|N_{(s-1)/2}(w)|$ to at most $k - 1$. Thus, the size of $N_{(s-1)/2}(N_{(s-1)/2}^\circ(v))$ is at most $k^2 - 2k + 1$. By that also the size of $N_{s-1}^\circ(v)$ is bounded by $k^2 - 2k + 1$. Reduction Rule 1 applied for each vertex $x \in N_{s-1}^\circ(v)$ bounds the size of $|N_{(s-1)/2}(x)|$ by $k - 1$. Thus the size of $N_{(s-1)/2}(N_{s-1}^\circ(v))$ is at most $k^3 - 3k^2 + 3k - 1$.

Clearly, the size of $N_{(s-1)/2}[v] \cup N_{(s-1)/2}(N_{(s-1)/2}^\circ(v)) \cup N_{(s-1)/2}(N_{s-1}^\circ(v))$ is at most $k + k^2 - 2k + 1 + k^3 - 3k^2 + 3k - 1 \leq k^3$ and by Claim 4.2 we know that $|N_s[v]| \leq k^3$. By Lemma 4.4 we know that applying Reduction Rule 1 to the input graph G takes at most $O(n(n + m))$ time. This is sufficient, since reducing the input graph G , reduces also the s -neighborhood of each vertex in G . \square

Note that the Turing kernel size is completely independent of the parameter s . This leads directly to another result:

Theorem 4.4. *The s -CLUB problem is fixed-parameter tractable with respect to the parameter k .*

Proof. Using the kernelization results from Theorem 4.3 we can compute n independent k^3 -vertex kernels in $O((n + m) \cdot n)$ time, one for each vertex $v \in V$. Then we run on each of the n vertex sets of size $\leq k^3$ a brute-force algorithm, which needs $2^{O(k^3)}$ time to enumerate each possible subset of the at most k^3 vertices and $O(k^3)$ time to verify if this set is a solution via an all pairs shortest path algorithm, like the Floyd Warshall algorithm [CLRS01]. Over all, we need $O((n + m) \cdot n + n \cdot k^3 \cdot 2^{O(k^3)})$ time. \square

In the following we will show an interesting connection between the Turing kernel size and approximation algorithms for s -CLUB. The idea is to give an algorithm which computes both a Turing kernel and an approximation for s -CLUB. In the Turing kernelization given in this section, Reduction Rule 1 is applied to each vertex $v \in V$. The Reduction Rule 1 computes the set $N_{\lfloor s/2 \rfloor}[v]$ and checks whether the size of this set is at least k . By Lemma 4.4 the set $N_{\lfloor s/2 \rfloor}[v]$ is an s -club. Therefore we can use $N_{\lfloor s/2 \rfloor}[v]$ as an approximate solution for s -CLUB. Hence, the *combined s -club algorithm* proceeds as the Turing kernelization described in this section, but additionally computes the maximum set $N_{\lfloor s/2 \rfloor}[v]$ over all $v \in V$ and returns this set. Since the comparison of size of n set sizes takes $O(n)$ time and by the arguments of Lemma 4.4, the combined s -club algorithm needs $O(n(n + m))$ time as well.

In the following the performance, regarding the kernel size and the approximation factor, of the combined s -club algorithm is evaluated. To start with, two extremal

examples are given. In the worst case for the given Turing kernelization, at least one vertex $v \in V$ has closed $\lfloor \frac{s}{2} \rfloor$ -neighborhood of size $k - 1$. The combined s -club algorithm will identify the vertex v and return $N_{\lfloor s/2 \rfloor}[v]$. $N_{\lfloor s/2 \rfloor}[v]$ is a factor- $\frac{k-1}{k}$ approximation for s -CLUB, which is for large k close to an optimal solution. The other extremal example is the lower bound for a polynomial time approximation factor, which was shown by Marincek and Mohar as $k^{1/3-\epsilon}$ [MM02], for any $\epsilon \geq 0$. Since the combined s -club algorithm operates in polynomial time, this also upper-bounds the $\lfloor \frac{s}{2} \rfloor$ -neighborhood of each vertex in G to $k^{1/3}$. Using the kernelization idea given in this section, this results in a k -vertex Turing kernel, which gives a polynomial time recognizable optimal solution for s -CLUB, if exists. In the following theorem we generalize the kernel size and the approximation factor the combined s -club algorithm will return.

Theorem 4.5. *For even s (odd s), given an instance (G, k, s) for which the combined s -club algorithm computes:*

1. *A Turing kernel of size at least $(k - l)^2$ (at least $(k - l)^3$) vertices, with $l \leq k$, for s -CLUB, the combined s -club algorithm computes a polynomial time approximation of at least factor- $\frac{k-l}{k}$ for s -CLUB.*
2. *A factor- $\frac{k-l}{k}$, with $l \leq k$, polynomial time approximation for s -CLUB, the combined s -club algorithm computes a Turing kernel of size at most $(k - l)^2$ (at most $(k - l)^3$) vertices for s -CLUB.*

Proof. 1. In an instance, in which the combined s -club algorithm computes a Turing kernel of vertex size at least $(k - l)^2$ (at least $(k - l)^3$) for s -CLUB, there exists at least one vertex v with $|N_{\lfloor s/2 \rfloor}[v]| = k - l$. Since the combined s -club algorithm computes the set $N_{\lfloor s/2 \rfloor}[v]$ over each $v \in V(G)$ and returns the set with maximum size, the algorithm will return an approximation set, which is at least a factor- $\frac{k-l}{k}$ approximation for s -CLUB. Since the combined s -club algorithm has a running time of $O(n(n + m))$, this is a polynomial time approximation.

2. Since the combined s -club algorithm computes the set $N_{\lfloor s/2 \rfloor}[v]$ over each $v \in V(G)$ and returns the set with maximum size, the size of $|N_{\lfloor s/2 \rfloor}[v]|$, in an instance in which the combined s -club algorithm returns a factor- $\frac{k-l}{k}$ polynomial time approximation for s -CLUB, is upper-bounded to $k - l$ for every $v \in V$. Therefore and by the arguments given in Theorem 4.2 (Theorem 4.3) the vertex Turing kernel size computed by the combined s -club algorithm is upper-bounded by $(k - l)^2$ (by $(k - l)^3$).

□

In other words, by using the combined *s-club* algorithm either we achieve a Turing kernel of “small” vertex-size or a “good” approximation for *s-CLUB*.

4.2 A Branching Algorithm for *s-Club*

In this section we give a branching algorithm which solves *s-CLUB* on general graphs. The idea of this algorithm (Figure 4.2) is to initialize each vertex of the input graph as a start vertex set of size one, then we branch over all adjacent vertices which are possible enhancements of this set, until this sets have reached size k . Afterwards we check for each computed set whether it forms an *s-club*.

Lemma 4.5. *The algorithm Solve-s-club given in Figure 4.2 is correct.*

Proof. In the iteration starting in line 1 the function *Iterate-s-club*, described in Figure 4.3, is called for each vertex u of the graph G .

In function *Iterate-s-club* the size of the set S is checked. If the size of S is less than k , then we start a nested for-loop in line 2, which iterates over each vertex v in S and over each vertex w that is adjacent to v and not already in S . In each iteration step of the loop we build a new set, which consists of S together with the adjacent vertex w and call *Iterate-s-club* for this new set. Thereby each possible adjacent enhancement of the set S will be checked. It is sufficient to check only adjacent vertices, since $G[S]$ has to be one connected component in order to have a diameter of at most s . If the set S has size at least k , then we test in line 8 if S is an *s-club*. If S is an *s-club*, there will be an output of S in line 9. Hence, each possible connected vertex set which is of size k and contains u , will be checked whether it forms an *s-club*.

Since *Solve-s-club* called the function *Iterate-s-club* for each vertex u of the graph G , all *s-clubs* of size k in G will be provided as output. \square

In general the algorithm *Solve-s-club* enumerates all *s-clubs* of size k . However, in the following, we run the algorithm on a graph, which is reduced by Reduction Rule 1. Therefore the algorithm will only enumerate non-trivial *s-clubs* instances, but in order to solve *s-CLUB* it provides a more efficient running time.

Theorem 4.6. *The *s-CLUB* problem can be solved in $O(n(n + m) + n((k - 2)^k \cdot k! \cdot k^3))$ time.*

Algorithm: *Solve- s -club* (G)

Input: A graph G , an integer k

Output: All s -clubs of size k

```

1   for each  $u \in V(G)$ 
2       call Iterate- $s$ -club( $\{u\}, G, k$ )
3   endfor

```

Figure 4.2: Pseudo-code of the algorithm which, together with the algorithm in Figure 4.3, outputs all s -clubs of size k in G .

Algorithm: *Iterate- s -club* (S, G)

Input: A graph G , a set S , an integer k

Output: An s -club of size k

```

1   if  $|S| \leq k$  then
2       for each  $v \in S$ 
3           for each  $w \in \{N_1(v) \setminus S\}$ 
4               call Iterate- $s$ -club( $S \cup \{w\}, G, k$ )
5           endfor
6       endfor
7   else
8       if  $S$  is  $s$ -club then
9           output  $S$ 

```

Figure 4.3: Pseudo-code of the recursive function which is called by the algorithm in Figure 4.2.

Proof. By Lemma 4.5 we know the algorithm *Solve- s -club* given in Figure 4.2 is correct. We now prove the theorem by bounding the running time of algorithm *Solve- s -club* on an input graph, which is reduced by Reduction Rule 1. The reduction takes $O(n(n + m))$ time and upper-bounds the size of the open $\lfloor \frac{s}{2} \rfloor$ -neighborhood to $k - 2$, due to Lemma 4.4.

In the iteration starting in line 1 the function *Iterate- s -club* is called n times. In this function, a nested for loop is started in line 2, which iterates over each vertex v of the set S and over each vertex w that is adjacent to v and not already in S . By Reduction Rule 1 there are at most $k - 2$ possible vertices for w . Thereby each iteration makes for a vertex set S of size i at most $(k - 2) \cdot i$ steps in each calling a new instance of *Iterate- s -club* of size $i + 1$. In order to obtain the running time over all recursive calls, we have to multiply over all i and this results in a

running time of $(k - 2) \cdot 1 \cdot (k - 2) \cdot 2 \cdot \dots \cdot (k - 2) \cdot k \sim (k - 2)^k \cdot k!$. This is correct, since for a set of size $\geq k$ we abort the iteration and thereby do not enter recursive calls. Testing if S is an s -club takes $O(k^3)$ time, using an all pair shortest path algorithm, like the Floyd Warshall algorithm [CLRS01]. Over all we need $O(n(n + m))$ time for data reduction, then we make n calls to a function which calls itself up to $(k - 2)^k \cdot k!$ times, and additionally tests the at most $(k - 2)^k \cdot k!$ resulting sets in $O(k^3)$ time. Hence, the overall running time bound follows. \square

5 s -Club on Special Graph Classes

In this chapter, we analyze the complexity of the s -CLUB problem on several special graph classes. More specifically, polynomial time algorithms for trees are given in Section 5.1 and for interval graphs in Section 5.2. For bipartite graphs the complexity of the 2-CLUB problem will be given in Section 5.3. The complexity of s -CLUB will be discussed for distance-hereditary graphs in Section 5.4 and for planar graphs in Section 5.5.

5.1 Trees

In this section, we give an algorithm that solves s -CLUB on trees in $O(n \cdot s^2)$ time. In this algorithm we use a special property regarding the neighborhood of lowest leaves in rooted trees. This property is shown in the following claim:

Claim 5.1. *For a lowest leaf u in a rooted tree, the set $S_u = \{v \mid d(u, v) \leq s\}$ is the uniquely determined maximum s -club containing u .*

Proof. To prove the claim we have to show that $\forall a, b \in S_u : d(a, b) \leq s$. Without loss of generality let $d(u, a) \geq d(u, b)$. Consider P_a as the shortest path between u and a and P_b as the shortest path between u and b . Since u is a leaf there exists the path $P' = P_a \cap P_b$. Now let t be the vertex in P' with the shortest distance to a , as illustrated in Figure 5.1(a). Since u is a lowest leaf, it follows that $d(b, t) \leq d(u, t)$. Thus $d(b, t) + d(t, a) \leq d(u, t) + d(t, a) \leq s$. This means the distance between a and b is not higher than the distance between u and a and therefore at most s . Hence, the pairwise distance of any two vertices a, b in $S_u = \{v \mid d(u, v) \leq s\}$ is at most s . Therefore, S_u is an s -club. Since an s -club containing u cannot contain a vertex with distance greater than s , the set S_u is clearly the uniquely determined maximum s -club containing u . \square

In the following we show that if u is not a lowest leaf this idea will generally not work. For ease of presentation, we consider the same variables as in the proof of Claim 5.1. Now we give a simple example, which is illustrated in Figure 5.1(b).

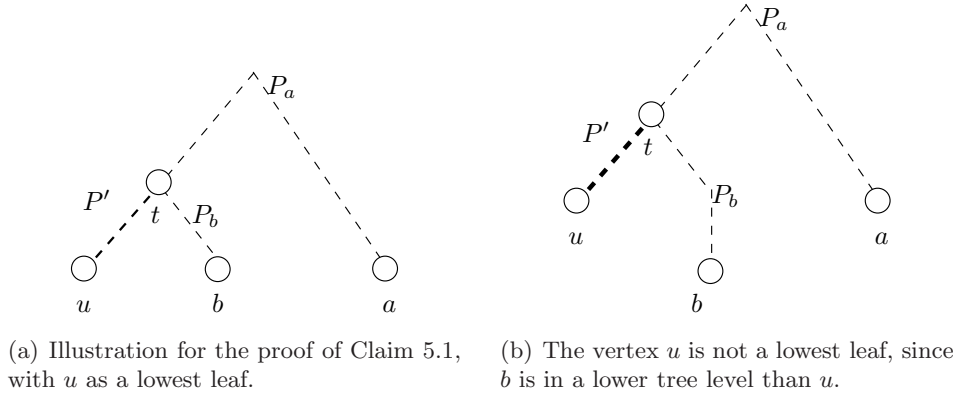


Figure 5.1: Illustration for Claim 5.1.

Assume, u is not a lowest leaf, and $d(u, t) = \frac{1}{4}s$, $d(t, a) = \frac{3}{4}s$, $d(t, b) = \frac{1}{2}s$ and therefore $a, b \in S_u = \{v \mid d(u, v) \leq s\}$. But the pairwise distance in S_u is not at most s : $d(a, b) = \frac{3}{4}s + \frac{1}{2}s \geq s$. Hence, S_u is not an s -club.

In Figure 5.2 we give the algorithm *Tree- s -club*. The algorithm computes, given a tree and a diameter as input, a maximum-cardinality s -club in this tree. The idea of this algorithm is to apply Claim 5.1 to each lowest leaf. After a lowest leaf is processed it is removed from the graph. By this, level-wise each vertex of the tree is processed until the root vertex r is reached. One may give a much simpler algorithm which computes the maximum s -club with a running time $O(n^2)$. In order to obtain an algorithm which does not compute n times the set S_u , we perform a precomputation resulting in the table $C[v, d]$. $C[v, d]$ stores for each vertex v in the tree, the number of vertices with distance exactly d in the induced subtree of v . See Figure 5.3(a) for example entries in the table $C[v, d]$. Then a path P from the lowest leaf towards the root is spanned. Then in order to obtain the size of a maximum s -club containing a current lowest leaf, the number of vertices in the sublevels, which have a sufficient distance, of the path vertices in P are added up. This path is illustrated in Figure 5.3(b), the triangles illustrate the vertices in the sublevels with sufficient distance. Thereby we obtain an algorithm which has an over all running time of $O(n \cdot s^2)$. This algorithm always computes a maximum s -club S of the tree; we may check afterwards whether the size of S is at least k and therefore a solution for s -CLUB.

Algorithm: *Tree-s-club* (T, s)

Input: A tree T , a diameter s .

Output: A maximum-cardinality s -club S .

```

1   Root the tree  $T$  at an arbitrarily root vertex  $r$ 
2    $T_{\text{copy}} := T$ 
3    $\text{sizeS}_{\text{sol}} := 0$ 
4   for each  $v \in V(T)$  //loop to initialize the table  $C$ 
5        $C[v, 0] := 1$  //each vertex is initialized for  $d = 0$  with value 1
6   endfor
7   for  $o := 1$  to  $l(r)$  //calculate the vertices in each sublevel for each vertex
8       for each  $v$  with  $l(v) = o$ 
9           for  $d := 1$  to  $\min(o, s)$  //d is depth in subtree
10               $C[v, d] := \sum_{\{u|u \text{ child}(v)\}} C[u, d - 1]$ 
11          endfor
12      endfor
13      repeat //process lowest leaf and delete
14          Take a vertex  $v_0$  from the lowest tree level (a leaf)
15           $q := \min\{l(r), s\}$ 
16          Get the shortest path  $P := (v_0, v_1, \dots, v_q)$  of length  $q$  from  $v_0$  towards  $r$ 
17          repeat //calculate the  $s$ -club by utilizing  $C$  for  $v_q \in P_v$ 
18              if  $l(v_q) > \lfloor \frac{s}{2} \rfloor$  then  $\text{sizeS} := \text{sizeS} + \sum_{i=0}^{s-l(v_q)} C[v_q, i] - 1$ 
19              else  $\text{sizeS} := \text{sizeS} + \sum_{i=0}^{l(v_q)} C[v_q, i]$ 
20               $q := l(v_q) - (s - l(v_q))$  //index update
21          until  $q \leq \lfloor \frac{s}{2} \rfloor$  //until all subtree levels have distance at most  $s$ 
22          if  $\text{sizeS} \geq \text{sizeS}_{\text{sol}}$ 
23               $\text{sizeS}_{\text{sol}} := \text{sizeS}$ 
24               $u_{\text{sol}} := v_0$  //save for later reconstruction of maximum
25          endif
26           $V(T) := V(T) \setminus \{u\}$  //delete processed vertex
27          for  $i := 1$  to  $\min\{l(r), s\}$ 
28               $C[v_i, i] := C[v_i, i] - 1$  //table update for deleted vertex
29          endfor
30      until  $V(T) = \emptyset$  //until tree is empty
31       $S := \{v \mid d_{T_{\text{copy}}}(u_{\text{sol}}, v) \leq s \wedge l_{T_{\text{copy}}}(u_{\text{sol}}) \leq l_{T_{\text{copy}}}(v)\}$  // get solution
32      return  $S$ 

```

Figure 5.2: The Pseudo-code of the algorithm which computes an s -club on a tree. In Figure 5.3(a) the table entries $C[v, d]$ and the vertices in P are illustrated.

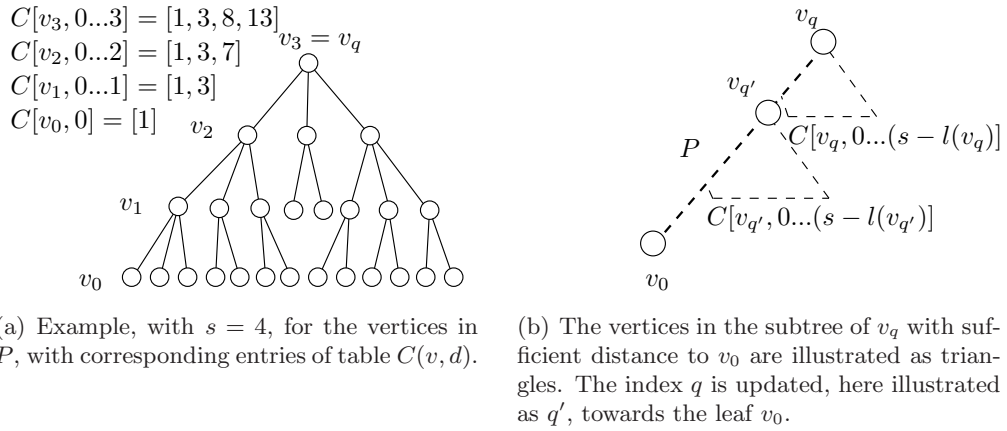


Figure 5.3: Illustration for the algorithm *Tree- s -club* from Figure 5.2. The term $C[v, 0\dots d]$ is the number of vertices in the induced subtree of v in the distances 0 to d .

Lemma 5.1. *The algorithm *Tree- s -club* is correct.*

Proof. The algorithm *Tree- s -club* is given in Figure 5.2. In line 3 the size of the current maximum s -club size S_{sol} is set to zero. In the loop starting in line 4 the entry of the table $C[v, 0]$ is initialized with 1. This is correct since in distance 0 of any vertex there exists the vertex itself. The iterations starting in the lines 7 and 8 iterate over each vertex v of the tree in a bottom-up way. In lines 9 and 10 the number of vertices in distance exactly d in the induced subtree of v are calculated and stored in table $C[v, d]$. This is done in a bottom up way, by summing up over the entries in $C[u, d - 1]$, whereby u are the children of v . This is correct since $d(v, u) = 1$ and table C is initialized correctly for leaves in line 4. This finishes the precomputation.

In line 13 we start iteratively checking the size of the maximum s -club containing a lowest leaf v_0 . By Claim 5.1 this s -club is determined as $S_{v_0} = \{z \mid d(v_0, z) \leq s\}$. In line 15 the value q is evaluated as the minimum from s and the level of the root r . In line 16 the path P is initialized as the path of length q from the lowest leaf v_0 towards the root r . If $v_q = r$, this is sufficient for our calculation since we know from table C how many vertices are in the sublevels in distance $0, \dots, (s - d(v_0, r))$. If $q = s$ this is correct, since vertices which are farther away than s , are too far away from v_0 to build an s -club with v_0 . The level q is also used as index in the loop starting in line 17. This loop calculates the value of $sizeS$, which is the size of the maximum s -club containing the leaf v_0 . In order to do so, decreasing from level q , each subtree level is added up, which has distance $s - q$ and therefore

distance at most s from v_0 . We subtract ones per iteration the value 1 to prevent a double counting of the new v_q in the next iteration. The idea of the summation is illustrated in Figure 5.3(b), the triangles illustrate the vertices in the sublevels with sufficient distance. This summation is done until a path vertex with level at most $\lfloor \frac{s}{2} \rfloor$ is reached, from this vertex one can add up all levels in the induced subtree. In line 22 the value sizeS is tested if it is greater than $\text{sizeS}_{\text{sol}}$, then a larger s -club is found, and the optimal leaf is stored in u_{sol} and its size in $\text{sizeS}_{\text{sol}}$. In line 26 the processed leaf v_0 is removed from the tree. Since the input tree T is deleted, we look in line 31 in a copy of the input tree for the solution set C . Since every set S_{v_0} is a maximum s -club and in the iteration every vertex will be a lowest leaf, and thereby checked, the algorithm is correct. \square

Theorem 5.1. *s -CLUB on trees can be solved in $O(n \cdot s^2)$ time.*

Proof. By Lemma 5.1 we know that algorithm *Tree-s-club* given in Figure 5.2 is correct. Hence, we prove the theorem by upper-bounding the running time of algorithm *Tree-s-club* to $O(n \cdot s^2)$. Making a copy of the input tree in line 2 takes $O(n)$ time. The for-loop beginning in line 4 takes $O(n)$ time. The iterations starting in lines 7 and 8 iterate over each tree level and every vertex in these levels. Therefore, the iterations process each vertex in each level exactly once, this takes $O(n)$ running-time. The loop starting in line 9 makes at most s iterations. The operation in line 10 makes in the whole algorithm at most as much summations as there are edges in the tree. Therefore, the operation takes constant time. Hence, the loop starting in line 7 makes at most $O(n \cdot s)$ iterations in which each operation takes at most $O(1)$ time. Therefore, the loop starting in line 7 takes at most $O(n \cdot s)$ time. The iteration beginning in line 13 makes at most as much iterations as there are vertices in the graph, these are at most n iterations. Getting the path P in line 16 of length at most s takes at most $O(s)$ time. In line 17 we start a repeat loop with at most s iterations. Each of the operations in lines 18 and 19 perform at most s additions, this takes $O(s)$ time as well. Therefore, the repeat loop started in line 17 takes at most $O(s^2)$ running time. In line 27 we start a for-loop with $O(s)$ iterations overall. Since we have seen that each operation in the iteration, which started in line 13, needs at most $O(s^2)$ time, we need $O(n \cdot s^2)$ time for the whole iteration. The calculation of the solution set in line 31 takes $O(n)$ time. Since every iteration in the algorithm takes at most $O(n \cdot s^2)$ time, the whole algorithm *Tree-s-club* takes $O(n \cdot s^2)$ time as well. \square

By using a table with a consumption of $O(n \cdot s)$ space, we gave an algorithm that solves s -CLUB on trees needing $O(n \cdot s^2)$ time. One may generalize this dynamic programming idea for solving s -CLUB on tree decompositions.

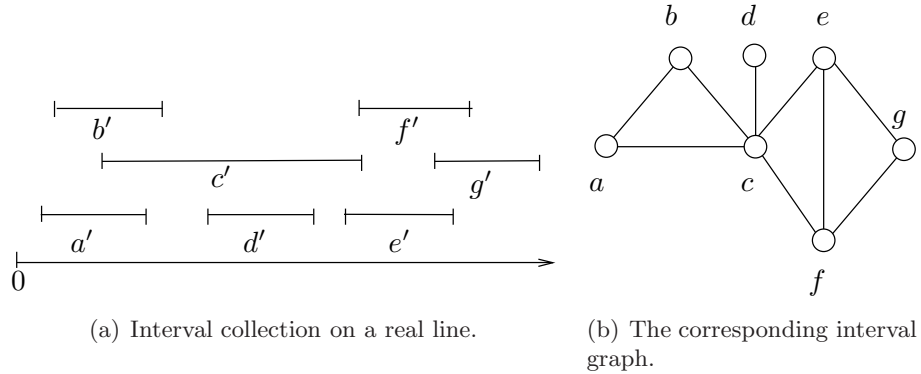


Figure 5.4: An example interval and the corresponding interval graph.

5.2 Interval Graphs

Interval graphs are a popular graph class in algorithmic graph theory. The applications are numerous. For example, problems in scheduling and storage translate to finding minimum colorings and clique covers in respective interval graphs [Gol80]. Since the CLIQUE problem is solvable in polynomial time on interval graphs [Gol80], the question arises whether or not the same holds for the s -CLUB problem. In this section, we give an algorithm that finds a maximum s -club for an interval graph in $O(n^2)$ time. To start with, interval graphs are defined.

Let $I = \{i'_1, i'_2, \dots, i'_n\}$ be a finite collection of intervals on the real line. The corresponding *interval graph* is $G = (V, E)$ where $V = \{i_1, i_2, \dots, i_n\}$ and $(i_x, i_y) \in E \Leftrightarrow i'_x \cap i'_y \neq \emptyset$. An example interval collection and the corresponding graph are illustrated in Figure 5.4. In the following, the left startpoint and right endpoint of an interval v' will be denoted by $l_{v'}$ and $r_{v'}$, where $l_{v'} \leq r_{v'}$.

In the following a vertex $u \in V$ will be called *s -simplicial* if the closed s -neighborhood $N_s[u] = \{v \in V \mid d(u, v) \leq s\}$ of u is an s -club. The idea of algorithms [Gol80] which solve CLIQUE on interval graphs is to use that the vertex corresponding to the interval with the maximum left startpoint is 1-simplicial. We generalize this idea for an algorithm which solves s -CLUB on interval graphs. In order to do so, we show that u is s -simplicial:

Lemma 5.2. *In an interval graph the vertex corresponding to the interval with the maximum left startpoint is s -simplicial, with $s \geq 1$.*

Algorithm: *Interval-s-club*

Input: A graph G , the corresponding interval collection I , a diameter s

Output: A maximum s -club S_{sol}

```

1    $S_{sol} := \emptyset$ 
2   repeat
3      $U := \{u \mid l_{u'} = \max\{l_{i'} \mid i = 1, \dots, n\}\}$ 
4     Choose arbitrary an  $u \in U$ 
5      $S_u := \{v \mid d(u, v) \leq s\}$ 
6     if  $|S_u| \geq |S_{sol}|$  then  $S_{sol} := S_u$ 
7      $V(G) := V(G) \setminus \{u\}$ ,  $I := I \setminus \{u'\}$ 
8   until  $V(G) = \emptyset$ 
9   return  $S_{sol}$ 

```

Figure 5.5: The Pseudo-code of the algorithm which computes a maximum s -club on an interval graph.

Proof. Let G be an interval graph with n vertices, and consider the corresponding interval representation I . Let u be a vertex of G whose corresponding interval u' has the maximum left startpoint $l_{u'}$ in I . It will be now shown that u is s -simplicial. Consider two vertices a and b such that $d(u, a) \leq s$ and $d(u, b) \leq s$. Without loss of generality, let $l_{a'} < l_{b'} < l_{u'}$. Consider a shortest path P between u and a , there must be a vertex $z \in P$ such that $l_{z'} \leq l_{b'} < r_{z'}$. Hence, $d(a, b) \leq d(u, z) + 1 \leq d(u, a) \leq s$. Therefore, $d(a, b) \leq s$ and vertex u is s -simplicial. \square

In Figure 5.5 the algorithm *Interval-s-club* is given, which uses Lemma 5.2 to compute an maximum s -club on interval graphs. The idea of the algorithm is that a vertex u , corresponding to an interval with the maximum left startpoint u' , is s -simplicial and therefore the vertex set $S_u = \{v \mid d(v, u) \leq s\}$ is the maximum s -club containing this vertex. If this vertex set S_u is of current maximum size, this set will be saved. After the size is checked, vertex u is deleted from the graph and interval u' is deleted from the collection of intervals. If we apply this idea repeatedly, every interval will be the interval with the maximum left startpoint at some point of the algorithm.

Lemma 5.3. *The algorithm *Interval-s-club* given in Figure 5.5 is correct.*

Proof. In line 1 the current maximal solution set S_{sol} is initialized with the empty set. In line 3 and line 4 a vertex u corresponding to an interval with the maximum left start is calculated. From Lemma 5.2 we know that u is s -simplicial. Hence,

$S_u := \{v \mid d(u, v) \leq s\}$ is the maximum s -club containing this vertex u . In line 6 the size of set S_u is checked. If the set S_u has size at least $|S_{sol}|$, the algorithm found a new maximal solution, and the algorithm saves S_u in line 6 as the current maximal solution S_{sol} . Since the size of the maximum s -club containing u is checked, it is correct to remove vertex u from the interval collection and the graph in line 7. This algorithm repeats until the graph has no vertices, therefore each vertex of the graph will, at some point of this algorithm, correspond to the interval with the maximum left startpoint. Thus each vertex is checked and the algorithm is correct. \square

Theorem 5.2. *Computing the maximum s -club on an interval graph can be solved in $O(n^2)$ time.*

Proof. From Lemma 5.3 we know that the algorithm *Interval- s -club* given in Figure 5.5 is correct. Hence, we prove the theorem by bounding the running time of algorithm *Interval- s -club* by $O(n^2)$. In line 2 we start an iteration over all the vertices of the graph. Finding the vertex u corresponding to the interval with the maximum left startpoint, which is calculated in line 3, takes at most $O(n)$ time. Calculating the maximum s -club containing u in line 5 takes at most $O(n)$ time. Lines 6 and 7 take at most $O(n)$ time. Since each step of the iteration takes at most $O(n)$ time and the iteration runs over all the vertices of the graph, the iteration takes at most $O(n^2)$ time. Since each iteration of the algorithm takes at most $O(n^2)$ time, the algorithm needs at most $O(n^2)$ running time. Hence, Theorem 5.2 is correct. \square

The worst-case running time of this algorithm can be reduced: Starting by a precomputation which sorts the intervals I . This sorting takes $O(n \log n)$ running time. By this the running time of line 3 is reduced to $O(1)$. The running time of line 5 can be reduced to $O(s^2)$, by a precomputation of a table similar to the table of the algorithm *Tree- s -club* given in Section 5.1. The running times of the lines 6 and 7 are depending on the implementation and the datastructure. The running time of line 6 can be reduced to $O(1)$ by using pointers. The running time of line 7 can be reduced to $O(1)$ by using an adjacency matrix instead of an adjacency list. Therefore, the running time of an algorithm which solves s -club on an interval graph can be reduced to $O(n \log(n) + n \cdot s^2)$ time.

5.3 2-Club on Bipartite Graphs

A *bipartite graph* is a graph $G(V, E)$ in which the vertex set V can be partitioned into two disjoint sets $A \subseteq V$ and $B \subseteq V$ such that no two graph vertices within the same set are adjacent. In other words, bipartite graphs one to one correspond 2-colorable graphs. Bipartite graphs have applications in coding theory [LMSS01], projective geometry [TN91] and are used to model matching problems [Eke95].

To solve 2-CLUB on bipartite graphs we show that a *biclique* is equivalent to the graph induced by a 2-club. A *complete bipartite graph* or *biclique* is a bipartite graph $G((S_A \cup S_B), E)$ such that every vertex of the first set S_A is adjacent to every vertex of the second set S_B .

Lemma 5.4. *Let $G = (V, E)$ be a bipartite graph and $S \subseteq V$. Then, $G[S]$ is biclique if and only if S is a 2-club.*

Proof. “ \Rightarrow ”: Since $G[S_A \cup S_B]$ is a biclique, every vertex in one of the sets S_A or S_B , without loss of generality S_A , has distance one to every vertex in S_B . Since every vertex in S_B is also adjacent with every vertex in S_A the distance between a vertex in S_A to another vertex in S_A is two. Thus every vertex in S has distance at most two to another vertex in S and therefore S is a 2-club.

“ \Leftarrow ”: Let S be a 2-club. Then every vertex in $G[S]$ has, by definition, distance at most two to every other vertex in S . Since S is a vertex set in a bipartite graph it can be partitioned into two subsets S_A and S_B where no vertex in S_A is adjacent to any other vertex in S_A , and analogously for S_B . Therefore, every vertex $a \in S_A$ has to be adjacent to every vertex $b \in S_B$, because otherwise, since all vertices in S_a are non adjacent, the shortest path from a to b would have length at least three. Hence, every vertex b is directly connected with every vertex a . Hence, S is a biclique. \square

Theorem 5.3. *The 2-CLUB problem on bipartite graphs can be solved in $O(n^5)$ time.*

Proof. By Lemma 5.4, we know that S is a 2-club if and only if $G[S]$ is biclique. The problem of finding a maximum cardinality biclique can be solved via reduction to the problem of finding a min-cut [DdFS07] in the complement graph $\overline{G} = (V, \overline{E})$ with $\overline{E} = \{(u, v) \in \overline{E} \mid (u, v) \notin E\}$. A minimum cut algorithm like the Edmonds-Karp algorithm runs in $O(n\overline{m}^2)$ time [EK72, Din70]. The number of edges in the complement graph \overline{m} can be upper-bounded by n^2 . Therefore, the problem of finding a maximum cardinality biclique can be solved in $O(n^5)$ time. \square

In this section we have shown that the 2-CLUB problem on bipartite graphs can be solved in $O(n^5)$ time. The complexity of s -CLUB on bipartite graphs, with $s \geq 3$, remains open for further investigations.

5.4 Graph Classes with bounded Clique- or Treewidth

In this section we will classify the complexity of the s -CLUB problem on graphs with bounded tree- or cliquewidth. Examples for graphclasses with bounded tree- [Bod98] or cliquewidth [GR99, GHN06] are: Trees and Forests (treewidth 1), series-parallel graphs (treewidth 2), outerplanar graphs (treewidth 2), Halin graphs (treewidth 3), cographs (cliquewidth 2), complete graphs (cliquewidth 2) and distance-hereditary graphs (cliquewidth 3).

The graphs with bounded treewidth can be recognized in linear time [Bod96], with a running time exponential in the bounded treewidth. Graphs with cliquewidth at most ≤ 3 are polynomial time recognizable [CHL⁺00]. Whether it is possible to recognize graphs of any constant cliquewidth in polynomial time is open [FRRS06]. The clique or tree decomposition of a graph can be computed by polynomial time approximation algorithms [BKMT04, Oum08]. For a general introduction to the concepts of treewidth and cliquewidth see the survey by Seese et al. [SHOG07].

It is known that on a graph with bounded clique- or treewidth, every graph problem that is expressible in Monadic Second Order Logic can be solved in polynomial time, given the respective decomposition [CO00, CMR01]. Since Equation 3.1 shows that the s -CLUB problem is expressible in Monadic Second Order Logic the following proposition is correct:

Proposition 5.1. *On a graph with bounded tree- or cliquewidth the s -CLUB problem is solvable in polynomial time.*

This is a classification result, since the running time is exponential in the respective width. The fact that the width is bounded by a constant secures Proposition 5.1. We gave no explicit algorithm to solve s -CLUB on those decompositions, but one may generalize the dynamic programming technique from Section 5.1 to solve s -CLUB on tree decompositions.

5.5 Planar Graphs

A graph is *planar* if it can be drawn in a plane without crossing graph edges. Planar graphs arise in many applications, such as road networks or printed circuit boards, which are naturally planar because they are defined by surface structures. Therefore, planning routes on roads without underpasses [SS07] or laying out circuits on computer chips [HSM82] are planar graph problems.

In a planar graph there is no induced K_5 . This arises from the fact that a K_5 can not be drawn without intersecting edges. This fact can be used to show that the number of maximal cliques in planar graphs is at most $p(n) = (7n/3 - 6)$ [Pri95]. This bound helps to give an efficient polynomial time algorithm for the CLIQUE problem on planar graphs. One may first apply the algorithm from Tsukiyama et al. [TIAS77] to find all maximal cliques in time $O(nmp(n))$. Then, to find the maximum clique we iterate over the list of all maximal cliques, selecting the largest one. This algorithm is bounded by the number of cliques in a planar graph $p(n)$.

Even though the fact that we cannot draw a K_5 seems to be of no help regarding the s -CLUB problem, we can use properties of planar graphs to classify the complexity of s -CLUB on planar graphs. Such a property is given by the following theorem:

Theorem 5.4. [Bak94, FG06] *A planar graph with n vertices and of diameter D has a tree decomposition of width at most $3D$ that can be found in time $O(D \cdot n)$.*

For an introduction to treewidth see the publications by Bodlaender [BK08], Flum and Grohe [FG06] or Niedermeier [Nie06]. However, in the following the idea is to compute n s -neighborhoods, one for each vertex of the input graph. This idea is based on Observation 4.1. Recall Observation 4.1, this is that, if a vertex v is part of a solution S for the s -CLUB problem, then only vertices in the s -neighborhood of v can also be part of S . Each of the n s -neighborhoods have clearly diameter at most $2s$. Hence, by Theorem 5.4 one can find n tree decompositions, one for each s -neighborhood, each of width at most $6s$ in $O(2s \cdot n^2)$ time. In the following we will utilize these tree decompositions to solve s -CLUB on planar graphs.

Courcelle's famous theorem [Cou90, Cou91, CM93] states that if a problem is expressible in MSO Logic, then it can be solved in time only exponential in the treewidth on graphs with a given tree decomposition. Equation 3.1 shows that s -CLUB is expressible in Monadic Second Order (MSO) Logic. Since the width of a tree decomposition of an s -neighborhood is bounded by $6s$, the s -CLUB problem on such a decomposition is fixed-parameter tractable with respect to parameter

s . Hence, by solving the s -CLUB problem on each of the n tree decompositions and by Observation 4.1 the following statement clearly holds:

Corollary 5.1. *On a planar graph the s -CLUB problem is fixed-parameter tractable with respect to parameter s .*

If we consider s as a constant, the result changes to:

Corollary 5.2. *On a planar graph for constant s , the s -CLUB problem is decidable in polynomial time.*

This is a classification result. The running time is still exponential in the treewidth. Only the fact that the treewidth is bounded, here by the constant $6s$, secures Corollary 5.2.

6 Vertex Deletion Problems

In this chapter, we introduce the problems s -CLUB VERTEX DELETION in Section 6.1 and s -CLUB CLUSTER VERTEX DELETION in Section 6.2.

6.1 Obtaining an s -Club

Fixed-parameter algorithms are efficient for small parameters. In Section 3.1.5 we mentioned the idea of a different parameterization for the s -CLUB problem. The motivation of such a parameterization is the following. If the size of a maximum s -club is almost n , then the proposed fixed-parameter algorithms are inefficient. In this case, however, the number of vertices that have to be deleted is relatively small. Hence, in this section we introduce the s -CLUB VERTEX DELETION problem, in which one wants to identify the set of vertices which are not in the s -club.

We define the s -CLUB VERTEX DELETION problem as follows:

s -CLUB VERTEX DELETION

Input: An undirected graph $G = (V, E)$ and integers $s, k \geq 2$.

Question: Is there a subset of vertices $C \subseteq V$ of size at most k such that $G[V - C]$ has diameter at most s ?

In classical complexity theory the complexities of s -CLUB and s -CLUB VERTEX DELETION are equivalent. Hence, s -CLUB VERTEX DELETION is NP-complete as well. We show fixed-parameter tractability in Section 6.1.1. To our knowledge s -CLUB VERTEX DELETION has not been studied yet.

The s -CLUB VERTEX DELETION problem is related to the VERTEX COVER problem, which is defined as follows:

VERTEX COVER

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Question: Is there vertex set $C \subseteq V$ of size at most k , such that each edge of G is incident with at least one element of C ?

While VERTEX COVER can be utilized to identify a maximum clique, s -CLUB VERTEX DELETION can be used to identify a maximum s -club. In detail, let $S \subseteq V$ be a clique in G and $\overline{G} = (V, \overline{E})$ with $\overline{E} = \{(u, v) \in \overline{E} \mid (u, v) \notin E\}$, then $V \setminus S$ is a solution of VERTEX COVER in \overline{G} . And let $S \subseteq V$ be an s -club in G , then $V \setminus S$ is a solution for s -CLUB VERTEX DELETION in G . In contrast to clique, s -club is a non-hereditary graph property. Therefore, standard techniques like characterization by forbidden subgraphs are not directly applicable.

6.1.1 s -Club Vertex Deletion is Fixed-Parameter Tractable

To show the fixed-parameter tractability of s -CLUB VERTEX DELETION we describe a search tree algorithm with a search tree size upper-bounded by 2^k .

Theorem 6.1. *The s -CLUB VERTEX DELETION problem is fixed-parameter tractable with respect to parameter k , that is, it can be solved in $O(2^k \cdot n(n+m))$ time.*

Proof. The search tree strategy proceeds as follows. Search for a pair of vertices (u, v) with $d(u, v) \geq s + 1$. If no such pair exists then the graph is already an s -club and no further vertices need to be deleted. The search for these pairs takes $O(n(n+m))$ running time using breadth-first search for each of the n vertices in the graph. Branch into the two subcases of deleting either u or v and set $k := k - 1$. This is correct, since u and v have too high distance, consequently one of them have to be deleted. The number of cases in each search tree node is bounded by two and the algorithm terminates if either $k = 0$ or a valid solution set of size $\leq k$ has been found. Hence, the size of the search tree is bounded to 2^k . \square

The algorithm given in proof of Theorem 6.1 is rather simple. Improved branching rules are a standard technique to reduce the running time bound of search tree algorithms. In the following, we give improved branching rules that will work until we remain with certain graph instances. Then we will talk about difficulties to give an improved branching strategy for such graph instances. This discussion is useful for a better understanding of the s -CLUB VERTEX DELETION problem. For ease of presentation, we call two vertices (u, v) with $d(u, v) = s + 1$ a *conflict pair*.

For s -CLUB VERTEX DELETION, one can improve the running time if one can process in each case distinction more than just two vertices. This is the case if for a vertex u the set of vertices $\{v \mid d(u, v) \geq s + 1\}$ contains more than one vertex.

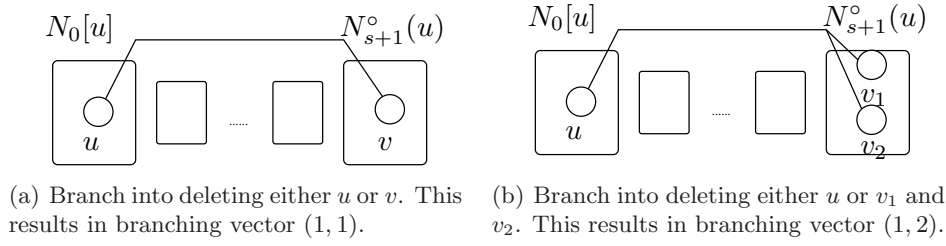


Figure 6.1: Different sizes of $|V \setminus N_s[u]|$ result in different branching situations.

This allows a branching between u and the set $\{v \mid d(u, v) \geq s + 1\}$. An example is illustrated in Figure 6.1: If there are two vertices v_1 and v_2 in the exact $s + 1$ neighborhood of u , then in order to obtain a graph of diameter s , we branch into two cases of either deleting u or v_1 and v_2 , resulting in branching vector $(1, 2)$. If this branching can always be performed we obtain a search tree size of $O((1.62)^k)$. If this branching is no longer applicable, we remain with graph instances in which $\forall v \in V : |V \setminus N_s[v]| \leq 1$. More informally, these are graphs with diameter $s + 1$ in which the size of the exact $(s + 1)$ -neighborhood of each vertex is exactly one.

In the following we explain the difficulties for an improved branching on these graph instances, by the example graph given in Figure 6.2. This graph has diameter $s + 1$, with $s = 2$, and each vertex in the graph has an exact $(s + 1)$ -neighborhood of size at most one. In particular in this graph we have the conflict pairs (a, a') , (b, b') , (c, c') and (u, u') , which are marked as grey vertices. These vertices have an exact $(s + 1)$ -neighborhood of size one. The white vertices have, at least for now, an exact $(s + 1)$ -neighborhood of size zero. In the following it is described why it is not easy to get a better branching vector than $(1, 1)$ for these instances. The major difficulty are dependencies which arise by the fact that the s -club property is non-hereditary. Look for example at conflict pair (a, a') in Figure 6.2. If we start by deleting a' , the conflict (b, b') enhances to (b, b') and (b', c') . If we instead delete a , then (c, c') enhances to (c, c') and (c, d) . However, d was not part of any conflict pair before. This means that a vertex, like a , could be an element of the only shortest path connecting a pair of other vertices, like (c, d) . Since one has to consider all shortest paths which could become important through future deletions, it seems hard to find an improved branching. This also implies that it is not easy to identify vertices on which one never has to branch.

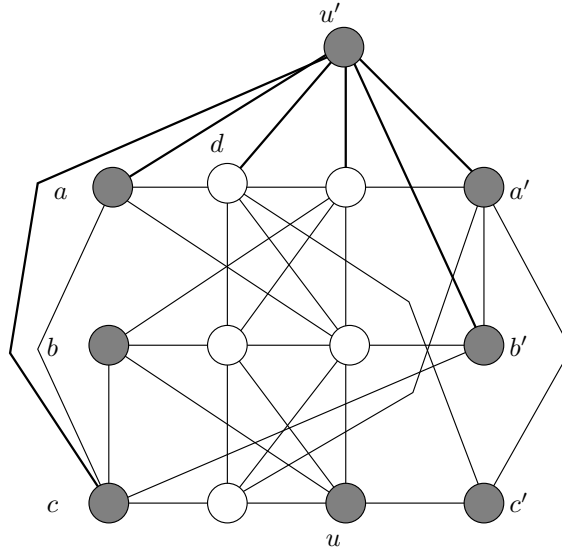


Figure 6.2: A graph with diameter 3, for $s = 2$ there exist the conflict pairs (a, a') , (b, b') , (c, c') and (d, d') , which are marked as grey vertices.

6.1.2 Data Reduction

In this section, we will give some data reduction rules for s -CLUB VERTEX DELETION. Although we will not show a problem kernel, these rules could be helpful in practical implementations. We start with a reformulation of Reduction Rule 1 for s -CLUB from Section 4.1.2:

Reduction Rule 2. *If there exists a vertex $v \in V$ with $|N_{\lfloor s/2 \rfloor}[v]| \geq n - k$, then delete $V \setminus N_{\lfloor s/2 \rfloor}[v]$ and answer yes.*

Lemma 6.1. *Reduction Rule 2 is correct and takes $O(n(n + m))$ time.*

Proof. To show that Reduction Rule 2 is correct, we show that $G[N_{\lfloor s/2 \rfloor}[v]]$ is an s -club, that is, the pairwise distance of any two vertices $x, y \in N_{\lfloor s/2 \rfloor}[v]$ in $G[N_{\lfloor s/2 \rfloor}[v]]$ is at most s . Since $x, y \in N_{\lfloor s/2 \rfloor}[v]$ and by the definition of $N_{\lfloor s/2 \rfloor}[v]$, the distance from x to v is at most $\lfloor \frac{s}{2} \rfloor$ and the distance from v to y is at most $\lfloor \frac{s}{2} \rfloor$. Consequently, the distance between x and y is at most s . Thus, Reduction Rule 2 is correct. Computing the set $N_{\lfloor s/2 \rfloor}[v]$ can be done in $O(n + m)$ time for one vertex, by breadth-first search. Checking whether Reduction Rule 2 can be applied to any vertex takes $O(n(n + m))$ total time. \square

Reduction Rule 2 finds easy yes-instances. In a reduced instance (G', k', s') with $G' = (V', E')$ the following holds: $\forall v \in V' : |V' \setminus N_{\lfloor s/2 \rfloor}[v]| \geq k'$.

Reduction Rule 3. *If there exists a vertex $v \in V$ that has distance at least $s + 1$ to more than k vertices, then delete v and set $k := k - 1$.*

Lemma 6.2. *Reduction Rule 3 is correct and takes $O(n(n + m))$ time.*

Proof. Assume that we do not remove the vertex v . Then, we have to remove all vertices in distance $\geq s + 1$, otherwise we remain with a graph with diameter at least $s + 1$. Thereby we have to delete more than k vertices. By the definition of s -CLUB VERTEX DELETION we cannot make more than k deletions. Hence, this is a contradiction and we have to delete v . Thus Reduction Rule 3 is correct. Computing the set $|V \setminus N_{s+1}[v]|$ can be done in $O(n + m)$ time for one vertex, by breadth-first search. Applying Reduction Rule 3 for each vertex takes $O(n(n + m))$ total time. \square

This rule finds easy no-instances. In a reduced instance (G', k', s') with $G' = (V', E')$ the following holds: $\forall v \in V' : |V' \setminus N_{s'+1}[v]| \leq k'$ and $\text{diam}(G') \leq s' + 1 + k'$.

A difficulty in finding a problem kernel for s -CLUB VERTEX DELETION is the following. In order to upper-bound the vertex size of a reduced instance (G', k', s') by a function of k' , one also has to upper-bound the number of vertices in the s -club, one wishes to obtain. The size of this s -club depends on $n - k$, thus one has to identify vertices in the s -club, which will never be part of a solution for s -CLUB VERTEX DELETION. But because of future deletions, and the unforeseen conflicts which could arise, it is difficult to decide which vertices this will be.

6.2 Obtaining an s -Club Cluster Graph

Data clustering is the assignment of a set of observations into subsets, called clusters, so that observations in the same cluster are similar. Data clustering is a central task in many fields like data mining, pattern recognition, machine learning, and bioinformatics. In the following we see similarity as a distance of at most s between the observations, however distance in the specific application is defined. In the following let an s -club cluster graph be a graph, in which every connected component got diameter s . Our task will be to identify the set of outliers such that the graph without this set forms an s -club cluster graph.

Hence, we define the s -CLUB CLUSTER VERTEX DELETION problem as follows:

s-CLUB CLUSTER VERTEX DELETION

Input: An undirected graph $G = (V, E)$ and nonnegative integers $s, k \geq 2$.

Question: Is there a subset of vertices $C \subseteq V$ of size at most k such that $G[V - C]$ is an s -club cluster graph?

This problem is a relaxation of CLUSTER VERTEX DELETION which has been studied by Hüffner et al. [HKMN09]. Roughly speaking, CLUSTER VERTEX DELETION is the problem of clustering into cliques, while *s*-CLUB CLUSTER VERTEX DELETION is the problem of clustering into s -clubs. In contrast to clique, s -club is a non hereditary graph property. Therefore standard techniques like characterization by forbidden subgraphs or iterative compression [FGMN09] are not directly applicable. To our knowledge, the *s*-CLUB CLUSTER VERTEX DELETION problem has not been studied yet. We show NP-completeness in Section 6.2.1 and fixed-parameter tractability in Section 6.2.2.

The *s*-CLUB CLUSTER VERTEX DELETION problem is related to the GRAPH *s*-CLUSTERING problem:

GRAPH *s*-CLUSTERING

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

Question: Is there a partition of V into disjoint sets C_1, \dots, C_l such that the diameter of $G[C_i]$ is at most s .

This problem asks for a partition of V such that each part is an s -club, while *s*-CLUB CLUSTER VERTEX DELETION asks for a subset C of V to delete such that each component is an s -club. Deogun et al. [DKS97] showed that there is an $\epsilon \geq 0$ such that no polynomial time algorithm approximates k -CLUSTERING better than n^ϵ , unless $P=NP$. Farley et al. [FHP81] gave a linear-time algorithm for trees. Abbas and Stewart [AS99] gave a linear-time algorithm for interval graphs, bipartite permutation graphs, and showed NP-completeness on chordal and interval graphs.

6.2.1 *s*-Club Cluster Vertex Deletion is NP-complete

Theorem 6.2. *s*-CLUB CLUSTER VERTEX DELETION is NP-complete, even if the input graph G is planar.

Proof. The *s*-CLUB CLUSTER VERTEX DELETION problem is in NP. It is easy to construct a nondeterministic algorithm that guesses a subset of vertices $S \subseteq V$

and checks in polynomial time whether the connected components of $G[V \setminus S]$ have diameter at most s or not.

It remains to show that s -CLUB CLUSTER VERTEX DELETION is NP-hard. We show this by transforming any instance of VERTEX COVER into an instance of s -CLUB CLUSTER VERTEX DELETION. A *vertex cover* is defined as a subset of vertices $C \subseteq V$ such that each edge of G is incident with at least one element of C . Following this definition the VERTEX COVER problem is defined as:

VERTEX COVER

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k .

Question: Is there vertex cover C of size at most k ?

VERTEX COVER is known to be NP-complete even for planar graphs [GJ77].

Let $G = (V, E)$ be the input graph of a planar VERTEX COVER instance (G, k) . We construct an s -CLUB CLUSTER VERTEX DELETION instance (G', k', s) as follows. Initially, we set $G' = G$. Every edge and every vertex in G' that is taken from G are in the following called *natural*. Every further element added to G' is called *artificial*. For each natural vertex u of G , we add s artificial vertices u'_1, u'_2, \dots, u'_s , together with the artificial path $P'_u = (u, u'_1, u'_2, \dots, u'_s)$, as illustrated in Figure 6.3(c). This transformation can clearly be done in polynomial time and if G is planar, then G' is also planar.

To show NP-hardness of s -CLUB CLUSTER VERTEX DELETION, it remains to show the following:

There exists a vertex cover in G of size k if and only if there exists a solution of s -CLUB CLUSTER VERTEX DELETION in G' of size k :

“ \Rightarrow ”: Let C be a size- k vertex cover of G . We show that C is a s -CLUB CLUSTER VERTEX DELETION solution as well. By construction, which is illustrated in Figure 6.3(c), each artificial path P'_i in G' is connected to another artificial path P'_j if and only if $(i, j) \in E$. Since C is a vertex cover in G , one of i or j are $\in C$. Thus, deleting the k vertices of C in G' deletes at least one of the vertices i, j . Thereby, in the remaining graph $G'[V' \setminus C]$ each pair of artificial paths P'_i and P'_j is unconnected. In detail these paths consist of the following:

1. If $v \in C : G[P'_v \setminus \{v\}]$ remains as an artificial path without the corresponding natural vertex. This component has diameter $s - 1$.
2. If $v \notin C : G[P'_v]$ remains as an artificial path of length s . This component has diameter s .

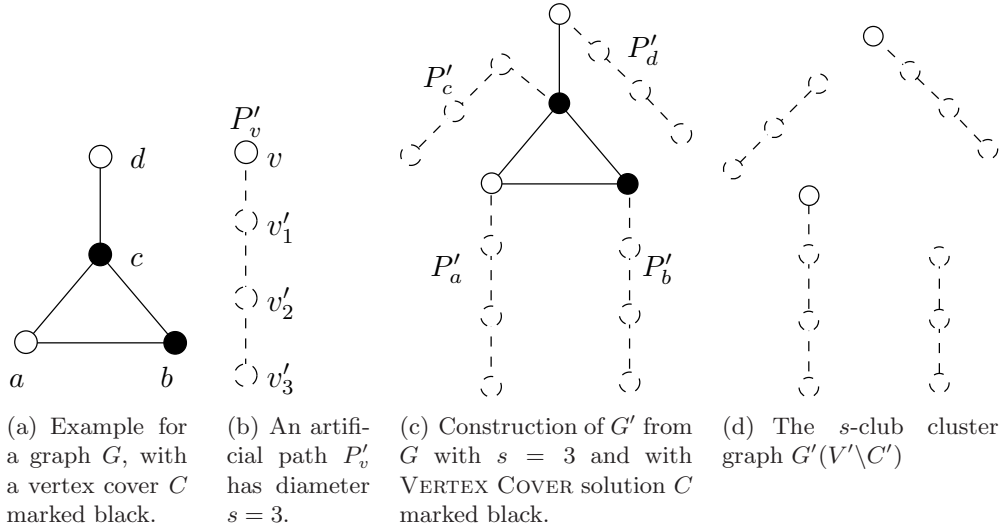


Figure 6.3: The construction of G' from G together with the s -CLUB CLUSTER VERTEX DELETION solution $C' = C$. Artificial vertices and edges are marked dotted.

In Case 1, the natural vertex of the artificial path is part of the vertex cover C in G and is therefore not in $G'[V' \setminus C]$. In Case 2, the connected natural vertex is not part of the vertex cover C in G and therefore still exists in $G'[V' \setminus C]$. Since $G'[V' \setminus C]$ consists only of connected components of diameter at most s , as illustrated in Figure 6.3(d), $G'[V' \setminus C]$ is clearly an s -club cluster graph and C is a size- k solution for s -CLUB CLUSTER VERTEX DELETION in G .

“ \Leftarrow ”: Let C' be a size- k' s -CLUB CLUSTER VERTEX DELETION of G' solution. We show that there is a size- k vertex cover C for G with $k \leq k'$. The construction of G' guarantees that for every natural edge $e \in E$ there exists a path of length $2s + 1$ in G' , which is a subgraph of G' with diameter $2s + 1$, as illustrated in Figure 6.4(a). This subgraph needs to be destroyed, which can only be done by deleting at least one vertex from this subgraph. In order to build the vertex cover C from C' , C is initialized empty, then two cases for the vertices in C' are separated:

1. If $\exists v \in C' : v \in E$ then set $C := C \cup \{v\}$. Illustrated in Figure 6.4(b).
2. If $\exists v'_i \in C' : v'_i \in E'$ then set $C := C \cup \{v\}$. Illustrated in Figure 6.4(c).

In Case 1 a vertex from a natural edge is in C' . Then we simply take this vertex in our vertex cover C . In Case 2 the vertex v'_i of C' is an artificial vertex, instead of

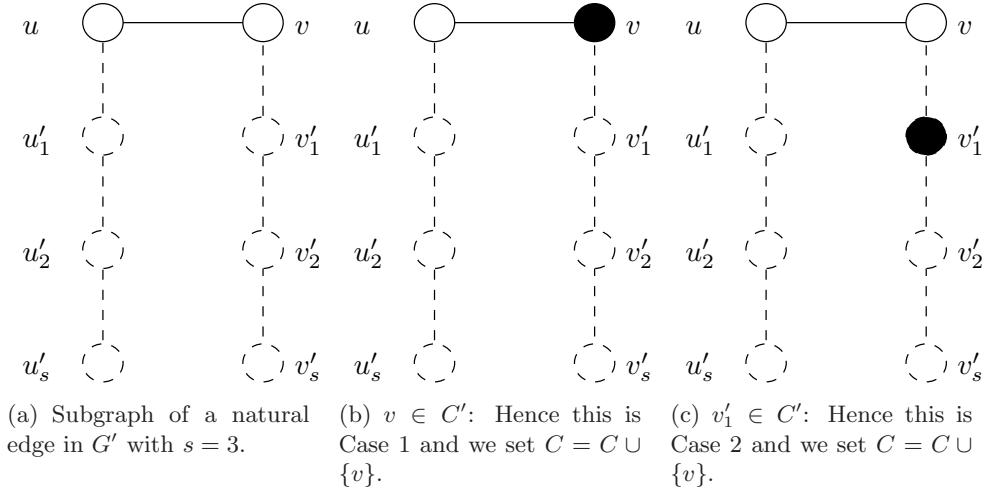


Figure 6.4: The subgraph of a natural edge in G' and the construction of an VERTEX COVER solution C .

v'_i we take the natural vertex v which is incident with the corresponding artificial path $P'_v = (v, v'_1, \dots, v'_s)$ in the vertex cover C .

Claim: C is vertex cover of size $\leq k'$ in G .

Assume that C is not a vertex cover in G . Then we would have at least two connected artificial paths P'_i and P'_j in G' , for which neither one vertex from P'_i nor from P'_j is in C' . Therefore $G[V' \setminus C']$ would have at least one subgraph $G[V(P'_i \cup P'_j)]$ with diameter $2s + 1$ and thus C' would not be a solution of s -CLUB CLUSTER VERTEX DELETION in G' . It follows that the assumption is wrong and the Claim is correct. Thus there exists a size- k solution for VERTEX COVER in G . \square

6.2.2 s -Club Cluster Vertex Deletion is FPT

To show the fixed-parameter tractability of s -CLUB CLUSTER VERTEX DELETION we give a search tree with a size bounded from above by a function of k .

Theorem 6.3. *The s -CLUB CLUSTER VERTEX DELETION problem is fixed-parameter tractable with respect to parameters k and s , that is, it can be solved in $O((2 + s)^k \cdot n(n + m))$ time.*

Proof. The search tree strategy proceeds as follows. Search for a pair of vertices (u, v) with $d(u, v) = s + 1$. If no such pair exists then we already found a solution

set. The search for a pair takes $O(n(n+m))$ running time using breadth-first search for each of the n vertices in the graph. Let P be the shortest path between u and v . Then branch into $s+2$ subcases of deleting one of the $s+2$ vertices in P and set $k := k-1$. Hence, the number of cases in each search tree node is bounded by $s+2$. The algorithm terminates if either $k=0$ or a valid solution set of size $\leq k$ has been found. Hence, the size of the search tree is bounded to $(s+2)^k$. \square

6.2.3 s -Club Cluster Vertex Deletion on Trees

In this section, we give an algorithm that solves s -CLUB CLUSTER VERTEX DELETION on trees in $O(n \cdot s)$ time. To show the correctness of this algorithm, we need the following two lemmas. The vertices described in the lemmas are illustrated in Figure 6.5.

Lemma 6.3. *Let v be a vertex with one child and let u be this child of v , where T_u has diameter s . If $h(T_u) = s$ then deleting v is optimal; otherwise, $V(T_v)$ is an s -club.*

Proof. If $h(T_u) = s$, then T_v cannot have diameter s and therefore we need to delete at least one vertex from T_v . The vertex v has the maximum possible height of all vertices from T_v . Hence, it is optimal to include v into the solution C . If $h(T_u)$ is less than s , then $h(T_v) = 1 + h(T_u)$ is at most s and clearly $\text{diam}(T_v) = \max(1 + h(T_u), \text{diam}(T_u))$ is at most s , and $V(T_v)$ is an s -club. \square

In the following Lemma 6.4 we will use the term highest subtree. Let T be a subtree, and let d be the distance between the root and a lowest leaf of the tree. Then the highest subtree of a set of subtrees, is the subtree that maximizes d over all subtrees in the set.

Lemma 6.4. *Let u be a vertex with more than one child and let t_1, \dots, t_i be those children of u , where each $T_{t_1 \dots i}$ is an s -club. Now let T_{t_1} and T_{t_2} be the highest subtrees over all subtrees $T_{t_1 \dots i}$. If $h(T_{t_1}) + 2 + h(T_{t_2})$ is greater than s , then deleting u is optimal, otherwise $h(T_{t_1}) + 2 + h(T_{t_2})$ is at most s and thereby $V(T_u)$ is an s -club.*

Proof. Since t_1 and t_2 are the vertices that induce the highest subtrees over all children of u and T_{t_1}, T_{t_2} have diameter s and the vertices t_1 and t_2 have distance two, the diameter of T_u is greater s if and only if $h(T_{t_1}) + 2 + h(T_{t_2})$ is greater than s . So if $\text{diam}(T_u) \geq s$, then we need to delete at least one vertex from T_u . In

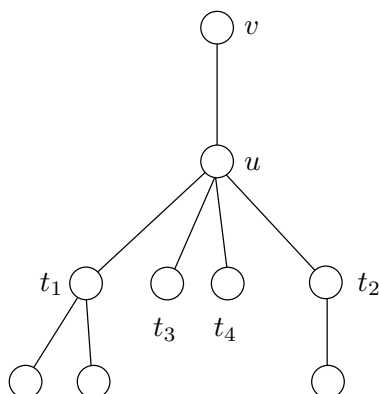


Figure 6.5: Illustration of the vertices from Lemma 6.3 and Lemma 6.4. Vertex u has four children t_1, t_2, t_3 , and t_4 , where t_1 and t_2 are the two children which induce the highest subtree.

T , vertex u has the maximum height of all vertices from T_u . Hence, it is optimal to add u to a solution of s -CLUB CLUSTER VERTEX DELETION. \square

In Figure 6.6 we give the algorithm *Tree- s -Club-Cluster-Vertex-Deletion*. In the algorithm we use the operator `children()`, which will give us the number of children of a certain vertex. The idea of this algorithm is to apply the knowledge from Lemmas 6.3 and 6.4 to each vertex of the tree in a bottom-up way. This means starting at tree-level one, we process the vertices levelwise until we finish with the root vertex r . In each level we check for each vertex the purely local measurements `children()` and `h()`. The algorithm always gives a solution set C . We may check after computing C whether the size of C is at most k and thus is a solution of s -CLUB CLUSTER VERTEX DELETION.

Lemma 6.5. *The algorithm *Tree- s -Club-Cluster-Vertex-Deletion* given in Figure 6.6 is correct.*

Proof. In the algorithm given in Figure 6.6 we do not process any vertices that have no child. Vertices without a child induce a subtree with diameter one, which is less than s , so we do not have to process them. In the iteration starting in line 3 we process every vertex with one child, if this vertex has $h(v) = s$ we add the vertex v to the solution C and remove its subtree from the tree. By Lemma 6.3 this is correct. By removing the induced subtree we ensure the correct values for future height calculations. In line 5 we process every v with at least two children, if the two children t_1 and t_2 with the maximum height fulfill $(h(t_1) + h(t_2) + 2 > s)$,

Algorithm: *Tree-s-Club-Cluster-Vertex-Deletion*

Input: A tree T

Output: A minimum solution set C for s -CLUB-CLUSTER-VERTEX-DELETION on T

```

1   Root the tree  $T$  arbitrarily
2    $l_{max} := h(\text{root})$ 
3   for  $r := 1$  to  $l_{max}$ 
4       for each  $v \in V$  with  $l(v) = r$ 
5           if  $(\text{children}(v) = 1) \wedge (h(v) = s)$  then set  $C := C \cup \{v\}$ ,
               $T := T - V(T_v)$ 
6           if  $(\text{children}(v) > 1) \wedge \max\{h(t_1) + h(t_2) + 2 \mid t_1, t_2 \in \text{child}(v) \wedge (t_1 \neq t_2)\} > s$  then set  $C := C \cup \{v\}$ ,  $T := T - V(T_v)$ 
7       endfor
8   endfor
9   return  $C$ 

```

Figure 6.6: Pseudo-code of the algorithm *Tree-s-Club-Cluster-Vertex-Deletion*.

we add v to the solution C and remove its subtree from the tree. By Lemma 6.4 this is correct. By Lemmas 6.3 and 6.4 we also know that in any other case the induced subtree T_v of a vertex v has diameter s . Since we processed all relevant vertices of the tree and destroyed each subtree which is not an s -club, we finish in line 9 with a correct solution C . \square

Theorem 6.4. s -CLUB CLUSTER VERTEX DELETION on trees can be solved in $O(n \cdot s)$ time.

Proof. By Lemma 6.5 we know the algorithm *Tree-s-Club-Cluster-Vertex-Deletion* given in Figure 6.6 is correct. Thus we need to show that the algorithm *Tree-s-Club-Cluster-Vertex-Deletion* needs $O(n \cdot s)$ running time. Since the height of a subtree to process is at most s , height calculation in this algorithm takes $O(s)$ time. Thus, each of the lines 5 and 6 take $O(s)$ time. Since the algorithm iterates over the vertices of the tree, the iteration takes $O(n \cdot s)$ time. Each other step of the algorithm runs in constant time. Thus the algorithm runs in $O(n \cdot s)$ time. \square

7 Outlook and Conclusion

In this chapter, we give some conclusions from this thesis in Section 7.1 and an outlook for future research in Section 7.2.

7.1 Conclusion

In this work, we studied the parameterized complexity of the s -CLUB and related vertex deletion problems. In Chapter 2 we defined the basic notation needed for this work and gave a brief introduction to parameterized complexity. Chapter 3, introduced the s -CLUB problem in detail and discussed its properties. We showed the following results for:

s -Club on General Graphs

- Parameterized s -CLUB cannot have a polynomial kernel, unless the polynomial hierarchy collapses to the third level.
- For even s , s -CLUB admits a k^2 -vertex Turing kernel, which can be computed in $O(n(n + m))$ time.
- For odd s , s -CLUB admits a k^3 -vertex Turing kernel, which can be computed in $O(n(n + m))$ time.
- The s -CLUB problem can be solved in $O(n(n + m) + n((k - 2)^k \cdot k! \cdot k^3))$ time.

The data reduction results show intractability for many-to-one kernelization and tractability for Turing kernelization side by side. Therefore s -CLUB is another problem which demonstrates that Turing kernelization is a reasonable approach to parameterized complexity. Due to the kernelization, we also strengthened the result given by Komusiewicz [Kom07], by showing that s -CLUB problem is fixed-parameter tractable with respect to the parameter k . Furthermore, we gave a

combined algorithm that gives an approximation and a Turing kernel for s -CLUB. We have shown that this algorithm never outputs a large vertex Turing kernel and a bad approximation for the same instance.

s -Club on Special Graph Classes

- s -CLUB on trees can be solved in $O(n \cdot s^2)$ time.
- Computing the maximum s -club on interval graphs can be solved in $O(n^2)$ time.
- The 2-CLUB problem on bipartite graphs can be solved in $O(n^5)$ time.
- On graph classes with bounded clique- or treewidth the s -CLUB problem is decidable in polynomial time.
- On planar graphs the s -CLUB problem is fixed-parameter tractable with respect to parameter s .

Vertex Deletion Problems In Chapter 6, we introduced two vertex deletion problems, which are closely related to s -CLUB. For these deletion problems we have shown the following results:

- The s -CLUB VERTEX DELETION problem is fixed-parameter tractable with respect to parameter k , that is, it can be solved in $O(2^k \cdot n(n + m))$ time.
 - s -CLUB CLUSTER VERTEX DELETION is NP-complete, even if the input graph G is planar.
 - The s -CLUB CLUSTER VERTEX DELETION problem is fixed-parameter tractable with respect to parameters k and s , that is, it can be solved in $O((2 + s)^k \cdot n(n + m))$ time.
 - s -CLUB CLUSTER VERTEX DELETION on trees can be solved in $O(n \cdot s)$ time.
-

7.2 Outlook

Although we gave some interesting results in this work, many questions remain open. We have shown a k^3 -vertex Turing kernel for s -CLUB. So far, however our branching algorithm does not utilize this kernel in a sufficient way. Furthermore, we achieved this Turing kernel with one simple reduction rule. Therefore, we ask two questions:

- Can we utilize the Turing kernels in a better way to obtain a faster algorithm for s -CLUB on general graphs?
- Is there a linear-size Turing kernel for s -CLUB?

We have shown that s -CLUB is polynomial time solvable on a variety of graph classes. But for those graph classes the CLIQUE problem is polynomial time solvable as well. In order to work out the difference between CLIQUE and s -CLUB, the following question is of interest:

- Is there a graph class for which CLIQUE is NP-hard and s -CLUB is polynomial time solvable, or vice versa?

We have shown fixed-parameter tractability for s -CLUB VERTEX DELETION and s -CLUB CLUSTER VERTEX DELETION. Recall, that since being an s -club is a non-hereditary graph property, there is no characterization via forbidden subgraphs for s -CLUB VERTEX DELETION and s -CLUB CLUSTER VERTEX DELETION. Vertex deletion problems which can be characterized over a fixed number of forbidden subgraphs always admit a polynomial problem kernel [Mos09]. In contrast, little is known about kernels for vertex deletion problems which cannot be characterized in such a way. Therefore, the following two questions, besides the practical interest, are also of great theoretical interest.

- Is there a nontrivial problem kernel of polynomial-size for s -CLUB VERTEX DELETION or s -CLUB CLUSTER VERTEX DELETION?
 - Is there a $o^*(2^k)$ -algorithm for s -CLUB VERTEX DELETION or $o^*((s+2)^k)$ -algorithm for s -CLUB CLUSTER VERTEX DELETION?
-

7.3 Acknowledgements

I would like to thank my advisers Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier for their great support. They always had an open ear for my questions, spent hours in discussing my issues and proofreading my drafts. I am especially thankful for the always friendly working atmosphere they provided. Moreover, I would like to thank Anja Schäfer, Christian Zentgraf, Ramona Starke and Michael Volkhardt for reading my drafts and giving me some helpful comments. Finally, I would like to thank my family and my girlfriend Maria for their continued and invaluable moral support in difficult times.

Bibliography

- [Alb73] R. D. Alba. A graph-theoretic definition of a sociometric clique. *Journal of Mathematical Sociology*, 3:3–113, 1973. →[1, 12]
- [AS99] N. Abbas and L. Stewart. Clustering bipartite and chordal graphs: Complexity, sequential and parallel algorithms. *Discrete Applied Mathematics*, 91(1-3):1–23, 1999. →[46]
- [ASBS00] L. A. N. Amaral, A. Scala, M. Barth, and H. E. Stanley. Classes of small-world networks. *Proceedings of the National Academy of Sciences of the United States of America*, 97(21):11149–11152, October 2000. →[2]
- [Bak94] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994. →[39]
- [BBHS09] B. Balasundaram, S. Butenko, I. V. Hicks, and S. Sachdeva. Clique relaxations in social network analysis: The maximum k -plex problem. *Operations Research*, 2009. To appear. →[16]
- [BBT05] B. Balasundaram, S. Butenko, and S. Trukhanov. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization*, 10(1):23–39, 2005. →[3, 11, 12]
- [BDFH09] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, April 2009. →[19, 20]
- [BK08] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Computer Journal*, 51(3):255–269, 2008. →[39]
- [BKMT04] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Discrete Applied Mathematics*, 136(2-3):183–196, 2004. CTW’04: Proceedings of the 1st Cologne-Twente Workshop on Graphs and Combinatorial Optimization. →[38]
- [BLP00] J. Bourjolly, G. Laporte, and G. Pesant. Heuristics for finding k -clubs in an undirected graph. *Computers and Operations Research*, 27(6):559–569, 2000. →[3]

-
- [BLP02] J. Bourjolly, G. Laporte, and G. Pesant. An exact algorithm for the maximum k -club problem in an undirected graph. *European Journal of Operational Research*, 138(1):21–28, April 2002. →[3, 11, 13, 15, 20]
- [Bod96] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. →[38]
- [Bod98] H. L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998. →[38]
- [BP07] S. Butenko and O. Prokopyev. On k -club and k -clique numbers in graphs. Technical report, Texas A and M University, 2007. Submitted. →[3, 11]
- [BTY09] H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. In *ESA '09: Proceedings of the 17th Annual European Symposium on Algorithms*, volume 5757 of *Lecture Notes in Computer Science*, pages 635–646. Springer, 2009. →[19]
- [CHL⁺00] D. G. Corneil, M. Habib, J. Lanlignel, B. A. Reed, and U. Rotics. Polynomial time recognition of clique-width ≤ 3 graphs (extended abstract). In *LATIN '00: Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, volume 1776 of *Lecture Notes in Computer Science*, pages 126–134. Springer, 2000. →[38]
- [Chu87] F. R. K. Chung. Diameters of graphs: Old problems and new results. *Congressus Numerantium*, 60, 1987. →[17]
- [CLRS01] T. H. Cormen, C. E. Leiserson, R. L Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001. →[24, 28]
- [CM93] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1-2):49–82, 1993. →[39]
- [CMR01] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1):23–52, 2001. →[12, 38]
- [CO00] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. →[12, 38]
-

-
- [Cou90] B. Courcelle. Graph rewriting: an algebraic and logic approach. In *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*, pages 193–242. MIT Press, 1990. →[39]
- [Cou91] B. Courcelle. The monadic second-order logic of graphs V: on closing the gap between definability and recognizability. *Theoretical Computer Science*, 80(2):152–202, 1991. →[39]
- [Dam73] R. M. Damerell. On moore graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 74(02):227–236, 1973. →[1, 17]
- [DdFS07] V. M. F. Dias, C. M. H. de Figueiredo, and J. L. Szwarcfiter. On the generation of bicliques of a graph. *Discrete Applied Mathematics*, 155(14):1826–1832, 2007. →[37]
- [Del85] C. Delorme. Large bipartite graphs with given degree and diameter. *Journal of Graph Theory*, 9(3):325–334, 1985. →[1, 17]
- [DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in computer science. Springer, 1999. →[2, 6, 8, 11]
- [Die05] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, third edition, 2005. →[5]
- [Din70] E. A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math Doklady*, 11:1277–1280, 1970. →[37]
- [DKS97] J. S. Deogun, D. Kratsch, and G. Steiner. An approximation algorithm for clustering graphs with dominating diametral path. *Information Processing Letters*, 61(3):121–127, 1997. →[46]
- [DLS09] M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and IDs. In *ICALP '09: Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2009. →[19]
- [EFH80] P. Erdős, S. Fajtlowicz, and A. J. Hoffman. Maximum degree in graphs of diameter 2. *Networks*, 10(1):87–90, 1980. →[1]
- [EK72] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972. →[37]
- [Eke95] S. M. Eker. Associative-commutative matching via bipartite graph matching. *The Computer Journal*, 38(5):381–399, 1995. →[37]
- [ER62] P. Erdős and A. Rényi. On a problem in the theory of graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 7(A):623–641, 1962. →[1]
-

-
- [FFL⁺09] H. Fernau, F. V. Fomin, D. Lokshtanov, D. Raible, S. Saurabh, and Y. Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In *STACS '09: Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science*, pages 421–432. IBFI Schloss Dagstuhl, 2009. →[19, 21]
- [FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. →[12, 39]
- [FGMN09] M. R. Fellows, J. Guo, H. Moser, and R. Niedermeier. A complexity dichotomy for finding disjoint solutions of vertex deletion problems. In *MFCS '09: Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science 2009*, pages 319–330. Springer, 2009. →[46]
- [FHP81] A. Farley, S. Hedetniemi, and A. Proskurowski. Partitioning trees: matching, domination, and maximum diameter. *International Journal of Parallel Programming*, 10(1):55–61, 1981. →[46]
- [FHS95] M. Fellows, P. Hell, and K. Seyffarth. Large planar graphs with given diameter and maximum degree. *Discrete Applied Mathematics*, 61(2):133–153, 1995. →[1, 17]
- [FRRS06] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. Clique-width minimization is NP-hard. In *STOC '06: Proceedings of the 38th annual ACM symposium on Theory of computing*, pages 354–362. ACM, 2006. →[38]
- [FS08] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *STOC '08: Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 133–142. ACM, 2008. →[20]
- [FW00] D. A. Fell and A. Wagner. The small world of metabolism. *Nature Biotechnology*, 18(11):1121–1122, November 2000. →[2]
- [GHN06] O. Giménez, P. Hliněný, and M. Noy. Computing the tutte polynomial on graphs of bounded clique-width. *SIAM Journal on Discrete Mathematics*, 20(4):932–946, 2006. →[38]
- [GJ77] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977. →[47]
- [GKNU09] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. A more relaxed model for graph-based data clustering: s -plex editing. In *Proceedings of the 5th International Conference on Algorithmic Aspects*
-

-
- in *Information and Management (AAIM '09)*, volume 5564 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 2009. →[16]
- [Gol80] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980. →[34]
- [GR99] M. C. Golumbic and U. Rotics. On the clique-width of perfect graph classes. In *WG '99: Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 1665 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 1999. →[38]
- [Gro09] J Grossman. The erdos number project, 2009. [Online; accessed 16-November-2009]. →[2]
- [HKMN09] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 2009. To appear. →[46]
- [HS60] A. J. Hoffman and R. R. Singleton. On moore graphs with diameters 2 and 3. *Journal of Research and Development*, 4:497–504, 1960. →[1, 17]
- [HSM82] W. R. Heller, G. Sorkin, and K. Maling. The planar package planner for system designers. In *DAC '82: Proceedings of the 19th Design Automation Conference*, pages 253–260. IEEE Press, 1982. →[39]
- [II81] M. Imase and M. Itoh. Design to minimize diameter on building-block network. *IEEE Transactions on Computers*, 30(6):439–442, 1981. →[1, 17]
- [JC04] B. Jiang and C. Claramunt. Topological analysis of urban street networks. *Environment and Planning B: Planning and Design*, 31(1):151–162, January 2004. →[2]
- [KHMN09] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38-40):3640–3654, 2009. →[16]
- [Kom07] C. Komusiewicz. Various isolation concepts for the enumeration of dense subgraphs. Diplomarbeit, Friedrich-Schiller-Universität Jena, March 2007. →[2, 3, 15, 53]
- [KW01] B. Kogut and G. Walker. The small world of germany and the durability of national networks. *American Sociological Review*, 66(3):317–335, 2001. →[2]
- [KW09] S. Kratsch and M. Wahlström. Two edge modification problems without polynomial kernels. In *IWPEC '09: Proceedings of the 4th Inter-*
-

-
- national Workshop on Parameterized and Exact Computation*, Lecture Notes in Computer Science. Springer, 2009. To appear. →[19]
- [LMSL92] C. Li, S. T. McCormick, and D. Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Operations Research Letters*, 11(5):303–308, 1992. →[18]
- [LMSS01] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001. →[37]
- [Lok09] D. Lokshantov. *New Methods in Parameterized Algorithms and Complexity*. PhD thesis, Universitetet i Bergen, Bergen, Norway, April 2009. →[9]
- [ML06] N. Memon and H. Legind Larsen. Structural analysis and destabilizing terrorist networks. In *DMIN '06: Proceedings of the 2006 International Conference on Data Mining*, pages 296–302. CSREA Press, 2006. →[2]
- [MM02] J. Marincek and B. Mohar. On approximating the maximum diameter ratio of graphs. *Discrete Mathematics*, 244(1-3):323–330, 2002. →[3, 25]
- [MNS09] H. Moser, R. Niedermeier, and M. Sorge. Algorithms and experiments for clique relaxations—finding maximum s-plexes. In *SEA '09: Proceedings of the 8th International Symposium on Experimental Algorithms*, pages 233–244. Springer, 2009. →[16]
- [Mos09] H. Moser. *Finding optimal solutions for covering and matching problems*. PhD thesis, Friedrich-Schiller-Universität Jena, Jena, Germany, June 2009. →[55]
- [MS02] J. M. MONTROYA and R. V. SOLÉ. Small world patterns in food webs. *Journal of Theoretical Biology*, 214(3):405–412, 2002. →[2]
- [MS05] M. Miller and J. Siran. Moore graphs and beyond: A survey of the degree/diameter problem. *The Electronic Journal of Combinatorics*, 14:1–61, 2005. →[17]
- [NG09] J. Nastos and Y. Gao. A note on the hardness of graph diameter augmentation problems. *Computing Research Repository*, abs/0909.3877, 2009. →[18]
- [NGD⁺06] A. A. Nanavati, S. Gurusurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjea, and A. Joshi. On the structural properties of massive telecom call graphs: findings and implications. In *CIKM '06:*
-

-
- Proceedings of the 15th ACM International Conference on Information and Knowledge management*, pages 435–444. ACM, 2006. →[2]
- [Nie06] R. Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, March 2006. →[7, 12, 39]
- [Oum08] S. Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):1–20, 2008. →[38]
- [Pas08] S. Pasupuleti. Detection of protein complexes in protein interaction networks using n -clubs. In *EvoBIO '08: Proceedings of the 6th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, volume 4973 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2008. →[2]
- [Pei09] B. W. X. Pei. A parallel algorithm for enumerating all the maximal k -plexes. In *PAKDD '09: Proceedings of the 13th Pacific-Asia Conference on Emerging Technologies in Knowledge Discovery and Data Mining*, volume 4819 of *Lecture Notes in Computer Science*, pages 476–483. Springer, 2009. →[16]
- [Ple74] J. Plesnik. One method for proving the impossibility of certain moore graphs. *Discrete Mathematics*, 8(4):363–376, 1974. →[17]
- [Pri95] E. Prisner. Graphs with few cliques. In *Proceedings of the 7th Conference on Graph Theory, Combinatorics, Algorithms and Applications 1992*, pages 945–956, 1995. →[39]
- [SBL87] A. A. Schoone, H. L. Bodlaender, and J. Van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11(3):409–427, 1987. →[18]
- [Sch98] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, June 1998. →[13]
- [SF78] S.B. Seidman and B.L. Foster. A graph theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978. →[16]
- [SHOG07] D. Seese, P. Hlineny, S. Oum, and G. Gottlob. Width parameters beyond tree width and their applications. *Computer Journal*, 41:1–37, Oktober 2007. →[38]
- [Sit07] H. Situngkir. Small world network of athletes: Graph representation of the world professional tennis player, July 2007. →[2]
- [SS07] D. Schultes and P. Sanders. Dynamic highway-node routing. In *WEA '07: Proceedings of the 6th International Workshop on Experimental*
-

- Algorithms*, volume 4525 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 2007. →[39]
- [SZ04] O. Sporns and J. D. Zwi. The small world of the cerebral cortex. *Neuroinformatics*, 2(23):145–162, 2004. →[2]
- [TIAS77] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, 6(3):505–517, 1977. →[39]
- [TM69] J. Travers and S. Milgram. An experimental study of the small world problem. *Sociometry*, 32(4):425–443, 1969. →[2]
- [TN91] T. Tokuyama and J. Nakano. Geometric algorithms for a minimum cost assignment problem. In *SCG '91: Proceedings of the 7th annual symposium on Computational geometry*, pages 262–271. ACM, 1991. →[37]
- [TS76] S. Tanaka and H. A. Scheraga. Medium- and long-range interaction parameters between amino acids for predicting three-dimensional structures of proteins. *Macromolecules*, 9(6):945–950, 1976. →[1]
- [WS98] D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, June 1998. →[2]
-

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Jena, 8. Dezember 2009