MASTER'S THESIS

Analysis of a Maximal Clique Finding Algorithm with respect to Runtime and Effectiveness in High Dimensional Data

Author: Leonie Friederike Litzka

Supervisor: Prof. Dr. Moritz Grosse-Wentrup

Institut für Statistik

Fakultät für Mathematik, Informatik und Statistik

Ludwig-Maximilians-Universität München

28.08.2019

Abstract

A couple of applications require enumerating of all maximal cliques of a graph in high dimensional data. As this is an NP-complete problem, there is a big interest in finding algorithms, that are capable of efficiently finding maximal cliques.
In this thesis three different types of graphs are considered for investigating an algorithm put forward by Ding et al. [2008] for finding one maximal clique in a graph. This algorithm is studied regarding its runtime and its effectiveness for all three types of graphs. Effectiveness refers to the question, whether the algorithm carried out on different inputs is able to find different existing maximal cliques in the graph. Runtime is considered depending on the size of the graph.
The three considered types of graphs cover first manually set graphs with thoughtful clique structures, secondly random graphs and thirdly random intersection graphs.
For all three types of graphs the mean CPU-runtime of the algorithm was found to be increasing over-proportionally with the size of the graph within the segment of considered graph sizes. The non-linearity was examined in more detail by modelling the logarithm of the mean runtimes and the logarithm of the graph sizes. It indicates in most of the investigated cases, that the logarithm of the mean runtime increases more than linear with the logarithm of the number of nodes.
Permutation tests were used for testing differences between the mean runtimes of the manually set graphs with same size but different clique structures. They indicate, that in particular graphs with non-overlapping maximal cliques have lower mean runtimes compared to corresponding graphs with overlapping maximal cliques.
The proportion of found different maximal cliques after having applied the algorithm on various inputs was found to be relatively low for some graphs, in most cases even decreasing with the increasing graph size.

Contents

List of Figures

List of Tables

1 Introduction

In these days we are generally confronted with a growing amount of data in many applications. Dealing with high dimensional data hence is an important task and analysis techniques must be able to do so. In particular finding maximal cliques in a graph is required for example in data-intensive fields like the detection of consistently co-expressed genes in systems biology [Pavlopoulos et al., 2011].
Enumerating all maximal cliques of a graph is an NP-complete problem and therefore sensitive to increasing dimensionality of the data. Correspondingly there is a strong need for algorithms, that are capable of efficiently finding maximal cliques. [Hou et al., 2016]
This thesis investigates an algorithm put forward by Ding et al. [2008] for finding one maximal clique in a graph per run. The thesis studies the algorithm regarding runtime and effectiveness for three types of graphs, which differ with respect to the structure of their cliques. While the runtime was examined as function of the size of the graph, i.e. the number of nodes, effectiveness addresses the question, to what extent the algorithm is able to find different existing maximal cliques in the graph.
For investigating these questions, three types of graphs were considered. In the first group of graphs their clique structure was manually set, for understanding the impact of the clique structure on runtime and effectiveness. The second group were random graphs [Erdős and Rényi, 1959], the third random intersection graphs [Behrisch and Taraz, 2006], which were chosen as an attempt of being closer to real-world examples. In order to examine the runtimes, for all these three groups of graphs various graph sizes were taken into account.
The thesis is structured as follows. Section 2 will introduce the theory of maximal clique finding, non-negative matrix factorization and the algorithm published by Ding et al. [2008]. The implementation of this algorithm and the methods, which were applied for investigating runtime and effectiveness, will be described in section 3. This is followed by the presentation of the results in section 4. Finally section 5 will summarize the results and give an outlook.

2 Theoretical Background

Aim of the following sections is to give the theoretical background for the idea of efficiently searching maximal cliques in high dimensional data.
Section 2.1 will first introduce the notation of this thesis, after that 2.2 will explain the concept of maximal clique finding. Section 2.3 will take a closer look at non-negative matrix factorization. This is followed in section 2.4 by the introduction of the algorithm put forward by Ding et al. [2008], which is studied in this thesis regarding its runtime and effectiveness.

2.1 Notation

The notation of formulas coming from cited sources is adapted on the notation of this thesis.
Capital letters like B are used for matrices. In particular A denotes adjacency matrices, which are introduced in section 2.2. An entry of a matrix B in the i-th row and j-th

column is denoted by B_{ij}. $B_{i.}$ is the i-th row, $B_{.j}$ the j-th column of B. A vector is indicated by a lower-case letter like x, the i-th element of a vector by x_i. The letters i, j and k are used for indices, running indices as well as for nodes. The letters r, m and n represent numbers and dimensions of matrices. Especially n also stands for the number of nodes of a graph.
If vectors or matrices are iteratively updated, the current iteration respectively update step is marked via a superscript (k) in round brackets like $x^{(k)}$. In the following section the theory of maximal clique finding will be presented.

2.2 Maximal Clique Finding

Finding maximal cliques is a problem of the field of graph theory. A graph consists of a set of nodes respectively nodes and edges, which do or do not connect the nodes of the graph. If two nodes are connected by an edge, they are called adjacent. A graph can be directed or undirected, which depends on whether the edges of the graph are directed or undirected. This thesis considers undirected graphs, in which all edges have no direction i.e. no in and out ends. [Pearl et al., 2016]

An undirected graph $G = (V, E)$ consists of two sets V and E. $V = \{1, 2, ..., n\}$ is a set containing the n nodes of the graph. These nodes in V are connected via edges, which are the elements of the set E. If two nodes i, j are connected via an edge (i, j), then $(i, j) \in E$. [Belachew and Gillis, 2017]
The so called adjacency matrix A of an undirected graph G is a binary matrix filled with 0 and 1. A is quadratic and has as many columns respective rows as there are nodes in the graph. In particular A is symmetric, since the matrix element in row i and column j is the same as the element in row j and column i. It indicates, whether node i and node j are adjacent. The nodes are adjacent if they are connected via an edge, i.e. if $(i, j) \in E$. So $A_{ij} = A_{ji} = 1$ if and only if the nodes i and j are adjacent, otherwise these entries in A are equal to zero. [Belachew and Gillis, 2017]

A so called "clique in an undirected graph G is a subset of its [...] [nodes] such that the corresponding subgraph is complete" [Belachew and Gillis, 2017, p. 280]. So every node in a clique C is adjacent to all other nodes, that are part of the same clique. This means every node is connected via an edge to every other node in the clique:

$$\forall \quad i, j \in C \quad \text{with} \quad i \neq j : \quad (i, j) \in E$$

A clique C is called a maximal clique, if there exists no larger clique, in which the clique C is contained. So if there would exist a node k, which is not part of a clique C and which is adjacent to all nodes of the clique, this clique would not be maximal. There would exist a larger clique C_{larger}, in which C is contained. This larger clique would be the union of the nodes of the clique C and the node k: $C_{larger} = C \cup \{k\}$. [Belachew and Gillis, 2017]
The largest clique of all maximal cliques of an undirected graph G is called the maximum clique of graph G. The maximum clique has the largest number of nodes and therefore also the largest number of edges, as all nodes of a clique have to be adjacent per definition of a clique. The number of nodes contained in the maximum clique is referred to as the clique number of the graph G. [Belachew and Gillis, 2017]

Now the question arises, how to find such maximal cliques of a graph. Corresponding to Ding et al. [2008] the formulation of an optimization problem can help to compute maximal cliques of a graph. The so called Motzkin and Straus formulation is as follows with setting $A_{ii} = 0$ [Ding et al., 2008]:

$$\max_x \quad x^T Ax, \quad s.t. \sum_{i=1}^{n} x_i = 1, \quad x_i \geq 0$$

Motzkin and Straus [1964] consider a maximal clique of size k of a graph G with n nodes, corresponding adjacency matrix A and a simplex S in euclidean n-dimensional space. This simplex is given by $S = \{x \in \mathbb{R}^n \quad | \quad x_i \geq 0, \quad \sum x_i = 1\}$. They show, that the maximum of the function $\max\limits_{x \in S} \quad \sum_{(i,j) \in G} x_i x_j$ is given by $f(G) := \frac{1}{2}(1 - \frac{1}{k})$ for x with $x_1 = \cdots = x_k = \frac{1}{k}$ and $x_{k+1} = \cdots = x_n = 0$. So the maximum of $f(G)$ is given for a vector x, that indicates the membership in the clique for the nodes $1, \ldots, k$ via the entry $\frac{1}{k}$, whereas the nodes in G, that are not part of the clique, are marked by the entry 0. [Motzkin and Straus, 1964]

Gibbons et al. [1996] look at the Motzkin-Straus formulation of the Maximum Clique Problem

$$max \quad x^T Ax/2 \quad s.t. \quad e^T x = 1, \quad x \geq 0$$

for a graph G and its adjacency matrix A. They derive, that maximal cliques of the graph G correspond to local maxima of the optimization problem. Hence for finding maximal cliques of a graph, the Motzkin-Straus formulation can be used, whilst computing the local maxima of this optimization problem. [Gibbons et al., 1996]

Finding all maximal cliques of a graph is also called maximal clique enumeration [Hou et al., 2016]. This is "NP-Complete and thus computationally intensive at scale" [Hou et al., 2016, p. 219]. So for high-dimensional data and resulting large graphs it is important to find an efficient algorithm for finding the maximal cliques. Ding et al. [2008] provide an algorithm for finding a maximal clique via the framework of non-negative matrix factorization. This algorithm will be put forward in section 2.4. First the theory of non-negative matrix factorization itself will be presented in the following section.

2.3 Non-negative Matrix Factorization

Non-negative matrix factorization is praised by Ding et al. [2008] as having great success in machine learning literature. Ding et al. [2008] use non-negative matrix factorization as a framework for several optimization problems of the field of data mining. One of these optimization problems is maximal clique finding, for which the resulting algorithm will be introduced in section 2.4.

It's the aim of non-negative matrix factorization to factorize a non-negative matrix V into two non-negative matrices W and H, so that $V \approx WH$. This non-negativity constraint is based on the idea of learning a parts-based representation as explained by Lee and Seung [2001]. The matrix H contains so called hidden variables, whereas V consists of the visible variables. "Each hidden variable coactivates a subset of visible variables, or

'part'. Activation of a constellation of hidden variables combines these parts additively to generate a whole." [Lee and Seung, 1999, p. 790]
Lee and Seung [1999] use facial images for demonstrating the parts-based representation. Because all elements of W and H are required to be non-negative, only additive combinations are possible, no subtractions. This corresponds to the idea of putting parts together to build a whole [Lee and Seung, 1999]. Another motivation is mentioned by Hoyer [2004] and aims on applications, where the involved quantities cannot be negative because of rules of physics as an example.

In particular the representation WH of V found by the non-negative matrix factorization is not only parts-based but also linear. Define V as a $n \times m$ matrix, W as a $n \times r$ matrix and H as a $r \times m$ matrix. As Hoyer [2004] explains, for each measurement vector $V_{.k}$ with $k \in 1, ..., m$ in V an approximation is given by $V_{.k} \approx \sum_{i=1}^{r} W_{.i} H_{ik} = WH_{.k}$. The columns of W are the r so called basis vectors $W_{.i}$, that also can be seen as the "'building blocks' of the data" [Hoyer, 2004, p. 1458]. Underlying features should be extracted in the form of these basis vectors [Berry et al., 2007] and make the "latent structure in the data explicit" [Hoyer, 2004, p. 1457]. The vectors $H_{.k}$, that were called the hidden variables by Lee and Seung [1999], contain the coefficients, that indicate, "how strongly each building block is present in the measurement vector $V_{.k}$". Hoyer [2004] emphasizes, that non-negative matrix factorization is - as well as for example the principal component analysis - a matrix factorization, only with different objective functions and/or constraints.

An important point of non-negative matrix factorization is data compression. If r is smaller than n or m and hence W and H also smaller than V, corresponding to Lee and Seung [2001] these two matrices build a "compressed version of the original data matrix [V]" [Lee and Seung, 2001, p. 2]. So dimension reduction can be achieved via non-negative matrix factorization. Lee and Seung [1999] call r the rank of the factorization and give as a rule for it $(n+m)r < nm$, to achieve WH being a compressed version of V. Berry et al. [2007] state the choice of r being a critical point and often chosen as "[...][r] $<< min(m,n)$".
Lee and Seung [2001] point out, that "each data vector [...] [$V_{.k}$] is approximated by a linear combination of the columns of W, weighted by the components of [...] [$H_{.k}$]"[Lee and Seung, 2001, p. 2]. As aforementioned r and thus the number of vectors, that are used to represent the data vectors $V_{.k}$, is chosen relatively small, to gain data compression. So Lee and Seung [2001] reason the quality of the approximation of V via WH being dependent on whether latent structure in the data is discovered by the approximation.

To obtain the matrices W and H, there exist numerical approaches, which can be categorized corresponding to Berry et al. [2007] into three classes. One class consists of multiplicative update algorithms based on an algorithm provided by Lee and Seung [2001]. A second class represents gradient descent algorithms, whereas the third class consists of alternating least squares algorithms.

The multiplicative update algorithm explained by Lee and Seung [2001] needs a cost function for assessing the goodness of the approximation of V by WH. For this purpose two measures of distance are proposed by Lee and Seung [2001], the squared Euclidean

Distance and the Divergence. As an example the Euclidean Distance for two matrices P and Q is given here [Lee and Seung, 2001]:

$$\|P - Q\|^2 \quad = \quad \sum_{ij} (P_{ij} - Q_{ij})^2$$

With each of those two cost functions the non-negative matrix factorization can be formulated as an optimization problem. It is minimizing the respective cost function "with respect to W and H, subject to the constraints $W, H \geq 0$" [Lee and Seung, 2001, p. 3]. Lee and Seung [2001] state, that the two cost functions Euclidean Distance and Divergence are not convex in both W and H at once, but in both of them by itself. They formulate hence the aim of finding local instead of global minima via numerical optimization. [Lee and Seung, 2001]

The optimization problem using the Euclidean Distance as cost function is "[...] [minimizing] $\|V - WH\|^2$ with respect to W and H, subject to the constraints $W, H \geq 0$" [Lee and Seung, 2001, p. 3]. Doing this by multiplicative update rules is suggested by Lee and Seung [2001] as a "good compromise between speed and ease of implementation" [Lee and Seung, 2001, p. 3]. The multiplicative update rule of Lee and Seung [2001] is as follows, if the Euclidean Distance is used as cost function:

$$H_{ij} \leftarrow H_{ij} \frac{(W^T V)_{ij}}{(W^T W H)_{ij}}, \qquad W_{kl} \leftarrow W_{kl} \frac{(V H^T)_{kl}}{(W H H^T)_{kl}}$$

As Berry et al. [2007] report, the algorithm using these multiplicative update rules does not converge to a local minimum. The proof of convergence to a local minimum by Lee and Seung [2001] corresponding to Berry et al. [2007] only "shows a continual descent property, which does not preclude descent to a saddle point" [Berry et al., 2007, p. 158]. Therefore several references are given by Berry et al. [2007]. Berry et al. [2007] instead state about the algorithm of Lee and Seung [2001]: "When the algorithm has converged to a limit point in the interior of the feasible region, this point is a stationary point. This stationary point may or may not be a local minimum. When the limit point lies on the boundary of the feasible region, its stationarity cannot be determined" [Berry et al., 2007, p. 158].

Berry et al. [2007] mark, that algorithms using the above mentioned multiplicative update rules, often converge in practice, but slower than algorithms of the other two at the beginning of this section mentioned classes of gradient descent and alternating least squares algorithms.

The framework of non-negative matrix factorization is used by Ding et al. [2008] in the next section 2.4 for finding maximal cliques.

2.4 Algorithm

The algorithm studied in this thesis regarding its runtime and effectiveness comes from Ding et al. [2008]. They use the concept of non-negative matrix factorization, which was described in the previous section 2.3 as an optimization framework for different purposes, amongst other things for finding maximal cliques.

The basis of the algorithm described by Ding et al. [2008] is the theorem of Motzkin and Straus [1964]. As explained in section 2.2 maximal cliques can be found via solving this optimization problem. Ding et al. [2008] use a generalized version of this optimization problem, which is given as follows:

$$\max_x \quad x^T Ax, \quad s.t. \sum_{i=1}^{n} x_i^\beta = 1, \quad x_i \geq 0$$

The parameter β incorporates a L_p-norm constraint $\sum_{i=1}^{n} x_i^\beta = 1$. Ding et al. [2008] show, that for an adjacency matrix, whose main diagonal is filled with 1, and for $\beta = 1+\theta, \quad 0 < \theta << 1$, a maximal clique is found. If the nonzero elements $C = \{i|x_i > 0\}$ of the vector x all have the same value, C is a maximal clique. The vector x then contains as many nonzero elements as the found maximal clique contains nodes, i.e. $|C|$ nonzero elements. All other elements are equal to zero. The nonzero elements are all equal to $\frac{1}{|C|^{\frac{1}{\beta}}}$ and are placed in the entries of x, whose indices correspond to the indices of the nodes in the adjacency matrix A. [Ding et al., 2008]

First input of the algorithm by Ding et al. [2008] is the adjacency matrix A of an undirected graph, of which maximal cliques should be found. The adjacency matrix is filled with 0 and 1. The elements in the main diagonal are all equal to 0. As second input a vector $x^{(0)}$ is required, which will be updated by the algorithm step by step. The length of this vector has to be equal to the number of rows respectively columns of the adjacency matrix. [Ding et al., 2008]

The algorithm incorporates a multiplicative update rule also used by Pelillo [1995]. The update rule is given for each element $x_i^{(t)}$ of a current solution $x^{(t)}$ as [Ding et al., 2008]:

$$[x_i^{(t+1)}]^\beta = x_i^{(t)} \frac{(Ax^{(t)})_i}{[x^{(t)}]^T Ax^{(t)}}$$

For proving the optimality of the algorithm, Ding et al. [2008] show, that the first Karush-Kuhn-Tucker conditions are fulfilled. The Lagrangian has the following form with λ as the "Lagrangian multiplier for enforcing the L_p-norm constraint" [Ding et al., 2008, p. 188]:

$$L = x^T Ax - \lambda(\sum_i x_i^\beta - 1)$$

The Karush-Kuhn-Tucker condition corresponding to this Lagrangian is [Ding et al., 2008, p. 188]:

$$[2(Ax)_i - \lambda\beta[x_i]^{\beta-1}]x_i = 0$$

The KKT-condition is satisfied at convergence by the following update rule, which yields the previously presented update rule by substituting $\lambda\beta = \lambda\beta \sum_{i=1}^{n} [x_i]^\beta = 2x^T Ax$ [Ding et al., 2008, p. 188]:

$$[x_i^{(t+1)}]^\beta = x_i^{(t)} \frac{(Ax^{(t)})_i}{\frac{\lambda\beta}{2}} = x_i^{(t)} \frac{(Ax^{(t)})_i}{\frac{2[x^{(t)}]^T Ax^{(t)}}{2}} = x_i^{(t)} \frac{(Ax^{(t)})_i}{[x^{(t)}]^T Ax^{(t)}}$$

As the "Lagrangian function L [...] is monotonically increasing (nondecreasing)" [Ding et al., 2008, p. 189] and the objective function is bounded from above, the convergence of the algorithm is given, as Ding et al. [2008] argue.

The following section will present the methods, which were used to study the just introduced algorithm regarding its runtime and effectiveness for three types of graphs.

3 Methods

After the theoretical background was presented in section 2, section 3.1 now explains, how the algorithm of Ding et al. [2008] is implemented for generating results, that can be used for studying its runtime and effectiveness for three different types of graphs.

3.1 Python Implementation of the Algorithm

The algorithm, the analyses of runtime and effectiveness and their depictions were implemented in Python, version 3.7.3 [Python Core Team, 2019]. Important Python modules, that were used for these purposes, are numpy [van der Walt et al., 2011], matplotlib [Hunter, 2007], timeit [timeit], time [time], networkx [Hagberg et al., 2008], seaborn [Michael Waskom et al., 2017], pandas [McKinney, 2010], statsmodels [Seabold and Perktold, 2010] and permute.core [Millman et al., 2019].

For finding maximal cliques in a given graph, the algorithm requires the corresponding adjacency matrix and startvectors. These had to be generated. Therefore a seed was set, so that the analyses are reproducible. One startvector corresponds to a vector, whose length is equal to the number of rows respectively columns of the adjacency matrix. So every element in a startvector corresponds to the node of the graph underlying the given adjacency matrix, which in turn corresponds to the respective row and column in the adjacency matrix. The elements of the startvectors were randomly chosen in the interval $]0, 1[$.

The given adjacency matrix was modified according to Ding et al. [2008] via setting all elements on the main diagonal to zero. With every startvector then the following procedure was performed.
The update step of the algorithm is repeatedly applied on the current startvector until a stopping criterion is met. This criterion is explained in more detail in section 3.1.1. The number of iterations is restricted to 10000. When the updating step is done, either ended by the stopping criterion or the restriction of the number of iterations, the resulting solution vector is found.
Then a strategy also used by Belachew [2014] is applied to extract the elements of the found solution vector, i.e. the nodes, that build the clique in the found solution. Therefore the elements of the solution vector are ordered descendingly. Each element of the ordered vector is added to the clique one by one, if it is adjacent to all elements, that were added to the clique before. When the first element is not adjacent to all previously added elements, the process is stopped and the clique of the current solution is found.
To be sure, that the identified clique is really a clique as well as a maximal clique, the adjacency matrix is used. If all elements identified as clique members are pairwise adjacent in the adjacency matrix, the found clique indeed is a clique. After that it's checked, whether there exists another node, which is pairwise adjacent to all the identified clique

members. If not, the identified clique is in particular a maximal clique.
Hereinafter in this thesis always maximal cliques are meant, even if for shortness only clique is written.

For measuring the runtime of the algorithm, the process time of the update step was measured for every startvector. I.e. the process time was measured from the first update of the startvector until the stopping criterion was met or the restriction of the number of iterations stopped the updating step. The function timeit of the timeit module [timeit] in Python was used with setting its timer to "process_time" for measuring the CPU-time instead of wall clock time. The timeit function offers the possibility of repeating a time measurement. In this thesis for each startvector the time measurement was repeated 10 times. The arithmetic mean of these 10 measurements was used as estimation of the runtime of the algorithm on the respective adjacency matrix and the respective startvector. Hereinafter the expression mean runtime refers to this arithmetic mean of the 10 repeated measurements.

Also the effectiveness of the algorithm was investigated. To do so, the solutions, that resulted when applying the algorithm on the startvectors, were used. For each adjacency matrix there were 100 startvectors randomly chosen as described previously and the algorithm was run on each startvector. Each startvector led to a solution, which was then identified as a clique as described above.

The considered adjacency matrices will be described in more detail in section 3.2. The analyses, that were applied on the runtimes and solutions for studying the runtime and effectiveness of the algorithm, will be presented in the subsequent section 3.3. Beforehand section 3.1.1 will give some more details on the stopping criterion of the algorithm.

3.1.1 Stopping Criterion

The algorithm needs a stopping criterion to decide, when to stop the iterations for updating the current startvector x. Belachew [2014] uses $\|x^{(k+1)} - x^{(k)}\|_2^2 < 10^{-10}$ as stopping criterion as suggested by Pelillo [1995]. This stopping criterion looks at the squared Euclidean Distance between two successive states $x^{(k+1)}$ and $x^{(k)}$.

The stopping criterion, that was used in the implementation for this thesis, is given as $\sum_{i=1}^{n}(x_i^{(k+1)} - x_i^{(k)})^2 / \sum_{i=1}^{n}(x_i^{(k)})^2 < 10^{-9}$. The intention behind this choice was, to obtain a convergence criterion, which is independent from the length of the vector x, i.e. the number of nodes in the graph.

The following section will introduce the three groups of graphs and the associated adjacency matrices, that are were for investigating runtime and effectiveness of the algorithm.

Binary characterizations	Clique properties	
characterization 1	non-overlapping	overlapping
characterization 2	few big	many small
characterization 3	equal-sized	different-sized

Table 1: Overview of the three binary characterizations used for building eight adjacency matrix structures.

3.2 Adjacency matrices

As aforementioned for this thesis various types of graphs and thus various groups of adjacency matrices were considered. Background for choosing such different adjacency matrices were considerations about possible influencing factors on the runtime of the algorithm as well as its effectiveness.

First of all the size of the graphs respectively of the adjacency matrices was considered. As explained in section 2.2 an adjacency matrix is symmetric and its number of rows respectively number of columns corresponds to the number of nodes in the underlying graph of the adjacency matrix. The numbers of nodes taken into account here reach for most of the matrices from 100 over 500, 1000, 1500, 2000, 4000, 6000, 8000 to 10000.

Then there are three types of graphs resulting in three groups of adjacency matrices, that were considered in this thesis. For each of these groups additionally the different matrix sizes were used. Hereinafter the types of graphs will be referred to as the groups of adjacency matrices, since the algorithm, which was investigated regarding runtime and effectiveness, takes an adjacency matrix as input.

For the first type of graphs and thus the first group of adjacency matrices, there were three binary characterizations used for creating eight different clique structures of graphs, which result in corresponding different structures of the associated adjacency matrices. These characterizations are given in table 1 and stand for different properties of cliques in a graph.
Combinations of these three characterizations lead to the eight different clique structures and thus eight different adjacency matrix structures, that were used for examining the runtime and effectiveness of the algorithm. These structures are given in table 2 and describe the properties of the cliques in adjacency matrices with such structures.
The number of cliques differs for the eight matrix structures as given in table 2. In addition for each of the eight structures the aforementioned nine matrix sizes were considered. So for each structure nine matrices were created, each with one of the sizes. The cliques grow proportional with the matrix size, so that all matrices of the same structure always have the same number of cliques.

Only for one matrix structure, “non-overlapping few big equal-sized cliques”, additionally the matrix size of 15000 nodes was considered.
Also the matrices of this structure were used for looking at the effect of more disorganized adjacency matrices on the runtime of the algorithm. So far the adjacency matrices con-

Characterization 1	Characterization 2	Characterization 3	Number of cliques
non-overlapping	few big	equal-sized	5
	many small		20
	few big	different-sized	3
	many small		20
overlapping	few big	equal-sized	9
	many small		24
	few big	different-sized	4
	many small		34

Table 2: Overview of the eight adjacency matrix structures.

structed by the above mentioned structures have a block-wise structure, as it was easier to construct them in this way. To investigate the impact of more disorganized matrices on the runtime, the adjacency matrices the structure “non-overlapping few big equal-sized cliques” were randomly permuted for creating additionally permuted versions of the adjacency matrices. Therefore a permutation was randomly chosen for each matrix size by using a seed. Then first the columns and after that the rows were permuted according to the permutation.
To be able to properly compare the mean runtimes of the non-permuted and permuted adjacency matrices of the matrix structure “non-overlapping few big equal-sized cliques”, it seemed important to ensure, that the results are the same for both non-permuted and permuted matrices. Hence the permuted matrices needed the startvectors of the non-permuted matrices permuted in the same way, as the matrices were permuted compared to the non-permuted matrices. Having this guaranteed, for each permuted startvector the solution of a permuted matrix was the same as the solution of the corresponding non-permuted matrix and non-permuted startvector. Matrix sizes up to 6000 were considered.

The second type of graphs, the random graphs [Erdős and Rényi, 1959], and thus the second group of adjacency matrices refers to another interesting type of adjacency matrices, namely randomly assigned matrices. In such adjacency matrices every edge occurs with a certain fixed probability independently from the other edges. These matrices differ from the first group of matrices in the view, that they do not contain special thoughtful clique structures but instead are completely randomly filled. [Gramm et al., 2009]
Behrisch and Taraz [2006] refer to such randomly filled adjacency matrices as the random graph model $G_{n,p}$, introduced by Erdős and Rényi [1959], where n denotes the number of nodes and p the probability that an edge exists. The matrices used in this thesis were created using an edge probability of $p = 0.1$ like in Gramm et al. [2009]. This means, that all off-diagonal elements of an adjacency matrix are independently set to one with probability $p = 0.1$. The matrix sizes, i.e. the number of nodes of the graphs and thus the number of rows respectively columns of the adjacency matrices, comprised 100, 500, 1000, 1500, 2000, 4000, 6000 and 8000.

Adjacency matrices, that are an attempt of being closer to real-world scenarios than the previously two considered groups of adjacency matrices, form the third group. They were generated via using random intersection graphs. Gramm et al. [2009] argue that real-

world scenarios "are not completely random; in particular, in most sensible applications the clique cover is expected to be much smaller than that of a random graph" [Gramm et al., 2009, p. 12]. Behrisch and Taraz [2006] see a reason for the lack of reality of randomly assigned matrices from random graphs in the independence of the edges, which leads to missing transitivity. They argue that, in real-world scenarios "relations between [...] [nodes] [...] [i] and [...] [j] on the one hand and between [...] [j] and [...] [k] on the other hand suggest a connection of some sort between [...] [nodes] [...] [i] and [...] [k]" [Behrisch and Taraz, 2006, p. 37].
For generating random intersection graphs and corresponding adjacency matrices, the $G_{n,m,p}$ model from Behrisch and Taraz [2006] was used like in Gramm et al. [2009]. A random intersection graph consists of n nodes and m features and each node "chooses each feature independently with probability p" [Behrisch and Taraz, 2006, p. 38]. As Behrisch and Taraz [2006] explain, this probability model incorporates transitivity. If two edges (i, j) and (j, k) are induced by the same feature [...], then this will also induce the edge (i, k) [Behrisch and Taraz, 2006, p. 38].
For a fixed number n of nodes there are the two parameters m and p, that have influence on the adjacency matrix. Behrisch and Taraz [2006] recommend choosing m depending on n as $m = n^\delta$ and focus on $0 < \delta < 1$. In this thesis m was instead chosen fixed as $m = 5$ and p was chosen as $p = 0.8$. Aim of these choices was, to obtain adjacency matrices, that contain not too many cliques and are all based on the same number of features. As matrix sizes $n \in \{100, 500, 1000, 1500, 2000, 4000\}$ was used.

Summarizing, this leads to 95 adjacency matrices. For each of the considered matrix sizes 100 startvectors were randomly chosen. So all adjacency matrices among these 95, which are of the same size, were used with the same 100 startvectors. For each adjacency matrix and each startvector on the one hand the algorithm was applied for finding a solution - i.e. detecting a maximal clique. On the other hand the CPU-time was measured, until a solution was found by the algorithm. How these runtime measurements and solutions were analysed regarding runtime and effectiveness, will be explained in the next section.

3.3 Analysis of the algorithm's runtime and solutions

Runtime and effectiveness are the properties of the algorithm, that are studied in this thesis. As mentioned before 100 startvectors were used for every adjacency matrix, for measuring the runtime and of course also for calculating solutions, such that the effectiveness could be considered as well.

For each adjacency matrix and startvector 10 repetitions of the runtime measurement were taken, the arithmetic mean of these 10 repetitions was considered as described in section 3.1.

The runtime can be viewed from two perspectives. For all three groups of adjacency matrices, the mean runtime was considered depending on the matrix size, i.e. the number of nodes. This was done for every of the eight structures and the permuted version of

structure "‘non-overlapping few big equal-sized cliques"’, which are the matrices of the first group. For the matrices coming from random graphs, that build group two, this was likewise done as well as for the matrices coming from random intersection graphs, which form the third group.
Additionally regression models were used for fitting the mean runtimes depending on the number of nodes. This approach will be explained in section 3.3.1.

For the adjacency matrices of the first group, which were created using structures described in section 3.2, a second perspective exists for studying the mean runtimes. It means looking at the mean runtimes of one matrix size - but for all eight structures. So the measurements between these structures given a specific matrix size could be compared. For properly comparing these structures, permutation tests were used. The procedure of the permutation tests will be presented in detail in section 3.3.2.

For investigating the effectiveness of the algorithm, the solutions were used instead of the runtime measurements. Each startvector together with an adjacency matrix led to a solution, which was then identified as a clique. For each adjacency matrix 100 startvectors were used as already mentioned previously, which resulted in 100 solutions per adjacency matrix.
For using the algorithm for enumerating the all maximal cliques as mentioned at the end of section 2.2, whilst applying it on different randomly chosen startvectors, it would be desirable, that among 100 different startvectors preferably many different cliques are detected and not - to mention an extreme example - 100 times the same clique is found.
Therefore the proportion of found different cliques after 100 startvectors was determined for every matrix, which belongs to the first group and is thus created by a matrix structure. It was calculated as the number of detected different cliques among the 100 solutions divided by the number of existing maximal cliques in the adjacency matrix. For these adjacency matrices based on the adjacency structures the number of existing cliques was known by design.
For the other two groups of adjacency matrices, that belong to random graphs and random intersection graphs, the number of existing cliques was not known. The routine find_cliques from networkx [Hagberg et al., 2008] in Python was used to calculate the number of existing maximal cliques of these matrices. These numbers were much bigger compared to the previously considered matrices created by the adjacency structures. For example the smallest matrix, which belongs to the second group, had 100 nodes and contained 319 cliques. In particular the numbers of existing cliques exceeded the number of startvectors for these matrices.
Additionally for the biggest matrix sizes networkx did not manage to assess the number of cliques because of memory issues. This was the case for the sizes 6000 and 8000 among the random graphs and the sizes 2000 and 4000 among the random intersection graphs. Hence for matrices of these two groups not the proportions of found different cliques among the existing cliques after 100 startvectors were considered. Instead the proportions of detected different cliques among the 100 startvectors were used. These proportions were calculated as the number of detected different cliques after 100 startvectors divided by the number of startvectors, which is 100 for every matrix.

For all adjacency matrices of all three groups, the corresponding proportion could also be calculated after each startvector. Thereby it could be investigated, when a new clique, which was not found so far by the preceding startvectors, was detected depending on the index of the corresponding startvector. This was also combined with the mean runtime via considering the cumulative mean runtime. This means after each startvector the proportion of found different cliques so far was calculated as well as the cumulative mean runtime up to this startvector. I.e. up to the currently considered startvector for each preceding startvector its corresponding mean runtime was taken and summed up to compute the cumulative mean runtime.

For adjacency matrices of the first group, which are based on the four structures with different-sized cliques, also the size of the cliques was taken into account. The question behind this was, whether cliques are found more often than other cliques depending on their size.

In the following two sections the models and permutation tests, that were used to investigate the mean runtimes, will be described in more detail.

3.3.1 Regression

To investigate the mean runtime measurements, models were used for fitting these depending on the number of nodes via ordinary least squares estimation of the module statsmodels [Seabold and Perktold, 2010] in Python. In particular the logarithm to base 10 of the mean runtime measurements and the logarithm to base 10 of the number of nodes was used for investigating non-linearity of the mean runtime measurements. Hereinafter for shortness the logarithm to base 10 of the mean runtimes respectively number of nodes will be referred to as the log mean runtimes and log number of nodes.
Under the assumption, that the mean runtimes $t_i, \quad i \in \{1, ..., m\}$ depend on the number of nodes $n_i, \quad i \in \{1, ..., m\}$ polynomially, i.e. $y_i = c \cdot n_i^{\gamma_1}$ with an exponent γ_1 and a constant factor c, it follows that $log_{10}(t_i) = log_{10}(c \cdot n_i^{\gamma_1}) = log_{10}(c) + \gamma_1 \cdot log_{10}(n_i)$. So if this equation would be fulfilled, the log mean runtimes plotted against the log number of nodes should show a straight line with slope γ_1 and intercept $\gamma_0 = log_{10}(c)$. To investigate this, a linear model was used. The model equation is given below [Fahrmeir et al., 2009].

$$log_{10}(t_i) = \gamma_0 + \gamma_1 \cdot log_{10}(n_i) + \epsilon_i, \quad i \in \{1, ..., m\}$$

The log mean runtime is the dependent variable, an intercept and the log number of nodes build the design matrix. The proportion of explained variance by such a model was calculated as the explained sum of squares divided by the total sum of squares [Fahrmeir et al., 2011].

Furthermore a second, polynomial model, which includes additionally a quadratic term of the log mean runtime, was estimated. It regards the question, whether the linear term is sufficient to model the log mean runtimes depending on the log number of nodes and thus whether the mean runtimes depend on the number of nodes like described above. Its model equation is shown in the following [Fahrmeir et al., 2009].

$$log_{10}(t_i) = \gamma_0 + \gamma_1 \cdot log_{10}(n_i) + \gamma_2 \cdot (log_{10}(n_i))^2 + \epsilon_i, \quad i \in \{1, ..., m\}$$

For studying the just formulated question, it should be considered, how much much of the variance in the data was explained by the model with a linear term and whether the quadratic term in the second model was significant. As well it should be kept in mind, that in general a model including also higher polynomial terms could be able to model noise better compared to a model with a less polynomial degree [Fahrmeir et al., 2009].

The next section will explain, how the mean runtimes were compared for different adjacency structures and same matrix sizes.

3.3.2 Permutation test

For comparing the mean runtimes of different adjacency structures as described in section 3.3, permutation test were used. First their theory and after that their implementation are described here in more detail.

3.3.2.1 Theory

Permutation tests build a class of hypothesis tests, where permutations of the data are used to generate a reference distribution. The test statistic is once computed on the non-permuted data. To generate a distribution under the null-hypothesis and to decide whether to keep or reject the null-hypothesis, the data is randomly permuted multiple times. On the basis of each of these permuted datasets the test statistic is again calculated "to obtain the permutation distribution of [...] [the test statistic]" [Ernst, 2004, p. 681]. Subsequently this distribution is used for computing a p-value.

To achieve the permutation distribution and thus the exact p-value it is necessary to use all possible permutations of the data. The number of these permutations can become "very large as the samples become only moderate in size" [Ernst, 2004, p. 682]. Therefore Monte Carlo Sampling can be applied. Here the exact p-value is estimated via using a sample from the permutation distribution. As Ernst [2004] explains this estimated p-value can be calculated as the proportion of test statistics coming from permuted data sets, that are "as extreme or more extreme than the observed value" [Ernst, 2004, p. 682] of the test statistic computed on the non-permuted data. So permutations of the data and their corresponding test statistics are randomly chosen and the proportion can be calculated. Ernst [2004] argues, that a few thousand test statistics from the permutation distribution should be adequate for calculating an accurate estimate of the exact p-value. [Ernst, 2004]

A special case concerns paired samples, i.e. there exist measurements x and y of two variables X and Y on the same subjects. The null hypothesis reads, "that $F_X = F_Y$ against the alternative that the distributions differ by a location shift" [Einsporn and Habtzghi, 2013, p. 768]. Einsporn and Habtzghi [2013] describe a standard permutation test for such a scenario. Permuting means in the context of paired samples "possible interchanges of [...] [the two variables] within each of the [...] pairs" [Einsporn and Habtzghi, 2013, p. 768-769]. For n pairs there exist 2^n possibilities. As test statistic the mean difference $\overline{d} = \overline{x} - \overline{y}$ is used. This test statistic is calculated for the non-permuted data as well as for the permuted versions of the data, i.e. the interchanged versions as described above. For

Clique subgroup	Non-overlapping structure	Overlapping structure
few big equal-sized	non-ov. few big eq.-sized	ov. few big eq.-sized
many small equal-sized	non-ov. many small eq.-sized	ov. many small eq.-sized
few big different-sized	non-ov. few big diff.-sized	ov. few big diff.-sized
many small different-sized	non-ov. many small diff.-sized	ov. many small diff.-sized

Table 3: Table containing the clique subgroups and the corresponding non-overlapping and overlapping structures, that are compared in permutation tests.

large n again a sample of the permutations can be used to calculate the p-value. Einsporn and Habtzghi [2013]

The following section will introduce, how the permutation tests were applied on the mean runtime measurements of the different adjacency matrices, that were created using the structures described in section 3.2.

3.3.2.2 Implementation

As for each of the considered adjacency matrices 100 startvectors were used and for each of the startvectors the mean runtime was measured, there resulted 100 mean runtimes per adjacency matrix. The startvectors were randomly chosen as described above for each matrix size. So the mean runtimes of those adjacency matrices sharing the same size are based on the same startvectors.
For comparing the mean runtimes of adjacency matrices, which belong to the first group and have the same size but differ in their structures, the permutation test was used. Since the mean runtimes are based on the same startvectors, they were treated as a paired sample. So the permutation test was conducted as described in the previous section for a paired sample, to compare the mean runtimes of two different matrices. This was done in Python using the module permute.core [Millman et al., 2019]. For each test 10000 permutations were considered.

Of special interest was the comparison of the three characterizations, on which the matrix structures are invented in section 3.2. Permutation tests were used to properly test on differences in the mean runtimes between the different structures given the same matrix size. First two-sided tests were conducted with the null hypothesis, that the distributions of the runtimes of two different adjacency matrices are the same.
The characterization “non-overlapping against overlapping cliques” is now used to describe, which tests were conducted. Aim is comparing the mean runtimes of matrices of the same size regarding this characterization. For one matrix size there exist eight structures, four with non-overlapping cliques and four with overlapping cliques. Among these two categories there is a one-to-one correspondence of the structures regarding the other two characterizations. Hence for each matrix size four permutation tests were conducted, to compare the mean runtimes of matrices with non-overlapping and overlapping cliques in those four subgroups. The tests are listed here in table 3.

For the other two characterizations the subgroups and tests were formulated in the same fashion. Additionally the mean runtimes of the structure “non-overlapping few big equal-sized cliques” were tested against the mean runtimes of the permuted version of

this structure for testing on differences between the non-permuted and permuted time measurements.
All these two-sided tests were additionally conducted as a one-sided test. The corresponding alternative hypotheses will be described in detail in section 4.1, when the results will be displayed.

If one wants to take a test decision, a significance level has to be chosen. As the runtime data of one structure and one matrix size was used in several tests - one test for testing the characterization “non-overlapping against overlapping cliques”, one for “few big against many small cliques” and one for “equal-sized against different-sized cliques” - some adjustment for multiple testing should be chosen. In the case of structure “non-overlapping against overlapping cliques” even another test was conducted for testing on differences to the permuted version. A possibility for adjustment is offered by the Bonferroni correction [Bland and Altman, 1995].
Given one matrix size in total 26 tests were conducted, if a test on the permuted version was conducted, which was the case for the matrix sizes up to 6000. For bigger matrix sizes 24 tests were conducted. In this thesis the Bonferroni correction is used to adjust $\alpha = 0.05$ for multiple testing. The choice of the denominator for this thesis was conservative and corresponds to the number of tests conducted for one matrix size. So the significance level is $\alpha = 0.05/26$ respectively $\alpha = 0.05/24$ depending on the matrix size. Another possibility may be choosing the maximum number of tests, conducted on the same data - which were eight here.

The results of these analyses will be presented in section 4. First some more details concerning the realization of the runtime measurements are given in the next section.

3.4 Simulation for measuring the runtimes

The runtime measurements were conducted on the CoolMUC2 Infiniband cluster of the Leibniz Supercomputing Centre (LRZ). The CPU is Intel Xeon E5-2697 v3 ("Haswell") with 28 Cores and 64 GB RAM. The batch system on the LRZ is SLURM.
In section 3.2 the considered adjacency matrices were described. For every of these adjacency matrices an own job was used for measuring the runtime of the algorithm for the respective adjacency matrix. To be sure, that the runtime measurements are not biased by other processes, the order, with which these jobs are submitted, was randomly chosen. So for example not the jobs of all randomly chosen adjacency matrices were submitted one after the other followed by the random intersection graphs and then the jobs of the matrices of the different structures. In such a setting the runtime measurements could be biased just as comparative analyses based on these measurements. To avoid this, the runtime measurements were realized like explained above.
The following section now will present the results regarding runtime and effectiveness of the algorithm.

4 Results

4.1 Runtime

As first property of the algorithm the results regarding the runtime are presented. As for each adjacency matrix and startvector 10 repetitions of the runtime measurement were taken, the arithmetic mean of these 10 repetitions is considered as described in section 3.1.

4.1.1 Adjacency Structures

First in this section the runtime measurements of the adjacency matrices of the first group are considered. These adjacency matrices were created by using the eight structures described in section 3.2.
For these matrices belonging to the first group, the mean runtimes can be viewed from two perspectives. On the one hand the mean runtime can be considered for one of the eight matrix structures depending on the matrix size, i.e. the number of nodes. The other possibility is to look at one matrix size but all eight matrix structures and compare the measurements between the structures given a specific matrix size. The latter will be done in section 4.1.1.2, the former in the following section.

4.1.1.1 Runtime for different matrix sizes

The runtime in connection with the adjacency matrix sizes is considered in this section. As an example in this entire section the results of the structure “non-overlapping few big equal-sized cliques” are presented, since the results are very similar for all the structures.
There exist ten matrices with this specific structure, since only for this structure an adjacency matrix with 15000 nodes was used. As stated previously in section 3.2 the other seven structures only comprise nine different matrices with up to 10000 nodes.

Figure 1 shows the number of nodes on the x-axis and the arithmetic mean of the 10 repeated runtime measurements in seconds on the y-axis. Every point refers to one startvector of one adjacency matrix. The colours additionally visualize the number of nodes and are used in all figures hereinafter, that display results of matrices of different sizes. For figure 1 the adjacency matrices with the structure “non-overlapping few big equal-sized cliques” are used, they differ in their matrix sizes i.e. their number of nodes. As for each of the considered numbers of nodes 100 startvectors were used, these 100 points partially overlap in the figure. For the segment from 100 to 15000 nodes on the x-axis one can see, that the mean runtime increases with the number of nodes, i.e. with the increasing size of the adjacency matrix. For each of the other seven structures such a dotplot of the mean runtime measurements looks similar to figure 1 apart from the fact, that it covers only a segment from 100 to 10000 nodes on the x-axis.

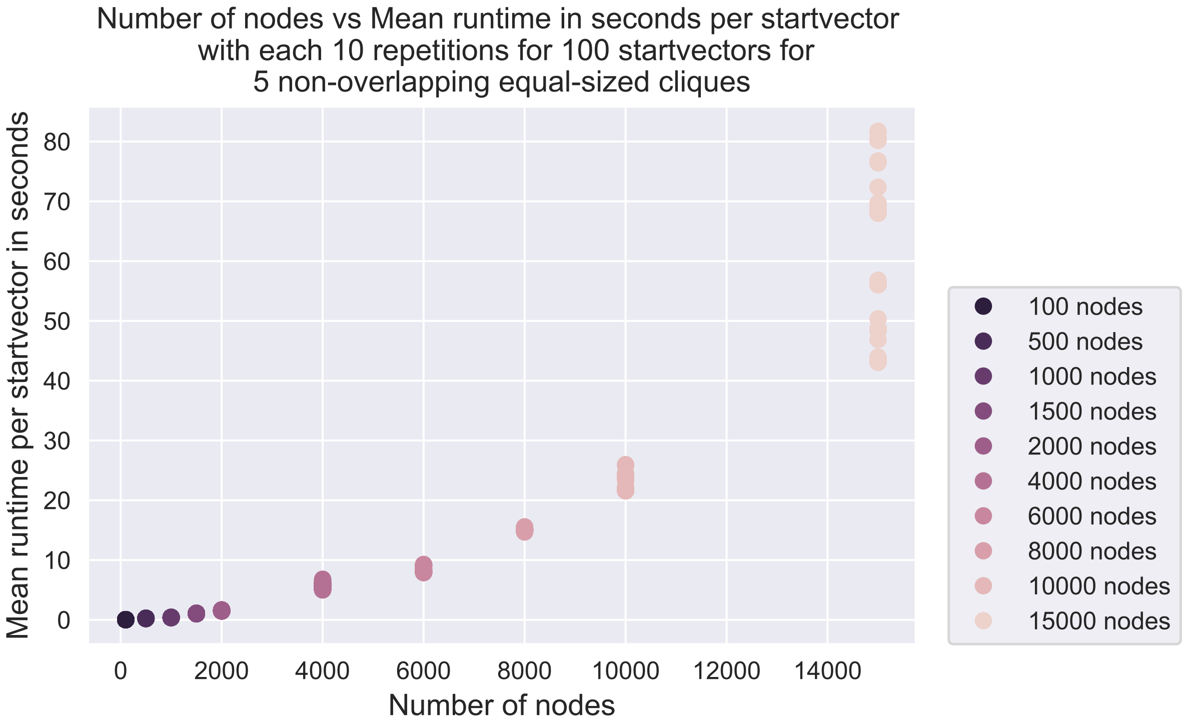

Figure 1: Dotplot of the mean runtimes in seconds of the adjacency matrices with structure "non-overlapping few big equal-sized cliques" with the number of nodes on the x-axis and the mean runtime per startvector and adjacency matrix on the y-axis. The colours represent the number of nodes.

In particular the mean runtime seems to increase non-linearly and over-proportionally with the number of nodes. To illustrate this, figure 2 shows the boxplots of the mean runtimes shown in figure 1 divided by the corresponding number of nodes. This can be interpreted as the mean runtime per node. The boxes are obviously not on the same horizontal line, i.e. the mean runtime per node is not the same for the different matrix sizes. As the mean runtime per node particularly increases with the number of nodes, the mean runtime increases over-proportionally with the number of nodes. For the other structures the boxes are as well not on the same horizontal line.

To take a closer look at the non-linearity of the increase of the mean runtime for the matrices with structure "non-overlapping few big equal-sized cliques", figure 3 shows a dotplot with the log mean runtimes on the y-axis and the log number of nodes on the x-axis. Additionally a fitted regression line is drawn. The corresponding linear model is described in the next paragraph. Under the assumption, that the mean runtimes depend on the number of nodes polynomially, the points in figure 3 should show a straight line as explained in section 3.3.1. To investigate this, a linear model is used with the log mean runtime as the dependent variable, an intercept and the log number of nodes building the design matrix. The resulting fitted regression line is drawn in figure 3, its slope corresponds to the coefficient estimate in table 4 and amounts to 1.4756, its p-value is 0.0. The proportion of explained variance by this model with a linear term is 98.5%.

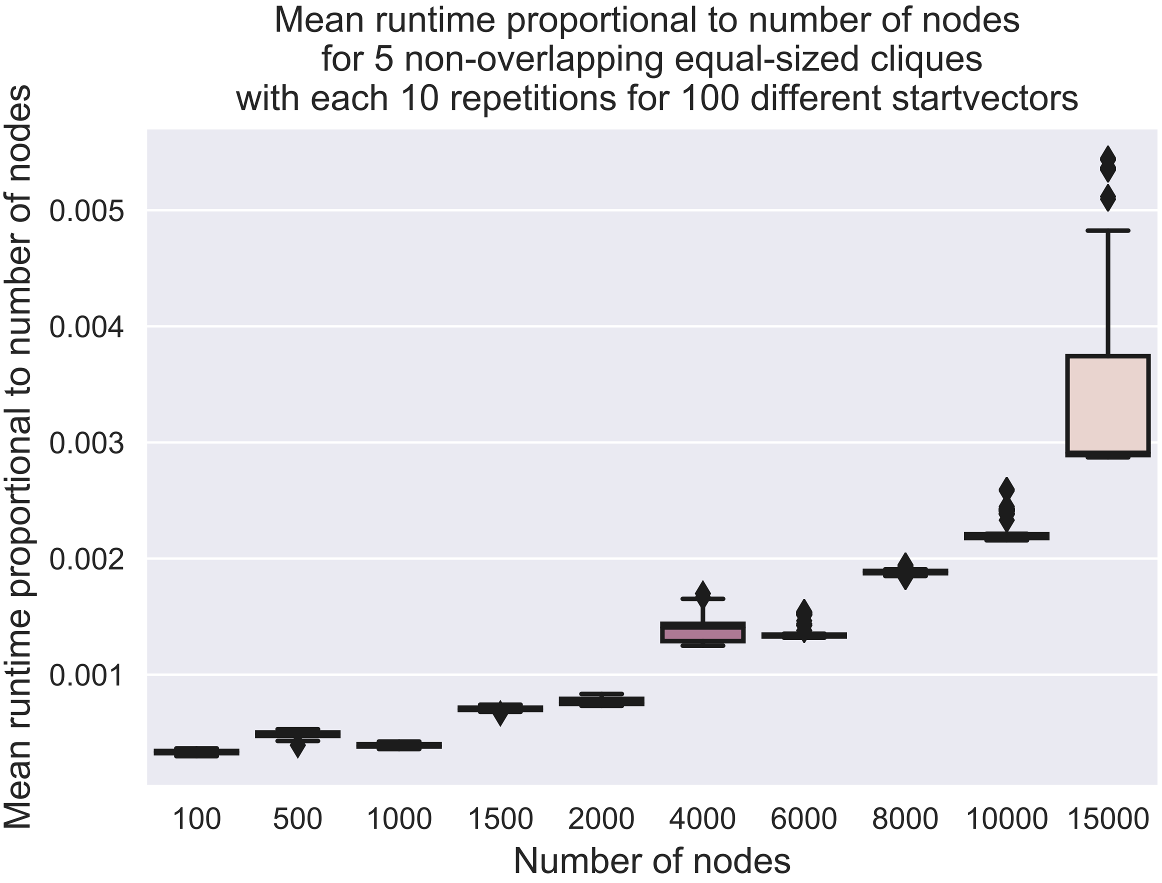

Figure 2: Boxplot of the mean runtimes in seconds divided by the number of nodes of the adjacency matrices with structure “non-overlapping few big equal-sized cliques”. The whiskers end on the last point less than the third quartile plus 1.5 times the interquartile range respectively the last point greater than the first quartile minus 1.5 times the interquartile range. The colours represent the number of nodes.

Table 4 shows the estimated coefficients and their corresponding p-values for this linear and a second, polynomial model, which includes additionally a squared term of the log number of nodes. This squared term has as well a p-value of 0.0. Regarding the question, whether the linear term is sufficient to model the log mean runtimes depending on the log number of nodes and thus whether the mean runtimes depend polynomially on the number of nodes, the following points should be considered. First the model with a linear term explains much of the variance in the data. Second a model including also a polynomial term of order two could be able to model noise better than a model without. On the whole for being able to make a statement about the above formulated question, it will be helpful to investigate this further with more runtime measurements for even bigger matrices than considered in this thesis.

Model	Covariable	Coefficient	P-value
linear model	intercept	-4.6032	0.0
	log_{10}(number of nodes)	1.4756	0.0
polynomial model	intercept	-2.5843	0.0
	log_{10}(number of nodes)	0.1251	0.0
	$(log_{10}(\text{number of nodes}))^2$	0.2153	0.0

Table 4: Table containing the coefficients and their p-values for two models fitted for the structure “non-overlapping few big equal-sized cliques” with the log mean runtime to the basis 10 as dependent variable.

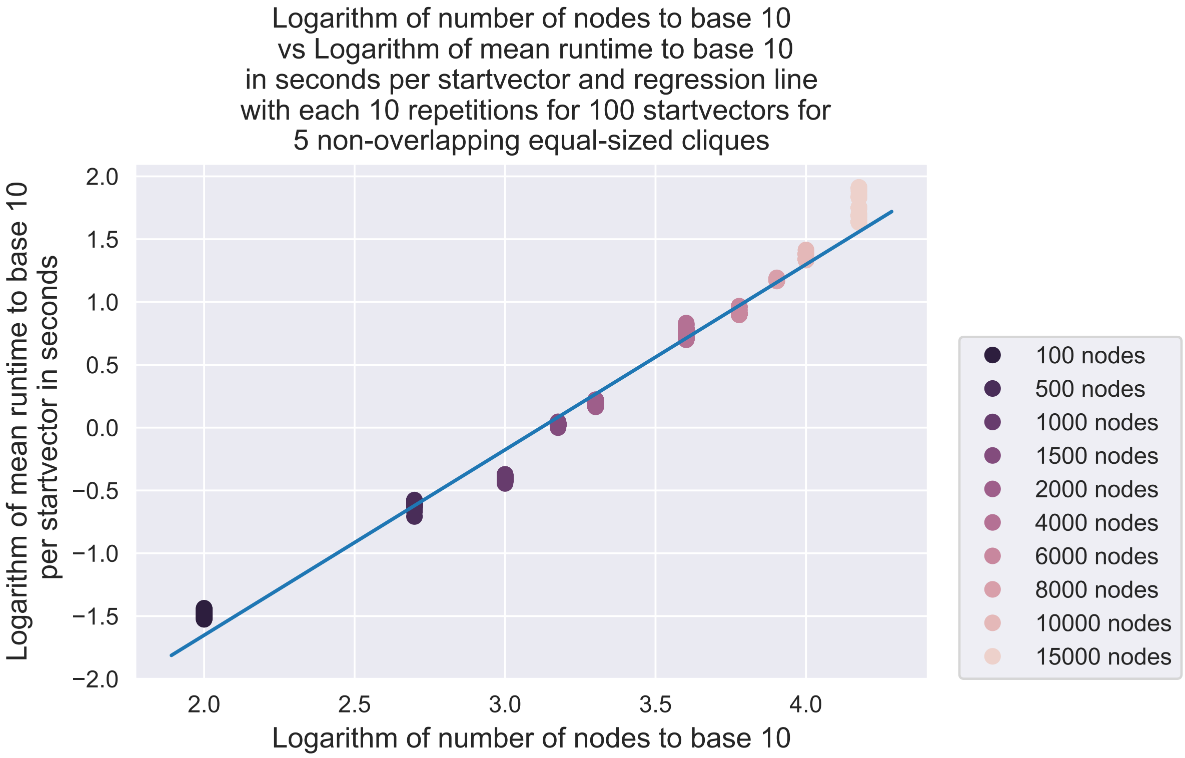

Figure 3: Dotplot of the log mean runtimes in seconds to the basis 10 of the adjacency matrices with structure “non-overlapping few big equal-sized cliques” with the log number of nodes to the basis 10 on the x-axis and the log mean runtime to the basis 10 per startvector and adjacency matrix on the y-axis. A regression line of the linear model fitted on this data is drawn. The colours represent the number of nodes.

After having studied the mean runtimes of the first group of adjacency matrices depending on the matrix size in this section, the following one will compare these mean runtimes between the different structures given the same matrix size.

4.1.1.2 Runtime for different structures

The differences of the mean runtimes between the structures are compared in this section given a specific matrix size. The smallest considered matrix size with 100 nodes is used for illustrating the mean runtimes. Figure 4 contains the boxplots of the mean runtimes in seconds for the matrix size 100 for all eight structures. Based on figure 4 the eight structures can be compared regarding the three characterizations, that were used to create the structures as described in section 3.2.

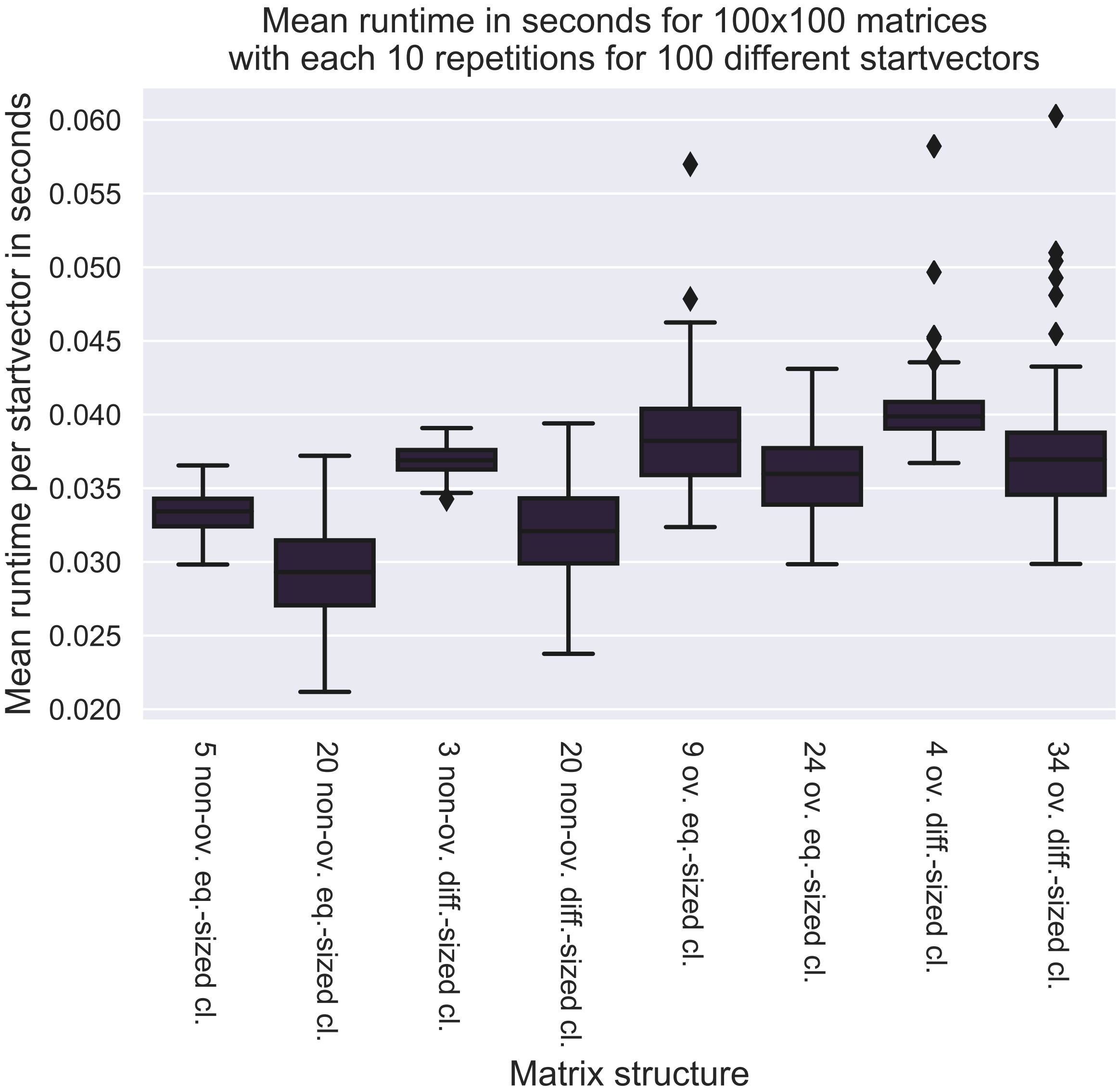

Figure 4: Boxplot of the mean runtimes in seconds for all eight matrix structures and 100 nodes. The whiskers end on the last point less than the third quartile plus 1.5 times the interquartile range respectively the last point greater than the first quartile minus 1.5 times the interquartile range.

For comparing the two categories of the first characterization - non-overlapping and overlapping cliques - e.g. “non-overlapping few big equal-sized cliques” among the non-overlapping structures is compared to its corresponding overlapping structure “overlapping few big equal-sized cliques”. Regarding a box of a non-overlapping structure, this box is shifted downwards compared to the respective box of the corresponding overlapping structure. So the structures with non-overlapping cliques seem to have on average slightly shorter mean runtimes compared to those with overlapping cliques.
Another recognizable pattern is given by comparing via the second characterization the structures with few big cliques to those structures with many small cliques in the same fashion as explained above for the first characterization. Hence the structure “non-overlapping few big equal-sized cliques” now is compared to “non-overlapping many small equal-sized cliques”. In figure 4 it seems, that the structures with few big cliques have on average higher mean runtimes compared to the structures with many small cliques, as the boxes of structures with few big cliques are shifted upwards compared to those with many small cliques.
The third option for comparing the matrix structures is looking at differences in the mean runtimes between structures with equal-sized cliques and such with different-sized cliques. Therefore e.g. “non-overlapping few big equal-sized cliques” is compared to the structure “non-overlapping few big different-sized cliques”. Here it seems referring to the boxes in figure 4, that structures with equal-sized cliques have on average lower mean runtimes than those with different-sized cliques.
So for all three characterizations, that were used to create the structures of the adjacency matrices, the just described patterns in the mean runtimes can be suspected after looking at figure 4. These patterns partly show up in the corresponding figures for the bigger matrix sizes. But there are also many exceptions and cases, where the patterns seem to be the other way round than in figure 4. In addition the figures of the other matrix sizes are much more disorganized than figure 4. That means, there are greater differences in the interquartile range as well as the length of the whiskers as well as the existence of outliers between the boxes than in figure 4.

Permutation tests are used to properly test the mean runtimes of two different matrix structures given the same matrix size. First two-sided tests were conducted as described in section 3.3.2. The results are displayed exemplary for the matrix size 100 in table 5. Correcting the significance level by using the Bonferroni correction as described in section 3.3.2 all the tests in table 5 are significant. In the first test in table 5 for example the mean runtimes of adjacency matrices with few big equal-sized cliques are considered. According to this test, the differences between the mean runtimes of the non-overlapping cliques and the overlapping cliques are significantly different from zero for adjacency matrices with few big equal-sized cliques and 100 nodes.

Now the test results for all matrix sizes are considered. For testing on differences in mean runtime between non-overlapping and overlapping cliques, for each matrix size and each of the four corresponding clique subgroups given in the table 5 a two-sided test was conducted. All these tests were significant using the adjusted significance level using the Bonferroni correction. The differences in mean runtime between few big and many small cliques as well as the differences between equal-sized and different sized cliques are

Matrix size	Kind of test	Clique subgroup	P-value
100 nodes	non-overlapping vs overlapping	few big equal-sized	$1.9 \cdot 10^{-5}$
		many small equal-sized	$1.9 \cdot 10^{-5}$
		few big different-sized	$1.9 \cdot 10^{-5}$
		many small different-sized	$1.9 \cdot 10^{-5}$
	few big vs many small	non-overlapping equal-sized	$1.9 \cdot 10^{-5}$
		non-overlapping different-sized	$1.9 \cdot 10^{-5}$
		overlapping equal-sized	$1.9 \cdot 10^{-5}$
		overlapping different-sized	$1.9 \cdot 10^{-5}$
	equal-sized vs different-sized	non-overlapping few big	$1.9 \cdot 10^{-5}$
		non-overlapping many small	$1.9 \cdot 10^{-5}$
		overlapping few big	$1.9 \cdot 10^{-5}$
		overlapping many small	$1.9 \cdot 10^{-5}$

Table 5: Table containing the p-values of the two-sided permutation tests conducted on the mean runtimes of matrices of all eight adjacency structures and the number of nodes 100.

significant in all matrix sizes and all respective clique subgroups, too.

As the patterns suspected in figure 4 are quite regularly compared to patterns, one would formulate looking at the more disorganized figures of the bigger matrix sizes, the patterns of figure 4 were taken as alternative hypotheses of one-sided permutation tests. The first hypothesis taken from figure 4 concerns the comparison of the mean runtimes of adjacency matrices containing non-overlapping and overlapping cliques. For each matrix size and each of the four corresponding clique subgroups a one-sided test was conducted using the alternative hypothesis, that matrices containing non-overlapping cliques have on average shorter mean runtimes compared to matrices with overlapping cliques. The results of all these tests are displayed in table 9 in the appendix. All these one-sided tests have a p-value of $9.9 \cdot 10^{-6}$. Thus the tests are significant using the corrected significance level. So in every of the four clique subgroups and all matrix sizes, the null hypothesis can be rejected in favor of the alternative, which is based on a pattern of figure 4.

The second hypothesis taken from figure 4 covers the comparison of the mean runtimes of adjacency matrices with few big and many small cliques. The one-sided test was applied on each matrix size and each of the four corresponding clique subgroups. The alternative hypothesis reads, that matrices with few big cliques have on average higher mean runtimes than those with many small matrices. The results of all these tests are displayed in table 10 in the appendix. For bigger matrix sizes than 100 at least two clique subgroups have a test result with a p-value of 1 except for the matrix size 6000. So the null hypothesis cannot be rejected in at least two clique subgroups of most of the matrix sizes.

The third alternative hypothesis is that matrices with equal-sized cliques have on average shorter mean runtimes than matrices containing different-sized cliques. A one-sided test was applied for every matrix size and every corresponding clique subgroup, the results are shown in table 11 in the appendix. Except for the matrix sizes 6000 and 8000 every matrix size bigger than 100 has at least in two of the four subgroups a test result with a p-value of 1. So the null hypothesis cannot be rejected for most of the matrix sizes in at least two clique subgroups.

Summing up the results of the one-sided tests, that use the three alternative hypotheses based on the patterns found in figure 4, only the first null hypothesis regarding the comparison of non-overlapping and overlapping cliques can be rejected in all matrix sizes and all subgroups, if the adjusted significance level using the Bonferroni correction is applied. The other two hypotheses are not significant in all matrix sizes and all subgroups.

In this section all eight structures were compared given the same matrix size. For one of the eight structures also permuted versions of the adjacency matrices were created. The runtimes of these permuted matrices will be compared to the runtimes of the corresponding non-permuted matrices in the following section.

4.1.1.3 Runtime for permuted adjacency matrices

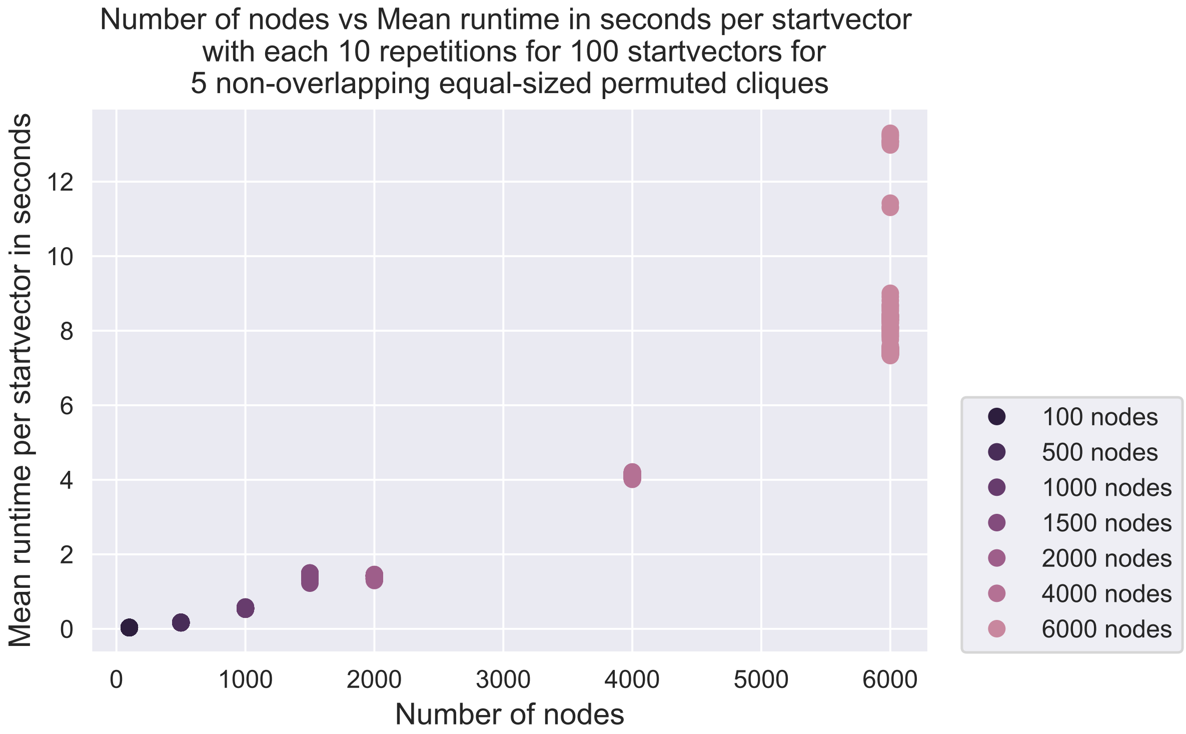

Figure 5: Dotplot of the mean runtimes in seconds of the permuted adjacency matrices with structure “non-overlapping few big equal-sized cliques” with the number of nodes on the x-axis and the mean runtime per startvector and adjacency matrix on the y-axis. The colours represent the number of nodes.

For investigating the impact of more disorganized matrices on runtime, the adjacency matrices of the structure “non-overlapping few big equal-sized cliques” were randomly permuted.

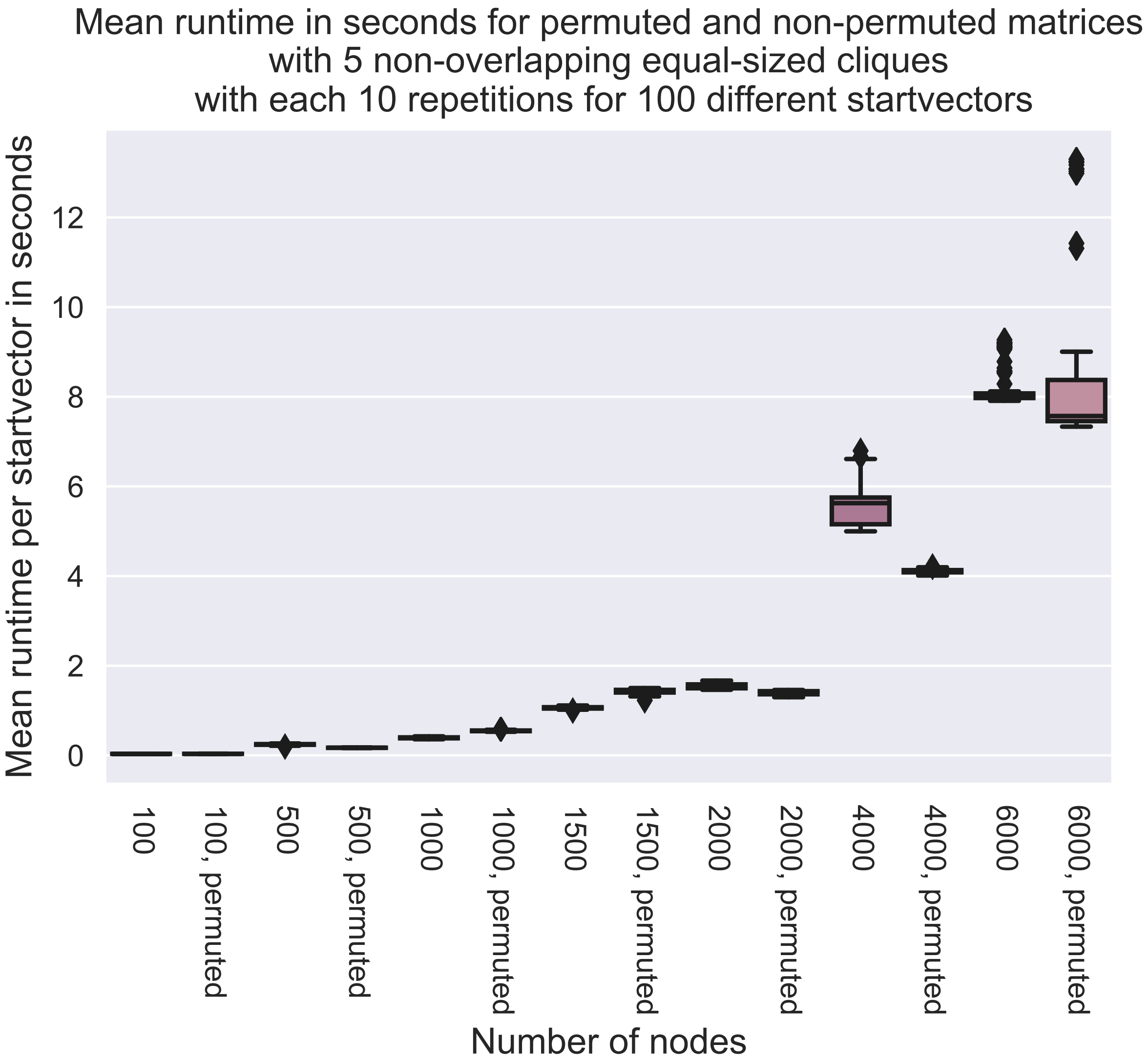

Figure 6: Boxplot of the mean runtimes in seconds of the adjacency matrices with structure “non-overlapping few big equal-sized cliques” next to the boxplot of the permuted adjacency matrices of the structure “non-overlapping few big equal-sized cliques”. The whiskers end on the last point less than the third quartile plus 1.5 times the interquartile range respectively the last point greater than the first quartile minus 1.5 times the interquartile range. The colours represent the number of nodes.

The dotplot of the mean runtimes of these permuted matrices in figure 5 is the counterpart of figure 1 described in 4.1.1.1. The curve in figure 5 looks very similar to the one of the corresponding block-wise matrices in figure 1. The mean runtimes seem to increase non-linearly with increasing matrix size.

As in section 4.1.1.1 two models were used for investigating the non-linearity of this

Model	Covariable	Coefficient	P-value
linear model	intercept	-4.2668	0.0
	log_{10}(number of nodes)	1.3551	0.0
polynomial model	intercept	-2.9305	0.0
	log_{10}(number of nodes)	0.3901	0.0
	$(log_{10}(\text{number of nodes}))^2$	0.1671	0.0

Table 6: Table containing the coefficients and their p-values for two models fitted for the permuted version of the structure "non-overlapping few big equal-sized cliques" with the log mean runtime to the basis 10 as dependent variable.

increase. The corresponding plot of the log mean runtimes and log mean number of nodes with the fitted regression line of the linear model can be found in the appendix in figure 23. The proportion of explained variance of the linear model is high with 98.33% - similar as in section 4.1.1.1 for the non-permuted adjacency matrices. The quadratic term in the second, polynomial model has likewise a p-value of 0.0. To be able to make a statement about the sufficiency of a linear term for modelling the log mean runtimes depending on the log nodes, additional bigger matrix sizes should be considered in further investigation as recommended for the non-permuted matrices of structure "non-overlapping few big equal-sized cliques".

For comparing the mean runtimes of permuted and non-permuted matrices, figure 6 shows for all matrix sizes up to 6000 the boxplots of the mean runtimes of the non-permuted and the permuted matrices of the structure "non-overlapping few big equal-sized cliques". Next to each other the boxes of the same matrix size are drawn. The boxes of the permuted matrices are shifted compared to the boxes of the respective matrix size of the non-permuted matrices - but not in a regular way. Some boxes of the permuted matrices are shifted upwards, some downwards.

Two-sided permutation tests were used for comparing the mean runtimes of the non-permuted and the permuted adjacency matrices for each matrix size. The p-value is $1.9998 \cdot 10^{-5}$ for every tested matrix size. Additionally one-sided tests were conducted for studying the alternative hypothesis, that non-permuted matrices have shorter mean runtimes than permuted matrices. The results are shown in table 7. The null hypothesis of this test cannot be rejected for the matrix sizes 500, 2000 and 4000, since the corresponding p-values are 1.

4.1.2 Random graphs

So far the results concerning the runtime of the adjacency matrices of the first group, which are based on eight structures, were presented. In this section the runtime of the second group of considered adjacency matrices, created on random graphs, are demonstrated.

Matrix size	Kind of test	Clique subgroup	P-value
100 nodes	non-permuted vs permuted	non-overlapping few big equal-sized	$9.9 \cdot 10^{-6}$
500 nodes			1.0
1000 nodes			$9.9 \cdot 10^{-6}$
1500 nodes			$9.9 \cdot 10^{-6}$
2000 nodes			1.0
4000 nodes			1.0
6000 nodes			$9.9 \cdot 10^{-6}$

Table 7: Table containing the p-values of the one-sided permutation tests conducted on the mean runtimes of the permuted and non-permuted matrices of structure “non-overlapping few big equal-sized cliques”.

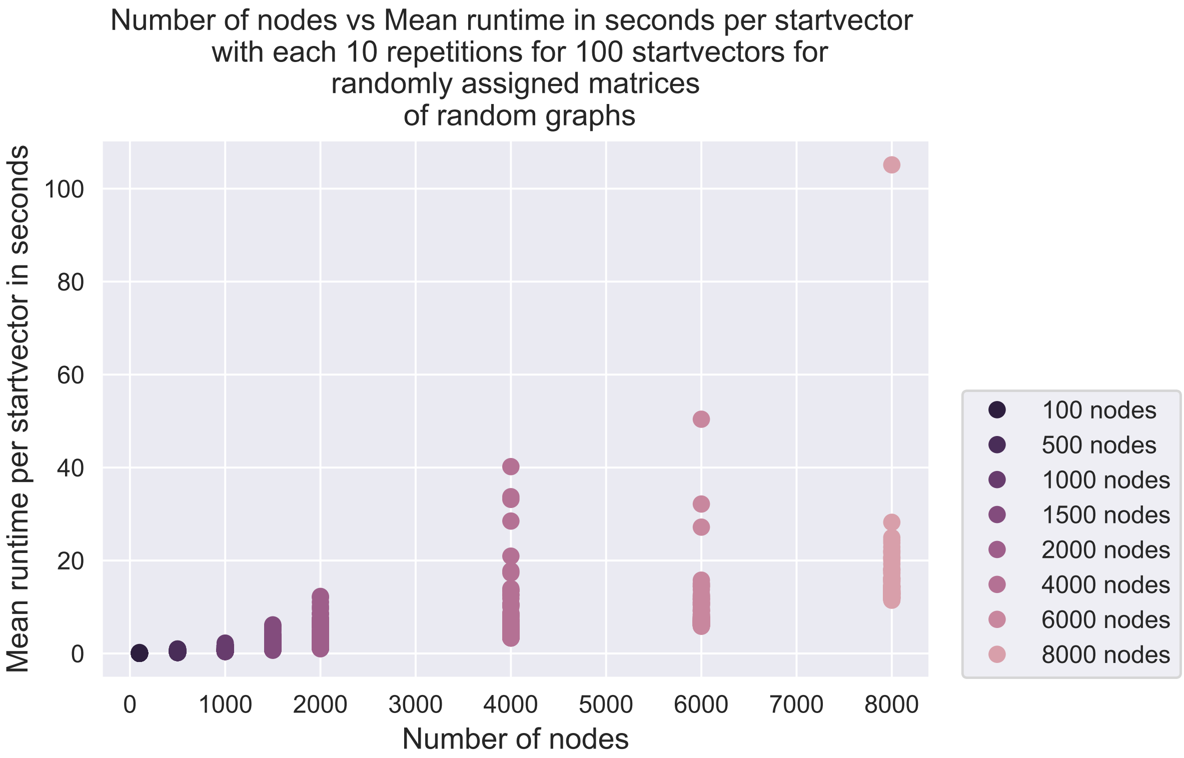

Figure 7: Dotplot of the mean runtimes in seconds of adjacency matrices of random graphs with the number of nodes on the x-axis and the mean runtime per startvector and adjacency matrix on the y-axis. The colours represent the number of nodes.

Figure 7 shows, that the mean runtimes of matrices of random graphs increase with increasing matrix size. It is difficult to tell from the figure, whether the increase is over-proportional in the number of nodes or not. Looking at the mean runtimes per node in figure 8 does not help much either. The boxes are not much shifted compared to each other. It cannot be told from figure 8, whether the mean runtimes per node differ for varying matrix sizes.

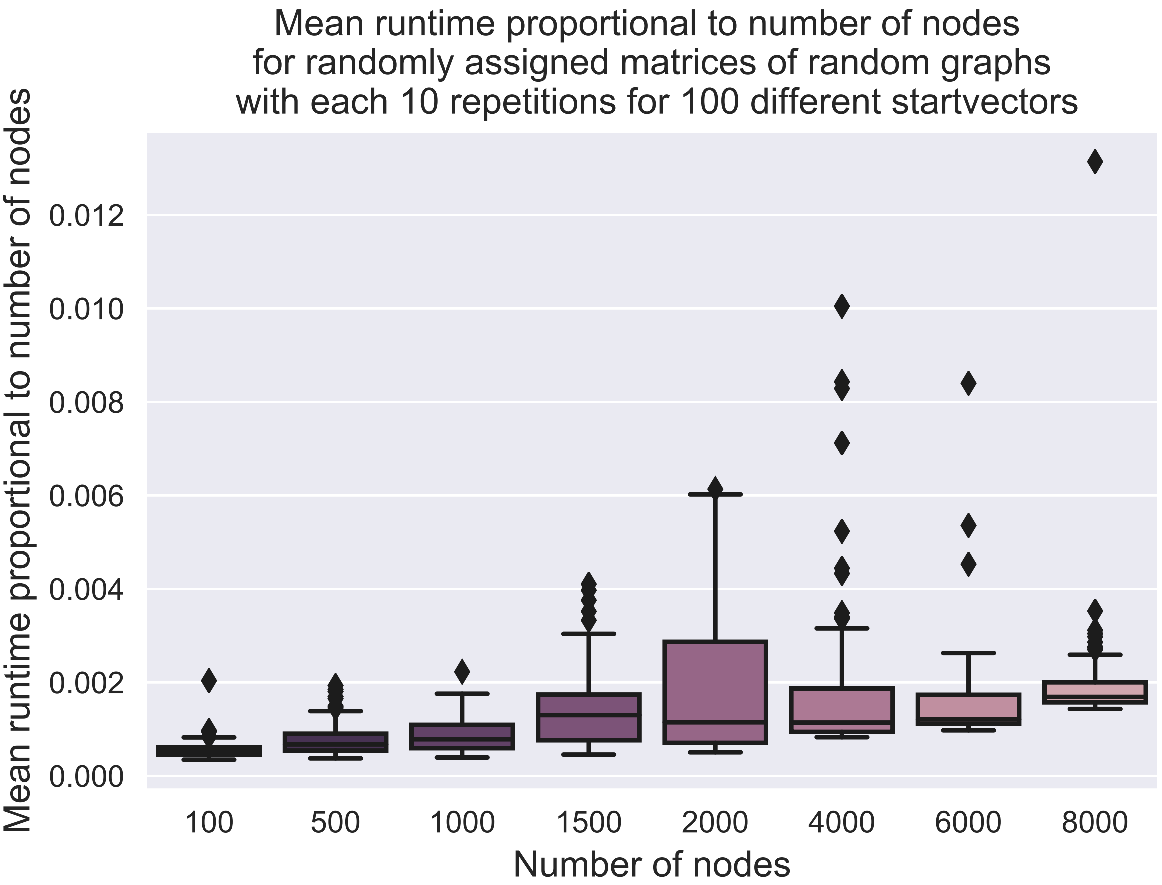

Figure 8: Boxplot of the mean runtimes in seconds divided by the number of nodes of adjacency matrices of random graphs. The whiskers end on the last point less than the third quartile plus 1.5 times the interquartile range respectively the last point greater than the first quartile minus 1.5 times the interquartile range. The colours represent the number of nodes.

For the matrices of the random graphs as well the log mean runtimes are considered. Figure 9 shows on the y-axis the log mean runtimes against the log number of nodes on the x-axis. The log mean runtimes have higher variation than the ones of the previously considered matrices, which belong to the first group. Additionally the fitted regression line of a linear model with a linear term is drawn in figure 9. The proportion of explained variance by this model amounts to 92.80%. The quadratic term of the second fitted model has a p-value of 0.341. For a significance level of $\alpha = 0.05$ the coefficient of the quadratic term is not significantly different from zero. Hence it seems, that it could be sufficient to model the log mean runtimes of randomly filled adjacency matrices with a linear term of the log number of nodes for the interval of 100 to 8000 nodes. The coefficient of the linear model, whose results are displayed in table 8, amounts to 1.2795, i.e. $log_{10}(t_i) = -3.8505 + 1.2795 \cdot log_{10}(n_i)$. This means corresponding to section 3.3.1, that the exponent γ_1 of the relationship between the mean runtimes and the number of nodes is estimated as 1.2795 for a segment of nodes from 100 to 8000. Apart from these results of the model estimations, for the randomly filled adjacency matrices as well bigger

Model	Covariable	Coefficient	P-value
linear model	intercept log_{10}(number of nodes)	-3.8505 1.2795	0.0 0.0
polynomial model	intercept log_{10}(number of nodes) $(log_{10}$(number of nodes)$)^2$	-3.6862 1.1642 0.0193	0.0 0.0 0.341

Table 8: Table containing the coefficients and their p-values for two models fitted for adjacency matrices of random graphs with the log mean runtime to the basis 10 as dependent variable.

matrix sizes should be considered in future research, for investigating the non-linearity of the mean runtimes depending on the number of nodes in more detail and for even bigger matrix sizes than 8000.

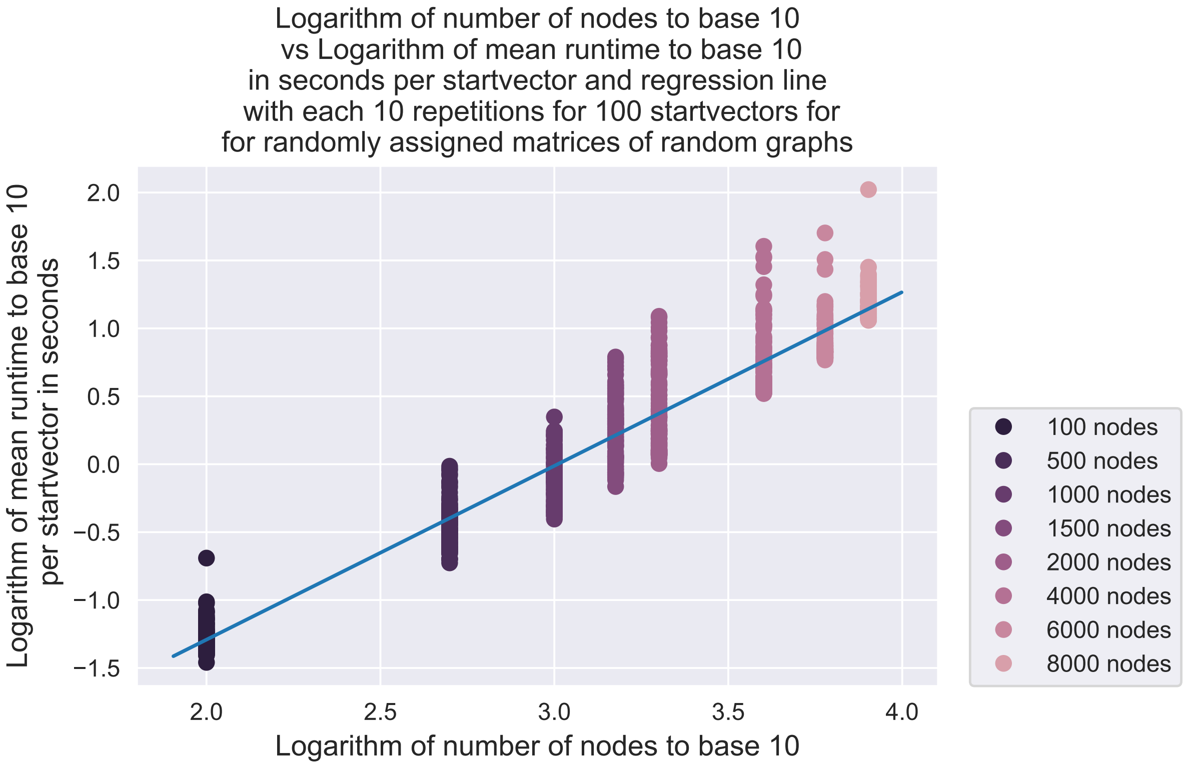

Figure 9: Dotplot of the log mean runtimes in seconds to the basis 10 of adjacency matrices of random graphs with the log number of nodes to the basis 10 on the x-axis and the log mean runtime to the basis 10 per startvector and adjacency matrix on the y-axis. A regression line of the linear model fitted on this data is drawn. The colours represent the number of nodes.

4.1.3 Random intersection graphs

The third group of considered adjacency matrices to investigate runtime and effectiveness of the algorithm comprises adjacency matrices created by using the concept of random intersection graphs. Their runtime will be studied in this section.

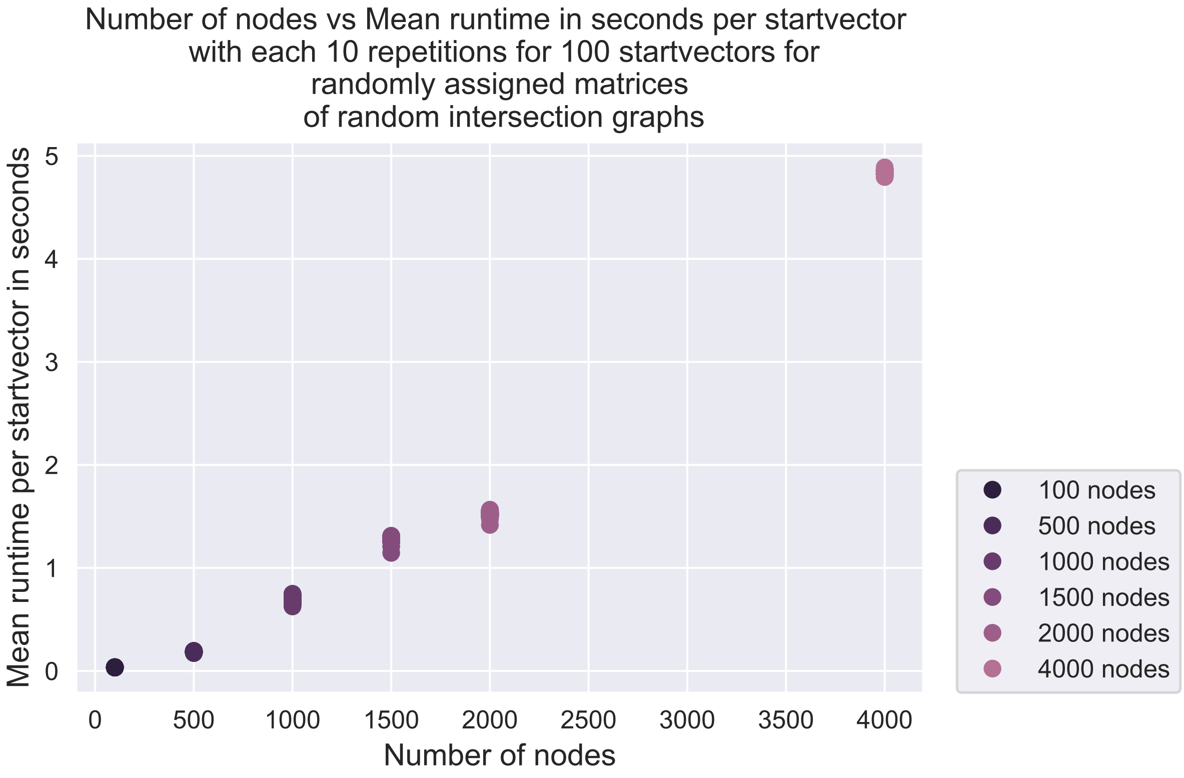

Figure 10: Dotplot of the mean runtimes in seconds of adjacency matrices of random intersection graphs with the number of nodes on the x-axis and the mean runtime per startvector and adjacency matrix on the y-axis. The colours represent the number of nodes.

Figure 10 shows the mean runtimes for such adjacency matrices. For the considered segment of 100 to 4000 nodes the mean runtime is increasing with the number of nodes. Figure 11 shows on the y-axis the log mean runtimes against the log number of nodes on the x-axis. For studying, whether the log mean runtimes could depend linearly on the log number of nodes, again a linear model with a linear term was fitted. The corresponding fitted regression line is drawn in figure 11, the proportion of explained variance of this model with a linear term amounts to 98.80%. The quadratic term of a second, polynomial model has a p-value of 0.0. The coefficients and p-values of both models are displayed in table 12 in the appendix. Runtime measurements on bigger matrix sizes could help for further investigating the non-linearity of the mean runtimes dependent on the number of nodes.

For all three groups of considered adjacency matrices the runtime of the algorithm was examined in this completed section 4.1. The effectiveness will now be studied in the following section.

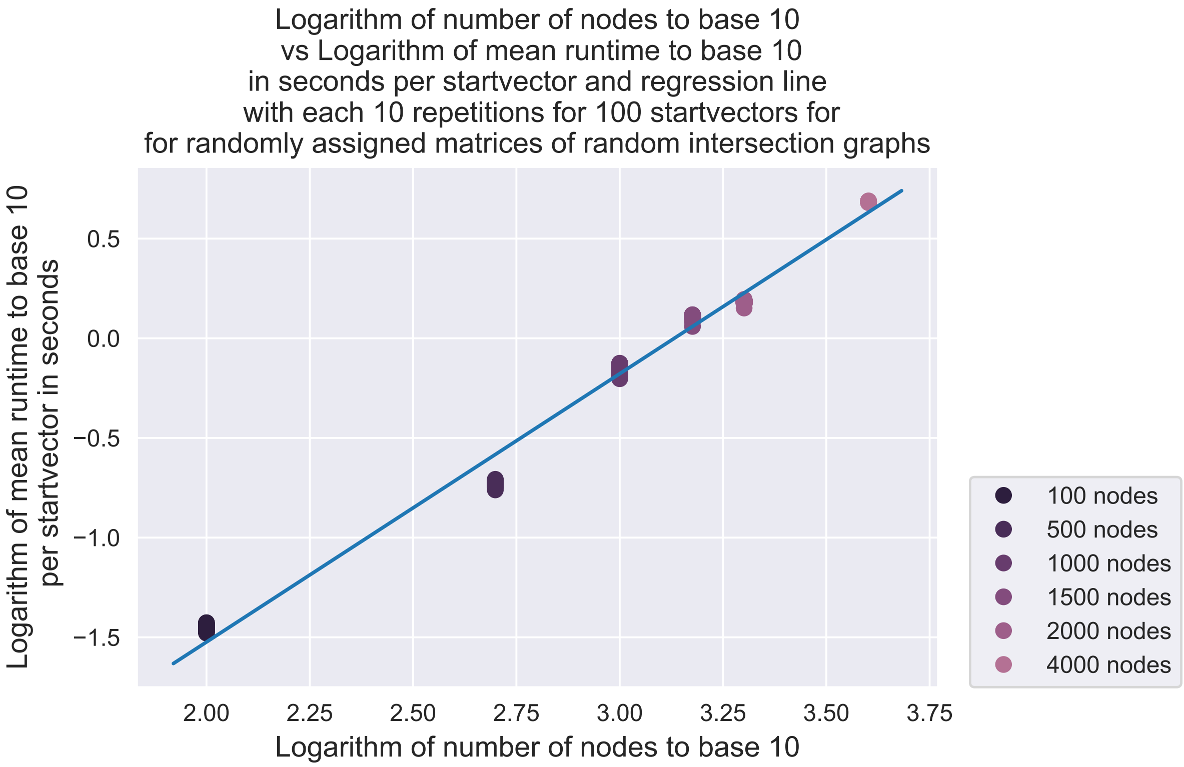

Figure 11: Dotplot of the log mean runtimes in seconds to the basis 10 of adjacency matrices of random intersection graphs with the log number of nodes to the basis 10 on the x-axis and the log mean runtime to the basis 10 per startvector and adjacency matrix on the y-axis. A regression line of the linear model fitted on this data is drawn. The colours represent the number of nodes.

4.2 Effectiveness

Not only runtime is an interesting property of the algorithm. Also the detected different cliques are informative and can tell more about the effectiveness of the algorithm.

4.2.1 Adjacency Structures

Again the adjacency matrices of the first group, that were created by using the eight matrix structures coming from three binary characterizations, are considered first in this section. As for studying the runtime, two perspectives can be used for examining the effectiveness. For one of the eight matrix structures effectiveness can be investigated depending on the matrix size or for one matrix size compared between the eight matrix structures. The latter is done in section 4.2.1.2, the former in the following section.

4.2.1.1 Effectiveness for different matrix sizes

The structure "non-overlapping few big equal-sized cliques" is chosen for illustrating the proportion of found different cliques among all existing cliques after 100 startvectors. For every adjacency matrix created by the structures the number of existing cliques in the matrix is known by design. The proportion of found different cliques was calculated as the number of different cliques found by the algorithm after 100 startvectors divided by the number of existing cliques in the adjacency matrix. All matrices of the structure "non-overlapping few big equal-sized cliques" contain five cliques. Figure 12 shows, that for all these matrices all five cliques are found after 100 startvectors.

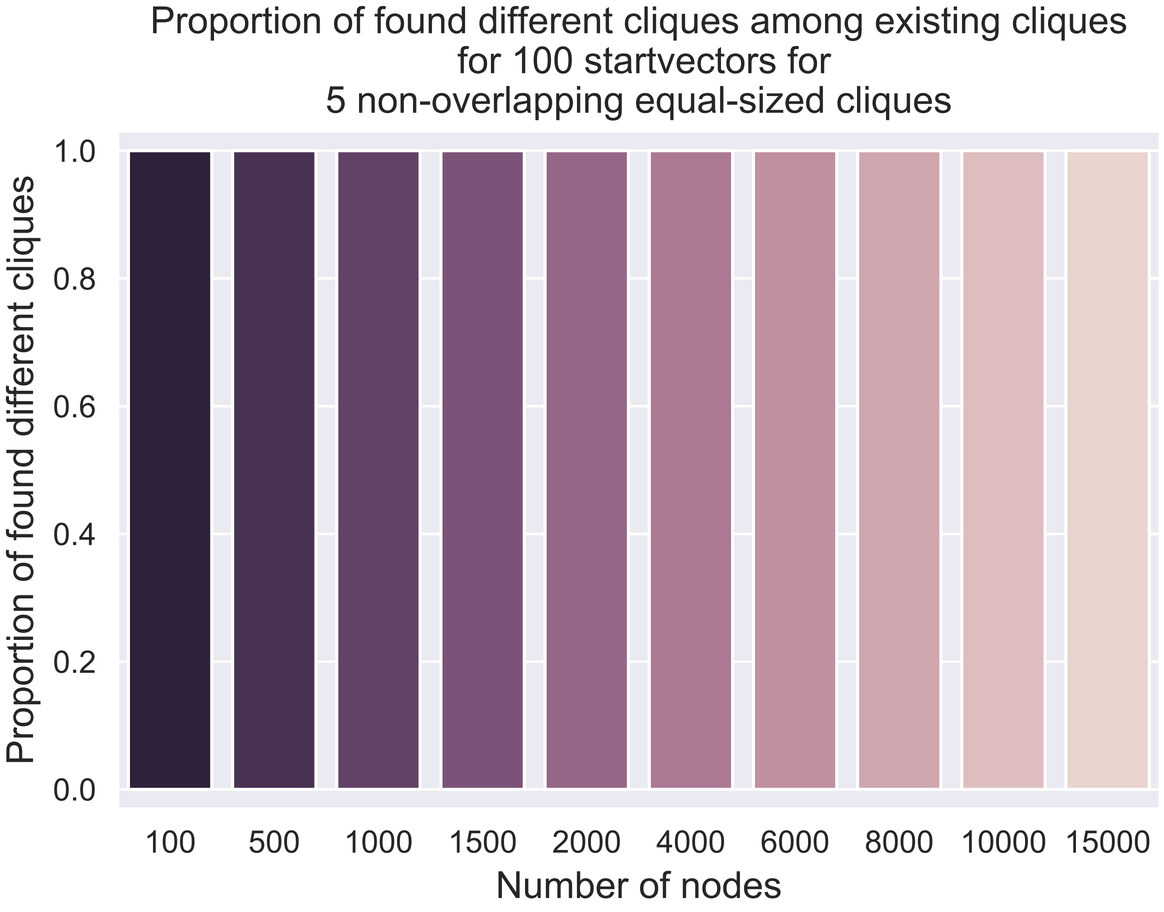

Figure 12: Barplot of the proportion of found different cliques among the five existing cliques after 100 startvectors in adjacency matrices of the structure "non-overlapping few big equal-sized cliques". The colours represent the number of nodes.

The just considered structure and "non-overlapping many small equal-sized cliques" are the only two structures, for which in all matrix sizes 100% of the existing cliques were found after 100 startvectors. These two structures have both non-overlapping equal-sized cliques.

For all other structures in none of the matrix sizes 100% of the existing cliques were found after 100 startvectors. Their proportions of found different cliques among the existing cliques after 100 startvectors in particular decrease with increasing matrix size. As an example figure 24 shows this for the structure “overlapping few big equal-sized cliques” in the appendix. The highest proportion is reached for the smallest matrix size. With increasing matrix size the proportion diminishes.

Another interesting point to investigate is, when the different cliques are found for the first time, i.e. at which startvector. Therefore the proportion of found different cliques was calculated after each startvector like explained in section 3.3. In figure 13 these proportions are drawn as points. Whenever the proportion goes up, this means a new clique was found. The number of used startvectors up to this point is drawn on the x-axis and the respective proportion on the y-axis. The different colours again stand for the different matrix sizes.

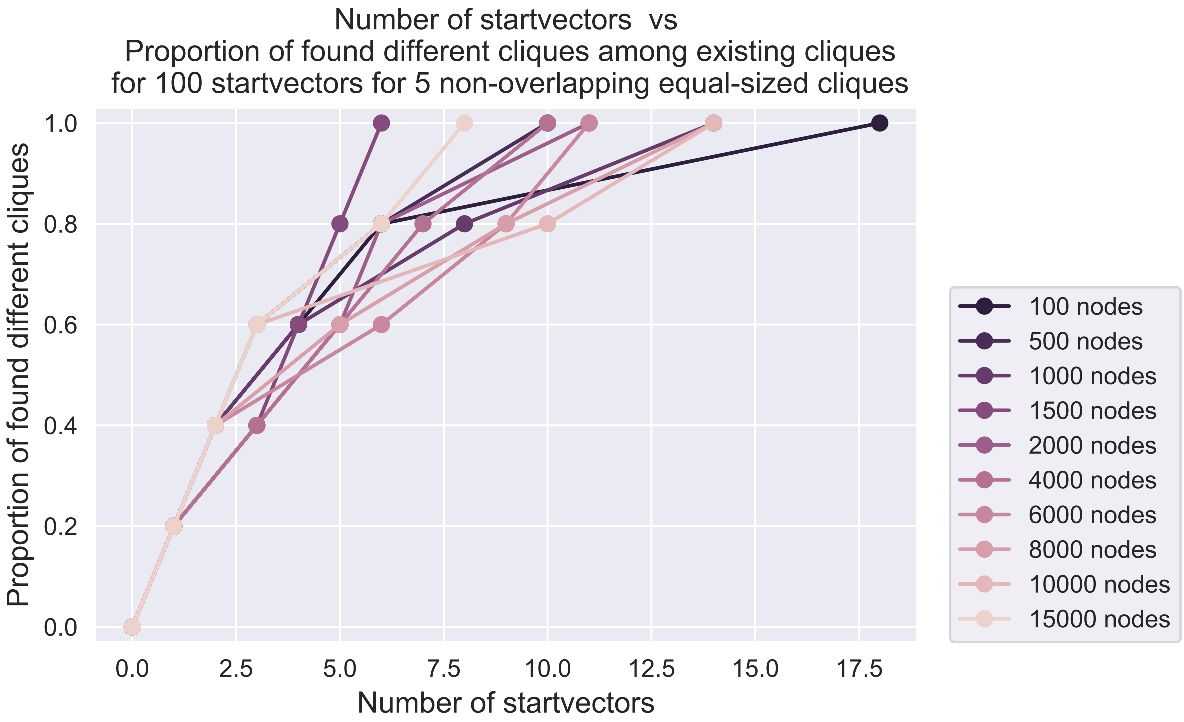

Figure 13: Dotplot of development of proportions of found different cliques among the existing cliques for the adjacency matrices of the structure “non-overlapping few big equal-sized cliques”. The proportion of found different cliques is depicted on the y-axis, the number of the startvectors on the x-axis. Every dot represents a change in the proportion because of a new found clique in an adjacency matrix. The colours represent the number of nodes.

It seems in figure 13, that there is no clear tendency between the curves of the different matrix sizes. One could maybe say, that the curves of the bigger matrices seem to be slightly shifted to the right on the x-axis compared to the curves of the smaller matrices. This would mean, that bigger matrices need slightly more startvectors to reach a certain proportion of found different cliques. But there are also exceptions like for example the curve of the matrix with 100 nodes, which needs most startvectors of all matrices of the structure “non-overlapping few big equal-sized cliques” to find all five existing cliques. Overall, one can say that the differences between the curves seem to be small and without a clear tendency concerning the matrix sizes. All matrices reach 100% of the existing cliques between the sixth and 18^{th} startvector. This appears as the corresponding curves stop ahead of the respective mark on the x-axis. This means, that the following startvectors only find cliques, that were already detected previously.

The other structures show similar figures and seem to have as well no clear tendency of a shift on the x-axis concerning the matrix size. As 100% of the existing cliques is only reached for matrices of the two structures with non-overlapping equal-sized cliques, these curves of the corresponding figures of the other structures don’t reach up to 1 on the y-axis. Figure 25 displays this exemplary for the structure “overlapping many small equal-sized cliques” in the appendix.

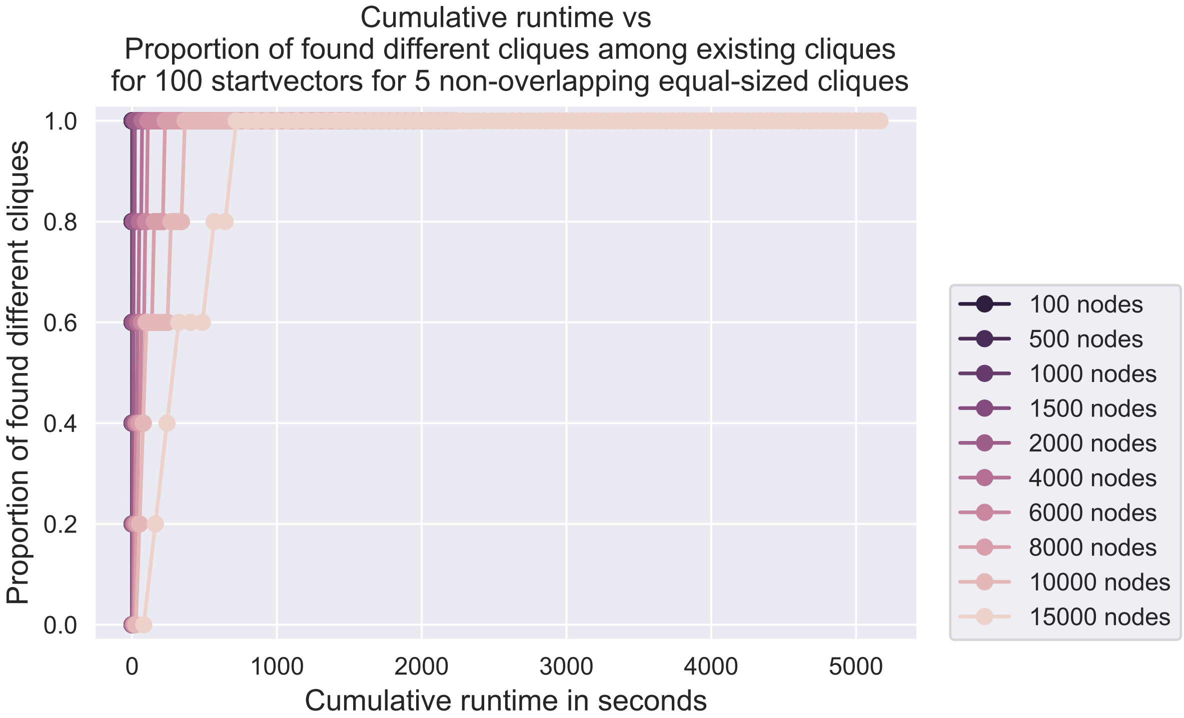

Figure 14: Dotplot of development of proportions of found different cliques among the existing cliques for the adjacency matrices of the structure “overlapping many small equal-sized cliques”. The proportion of found different cliques is depicted on the y-axis, the cumulative mean runtime on the x-axis. Every dot represents the proportion for each startvector and each matrix. The colours represent the number of nodes.

Instead of plotting the proportion of found different cliques against the number of startvectors as in 13 and 25, there is also the possibility of plotting it against the cumulative mean runtime in seconds as explained in section 3.3. This is plotted in figure 14.

In this figure for every startvector a dot is plotted and not as in figure 13 only for those startvectors, which find a new clique. As expected, the curves are shifted to the right with increasing number of nodes. Based on the preceding figures 1 and 13 it seems, that this shift is more due to the mean runtimes increasing with the matrix size as shown in figure 1 than to differences in required startvectors for reaching a certain proportion of found different cliques in figure 13. This is similar for the other structures.

For the structures with different-sized cliques, another interesting question is, whether - and if yes - which of the cliques are found more often than other cliques depending on their size. As for every adjacency matrix 100 startvectors were used, a clique can be found at maximum 100 times.
As an example the structure “non-overlapping few big different-sized cliques” chosen for illustrating. The matrices of this structure were created such, that the number of existing cliques is three for each adjacency matrix. The biggest clique contains 45% of the nodes for every matrix size, the other two cliques 35% and 20% of the nodes. Figure 15 shows, that the biggest clique was found in most of the cases for the matrix size 100. For the bigger matrix sizes the biggest clique was found by even all 100 startvectors.

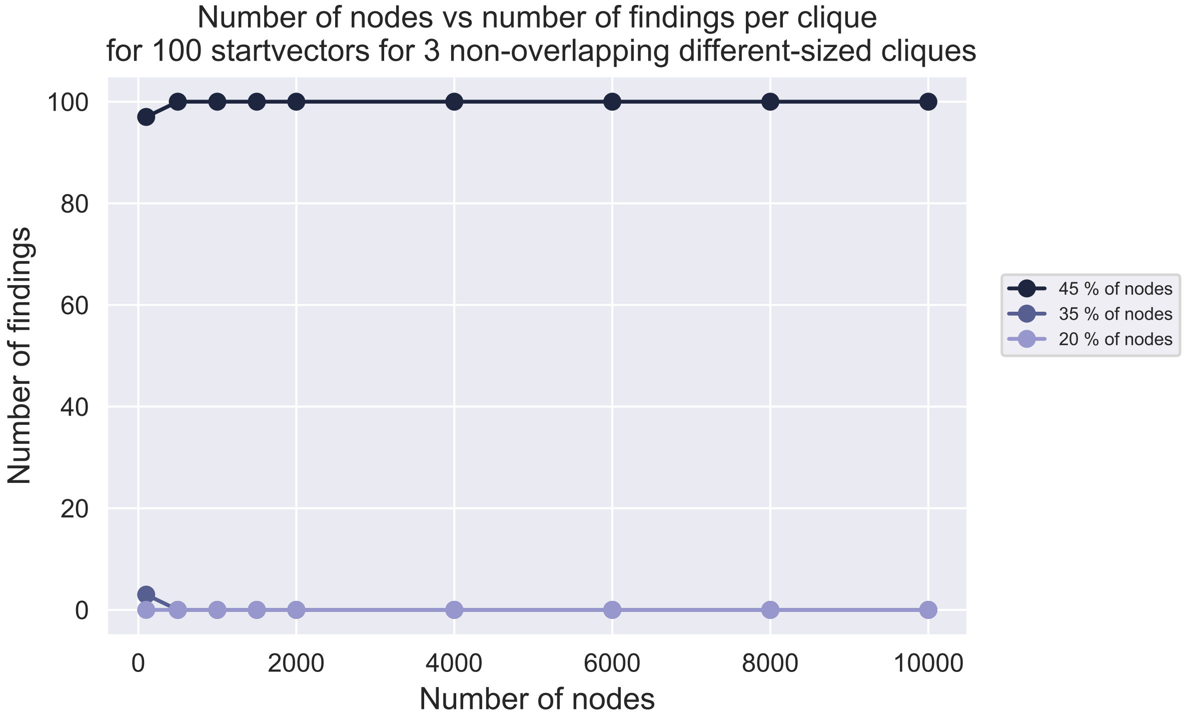

Figure 15: Dotplot of the number of findings of the different cliques in the adjacency matrices of the structure “non-overlapping few big different-sized cliques”. On the y-axis the number of findings is depicted, on the x-axis the number of nodes. The colours represent the proportion of nodes the cliques are covering.

As another example adjacency matrices of the structure “non-overlapping many small different-sized cliques” contain 20 cliques. Their sizes correspond to either 4%, 5% or 6%

of the nodes. Five of the 20 cliques are the biggest cliques covering 6% of the nodes. These five cliques were most often found among all the considered matrix sizes, as one can see in figure 16. In particular for a matrix size of 4000 nodes and bigger, these five cliques are the only ones, that were found.

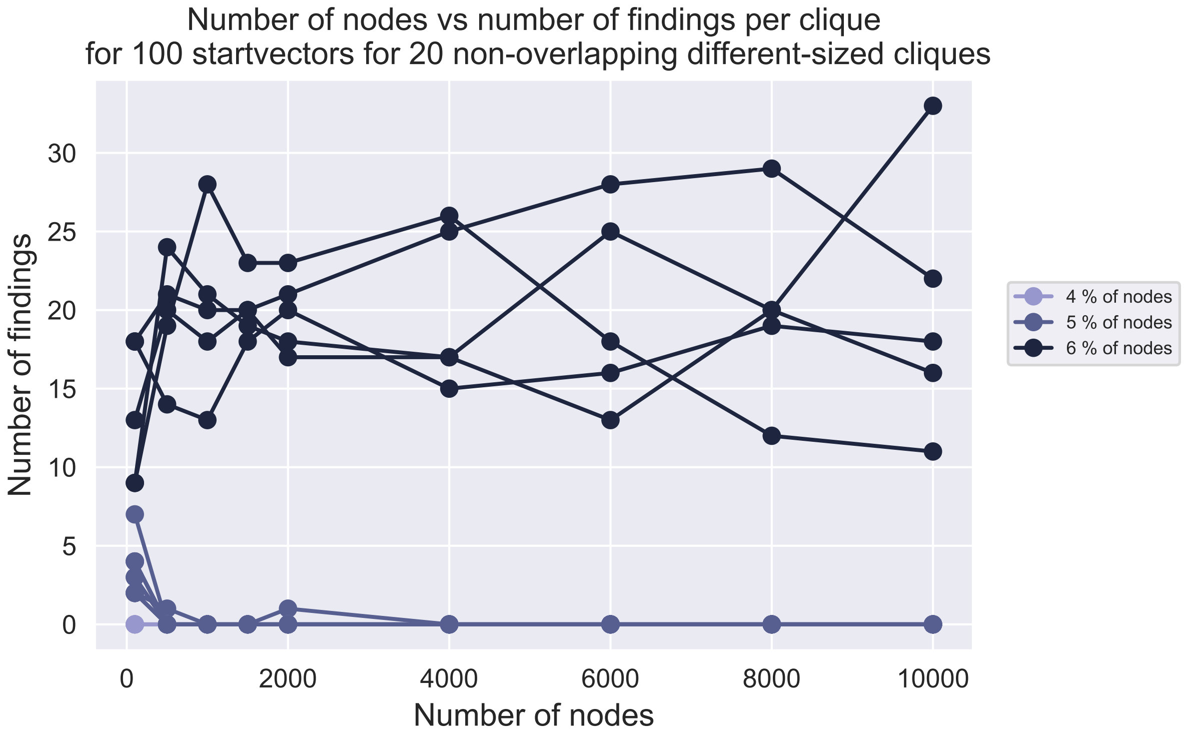

Figure 16: Dotplot of the number of findings of the different cliques in the adjacency matrices of the structure "non-overlapping many small different-sized cliques". On the y-axis the number of findings is depicted, on the x-axis the number of nodes. The colours represent the proportion of nodes the cliques are covering.

For the overlapping structures with different-sized cliques the same patterns can be discovered. Cliques with the biggest size were most often found. With increasing matrix size at some point the cliques with biggest size were the only ones to be found.

4.2.1.2 Effectiveness for different structures

As well as for the runtime also the proportion of found different cliques can be considered for matrices of the same size but with different structures, which is presented in this section.

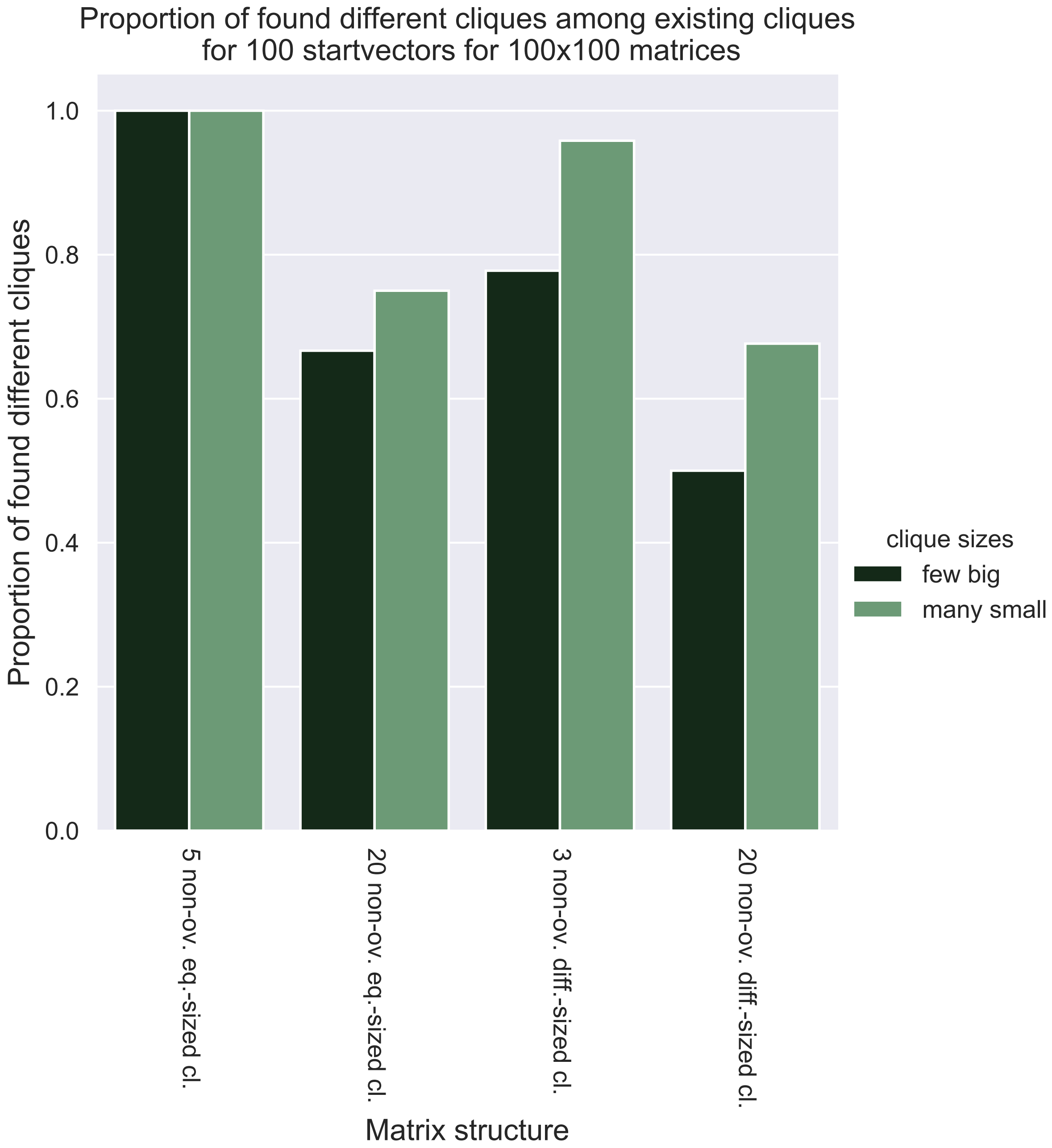

Figure 17: Barplot of proportions of found different cliques among the existing cliques after 100 startvectors for the adjacency matrices of all structures and 100 nodes. The proportion of found different cliques is depicted on the y-axis. The colours represent, whether a structure has few big or many small cliques.

The proportion of found different cliques among the existing cliques after 100 startvectors is shown in figure 17 exemplary for the matrix size 100. It's a barplot for all eight structures. There one can see, that for the two structures with non-overlapping equal-sized cliques after 100 startvectors all existing cliques were found. For all matrix sizes this pattern also shows up, whereas for the other six structures this is never the case. A second recognizable pattern is among those six last mentioned structures and matrix size

100, that these structures with few big cliques have lower proportions than the structures with many small cliques. This pattern also shows up for the matrix size with 500 nodes. But then it changes for the structures with different-sized cliques. The structures with many small different-sized cliques have smaller proportions than the structures with few big different-sized cliques for matrix size 1500 and bigger matrix sizes. The corresponding barplot of the matrix size 1500 in figure 18 shows this.

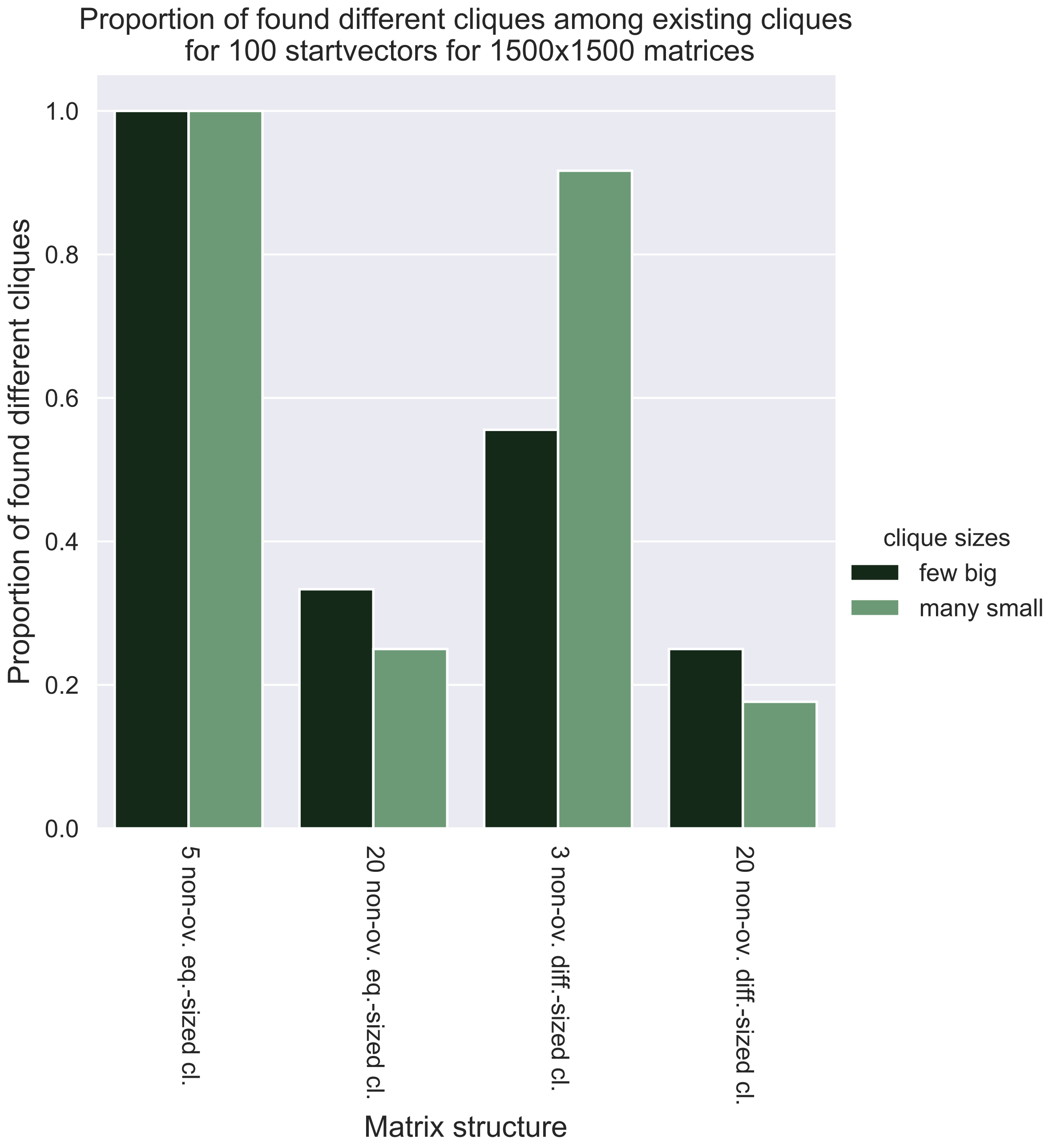

Figure 18: Barplot of proportions of found different cliques among the existing cliques after 100 startvectors for the adjacency matrices of all structures and 1500 nodes. The proportion of found different cliques is depicted on the y-axis. The colours represent, whether a structure has few big or many small cliques.

As in section 4.2.1.1 for adjacency matrices of one structure one can also look at the proportion of found different cliques after each startvector for adjacency matrices of one matrix size. The corresponding plot for matrix size 100 is shown in figure 19. For those four structures with many small cliques there are still many new cliques found after 20 startvectors. For the other structures with few big cliques, nearly all cliques that were found, were detected before the mark of the 20^{th} startvector.

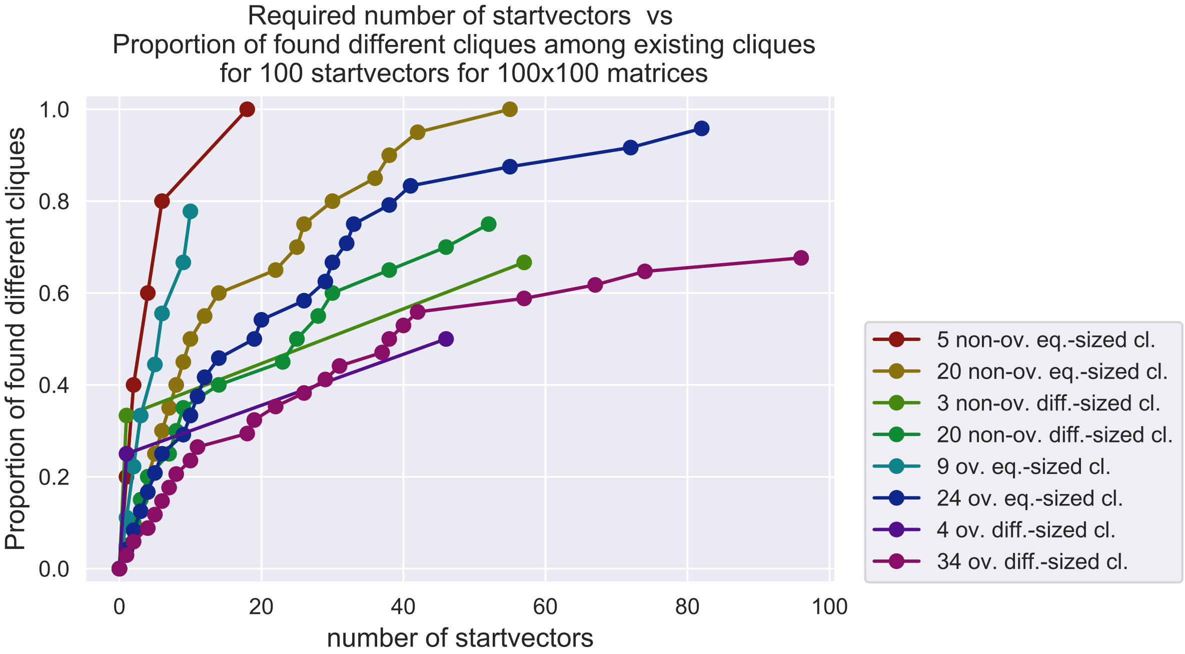

Figure 19: Dotplot of development of proportions of found different cliques among the existing cliques for the adjacency matrices of all eight matrix structures and 100 nodes. The proportion of found different cliques is depicted on the y-axis, the number of the startvectors on the x-axis. Every dot represents a change in the proportion because of a new found clique in an adjacency matrix. The colours represent the eight matrix structures.

Regarding the bigger matrix sizes the two structures with many small equal-sized cliques are remarkable as there was still a reasonable number of new cliques detected after 20 startvectors in all matrix sizes. Figure 26 shows this exemplary for the matrix size of 8000 nodes in the appendix.

The next section shortly addresses the effectiveness of the permuted adjacency matrices of the structure “non-overlapping few big equal-sized cliques”.

4.2.1.3 Effectiveness for permuted adjacency matrices

The effectiveness of the algorithm considered for the permuted adjacency matrices of the structure “non-overlapping few big equal-sized cliques” leads as in section 3.2 expected to the same results as for the non-permuted matrices of this structure. The proportions after 100 startvectors based on these permuted matrices are displayed in figure 27 in the appendix. The proportions are the same as in figure 12, as the startvectors were permuted in the same fashion as the adjacency matrices and hence the results of the algorithm are the same. The only difference is, that the permuted matrices were only considered up to matrix size 6000.

The completed section 4.2.1 addressed the effectiveness for the first group of adjacency matrices. The following section will look at the effectiveness for adjacency matrices of the second group.

4.2.2 Random graphs

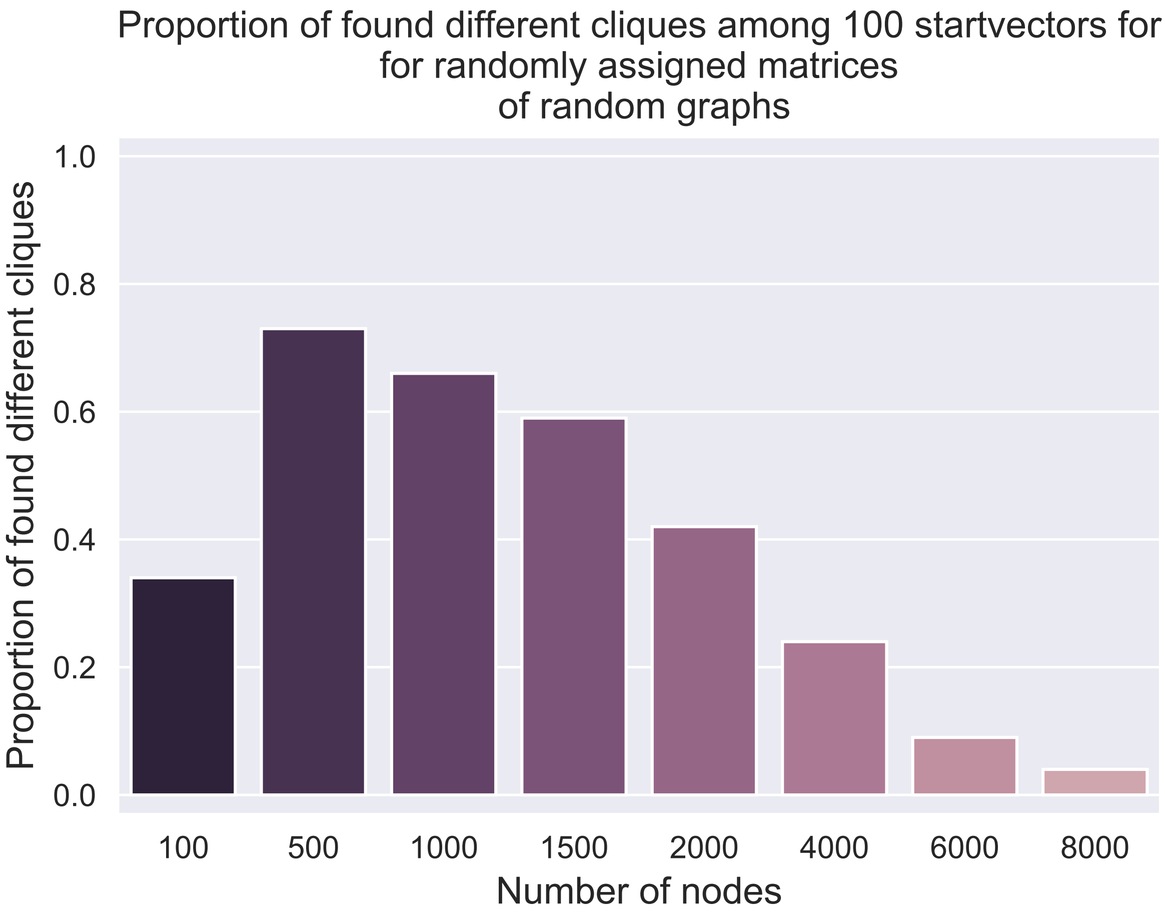

Figure 20: Barplot of the proportion of found different cliques among 100 startvectors after 100 startvectors in adjacency matrices of random graphs. The colours represent the number of nodes.

As motivated in section 3.3 for matrices of random graphs not the proportions of found different cliques among the existing cliques are considered. Instead the proportions of detected different cliques among the 100 startvectors were used. These proportions considered in this section were calculated as the number of detected different cliques after 100 startvectors divided by the number of startvectors, which is 100.

Figure 20 shows the proportion of found different cliques for randomly filled adjacency matrices with up to 8000 nodes. The proportions decrease with increasing matrix size, the only exception is the smallest matrix size with 100 nodes. Here the proportion is remarkably lower than the proportion of the next higher matrix size.

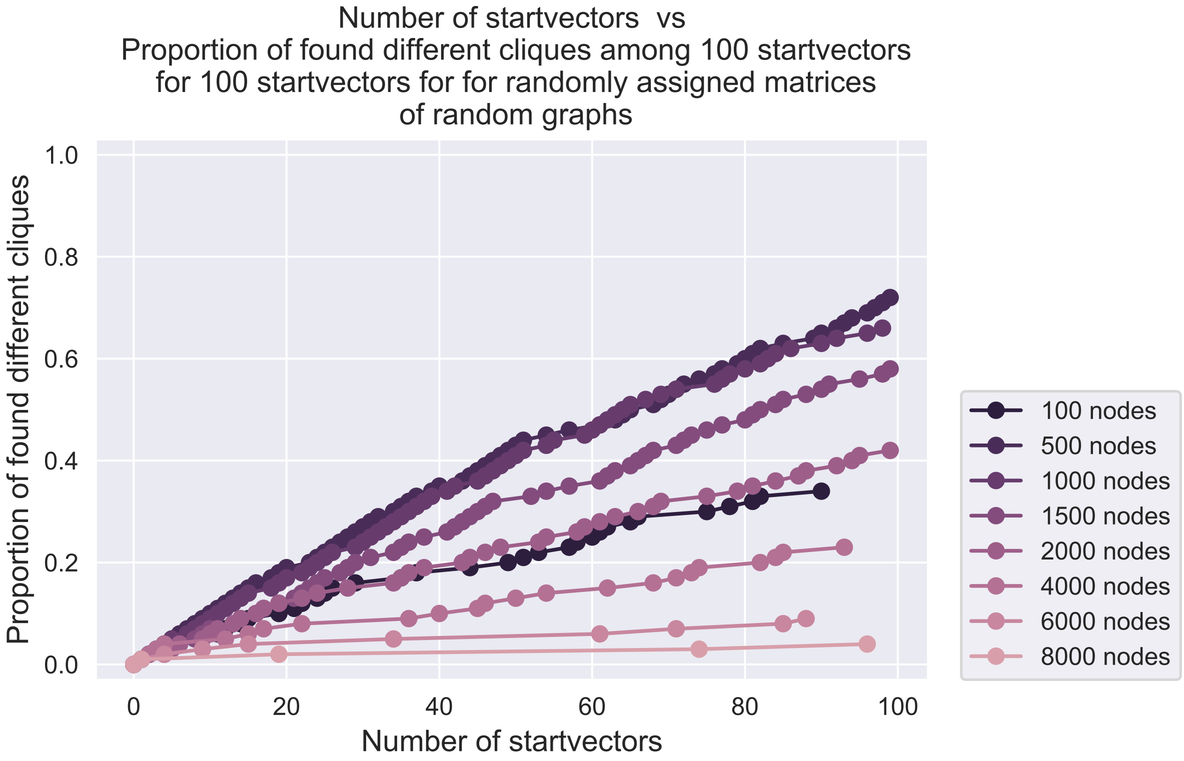

Figure 21: Dotplot of development of proportions of found different cliques among 100 startvectors for adjacency matrices of random graphs. The proportion of found different cliques is depicted on the y-axis, the number of the startvectors on the x-axis. Every dot represents a change in the proportion because of a new found clique in an adjacency matrix. The colours represent the number of nodes.

For all matrix sizes new cliques were detected over all the 100 startvectors, as one can see in figure 21. This is different compared to the adjacency structures from section 4.2.1, where for some structures no new cliques were found anymore after some first startvectors.

Finally the next section will examine the adjacency matrices of the third group regarding the effectiveness of the algorithm.

4.2.3 Random intersection graphs

For the adjacency matrices of the third group, that are created by using random intersection graphs, the proportions among 100 startvectors are calculated in the same way as for the matrices of random graphs. Matrix sizes up to 4000 are considered.
Figure 22 makes clear, that for every matrix size among 100 startvectors only one clique is found. For each matrix the respective clique is found 100 times. So the figure concerning the startvectors, at which new cliques are found, is not necessary.

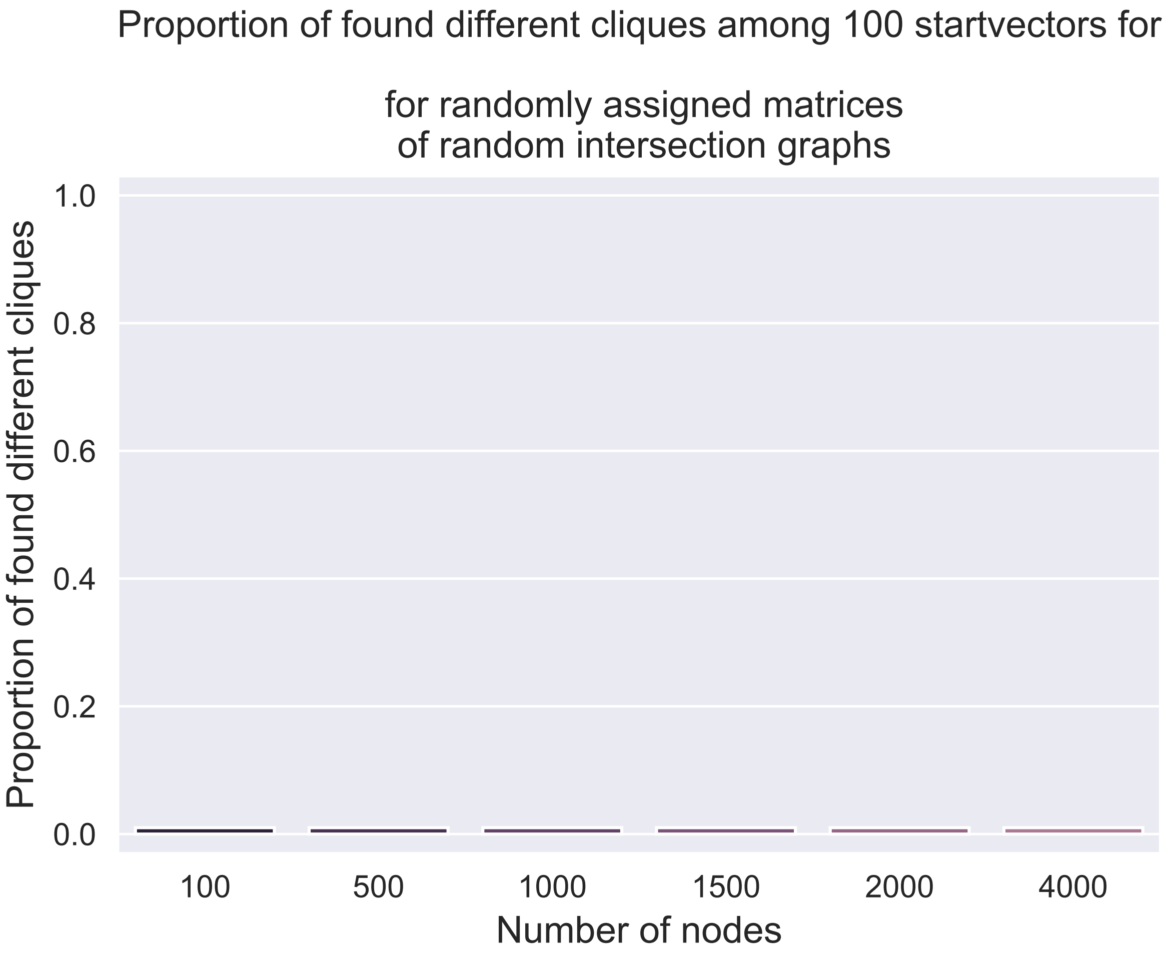

Figure 22: Barplot of the proportion of found different cliques among 100 startvectors after 100 startvectors in adjacency matrices of random intersection graphs. The colours represent the number of nodes.

In this and the previous section 4.1 the results of the algorithm regarding runtime and effectiveness for the three types of graphs respectively groups of adjacency matrices were presented. Based on that the following section will assess the possibility of applying the algorithm for enumerating all maximal cliques of a graph.

4.3 Algorithm for enumerating all maximal cliques

Based on the results of the sections 4.1 and 4.2 it seems, that the algorithm is not very appropriate for enumerating all maximal cliques of a graph for high dimensional data. First especially for random intersection graphs, which should be closer to real-world examples than the other two groups of adjacency matrices, only one clique was found among 100 startvectors. Hence it seems difficult to ensure, that all maximal cliques can be found by applying the algorithm on randomly chosen startvectors. Second the mean runtime seems to increase non-linearly for some of the considered adjacency matrices, which is not optimal regarding the application for high dimensional data.

5 Summary and Outlook

This thesis investigated an algorithm for finding one maximal clique in a graph, which was proposed by Ding et al. [2008]. It was studied regarding its effectiveness and runtime, which was measured as mean CPU-runtime. For this purpose three types of graphs as well as various different graph sizes, i.e. adjacency matrix sizes, were used. The first type of graphs comprises eight adjacency matrix structures, which were manually defined, one of them was additionally permuted. The second type of graphs are random graphs, the third type are random intersection graphs.

For all the adjacency matrices of all three types of graphs the mean runtime of the algorithm per startvector was found to be increasing over-proportionally with the number of nodes within the each considered segment of the matrix size.
In order to characterize the non-linearity of the runtime increase, both the runtimes and the numbers of nodes were transferred to the logarithmic space with the base of 10, thus leading to the logarithmic mean runtime as function of the logarithmic number of nodes. Based on this, linear models and polynomial models with quadratic term were estimated. The linear models have quite high proportions of explained variance. But also the additional quadratic term of the quadratic models was significant for every type of graphs except for the random graphs. Further research should examine the need for the quadratic term with even bigger matrix sizes.
In the first group of graphs two-sided permutation tests revealed significant differences between the mean runtimes of matrices with same size but different structures. Matrix structures with non-overlapping maximal cliques were found to have significantly lower mean runtimes compared to their corresponding structures with overlapping maximal cliques in one-sided tests, which were additionally conducted.

The algorithms' effectiveness was investigated with the first type of graphs by computing the proportion of found maximal cliques of existing maximal cliques. Beside this, the graphs of types two and three were used to examine the number of found maximal cliques when feeding the algorithm with 100 different startvectors. Both proportions are relatively low for some of the graphs and even decreasing with increasing matrix size for most of the graphs, which is a major drawback regarding the application of the algorithm for finding all maximal cliques of a graph in the area of high dimensional data. Additionally bigger maximal cliques are more often detected than smaller ones.

Some topics need further investigation in future.
The algorithm's runtime and effectiveness should be examined with even bigger matrix sizes and in particular with real-world datasets.
As the chosen startvectors have a strong impact on the found maximal cliques, optimized initialization strategies might improve the algorithm's effectiveness and might even speed up its convergence [Berry et al., 2007].

References

M. Behrisch and A. Taraz. Efficiently covering complex networks with cliques of similar vertices. *Theoretical Computer Science*, 355(1):37–47, 2006.

M. T. Belachew. Nmf-based algorithms for data mining and analysis: Feature extraction, clustering, and maximum clique finding: Phd thesis, 2014. URL `https://galileo.dm.uniba.it/dottorato/dottorato-di-ricerca-disattivato/tesi_dottorato/tesimelisewteferabelachew.pdf`.

M. T. Belachew and N. Gillis. Solving the maximum clique problem with symmetric rank-one non-negative matrix approximation. *Journal of Optimization Theory and Applications*, 173(1):279–296, 2017.

M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, and R. J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics & Data Analysis*, 52(1):155–173, 2007. ISSN 01679473. doi: 10.1016/j.csda.2006.11.006.

J. M. Bland and D. G. Altman. Multiple significance tests: the bonferroni method. *BMJ*, 310(6973):170, 1995.

C. Ding, T. Li, and M. I. Jordan. Nonnegative matrix factorization for combinatorial optimization: Spectral clustering, graph matching, and clique finding. In *2008 Eighth IEEE International Conference on Data Mining*, pages 183–192. IEEE, 2008.

R. L. Einsporn and D. Habtzghi. Combining paired and two-sample data using a permutation test. *Journal of Data Science*, 11(4):767–779, 2013.

P. Erdős and A. Rényi. On random graphs i. *Publicationes Mathematicae*, 6:290–297, 1959.

M. D. Ernst. Permutation methods: A basis for exact inference. *Statistical Science*, 19 (4):676–685, 2004.

L. Fahrmeir, T. Kneib, and S. Lang. *Regression: Modelle, Methoden und Anwendungen.* Statistik und ihre Anwendungen. Springer, Berlin Heidelberg, 2nd ed. edition, 2009.

L. Fahrmeir, R. Künstler, I. Pigeot, and G. Tutz. *Statistik: Der Weg zur Datenanalyse.* Springer-Lehrbuch. Springer, Berlin and Heidelberg, 7th ed., corr. reprint edition, 2011.

L. E. Gibbons, D. W. Hearn, P. M. Pardalos, and M. V. Ramana. Continuous characterizations of the maximum clique problem. *DIMACS Technical Report*, (9), 1996.

J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction and exact algorithms for clique cover. *Journal of Experimental Algorithmics*, 13, 2009.

A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

B. Hou, Z. Wang, Q. Chen, B. Suo, C. Fang, Z. Li, and Z. G. Ives. Efficient maximal clique enumeration over graph data. *Data Science and Engineering*, 1(4):219–230, 2016.

P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

D. Lee and H. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems 13 - Proceedings of the 2000 Conference, NIPS 2000*. Neural information processing systems foundation, 1 2001.

D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

W. McKinney. Data structures for statistical computing in python. In S. van der Walt and J. Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, Chris Fonnesbeck, Antony Lee, and Adel Qalieh. Mwaskom/seaborn: V0.8.1 (september 2017), 2017.

K. J. Millman, K. Ottoboni, and P. B. Stark. Core functions — permutation tests and confidence sets 0.1.alpha5 documentation, 2019. URL `https://statlab.github.io/permute/api/core.html`.

T. S. Motzkin and E. G. Straus. Maxima for graph and a new proof of a theorem of turán. *American Mathematical Society, Notices*, 11(382):533–540, 1964.

G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos. Using graph theory to analyze biological networks. *BioData Mining*, 4(10):1–27, 2011.

J. Pearl, M. Glymour, and N. P. Jewell. *Causal inference in statistics: A primer.* John Wiley & Sons Ltd, Chichester, West Sussex, UK, 2016.

M. Pelillo. Relaxation labeling networks for the maximum clique problem. *Journal of Artificial Neural Networks*, 2(4):313–328, 1995.

Python Core Team. *Python: A dynamic, open source programming language.* Python Software Foundation, 2019. URL `https://www.python.org/`.

S. Seabold and J. Perktold. Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

time. time — time access and conversions — python 3.7.4 documentation, 2019. URL `https://docs.python.org/3.7/library/time.html`.

timeit. timeit — measure execution time of small code snippets — python 3.7.4 documentation, 2019. URL `https://docs.python.org/3.7/library/timeit.html`.

S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. ISSN 1521-9615.

Appendix

Electronic Appendix

The electronic appendix **Masterthesis_Litzka** is structured as follows. It contains seven folders.

The folder **data** consists again of four folders. For all considered adjacency matrices both runtime measurements were done as well as the results of the algorithm, i.e. the cliques, were computed. The runtime measurements of all matrices are stored in folder **data_sim**, the results of the matrices of the first group in folder **data_results_struct**, the results of the matrices of the other two groups in folder **data_results_rand**. The fourth folder **test_results** contains the results of the permutation tests.

The code files for performing the runtime measurements are embodied in the folder **code_sim**. Every file is identified by a number and corresponds to one adjacency matrix. The folder **batch_files_sim** contains the corresponding batch files. Each file is as well numbered. A batch file does not call the code file with its corresponding number, instead the number of the code file corresponds to the order, that was randomly chosen for measuring the runtimes of the adjacency matrices. The additional file construct_adjacencies.py in folder **code_sim** is not intended for being run, instead it contains the code snippets for creating all adjacency matrices and their numbering as well as the choice of the order, with which the runtime measurements were taken.

The respective code files for computing the solutions, i.e. the cliques, of the adjacency matrices can be found in folder **code_sol**. The folder **batch_files_sol** contains the corresponding batch files.

The code files for analysing the runtime measurements and the solutions are embodied in folder **code_analysis**. The file results_struct.py creates the figures for the matrices of the first group. The corresponding file for the two other groups is results_rand.py. The figures for displaying, which of the cliques are found for matrices of the first group with different-sized structures, were created in a third file called results_cliques.py. The permutation test were conducted in file permtest.py, the models of the log mean runtimes and their figures in file regression_timeits.py.

All the figures are stored in the sixth folder **plot_results**.

In all code files there is a comment at some point, where the working directory should be set. After this point relative paths are used. These relative paths only work, if the folder **Masterthesis_Litzka** is chosen as working directory.

Additional

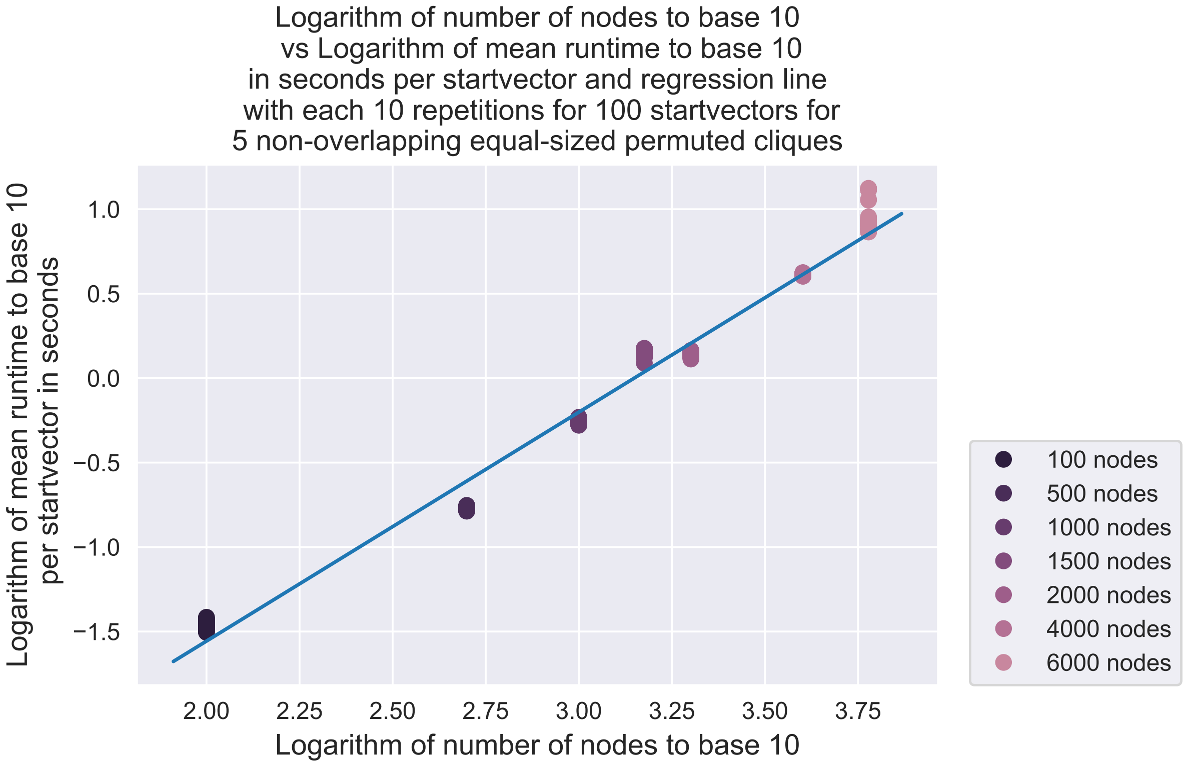

Figure 23: Dotplot of the log mean runtimes in seconds to the basis 10 of the permuted adjacency matrices with structure “non-overlapping few big equal-sized cliques” with the log number of nodes to the basis 10 on the x-axis and the log mean runtime to the basis 10 per startvector and adjacency matrix on the y-axis. A regression line of the linear model fitted on this data is drawn. The colours represent the number of nodes.

Kind of test	Matrix size	Clique subgroup	P-value
non-overlapping vs overlapping	100 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$
	500 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$
	1000 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$
	1500 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$
	2000 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$
	4000 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$
	6000 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$
	8000 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$
	10000 nodes	few big equal-sized	$9.9 \cdot 10^{-6}$
		many small equal-sized	$9.9 \cdot 10^{-6}$
		few big different-sized	$9.9 \cdot 10^{-6}$
		many small different-sized	$9.9 \cdot 10^{-6}$

Table 9: Table containing the p-values of the one-sided permutation tests for comparing the mean runtimes of matrices with non-overlapping and overlapping cliques in the four resulting clique subgroups and all numbers of nodes up to 10000.

Kind of test	Matrix size	Clique subgroup	P-value
few big vs many small	100 nodes	non-overlapping equal-sized	$9.9 \cdot 10^{-6}$
		non-overlapping different-sized	$9.9 \cdot 10^{-6}$
		overlapping equal-sized	$9.9 \cdot 10^{-6}$
		overlapping different-sized	$9.9 \cdot 10^{-6}$
	500 nodes	non-overlapping equal-sized	$9.9 \cdot 10^{-6}$
		non-overlapping different-sized	1
		overlapping equal-sized	$9.9 \cdot 10^{-6}$
		overlapping different-sized	1
	1000 nodes	non-overlapping equal-sized	1
		non-overlapping different-sized	1
		overlapping equal-sized	1
		overlapping different-sized	1
	1500 nodes	non-overlapping equal-sized	$9.9 \cdot 10^{-6}$
		non-overlapping different-sized	1
		overlapping equal-sized	$9.9 \cdot 10^{-6}$
		overlapping different-sized	1
	2000 nodes	non-overlapping equal-sized	$9.9 \cdot 10^{-6}$
		non-overlapping different-sized	1
		overlapping equal-sized	$9.9 \cdot 10^{-6}$
		overlapping different-sized	1
	4000 nodes	non-overlapping equal-sized	1
		non-overlapping different-sized	$9.9 \cdot 10^{-6}$
		overlapping equal-sized	1
		overlapping different-sized	$9.9 \cdot 10^{-6}$
	6000 nodes	non-overlapping equal-sized	$9.9 \cdot 10^{-6}$
		non-overlapping different-sized	$9.9 \cdot 10^{-6}$
		overlapping equal-sized	$9.9 \cdot 10^{-6}$
		overlapping different-sized	$9.9 \cdot 10^{-6}$
	8000 nodes	non-overlapping equal-sized	1
		non-overlapping different-sized	1
		overlapping equal-sized	1
		overlapping different-sized	1
	10000 nodes	non-overlapping equal-sized	$9.9 \cdot 10^{-6}$
		non-overlapping different-sized	1
		overlapping equal-sized	$9.9 \cdot 10^{-6}$
		overlapping different-sized	1

Table 10: Table containing the p-values of the one-sided permutation tests for comparing the mean runtimes of matrices with few big and many small cliques in the four resulting clique subgroups and all numbers of nodes up to 10000.

Kind of test	Matrix size	Clique subgroup	P-value
equal-sized vs different-sized	100 nodes	non-overlapping few big	$9.9 \cdot 10^{-6}$
		non-overlapping many small	$9.9 \cdot 10^{-6}$
		overlapping few big	$9.9 \cdot 10^{-6}$
		overlapping many small	$9.9 \cdot 10^{-6}$
	500 nodes	non-overlapping few big	1
		non-overlapping many small	1
		overlapping few big	1
		overlapping many small	1
	1000 nodes	non-overlapping few big	1
		non-overlapping many small	$9.9 \cdot 10^{-6}$
		overlapping few big	1
		overlapping many small	$9.9 \cdot 10^{-6}$
	1500 nodes	non-overlapping few big	1
		non-overlapping many small	1
		overlapping few big	1
		overlapping many small	1
	2000 nodes	non-overlapping few big	1
		non-overlapping many small	1
		overlapping few big	1
		overlapping many small	1
	4000 nodes	non-overlapping few big	$9.9 \cdot 10^{-6}$
		non-overlapping many small	1
		overlapping few big	$9.9 \cdot 10^{-6}$
		overlapping many small	1
	6000 nodes	non-overlapping few big	$9.9 \cdot 10^{-6}$
		non-overlapping many small	$9.9 \cdot 10^{-6}$
		overlapping few big	$9.9 \cdot 10^{-6}$
		overlapping many small	$9.9 \cdot 10^{-6}$
	8000 nodes	non-overlapping few big	$9.9 \cdot 10^{-6}$
		non-overlapping many small	$9.9 \cdot 10^{-6}$
		overlapping few big	$9.9 \cdot 10^{-6}$
		overlapping many small	$9.9 \cdot 10^{-6}$
	10000 nodes	non-overlapping few big	1
		non-overlapping many small	$9.9 \cdot 10^{-6}$
		overlapping few big	1
		overlapping many small	$9.9 \cdot 10^{-6}$

Table 11: Table containing the p-values of the one-sided permutation tests for comparing the mean runtimes of matrices with equal-sized and different-sized cliques in the four resulting clique subgroups and all numbers of nodes up to 10000.

Model	Covariable	Coefficient	P-value
linear model	intercept	-4.2159	0.0
	log_{10}(number of nodes)	1.3460	0.0
polynomial model	intercept	-2.8723	0.0
	log_{10}(number of nodes)	0.3371	0.0
	$(log_{10}(\text{number of nodes}))^2$	0.1821	0.341

Table 12: Table containing the coefficients and their p-values for two models fitted for adjacency matrices of random intersection graphs with the log mean runtime to the basis 10 as dependent variable.

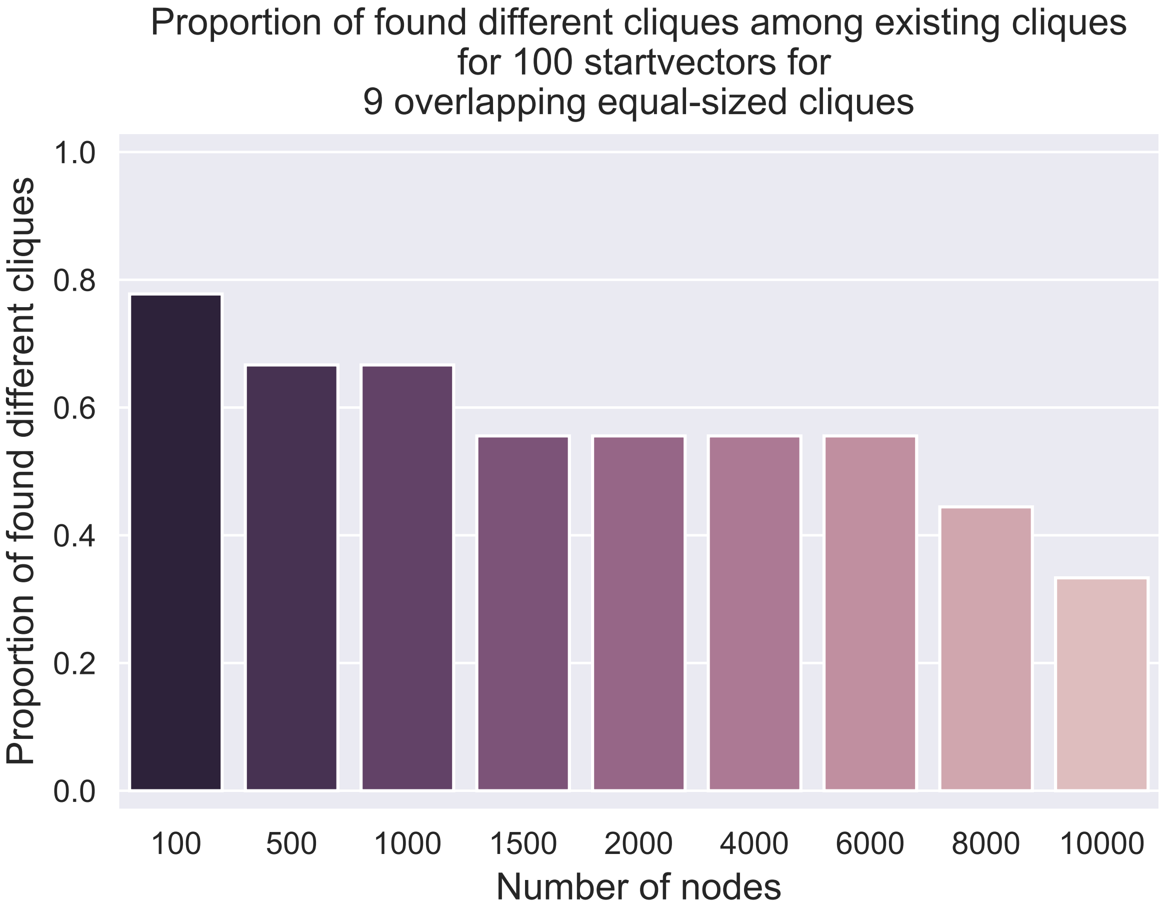

Figure 24: Barplot of the proportion of found different cliques among the twenty existing cliques after 100 startvectors in adjacency matrices of the structure “overlapping few big equal-sized cliques”. The colours represent the number of nodes.

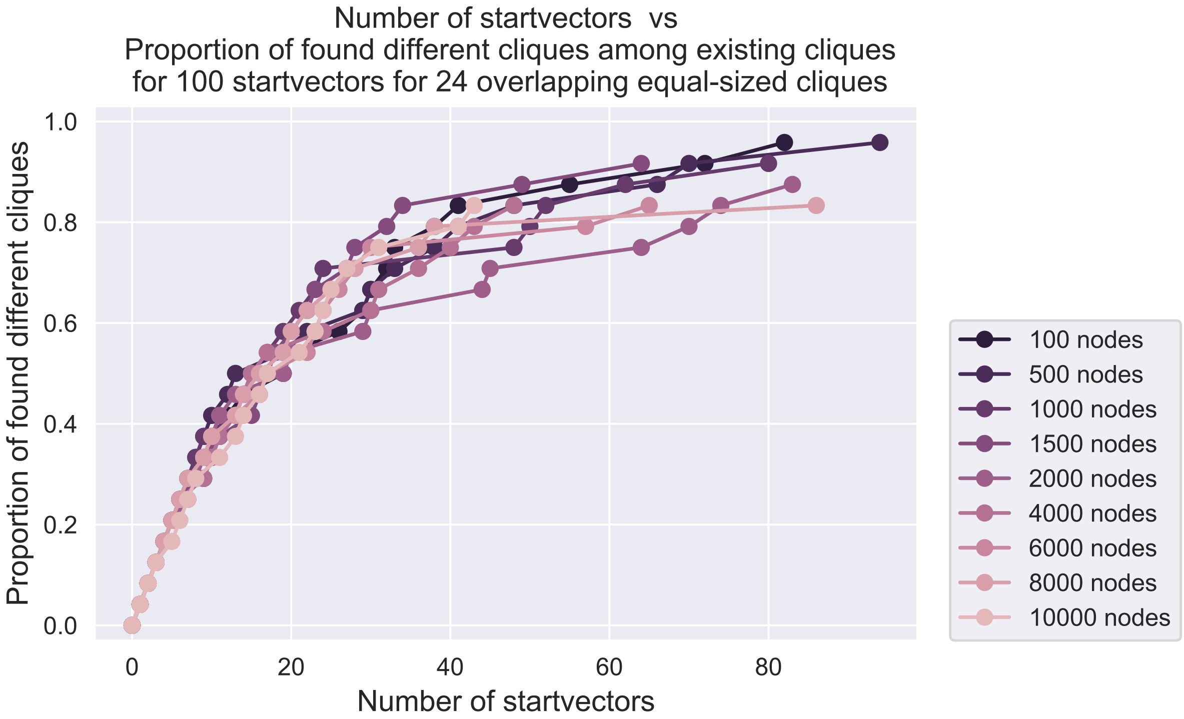

Figure 25: Dotplot of development of proportions of found different cliques among the existing cliques for the adjacency matrices of the structure "overlapping many small equal-sized cliques". The proportion of found different cliques is depicted on the y-axis, the number of the startvectors on the x-axis. Every dot represents a change in the proportion because of a new found clique in an adjacency matrix. The colours represent the number of nodes.

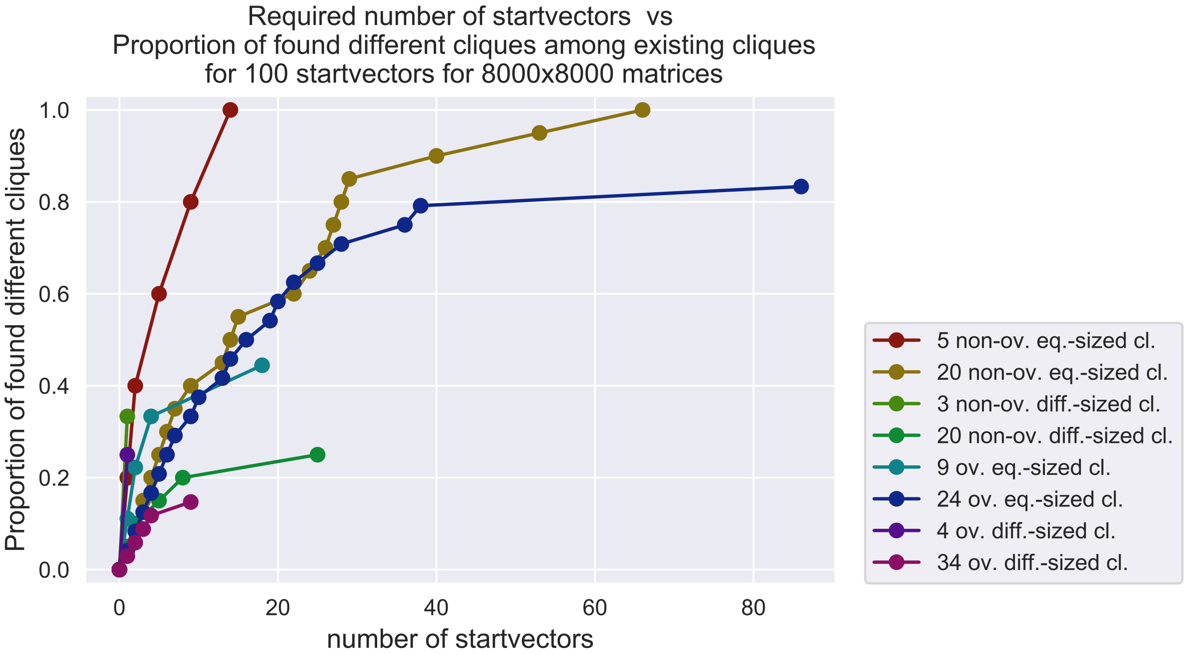

Figure 26: Dotplot of development of proportions of found different cliques among the existing cliques for the adjacency matrices of all eight matrix structures and 8000 nodes. The proportion of found different cliques is depicted on the y-axis, the number of the startvectors on the x-axis. Every dot represents a change in the proportion because of a new found clique in an adjacency matrix. The colours represent the eight matrix structures.

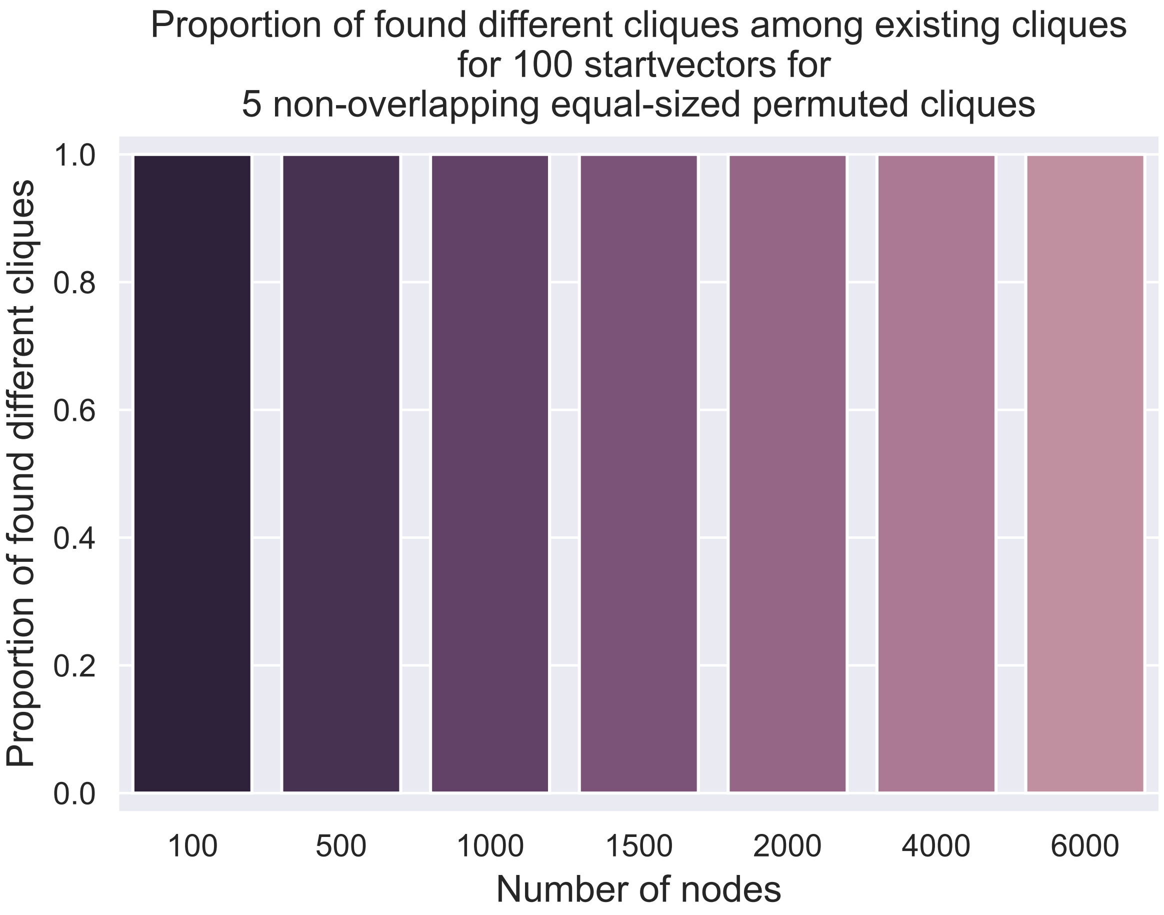

Figure 27: Barplot of the proportion of found different cliques among the five existing cliques after 100 startvectors in permuted adjacency matrices of the structure “non-overlapping few big equal-sized cliques”. The colours represent the number of nodes.

Statutory Declaration

I declare that I, Leonie Friederike Litzka, have authored this thesis independently, that I have not used other than the declared sources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place, Date

Signature