6 Local Density

Sven Kosub

Actors in networks usually do not act alone. By a selective process of establishing relationships with other actors, they form groups. The groups are typically founded by common goals, interests, preferences or other similarities. Standard examples include personal acquaintance relations, collaborative relations in several social domains, and coalitions or contractual relationships in markets. The cohesion inside these groups enables them to influence the functionality of the whole network.

Discovering cohesive groups is a fundamental aspect in network analysis. For a computational treatment, we need formal concepts reflecting some intuitive meaning of cohesiveness. At a general level, the following characteristics have been attributed to cohesive groups [569]:

- *Mutuality:* Group members choose each other to be included in the group. In a graph-theoretical sense, this means that they are adjacent.
- *Compactness:* Group members are well reachable for each other, though not necessarily adjacent. Graph-theoretically, this comes in two flavors: being well reachable can be interpreted as having short distances or high connectivity.
- *Density:* Group members have many contacts to each other. In terms of graph theory, that is group members have a large neighborhood inside the group.
- *Separation:* Group members have more contacts inside the group than outside.

Depending on the network in question, diverse concepts can be employed, incorporating cohesiveness characteristics with different emphases. Notions where density is a dominant aspect are of particular importance.

Density has an outstanding relevance in social networks. On the one hand, recent studies have found that social networks show assortative mixing on vertices [441, 444, 446], i.e, they tend to have the property that neighbors of vertices with high degree have also high degree. Assortative mixing is an expression of the typical observation that social networks are structured by groups of high density.[1] On the other hand, there are several mathematical results demonstrating that high density implies the other characteristics of cohesiveness. For instance, one classical result [431] says that if each member of a group shares ties with at least

[1] Assortativity is now considered as one statistical criterion separating social networks and non-social networks [446]. For instance, some experimental analyses have shown that in the Internet at the level of autonomous systems, the mean degree of the neighbors of an autonomous system with k neighbors is approximately $k^{-1/2}$ [468]. At this level, the Internet is disassortatively mixed.

U. Brandes and T. Erlebach (Eds.): Network Analysis, LNCS 3418, pp. 112–142, 2005.

a $\frac{1}{k}$-fraction of the other members of the group, then the tie distance within the group is at most k. Results comparable to that can be proven for connectivity as well. Here, however, the dependency from density is not as strong as in the case of distances (see Chapter 7).

In this chapter, we survey computational approaches and solutions for discovering *locally* dense groups. A graph-theoretical group property is local if it is definable over subgraphs induced by the groups only. Locality does not correspond to the above-mentioned separation characteristic of cohesiveness, since it neglects the network outside the group. In fact, most notions that have been defined to cover cohesiveness have a maximality condition. That is, they require for a group to be cohesive with respect to some property Π, in addition to fulfilling Π, that it is not contained in any larger group of the network that satisfies Π as well. Maximality is non-local. We present the notions on the basis of their underlying graph-theoretical properties and without the additional maximality requirements. Instead, maximality appears in connection with several computational problems derived from these notions. This is not a conceptual loss. Actually, it emphasizes that locality reflects an important hidden aspect of cohesive groups: being invariant under network changes outside the group. Interior robustness and stability is an inherent quality of groups. Non-local density notions and the corresponding algorithmic problems and solutions are presented in Chapter 8. A short list of frequently used non-local notions is also discussed in Section 6.4.

The prototype of a cohesive group is the clique. Since its introduction into sociology in 1949 [401], numerous efforts in combinatorial optimization and algorithms have been dedicated to solving computational problems for cliques. Therefore, the treatment of algorithms and hardness results for clique problems deserves a large part of this chapter. We present some landmark results in detail in Section 6.1. All other notions that we discuss are relaxations of the clique concept. We make a distinction between structural and statistical relaxations. A characteristic of structural densities is that all members of a group have to satisfy the same requirement for group membership. These notions (plexes, cores) admit strong statements about the structure within the group. Structurally dense groups are discussed in Section 6.2. In contrast, statistical densities average over members of a group. That is, the property that defines group membership needs only be satisfied in average (or expectation) over all group members. In general, statistically dense groups reveal only few insights into the group structure. However, statistical densities can be applied under information uncertainty. They are discussed in Section 6.3.

All algorithms are presented for the case of unweighted, undirected simple graphs exclusively. Mostly, they can be readily translated for directed or weighted graphs. In some exceptional cases where new ideas are needed, we mention these explicitly.

6.1 Perfectly Dense Groups: Cliques

The graph with perfect cohesion is the clique [401].

Definition 6.1.1. *Let $G = (V, E)$ be an undirected graph. A subset $U \subseteq V$ is said to be a* clique *if and only if $G[U]$ is a complete graph.*

In a clique, each member has ties with each other member. A clique U is a maximal clique in a graph $G = (V, E)$ if and only if there is no clique U' in G with $U \subset U'$. A clique is a maximum clique in graph G if and only if it has maximum cardinality among all cliques in G.

The striking reasons for perfectness of cliques as cohesive structures are obvious:

1. Cliques are perfectly dense, i.e., if U is a clique of size k, then $\delta(G[U]) = \bar{d}(G[U]) = \Delta(G[U]) = k - 1$. A higher degree is not possible.
2. Cliques are perfectly compact, i.e., $\mathrm{diam}(G[U]) = 1$. A shorter distance between any two vertices is not possible.
3. Cliques are perfectly connected, i.e., if U is a clique of size k, then U is $(k-1)$-vertex-connected and $(k-1)$-edge-connected. A higher connectivity is not possible.

The structural properties of a clique are very strong. In real-world settings, large cliques thus should be rarely observable. The famous theorem of Turán [554] gives precise sufficient conditions for the guaranteed existence of cliques of certain sizes with respect to the size of the entire network.

Theorem 6.1.2 (Turán, 1941). *Let $G = (V, E)$ be an undirected graph. If $m > \frac{n^2}{2} \cdot \frac{k-2}{k-1}$, then there exists a clique of size k within G.*

An immediate consequence of this theorem is that a network itself needs to be dense in order to surely possess a large clique. However, as social networks are usually sparse, we have no *a priori* evidence for the existence of a clique. Identifying cliques becomes an algorithmic task. Note that, as we will see below, even if we knew that there is a clique of a certain size in a network, we would not be able to locate it in reasonable time.

Maximal cliques do always exist in a graph. In fact, there are many of them and they tend to overlap, i.e., in general it can be the case that maximal cliques U_1 and U_2 exist satisfying $U_1 \neq U_2$ and $U_1 \cap U_2$ is non-empty. Another classical result due to Moon and Moser [432] gives a tight estimation of the number of maximal cliques:

Theorem 6.1.3 (Moon and Moser, 1965). *Every undirected graph G with n vertices has at most $3^{\lceil \frac{n}{3} \rceil}$ maximal cliques.*

In reality, the expected enormous number of maximal cliques leads to the serious problem of how to identify the more important ones among them. There are only few algorithmic techniques available providing helpful interpretation of the

maximal-clique collection. Prominent examples for methods are based on the co-membership matrix or the clique overlap centrality [192].

The family of all cliques of a certain graph shows some structure:

1. Cliques are closed under exclusion, i.e., if U is a clique in G and $v \in U$, then $U - \{v\}$ is also a clique.[2]
2. Cliques are nested, i.e., each clique of size n contains a clique of size $n-1$ (even n cliques of size $n-1$). Though this is an immediate consequence of the closure under exclusion, it is a property to be proved for related notions that are not closed under exclusion.

Distance-based cliques. There is a number of approaches to generalize the notion of a clique that are relevant in several settings of social-network theory. We list some of them [400, 14, 429]. Let $G = (V, E)$ be an undirected graph, let U be a vertex subset of V, and let $N > 0$ be any natural number.

1. U is said to be an N-clique if and only if for all $u, v \in U$, $d_G(u, v) \leq N$.
2. U is said to be an N-club if and only if $\mathrm{diam}(G[U]) \leq N$.
3. U is said to be an N-clan if and only if U is a maximal N-clique and $\mathrm{diam}(G[U]) \leq N$.

N-cliques are based on non-local properties, as the distance between vertices u and v is measured with respect to graph G, and not with respect to $G[U]$. An immediate consequence is that N-cliques need not be connected for $N > 1$. Though clubs and clans are local structures (except the maximality condition), they are of minor interest in our context, since they emphasize distances rather than density. Moreover, there has been some criticism of distance-based cliques, which was sparked off by at least two facts (cf., e.g., [514, 189]). First, in many cases real-world networks have globally a small diameter, thus, the distance is a rather coarse measure to identify meaningful network substructures. Second, distance-based cliques are in general neither closed under exclusion nor nested.

6.1.1 Computational Primitives

In many respects, cliques are simple objects, easily manageable from an algorithmic point of view. We have fast algorithms with run-time $\mathcal{O}(n+m)$ at hand for several computational primitives:

1. *Determine if a given set $U \subseteq V$ of vertices is a clique in G.* We simply test whether each pair of vertices of U is an edge in G. Note that these are up to $\binom{n}{2}$ pairs, but even if we have much fewer edges, after testing m pairs we are done in any case.
2. *Determine if a given clique $U \subseteq V$ is maximal in G.* We simply test whether there exists a vertex in $V - U$ which is adjacent to all vertices in U. Again, after testing m edges we are done in the worst case.

[2] In graph theory, a property Π is called *hereditary* if and only if, whenever a graph satisfies Π, so does every induced subgraph. Being a clique is a hereditary property of graphs.

Another efficiently computable primitive is finding some maximal clique. For later use, we state this in a more general form. Suppose that the vertex set V of a graph $G = (V, E)$ is ordered. We say that a set $U \subseteq V$ is lexicographically smaller than a set $U' \subseteq V$ if and only if the first vertex that is not in both U and U' belongs to U. Our third primitive is the following:

3. *Compute the lexicographically smallest maximal clique containing some clique* U'. We start with setting $U := U'$, iterate over all $v \in V - U$ in increasing order, and test for each v whether $U \subseteq N(v)$; if this is the case then add vertex v to U. After completing the iteration, U is a maximal clique containing U'. This works in time $\mathcal{O}(n + m)$.

Algorithmic difficulties appear only when we are interested in finding cliques of certain sizes or maximum cliques. For these problems, no algorithms with running times comparable to the one above are known (and, probably, no such algorithms exist).

6.1.2 Finding Maximum Cliques

We discuss several aspects of the maximum clique problem. Of course, it is easy to compute a clique of maximum size, if we do not care about time. The obvious approach is exhaustive search. In an exhaustive search algorithm, we simply enumerate all possible candidate sets $U \subseteq V$ and examine if U is a clique. We output the largest clique found. A simple estimation gives a worst-case upper bound $\mathcal{O}(n^2 \cdot 2^n)$ on the time complexity of the algorithm.

Computational hardness. The problem arises whether we can improve the exhaustive search algorithm significantly with respect to the amount of time. Unfortunately, this will probably not be the case. Computationally, finding a maximum clique is an inherently hard problem. We consider the corresponding decision problem:

Problem:	CLIQUE
Input:	Graph G, Parameter $k \in \mathbb{N}$
Question:	Does there exist a clique of size at least k within G?

Let $\omega(G)$ denote the size of a maximum clique of a graph G. Note that if we have an algorithm that decides CLIQUE in time $T(n)$ then we are able to compute $\omega(G)$ in time $\mathcal{O}(T(n) \cdot \log n)$ using binary search. The other way around, any $T(n)$ algorithm for computing $\omega(G)$, gives a $T(n)$ algorithm for deciding CLIQUE. Thus, if we had a polynomial algorithm for CLIQUE, we would have a polynomial algorithm for maximum-clique sizes, and vice versa. However, CLIQUE was among the first problems for which $\mathcal{NP}$-completeness was established [345].

Theorem 6.1.4. CLIQUE *is* $\mathcal{NP}$*-complete.*

Proof. Note that testing whether some guessed set is a clique is possible in polynomial time. This shows the containment in $\mathcal{NP}$. In order to prove the $\mathcal{NP}$-hardness, we describe a polynomial-time transformation of SATISFIABILITY into CLIQUE. Suppose we are given a Boolean formula H in conjunctive normal form consisting of m clauses $C_1, \ldots, C_k$. For H we construct a k-partite graph G_H where vertices are the literals of H labeled by their clause, and where edges connect literals that are not negations of each other. More precisely, define $G_H = (V_H, E_H)$ to be the following graph:

$$V_H =_{\text{def}} \{\ (L,i)\ \mid\ i \in \{1,\ldots,k\} \text{ and } L \text{ is a literal in clause } C_i\ \}$$
$$E_H =_{\text{def}} \{\ \{(L,i),(L',j)\}\ \mid\ i \neq j \text{ and } L \neq \neg L'\ \}$$

Clearly, the graph G_H can be computed in time polynomial in the size of the formula H. We show that H is satisfiable if and only if the graph G_H contains a clique of size k.

Suppose that H is satisfiable. Then there exists a truth assignment to variables $x_1, \ldots, x_n$ such that in each clause at least one literal is true. Let $L_1, \ldots, L_k$ be such literals. Then, of course, it must hold that $L_i \neq \neg L_j$ for $i \neq j$. We thus obtain that the set $\{(L_1,1),\ldots,(L_k,k)\}$ is a clique of size k in G_H.

Suppose now that $U \subseteq V_H$ is a clique of size k in graph G_H. Since G_H is k-partite, U contains exactly one vertex from each part of V_H. By definition of set V_H, we have that for all vertices (L,i) and (L',j) of U, $L \neq \neg L'$ whenever $i \neq j$. Hence, we can assign truth values to variables in such a way that all literals contained in U are satisfied. This gives a satisfying truth assignment to formula H. □

So, unless $\mathcal{P} = \mathcal{NP}$, there are no algorithms with a running time polynomial in n for solving CLIQUE with arbitrary clique size or computing the maximum clique. On the other hand, even if we have a guarantee that there is a clique of size k in graph G, then we are not able to find it in polynomial time.

Corollary 6.1.5. *Unless $\mathcal{P} = \mathcal{NP}$, there is no algorithm running in polynomial time to find a clique of size k in a graph which is guaranteed to have a clique of size k.*

Proof. Suppose we have an algorithm A that runs in polynomial time on each input (G,k) and outputs a clique of size k, if it exists, and behaves in an arbitrary way in the other cases. A can be easily modified into an algorithm A' that decides CLIQUE in polynomial time. On input (G,k), run algorithm A, if A produces no output, then reject the instance. If A outputs a set U, then test whether U is a clique. If so, accept, otherwise reject. This procedure is certainly polynomial time. □

Note that the hardness of finding the hidden clique does not depend on the size of the clique. Even very large hidden cliques (of size $(1-\varepsilon)n$ for $\varepsilon > 0$) cannot be found unless $\mathcal{P} = \mathcal{NP}$ (see, e.g., [308, 37]). The situation becomes slightly better if we consider randomly chosen graphs, i.e., graphs where each edge appears

with probability $\frac{1}{2}$. Suppose we additionally place at random a clique of size k in such a random graph of size n. How fast can we find this clique? It has been observed that, if $k = \Omega(\sqrt{n \log n})$, then almost surely the k vertices with highest degree form the clique [374]. This gives a trivial $\mathcal{O}((n+m)\log n)$ algorithm (which can be improved to an $\mathcal{O}(n+m)$ algorithm with a technique discussed in Theorem 6.2.7). For $k = \Omega(\sqrt{n})$, algorithms based on spectral techniques have been proven to find hidden cliques of size k in polynomial time [22] (even in some weaker random graph models [202]). However, many natural algorithmic techniques do not achieve the goal of finding hidden cliques of size $k = o(\sqrt{n})$ [328].

Better exponential algorithms. Even though we will probably never have a polynomial-time algorithm for finding maximum cliques, we can try to design fast, super-polynomial algorithms. Exhaustive search gives the upper bound $\mathcal{O}(n^2 \cdot 2^n)$, or $\mathcal{O}^*(2^n)$ when omitting polynomial factors. Our goal is to design algorithms having running times $\mathcal{O}^*(\beta^n)$ with β as small as possible. The following theorem that can be found in [590] shows that we can do better than exhaustive search.

Theorem 6.1.6. *There exists an algorithm for computing a maximum clique in time* $\mathcal{O}^*(1.3803^n)$.

Sketch of Proof. We use a backtracking scheme with pruning of the recursion tree. Let G be a graph having n vertices and m edges. Let $v \in V$ be any vertex of minimum degree. If $\delta(G) \geq n-3$ then the graph misses collections of pairwise disjoint cycles and paths, for being a complete graph. In this case, it is fairly easy to compute a maximum clique in $\mathcal{O}(n+m)$.[3] Assume that there is a vertex v with degree $d_G(v) \leq n-4$. Every maximum clique either contains v or not. Corresponding to these two cases, a maximum clique of G is either $\{v\}$ combined with a maximum clique of the induced subgraph $G[N(v)]$ or a maximum clique of the induced subgraph $G[V - \{v\}]$. We recursively compute maximum cliques in both subgraphs and derive from them a solution for G (breaking ties arbitrarily). The worst-case time $T(n)$ essentially depends on the following recursive inequality:

$$T(n) \leq T(n-4) + T(n-1) + c \cdot (n+m) \quad \text{for some } c > 0$$

Using standard techniques based on generating functions, we calculate that $T(n)$ is within a polynomial factor of β^n where $\beta \approx 1.3803$ is the largest real zero of the polynomial $\beta^4 - \beta^3 - 1$. □

[3] It might be easier to think of independent sets rather than cliques. An independent set in a graph G is a set U of vertices such that $G[U]$ has no edges. A clique in graph G corresponds to an independent set in graph $\overline{G}$, where in $\overline{G}$ exactly those vertices are adjacent that are not adjacent in G. Independent sets are a little bit easier to handle, since we do not have to reason about edges that are not in the graph. In fact, many algorithms in the literature are described for independent sets.

The intuitive algorithm in the theorem captures the essence of a series of fast exponential algorithms for the maximum clique problem. It started with an $\mathcal{O}^*(1.286^n)$ algorithm [543] that follows essentially the ideas of the algorithm above. This algorithm has been subsequently improved to $\mathcal{O}^*(1.2599^n)$ [545], by using a smart and tedious case analysis of the neighborhood around a low-degree vertex. The running time of the algorithm has been further improved to $\mathcal{O}^*(1.2346^n)$ [330], and, using combinatorial arguments on connected regular graphs, to $\mathcal{O}^*(1.2108^n)$ [495]. Unfortunately, the latter algorithm needs exponential space. This drawback can be avoided: there is a polynomial-space algorithm with a slightly weaker $\mathcal{O}^*(1.2227^n)$ time complexity [54]. A non-trivial lower bound on the basis of the exponential is still unknown (even under some complexity-theoretic assumptions).

6.1.3 Approximating Maximum Cliques

Since we are apparently not able to compute a maximum clique in moderate time, we could ask up to what size we can recognize cliques in justifiable time. Recall that $\omega(G)$ denotes the size of the largest clique in G. We say that an algorithm approximates $\omega(G)$ within factor $f(n)$ if and only if the algorithm produces, on input G, a clique U in G such that $\omega(G) \leq f(n) \cdot |U|$. Note that, since a maximum clique consists of at most n vertices, we can trivially approximate maximum clique within factor $\mathcal{O}(n)$, simply by outputting some edge, if there is one in the graph. With a lot of work and combinatorial arguments, we arrive at the next theorem [79], which is unfortunateley not very much better than the trivial ratio.

Theorem 6.1.7. *There exists a polynomial-time algorithm whose output, for graph G with n vertices, is a clique of size within factor $\mathcal{O}\left(\frac{n}{(\log n)^2}\right)$ of $\omega(G)$.*

The approximation ratio stated in the theorem is the best known. The following celebrated result [287] indicates that in fact, there is not much space for improving over that ratio.

Theorem 6.1.8. *Unless $\mathcal{NP} = \mathcal{ZPP}$,[4] there exists no polynomial-time algorithm whose output for a graph G with n vertices is a clique of size within factor $n^{1-\varepsilon}$ of $\omega(G)$ for any $\varepsilon > 0$.*

The complexity-theoretic assumption used in the theorem is almost as strong as $\mathcal{P} = \mathcal{NP}$. The inapproximability result has been strengthened to subconstant values of ε, first to $\mathcal{O}\left(\frac{1}{\sqrt{\log \log n}}\right)$ [177] and further to $\mathcal{O}\left(\frac{1}{(\log n)^\gamma}\right)$ [353] for some $\gamma > 0$. These results are based on much stronger complexity assumptions – essentially, that no $\mathcal{NP}$-complete problem can be solved by randomized algorithms with quasi-polynomial running time, i.e., in time $2^{(\log n)^{\mathcal{O}(1)}}$. Note that the ratio

[4] $\mathcal{ZPP}$ is the class of all problems that can be solved with randomized algorithms running in expected polynomial time while making no errors. Such algorithms are also known as (polynomial-time) Las Vegas algorithms.

$\frac{n}{(\log n)^2}$ is expressible as $\Omega\left(\frac{\log\log n}{\log n}\right)$ in terms of ε. The gap between the lower bound and the upper bound for approximability is thus pretty close.

Also many heuristic techniques for finding maximum cliques have been proposed. They often show reasonable behavior, but of course, they cannot improve over the theoretical inapproximability ratio. An extensive discussion of heuristics for finding maximum cliques can be found in [70].

In the random graph model, we known that, with high probability, $\omega(G)$ is either $(2 + o(1)) \log n$ rounded up or rounded down, for a random graph of size n (see, e.g., [24]). There are several polynomial-time algorithms producing cliques of size $(1+o(1)) \log n$, i.e., they achieve an approximation ratio of roughly two [263]. However, it is conjectured that there is no polynomial-time algorithm outputting a clique of size at least $(1 + \varepsilon) \log n$ for any $\varepsilon > 0$ [328, 347].

6.1.4 Finding Fixed-Size Cliques

In many cases, it might be appropriate to search only for cliques of bounded sizes. Technically that is, we consider the clique size not as part of the input. For instance, exhaustive search has running time $\Theta(n^k)$ when the clique size k is fixed. A nice trick helps us to obtain an algorithm for detecting cliques of size three (triangles) faster than $\mathcal{O}(n^3)$. The idea to the algorithm in the following theorem can be found in [321].

Theorem 6.1.9. *There exists an algorithm for testing a graph for triangles that runs in time* $\mathcal{O}(n^{2.376})$.

Proof. Let G be any graph with n vertices. Let $A(G)$ denote the adjacency matrix of G, i.e., entry a_{ij} of $A(G)$ is one if vertices v_i and v_i are adjacent, and zero otherwise. Consider the matrix $A(G)^2 = A(G) \cdot A(G)$ where $\cdot$ is the usual matrix multiplication. The entry b_{ij} of the matrix $A(G)^2$ is exactly the number of walks of length two between v_i and v_j. Suppose there exists an entry $b_{ij} \geq 1$. That is, there is at least one vertex $u \in V$ different to v_i and v_j which is adjacent to both v_i and v_j. If the graph G has an edge $\{v_i, v_j\}$, then we know that G contains the triangle $\{v_i, v_j, u\}$. Thus, an algorithm for triangle-testing simply computes $A(G)^2$ and checks whether there exists an edge $\{v_i, v_j\}$ for some non-zero entry b_{ij} in $A(G)^2$. Since fast square matrix multiplication can be done in time $\mathcal{O}(n^\alpha)$ where $\alpha < 2.376$ [132], the algorithm runs in time $\mathcal{O}(n^{2.376})$. □

Note that for sparse graphs there is an even faster algorithm running in time $\mathcal{O}(m^{\frac{2\alpha}{\alpha+1}}) = \mathcal{O}(m^{1.41})$ for finding triangles which makes use of the same technique [26] (see also Section 11.5).

Once we have reached this point, we would like to apply the matrix-multiplication technique to come up with algorithms for clique size larger than three as well. However, the direct argument does not work for some reasons. For instance, there exists always a walk of length three between adjacent vertices. This makes the matrix $A(G)^3$ and all higher powers ambiguous. We need a more sophisticated approach [174, 440].

Theorem 6.1.10. *For every $k \geq 3$ there exists an algorithm for finding a clique of size k in a graph with n vertices that runs in time $\mathcal{O}(n^{\beta(k)})$ where $\beta(k) = \alpha(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)$ and multiplying an $n^r \times n^s$-matrix with an $n^s \times n^t$-matrix can be done in time $\mathcal{O}(n^{\alpha(r,s,t)})$.*

Proof. Let k_1 denote $\lfloor k/3 \rfloor$, let k_2 denote $\lceil (k-1)/3 \rceil$, and let k_3 denote the value $\lceil k/3 \rceil$. Note that $k = k_1+k_2+k_3$. Let G be any graph with n vertices and m edges. We first construct a tripartite auxiliary graph $\tilde{G}$ as follows: the vertex set $\tilde{V}$ is divided into three sets $\tilde{V}_1, \tilde{V}_2$, and $\tilde{V}_3$ where $\tilde{V}_i$ consists of all cliques of size k_i in G. Define two vertices $U \in \tilde{V}_i$ and $U' \in \tilde{V}_j$ to be adjacent in $\tilde{G}$ if and only if $i \neq j$ and $U \cup U'$ is a clique of size $k_i + k_j$ in G. The algorithm now tests the auxiliary graph $\tilde{G}$ for triangles. If there is such a triangle $\{U_1, U_2, U_3\}$, then the construction of $\tilde{G}$ implies that $U_1 \cup U_2 \cup U_3$ is a clique of size k in G. Testing the graph $\tilde{G}$ for triangles can be done by matrix multiplication as described in Theorem 6.1.9. However, we now have to multiply an $n^{k_1} \times n^{k_2}$ adjacency matrix, representing edges between $\tilde{V}_1$ and $\tilde{V}_2$, with an $n^{k_2} \times n^{k_3}$ adjacency matrix, representing edges between $\tilde{V}_2$ and $\tilde{V}_3$. This step can be done in time $\mathcal{O}(n^{\beta(k)})$. Computing the three matrices needs in the worst case $\mathcal{O}(n^{\max\{k_1+k_2, k_1+k_3, k_2+k_3\}}) = \mathcal{O}(n^{\lceil \frac{2k}{3} \rceil})$, which is asymptotically dominated by the time for the fast rectangular matrix multiplication [318]. □

We give an impression of the algorithmic gain of using matrix multiplication (see, e.g., [260]).

Clique size	Exhaustive search	Matrix multiplication
3	$\mathcal{O}(n^3)$	$\mathcal{O}(n^{2.376})$
4	$\mathcal{O}(n^4)$	$\mathcal{O}(n^{3.376})$
5	$\mathcal{O}(n^5)$	$\mathcal{O}(n^{4.220})$
6	$\mathcal{O}(n^6)$	$\mathcal{O}(n^{4.751})$
7	$\mathcal{O}(n^7)$	$\mathcal{O}(n^{5.751})$
8	$\mathcal{O}(n^8)$	$\mathcal{O}(n^{6.595})$

The theorem has a nice application to the membership counting problem for cliques of fixed size. The following result is due to [260].

Theorem 6.1.11. *For every $k \geq 3$, there exists an algorithm that counts the number of cliques of size k to which each vertex of a graph on n vertices belongs, in time $\mathcal{O}(n^{\beta(k)})$ where $\beta(k)$ is the same function as in Theorem 6.1.10.*

Proof. The theorem is based on the observation that for the case $k = 3$ (see Theorem 6.1.9), it is not only easy to check whether two vertices v_i and v_j belong to some triangle in G, but also to compute in how many triangles they lie: if the edge $\{v_i, v_j\}$ exists in G, then the number is just the entry b_{ij} in the square of the adjacency matrix $A(G)$. In general, we apply this observation to the auxiliary graph $\tilde{G}$. For any vertex $v \in V$, let $C_k(v)$ denote the number of different cliques of size k in G in which v is contained. Similarly, let $\tilde{C}_3(U)$ denote the number of triangles to which node U of $\tilde{G}$ belongs. Notice that U is a clique in G of size smaller than k. Clearly, cliques of G of size k may have many

representations in graph $\tilde{G}$. The exact number is the number of partitionings of a set of cardinality k into sets of cardinalities k_1, k_2, and k_3, i.e., $\binom{k}{k_1,k_2,k_3}$ where k_1, k_2, and k_3 are defined as in the proof of Theorem 6.1.10. Without loss of generality, let k_1 be the minimum of these three parameters. Let $\mathcal{U}(v)$ be the set of all cliques U of size k_1 in G such that $v \in U$. We then obtain the following equation:

$$\sum_{U \in \mathcal{U}(v)} \tilde{C}_3(U) = \binom{(k-1)}{(k_1-1),k_2,k_3} \cdot C_k(v) \tag{6.1}$$

Clearly, using Theorem 6.1.10, the left-hand side of this equation can be computed in time $\mathcal{O}(n^{\beta(k)})$ (first, compute the matrices and second, search entries for all U containing v). We now easily calculate $C_k(v)$ from Equation 6.1. □

A recent study of the corresponding decremental problem [260], i.e., the scenario where starting from a given graph vertices and edges can be removed, has shown that we can save roughly $n^{0.8}$ time compared to computing the number of size-k cliques to which the vertices belong each time from the scratch. For example, the problem of counting triangles in a decremental setting now takes $\mathcal{O}(n^{1.575})$.

Fixed-parameter tractability. A way to study which time bounds we might expect for fixed-parameter clique problems is parameterized complexity [168]. The goal here is to figure out which input parameter makes a problem computationally hard. We say that a parameterized problem (with parameter k) is fixed-parameter tractable if and only if there is an algorithm for the problem that needs time polynomial in input size n, if k is fixed, and which is asymptotically independent of k. More precisely, the time complexity of the algorithm has the form $\mathcal{O}(f(k) \cdot p(n))$ where p is some polynomial independent of k and f is an arbitrary function independent of n. Note that the algorithm above does not satisfy such a bound. A good bound would be, e.g., $\mathcal{O}(k^k \cdot n^2)$. However, we are far from proving such bounds, and in fact, we should not even expect to obtain such algorithms. Let $\mathcal{FPT}$ denote the class of fixed-parameter tractable problems. We know that parameterized Clique is complete for the class $\mathcal{W}[1]$, a superclass of $\mathcal{FPT}$ [167]. However, it is widely believed that $\mathcal{FPT} \neq \mathcal{W}[1]$, which would imply both $\mathcal{P} \neq \mathcal{NP}$ and Clique is not fixed parameter tractable.

6.1.5 Enumerating Maximal Cliques

Enumerative algorithms for the clique problem have some tradition (cf., e.g., [70]), with probably the first appearing already in 1957 [284]. Several other, now classical, algorithms were proposed (e.g., [473, 103]). Most recently, also algorithms for the dynamic graph setting have been investigated [534].

We are interested in having efficient algorithms for enumerating maximal cliques. There are some gradations in the meaning of 'efficient.' Most of the interesting combinatorial problems have an exponential number of configurations; in our case indicated by the $3^{\lceil \frac{n}{3} \rceil}$ matching upper bound for the number of maximal cliques. A typical requirement for an enumerative algorithm to be efficient

is polynomial *total* time. That is, the algorithm outputs all C possible configurations in time bounded by a polynomial in C and the input size n. Exhaustive search is not polynomial total time. In contrast, one of the classical algorithms [473] first runs $\mathcal{O}(n^2C)$ steps with no output and then outputs all C maximal cliques all at once. However, an algorithm for the enumeration of all *maximum* cliques that runs in polynomial total time does not exist, unless $\mathcal{P} = \mathcal{NP}$ [382].

We next review enumerative algorithms for maximal cliques with polynomial total time having some further desirable properties.

Polynomial delay. An algorithm fulfilling this condition generates the configurations, one after the other in some order, in such a way that the delay until the first output, the delay between any two consecutive configurations, and the delay until it stops after the last output is bounded by a polynomial in the input size. For maximal cliques we know such algorithms that in addition, require only linear space [553].

Theorem 6.1.12. *There is an algorithm enumerating all maximal cliques of a graph with polynomial delay $\mathcal{O}(n^3)$ using only $\mathcal{O}(n+m)$ space.*

Proof. We construct a binary tree with n levels and leaves only at level n. Each level is associated with a vertex, i.e., at level i we consider vertex v_i. The nodes of the tree at level i are all maximal cliques of $G[\{v_1, \ldots, v_i\}]$. It immediately follows that the leaves are exactly the maximal cliques of G. Fix level i and maximal clique U in $G[\{v_1, \ldots, v_i\}]$. We want to determine the children of U at level $i+1$. We have two main cases:

1. Suppose all vertices of U are adjacent to v_{i+1} in G. Then $U \cup \{v_{i+1}\}$ is maximal clique in $G[\{v_1, \ldots, v_i, v_{i+1}\}]$. Note that this is the only way to obtain a maximal clique of $G[\{v_1, \ldots, v_i, v_{i+1}\}]$ that contains U. In this case U has only one child in the tree.
2. Suppose there is a vertex in U not adjacent to v_{i+1} in G. Here, we can obtain maximal cliques in $G[\{v_1, \ldots, v_i, v_{i+1}\}]$ in two different ways: U itself is certainly a maximal clique, and another clique is $(U - \overline{N}(v_{i+1})) \cup \{v_{i+1}\}$, where $\overline{N}(v_{i+1})$ are all vertices of G not adjacent to v_{i+1}. If the latter set is a maximal clique, U would have two children. However, as the set $(U - \overline{N}(v_{i+1})) \cup \{v_{i+1}\}$ is potentially a child of several sets, we define it to be the child of the lexicographically smallest set U, if it is maximal.

By this definition, we have a tree where all internal nodes have one or two children, thus a binary tree, and all leaves are at level n.

Our enumerative algorithm now simply traverses the tree using a depth-first search and outputs all leaves. All we need to be able to perform the computation, given a node U of the tree at level i, is the following:

– $\mathsf{Parent}(U, i)$: According to the definition of the tree, the parent node of U is the lexicographically smallest maximal clique in $G[\{v_1, \ldots, v_{i-1}\}]$ containing the clique $U - \{v_i\}$. This is one of our efficiently computable primitives: the set can be computed in time $\mathcal{O}(n+m)$.

– $\mathsf{LeftChild}(U, i)$: If $U \subseteq N(v_{i+1})$ (the first case above), then the left child is $U \cup \{v_{i+1}\}$. If $U \not\subseteq N(v_{i+1})$ (one part of the second case above), then the left child is U. Checking which case has to be applied needs $\mathcal{O}(n+m)$ time.
– $\mathsf{RightChild}(U, i)$: If $U \subseteq N(v_{i+1})$, then there is no right child defined.If $U \not\subseteq N(v_{i+1})$, then the right child of U is $(U - \overline{N}(v_{i+1})) \cup \{v_{i+1}\}$ if it is a maximal clique and $U = \mathsf{Parent}((U - \overline{N}(v_{i+1})) \cup \{v_{i+1}\}, i+1)$, otherwise the right child is not defined. Note that we only need $\mathcal{O}(n+m)$ processing time.

The longest path between any two leaves in the tree is $2n-2$ passing through $2n-1$ nodes. For each node we need $\mathcal{O}(n+m)$ time. Since any subtree of our tree has a leaf at level n, this shows that the delay between outputs is $\mathcal{O}(n^3)$. Note that the algorithm only needs to store while processing a node, the set U, the level i, and a label indicating whether it is the left or the right child. Hence, the amount of space is $\mathcal{O}(n+m)$. □

Specified order. A more difficult problem is generating maximal cliques in a specific order, such as lexicographic order. If we only insist in polynomial total time, this is obviously not a restriction, since we need only collect all outputs and sort them for outputting in lexicographic order. Considering orders is only interesting in the case of polynomial delay. Note that the DFS-based polynomial-delay algorithm in Theorem 6.1.12 does not produce its outputs in lexicographic order. Another DFS-based algorithm [395] has been proposed that produces the outputs in lexicographic order but is not polynomial delay. We first observe that it is not obvious how to break the tradeoff.

Theorem 6.1.13. *Deciding for any given graph G and any maximal clique U of G, whether there is a maximal clique U' lexicographically larger than U, is $\mathcal{NP}$-complete.*

The theorem is proven by a polynomial transformation from Satisfiability [334]. It has some immediate consequences, e.g., it rules out polynomial-delay algorithms with respect to inverse lexicographic order.

Corollary 6.1.14. *1. Unless $\mathcal{P} = \mathcal{NP}$, there is no algorithm that generates for any given graph G and any maximal clique U in G the lexicographically next maximal clique in polynomial time.*
2. Unless $\mathcal{P} = \mathcal{NP}$, there is no algorithm that generates for any given graph all maximal cliques in inverse lexicographic order with polynomial delay.

It might seem surprising that algorithms exist generating all maximal cliques in lexicographic order, with polynomial delay. The idea of such an algorithm is simply that while producing the current output, we invest additional time in producing lexicographically larger maximal cliques. We store these cliques in a priority queue Q. Thus, Q contains a potentially exponential number of cliques and requires potentially exponential space. The following algorithm has been proposed in [334] and uses in a clever way the tree structure employed in Theorem 6.1.12.

Algorithm 9: Lexicographic enumeration of maximal cliques [334]

Input: Graph $G = (V, E)$
Output: Sequence of maximal cliques of G in lexicographic order

```
Let U_0 be the lexicographically first maximal clique;
Insert U_0 into priority queue Q;
while Q is not empty do
    U :=ExtractMin(Q);
    Output U;
    foreach vertex v_j of G not adjacent to some vertex v_i ∈ U with i < j do
        U_j := U ∩ {v_1, ..., v_j};
        if (U_j − N̄(v_j)) ∪ {v_j} is a maximal clique in G[{v_1, ..., v_j}] then
            Let T be the lexicographically smallest maximal clique which
            contains (U_j − N̄(v_j)) ∪ {v_j};
            Insert T into Q
```

Theorem 6.1.15. *Algorithm 9 enumerates all maximal cliques of a graph with n vertices in lexicographic order, and with delay $\mathcal{O}(n^3)$.*

Proof. For the correctness of the algorithm, first observe that the set T being inserted into Q when considering U is lexicographically greater than U. Thus, we store only sets into the queue that have to be output after U. Hence, the sequence of maximal cliques we produce is indeed lexicographically ascending. We also have to show that all maximal cliques are in the sequence. We do this by proving inductively: if U is the lexicographically first maximal clique not yet output, then U is in Q.

Base of induction: Suppose $U = U_0$. Then the statement is obviously true.

Step of induction: Suppose U is lexicographically greater than U_0. Let j be the largest index such that $U_j = U \cap \{v_1, \ldots, v_j\}$ is *not* a maximal clique in the graph restricted to vertices $v_1, \ldots, v_j$. Such an index must exist, since otherwise we would have $U = U_0$. Moreover, we have that $j < n$, since U is a maximal clique in the whole graph G. By maximality of j, we must have $v_{j+1} \in U$. There exists a non-empty set S such that $U_j \cup S$ is a maximal clique in $G[\{v_1, \ldots, v_j\}]$. Again, by maximality of j, the vertex v_{j+1} is not adjacent to all vertices in S. We conclude that there is a maximal clique U' containing $U_j \cup S$ but not vertex v_{j+1}. Note that U' is lexicographically smaller than U, since they differ on set S. By induction hypothesis, U' has already been output. At the time when U' was output, the vertex v_{j+1} was found not to be adjacent to some vertex v_i in U' with index $i < j + 1$. Clearly, we have $(U'_{j+1} - \overline{N}(v_{j+1})) \cup \{v_{j+1}\} = U_{j+1}$ and U_{j+1} is a maximal clique in $G[\{v_1, \ldots, v_{j+1}\}]$. So the lexicographically first maximal clique T containing U_{j+1} was inserted into Q. Once more by maximality of j, U and T coincide on the first $j + 1$ vertices. Assume that $U \neq T$. Let k be the first index such that U and T disagree on v_k. It follows that $k > j + 1$. Since T is lexicographically less than U, we have $v_k \in T$ and $v_k \notin U$. Hence, U_k is not a maximal clique in $G[\{v_1, \ldots, v_k\}]$, a contradiction to maximality of j. Therefore, $U = T$ and so U is in Q. This proves the induction step.

For the time bound, the costly operations are the extraction of the lexicographically smallest maximal clique from Q (which needs $\mathcal{O}(n \log C)$), the n computations of maximal cliques containing a given set (which takes $\mathcal{O}(n+m)$ for each set), and attempting to insert a maximal clique into Q (at costs $\mathcal{O}(n \log C)$ per clique). Since $C \leq 3^{\lceil \frac{n}{3} \rceil}$, the total delay is $\mathcal{O}(n^3)$ in the worst case. □

Counting complexity. We conclude this section with some remarks on the complexity of counting the number of maximal cliques. An obvious way to count maximal cliques is to enumerate them with some of the above-mentioned algorithms and increment a counter each time a clique is output. This, however, would take exponential time. The question is whether it is possible to compute the number more directly and in time polynomial in the graph size. To study such issues the class $\#\mathcal{P}$ has been introduced [559], which can be considered as the class of all functions counting the number of solutions of instances of $\mathcal{NP}$-problems. It can be shown that counting the number of maximal cliques is $\#\mathcal{P}$-complete (with respect to an appropriate reducibility notion) [560]. An immediate consequence is that if there is a polynomial-time algorithm for computing the number of maximal cliques, then Clique is in $\mathcal{P}$, and thus, $\mathcal{P} = \mathcal{NP}$. Note that in the case of planar, bipartite or bounded-degree graphs there are polynomial-time algorithms for counting maximal cliques [557].

6.2 Structurally Dense Groups

We review two relaxations of the clique concept based on minimal degrees [515, 514, 513]. Both relaxations are structural, as they impose universal constraints on individuals in a group.

6.2.1 Plexes

We generalize the clique concept by allowing members in a group to miss some ties with other group members, but only up to a certain number $N \geq 1$. This leads to the notion of an N-plex [514, 511].

Definition 6.2.1. *Let $G = (V, E)$ be any undirected graph and let $N \in \{1, \ldots, n-1\}$ be a natural number. A subset $U \subseteq V$ is said to be an N-*plex *if and only if $\delta(G[U]) \geq |U| - N$.*

Clearly, a clique is simply a 1-plex, and an N-plex is also an $(N+1)$-plex. We say that a subset $U \subseteq V$ is a maximal N-plex if and only if U is an N-plex and it is not strictly contained in any larger N-plex of G. A subset $U \subseteq V$ is a maximum N-plex if and only if U has a maximum number of vertices among all N-plexes of G.

It is easily seen that any subgraph of an N-plex is also an N-plex, that is, N-plexes are closed under exclusion. Moreover, we have the following relation between the size of an N-plex and its diameter [514, 189, 431].

Proposition 6.2.2. *Let $N \in \{1, \ldots, n-1\}$ be a natural number. Let $G = (V, E)$ be any undirected graph on n vertices.*

1. *If V is an N-plex with $N < \frac{n+2}{2}$, then $\operatorname{diam}(G) \leq 2$ and, if additionally $n \geq 4$, G is 2-edge-connected.*
2. *If V is an N-plex with $N \geq \frac{n+2}{2}$ and G is connected, then $\operatorname{diam}(G) \leq 2N - n + 2$.*

Proof. 1. Suppose $N < \frac{n+2}{2}$. Let $u, v \in V$ be vertices such that $u \neq v$. If u and v are adjacent, the distance between them is one. Now, suppose u and v are not adjacent. Assume that the distance between u and v is at least three, i.e., with respect to neighborhoods it holds $N(u) \cap N(v) = \emptyset$. We obtain

$$n - 2 \geq |N(u) \cup N(v)| \geq 2\delta(G) \geq 2(n - N) > 2\left(n - \frac{n+2}{2}\right) = n - 2,$$

a contradiction. Thus, the distance between u and v is at most two. Hence, $\operatorname{diam}(G) \leq 2$. To verify that for $n \geq 4$, G is 2-edge-connected, assume to the contrary that there is a bridge, i.e., an edge e such that after removing it, $G - \{e\}$ consists of two connected components V_1 and V_2. Obviously, every shortest path from a vertex in V_1 to a vertex in V_2 must use that bridge. Since $\operatorname{diam}(G) \leq 2$, one component is a singleton. This implies that the vertex in this component has degree one. However, as V is an N-plex with $n \geq 4$ vertices, we obtain for the degree of this vertex $n - N > n - (n+2)/2 = (n-2)/2 \geq 1$, a contradiction. Thus, a bridge cannot exist in G.

2. Suppose $N \geq \frac{n+2}{2}$. Let $\{v_0, v_1, \ldots, v_r\}$ be the longest shortest path of G, i.e., a path that realizes the diameter r. We may suppose that $r \geq 4$. Since there is no shorter path between v_0 and v_r, we have that v_i is not adjacent to $v_0, \ldots, v_{i-2}, v_{i+2}, \ldots, v_r$ for all $i \in \{0, \ldots, r\}$ (where vertices with negative index do not exist). Furthermore, there cannot exist a vertex adjacent to both v_0 and v_3. Thus, the following inclusion is true:

$$\{v_0\} \ \cup \ \{v_2, v_3, \ldots, v_r\} \ \cup \ (N(v_3) - \{v_2, v_4\}) \ \subseteq \ \overline{N}(v_0)$$

Note that we have a disjoint union on the left-hand side. We thus obtain the inequality $1 + (r-1) + d_G(v_3) - 2 \leq N$. It follows $r + n - N - 2 \leq N$. Hence, $\operatorname{diam}(G) = r \leq 2N - n + 2$. □

From a computational point of view, finding maximum plexes is not easier than finding maximum cliques. This is immediate when we consider the variable decision problem for plexes, where the problem instance consists of graph G, the size parameter k, and the plex parameter N. Since Clique appears as instances of the form $(G, k, 1)$, the problem is $\mathcal{NP}$-complete. We discuss the complexity of finding N-plexes of certain sizes for fixed N. For any natural number $N > 0$, we define the following decision problem:

Problem:	N-Plex
Input:	Graph G, Parameter $k \in \mathbb{N}$
Question:	Does there exist an N-plex of size at least k within G?

As 1-Plex = Clique, and thus 1-Plex is $\mathcal{NP}$-complete, it is not surprising that finding maximum N-plexes is $\mathcal{NP}$-hard for all $N > 0$ as well.

Theorem 6.2.3. *N-Plex is $\mathcal{NP}$-complete for all natural numbers $N > 0$.*

Proof. It suffices to consider the case $N > 1$. There is a generic proof of the theorem which is based on the fact that being an N-plex is a hereditary graph property (see, e.g., [240]). We give a direct proof in order to demonstrate the structural similarity between cliques and plexes. We describe a polynomial transformation of Clique into N-Plex. Let (G, k) be any instance of the clique problem. We construct a new graph G' in the following way: we take $N-1$ copies of each vertex of G, connect them to each other by an edge, and all new vertices to the vertices of G except to the original one. More specifically, let $G' = (V', E')$ be the graph defined as follows:

$$\begin{aligned} V' &=_{\text{def}} V \times \{0, 1, \ldots, N-1\} \\ E' &=_{\text{def}} \left\{ \{(u,0),(v,0)\} \mid \{u,v\} \in E \right\} \cup \\ &\qquad \cup \left\{ \{(u,i),(v,j)\} \mid u,v \in V \text{ and } i,j > 0 \right\} \cup \\ &\qquad \cup \left\{ \{(u,0),(v,i)\} \mid u,v \in V \text{ with } u \neq v \text{ and } i > 0 \right\} \end{aligned}$$

The graph G' can certainly be computed in time polynomial in the size of G. A crucial observation is that copy vertices, i.e., vertices in $V \times \{1, \ldots, N-1\}$ are adjacent to all vertices in V' except one. We will show that G contains a clique of size k if and only if G' contains an N-plex of size $k + (N-1)n$.

Suppose there exists a clique $U \subseteq V$ of size exactly k in G. Let U' denote the vertex set in G' consisting of all original vertices of U and all copies of vertices of V, i.e., $U' = U \times \{0\} \cup V \times \{1, \ldots, N-1\}$. Notice that U' has cardinality $k + (N-1)n$. Each vertex with label $i \in \{1, \ldots, N-1\}$ is directly connected to each other vertex in U' except one vertex with label zero, thus has degree $|U'| - 2 = k + (N-1)n - 2$. Each vertex $(u, 0)$ is adjacent to all vertices in U' except (u, i) with $i > 0$. That is, $(u, 0)$ has degree $k + (N-1)n - 1 - (N-1)$. Hence, U' is an N-plex.

Suppose there is no clique of size k in G. Thus, any induced subgraph of G having $k' \geq k$ vertices has minimal degree at most $k' - 2$. Let $U \subseteq V'$ be any vertex set with $k + (N-1)n$ vertices. Then there is another set $U' \subseteq V'$ on $k + (N-1)n$ vertices such that $\delta(G'[U']) \geq \delta(G'[U])$ and U' contains all copy vertices of G', i.e., $U' \supseteq V \times \{1, \ldots, N-1\}$. This follows from the fact that there is always a vertex in $U_0 = U \cap (V \times \{0\})$ that is not adjacent to some other vertex in U_0 (otherwise U_0 would induce a clique of size $|U_0| \geq k$ in G). Remembering the observation above, we are now allowed to recursively exchange such vertices by vertices of $V \times \{1, \ldots, N-1\}$ as long as possible, without decreasing minimum degrees. We end up with a desired set $U' \subseteq V'$. Since we have no size-k clique in G, we may conclude $\delta(G'[U]) \leq \delta(G'[U']) \leq k + (N-1)n - 2 - (N-1)$. Hence, there is no N-plex in G'. □

6.2.2 Cores

A concept dual to plexes is that of a core. Here, we do not ask how many edges are missing in the subgraph for being complete, but we simply fix a threshold in terms of a minimal degree for each member of the subgroup. One of the most important things to learn about cores is that there exist polynomial-time algorithms for finding maximum cores. Cores have been introduced in [513].

Definition 6.2.4. *Let $G = (V, E)$ be any undirected graph. A subset $U \subseteq V$ is said to be an* N-core *if and only if $\delta(G[U]) \geq N$.*

The parameter N of an N-core is the order of the N-core. A subset $U \subseteq V$ is a maximal N-core if and only if U is an N-core and it is not strictly contained in any larger N-core of G. A subset $U \subseteq V$ is a maximum N-core if and only if U has maximum number of vertices among all N-cores of G. Maximum cores are also known as main cores.

Any $(N+1)$-core is an N-core and any N-core is an $(n-N)$-plex. Moreover, if U and U' are N-cores, then $U \cup U'$ is an N-core as well. That means maximal N-cores are unique. However, N-cores are not closed under exclusion and are in general not nested. As an example, a cycle is certainly a 2-core but any proper subgraph has at least one vertex with degree less than two. N-cores need not be connected. The following proposition relates maximal connected N-cores to each other.

Proposition 6.2.5. *Let $G = (V, E)$ be any undirected graph and let $N > 0$ be any natural number. Let U and U' be maximal connected N-cores in G with $U \neq U'$. Then there exists no edge between U and U' in G.*

Proof. Assume there is an edge $\{u, v\}$ with $u \in U$ and $v \in U'$. It follows that $U \cup U'$ is an N-core containing both U and U'. Furthermore, it is connected, since U and U' are connected. □

Some immediate consequences of the proposition are the following: the unique maximum N-core of a graph is the union of all its maximal connected N-cores, the maximum 2-core of a connected graph is connected (notice that the internal vertices of a path have degree two), and a graph is a forest if and only if it possesses no 2-cores. The next result is an important algorithmic property of N-cores, that was exhibited in [46].

Proposition 6.2.6. *Let $G = (V, E)$ be any undirected graph and let $N > 0$ be any natural number. If we recursively remove all vertices with degree strictly less than N, and all edges incident with them, then the remaining set U of vertices is the maximum N-core.*

Proof. Clearly, U is an N-core. We have to show that it is maximum. Assume to the contrary, the N-core U obtained is not maximum. Then there exists a non-empty set $T \subseteq V$ such that $U \cup T$ is the maximum N-core, but vertices of T have been removed. Let t be the first vertex of T that has been removed. At that time, the degree of t must have been strictly less than N. However, as t has

at least N neighbors in $U \cup T$ and all other vertices have still been in the graph when t was removed, we have a contradiction. □

The procedure described in the proposition suggests an algorithm for computing N-cores. We extend the procedure for obtaining auxiliary values which provide us with complete information on the core decomposition of a network. Define the core number of a vertex $v \in V$ to be the highest order N of a maximum N-core vertex v belongs to, i.e.,

$$\xi_G(v) =_{\text{def}} \max\{\ N \mid \text{there is an } N\text{-core } U \text{ in } G \text{ such that } v \in U\ \}.$$

A method, according to [47], for computing all core numbers is shown in Algorithm 10. The algorithm is correct due to the following reasons: any graph G is certainly a $\delta(G)$-core, and each neighbor of vertex v having lower degree than v decrements the potential core number of v. A straightforward implementation of the algorithm yields a worst-case time bound of $\mathcal{O}(mn \log n)$ – the most costly operations being sorting vertices with respect to their degree. A more clever implementation guarantees linear time [47].

Algorithm 10: Computing core numbers [47]

Input: Graph $G = (V, E)$
Output: Array ξ_G containing the core numbers of all vertices in G

```
Compute the degrees of all vertices and store them into D;
Sort V in increasing degree-order D;
foreach v ∈ V in sorted order do
    ξ_G(v):=D[v];
    foreach vertex u adjacent to v do
        if D[u] > D[v] then
            D[u] := D[u] − 1;
            Resort V in increasing degree-order of D
```

Theorem 6.2.7. *There is an implementation of Algorithm 10 that computes the core numbers of all vertices in a given graph $G = (V, E)$ having n vertices and m edges in time $\mathcal{O}(n + m)$.*

Proof. To reduce the running time of the algorithm, we have to speed up the sorting operations in the algorithm. This can be achieved by two techniques.

1. Since the degree of a vertex lies in the range $\{0, \ldots, n-1\}$, we do sorting using n buckets, one for each vertex degree. This gives us an $\mathcal{O}(n)$ time complexity for initially sorting the vertex-set array V.
2. We can save resorting entirely, by maintaining information about where in the array V, which contains the vertices in ascending order of their degree, a new region with higher degree starts. More specifically, we maintain an array

J where entry $J[i]$ is the minimum index j such that for all $r \geq j$, vertex $V[r]$ has degree at least i. We can now replace the 'resort'-line in Algorithm 10 by the following instructions:

> **if** $u \neq$ vertex w at position $J[D[u]+1]$ **then** swap vertices u and w in V; Increment $J[D[u]+1]$

Resorting the array V in order to maintain the increasing order of degrees now takes $\mathcal{O}(1)$ time. Notice that the array J can initially be computed in time $\mathcal{O}(n)$.

For the total running time of Algorithm 10, we now obtain $\mathcal{O}(n)$ for initializing and sorting and $\mathcal{O}(m)$ for the main part of the algorithm (since each edge is handled at most twice). This proves the $\mathcal{O}(n+m)$ implementation. □

Corollary 6.2.8. *For all $N > 0$, the maximum N-core for a graph with n vertices and m edges can be computed in time $\mathcal{O}(n+m)$, which is independent of N.*

6.3 Statistically Dense Groups

In general, statistical measures over networks do not impose any universal structural requirements on individuals. This makes them more flexible than structural measures but usually harder to analyze. We turn to statistical measures for densities of graphs.

6.3.1 Dense Subgraphs

The natural notion of density of a graph is the following. Let $G = (V, E)$ be any undirected graph with n vertices and m edges. The density $\varrho(G)$ of G is the ratio defined as

$$\varrho(G) =_{\text{def}} \frac{m}{\binom{n}{2}}.$$

That is, the density of a graph is the percentage of the number of edges of a clique, observable in a graph. We are interested in subgraphs of certain densities.

Definition 6.3.1. *Let $G = (V, E)$ be an undirected graph and let $0 \leq \eta \leq 1$ be a real number. A subset $U \subseteq V$ is said to be an* η-dense subgraph *if and only if $\varrho(G[U]) \geq \eta$.*

In an η-dense subgraph, the interpretation is that any two members share with probability (or frequency) at least η a relationship with each other. It is, however, immediate that even graphs of fairly high density are allowed to have isolated vertices.

A clique, as the subgraph with highest density, is a 1-dense subgraph. An N-plex has density $1 - \frac{N-1}{n-1}$. Thus, for n approaching infinity, the density of an N-plex approaches one. A little bit more exactly, for all $N > 0$ and for all

$0 \leq \eta \leq 1$, an N-plex of size at least $\frac{N-\eta}{1-\eta}$ is an η-dense subgraph. But evidently, not every $(1 - \frac{N-1}{n-1})$-dense subgraph (when allowing non-constant densities) is an N-plex. An N-core is an $\frac{N}{n-1}$-dense subgraph, which can have a density arbitrarily close to zero for large n.

In general, η-dense subgraphs are not closed under exclusion. However, they are nested.

Proposition 6.3.2. *Let $0 \leq \eta \leq 1$ be real number. An η-dense subgraph of size k in a graph G contains an η-dense subgraph of size $k-1$ in G.*

Proof. Let U be any η-dense subgraph of G, $|U| = k$. Let m_U denote the number of edges in $G[U]$. Let v be a vertex with minimal degree in $G[U]$. Note that $\delta(G[U]) \leq \bar{d}(G[U]) = \frac{2m_U}{k} = \varrho(G[U])(k-1)$. Consider the subset U' obtained by excluding vertex v from U. Let $m_{U'}$ denote the number of edges of U'. We have

$$m_{U'} = m_U - \delta(G[U]) \geq \varrho(G[U])\binom{k}{2} - \varrho(G[U])(k-1) = \varrho(G[U])\binom{k-1}{2}$$

Hence, $\varrho(G[U']) \geq \varrho(G[U]) \geq \eta$. Thus, U' is an η-dense subgraph. □

The proposition suggests a greedy approach for obtaining η-dense graphs: recursively deleting a vertex with minimal degree until an η-dense subgraph remains. However, this procedure can fail drastically. We will discuss this below.

Walks. The density averages over edges in subgraphs. An edge is a walk of length one. A generalization of density can involve walks of larger length. To make this more precise, we introduce some notations. Let $G = (V, E)$ be any undirected graph with n vertices. Let $\ell \in \mathbb{N}$ be any walk-length. For a vertex $v \in V$, we define its degree of order ℓ in G as the number of walks of length ℓ that start in v. Let $d^\ell_G(v)$ denote v's degree of order ℓ in G. We set $d^0_G(v) = 1$ for all $v \in V$. Clearly, $d^1_G(v)$ is the degree of v in G. The number of walks of length ℓ in a graph G is denoted by $W_\ell(G)$. We have the following relation between the degrees of higher order and the number of walks in a graph.

Proposition 6.3.3. *Let $G = (V, E)$ be any undirected graph. For all $\ell \in \mathbb{N}$ and for all $r \in \{0, \ldots, \ell\}$, $W_\ell(G) = \sum_{v \in V} d^r_G(v) \cdot d^{\ell-r}_G(v)$.*

Proof. Any walk of length ℓ consists of vertices $v_0, v_1, \ldots, v_\ell$. Fix an arbitrary $r \in \{0, \ldots, \ell\}$. Consider the element v_r. Then the walk $v_0, v_1, \ldots, v_r$ contributes to the degree of order r of v_r, and the walk $v_r, v_{r+1}, \ldots, v_\ell$ contributes to the degree of order $\ell - r$ of v_r. Thus, there are $d^r_G(v_r) \cdot d^{\ell-r}_G(v_r)$ walks of length ℓ having vertex v_r at position r. Summing over all possible choices of a vertex at position r shows the statement. □

It is clear that the maximum number of walks of length ℓ in a graph with n vertices is $n(n-1)^\ell$. We thus define the density of order ℓ of a graph G as

$$\varrho_\ell(G) =_{\text{def}} \frac{W_\ell(G)}{n(n-1)^\ell}.$$

Note that $\varrho_1(G) = \varrho(G)$ as in $W_1(G)$ each edge counts twice. We easily conclude the following proposition.

Proposition 6.3.4. *It holds $\varrho_\ell(G) \leq \varrho_{\ell-1}(G)$ for all graphs G and all natural numbers $\ell \geq 2$.*

Proof. Let $G = (V, E)$ be any undirected graph with n vertices. By Proposition 6.3.3, $W_\ell(G) = \sum_{v \in V} d^1_G(v) \cdot d^{\ell-1}_G(v) \leq (n-1) \sum_{v \in V} d^{\ell-1}_G(v) = (n-1) \cdot W_{\ell-1}(G)$. Now, the inequality follows easily. □

For a graph $G = (V, E)$ we can define a subset $U \subseteq V$ to be an *η-dense subgraph of order ℓ* if and only if $\varrho_\ell(G[U]) \geq \eta$. From the proposition above, any η-dense subgraph of order ℓ is an η-dense subgraph of order $\ell - 1$ as well. The η-dense subgraphs of order $\ell \geq 2$ inherit the property of being nested from the η-dense subgraphs. If we fix a density and consider dense subgraphs of increasing order, then we can observe that they become more and more similar to cliques. A formal argument goes as follows. Define the density of infinite order of a graph G as

$$\varrho_\infty(G) =_{\text{def}} \lim_{\ell \to \infty} \varrho_\ell(G).$$

The density of infinite order induces a discrete density function due to the following zero-one law [307].

Theorem 6.3.5. *Let $G = (V, E)$ be any undirected graph.*

1. *It holds that $\varrho_\infty(G)$ is either zero or one.*
2. *V is a clique if and only if $\varrho_\infty(G) = 1$.*

The theorem says that the only subgroup that is η-dense for some $\eta > 0$ and for all orders, is a clique. In a sense, the order of a density functions allows a scaling of how important compactness of groups is in relation to density.

Average degree. One can easily translate the density of a graph with n vertices into its average degree (as we did in the proof of Proposition 6.3.2): $\bar{d}(G) = \varrho(G)(n-1)$. Technically, density and average degree are interchangeable (with appropriate modifications). We thus can define dense subgraphs alternatively in terms of average degrees. Let $N > 0$ be any rational number. An *N-dense subgraph* of a graph $G = (V, E)$ is any subset $U \subseteq V$ such that $\bar{d}(G[U]) \geq N$. Clearly, an η-dense subgraph (with respect to percentage densities) of size k is an $\eta(k-1)$-dense subgraph (with respect to average degrees), and an N-dense subgraph (with respect to average degrees) of size k is an $\frac{N}{k-1}$-dense subgraph (with respect to percentage densities). Any N-core is an N-dense subgraph. N-dense subgraphs are neither closed under exclusion nor nested. This is easily seen by considering N-regular graphs (for $N \in \mathbb{N}$). Removing some vertices decreases the average degree strictly below N. However, average degrees allow a more fine-grained analysis of network structure. Since a number of edges quadratic

in the number of vertices is required for a graph to be denser than some given percentage threshold, small graphs are favored. Average degrees avoid this pitfall.

Extremal graphs. Based upon Turán's theorem (see Theorem 6.1.2), a whole new area in graph theory has emerged which has been called extremal graph theory (see, e.g., [66]). It studies questions like the following: how many edges may a graph have such that some of a given set of subgraphs are not contained in the graph? Clearly, if we have more edges in the graph, then all these subgraphs must be contained in it. This has been applied to dense subgraphs as well. The following classical theorem due to Dirac [156] is a direct strengthening of Turán's theorem.

Theorem 6.3.6 (Dirac, 1963). *Let $G = (V, E)$ be any undirected graph. If $m > \frac{n^2}{2} \cdot \frac{k-2}{k-1}$, then G contains subgraphs of size $k + r$ having average degree at least $k + r - 1 - \frac{r}{k+r}$ for all $r \in \{0, \ldots, k-2\}$ and $n \geq k + r$.*

Notice that the case $r = 0$ corresponds to the existence of size-k cliques as expressed by Turán's theorem. In many cases, only asymptotic estimations are possible. For example, it can be shown that, for a graph $G = (V, E)$ on n vertices and m edges, if $m = \omega\left(n^{2-\sqrt{\frac{2}{d \cdot k}}}\right)$, then G has a subgraph with k vertices and average degree d [368, 262]. It follows that to be sure that there are reasonably dense subgraphs of sizes not very small, the graph itself has to be reasonably dense. Some more results are discussed in [262].

6.3.2 Densest Subgraphs

We review a solution for computing a densest subgraph with respect to average degrees. Let $\gamma^*(G)$ be the maximum average degree of any non-empty induced subgraph of G, i.e.,

$$\gamma^*(G) =_{\text{def}} \max\{\ \bar{d}(G[U]) \mid\ U \subseteq V \text{ and } U \neq \emptyset\ \}.$$

As in the case of N-cores, the maximal subgraph realizing $\gamma^*(G)$ is uniquely determined. We consider the following problem:

Problem:	Densest Subgraph
Input:	Graph G
Output:	A vertex set of G that realizes $\gamma^*(G)$

This problem can be solved in polynomial time using flow techniques [477, 248, 239]; our proof is from [248].

Theorem 6.3.7. *There is an algorithm for solving* Densest Subgraph *on graphs with n vertices and m edges in time $\mathcal{O}(mn(\log n)(\log \frac{n^2}{m}))$.*

Proof. We formulate DENSEST SUBGRAPH as a maximum flow problem depending on some parameter $\gamma \in \mathbb{Q}^+$. Let $G = (V, E)$ be any undirected graph with n vertices and m edges. Consider a flow network consisting of graph $G' = (V', E')$ and capacity function $u_\gamma : E' \to \mathbb{Q}^+$ given as follows. Add to V a source s and a sink t; replace each edge of G (which is undirected) by two directed edges of capacity one each; connect the source to all vertices of V by an edge of capacity m; and connect each vertex $v \in V$ to the sink by an edge of capacity $m+\gamma-d_G(v)$. More specifically, the network is defined as

$$\begin{aligned} V' &=_{\text{def}} V \cup \{s, t\} \\ E' &=_{\text{def}} \{(v, w) \mid \{v, w\} \in E\} \cup \{(s, v) \mid v \in V\} \cup \{(v, t) \mid v \in V\} \end{aligned}$$

and for $v, w \in V'$ the capacity function u_γ is defined as

$$u_\gamma(v, w) =_{\text{def}} \begin{cases} 1 & \text{if } \{v, w\} \in E \\ m & \text{if } v = s \\ m + \gamma - d_G(v) & \text{if } w = t \\ 0 & \text{if } (v, w) \notin E' \end{cases}$$

We consider capacities of cuts in the network. Let S, T be any partitioning of V' into two disjoint vertex sets with $s \in S$ and $t \in T$, $S_+ = S - \{s\}$ and $T_+ = T - \{t\}$. Note that $S_+ \cup T_+ = V$. If $S_+ = \emptyset$, then the capacity of the cut is $c(S, \overline{S}) = m|V|$. Otherwise we obtain:

$$\begin{aligned} c(S, T) &= \sum_{v \in S, w \in T} u_\gamma(v, w) \\ &= \sum_{w \in T_+} u_\gamma(s, w) + \sum_{v \in S_+} u_\gamma(v, t) + \sum_{v \in S_+, w \in T_+} u_\gamma(v, w) \\ &= m|T_+| + \left(m|S_+| + \gamma|S_+| - \sum_{v \in S_+} d_G(v) \right) + \sum_{\substack{v \in S_+, w \in T_+ \\ \{v, w\} \in E}} 1 \\ &= m|V| + |S_+| \left(\gamma - \frac{1}{|S_+|} \left(\sum_{v \in S_+} d_G(v) - \sum_{\substack{v \in S_+, w \in T_+ \\ \{v, w\} \in E}} 1 \right) \right) \\ &= m|V| + |S_+|(\gamma - \bar{d}(G[S_+])) \end{aligned} \tag{6.2}$$

It is clear from this equation that γ is our guess on the maximum average degree of G. We need to know how we can detect whether γ is too big or too small. We prove the following claim.

Claim. Let S and T be sets that realize the minimum capacity cut, with respect to γ. Then we have the following:

1. If $S_+ \neq \emptyset$, then $\gamma \leq \gamma^*(G)$.
2. If $S_+ = \emptyset$, then $\gamma \geq \gamma^*(G)$.

The claim is proven by the following arguments.

1. Suppose $S_+ \neq \emptyset$. Since $c(\{s\}, V' - \{s\}) = m|V| \geq c(S,T)$, we have $|S_+|(\gamma - \bar{d}(G[S_+])) \leq 0$. Hence, $\gamma \leq \bar{d}(G[S_+]) \leq \gamma^*(G)$.
2. Suppose $S_+ = \emptyset$. Assume further to the contrary, that $\gamma < \gamma^*(G)$. Let $U \subseteq V$ be any non-empty vertex subset satisfying $\bar{d}(G[U]) = \gamma^*(G)$. By Equation 6.2, we obtain

$$c(U \cup \{s\}, \overline{U} \cup \{t\}) = m|V| + |U|(\gamma - \gamma^*(G)) < m|V| = c(S,T),$$

a contradiction to the minimality of the cut capacity $c(S,T)$. Thus, $\gamma \geq \gamma^*(G)$.

The claim suggests an algorithm for finding the right guess for γ by binary search. Notice that $\gamma^*(G)$ can have only a finite number of values, i.e.,

$$\gamma^*(G) \in \left\{ \frac{2i}{j} \;\middle|\; i \in \{0,\ldots,m\} \text{ and } j \in \{1,\ldots,n\} \right\}.$$

It is easily seen that the smallest possible distance between two different points in the set is $\frac{1}{n(n-1)}$. A binary search procedure for finding a maximum average degree subgraph is given as Algorithm 11.

Algorithm 11: Densest subgraph by min-cut and binary search [248]

Input: Graph $G = (V, E))$
Output: A set of k vertices of G

Initialize $l := 0$, $r := m$, and $U := \emptyset$;
while $r - l \geq \frac{1}{n(n-1)}$ **do**
  $\gamma := \frac{l+r}{2}$;
  Construct flow network (V', E', u_γ);
  Find minimum cut S and T of the flow network;
  if $S = \{s\}$ **then**
    $r := \gamma$
  else
    $l := \gamma$;
    $U := S - \{s\}$
Return U

For a time bound, note that we execute the iteration $\lceil \log((m+1)n(n-1)) \rceil = \mathcal{O}(\log n)$ times. Inside each iteration we have to run an algorithm which finds a minimum capacity cut. If we use, e.g., the push-relabel algorithm [252] for max-flow computations, we can do this in time $\mathcal{O}(nm \log \frac{n^2}{m})$ in a network with n vertices and m edges. Out network has $n+2$ vertices and $2m+2n$ edges. This does not change the complexity of the max-flow algorithm asymptotically. We thus obtain the overall time bound $\mathcal{O}(nm(\log n)(\log \frac{n^2}{m}))$. □

Parametric maximum flow algorithms [239, 6] have been employed to improve the time bound to $\mathcal{O}(nm \log \frac{n^2}{m})$ [239]. In [113], Densest Subgraph has been solved by linear programming. This gives certainly a worse upper bound for the time complexity, but has some extensions to the case of directed graphs.

Directed graphs. There is no obvious way to define the notion of density in directed graphs. Since average in-degree and average out-degree in a directed graph are always equal, both measures are not sensitive to orientedness. One approach followed in the literature [342, 113] is based on considering two vertex sets S and T, which are not necessarily disjoint, to capture orientations. For any directed graph $G = (V, E)$ and non-empty sets $S, T \subseteq V$, let $E(S, T)$ denote the set of edges going from S to T, i.e., $E(S, T) = \{(u, v) \mid u \in S \text{ and } v \in T\}$. We define an average degree of the pair (S, T) in the graph as [342]:

$$\bar{d}_G(S, T) =_{\text{def}} \frac{|E(S, T)|}{\sqrt{|S| \cdot |T|}}.$$

This notion was introduced to measure the connectedness between hubs and authorities in web graphs. The set S is understood as the set of hubs, and the set T is understood as the set of authorities in the sense of [359], or fans and centers in the sense of [376]. If $S = T$ then $\bar{d}_G(S, T)$ is precisely the average degree of $G[S]$ (i.e., the sum of the average in-degree and the average out-degree of $G[S]$). The maximum average degree for a directed graph $G = (V, E)$ is defined as

$$\gamma^*(G) =_{\text{def}} \max\{\ \bar{d}_G(S, T) \mid S, T \subseteq V \text{ and } S \neq \emptyset,\ T \neq \emptyset\ \}.$$

Densest Subgraph on directed graphs can be solved in polynomial time by linear programming [113]. To do so, we consider the following LP relaxations LP_γ, where γ ranges over all possible ratios $|S|/|T|$:

$$\begin{aligned} \max\ & \textstyle\sum_{(u,v)\in E} x_{(u,v)} \\ s.t.\ & x_{(u,v)} \leq s_u \text{ for all } (u, v) \in E \\ & x_{(u,v)} \leq t_v \text{ for all } (u, v) \in E \\ & \textstyle\sum_{u\in V} s_u \leq \sqrt{\gamma} \\ & \textstyle\sum_{v\in V} t_v \leq \frac{1}{\sqrt{\gamma}} \\ & x_{(u,v)}, s_u, t_v \geq 0 \text{ for all } u, v \in V \text{ and } (u, v) \in E \end{aligned}$$

It can be shown that the maximum average degree for G is the maximum of the optimal solutions for LP_γ over all γ. Each linear program can be solved in polynomial time. Since there are $\mathcal{O}(n^2)$ many ratios for $|S|/|T|$ and thus for γ, we can now compute the maximum average degree for G (and a corresponding subgraph as well) in polynomial time by binary search.

6.3.3 Densest Subgraphs of Given Sizes

The densest subgraph of a graph is highly fragile, as a graph with some average degree need not possess a subgraph with the same average degree. We are thus

not able to deduce easily information on the existence of subgraphs with certain average degrees *and* certain sizes, from a solution of Densest Subgraph. We discuss this problem independently. For an undirected graph $G = (V, E)$ and parameter $k \in \mathbb{N}$, let $\gamma^*(G, k)$ denote the maximum value of the average degrees of all induced subgraphs of G having k vertices, i.e.,

$$\gamma^*(G, k) =_{\text{def}} \max\{\ \bar{d}(G[U]) \mid\ U \subseteq V \text{ and } |U| = k\ \}.$$

The following optimization problem has been introduced in [201]:

Problem: Dense k-Subgraph
Input: Graph G, Parameter $k \in \mathbb{N}$
Output: A vertex set of G that realizes $\gamma^*(G, k)$

In contrast to Densest Subgraph, this problem is computationally difficult. It is clear that Dense k-Subgraph is $\mathcal{NP}$-hard (observe that the instance $(G, k, k-1)$ to the corresponding decision problem means searching for a clique of size k in G). The best we may hope for is a polynomial algorithm with moderate approximation ratio. A natural approach for approximating $\gamma^*(G, k)$ is based on greedy methods. An example of a greedy procedure due to [201] is given as Algorithm 12.

Algorithm 12: Greedy procedure

Input: Graph $G = (V, E)$ and even parameter $k \in \mathbb{N}$ (with $|V| \geq k$)
Output: A set of k vertices of G

Sort the vertices in decreasing order of their degrees;
Let H be the set of $\frac{k}{2}$ vertices of highest degree;
Compute $N_H(v) = |\bar{N}(v) \cap H|$ for all vertices $v \in V - H$;
Sort the vertices in $V - H$ in decreasing order of the N_H-values;
Let R be the set of $\frac{k}{2}$ vertices of $V - H$ of highest N_H-values;
Return $H \cup R$

Theorem 6.3.8. *Let G be any graph with n vertices and let $k \in \mathbb{N}$ be an even natural number with $k \leq n$. Let $A(G, k)$ denote the average degree of the subgraph of G induced by the vertex set that is the output of Algorithm 12. We have*

$$\gamma^*(G, k) \leq \frac{2n}{k} \cdot A(G, k).$$

Proof. For subsets $U, U' \subseteq V$, let $E(U, U')$ denote the set of edges consisting of one vertex of U and one vertex of U'. Let m_U denote the cardinality of the edge set $E(G[U])$. Let d_H denote the average degree of the $\frac{k}{2}$ vertices of G with highest degree with respect to G. We certainly have, $d_H \geq \gamma^*(G, k)$. We obtain

$$|E(H, V - H)| = d_H \cdot |H| - 2m_H \geq \frac{d_H \cdot k}{2} - 2m_H \geq 0.$$

By the greedy rule, at least the fraction of

$$\frac{|R|}{|V-H|} = \frac{k}{2n-k} > \frac{k}{2n}$$

of these edges has been selected to be in $G[H \cup R]$. Hence, the total number of edges in $G[H \cup R]$ is at least

$$\left(\frac{d_H \cdot k}{2} - 2m_H\right) \cdot \frac{k}{2n} + m_H \geq \frac{d_H \cdot k^2}{4n}.$$

This proves the inequality for the average degree. □

The greedy procedure is the better the larger k is in relation to n. It is an appropriate choice if we want to find large dense regions in a graph. However, for very small parameters, e.g., for $k = \mathcal{O}(1)$, it is almost as bad as any trivial procedure. An approximation ratio $\mathcal{O}(\frac{n}{k})$ has been obtained by several other approximation methods, e.g., by greedy methods based on recursively deleting vertices of minimal degree [38] or by semidefinite programming [204, 531]. However, to overcome the connection between n and k, we need complementary algorithms that work well on smaller values of k. In the light of the following theorem [201], this seems possible for up to $k = \mathcal{O}(n^{\frac{2}{3}})$.

Theorem 6.3.9. Dense k-Subgraph *can be approximated in polynomial time within ratio* $\mathcal{O}(n^{\frac{1}{3}-\varepsilon})$ *for some* $\varepsilon > 0$.

No better bound for the general problem is known. In special graph classes, however, approximation can be done within better ratio. For instance, on families of dense graphs, i.e., graphs with $\Omega(n^2)$ edges, there exist polynomial-time approximation algorithms with ratio arbitrary close to one [35, 137]. A drawback here is that most of the social networks are sparse, not dense. As to lower bounds on the approximation ratio, it has recently been proven that an approximation ratio of $1+\varepsilon$ for all $\varepsilon > 0$ cannot be achieved unless all $\mathcal{NP}$ problems can be simulated by randomized algorithms with double-sided error and sub-exponential running time (more specifically, in time $\mathcal{O}(2^{n^\varepsilon})$ for all $\varepsilon > 0$)[354]. Moreover, it is even conjectured that there is no polynomial-time algorithm with approximation ratio $\mathcal{O}(n^\varepsilon)$ for all $\varepsilon > 0$ [201].

6.3.4 Parameterized Density

As we have argued, the decision version of Dense k-Subgraph is $\mathcal{NP}$-complete. In contrast to this variable decision problem (note that the density parameter is part of the input), we are now interested in studying the fixed-parameter version. A function $\gamma : \mathbb{N} \to \mathbb{Q}_+$ is a density threshold if and only if γ is computable in polynomial time and $\gamma(k) \leq k-1$ for all $k \in \mathbb{N}$. For any density threshold γ, a γ-dense subgraph of a graph $G = (V, E)$ is any subset $U \subseteq V$ such that $\bar{d}(G[U]) \geq \gamma(|U|)$. We consider the following problem:

Problem: γ-Dense Subgraph
Input: Graph G, Parameter $k \in \mathbb{N}$
Question: Does there exist γ-dense subgraph of size k within G?

Clearly, on the one hand, if we choose $\gamma(k) = k - 1$ for all $k \in \mathbb{N}$, then we obtain γ-Dense Subgraph = Clique, and thus an $\mathcal{NP}$-complete problem. On the other hand, if we choose $\gamma(k) = 0$, then any choice of k vertices induces a γ-dense subgraph and thus γ-Dense Subgraph is solvable in polynomial time. The question is: which choices of γ do still admit polynomial-time algorithms and for which γ does the problem become $\mathcal{NP}$-complete? This problem has been studied by several authors [204, 37, 308]. The following theorem due to [308] gives a sharp boundary, which also shows that a complexity jump appears very early.

Theorem 6.3.10. *Let γ be any density threshold.*

1. *If $\gamma = 2 + \mathcal{O}\left(\frac{1}{k}\right)$, then γ-Dense Subgraph is solvable in polynomial time.*
2. *If $\gamma = 2 + \Omega\left(\frac{1}{k^{1-\varepsilon}}\right)$ for some $\varepsilon > 0$, then γ-Dense Subgraph is $\mathcal{NP}$-complete.*

A direct application of the theorem gives the following result for the case of constant density functions.

Corollary 6.3.11. *Finding a k-vertex subgraph with average degree at least two can be done in polynomial time. However, there is no algorithm for finding a k-vertex subgraph with average degree at least $2+\varepsilon$ for any $\varepsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$.*

This result should be contrasted with the corresponding result for N-cores, where detecting N-cores of size k can be done in linear time in the graph size, even for all $N > 0$. This demonstrates a drastic computational difference between statistical and structural density.

Results similar to Theorem 6.3.10 have been proven for the case of special network classes with real-world characteristics, in particular, for power-law graphs and general sparse graphs [306].

6.4 Chapter Notes

In this chapter, we studied computational aspects of notions of local densities, i.e., density notions defined over induced subgraphs only, consequently suppressing network structure outside a subgroup. We considered structural (N-plexes, N-cores) and statistical relaxations (η-dense subgraphs) of the clique concept, which is the perfectly cohesive subgroup. Although many algorithmic problems for these notions are computationally hard, i.e., we do not know polynomial algorithms for solving them, there are several cases where fast algorithms exist producing desirable information on the density-based cohesive structure of a network, e.g., the number of small cliques in graphs, core numbers, or the maximum average degree reachable by a subgroup in a directed and undirected network.

An observation coming up from the presented results is that there is a seemingly hard tradeoff between mathematical soundness and meaningfulness of these notions and their algorithmic tractability. This is evident from the following table summarizing properties of our main notions:

subgroup	closed under exclusion	nested	tractable
clique	+	+	−
N-plex (for $N \in \mathbb{N}$)	+	+	−
N-core (for $N \in \mathbb{N}$)	−	−	+
η-dense subgraph (for $\eta \in [0,1]$)	−	+	−

Here, we see that nestedness, as a meaningful structure inside a group, excludes fast algorithms for computing subgroups of certain sizes. This exclusion is also inherited by some further relaxations. However, we have no rigorous proof for this observation in case of general locally definable subgroups. On the other hand, a similar relation is provably true for closure under exclusion and efficiently detecting subgroups of a given size: we cannot achieve both with an appropriate notion of density (see, e.g., [240, GT21,GT22]).

We conclude this chapter with a brief discussion of a selection of non-local concepts of cohesive subgroups that have attracted interest in social network analysis. Since non-locality emphasizes the importance for a cohesive subgroup to be separated from the remaining network, such notions play an important role in models for core/periphery structures [84, 193]. An extensive study of non-local density notions and their applications to network decomposition problems can be found in Chapter 8 and Chapter 10.

LS sets (Luccio-Sami sets). The notion of an LS set has been introduced in [399, 381]. An LS set can be seen as a network region where internal ties are more significant than external ties. More specifically, for a graph $G = (V, E)$ a vertex subset $U \subseteq V$ is said to be an *LS set* if and only if for all proper, non-empty subsets $U' \subset U$, we have

$$|E(U', V - U')| > |E(U, V - U)|.$$

Trivially, V is an LS set. Also the singleton sets $\{v\}$ are LS sets in G for each $v \in V$. LS sets have some nice structural properties. For instance, they do not non-trivially overlap [399, 381], i.e., if U_1 and U_2 are LS sets such that $U_1 \cap U_2 \neq \emptyset$, then either $U_1 \subseteq U_2$ or $U_2 \subseteq U_1$. Moreover, LS sets are rather dense: the minimum degree of a non-trivial LS set is at least half of the number of outgoing edges [512]. Note that the structural strength of LS sets depends heavily on the universal requirement that *all* proper subsets share more ties with the network outside than the set U does (see [512] for a discussion of this point). Some relaxations of LS sets can be found in [86].

Lambda sets. A notion closely related to LS sets is that of a lambda set. Let $G = (V, E)$ be any undirected graph. For vertices $u, v \in V$, let $\lambda(u, v)$ denote the number of edge-disjoint paths between u and v in G, i.e., $\lambda(u, v)$ measures the edge connectivity of u and v in G. A subset $U \subseteq V$ is said to be a *lambda set* if and only if

$$\min_{u,v \in U} \lambda(u, v) > \max_{u \in U, v \in V-U} \lambda(u, v).$$

In a lambda set, the members have more edge-disjoint paths connecting them to each other than to non-members. Each LS set is a lambda set [512, 86]. Lambda sets do not directly measure the density of a subset. However, they have some importance as they allow a polynomial-time algorithm for computing them [86]. The algorithm essentially consists of two parts, namely computing the edge-connectivity matrix for the vertex set V (which can be done by flow algorithms in time $\mathcal{O}(n^4)$ [258]) and based on this matrix, grouping vertices together in a level-wise manner, i.e., vertices u and v belong to the same lambda set (at level N) if and only if $\lambda(u, v) \geq N$. The algorithm can also be easily extended to compute LS sets.

Normal sets. In [285], a normality predicate for network subgroups has been defined in a statistical way over random walks on graphs. One of the most important reasons for considering random walks is that typically the resulting algorithms are simple, fast, and general. A random walk is a stochastic process by which we go over a graph by selecting the next vertex to visit at random among all neighbors of the current vertex. We can use random walks to capture a notion of cohesiveness quality of a subgroup. The intuition is that a group is the more cohesive the higher the probability is that a random walk originating at some group member does not leave the group. Let $G = (V, E)$ be any undirected graph. For $d \in \mathbb{N}$ and $\alpha \in \mathbb{R}_+$, a subset $U \subseteq V$ is said to be (d, α)-*normal* if and only if for all vertices $u, v \in U$ such that $d_G(u, v) \leq d$, the probability that a random walk starting at u will reach v before visiting any vertex $w \in V - U$, is at least α. Though this notion is rather intuitive, we do not know how to compute normal sets or decomposing a network into normal sets. Instead, some heuristic algorithms, running in linear time (at least on graphs with bounded degree), have been developed producing decompositions in the spirit of normality [285].