Heidelberg University
Faculty of Mathematics and Computer Science
Theoretical Computer Science and Discrete Mathematics

Master's thesis

Motif-free Graph Generation

Name:	Dominik Schweisgut
Student number:	4083524
Supervisor:	Professor Felix Joos
Co-Supervisor:	Professor Christian Schulz
Date of Submission:	June 20, 2024

Ich versichere, dass ich diese Master-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe und die Grundsätze und Empfehlungen "Verantwortung in der Wissenschaft" der Universität Heidelberg beachtet wurden.

Abgabedatum: June 20, 2024

Acknowledgments

I would like to express my profound gratitude to Professor Felix Joos and Professor Christian Schulz. They provided me with the opportunity to work on a topic that is not only of high theoretical interest but also can be explored through practical approaches. This has been a valuable chance to delve into both fields.

Professor Felix Joos consistently offered deep insights into the theory, which not only significantly aided my understanding of theoretical concepts but also proved invaluable for the practical aspects. It was a pleasure to learn from his intuition and profound understanding of graph theory. I am truly thankful for his patience in discussing my questions and for the substantial support he provided throughout my thesis.

Furthermore, I am particularly grateful for the opportunities that Professor Christian Schulz has given me over the recent years. Starting with programming projects and continuing as a Student Research Assistant, he has imparted extensive insights into the field of Algorithm Engineering. His guidance has revealed new dimensions of my studies, and he will always remain an inspiration to me. I can always approach him with any questions, and he is always willing to help.

I also want to extend my thanks to Adil Chhabra, who has always been available to answer my questions and provided numerous helpful insights on various topics, significantly aiding my job and studies.

Lastly, I want to thank my parents for their unwavering support throughout my life. Their dedication to giving me the best possible life is something I will forever be grateful for.

Zusammenfassung

Die Theorie von Zufallsgraphen und insbesondere der zufälligen Graphprozesse hat sich als leistungsfähige Methode für verschiedene Probleme der Graphentheorie erwiesen. Insbesondere hat Erdös 1947 die Theorie der Zufallsgraphen mit dem Gebiet der Ramsey-Theorie verbunden. In dieser Arbeit untersuchen wir zwei Arten von zufälligen Graphprozessen. Für bestimmte streng 2-balancierte Graphen H untersuchen wir den H-freien Prozess und den H-Entfernungs Prozess. Insbesondere der Dreiecks-freie Prozess hat viel Aufmerksamkeit erregt, da er eine neue untere Schranke für die Ramsey-Zahl $R(3,k)$ liefert. Bei diesem Verfahren werden zufällige Kanten von K_n nacheinander zu einem leeren Graphen hinzugefügt, solange sie kein Dreieck schließen. Der Dreiecks-Entfernungs Prozess funktioniert in umgekehrter Weise, wobei man von einem K_n ausgeht und die Kanten einer zufälligen Kopie eines Dreiecks entfernt, bis keine solche Kopie mehr übrig ist. In beiden Fällen ist unser Hauptinteresse die Anzahl der Kanten, die der Zufallsgraph nach Abschluss des Prozesses enthält. Allerdings sind die Ergebnisse für die H-Entfernungs Prozesse nicht so stark wie für die H-freien Prozesse und für $H \in \{C_4, K_4\}$ ist bisher nur wenig bekannt. In der Tat ist der Dreiecks-freie Prozess der einzige Prozess, bei dem die endgültige Anzahl der Kanten bis auf einen Fehler im konstanten Faktor mit hoher Wahrscheinlichkeit bekannt ist. In dieser Arbeit wollen wir einen experimentellen Ansatz zu diesem Thema liefern, indem wir die oben genannten Graphprozesse simulieren. Nach einer ausführlichen Einführung in die Theorie stellen wir Algorithmen zur Simulation dieser Prozesse vor und passen Modelle an die gewonnenen Daten an. Dabei sind wir in der Lage, bekannte Ergebnisse für den Dreiecks-freien Prozess zu reproduzieren und damit fundierte Hinweise auf äquivalente Ergebnisse für andere zufällige Graphprozesse zu liefern, die von der Theorie bisher nicht behandelt werden konnten.

Abstract

The theory of random graphs and in particular random graph processes proved to provide powerful methods for various problems in graph theory. In particular, Erdös in 1947 connected the theory of random graphs with the field of Ramsey theory. In this thesis, we study two types of random graph processes. For certain strictly 2-balanced graphs H we study the H-free process and the H-removal process. Especially the triangle-free process has gained much attention for giving a new lower bound on the Ramsey number $R(3,k)$. In this process, random edges of K_n are successively added to an empty graph as long as they do not close a triangle. The triangle-removal process works the other way round, starting from a K_n and removing the edges of a random copy of a triangle until there is no such copy left. In both cases, our main interest is in the number of edges the random graph contains after the process has finished. However, the results for the H-removal processes are not as strong as for the H-free processes and for $H \in \{C_4, K_4\}$ only little is known so far. In fact, the triangle-free process is the only process where the final number of edges is known up to an error in the constant factor with high probability. In this work we provide an experimental approach to this topic by simulating the above mentioned graph processes. After giving an extensive introduction into the theory, we develop algorithms for the simulation of these processes and fit models to the obtained data. In doing so, we are able to reproduce known results for the triangle-free process and thus provide profound indications for equivalent results for other random graph processes that could not be treated by theory so far.

Contents

1. Introduction

1.1. Motivation

Ramsey theory is one of the most studied branches of discrete mathematics and was named after the seminal result of Ramsey [Ramsey, 1987]. In our context, Ramsey theory concerns the study of edge colourings of complete graphs. In particular, Ramsey's theorem states that one finds monochromatic cliques of a given size in such a colouring if the complete graph is sufficiently large. In particular, for given integers r, s the Ramsey number $R(r, s)$ is the smallest integer such that any edge colouring in red and blue of a complete graph on $n \geq R(r, s)$ vertices contains either a red copy of a K_r or a blue copy of a K_s. However, determining the Ramsey number in other than trivial cases remains hard until today. Erdös and Szekeres [Erdös and Szckeres, 1987] in 1935 proved the bound

$$R(k, l) \leq \binom{k+l-1}{l-1}$$

which implies for example $R(3, k) = O(k^2)$. Further, this theory has been a key motivation for various powerful techniques in graph theory such as the Probabilistic Method [Alon and Spencer, 2016]. In 1961, Erdös applied a deterministic algorithm to the random graph $G(n, p)$ and was able to prove a lower bound of order $k^2/(\log k)^2$ [Erdös, 1961]. In 1980, Ajtai et al. [Ajtai et al., 1980],[Ajtai et al., 1981] proved that $R(3, k) = O(k^2/\log k)$ and Shearer [Shearer, 1983] was able to improve the constant in this term. A complementary lower bound was obtained in 1995 by Kim [Kim, 1995], where further the so called *semi-random method* was developed. However, Bohman [Bohman, 2009] was able to give an alternative proof of Kim's result using the so called triangle-free process. In particular, Pontiveros, Griffiths and Morris [Fiz Pontiveros et al., 2020] were able to follow the process to its asymptotic end and significantly improved Kim's result to a lower bound for $R(3, k)$ which is within a factor of $4 + o(1)$ of Shearer's bound. This result motivates us to further examine random graph processes. However, random graph processes are today an interesting research topic for itself and have also produced high quality results for Turán numbers. The triangle-free process is thereby a special case of the H-free process where we suc-

cessively add random edges of a K_n to an empty graph as long as they do not close a copy of H. For strictly 2-balanced graphs this was famously examined by Bohman and Keevash [Bohman and Keevash, 2009] using the differential equations method for this. In general, the theory of random graph processes was introduced by Erdös and Rényi [Erdös et al., 1960] in 1959. In their work, they also introduced one of the most famous objects in combinatorics, the Erdös-Rényi random graph. The general idea for this graph model is to draw randomly edges from the complete graph K_n. This graph model plays an important role throughout this work. In general, the whole work is dedicated to certain random graph processes such as the triangle-free process. A random graph process is a sequence of graphs $(G_i)_{i\in\mathbb{N}_0}$, where for $j \geq i$ the graph G_j is obtained randomly according to some probabiliy distribution which depends on the sequence $(G_0, \ldots, G_{j-1})$, see [Fiz Pontiveros et al., 2020]. In fact, the probability distribution often only depends on the graph G_{j-1}. The theory of random graph processes is not only important for Ramsey theory. In particular, random graph processes are occurring more and more frequently in practice to model real-world scenarios. One example for this is the preferential-attachement model Barabási and Albert [Barabási and Albert, 1999] which is used for modelling social structures for example. Further, there exists powerful algorithms to generate the random graphs obtained by this process such as the parallel algorithm of Schulz and Sanders [Sanders and Schulz, 2016].

In this work, we examine two types of random graph processes. For a given graph $H \in \{K_3, C_4, K_4\}$, which is in this work always strictly 2-balanced, we look at the above mentioned H-free process and the H-removal process. In particular, our goal is to simulate these processes to gain more insights about them. In the H-removal process we start from a complete graph on n vertices and in every step we choose one copy of H uniformly at random and remove its edges. The process ends when there is no copy of H left. Note that both processes generate an H-free graph. However, we will see that they are not equivalent. In particular, to the best of our knowledge only the triangle-free process was followed until its asymptotic end. In fact, Fiz Pontiveros et al. [Fiz Pontiveros et al., 2020] show that the final number of edges is

$$\left(\frac{1}{2\sqrt{2}} + o(1)\right) n^{3/2} (\log n)^{1/2}$$

with high probability. This means that the error is only within the constant factor. For all other processes under consideration this has not been achieved yet. However, what is remarkable about the result of Fiz Pontiveros et al. [Fiz Pontiveros et al., 2020] is that they basically show that the graphs obtained by the triangle-free process closely resemble the Erdös-Rényi random graph with the same number of edges, except the fact

that the resulting graph of the triangle-free process contains no triangles. This results in the fact that we can always use this random graph model for our intuition. This work exploits this intuition in serveral ways and it is used for example for a heuristic simulation algorithm for the H-removal process. Further, we expect that this intuition is true for the C_4-free process and the K_4-free process as well. For this, we use our intuition to suggest a similarly strong result as this was proven for the triangle-free process and afterwards we use simulation algorithms which suggest that these results might hold. We expect that final number of edges can be specified up to an error in the constant for these graph processes as well. For the H-removal process this is even more challenging. To the best of our knowledge, the strongest result for this kind of process was proved by Bohman et al. [Bohman et al., 2015] showing that the final number of edges is

$$n^{3/2+o(1)}$$

with high probability and we see that the error is here even in the exponent. We simulate this process and examine the results with respect to the possibility of similar results as for the triangle-free process. Further, for the C_4-removal process and the K_4-removal process not even such a bound is known. However, our simulations give strong evidence that a similar bound holds with the according exponents. To the best of our knowledge, this is the first work to study these graph processes with modern algorithms to this extent. The only other numerical simulation was done by Gordon et al. [Gordon et al., 1996] in 1996 for optimal covering designs with a greedy algorithm from which one can also draw conclusions for the triangle-removal process for example, see Bohman et al. [Bohman et al., 2015].

1.2. Objectives

The objective of this work is to gain more insights for the H-free process and the H-removal process for the cases $H \in \{K_3, C_4, K_4\}$. For this, we start by giving an extensive introduction to the theoretical results for the process if there are results available. Afterwards, we develop simulation algorithms by exploiting several properties of these processes. As we already mentioned above the goal, is to provide strong indications for more precise bounds on the final number of edges for the processes. In theory it is hard to follow theses processes until their asymptotic end since one has to keep track of several properties of the graphs during the process. However, since every process under consideration becomes stationary at some point, we are able to simulate the process until the end. By doing so for several numbers of vertices we first reproduce the strong

results for the triangle-free process to see how well this approach works. Afterwards, we apply this to the other processes to make predictions for similar bounds. This is the main goal of this work. Further, we shortly analyze the time and memory performance of the algorithms which is only to some extend optimizable since we have to follow the process until its end for reliable results. However, we see for the H-removal process that we can even use a heuristic algorithm for the simulation which has significant benfits in terms of time and memory performance.

1.3. Structure

The remainder of this thesis is structured as follows: In Chapter 2 we describe some fundamental concepts that we use frequently throughout this work. Here we introduce some basics that we need regarding the theory of random graphs. Further, we devote a section to the work of Bohman and Keevash [Bohman and Keevash, 2009]. In their work, they provide results for the H-free process for several choices of H that we will use in different places in this thesis. Afterwards, we devote every graph process under consideration one chapter. If there are already sophisticated results known, we give an extensive introduction to the current state of the theory by giving an overview on the proofs and the arguments used. Afterwards, we describe the algorithms that we use for the simulation of the according process. For the H-removal process we even develop heuristic algorithms for the simulation. Finally, we analyze the obtained data with respect to the final number of edges in the graph. We look at some basic statistical properties and fit different models to the data. Further, we also shortly analyze the time and memory performance of our algorithm. In the appendix, we present some experiments on the independence number for the triangle-free process and the C_4-free process. This is also interesting in theory for this kind of random graph process.

2. Prelimnaries

In this chapter, we introduce the basic notations and concepts we need throughout this thesis. Further, we present the work of Bohman and Keevash [Bohman and Keevash, 2009] in Section 2.3 since we need their concepts in various chapters. Since there is, to the best of our knowledge, no other published work trying to gain insights with the simulation of these graph processes, we give the related theoretical work to the processes in the corresponding chapters.

2.1. Notation and basic concepts

In this work, we set $\mathbb{N} := \{1, 2, 3, \dots\}$ and $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. Further, for any $k \in \mathbb{N}$ we write $[k] := \{1, \dots, k\}$. We use the usual Landau-notation and asymptotic notation is understood with respect to n if not otherwise stated. For $a, b, c \in \mathbb{R}$ we write $a \in (b \pm c)$ for $b - c \leq a \leq b + c$. The notation $0 < a \ll b$ means that there exists an increasing function $f(x)$ such that the arguments that are made with this assumption hold for $0 < a < f(b)$. By $\log n$ we always mean the natural logarithm. In the context of this thesis, a graph $G = (V, E)$ is a pair of sets where $e \in E \subseteq V \times V$ and we only look at simple graphs without self-loops. We refer to the elements of V as *vertices* and to the elements of E as *edges*. Sometimes we also write $V(G)$ instead of V and $E(G)$ instead of E if we want to emphasize which graph we are looking at. Further, we define $v(G) := |V(G)|$ and $e(G) := E(G)$. With this notation, we define the *edge density* of G to be $2m/n(n-1)$. Note that often we have $V = [n]$ and hence $v(G) = n$. If not otherwise stated, an edge $(u, v) \in E$ is always meant as undirected. Further, for any $v \in V$ we refer to the set of all adjacent vertices to v excluding v itself by $N(v)$. We write $d(v)$ for the size of the neighbourhood of v. If we want to emphasize the graph, we also write $d_G(v)$ for the degree and $N_G(v)$ for the neighbourhood of the vertex v. The *maximum degree* of G is denoted by $\Delta(G) := \max_{v \in V} d_G(v)$ and the *minimum degree* of G is denoted by $\delta(G) := \min_{v \in V} d_G(v)$. An independent set $S \subseteq V$ of the graph G is a subset of the vertices such that there are no two vertices $v \neq w \in S$ with $(v, w) \in E$. The maximum size of an independent set in the graph G is the *independence number* of G and is denoted by $\alpha(G)$. We say that a graph H is *2-balanced* if it has at least three

vertices, three edges, and if

$$\frac{e(H)-1}{v(H)-2} \geq \frac{e(F)-1}{v(F)-2} \tag{2.1}$$

holds for all proper subgraphs F with at least three vertices. We say that H is *strictly* 2-balanced if the inequality is sharp. Note that every graph H for which we simulate one of the processes is strictly 2-balanced and in particular complete graphs K_l and cycles C_l are strictly 2-balanced for $l \geq 3$. For $k, l \in \mathbb{N}$, the Ramsey number $R(k,l)$ is the smallest integer such that every red-blue coloring of the edges of a $K_{R(k,l)}$ contains either a red K_k or a blue K_l. For a sequence of random events $(A_n)_{n\in\mathbb{N}}$ we say that this sequence happens *with high probability* if

$$\lim_{n\to\infty} \mathbb{P}[A_n] = 1. \tag{2.2}$$

Sometimes we also say *w.h.p.* instead.

2.2. Random graphs

In this section, we introduce some basic concepts in the theory of random graphs. In the Erdös-Rényi graph model $G(n,m)$ we choose a graph uniformly at random from the collection of all graphs which have n vertices and m edges. Note that thereby, vertices are assumed to be labelled and hence a relabeling of the vertices is seen as a distinct graph. Closely related to that graph model and also named after Erdös and Rényi is the graph model $G(n,p)$. In this model, every edge $e \in E(K_n)$ is added to the graph with probability p. With this model, a fixed graph F on n vertices and m edges is obtained with probability

$$p^{e(H)}(1-p)^{\binom{n}{2}-e(H)}. \tag{2.3}$$

If a graph G is obtained by the random graph model $G(n,p)$, we denote this by $G \sim G(n,p)$. Further, for $G \sim G(n,p)$ we can easily obtain some basic properties. For example, we have

$$\mathbb{E}[e(G)] = \binom{n}{2} p \tag{2.4}$$

and for any $v \in V(G)$ we have

$$\mathbb{E}[d_G(v)] = (n-1)p. \tag{2.5}$$

In particular, we have

$$\mathbb{P}[d_G(v) = k] = \binom{n-1}{k} p^k (1-p)^{n-1-k}. \tag{2.6}$$

We use similar properties frequently throughout this thesis. In particular, the two models above are similar. In fact, we can assume that for $pn^2 \to \infty$ the graphs obtained by the models $G(n,p)$ and $G(n,n^2p)$ have similar properties. Since we are mainly interested in properties that hold with high probabilty, we use most of the time the $G(n,p)$ model since it is easier to handle. One of the main concepts that we look at in this thesis are random graph processes:

Definition 2.2.1. A random graph process is a sequence of graphs $(G_i)_{i\in\mathbb{N}_0}$ where for every $i \in \mathbb{N}$ the graph G_i is obtained according to a probability distribution which only depends on $\{G_0, \dots, G_{i-1}\}$.

Remark 2.2.1. Often the probability distribution of G_i only depends on G_{i-1}.

We give the definitions of the specific random graph processes in the corresponding chapters.

2.3. Lower bounds for the H-free process

In this section, we want to present some results from one of the most formative works in random graph process by Bohman et al. [Bohman and Keevash, 2009]. The main contribution for us is their proof of a lower bound for the number edges of the final graph in the H-free process for the case when H is strictly 2-balanced. The main technique for this is the differential equations method. In particular, they prove the following results by showing that certain graph parameters follow differential equations they provide.

Theorem 2.3.1 ([Bohman and Keevash, 2009], Theorem 1.1.)**.** Suppose that H is a strictly balanced graph with $v(H)$ vertices and $e(H)$ edges. Then for some $c > 0$ with high probability we get for the minimum degree $\delta(G_{H,n})$ of the H-free process

$$\delta(G_{H,n}) \geq cn^{1-\frac{v(H)-2}{e(H)-1}} (\log n)^{1-\frac{1}{e(H)-1}}. \tag{2.7}$$

Note that by Equation 2.7 we get

$$e(G_{H,n}) = \frac{1}{2} \sum_{v\in V(G_{H,n})} d_{G_{H,n}}(v) \geq \frac{1}{2} n \cdot \delta(G_{H,n}) \overset{2.3.1}{\geq} \frac{c}{2} n^{2-\frac{v(H)-2}{e(H)-1}} (\log n)^{\frac{1}{e(H)-1}}$$

with high probability and hence we get a lower bound for the number of edges in the final graph of the H-free process. In the chapters regarding the processes for $H = K_3$, $H = C_4$ and $H = K_4$ we see that this bound is tight up to a constant factor. In order to give an overview on how this result was proven and to obtain further helpful results for Chapter 5 and Chapter 7, we first have to introduce some concepts used in [Bohman and Keevash, 2009]. Let $(G_i)_{i\in\mathbb{N}_0}$ be a sequence of graphs representing the H-free process and let $G_i = (V(G_i) = [n], E_i)$ for all $i \in \mathbb{N}_0$. At step $i \in \mathbb{N}_0$ of the process let $O(i)$ be the pairs of open vertices, i.e., every open pair represents an edge that can be inserted to G_i. For a strictly 2-balanced graph H we write

$$p = n^{-\frac{v(H)-2}{e(H)-1}}. \tag{2.8}$$

We see the intuition behind this definiton at a later stage. Now let $\Gamma = (V(\Gamma), E(\Gamma))$ be any graph and let $A \subseteq V(\Gamma)$. Then we define

$$S_{A,\Gamma} = p^{e(\Gamma)-e(\Gamma[A])}n^{v(\Gamma)-|A|}. \tag{2.9}$$

Definition 2.3.1. For any graph $\Gamma = (V(\Gamma), E(\Gamma))$ and $A \subseteq V(\Gamma)$ we say that the pair (A, Γ) is *strictly balanced* if $S_{A,\Gamma[B]} > S_{A,\Gamma}$ for every $A \subsetneq B \subsetneq V(\Gamma)$. We say that the pair is *strictly dense* if $S_{A,\Gamma[B]} > 1$ for every $A \subsetneq B \subseteq V(\Gamma)$.

We want to quickly discribe the intuition behind p and $S_{A,\Gamma}$. Intuitively, we would expect that in the beginning of the process the graph G_i is similar to the Erdös-Renyi graph $G(n, i)$. This is simply because the fraction of the forbidden edges which can not be added to the graph is in this phase small compared to the number of open edges. In particular, Joel Spencer suggested that both graphs should be similar with respect to small subgraph counts during the initial phase of the process. In particular, this similarity should at least be given up to the point i when the number of edges in $G(n, i)$ is roughly equal to the number of copies of H in $G(n, i)$ [Bohman and Keevash, 2009]. Simple calculations in the $G(n, i)$ model show that this is the case when i is roughly pn^2 with p as in 2.8. Due to the similarity of the models $G(n, i)$ and $G(n, p)$ for $p = i/\binom{n}{2}$ we give future computations in this model using the $G(n, p)$ model. With this in mind, we see that $S_{A,\Gamma}$ is roughly the expected number of labeled extensions to Γ from the fixed vertex set A and hence we see that this is an important parameter to us. In particular, one of the main goals is to track extensions of a certain kind. This motivates the following definitions.

Definition 2.3.2. Let $\Gamma = (V(\Gamma), E(\Gamma))$ be any graph and let J be a spanning subgraph of Γ. Further, let $A \subseteq V(\Gamma)$ be an independent set in Γ and let $\phi : A \longrightarrow [n]$ be an

injective mapping. Then the *extension variable* $X_{\phi,J,\Gamma}(i)$ is defined as the number of injective maps $f : V(\Gamma) \longrightarrow [n]$ fulfilling

I. $f(e) \in O(i)$ for every $e \in E(\Gamma) \setminus E(J)$,

II. $f(e) \in E(i)$ for every $e \in E(J)$, and

III. f restricts to ϕ on A.

Now we say that the random variable $X_{\phi,J,\Gamma}(i)$ is *trackable* if one of the following conditions holds:

(a) (A, Γ) is strictly dense and Γ does not contain H as a subgraph.

(b) $S_{A,\Gamma} = 1$, (A, Γ) is strictly balanced, $E(J) \subsetneq E(\Gamma)$, and H is not a subgraph of the graph Γ' which is obtained from Γ by adding the edge (u, v) for all $u, v \in A$ with $(\phi(a), \phi(b)) \in E_i$.

Now we fix some constants and define functions that we need to formulate our next main theorem. We motivate some of these functions in the corresponding chapters where they are used, see Chapter 5 and Chapter 7. Let $\mu, \epsilon, W, V \in \mathbb{R}$ be constants which satisfy

$$0 < \mu \ll \epsilon \ll \frac{1}{W} \ll \frac{1}{V} \ll \frac{1}{e(H)}.$$

Further, we define the continous time variable $t := t(i) := i/pn^2$ where p is defined as in 2.8. We can see t as a scaled version of the steps of the process. In the following, the goal is to follow the before define trackable extension variables to time $t_{\max} = \mu(\log n)^{1/(e(H)-1)}$. By definition of the process and t we have at this time

$$m = \mu(\log n)^{1/(e(H)-1)} \cdot pn^2 \tag{2.10}$$

edges in the graph because we add in every step one edge to the graph. Now let $\mathcal{T}$ be the set of all triples (A, J, Γ) where A is an independent set in Γ, J is a spanning subgraph of Γ, $v(\Gamma), e(\Gamma) < n$ and the random variables $X_{\phi,J,\Gamma}(0)$ are trackable. Now we define functions

$$q(t) = e^{-2e(H)aut(H)^{-1}(2t)^{e(H)-1}},\ P(t) = W(t^{e(H)-1} + t),\ e(t) = e^{P(t)} - 1,\ s_e = n^{\frac{1}{2e(H)}-\epsilon}. \tag{2.11}$$

With this we have everything that we need to state our next main theorem which describes the evolution of trackable extension variables up to time $t_{\max}$.

Theorem 2.3.2 ([Bohman and Keevash, 2009], Theorem 1.4.)**.** With high probability, for every $i \leq m$ and trackable extension variable $X_{\phi,J,\Gamma}(i)$ corresponding to a triple in $\mathcal{T}$, we have

$$X_{\phi,J,\Gamma}(i) = (1 \pm \frac{e(t)}{s_e})(x_{A,J,\Gamma}(t) \pm \frac{1}{s_e})S_{A,J}, \tag{2.12}$$

where

$$x_{A,J,\Gamma}(t) = (2t)^{e(J)}q(t)^{e(\Gamma)-e(J)}. \tag{2.13}$$

Now we give some examples of quantities that are described by Theorem 2.3.2 to see how it can be used. For this let ϕ_0 be the unique function $\phi_0 : \emptyset \longrightarrow [n]$ and for $v \in [n]$ let $\phi_v : \{a\} \longrightarrow [n],\ \phi(a) = v$. Further, we write e for the graph $(V(e) = \{a, b\}, E(e) = \{(a, b)\})$ and $\overline{e}$ for the graph $(V(\overline{e}) = \{a, b\}, E(\overline{e}) = \emptyset)$, i.e., the edge and the empty graph on two vertices. Note that the empty set as well as the set consisting only of one vertex is always an independent set in any graph, as long as it has at least one vertex.

Lemma 2.3.3. Let $H \in \{K_3, C_4, K_4\}$. Further let $\phi_0 : \emptyset \longrightarrow [n]$, $J = \overline{e}$, $\Gamma = e$ as above and let $A = \emptyset$ be an independent set in Γ. Then for $i \leq m$ the random variable $X_{\phi_0,\overline{e},e}(i)$ is a trackable extension variable corresponding to a triple in $\mathcal{T}$ and $X_{\phi_0,\overline{e},e}(i) = 2|O(i)|$.

Proof. We first show that we have a triple in $\mathcal{T}$, i.e., that $X_{\phi_0,\overline{e},e}(0)$ is trackable. We show that (a) in Definition 2.3.2 is fulfilled. First, let $B = \{a\} \subseteq V(e)$. Then by definition, we have $S_{A,\Gamma[B]} = pn^2 > 1$ for every possible H. By symmetry, it follows that $S_{A,\Gamma[\{b\}]} > 1$. Further, we get for $B = V(e)$ that $S_{A,\Gamma[B]} = S_{A,\Gamma} = pn^2 > 1$ and hence $X_{\phi_0,\overline{e},e}(0)$ is trackable. Since the condition in (a) is independent from the step, we also get that $X_{\phi_0,\overline{e},e}(i)$ is trackable. It remains to show that $X_{\phi_0,\overline{e},e}(i) = 2|O(i)|$. For this we look at the maps $f : \{a, b\} \longrightarrow [n]$ which fulfill points (I)-(III) in Definition 2.3.2. Condition (I) simply means that the injective map f corresponds to an open edge, since f can only map one edge. Since $E(\overline{e}) = \emptyset$ condition (II) is empty and since $A = \emptyset$ condition (III) is also empty. Now note that for every open edge $(u, v) \in O(i)$ we get two possible maps one for each directed version of (u, v). Hence, it follows that $X_{\phi_0,\overline{e},e}(i) = 2|O(i)|$. □

Other important quantities for us are the degrees of the graph during the H-free process. This can also be described by trackable extension variables.

Lemma 2.3.4. Let $H \in \{K_3, C_4, K_4\}$ and $v \in [n]$. Further let $\phi_v : \{a\} \longrightarrow [n]$, $J = e$, $\Gamma = e$ as above and let $A = \{a\}$ be an independent set in Γ. Then for $i \leq m$ the random variable $X_{\phi_v,e,e}(i)$ is a trackable extension variable corresponding to a triple in $\mathcal{T}$ and $X_{\phi_v,e,e}(i) = d_{G_i}(v)$.

Proof. As in the lemma above, we start be showing that $X_{\phi_v,e,e}(0)$ and $X_{\phi_v,e,e}(i)$ are trackable by showing that condition (a) in Definition 2.3.2 holds. Note that by definition

we only have to check the case $B = V(e)$. Hence, we compute $S_{A,\Gamma[B]} = S_{A,\Gamma} = pn > 0$ for every possible H and we see that $X_{\phi_v,e,e}(0)$ and $X_{\phi_v,e,e}(i)$ are indeed trackable. It remains to show that $X_{\phi_v,e,e}(i) = d_{G_i}(v)$. For this, we check again the injective maps $f : \{a, b\} \longrightarrow [n]$ which fulfill the conditions (I)-(III) in Definition 2.3.2. Since $\Gamma = e = J$, condition (I) is empty. Further, condition (II) implies that f corresponds to an edge in G_i and condition (III) implies that this is happens in a unique way since it ensures $f(a) = v$. Together, this implies that every map f fulfilling these conditions correspond uniquely to an edge which includes v. Hence, we get $X_{\phi_v,e,e}(i) = d_{G_i}(v)$. □

What is remarkable about Theorem 2.3.2 is that proves our intuition that the H-free process closely resembles the Erdös-Renyi graph $G(n, i)$ during the initial phase of the algorithm for many graph parameters including small subgraph counts of course with the exception that there are no copies of H in the produced graph. The two lemmas above even underline this intuition since we proved that Theorem 2.3.2 can be applied in these situations. We will see in Chapter 3, Chapter 5 and Chapter 7 that the number of open edges in the $G(n, p)$ model is roughly what is predicted by Theorem 2.3.2 up to an error term. Further, in the $G(n, p)$ model the expected degree of a vertex is roughly np which is also resembled in Theorem 2.3.2. Hence, this theorem forms the basis for our considerations for the simulation of the H-free processes.

To conclude this section, we want to give a further corollary which directly implies Theorem 2.3.1. For this, we just have to define the corresponding trackable extension variable in a similar way as for the degrees of the graph to obtain a trackable extension variable which describes the set of common neighbors of a set of given size of vertices and apply Theorem 2.3.2.

Corollary 2.3.4.1 ([Bohman and Keevash, 2009], Corollary 1.5.)**.** With high probability, for every d with $p^d n > 1$, set A of d vertices and $i \leq m$, the number of common neighbours of A in G_i is $(1 + o(1))(2i/n^2)^d n$.

In particular, using $d = 1$ and noting that $pn > 1$ for every $H \in \{K_3, C_4, K_4\}$ we get for $i = m$ with high probability

$$d_{G_m}(v) = (1 + o(1))(\frac{2mn}{n^2}) = (4\mu + o(1))(\log n)^{\frac{1}{e(H)-1}} n^{1-\frac{v(H)-2}{e(H)-1}} \tag{2.14}$$

and since $\delta(G_m) \leq d_{G_m}(v)$ for all $v \in [n]$ and the minimum degree of the graph can only grow during the H-free process there is a constant $c > 0$ such that

$$\delta(G_{H,n}) \geq cn^{1-\frac{v(H)-2}{e(H)-1}} (\log n)^{1-\frac{1}{e(H)-1}}. \tag{2.15}$$

2.4. Experimental Methodology

In this section, we want shortly describe our methodology for the simulation of the processes. All algorithms were implemented using C++ and we compiled it using gcc 9.4.0 with full optimization enabled (-O3 flag). During the implementation we oriented us on the KaMIS framework (`https://github.com/KarlsruheMIS/KaMIS`). All experiments were performed on a machine with two sixteen-core Intel(R) Xeon(R) Silver 4216 processors running at 2.10GHz. The machine has 93GB of main memory, and 16MB of L2-Cache. The machine runs Ubuntu 20.04.1 LTS and is built on the GNU/Linux 5.4.0-152-generic kernel architecture.

H-free process. For the H-free process where $H \in \{K_3, C_4, K_4\}$ we run the algorithm a total of three times for reliability of the results of the algorithm. For each repetition we use different random seeds and a different rejection parameter α. Note that α does not influence the solution quality, only time and memory performance. In particular, the main focus of this work is on solution quality and not on performance of the algorihtms since our main goal is the prediction of the final number of edges. However, we did not repeat the process for each parameter α mutliple times as well due to time constraints. In particular, for the performance analysis we are only interested in how the paramter α influences the time and memory consumption and not in the total numbers and we are confident that this can be seen reliably without repeating the experiments for every α as well. For the statistical results we repeated the experiment with the same parameter α 200 times. We will see that this as well indicates that not many repetitions of the process are needed.

H-removal process. For the H-removal process we also repeated the experiments several times with different random seeds and rejection parameters α. The precise number of repetitions depends on the process and is based on observations of the simulation behavior. However, for reliability of the result we run the simulation in every case several times. For the heuristic simulation we run the simulation for different input edge densities and random seeds only once due to time constraints. However, our main interest here is the convergence of the algorithm result which can also be reliably seen in this way. For the case $H = K_3$, we did nine experiments per number of vertices to emphasize the convergence behaviour, and for the other cases we did this three times. Since there are no results known by theory, the evaluation of the reliability of this approach has to be handled differently. However, we provide again 200 repetitions for one number of vertices and one parameter α to justify our approach.

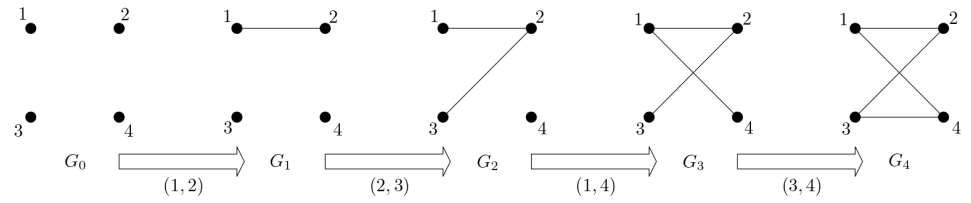


Figure 3.1.: Simple example for the triangle-free process on four vertices.

3. The Triangle-free Process

In the following section 3.1 we want to present theoretical results regarding the triangle-free process. Thereby, we will first introduce the triangle-free process and give some intuition for it. Remarkably, we see that the resulting graph is in its properties similar to a random graph $G(n, p)$ where n is the number of vertices and p is some edge density. Afterwards, we present some of the results of Fiz Pontiveros et al. [Fiz Pontiveros et al., 2020] and give a brief overview of the techniques used for the proof.

3.1. Theoretical Results

The triangle-free process is a special case of a random graph process. Let $G_0 = ([n], \emptyset)$ be the empty graph without any edges. Then, for any $m \in \mathbb{N}$, the graph G_m is obtained by choosing one edge uniformly at random from $E(K_n) \setminus E(G_{m-1})$ which does not close a triangle in G_{m-1}. The obtained graph process $(G_m)_{m\in\mathbb{N}}$ is called the *triangle-free process*. Since $e(K_n) < \infty$, this graph process becomes stationary at some point. To be precise, the process ends if there is a $\nu \in \mathbb{N}$ for which every non-edge would close a triangle in G_ν. We will denote such an element in the process as $G_{\triangle,n}$. The most interesting question for this process is how many edges the resulting graph contains. We shortly summerize the most important related work on this topic. After its introduction in 1990 by Bollobás and Erdös, the question for the order of magnitude of the size of the resulting graph remained open for nearly 20 years. The motivation for its introduction was that it might give a good lower bound on $R(3, k)$. One of the first sophisticated results was that with high probability $e(G_{\triangle,n}) \geq cn^{3/2}$ form some constant $c > 0$. This was a result by Erdös, Suen and Winkler [Erdös et al., 1995]. Bohmann [Bohman, 2009]

proved in 2009 that

$$e(G_{\triangle,n}) = \Theta(n^{3/2}\sqrt{\log(n)})$$

with high probability as $n \to \infty$. Thereby, he followed the triangle-free process for a constant fraction of its duration. This approach was later generalized by Bohman and Keevash [Bohman and Keevash, 2009] using similar techniques in a more general setting. We have already given an introduction to this work in Section 2.3. Another more general result was proven by Osthus and Taraz [Osthus and Taraz, 2001]. They proved for strictly 2-balanced graph H that

$$e(G_{H,n}) \geq c \cdot n^{2-1/m_2(H)}(\log n)^{1/(e(H)-1)}, \tag{3.1}$$

where

$$m_2(H) := \max_{F \subseteq H} \frac{e(F)-1}{v(F)-2}. \tag{3.2}$$

Further, they conjectured that this is within a constant factor of the truth. Finally, in the work that we present in this section, Fiz Pontiveros et al. [Fiz Pontiveros et al., 2020] improved Bohman's result by proving

$$e(G_{n,\triangle}) = \left(\frac{1}{2\sqrt{2}} + o(1)\right) n^{3/2}\sqrt{\log n} \tag{3.3}$$

with high probability. This is achieved by following the triangle-free process until only $o(n^{3/2}\sqrt{\log n})$ steps of the process are left. This means at most $o(n^{3/2}\sqrt{\log n})$ edges can be added and hence they follow the process until its asymptotic end. In doing so, they keep track of several parameters of the process, some of which we also introduce in this section. One interesting oberservation in this process is that the graphs that are obtained by this process closely resemble graphs obtained by the Erdös-Renyi model. Morally, we can simply consider the graphs G_m as $G(n,m)$ during the process as long as we do not come to the point where the process becomes stationary, except that they do not contain any triangles. This intuition can be seen, for example, from the fact that the size of the set of forbidden edges during the process is relatively small compared to the size of the non-edges in general if n is large. We will see that this intuition slightly changes if we look at other random graph processes like the triangle-removal process in Chapter 4.

We now formally introduce the triangle-free process. Further, we give an overview on the techniques used for thes latest results described above. For this, we follow the work of Fiz Pontiveros et al. [Fiz Pontiveros et al., 2020]. We start by giving the basic definitions needed. In this section, the random graph process that we are interested in

is the triangle-free process. We will see that Remark 2.2.1 applies to this process.

Definition 3.1.1. The random graph process $(G_i)_{i\in\mathbb{N}}$, where $G_0 = ([n], \emptyset)$ and G_i is obtained from G_{i-1} by selecting among all edges $e \in E(K_n \setminus G_{i-1})$ which would not close a triangle in G_{i-1} one edge uniformly at random and inserting it into G_{i-1}, is called the *triangle-free process*.

Especially with regard to the application to the Ramsey number $R(3,k)$ there are two properties of $G_{\triangle,n}$ which are of great interest. The first question that arises is how many edges the graph $G_{n,\triangle}$ has and the second interesting property is the independence number of the resulting graph. In this work, we are mainly interested in the size of $e(G_{\triangle,n})$ for which Fiz Pontiveros et al. [Fiz Pontiveros et al., 2020] proved the following result.

Theorem 3.1.1 ([Fiz Pontiveros et al., 2020], Theorem 1.1)**.**

$$e(G_{\triangle,n}) = \left(\frac{1}{2\sqrt{2}} + o(1)\right) n^{3/2}\sqrt{\log n} \tag{3.4}$$

with high probability as $n \to \infty$.

For the rest of this section we give an overview of this proof which contains some sophisticated techniques as well as a lot of lengthy computations. The basic idea is to track and control a large number of graph properties in every step until there are not enough edges in $E(K_n \setminus G_i)$ left to influence the result too much. With respect to Theorem 3.1.1, this will be precisely the case when there are only $o(n^{3/2}\sqrt{\log n})$ edges left. The method that will be used for this is the so called differential equations method which proved to be useful in other scenarios as well ([Beveridge et al., 2007], [Bohman, 2009]). The basic idea of this method is to show that a certain graph property behaves over time like a differential equation which can be controlled to some extend. Nonetheless, we will not see any differential equations in this work as we only want to give an outline of the proof here. We start basic definitions used in this work. Throughout the rest of this section $n \in \mathbb{N}$ is the number of vertices for the triangle-free process and $i \in \mathbb{N}$ is a fixed index resembling a certain step in the process.

Definition 3.1.2. An edge $e \in E(K_n \setminus G_i)$ is called *open* if and only if e could be inserted into G_i without closing a triangle in G_i. If e closes a triangle in G_i the edge is called *closed*.

Remark 3.1.1. Obviously, Definition 3.1.2 defines a subdivision of $E(K_n)$ in every step $i \in \mathbb{N}$ of the process. Further, an edge $e = (v,w)$ is open if and only if $v, w \in V(G_i)$ have no common neighbor in G_i.

The set of all open edges in step $i \in \mathbb{N}$ of the process is denoted by

$$O(i) := \{e = (v, w) \in E(K_n \setminus G_i) \ : \ N(v) \cap N(w) = \emptyset\} \tag{3.5}$$

and we set $Q(i) := |O(i)|$. Since $O(i)$ is precisely the set of edges which can be added to G_i in step $i + 1$, the triangle-free process ends when $Q(m) = 0$. For the proof of Theorem 3.1.1 this is the most important graph parameter we have to follow. In particular, the rate of change of this parameter is important to us. One of the main results of the work of [Fiz Pontiveros et al., 2020] is that the for all $i \leq (1+o(1))e(G_{\triangle,n})$ the graph G_i closely resembles the Erdös-Renyi graph $G(n, i)$, except that G_i contains no triangles. Hence, we can use the Erdös-Renyi model for our intuition. Using this intuition we can prove the following.

Lemma 3.1.2. Let $n \in \mathbb{N}$ and $0 \leq i \leq \binom{n}{2}$. Further, let $G \sim G(n, p)$, where $p = i/\binom{n}{2}$ is the edge density of G. Then we have

$$\mathbb{E}[|\{\text{open edges in G}\}|] = \binom{n}{2}(1 - p^2)^{n-2} \approx \binom{n}{2} \cdot e^{-p^2 n}. \tag{3.6}$$

Proof. In order to compute the expectation value in 3.6, we have to know the probability of each edge $e \in E(K_n)$ to be open in G. Since every edge has probability p to be in the graph, we have

$$\mathbb{P}[e = (u, v) \text{ is open}] = \mathbb{P}[N(u) \cap N(v) = \emptyset] = (1 - p^2)^{n-2} \approx (1 - p^2)^n \approx e^{-p^2 n},$$

because the probability that the vertices u, v do not share the neighbor $w \neq u, v$ is $1-p^2$ and this has to hold for all other $n - 2$ vertices. Using indicator variables $\mathbf{1}_{\{e \text{ open}\}}$ for every edge $e \in E(K_n)$ and the linearity of expectation, this yields

$$\mathbb{E}[|\{\text{open edges in G}\}|] = \sum_{e \in E[K_n]} \mathbb{E}[\mathbf{1}_{\{e \text{ open}\}}] = \binom{n}{2}\mathbb{P}[e = (u, v) \text{ is open}] \approx \binom{n}{2} e^{-p^2 n}.$$

□

What is remarkable is that this is close to the truth for the triangle-free process, except for an error term. Before we state the corresponding theorem we have to introduce some notation. For the rest of this section let $\epsilon > 0$ be sufficiently small.

Definition 3.1.3. For $i \in \mathbb{N}$ we define the *time* t of the triangle-free process after i steps by

$$i = t \cdot n^{3/2}. \tag{3.7}$$

With this definition in mind the goal is to follow the process up to time

$$t^* = \left(\frac{1}{2\sqrt{2}} - \epsilon\right)\sqrt{\log n} \tag{3.8}$$

and the according step of the process is then $i = t^* \cdot n^{3/2}$. With this notation we can state the following significant theorem.

Theorem 3.1.3 ([Fiz Pontiveros et al., 2020], Theorem 2.1)**.** With high probability,

$$Q(i) \in e^{-4t^2}\binom{n}{2} \pm e^{-2t^2} n^{7/4} (\log n)^3 \tag{3.9}$$

for every $i \leq (\frac{1}{2\sqrt{2}} - \epsilon)n^{3/2}\sqrt{\log n}$.

To see that this is actually close to our result in Lemma 3.1.2 we note that the edge density in $G \sim G(n,i)$ is $p = i/\binom{n}{2} \approx 2i/n^2$ and hence we get

$$e^{-p^2 n} \approx e^{-\frac{4i^2}{n^4}n} = e^{-4t^2}, \tag{3.10}$$

where the last equality follows immediately from Definition 3.1.3.

Remark 3.1.2. Another fact that is remarkable is that we can see that the absolute error in Theorem 3.1.3 is decreasing over time. This is because n is fixed throughout the process and $t = m \cdot n^{3/2}$ is increasing and hence $e^{-2t^2} > 0$ is decreasing towards 0.

In particular, we can show that Theorem 3.1.3 implies the lower bound in Theorem 3.1.1.

Lemma 3.1.4. Let $\epsilon > 0$ be arbitrary. With high probability as $n \to \infty$ we have $Q(i) > 0$ for all $i \leq (\frac{1}{2\sqrt{2}} - \epsilon)n^{3/2}\sqrt{\log n}$ and hence the lower bound in Theorem 3.1.1 holds.

Proof. First, we note that $i \leq (\frac{1}{2\sqrt{2}} - \epsilon)n^{3/2}\sqrt{\log n}$ is equivalent to $t \leq t^*$, where t is the time in step i of the process. Hence, we get

$$e^{2t^2} \leq e^{2t^{*2}} = e^{2\left(\frac{1}{2\sqrt{2}}-\epsilon\right)^2 \log n} = n^{2\left(\frac{1}{2\sqrt{2}}-\epsilon\right)^2} = n^{\left(\frac{1}{4}-\epsilon\sqrt{2}+2\epsilon^2\right)} \leq n^{\left(\frac{1}{4}+2\epsilon^2\right)}, \tag{3.11}$$

where the last inequality holds, for n sufficiently large. In particular, we get $e^{2t^2} \leq n^{\frac{1}{4}-\epsilon}$ for all sufficiently small ϵ. If we now look at the lower bound in 3.9 we get by Theo-

rem 3.1.3 for n large enough

$$\begin{aligned} Q(i) &\geq e^{-4t^2}\binom{n}{2} - e^{-2t^2}n^{7/4}(\log n)^3 \\ &= e^{-4t^2}\left(\binom{n}{2} - e^{2t^2}n^{7/4}(\log n)^3\right). \end{aligned}$$

Since, $e^{-4t^2} > 0$ it is sufficient to show

$$\binom{n}{2} - e^{2t^2}n^{7/4}(\log n)^3 > 0.$$

But by equation 3.11 we have

$$\begin{aligned} \binom{n}{2} - e^{2t^2}n^{7/4}(\log n)^3 &\geq \binom{n}{2} - n^{(1/4-\epsilon)}n^{7/4}(\log n)^3 \\ &= \binom{n}{2} - n^{(2-\epsilon)}(\log n)^3 \\ &= \frac{n^2}{2} - n^{(2-\epsilon)}(\log n^3) - \frac{n}{2} \\ &> 0, \end{aligned}$$

for large enough n. Hence, we have $Q(i) > 0$ with high probability as $n \to \infty$ for $i \leq (\frac{1}{2\sqrt{2}} - \epsilon)n^{3/2}\sqrt{\log n}$. In particular, $\epsilon > 0$ could be chosen arbitrarily small and hence the lower bound in Theorem 3.1.1 follows immediately by the definition of t^* in 3.8. □

To give a rough idea on how the random variables $Q(i)$ can be controlled, we give an overview of the necessary parameters and the idea behind them. In particular, these parameters are needed to control the rate of change of $Q(i)$.

Definition 3.1.4. Let $i \in \mathbb{N}$. Two open edges $e, f \in O(i)$ are called *Y-neighbors* in G_i if e and f form two sides of a triangle where the third side of the corresponding triangle is contained in $E(G_i)$.

Definition 3.1.5. For every edge $e \in E(K_n)$ and $i \in \mathbb{N}$ we define the random variable $Y_e(i)$ by

$$Y_e(i) := |\{f \in O(i) \ : \ e \text{ and } f \text{ are } Y\text{-neighbors in } G_i\}| \tag{3.12}$$

for $e \in O(i)$. If $e \in E(K_n) \setminus O(i)$, we set $Y_e(i) := Y_e(i-1)$.

Note that this definition makes sense since in step $i = 1$ we have $O(1) = E(K_n)$.

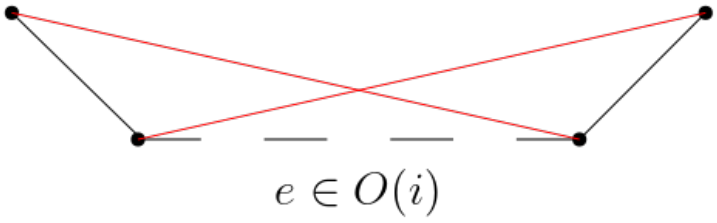


Figure 3.2.: The black edges are contained in the graph G_i. The red edges are in $O(i)$ and are contained in $Y_e(i)$.

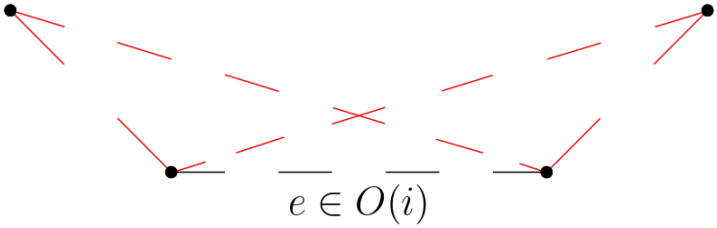


Figure 3.3.: The black edge is contained in $O(i)$. The red edges are in $O(i)$ and are contained in $X_e(i)$.

Further, we define the average number of Y-neighbors of an open edge $e \in O(i)$ by

$$\overline{Y}(i) := \frac{1}{Q(i)} \sum_{e \in O(i)} Y_e(i). \tag{3.13}$$

In general, how one can think of these random variables is that $Y_e(i)$ simply defines the distribution of the number of closed edges that occur when the edge $e \in O(i)$ gets inserted. An examples is shown in Figure 3.2. Before we give more intuition for these variables, we introduce another collection of random variables which are related to the the family $\{Y_e(i) \ : \ e \in O(i)\}$.

Definition 3.1.6. Let $i \in \mathbb{N}$. Two open edges $e, f \in O(i)$ are called X-neighbors in G_i if e and f form two sides of a triangle where the third side of the corresponding triangle is also contained in $O(i)$.

Definition 3.1.7. For every edge $e \in E(K_n)$ and $i \in \mathbb{N}$ we define the random variable $X_e(i)$ by

$$X_e(i) := |\{f \in O(i) \ : \ e \text{ and } f \text{ are } X\text{-neighbors in } G_i\}| \tag{3.14}$$

for $e \in O(i)$. If $e \in E(K_n) \setminus O(i)$, we set $X_e(i) := X_e(i-1)$.

Again, note that this definition makes sense since in the beginning of the process every edge is open. We show an example in Figure 3.3. Further, we are also interested in the average number of X-neighbors of an open edge $e \in O(i)$ for this family of random

variables, and hence we define

$$\overline{X}(i) := \frac{1}{Q(i)} \sum_{e \in O(i)} X_e(i). \tag{3.15}$$

Those will be the basic parameters needed to control the rate of change of $Q(i)$, but controlling these parameters turns out to be challenging. To give an overview of how this works, we introduce now the general setting. Let A be any graph parameter and let G be any graph. In general, we want to determine the rate of change of a parameter conditioned to the past. This means we define the function $\Delta A(G) : E[K_n] \to \mathbb{R}$ by

$$\Delta A(G)(e) \mapsto A(G \cup \{e\}) - A(G).$$

To move this setting to our random graph process we set $A(i) := A(G_i)$ and get

$$\mathbb{E}[\Delta A(i)] = \mathbb{E}[A(i+1) - A(i)|G_i],$$

where the expectation is over the uniformly at random chosen edge of G_i in the i-th step of the process. Since the process is Markovian, all information we need is encoded in the graph G_i, which is also intuitively clear from the definition of the triangle-free process. With this setting, our goal is to control $\mathbb{E}[\Delta Q(i)]$ and for this we need our family of random variables $\{Y_e(i) \ : \ e \in O(i)\}$. In fact, we get

$$\mathbb{E}[\Delta Q(i)] = -\overline{Y}(i) - 1. \tag{3.16}$$

This is simply the case since the variables $Y_e(i)$ determine the distribution of the number of edges that turn closed if edge $e \in O(i)$ is inserted to the graph. We then have to subtract 1 because the edge that gets inserted into G_i is counted twice. Once because it is no longer open and once because it is added to G_i. So the key to the proof of Theorem 3.1.1 is to control the variables $Y_e(i)$. For this, we need the random variables $X_e(i)$. This follows from the fact that one can quickly see the rate of change of the variable $Y_e(i)$ is governed by the equation

$$\mathbb{E}[\Delta Y_e(i)] = \frac{1}{Q(m)} \left(- \sum_{f \in Y_e(i)} Y_f(i) + X_e(i) \right). \tag{3.17}$$

As already mentioned, the intuition for this situation should be $G(n,i)$. This motivates the following result.

Lemma 3.1.5. Let $n \in \mathbb{N}$ and let $0 \leq i \leq \binom{n}{2}$. Further, let $G \sim G(n,p)$ be a graph obtained by the Erdös-Renyi model with edge density $p = i/\binom{n}{2}$. Then for n large enough and $e \in O(i)$ we have

1. $\mathbb{E}[X_e(i)] = 2(n-2)(1-p^2)^{(2n-4)} \approx 2e^{-8t^2}n$ and
2. $\mathbb{E}[Y_e(i)] = 2(n-2)p(1-p^2)^{(n-2)} \approx 2npe^{-4t^2} \approx 4te^{-4t^2}\sqrt{n}$.

Proof. We show 1 first. Let p be the edge density of G. To compute $\mathbb{E}[X_e(i)]$ we have to look at potential triangles containing the edge $e = (v_1, v_2)$. For this we have to look at the other $n-2$ vertices as they are the potential third vertex in the triangle. For every vertex $v_i \neq v_1, v_2$ we get

$$\begin{aligned}\mathbb{P}[(v_1,v_i) \text{ and } e \text{ are } X\text{-neighbors}] &= \mathbb{P}[(v_1,v_i) \in O(i)] \cdot \mathbb{P}[(v_i,v_2) \in O(i)] \\ &= (1-p^2)^{(n-2)}(1-p^2)^{(n-2)} \\ &= (1-p^2)^{(2n-4)},\end{aligned} \tag{3.18}$$

where the second equality follows from Lemma 3.1.2. Analogously, we get

$$\mathbb{P}[(v_2,v_i) \text{ and } e \text{ are } X\text{-neighbors}] = (1-p^2)^{(2n-4)}. \tag{3.19}$$

We define now the random variables V_i for $i \in \{3, \ldots, n\}$ by

$$V_i := |\{f \in O(m) \ : \ f \text{ is } X\text{-neighbor of } e \text{ and } v_i \in f\}|. \tag{3.20}$$

With our computations above we get, using indicator random variables,

$$\begin{aligned}\mathbb{E}[V_i] &= \mathbb{P}[(v_1,v_i) \text{ and } e \text{ are } X\text{-neighbors}] + \mathbb{P}[(v_2,v_i) \text{ and } e \text{ are } X\text{-neighbors}] \\ &= 2(1-p^2)^{(2n-4)}\end{aligned}. \tag{3.21}$$

Further, by linearity of expectation

$$\mathbb{E}[X_e(i)] = \sum_{i=3}^{n} \mathbb{E}[V_i] = 2(n-2)(1-p^2)^{(2n-4)} \approx 2ne^{-8t^2},$$

where the last step follows as in Equation 3.10.

Now we show 2. With the same notation as above we compute

$$\begin{aligned}\mathbb{P}[(v_1,v_i) \text{ and } e \text{ are } Y\text{-neighbors}] &= \mathbb{P}[(v_1,v_i) \in O(i)] \cdot \mathbb{P}[(v_i,v_2) \in E(G)] \\ &= (1-p^2)^{(n-2)}p\end{aligned}$$

and again we get analogously

$$\mathbb{P}[(v_2, v_i) \text{ and } e \text{ are } Y\text{-neighbors}] = (1-p^2)^{(n-2)}p.$$

If we define now for $i \in \{3, \dots, n\}$ the random variable U_i by

$$U_i := |\{f \in O(m) \ : \ f \text{ is } Y\text{-neighbor of } e \text{ and } v_i \in f\}|,$$

we get

$$\begin{aligned}\mathbb{E}[U_i] &= \mathbb{P}[(v_1, v_i) \text{ and } e \text{ are } Y\text{-neighbors}] + \mathbb{P}[(v_2, v_i) \text{ and } e \text{ are } Y\text{-neighbors}] \\ &= 2(1-p^2)^{(n-2)}p\end{aligned}$$

and hence by linearity of expectation

$$\mathbb{E}[Y_e(i)] = \sum_{i=3}^{n} \mathbb{E}[U_i] = 2(n-2)p(1-p^2)^{(n-2)} \approx 2npe^{-4t^2}.$$

□

In fact, the authors show that this is actually close to the truth. In particular, they show the following two results.

Theorem 3.1.6 ([Fiz Pontiveros et al., 2020], Theorem 2.4)**.** With high probability

$$\overline{X}(i) \in \left(1 \pm \frac{e^{2t^2}(\log n)^3}{n^{1/4}} \cdot 2e^{-8t^2}n\right)$$

and

$$\overline{Y}(i) \in \left(1 \pm \frac{e^{2t^2}(\log n)^3}{n^{1/4}}\right) \cdot 4te^{-4t^2}\sqrt{n}$$

for every $\omega \cdot n^{3/2} < i \leq (\frac{1}{2\sqrt{2}} - \epsilon)n^{3/2}\sqrt{\log n}$, where $\omega = \omega(n)$ is a function which goes to infinity sufficiently slowly as $n \to \infty$.

Again note that the absolute error is descreasing super-exponentially quickly.

Theorem 3.1.7 ([Fiz Pontiveros et al., 2020], Theorem 2.5)**.** With high probability,

$$Y_e(i) \in 4te^{-4t^2}\sqrt{n} \pm n^{1/4}(\log n)^3$$

for every open edge $e \in O(G_i)$ and every $i \leq \omega \cdot n^{3/2}$, and

$$Y_e(i) \in \left(1 \pm e^{2t^2} n^{-1/4} (\log n)^4)\right) \cdot 4te^{-4t^2}\sqrt{n}$$

for every open edge $e \in O(G_i)$ and every $\omega \cdot n^{3/2} < m \leq (\frac{1}{2\sqrt{2}} - \epsilon)n^{3/2}\sqrt{\log n}$, where $\omega = \omega(n)$ is a function as in Theorem 3.1.6.

To show Theorems 3.1.6 and 3.1.7 the authors use Equation 3.17 and count walks in certain graphs they construct. For this they also use techniques to count graph structures in the graphs G_i. Further, one of the main obstacles in the proofs is to find good bounds for $\mathrm{Var}(Y_e(i))$ and $\mathrm{Cov}(X_e(i), Y_e(i))$, since this is needed to control the rate of change of $X_e(i)$ and $Y_e(i)$. Nonetheless, explaining these methods here would be beyond the goal of this work and can be found in [Fiz Pontiveros et al., 2020], Section 2.

Finally, the authors show the following result.

Theorem 3.1.8 ([Fiz Pontiveros et al., 2020], Theorem 2.12)**.**

$$\Delta(G_{\triangle,n}) = \left(\frac{1}{\sqrt{2}} + o(1)\right)\sqrt{n \log n} \quad \textit{and} \quad \alpha(G_{\triangle,n}) \leq (\sqrt{2} + o(1))\sqrt{n \log n} \tag{3.22}$$

with high probability as $n \to \infty$.

The main idea for the first statement is to look at certain sets S and look at the probability that this set is the neighbourhood of a vertex in the resulting graph G_{m^*}, and then sum up those probabilities. However, the challenge is to track the number of open edges inside S. Here there are two cases. In the first case the set S has an intersection with a low number of neighbourhoods from vertices outside S. In the second case, S has an intersection with a lot of neighborhoods. This second case turns out to be more challenging. Nonetheless, we can show with this result the following.

Lemma 3.1.9. Theorem 3.1.8 implies the upper bound in Theorem 3.1.1.

Proof. By Theorem 3.1.8 we have

$$\Delta(G_{\triangle,n}) = \left(\frac{1}{\sqrt{2}} + o(1)\right)\sqrt{n \log n}$$

with high probability as $n \to \infty$. Hence, we get

$$\begin{aligned} e(G_{\triangle,n}) &= \frac{1}{2} \sum_{v \in V(G_{\triangle,n})} d_{G_{\triangle,n}}(v) \leq \frac{1}{2} \sum_{v \in V(G_{\triangle,n})} \Delta(G_{\triangle,n}) \\ &= \frac{1}{2} \sum_{v \in V(G_{\triangle,n})} \left(\frac{1}{\sqrt{2}} + o(1) \right) \sqrt{n \log n} \\ &= \frac{n}{2} \left(\frac{1}{\sqrt{2}} + o(1) \right) \sqrt{n \log n} \\ &= \left(\frac{1}{2\sqrt{2}} + o(1) \right) n^{3/2} \sqrt{\log n} \end{aligned}$$

with high probability as $n \to \infty$. □

3.2. Simulation of the Triangle-free Process

In this section, we present an algorithmic approach to simulate the triangle-free process for different numbers of vertices. Since the main goal of the algorithmic part of this work is to get an idea of the main properties of the random graph processes presented here, the goal for this section is to see how reliable our simulation results are. The triangle-free process is well suited for this purpose since is the most studied of the random graph processes presented in this work.
The rest of this section is structured as follows. First, we give an overview on the algorithms and discuss the computational bottle necks. Thereby, we see that the triangle-free process can be viewed in different ways leading to different algorithms. Afterwards, we present our simulation results for the final number of edges of the final graph. Finally, we discuss how well the experimental results fit the theory presented in Section 3.1.

Algorithms for the Triangle-free Process

In this section, we describe the algorithms used for our simulations. Intuitively, by the definition of the triangle-free process we would have to maintain a data structure for our graph $G_i = (V_i, E_i)$ which is generated during the process and a data structure for the open edges $O(i)$, see notation in Section 3.1. Hence, one possible algorithm would be to start with $O(0) := E(K_n)$ and $G_0 := ([n], \emptyset)$. Then in every step $i \geq 1$ we can choose one edge e_i uniformly at random from $O(i)$, set $G_i = G_{i-1} \cup \{e_i\}$ and update the set of open edges $O(i)$. However, this has the downside that we would have to start with $O(n^2)$ edges in our memory which is a problem if n gets large when we update this set over and over again. Hence, we look at an alternative description for the triangle-free process.

Algorithm 1: Rejection version of the triangle-free process

Data: Empty graph $G = (V_0 = [n], E_0 = \emptyset)$
Result: Resulting graph $G_{\triangle,n}$ of the triangle-free process

```
1 i ← 0;
2 while (O(i) > 0) do
3     Edge e ← randomEdge;
4     if e ∈ E[G_i] || e ∈ C(i) then
5         reject e;
6     else
7         G_i ← G_i ∪ {e};
8         i ← i + 1;
```

For this, we observe that the defining property of this random graph process is that in every step i every edge in $O(i)$ has the same probability to be chosen. Consequently, if we choose, in every step i, a candidate edge $e \in E(K_n)$ uniformly at random and reject it if $e \in G_{i-1}$ or $e \in C(i)$ until we find a candidate edge $e_i \in E[K_n]$ with $e_i \in O(i)$, every open edge still has the same probability

$$\mathbb{P}[\text{open edge gets chosen}] = \frac{|O(i)|}{\binom{n}{2}}$$

to get chosen. Consequently, we see that this version of the triangle-free process is equivalent to the triangle-free process as stated in Section 3.1. Note that of course this process is only well defined if the process has not come to an end. We describe this in Algorithm 1. Further, note that for an edge $e = (u, v) \in C(i)$, inserting this edge would close a triangle in G_{i-1}. However, this is precisely the case if $N(u) \cap N(v) \neq \emptyset$. The problem with this approach is of course that over time the size of the set of open edges decreases. Hence, the number of edge rejections will increase. Using our intuition the number of open edges in G_i should behave similar to the number of open edges in the Erdös-Renyi graph $G(n, p)$ with $p \approx 2i/n^2$. We expect by Lemma 3.1.2

$$|O(i)| \approx \binom{n}{2} e^{-p^2 n}$$

and hence

$$\mathbb{P}[\text{open edge gets chosen}] = \frac{|O(i)|}{\binom{n}{2}} \approx e^{-p^2 n} = e^{-\frac{4i^2}{n^3}}. \tag{3.23}$$

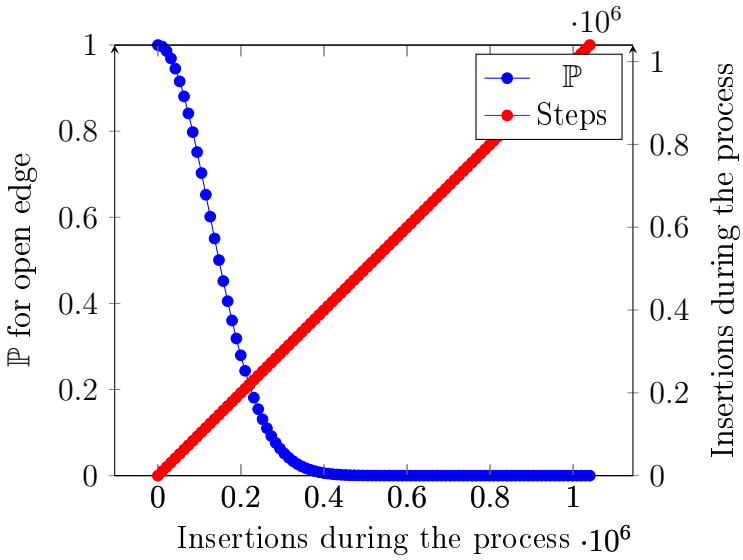
We can see how this behaves in Figure 8.

Algorithm 2: List version of the triangle-free process starting at step $i \in \mathbb{N}_0$

Data: Current graph $G = (V_i = [n], E_i)$, $O(i)$
Result: Resulting graph $G_{\triangle,n}$ of the triangle-free process

```
1 Õ(i) ← randomShuffle(O(i));
2 for e = (u, v) ∈ Õ(i) do
3     if N(u) ∩ N(v) = ∅ then
4         G_i ← G_i ∪ {e};
```

Evolution of the Probability for an open edge during the process on 5000 vertices



Hence, we see that this approach gets inefficient fast, especially for $i \geq n^3/2$. To resolve this, we look at the process again differently. Assume we are at step $i \in \mathbb{N}$ in the original triangle-free process. We look at the set of open edges $O(i)$. Instead of updating this set, we shuffle the open edges in a uniform order and then iterate over them. We insert the corresponding candidate edge if at the time when we reach it, it is still open. The algorithm describing this can be seen in Algorithm 2. Note that using $i = 0$, Algorithm 2 could also be used as an algorithm for the triangle-free process itself. However, this again has the downside that we would have to check $O(n^2)$ line 3 in Algorithm 2. This gets more and more inefficient when the graph grows. Hence, we look at an approach that combines Algorithm 1 and Algorithm 2. The idea behind this is the following. In the beginning of the process almost all edges are open and hence choosing an edge uniformly at random among all edges instead of choosing it among the open edges yields a high probability that the edge does not get rejected, see Equation 3.23. Hence, we start our algorithm by applying Algorithm 1 and we define a tuning parameter α which specifies when this phase ends. To be precise, α describes how many times we are allowed to

Algorithm 3: Triangle-free process using Algorihtm 1 and Algorithm 2

Data: Empty graph $G = ([n], \emptyset)$, tolerance α
Result: Resulting graph $G_{\triangle,n}$ of the triangle-free process

```
1  i ← 0;
2  steps ← 0;
3  openEdges = {};
4  while i ≤ α do
5      Edge e ← randomEdge;
6      while e ∈ E(G_steps) || e ∈ C(steps) do
7          reject e;
8          i ← i + 1;
9          if i > α then
10             break both while loops;
11     if e ∈ O(steps) then
12         G_steps ← G_steps ∪ {e};
13         steps ← steps+1;
14         i ← 0
15 for v ∈ [n] do
16     for u ∈ [n], u > v do
17         if (v, u) ∉ C(steps) and (v, u) ∉ E(G_steps) then
18             openEdges ← openEdges ∪ {(v, u)};
19 openEdges ← randomShuffle(openEdges);
20 for e ∈ openEdges do
21     if e ∉ C(steps) then
22         G_steps ← G_steps ∪ {e};
23         steps ← steps+1;
```

reject a candidate edge in a row. Afterwards, we switch to Algorithm 2. For this, we first have to build the list with all remaining open edges. For this, we iterate over every vertex in the graph. Then we look at every possible edge from this vertex and if the corresponding neighbor is neither already in the graph nor closed, we can add it safely to the graph. To check the existence of an edge we can use hash maps or just iterate over the neighborhood. With the simple observation that an edge $e = (u, v)$ is closed if and only if $|N(u) \cap N(v)| \geq 1$ we can check whether an edge is closed or not. Note that, for this criteria, we do not have to compute the whole intersection of the neighborhoods. We can stop as soon as we found any vertex in the intersection. After we build the list of all open edges, we shuffle it uniformly at random and check every edge in it. Note that during this process some of the later edges can become closed and hence we have to check them again. The pseudocode of this approach is detailed in Algorithm 3.

Sample mean	102 920,71
Sample standard deviation	83,92
Relative sample standard deviation	0,08
Sample variance	7 042,10

Table 3.1.: Some statistical results for the triangle-free process on 2 000 vertices with 200 repetitions.

3.2.1. Experimental Evaluation

In this section, we present the results of our simulations for the triangle-free process. Since we already pointed out the advantages and disadvantages of the different approaches in the previous section, we only use Algorithm 3 for the simulation of the triangle-free process with different values for the tuning parameter α which describes the number of rejections that we tolerate in a row. The probabilistic parts of the simulation are done using pseudo-randomness with different random seeds. The overall goal in this section is to see how close our experiments are to the theoretical observations that are presented in Section 3.1. This is important to see how reliable our predicitions are for the processes where only weaker results are known. The rest of this section is structured as follows. First, we want to observe some statistical data for the triangle-free process to see what standard deviation we can expect for the process. This is an important parameter in order to interpret the results of the simulations. Afterwards, we look at the influence of the tuning parameter α on running time and maximal memory consumption. Note that since Algorithm 3 is equivalent to the original triangle-free process, this parameter has no influence on the overall result quality of the process. Afterwards, we apply different models to our results to see how good the results fit Theorem 3.1.1. This is done using the least squares method and we apply some measures to see how good the models fit our results.

Statistical results. To evaluate the reliability of our simulation results, we have to know some statistical properties for the random variable $e(G_{\triangle,n})$. The reason is simply that we can not say how precise our results are without knowing which standard deviation we have to expect. Hence, we run 200 repetitions of the triangle-free process on 2 000 vertices with a fixed parameter α and look at different statistical parameters. This is shown in Table 3.2.1. Note that the relative standard deviation is given in percent. Hence, we can see that we do not have to expect a large standard deviation. In fact, it is less than one percent which is a solid base for our evaluation of the simulations. Further, it shows that we can expect to get reliable results with three repetitions with different random seeds for the process for every n under consideration. Note that we used every

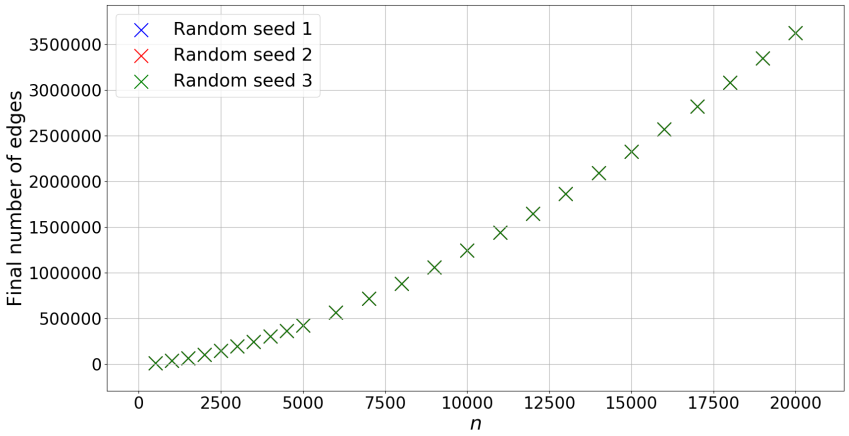


Figure 3.4.: Every repetition of the process simulation on n vertices with a different random seed and a different value for α.

time a different rejection parameter since the probabilistic behaviour of the process is not influenced by that. Table 3.2.1 is further underlined by Figure 3.4 where we plot every result of the process as one data point. We can see that the vertical distances between the data points are very low. Hence, we can also expect that the curve that we get by the data points does not change much for larger values of n.

Influence of the parameter α. Now we want look at the influence of the tuning parameter α on the performance of our algorithm. As we already described in the previous section, we can expect that the rejection phase gets towards the end of the algorithm inefficient. This is because of the decreasing size of the number of open edges $|O(i)|$ after an edge insertion. We show the results for different numbers of rejections in Table 3.2.1. What we can see is that, as we expected, a larger number of rejections in a row lead to less memory consumption since the size of the list of still open edges $O(\alpha)$ is smaller if we exclude more edges via the rejection process. However, we can see that this effect deminishes for a larger number of rejections in a row. This can be attributed to the fact that as the process progresses, the probability of selecting an open edge by randomly choosing one edge is reduced. This can be seen in Figure 8. In combination with the observation that we get a significant time increase for more rejections allowed, we draw the conclusion that it is useful to apply the rejection parameter in a way such that it fits the memory of the used machine but it is not useful in order to optimize the time performance of the simulation in this case.

Models for the simulation data. Our main goal of the simulation experiments

n	$\alpha = 500$		$\alpha = 2\,500$		$\alpha = 5\,000$	
	m[KB]	t[s]	m[KB]	t[s]	m[KB]	t[s]
10 000	26 092	561,16	19 500	1 052,66	18 464	1 715,20
11 000	38 472	711,50	30 784	1 398,17	29 796	1 762,4
12 000	44 016	898,73	34 540	1 612,53	33 164	2 429,35
13 000	48 520	1 097,83	37 052	2 095,62	37 020	3 058,24
14 000	54 900	1 329,51	41 244	2 524,36	39 388	3 667,84
15 000	56 608	1 651,21	47 948	2 671,62	41 684	3 863,99
16 000	61 992	2 003,97	45 976	3 436,29	43 992	4 742,15
17 000	67 428	2 305,36	52 196	3 747,84	48 080	5 199,79
18 000	77 744	2 626,07	62 860	3 607,06	50 504	6 516,96
19 000	80 336	3 113,47	55 992	5 116,96	51 908	6 616,3
20 000	84 768	3 676,14	60 088	5 795,20	55 988	8 804,72

Table 3.2.: Influence of the rejection parameter α on the maximal memory consumption and the overall time for the simulation of the triangle-free process on different numbers of vertices n.

is to see how good our approximation results fit the theory predictions in Section 3.1. In particular, we want to see how good our data fits the exponents occuring in the leading term

$$\left(\frac{1}{2\sqrt{2}} + o(1)\right) n^{3/2} (\log n)^{1/2} \tag{3.24}$$

of the number of edges for the final graph of the triangle-free process. Further, we are also interested in the prediction for the constant factor. For this, we fit different models to our data using the non-linear least squares method. For the triangle-free process we define

$$a^* := \frac{1}{2\sqrt{2}}, \quad b^* := \frac{3}{2}, \quad c^* := \frac{1}{2}. \tag{3.25}$$

First we look at the model

$$f_1(n) := a \cdot n^b \cdot \log n^c \tag{3.26}$$

to see which values for the leading term our data predicts. Note that it is possible that there is a second order term which influences our predictions. We handle this in another model. This fit can be seen in Figure 3.5. The values for the three parameters we want to predict are

$$a = 0.49, \quad b = 1.50, \quad c = 0.44. \tag{3.27}$$

Further, we measure the quality of our models with the standard error of our regression which is defined by

$$S = \sqrt{\frac{\sum_{i=1}^{m} (y_i - f(x_i))^2}{n - k}}, \tag{3.28}$$

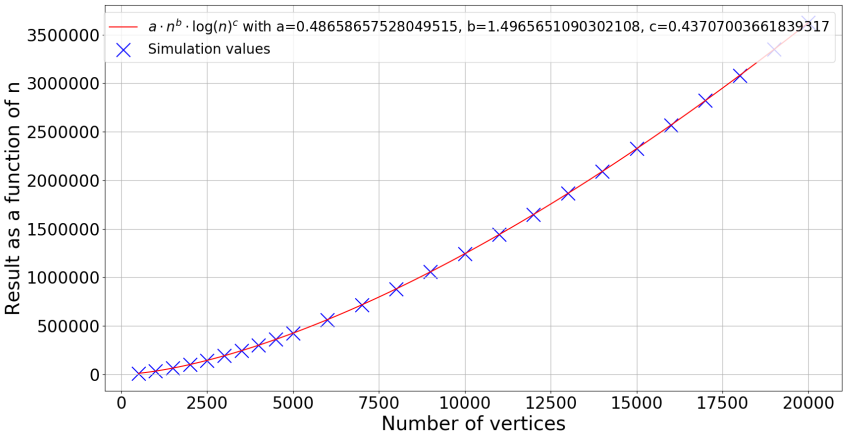


Figure 3.5.: Simulation values together with the predicted model for the model $f_1(n)$.

where k is the number of parameters in the model, y_i are the observed values and $f(x_i)$ the corresponding model values. A lower value for S indicates a better fit of the model. This value basically just measures how far our data points are from the regression line. Note that other values, like the coefficient of determination R^2, are not suitable since we use the non-linear least squares method which is a non-linear regression method [Spiess and Neumeyer, 2010]. For this model, we get

$$S = 121{,}46. \tag{3.29}$$

We will see how this compares to the other models used. The next model that we use is

$$f_2(n) = a \cdot n^{3/2} (\log n)^{1/2}. \tag{3.30}$$

Given the right exponents in the leading term the hope is to measure the constant well. For this model we get a standard error of

$$S = 4\,671{,}47 \tag{3.31}$$

with the value

$$a = 0.41. \tag{3.32}$$

Even though we see that the model fits the date worse, we can see that we are closer to the constant factor in 3.25. Further, we can see in Figure 3.6 that the fit is still good. The last model we want to look at is

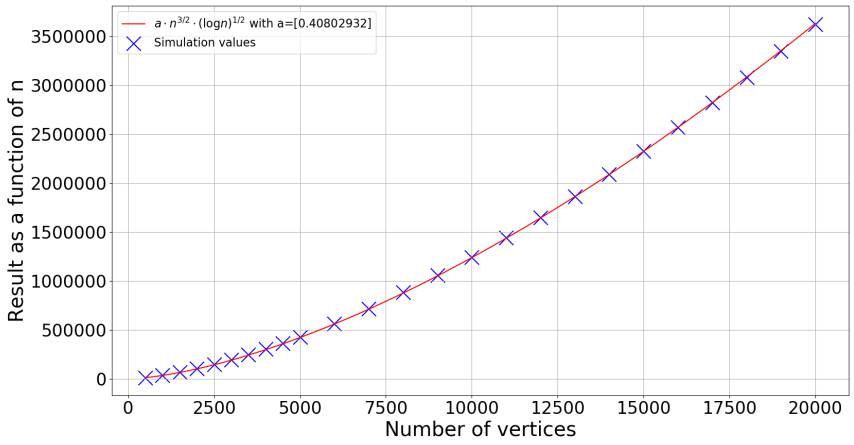


Figure 3.6.: Simulation values together with the predicted modelfor the model $f_2(n)$.

$$f_3(n) := a \cdot n^{3/2}(\log n)^{1/2} + b \cdot n^{3/2} \tag{3.33}$$

to take the possibility of a second order term, which influences the simulations, into account. With this model, wie achive a similar fit as for $f_1(n)$ with a standard error

$$S = 165{,}47 \tag{3.34}$$

and predicted parameters

$$a = 0.33, \quad b = 0.24. \tag{3.35}$$

Hence, we see that the second order term in the triangle-free process might play a role for the values of n under consideration. However, this model gets close to the values predicted by Theorem 3.1.1 with respect to the constant factor. Since it also gives us a good fit of the values, it is also a good candidate for the estimation of the constant factors in Chapter 5 and Chapter 7. The graph for this model together with the simulated values can be seen in Figure 3.7.

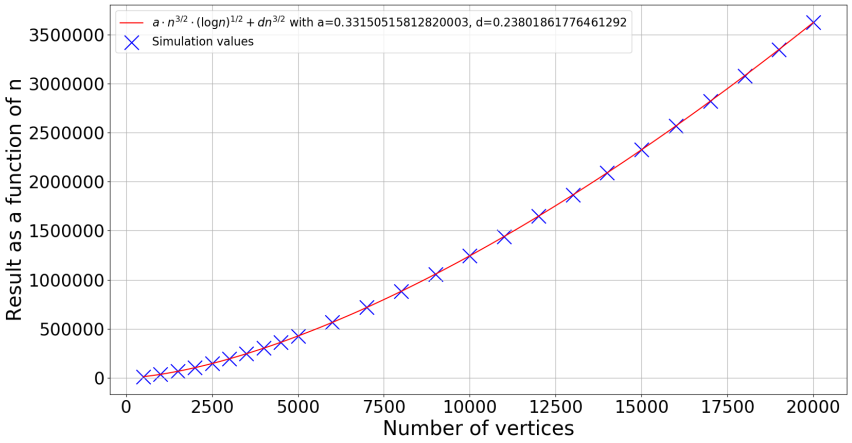


Figure 3.7.: Simulation values together with the predicted model for the model $f_3(n)$.

4. The Triangle-removal Process

In this chapter, we present the triangle-removal process which is a special case of a random graph process. For this process we pursue a similar objective as for the triangle-free process but we note that there are clear differences both in theory and in practice. We begin by introducing the process itself and present some theoretical results in Section 4.1. Further, we look at the differences in the processes in Section 4.2.1 where we present the data obtained by our simulation.

4.1. Theoretical Results

In this section, we mainly follow the work of Bohman et al. [Bohman et al., 2015]. For the triangle-removal process on $n \in \mathbb{N}$ vertices we define $G_0 = K_n$. Then, for every $i \in \mathbb{N}$, the graph G_{i+1} is obtained from the graph G_i by choosing among all triangles in G_i one triangle uniformly at random and deleting all its edges. The resulting random graph process $(G_i)_{i\in\mathbb{N}}$ is called the *triangle-removal process*. With the same arguments as for the triangle-free process it is obvious that this process becomes stationary at some point, i.e., $G_j = G_{j+1}$ for all $j \in \mathbb{N}$, $j \geq N$ for some $N \in \mathbb{N}$ large enough. Obviously, this is the case if the process produces a graph which contains no triangle. We call this graph the resulting graph of the process and denote it by G_{τ_0} where τ_0 is defined by

$$\tau_0 = \min_{i\in\mathbb{N}_0}\{i \ : \ G_i \text{ contains no triangles}\}.$$

Alternatively, we also denote this graph by $G^r_{\triangle,n}$. An example for this process is shown in Figure 4.1. The objective for this process is to determine the number of edges of the resulting graph G_{τ_0}. Note that it is enough to know the index τ_0 as we can simply compute the number of edges in G_{τ_0} by

$$e(G_{\tau_0}) = \binom{n}{2} - 3 \cdot \tau_0.$$

This is because we delete three edges from the graph in every step. Similarly to the triangle-free process, the triangle-removal process was also introduced with the motiva-

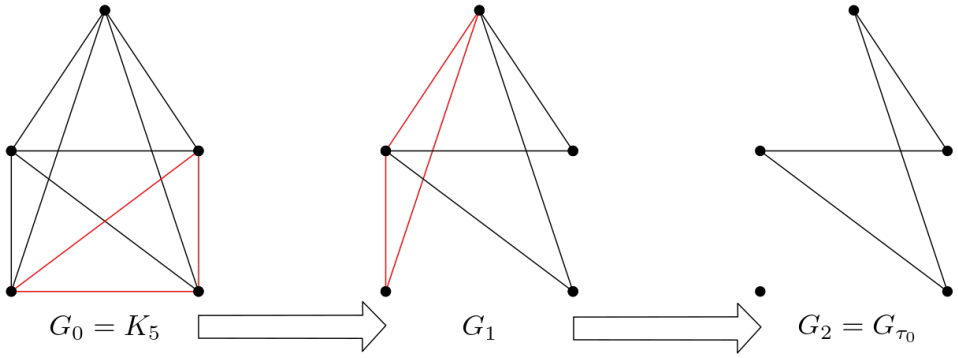


Figure 4.1.: Simple example for the triangle-removal process on five vertices.

tion to determine the Ramsey number $R(3,t)$. In 1990, Bollobás and Erdös conjectured that $e(G^r_{\triangle,n})$ is of order $n^{3/2}$. This can be found for example in [Bollobás, 1997]. In their work, they also suggested that a similar intuition as for the triangle-free process holds for this process. The obtained graphs should be similar to an Erdös-Rényi graph with the same edge density. However, the first non-trivial result for this process was shown by Spencer in 1995 [Spencer, 1995] showing that $e(G^r_{\triangle,n}) = o(n^2)$ with high probability. This was also shown independently by Rödl and Thoma in 1996 [Rödl and Thoma, 1996]. This was improved successively over the following years. Grable [Grable, 1997] showed an upper bound for the final number of edges of $n^{11/6+o(1)}$ with high probability. Further, he suggest that similar arguments can be used to even improve that to a bound of $n^{7/4+o(1)}$ with high probability. In 2011, Bohman et al. [Bohman et al., 2011] wer even able to go beyond this exponent, showing that the final number of edges can be bounded by $n^{5/3+o(1)}$ with high probability. Finally, in the work that we present in this section, Bohman et al. [Bohman et al., 2015] are able to show the conjecture from Bollobás and Erdös, showing that the final number of edges is $n^{3/2+o(1)}$ with high probability.

A natural question that rises is whether the triangle-free process and the triangle-removal process are equivalent or at least similar in any measurable way. Both processes generate a triangle-free graph when they get stationary. To make this similarity even clearer, we can rephrase the triangle-removal process. In a complete graph on n vertices K_n we have $\binom{n}{3}$ triangles as every triple of vertices forms a triangle. If we now take a uniform ordering of these triangles, we can iterate over the list of all triangles and check whether the triangle is in the graph. If this is the case, we delete its edges and look at the next triangle. Otherwise, we check the next triangle. The resulting process is equivalent to the triangle-removal process as defined above. This observation plays an important role for our simulation of the triangle-removal process in Section 4.2. Further, we obtain another similarity to the triangle-free process with this version. In every step,

we have a set of legal triangles corresponding to the triangles that are still in the graph and a set of illegal triangles. The illegal triangles are the set of triangles which have at least one of their edges already removed. This is similar to the concept of open edges for the triangle-free process. This division of the edges is one of the key considerations for the similarity of the graphs within the triangle-free process and graphs obtained by the Erdös-Renyi model. As long as the number of illegal edges is low compared to the number of open edges we get approximately an almost uniform insertion of the edges for the triangle-free process where the illegal edges are the closed edges. As this is a helpful insight in terms of the analysis of the process, the hope is that a similar intuiton holds for the triangle-removal process. However, the relation between legal and illegal triangles is changing faster and hence the analysis proves to be more difficult in this sense [Bohman et al., 2015]. We pick up on this in Section 4.2.1 where we also start the process from a $G(n,p)$ with different edge densities $p \in [0,1]$.

Intuitively, one could expect the resulting graph G_{τ_0} to have similar properties as the Erdös-Renyi graph with the same edge density except the fact that it has no triangles. To see this, we start with the following lemma.

Lemma 4.1.1. Let $n \in \mathbb{N}$, $0 \leq m \leq \binom{n}{2}$. Further, let $G \sim G(n,p)$, where $p = m/\binom{n}{2}$. Then

$$\mathbb{E}[\#\text{triangles in } G] = \binom{n}{3} p^3. \tag{4.1}$$

Proof. In K_n there are $\binom{n}{3}$ triangles because K_n is complete. Further, the probability for every possible triangle in G is p^3 since we need three edges per triangle. This probability follows from the definition of the Erdös-Renyi model. Additionally, we define for every triple (u,v,w) with $u,v,w \in V(G)$ the random indicator variable

$$\mathbf{1}_{[(u,v,w) \text{ is triangle in } G]}.$$

Then we get

$$\mathbb{E}[\mathbf{1}_{[(u,v,w) \text{ is triangle in } G]}] = \mathbb{P}[(u,v,w) \text{is triangle in } G] = p^3. \tag{4.2}$$

Hence, using linearity of expectation, we get

$$\begin{aligned} \mathbb{E}[\#\text{triangles in } G] &= \mathbb{E}[\sum_{\triangle \in K_n} \mathbf{1}_{\triangle \in G}] \\ &= \sum_{\triangle \in K_n} \mathbb{E}[\mathbf{1}_{\triangle \in G}]] \\ &= \binom{n}{3} p^3, \end{aligned}$$

where the last equality follows from our considerations above and 4.2. □

Now assume $n \in \mathbb{N}$, $m = \epsilon n^{3/2}$ and $G \sim G(n,p)$, where G has edge density $p = \frac{\epsilon n^3/2}{\binom{n}{2}} \approx 2\epsilon n^{-1/2}$. Hence, according to Lemma 4.1.1 G contains in expectation

$$\binom{n}{3}(2\epsilon n^{-1/2})^3 \approx \frac{n^3}{6}(2\epsilon n^{-1/2})^3 = \frac{4}{3}\epsilon^3 n^{3/2}$$

triangles. Thus, deleting all the triangles would delete less than $4\epsilon^3 n^{3/2}$ edges since we do not have to delete all triangles because they are not necessarily edge disjoint. If ϵ is small, this fraction of the edges is negligible and the resulting graph should still be close to $G(n,p)$ in its properties, except that it contains no triangles. Even though the results are not as strong as for the triangle-free process, the authors confirm this idea and show the following result.

Theorem 4.1.2 ([Bohman et al., 2015], Theorem 1)**.** Let τ_0 be the minimum number of steps it takes the triangle-removal process to terminate. Then with high probability as $n \to \infty$ we get

$$\tau_0 = \frac{n^2}{6} - n^{3/2+o(1)}. \tag{4.3}$$

As we already mentioned before, this result is enough to compute $e(G_{\tau_0})$ and we get

$$e(G_{\tau_0}) = \binom{n}{2} - 3\left(\frac{n^2}{6} - n^{3/2+o(1)}\right) = n^{3/2+o(1)} - \frac{n}{2}. \tag{4.4}$$

Hence, we have $e(G_{\tau_0}) = n^{3/2+o(1)}$ with high probability as $n \to \infty$. The remainder of this section is dedicated to providing an overview on the arguments of the proof of Theorem 4.1.2. Before we give some details, we will shortly mention the overall idea which is to track a large collection of random variables that are designed to support the analysis of the process. We will see that it does not suffice to only track the basic random variables in this process, and hence a large number of random variables will be necessary. One main ingredient for this analysis is to show that these variables have a self-correcting nature. This means that the further the variable deviates from its trajectory (which has to be determined), the stronger it drifts back towards its mean. The challenge in doing so is that these random variables are interacting with each other and hence the control over the error is much harder. This problem is resolved using a martingale technique [Bohman et al., 2015].

Upper bound

For the upper bound, the authors start by establishing a system of martingales to track a family of random variables. In the beginning, these variables are the co-degrees $Y_{u,v} := |N(u) \cap N(v)|$ for every pair of vertices $u \neq v \in V(G_i)$. These random variables are in turn used to control the random variables

$$Q(i) := \#\text{triangles in } G_i$$

for every $i \in \mathbb{N}$. How this works can be seen by the fact that

$$Q(i) := \frac{1}{3} \sum_{(u,v) \in E(G_i)} Y_{u,v}. \tag{4.5}$$

Equation 4.5 follows from the observation that we can count every triangle in a graph by summing over all edges $e = (u, v) \in E(G_i)$ and looking at how many triangles contain the edge e. This number is precisely given by $Y_{u,v}$. However, every triangle counted in this way is counted by the two other sides of the triangle as well and hence we count every triangle three times. Hence, the equation in 4.5 follows. For the co-degrees, we can already intuitively see why these variables have a self-correcting nature. Assume we have an edge $(u, v) \in E(G_i)$ such that $Y_{u,v}$ is higher than average. Then the probability that the triangle selected by the process in step $i+1$ is one that contains the edge (u, v) is also higher. Hence, the probability that $Y_{u,v}$ shrinks is higher and drives the random variable back to its mean. The analogous effect occurs if $Y_{u,v}$ is less than average. However, tracking only these variables will not suffice to follow the process longer than to the point where the graph has $n^{7/4}$ edges [Bohman et al., 2015]. We can see the triangle-removal process as a stochastic Markovian process which gives us a filtration which we call $(\mathcal{F})_{i \in \mathbb{N}_0}$. For every random variable X under consideration, let $\Delta X(i) := X(i+1) - X(i)$ be the rate of change in step $i \mapsto i+1$ of the variable X. Then by definition we have

$$\mathbb{E}[\Delta Q(i) \mid \mathcal{F}_i] = -\sum_{uvw \in Q} (Y_{u,v} + Y_{v,w} + Y_{u,w} - 2)/Q. \tag{4.6}$$

This can be seen as follows. If we delete a triangle $(u, v, w) \in G_i$ from the graph G_i, we also delete all triangles sitting above the edges (u, v), (u, w) and (v, w). Thereby, we count the triangle (u, v, w) itself three times and hence the average number of triangles

we delete in this step is

$$\sum_{uvw \in Q} (Y_{u,v} + Y_{v,w} + Y_{u,w} - 2)/Q$$

and the formula in Equation 4.6 follows. We introduce now the following entities

$$t = t(i) = i/n^2, \quad p = p(i,n) = 1 - \frac{6i}{n^2} = 1 - 6t \Rightarrow t = \frac{1-p}{6}.$$

Note that with the approximation $\binom{n}{2} \approx n^2/2$, p is roughly the edge density of the graph G_i since this graph has edge density

$$p(i) = \frac{\binom{n}{2} - 3i}{\binom{n}{2}} = 1 - \frac{3i}{\binom{n}{2}} \approx 1 - \frac{6i}{n^2}.$$

In particular, we have

$$\begin{aligned} e(G_i) &= \binom{n}{2} - 3i = \binom{n}{2} - 3n^2 t = \binom{n}{2} - \frac{n^2(1-p)}{2} \\ &= \frac{1}{2}(n^2 - n - n^2(1-p)) = \frac{1}{2}(n^2 p - n). \end{aligned} \tag{4.7}$$

Since $e(G \sim G(n,p)) \approx n^2/2 \cdot p$ we see that p is in line with the edge density of the random graph model $G(n,p)$, up to a negligible linear term. This intuition motivates the following lemma.

Lemma 4.1.3. Let $n \in \mathbb{N}$, $0 \leq m \leq \binom{n}{2}$ and let $p := \frac{m}{\binom{n}{2}}$. Further, let $G \sim G(n,p)$ Then

$$\mathbb{E}[Y_{u,v}] = (n-2)p^2 \approx np^2 \tag{4.8}$$

for every pair $u \neq v \in V(G)$.

Proof. Let $u \neq v \in V(G)$ be arbitrarily chosen. Then for every $w \in V(G) \setminus \{u,v\}$ we get by the random graph model $G(n,p)$

$$\mathbb{P}[w \in N(u) \cap N(v)] = p^2.$$

Hence, using the according indicator random variables and linearity of expectation yields

$$\begin{aligned} \mathbb{E}[Y_{u,v}] &= \mathbb{E}[\sum_{w \in V(G)\setminus\{u,v\}} \mathbf{1}_{w \in N(u) \cap N(v)}] = \sum_{w \in V(G)\setminus\{u,v\}} \mathbb{P}[w \in N(u) \cap N(v)] \\ &= (n-2)p^2 \approx np^2. \end{aligned}$$

□

Now, from our intuition and by lemmas 4.1.1 and 4.1.3 we would expect

$$Y_{u,v}^{(i)} \approx np^2 \quad \text{and} \quad Q(i) \approx \frac{1}{6}n^3p^3. \tag{4.9}$$

Further, note that for the co-degrees $Y_{u,v}$ the evolution is determined by

$$\mathbb{E}[\Delta Y_{u,v}|\mathcal{F}_i] = -\sum_{w\in N(u)\cap N(v)} \frac{Y_{u,w} + Y_{v,w} - \mathbf{1}_{(u,v)\in E(G_i)}}{Q}. \tag{4.10}$$

This can be seen as follows. Obviously, the co-degree of $u \neq v$ can only be affected if a triangle gets deleted where one of the vertices is in $N(u) \cap N(v)$. Then there are essentially three cases to look at. The first case is that the deleted triangle is of the form (a, b, c) where, without loss of generality, we have $a = u$, $b = w$, $c \neq v$. In this case, either $c \in N(u) \cap N(v)$ or $c \notin N(u \cap N(v))$. In the first case $Y_{u,v}$ decreases by two which is caused by a decrease of $Y_{u,w}$ by one and a decrease of $Y_{v,w}$ by one. In the other case $Y_{u,v}$ decreases by one which is caused by a decrease of $Y_{u,w}$ by one. The next case is the symmetric case where u and v switch roles and the last cases is where the triangle (u, v, w) is chosen. There, we also get a decrease of one for $Y_{u,v}$. Averaging yields exactly Equation 4.9, where the term $\mathbf{1}_{(u,v)\in E[G_i]}$ has to be subtracted due to double counting in case the edge (u, v) is in the graph.

In general, we see that the important parameters we have to know are $\mathbb{E}[\Delta Q|\mathcal{F}_i]$ and $\mathbb{E}[\Delta Y_{u,v}|\mathcal{F}_i]$. However, the key point to the work of [Bohman et al., 2015] is that tracking only these parameters is not enough to follow the process long enough. Using only these variables the authors in [Bohman et al., 2010] managed to show an upper bound of $O(n^{7/4}(\log n)^{5/4})$. However, if we reach at some point an edge density of $p = n^{-1/4}$ we get problems with the standard deviation, see [Bohman et al., 2015]. Hence, we see that the main obstacle of the proof is to establish enough parameters to gain a precise upper bound. This results in the following theorem which is central to us.

Theorem 4.1.4 ([Bohman et al., 2015], Theorem 2.1)**.** Define $\zeta = \zeta(p, n)$ to be

$$\zeta = n^{-1/2}p^{-1}\log n \tag{4.11}$$

and for some (arbitrarily large) constant $\kappa > 0$ let

$$\tau_Q^* = \min\left\{t \ : \ \left|\frac{Q}{\frac{1}{6}n^3p^3} - 1\right| \geq \kappa\zeta^2\right\}. \tag{4.12}$$

Then for every $M \geq 3$ with high probability

$$\left| \frac{Y_{u,v}}{np^2} - 1 \right| \leq 3^{3M-1}\zeta \tag{4.13}$$

for every $u, v \in V(G)$ and all t such that $t \leq \tau_Q^*$ and $p(t) \geq n^{-1/2+1/M}$.

This theorem basically means that if we deviate too much in the number of triangles from what we would expect by our intuition, we can bound the co-degrees and hence also the remaining triangles. Together with the next result we are able to show the upper bound in Theorem 4.1.2.

Theorem 4.1.5 ([Bohman et al., 2015] Theorem 2.2)**.** Set $\Phi(p, n) = p^{-2/\log n} \log n$ and for some fixed $\alpha > 0$ let

$$\tau_Y^* = \min\{t \ : \ \exists u, v \in V(G) \text{ such that } |Y_{u,v} - np^2| > \alpha n^{1/2} p\Phi\}. \tag{4.14}$$

Then with high probability as $n \to \infty$ as long as $t \leq \tau_Y^*$ and $p(t) \geq n^{-1/2}(\log n)^2$ we have

$$|Q - n^3p^3/6| \leq \alpha^2 n^2 p \Phi^2. \tag{4.15}$$

This theorem, along with Theorem 4.1.4, enables us to deduce a smaller error for the remaining triangles from an error of the co-degrees.

Proof of the upper bound in Theorem 4.1.2. The idea of the proof is to apply the theorems 4.1.4 and 4.1.5 in tandem as follows. First note that for all $p \geq n^{-1/2}$ we get for $\Phi = p^{-2/\log n}$ the bounds

$$\frac{\Phi}{\log n} = p^{-2/\log n} \leq n^{-1/2 \cdot -2/\log n} = n^{1/\log n} = e$$

and

$$\frac{\Phi}{\log n} \geq 1$$

since $p \leq 1$. So, together we get

$$1 \leq \frac{\Phi}{\log n} \leq e. \tag{4.16}$$

Thus, applying Theorem 4.1.5 and Equation 4.15 yield

$$|Q - n^3p^3/6| \leq \alpha^2 n^2 p \Phi^2 \leq \alpha^2 n^2 p (\log n)^2 e^2 = C^2 n^2 p (\log n)^2,$$

where the second inequality follows from $\Phi \leq \log n \cdot e$ and $C := \alpha e$. This means that we get for the relative error

$$\left| \frac{Q}{n^3p^3/6} - 1 \right| \leq \frac{C^2n^2p(\log n)^2}{n^3p^3/6} = 6C^2n^{-1}p^{-2}(\log n)^2 = 6C^2\zeta^2, \tag{4.17}$$

where $\zeta = n^{-1/2}\log n/p$ is defined as in Theorem 4.1.4. With the notation of Theorem 4.1.4 in mind, this means that we are in the scenario $t \leq \tau^*_Q$. Hence, we get from Theorem 4.1.4, for every $M \geq 3$ with high probability

$$\left| \frac{Y_{u,v}}{np^2} - 1 \right| \leq 3^{3M-1}\zeta, \tag{4.18}$$

where we chose $\kappa = 6C^2 > 0$. Note that we are still in the situation $p(t) \geq n^{-1/2+1/M}$ and $p(t) \geq n^{-1/2}(\log n)^2$ as we start the process with $p = 1$. Now 4.18 gives us

$$\begin{aligned} |Y_{u,v} - np^2| &\leq \alpha\zeta np^2 = \alpha n^{-1/2}p^{-1}\log np^2n \\ &= \alpha n^{1/2}p\log n \leq \alpha n^{1/2}p\Phi, \end{aligned}$$

where the last inequality follows from $\Phi \geq \log n$ and $\alpha := 3^{3M-1}$. But this means again that we are in the situation $t \leq \tau^*_Y$ and hence we get again

$$|Q - n^3p^3/6| \leq \alpha^2n^2p\Phi^2$$

and by the same computations we get

$$\left| \frac{Q}{n^3p^3/6} - 1 \right| \leq \tilde{C}\zeta^2$$

with $\tilde{C} := 6C^2 = 6(\alpha e)^2$. Hence, we showed that we can apply both theorems in tandem and because of $n^{-1/2+1/M} \geq n^{-1/2} \cdot (\log n)^2$ for n large enough and M fixed we can proceed this way as long as $p \geq n^{-1/2+1/M}$. But at the point $p = n^{-1/2+1/M}$ we get

by Equation 4.15

$$\begin{aligned} Q &\leq \frac{1}{6}n^3p^3 + \alpha^2 n^2 p\Phi^2 \\ &= \frac{1}{6}n^3(n^{-1/2+1/M})^3 + \alpha^2 n^2 n^{-1/2+1/M}\Phi^2 \\ &= \frac{1}{6}n^{-3/2+3/M} + \alpha^2\Phi^2 n^{3/2+1/M} \\ &= n^{-3/2+3/M}\left(\frac{1}{6} + \alpha^2\phi^2 n^{-2/M}\right) \\ &\leq n^{-3/2+3/M}\left(\frac{1}{6} + \alpha^2 e\log n n^{-2/M}\right) \\ &= n^{-3/2+3/M}\left(\frac{1}{6} + o(1)\right) \end{aligned}$$

since $\alpha^2 e\log n n^{-2/M} \to 0$ and $\Phi \geq e\cdot\log n$. Further, note that Theorem 4.1.5 guarantees that the process is still active at that point because if $Q = 0$ then 4.15 would not hold. On the other hand we get by 4.7

$$\begin{aligned} E(G_i) &= \frac{1}{2}(n^2p - n) = \frac{1}{2}(n^2 n^{-1/2+1/M} - n) \\ &= \frac{1}{2}(n^{3/2+1/M} - n) = n^{3/2+1/M}\left(\frac{1}{2} - n^{-1/2-1/M}\right) \\ &= n^{3/2+1/M}\left(\frac{1}{2} - o(1)\right) \end{aligned}$$

since $n^{-1/2-1/M} \to 0$. Hence, we can only get less edges and the process is still active. As $M \geq 3$ can be arbitrarily large by Theorem 4.1.4 this results in an upper bound of $n^{3/2+o(1)}$ edges for the graph G_{τ_0}. □

Lower Bound

The goal of this section is to finish the proof of our main theorem 4.1.2 by providing a lower bound on the number of edges of the graph G_{τ_0} with high probability. This turns out to be achievable by only tracking the co-degrees with asymptotically tight estimates for a certain time of the process. The main result from which we derive our lower bound is the following.

Theorem 4.1.6 ([Bohman et al., 2015], Theorem 6.1)**.** Suppose that for some fixed $0 < \epsilon < 1/6$, all co-degrees satisfy $Y_{u,v} = (1 + o(1))np^2$ throughout $p \geq p_0 = n^{-1/2+\epsilon}$. Then with high probability the final number of edges of the graph G_{τ_0} is at least $n^{3/2-6\epsilon-o(1)}$.

For the proof we first need a straightforward lemma which we will just cite here. The interested reader can find the proof in [Bohman et al., 2015], Section 6.

Lemma 4.1.7 ([Bohman et al., 2015], Lemma 6.3)**.** Let τ_c denote the minimal time where $Y_{u,v} > 2(np^2 + n^{1/3})$ for some $u, v \in V(G)$. With high probability, for all $t \leq \tau_c$ such that $p(t) \geq n^{-3/5}$, the number of triangles satisfies

$$Q - \frac{1}{6}(n^3p^3 + n^2p) \leq n^{7/3}p^2.$$

Now we have everything that we need to show Theorem 4.1.6. In the following, we use Q for the set of triangles in the graph, as well as the number of triangles in the graph in order to make notation easier and hope that this causes not to much confusion for the reader.

Proof of Theorem 4.1.6: We start by fixing some arbitrary $0 < \epsilon < 1/6$ and let i_0 be such that $p_0 = p(i_0) = n^{-1/2+\epsilon}$. Now assume we condition on the state $\mathcal{F}_{i_0}$ and assume that up to this point we had $Y_{u,v} = (1 + o(1))np^2$ for all $u, v \in V(G)$. Note that if we delete a triangle from the graph, we also delete every triangle adjacent to it via an edge. Hence, we introduce some notation for this. Let $uvw \in Q$ and let $xyz \neq uvw$ be another triangle in the graph. We write $uvw \sim xyz$ if the triangles share an edge. Hence, if we delete the triangle uvw from the graph we also delete every triangle $xyz \sim uvw$. However, note that this is not an equivalence relation. Further, let

$$B_{uvw} = |\{xyz \in Q \ : \ xyz \sim uvw\}| = Y_{u,v} + Y_{u,w} + Y_{v,w} - 3.$$

The subtraction of three comes from the fact that we count the triangle uvw for every co-degree but we do not want to include the triangle itself to this set, see Figure 4.2. This definition together with our assumption on the co-degrees implies at time i_0

$$B_{uvw} = 3(1 + o(1))np_0^2 - 3 = 3n^{2\epsilon} + 3o(1)n^{2\epsilon} - 3 = n^{2\epsilon+o(1)} \tag{4.19}$$

for all Q triangles. Now we make the following observation. Assume we have a triangle $uvw \in Q(j-1)$ then at the end of round j we have either $uvw \notin Q(j)$ or $B_{uvw}(j-1) - B_{uvw}(j) \leq 4$ and the co-degree of at most two edges of uvw has changed. To see why this claim is true, we have to consider three cases. The first case is that in step j the triangle selected is either uvw itself or a triangle $xyz \sim uvw$. In both cases we see immediately that $uvw \notin Q(j)$. The second case is when the triangle deleted is vertex disjoint to the triangle uvw. In this case of course none of the co-degrees $Y_{u,v}$, $Y_{v,w}$ or $Y_{u,w}$ is affected since for this the deleted triangle would have to share at least one vertex with uvw and hence we get $B_{uvw}(j-1) - B_{uvw}(j) \leq 4$ as claimed. Now we consider the last case where the deleted triangle xyz and uvw share exactly one vertex.

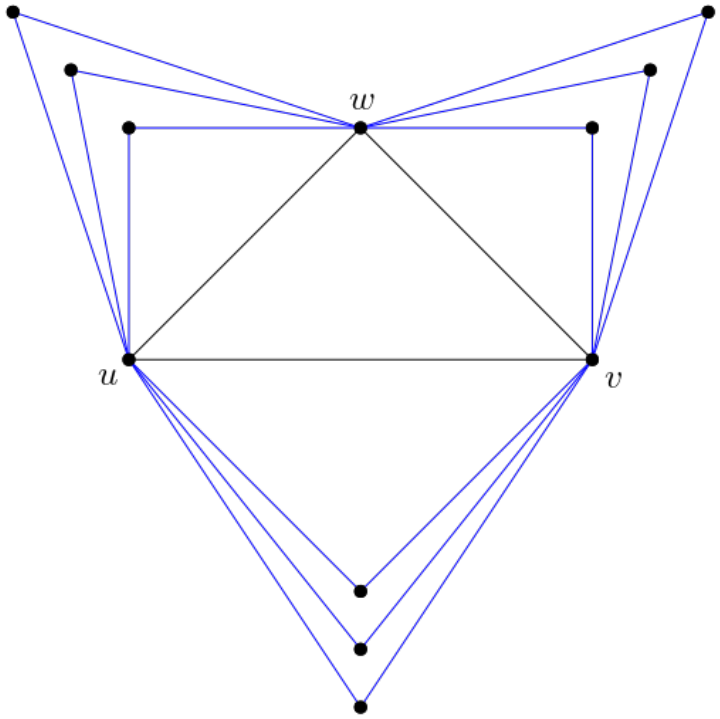


Figure 4.2.: The set B_{uvw} for a triangle $uvw \in Q$.

For this, consider, without loss of generality, $x = u$ and $\{y, z\} \cap \{v, w\} = \emptyset$. In this case, the only possible triangles that could be affected by the deletion of xyz, and hence could affect the co-degrees in the triangle uvw, are yuw, yuv, zuv and zuw. Hence, we also get in this case $B_{uvw}(j-1) - B_{uvw}(j) \leq 4$ as claimed. Further, we see that the co-degree of v and w was not affected and the overall claim follows.

For the further analysis of the lower bound, we construct a random subset $\mathcal{H} \subseteq Q(i_0)$. This works as follows. Let $\mathcal{H}(i_0) := \emptyset$ and we let $\mathcal{H}$ grow until the end of the process at step τ_0. We denote the final state of $\mathcal{H}$ as $\mathcal{H}^*$. To define how $\mathcal{H}$ grows we define another auxiliary set $\mathcal{Y}$ with $\mathcal{Y}(i_0) := \emptyset$. Further, we define for a triangle $uvw \in Q$ the sets

$$\mathcal{N}_1(uvw) := \{xyz \in Q \ : \ xyz \sim uvw\}$$

and

$$\mathcal{N}_2(uvw) := \mathcal{N}_1(uvw) \cup \{abc \in Q \ : \ abc \sim xyz \in \mathcal{N}_1(uvw)\}.$$

Now for $j > i_0$ we process all triangles in $Q(j)$ in a random order. Let uvw be the current triangle we look at. If $uvw \notin \mathcal{Y}$, and for one of the edges, j is the first time that the edges has co-degree one, we add uvw to $\mathcal{H}$ and add $\mathcal{N}_2(uvw)$ to $\mathcal{Y}$. Note that the sizes of $\mathcal{H}$ and $\mathcal{Y}$ can never decrease during the whole process. As we already saw in our claim above, if we delete a triangle xyz such that uvw is still intact afterwards, we can

affect triangles incident to at most two edges of uvw. Hence, if $j > i_0$ is the first round at the end of which uvw has some edge with co-degree one and uvw is intact, then at least one other edge of uvw has co-degree greater than one since we can only affect the co-degree of two of the edges of uvw. This implies that $B_{uvw} > 0$ when uvw is added to $\mathcal{H}$. The set $\mathcal{H}$ helps us later in the proof to establish a lower bound on the number of edges in the resulting graph.

Now we consider a time $i_1 > i_0$. We already saw in 4.19 that at time i_0, for every triangle $uvw \in Q$, all its co-degrees have size $n^{2\epsilon+o(1)} > 1$, by our assumption on the co-degrees. Now assume that $uvw \in Q(i_1)$ and $B_{uvw} = 0$. Then, by construction of $\mathcal{H}$, either $uvw \in \mathcal{H}$ or necessarily there is a time $i_0 < j \leq i_1$ at which we had $uvw \in \mathcal{N}_2(xyz)$ for some $xyz \in \mathcal{H}$ because otherwise there would have been a time when one of the co-degrees of uvw would have been one and then uvw would have been inserted in $\mathcal{H}$. Further, note that at time i_0 we had

$$|\mathcal{N}_2(xyz)| \leq n^{4\epsilon+o(1)}$$

by definition of $\mathcal{N}_2(xyz)$ as $B_{uvw} = n^{2\epsilon+o(1)}$ for all $uvw \in Q(i_0)$. Since all the co-degrees can only decrease during the process and with the above considerations we get by definition of the evolution of $\mathcal{H}$

$$|\mathcal{H}(i_1)| \geq |\{uvw \in Q(i_1) \ : \ B_{uvw} = 0\}| \cdot n^{-4\epsilon-o(1)}. \tag{4.20}$$

Now assume that we have i_1 such that $p_1 := p(i_1) = \frac{1}{\sqrt{n}\log n}$. At this step we have

$$e(G_{i_1}) = \frac{1}{2}(n^2 p_1 - n) = \frac{1}{2}\left(n^2 \frac{1}{\sqrt{n}\log n} - n\right) = n^{3/2}\log n^{-1}\left(\frac{1}{2} + o(1)\right).$$

Hence, if the process ends earlier we clearly have at least $n^{3/2-o(1)}$ edges in the final graph as this means we have even more edges than at time i_1 and we are done with the proof in this case.

Now assume otherwise. Note that by our assumptions on the co-degrees we are, for sufficiently large n and due to $\epsilon < 1/6$, in a scenario where we can apply Lemma 4.1.7,

i.e., $t(i_1) < \tau_c$. Hence, we get by this lemma

$$\begin{aligned} Q(i_1) &\leq \frac{1}{6}(n^3 p_1^3 + n^2 p_1) + n^{7/3 p_1^2} \\ &= \frac{1}{6} n^3 \frac{1}{n^{3/2}(\log n)^3} + \frac{1}{6} n^2 \frac{1}{\sqrt{n}\log n} + n^{7/3}\frac{1}{n(\log n)^2} \\ &= \frac{1}{6} n^{3/2}(\log n^{-3}) + \frac{1}{6} n^{3/2}(\log n)^{-1} + n^{4/3}(\log n)^{-2} \\ &= \left(\frac{1}{3} + o(1)\right) e(G_{i_1}). \end{aligned} \tag{4.21}$$

Now we are left with two cases for the rest of the proof. For the first case, assume $\#\{(u,v) \in E(G_{i_1}) \ : \ Y_{u,v} = 0\} \geq \delta e(G_{i_1})$ for some arbitrarily small but fixed $\delta > 0$. Obviously, if an edge is not contained in any triangle, then this edge will survive the triangle-removal process. Hence, by the assumption that least $\delta e(G_{i_1}) = n^{3/2-o(1)}$ edges will be contained in the resulting graph G_{τ_0} and we are done with the proof in this case. Now we look at the last case. In particular, assume there are at most $o(e(G_{i_1}))$ edges (u,v) with $Y_{u,v} = 0$. This results in

$$Q(i_1) \overset{4.5}{=} \frac{1}{3} \sum_{(u,v)\in E[G_{i_1}]} Y_{u,v} \geq \frac{1}{3}|\{(u,v) \in E(G_{i_1}) \ : \ Y_{u,v} \neq 0\}| \geq \left(\frac{1}{3} - o(1)\right) e(G_{i_1}).$$

However, by looking at 4.21 we can see that almost all of the triangles in $Q(i_1)$ are edge disjoint. This can be seen as follows. First, we order the set of triangles in $Q(i_1)$ in an arbitrary order and then we add their edges step by step to the edge set $\{(u,v) \in E[G_{i_1}] \ : \ Y_{u,v} = 0\}$. At some point, we arrive at the edge set of G_{i_1} by construction. Now assume there is some arbitrarily small but fixed $\delta > 0$ such that at least $\delta Q(i_1)$ triangles share any edges with the triangles preceding them in our ordering. Then the total number of edges would be at most

$$3Q(i_1) - \delta Q(i_1) + o(e(G_{i_1})).$$

However, this would imply

$$Q(i_1) \geq \frac{1}{3-\delta}(e(G_{i_1}) - o(e(G_{i_1})))$$

which is a contradiction to Equation 4.21. Hence, we can see that we can exclude $o(e(G_{i_1}))$ triangles from Q and be left with a set of edge disjoint triangles $\mathcal{A}$ with $|\mathcal{A}| = (1/3 - o(1))e(G_{i_1})$. Note that we can characterize this as $B_{uvw} = 0$ for every $uvw \in \mathcal{A}$. Note that re-adding the set of excluded triangles can increase the value of

B_{uvw} for at most $o(e(G_{i_1}))$ triangles since one added triangle can only affect at most three triangles in $\mathcal{A}$. This means that we get

$$|\{uvw \in Q(i_1) \ : \ B_{uvw} = 0\}| = \left(\frac{1}{3} - o(1)\right) e(G_{i_1}) = n^{3/2-o(1)}.$$

Together with Equation 4.20 this implies

$$|\mathcal{H}^*| \geq |\mathcal{H}(i_1)| \geq n^{3/2-o(1)} n^{-4\epsilon-o(1)} = n^{3/2-4\epsilon-o(1)}. \tag{4.22}$$

The lower bound in this case will now be derived by analyzing how many edges among the triangles in $\mathcal{H}^*$ survive the triangle-removal process. For this we define random variables $\mathcal{E}_{uvw}$ for each triangle $uvw \in \mathcal{H}^*$. For their definition, we first observe that the triangle-removal process as stated in this section is equivalent to the process where we look at time i_0 at all our triangles $Q(i_0)$ and then shuffle the triangles uniformly at random and iterate over them. In every step, we delete its edges in case the triangle is still intact. Let σ be the random permutation we just described. Let $j > i_0$ be the time at which uvw was added to $\mathcal{H}^*$. At the end of round j, by definition and without loss of generaliy, we have then $Y_{u,v} = 1$. Now let $xyz \in Q(j)$ be arbitrary with $xyz \in \mathcal{N}_1(uvw)$ at the end of round j. By construction and by definition of $\mathcal{H}$, remember that $B_{uvw} > 0$ so such a triangle xyz exists. Now let $\mathcal{E}_{uvw}$ be the event that our random permutation σ arranges the triangle xyz prior to all triangles in $\mathcal{N}_1(xyz)$ including the triangle uvw itself. As we already saw, we have $\mathcal{N}_1(xyz) \leq n^{2\epsilon+o(1)}$ by our assumption on the co-degrees. This means that we have

$$\mathbb{P}[\mathcal{E}_{uvw}] \geq n^{-2\epsilon-o(1)}. \tag{4.23}$$

Further, by definition of the event $\mathcal{E}_{uvw}$, the corresponding triangle xyz will be checked by the triangle-removal process before any other triangle $abc \sim xyz$. In particular, this means that at the time when xyz is examined by the process it will be still be intact and hence the process will delete its edges $\{(x,y),(x,z),(y,z)\}$ at that time. But this means that at the time when the triangle uvw is examined by the process, the triangle will not be deleted since one of the edges is missing. In particular, the edge (u,v) will survive the process since $Y_{u,v} = 1$, see Figure 4.3. Since the triangles in $\mathcal{H}^*$ are edge disjoint which is ensured by our auxiliary set $\mathcal{Y}$, the final graph G_{τ_0} contains at least

$$\sum_{uvw \in \mathcal{H}^*} \mathbf{1}_{\{\mathcal{E}_{uvw}\}}$$

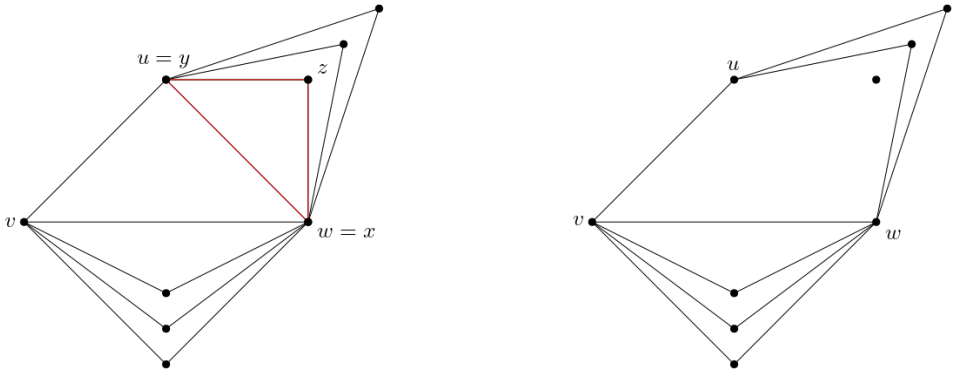


Figure 4.3.: The situation where we delete the triangle xyz while $Y_{u,v} = 1$.

edges. Hence, our goal is now to compute this term. Therefore, note that the event $\mathcal{E}_{uvw}$ is only dependent on our random permutation σ and its induced ordering of $\mathcal{N}_1(xyz) \subseteq \mathcal{N}_2(uvw)$ for an arbitrary triangle $xyz \in \mathcal{N}_1(uvw)$. By construction of $\mathcal{H}^*$ we exclude all triangles in $\mathcal{N}_2(uvw)$ to get added to $\mathcal{H}^*$ at a later point. Thus, we see that $\sum \mathbf{1}_{\{\mathcal{E}_{uvw}\}}$ stochastically dominates a binomial random variable with parameters $n = |\mathcal{H}^*|$ and $p = n^{-2\epsilon - o(1)}$, see Equation 4.23. Hence, we get by this dominance

$$\begin{aligned}\mathbb{E}\left[\sum_{uvw \in \mathcal{H}^*} \mathbf{1}_{\{\mathcal{E}_{uvw}\}}\right] &\geq |\mathcal{H}| \cdot n^{-2\epsilon - o(1)} \\ &\overset{4.22}{\geq} n^{3/2 - 4\epsilon - o(1)} \cdot n^{-2\epsilon - o(1)} \\ &= n^{3/2 - 6\epsilon - o(1)}.\end{aligned}$$

Hence, the final graph G_{τ_0} contains with high probability at least $n^{3/2-6\epsilon-o(1)}$ edges also in this case which concludes the proof. □

We will start our proof of the main result by showing that the assumption in Theorem 4.1.6 holds for the triangle-removal process.

Lemma 4.1.8. Let $0 < \epsilon < 1/6$ be arbitrary. Then with high probability all co-degrees satisfy $Y_{u,v} = (1 + o(1))np^2$ throughout $p \geq p_0 = n^{-1/2+\epsilon}$.

Proof. Let $0 < \epsilon < 1/6$ and $p \geq n^{-1/2+\epsilon}$. Further, let $M \in \mathbb{N}$ be sufficiently large such that $1/M \leq \epsilon$. As we saw in the proof for the upper bound, we can apply Theorem 4.1.4 and 4.1.5 in tandem as long as $p(t) \geq n^{-1/2+1/M}$ and since $n^{-1/2+1/M} \leq n^{-1/2+\epsilon}$ due to $1/M \leq \epsilon$ also as long as $p(t) \geq n^{-1/2+\epsilon}$. Hence, the estimate in Equation **??** holds as long as $p(t) \geq n^{-1/2+\epsilon}$. This means that we have for every co-degree $Y_{u,v}$ with high

probability as $n \to \infty$ and for $\zeta = n^{-1/2}p^{-1}\log n$ the estimate

$$\left|\frac{Y_{u,v}}{np^2} - 1\right| \leq 3^{3M-1}\zeta = 3^{3M-1}n^{-1/2}p^{-1}\log n$$
$$\leq 3^{3M-1}\frac{n^{-1/2}}{n^{-1/2}n^{\epsilon}}\log n = 3^{3M-1}\frac{\log n}{n^{\epsilon}} \overset{n\to\infty}{\longrightarrow} 0.$$

This means we get $Y_{u,v}/np^2 - 1 = o(1)$ and hence $Y_{u,v} = np^2(1 + o(1))$ for all co-degrees $Y_{u,v}$ with high probability as $n \to \infty$ throughout $p \geq n^{-1/2+\epsilon}$. □

Now the lower bound for Theorem 4.1.2 follows immediately.

Corollary 4.1.8.1. With high probability as $n \to \infty$ the number of edges of the graph G_{τ_0} is at least $n^{3/2+o(1)}$.

Proof. By Lemma 4.1.8 for any fixed $0 < \epsilon < 1/6$ all co-degrees satisfy $Y_{u,v} = (1 + o(1))np^2$ throughout $p \geq n^{-1/2+\epsilon}$. Hence, we can apply Theorem 4.1.6 and get that the number of edges of the graph G_{τ_0} is at least $n^{3/2-6\epsilon-o(1)}$ with high probability. Since ϵ can be chosen arbitrarily small, the claim follows. □

4.2. Simulation of the Triangle-removal Process

In this section, we want to present our algorithmic approach for the simulation of the triangle-removal process. This is interesting since the results for the triangle-removal process and for all the considered removal processes in general are not as precise as for the triangle-free process, see Section 3.1 and Section 4.1. Hence, it is particularly interesting to see how the results for the final number of edges evolve. We first present the algorithms developed and the motivation behind them. To this end, we also present algorithms which list all possible triangles in a given graph G. After this, we present our results for the simulations. This includes models for the results on the final number of vertices, performance results and some statistical properties.

Algorithms for the Triangle-removal Process

In this section, we describe our algorithms that we use for the triangle-removal process. Naively, we would follow the original description of the process as stated in Section 4.1. That is, we start with $G_0 = (V_0 = [n], E_0 = E(K_n))$ and in every step we keep a list of all triangles in the graph and choose one uniformly at random from it. However, this would mean that we hold the complete graph on n vertices in the memory and additionally every triangle that is in the complete graph. This means that at the first step we have

$O(n^3)$ triangles in the memory which is not efficient if n gets large. To overcome this problem, we make a few observations. First, note that choosing a triangle uniformly at random in every step is equivalent to the following. We look at a fixed step ξ and build a list of all triangles in G_ξ. Then we iterate in a uniform order over this list and check whether the triangle is still in the graph before removing it in every step. This is because every triangle has still the same probability to be chosen by the process at a later time.

Further, choosing one triangle uniformly at random is equivalent to choosing three vertices uniformly at random in this sense. If the vertices build a triangle, we delete it and otherwise we choose the next three vertices uniformly at random until we find a triangle. This follows from the fact that for the equivalence of the process, we only have to make sure that every triangle has the same probability to be chosen. Since every triangle has the same probability to be chosen in this way, this is guaranteed. Hence, we can define something similar to the rejection process for the triangle-free process, see Algorithm 4. We can start by choosing repeatedly three vertices $v \in [n]$ uniformly at random and check whether they build a triangle in the current graph. This is done as follows. First, we define the three edges that are built by the sampled vertices. Then we simply check whether the edges are still in the graph using a hash map or searching for it in the graph data structure. If this is the case, we can delete the triangle from the graph and otherwise we choose again three vertices until we find a triangle, see Algorithm 4. Note that this is of course only well defined if there still exists a triangle in the graph. For this purpose let

$$T(i) := \{(u,v,w) \in G_i \ : \ (u,v,w) \text{ is triangle in } G_i\}$$

be the set of triangles in the graph G_i, i.e., the set of triangles in the graph after the first $i-1$ triangle removals. Of course this approach has the advantage that we do not have to store the triangles that are contained in the graph. However, this approach is highly inefficient from a computational point of view. This is simply the case due to the small probability for a triangle to be chosen. Since we choose the vertices uniformly at random from $[n]$ in every step we get using our $G(n,p)$ intuition at step $i \in \mathbb{N}_0$

$$\mathbb{P}[\text{chosen vertices are a triangle in } G_i] \approx p^3, \tag{4.24}$$

where

$$p = \frac{\binom{n}{2} - 3i}{\binom{n}{2}} \approx 1 - \frac{6i}{n^2}$$

Algorithm 4: Rejection version of the triangle-removal process

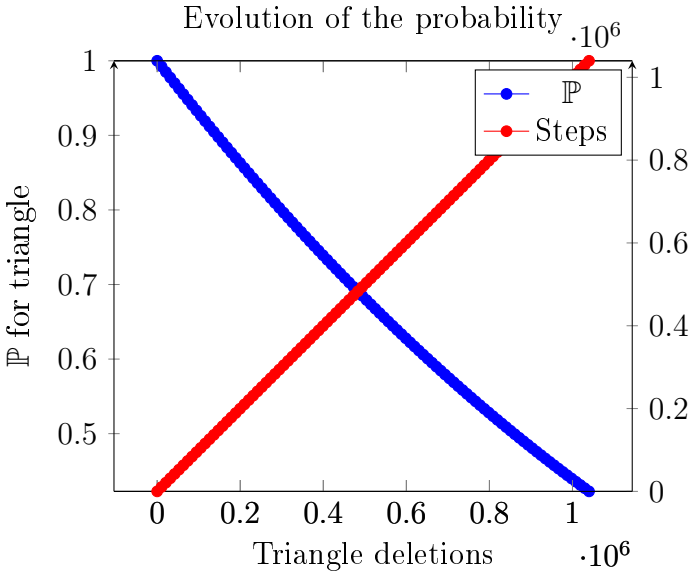
Data: Complete graph $G = (V_0 = [n], E_0 = E[K_n])$
Result: Resulting graph G_{τ_0} of the triangle-removal process

```
i ← 0;
while (T(i) > 0) do
    Vertex v1 ← randomVertex;
    Vertex v2 ← randomVertex;
    Vertex v3 ← randomVertex;
    Edge e1 ← {v1, v2};
    Edge e2 ← {v1, v3};
    Edge e3 ← {v2, v3};
    if v1, v2, v3 build triangle in G_i then
        delete {e1, e2, e3};
        i ← i + 1;
    else
        reject {e1, e2, e3};
```

is the edge density in the graph G_i. Hence, we get

$$\mathbb{P}[\text{chosen vertices are a triangle in } G_i] \approx \left(1 - \frac{6i}{n^2}\right)^3. \tag{4.25}$$

We can see the behaviour of 4.25 in Figure 13.



Hence, this approach is not suitable to simulate the triangle-removal process for a larger amount of vertices. Using our other observation, we could use a list of all triangles in K_n and iterate over it in a random order. However, as we already explained, this

approach is not suitable for large n. The idea is now again to combine both approaches to reduce the amount of triangles we have to store as much as possible without having too much computational effort. Hence, we start our simulation algorithm by fixing a certain threshold $\alpha \in \mathbb{N}$. Then in the beginning of the algorithm we start with our rejection approach, i.e., Algorithm 4. However, we only do this until we reject a candidate triangle α times in a row. We call the step at which this happens τ_α. The parameter α is consequently a tuning parameter which depends on the size of n. Afterwards, we look at the current graph G_{τ_α}. Our goal is now to make a list of all triangles in G_{τ_α} and afterwards iterate over them in a uniform order. In order to do so, we first have to think about an algorithm which enumerates all triangles in a given graph G. For this we use the `forward` algorithm presented by Schank et al. [Schank and Wagner, 2005] since the experimental data in [Schank and Wagner, 2005] suggests that it is the most suitable for our purpose since our graphs are still dense in comparison to real world instances where this kind of algorithm is typically used. Note that this algorithm is a refinement of an equivalent algorithm presented by Batagelj et al. [Batagelj and Mrvar, 2001]. The idea behind this algorithm is to start with the algorithm `edge-iterator`. Assume we have an arbitrary graph $G = (V, E)$. Then the algorithm simply iterates over all edges $e = (u, v)$. In every step, we look at the neighborhoods $N(u)$ and $N(v)$ and introduce a new triangle (u, v, w) if and only if $w \in N(u) \cap N(v)$. Note that if the neighborhoods are sorted arrays we can compute the intersection in $d(u) + d(v)$ time. Alternatively, we can use a hash map for this by assigning every vertex in $N(u)$ the value `true` and afterwards we iterate over the neighborhood $N(v)$ of v and add the corresponding vertex to the intersection if and only if the corresponding neighbor has the hash value `true`. In this way, we also get an amortized complexity of $d(u) + d(v)$ but we do not have to sort the neighborhoods. However, we get for this approach a running time of

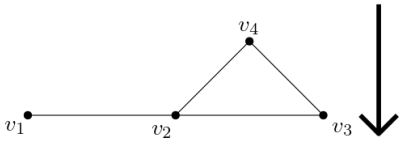
$$\sum_{(u,v)\in E} (d(u) + d(v)).$$

Now we can refine this approach for our `forward` algorithm by using dynamic data structures. Instead of using the actual neighborhoods we define a dynamic subset $A(v) \subseteq N(v)$ for all $v \in V$. In general, the idea behind this is to introduce an ordering of the vertices which does not interfere with finding all the triangles in the graph. This has the advantage that the running time only involves the in-degree of a vertex and hence the algorithm is faster than the `edge-iterator` algorithm. The idea is to still iterate over all edges $e = (u, v) \in E$ but now we look at $A(u) \cap A(v)$ instead of $N(u) \cap N(v)$ and let $A(w)$ grow during this process for every $w \in V$, see Algorithm 5. An example execution of this algorithm can be found in Figure 4.4. Note that for the

Algorithm 5: `forward` algorithm

Data: Arbitrary graph $G = (V = [n], E)$
Result: List T of all triangles in the graph.

```
1 T ← {};
2 for v ∈ V do
3 |  A(v) ← {};
4 for v ∈ [n] do
5 |  for w ∈ N(v) do
6 |  |  if v < w then
7 |  |  |  for u ∈ A(v) ∩ A(w) do
8 |  |  |  |  T ← T ∪ {(v, w, u)};
9 |  |  |  A(w) ← A(w) ∪ {v};
```



(v_1, v_2) : $A(v_2) = \{v_1\}, A(v_3) = \{\}, A(v_4) = \{\};\ T = \{\}$
(v_2, v_3) : $A(v_2) = \{v_1\}, A(v_3) = \{v_2\}, A(v_4) = \{\};\ T = \{\}$
(v_2, v_4) : $A(v_2) = \{v_1\}, A(v_3) = \{v_2\}, A(v_4) = \{v_2\};\ T = \{\}$
(v_3, v_4) : $A(v_2) = \{v_1\}, A(v_3) = \{v_2\}, A(v_4) = \{v_2, v_3\};\ T = \{(v_3, v_4, v_2)\}$

Figure 4.4.: On the left is the example graph. On the right from the top to the bottom we see the edges in the order the algorithm looks at them and the evolution of our dynamic data structure.

computation of the intersection of our dynamic data structure we can use hash maps as described above. Further, we see that the `forward` algorithm lists every triangle in the graph only once which is crucial for the random aspect in our simulation. For an analysis of the running time of this algorithm see [Schank and Wagner, 2005]. Now with the `forward` algorithm in mind, we can continue our simulation algorithm. Given the graph G_{τ_α} we apply the `forward` algorithm to obtain a list of all triangles in G_{τ_α} which we call T_{τ_α}. Afterwards, we shuffle T_{τ_α} uniformly at random and iterate over it to check whether the corresponding triangle is still in the graph, see Algorithm 6. Note that in line 23 of Algorithm 6 we do not have to check anymore whether the corresponding edges are a triangle, we only have to check whether all edges are still contained in the current graph. Now we want to describe how we build the remaining graph in line 22 of Algorithm 6. First, note that we use a hash map for the deleted edges instead of searching for the edges in the current graph from the beginning as the graph will be dense for the first steps. Hence, looking for an edge in this graph is inefficient as we have $d(v) = n$ for all vertices $v \in V$ in the beginning of the process. Consequently, we only store the value `true` for the deleted edges and if the value is `false` for an edge we already know that it is still in the graph. By complexity arguments for hash maps,

Algorithm 6: Simulation algorithm for the triangle-removal process

Data: Number of vertices n
Result: Resulting graph of the triangle-removal process G_{τ_0}.

```
1  T ← {};
2  i ← 0;
3  steps ← 0;
4  delEdges : \binom{[n]}{2} → {true, false};
5  while i ≤ α do
6      Vertex v_1 ← randomVertex;
7      Vertex v_2 ← randomVertex;
8      Vertex v_3 ← randomVertex;
9      Edge e_1 ← {v_1, v_2};
10     Edge e_2 ← {v_1, v_3};
11     Edge e_3 ← {v_2, v_3};
12     if G_steps[{e_1, e_2, e_3}] ≠ K_3 then
13         reject {e_1, e_2, e_3};
14         i ← i + 1;
15         if i > α then
16             break;
17     if {e_1, e_2, e_3} builds a triangle in G_steps then
18         delEdges(e_1) ← true;
19         delEdges(e_2) ← true;
20         delEdges(e_3) ← true;
21         i ← 0;
22 G_{τ_α} ← build remaining graph;
23 T_{τ_α} ← forward(G_{τ_α});
24 T_{τ_α} ← randomShuffle(T_{τ_α});
25 for Triangle t = {e_1, e_2, e_3} ∈ T_{τ_α} do
26     if delEdges(e_1) = delEdges(e_2) = delEdges(e_3) = false then
27         G_{τ_α} ← G_{τ_α} − {e_1, e_2, e_3};
28         delEdges(e_1) ← true;
29         delEdges(e_2) ← true;
30         delEdges(e_3) ← true;
```

this yields an effort of $O(1)$ for the amortized complexity to check whether an edge is in the graph or not. Also, it is possible to build the remaining graph, i.e., the graph after the first triangle deletions using this hash map. This is described in Algorithm 7. However, note that Algorithm 6 still has the disadvantage that the first phase of the algorithm, i.e., the rejection phase is inefficient even in the beginning of the triangle-removal process, see Figure 13. Hence, the question is whether there is another method to decrease the edge density enough such that the `forward` algorithm becomes more

Algorithm 7: Build remaining graph after the rejection phase in Algorithm 6

Data: Hash map `delEdges` : $E[K_n] \to \{\texttt{true}, \texttt{false}\}$
Result: Remaining graph G_{τ_α} after the rejection phase in Algorithm 6 with threshold α.

```
1 G_{τ_α} ← (V = [n], E = ∅);
2 for i ∈ [n] do
3     for j ∈ [n] do
4         if i < j ∧ delEdges((i, j)) = false then
5             G_{τ_α} ← G_{τ_α} ∪ {(i, j)};
```

efficient which is the case if we can decrease the number of edges before applying it. The idea is now to use a heuristic that is based on our insights from theory, see Section 4.1. Even though the results for the triangle-removal process are not as strong as for the triangle-free process (see Section 3.1), we see nonetheless that a lot of properties of the graph obtained by this random graph process closely resemble the properties of a $G(n, p)$ graph with the according edge density $p \in [0, 1]$. However, we can also observe that the higher p is, the higher the accuracy of this intuition. Hence, the idea is now to generate a graph $G(n, p)$ and then using the `forward` algorithm directly on it. We will see in the next section how this approach behaves for different p. To do so, we first have to look at the generation of a $G(n, p)$ graph. For this we use the algorithms presented by Funke et al. [Funke et al., 2018]. In this work the authors present distributed algorithms for various random graph models including the $G(n, p)$ model for $n \in \mathbb{N}$ and $p \in [0, 1]$. The advantage of this model is that it can use parallelization highly efficiently because the algorithm needs zero communication to generate a $G(n, p)$ with n and $p \in [0, 1]$ arbitrary. For the implementation we use the `KaGen` repository (`https://github.com/KarlsruheGraphGeneration/KaGen`) and use its library to integrate it directly to our algorithm. Hence, we do not have to print the graph and read it again for the algorithm. Consequently, we arrive at our first heuristic algorithm for the triangle-removal process where we have an input parameter p_α which is the edge density for our generated graph which we generate in the first step using `KaGen`. Afterwards, we run our `forward` algorithm on this graph and iterate over the shuffled list of triangles as in Algorithm 6, lines 25-30. Note that the goal for the heuristic is not just to generate large graphs, but also to have precise results which are consequently usable. Hence, we can not decrease our initial parameter p_α arbitrarily. We describe this algorithm in Algorithm 8. In the next section, we see that for ascending values of p_α, Algorithm 8 converges against the results of the original process obtained by Algorithm 6, suggesting that our intuition is indeed precise as long as p_α is not to small. This is also in line

Algorithm 8: Simulate the triangle-removal process starting from a $G(n,p)$ for $n \in \mathbb{N}$ and $p \in [0,1]$.

Data: Number of vertices $n \in \mathbb{N}$, initial edge density $p_\alpha \in [0,1]$.
Result: Triangle-free graph $G_{\triangle,p_\alpha}$ obtained by the triangle-removal process.

$G(n,p_\alpha) \leftarrow \texttt{KaGen}(n,p_\alpha)$;
$G_{\triangle,p_\alpha} \leftarrow G(n,p_\alpha)$;
$\texttt{delEdges} : \binom{[n]}{2} \rightarrow \{\texttt{true},\texttt{false}\}$;
$T_{p_\alpha} \leftarrow \texttt{forward}(G(n,p_\alpha))$;
$T_{p_\alpha} \leftarrow \texttt{randomShuffle}(T_{p_\alpha})$;
for *Triangle* $t = \{e_1,e_2,e_3\} \in T_{p_\alpha}$ **do**
  if $\texttt{delEdges}(e_1) = \texttt{delEdges}(e_2) = \texttt{delEdges}(e_3) = \texttt{false}$ **then**
    $G_{\triangle,p_\alpha} \leftarrow G_{\triangle,p_\alpha} - \{e_1,e_2,e_3\}$;
    $\texttt{delEdges}(e_1) \leftarrow \texttt{true}$;
    $\texttt{delEdges}(e_2) \leftarrow \texttt{true}$;
    $\texttt{delEdges}(e_3) \leftarrow \texttt{true}$;

with the theoretical findings where it turns out that our intuition gets worse the further the process has progressed. However, this yields the problem that even though we can choose p relatively small, we have to look at it rather as a small constant. Hence, even when we start the triangle-removal process from a $G(n,p)$, we still have a lot of triangles in the graph. To make the problem of memory consumption clear, recall Lemma 4.1.1 for the $G(n,p)$ model giving us

$$\mathbb{E}[\#\text{triangles in the graph}] = \binom{n}{3} \cdot p^3 \approx \frac{n^3}{6} \cdot p^3.$$

Hence, if we can not decrease p far enough we still have a high memory consumption. An additional step to resolve this could be to choose an edge according to its co-degree and afterwards, to choose one vertex in the intersection of the neighbourhoods uniformly at random. However, due to time limitation we were not able to further follow this approach.

4.2.1. Experimental Evaluation

In this section, we want to present the simulation results for the triangle-removal process. Thereby, our main goal is to further investigate the number of edges in the final graph for which we know

$$e(G^r_{\triangle,n}) = n^{3/2+o(1)}$$

Sample mean	10 938,69
Sample standard deviation	79,15
Relative sample standard deviation	0,72
Sample variance	6 265,06

Table 4.1.: Some statistical results for the triangle-removal process on 1 000 vertices with 200 repetitions.

with high probability, see Section 4.1. Hence, a natural question to ask is whether there is a logarithmic factor involved. Note that

$$(\log n)^c = n^{o(1)}$$

for every constant $c \in \mathbb{R}$. We examine this question using different models to fit the data in a similar way as for the triangle-free process. For this we use Algorithm 6. Further, we give some statistical properties for the simulation results of this algorithm. Additionally, we shortly look at the time and memory performance of the simulation. Afterwards, we show that the heuristic algorithm presented in Algorithm 8 converges against the results for the original process and show that the performance difference is significant with respect to the time for the simulation.

Statistical results. In order to assess the reliability of our results, we have to know what sample standard deviation we have to expect. For this, we repeat the simulation of the triangle-removal process on 1 000 vertices 200 times. The results for this can be seen in Table 4.2.1. What we can see is that similar to the triangle-free process we have a low sample standard deviation of less than one percent. This builds the basis for our further analysis of the evolution of $e(G^r_{\triangle,n})$. In particular, we can expect reliable results without too many repetions of the process.

Time and memory performance of Algorithm 6. Now we want to analyze how the number of rejections in a row inlfuence the performance of the algorithm. Note that this has no influence on the result quality of the simulation. We can see this in Table 4.2.1. What we can see in this table is that an increased number of rejections that we allow lead to significantly less memory consumption which is in general much higher than this was the case for the H-free processes. This is because we do not only have to maintain a list of open edges but we have to maintain a list of all triangles that are still in the graph. This is described in more detail in Section 4.2. However, the downside of an increase of α is that the time consumption increases significantly which is probably due to the low probability of a random triangle after sufficient triangle deletions. This

n	$\alpha = 500$		$\alpha = 2\,500$		$\alpha = 5\,000$	
	t[s]	m[KB]	t[s]	m[KB]	t[s]	m[KB]
5 000	1 137,63	1 820 892	6 154,02	346 344	12 147,7	246 576
6 000	1 787,67	1 880 612	8 095,95	461 160	17 562,9	337 780
7 000	2 208,13	3 516 668	12 084,1	529 284	23 835,5	481 824
8 000	2 345,37	6 738 948	12 534,9	856 088	32 501,0	551 024
9 000	4 222,61	6 824 220	22 131,1	1 043 516	38 431,1	850 564
10 000	5 172,7	13 383 128	26 599,8	1 547 524	50 777,5	1 010 252
11 000	6 815,29	13 504 708	30 982,0	1 663 988	60 885,9	1 275 648
12 000	6 858,8	26 215 688	32 385,8	2 579 344	-	-
13 000	8 619,41	26 409 668	44 511,6	2 713 688	-	-
14 000	9 374,36	51 731 964	45 236,7	4 462 612	-	-
15 000	11 700,1	51 890 864	57 232,3	4 640 084	-	-

Table 4.2.: Influence of the rejection parameter α on the memory consumption and the overall time for the simulation of the triangle-removal process on different numbers of vertices n.

also underlines the motivation for Algorithm 8 to safe the first triangle deletions by generating a $G(n, p)$ for the beginning of the process.

Algorithm 8. Now we want to show how our heuristic algorithm for the triangle-removal process peforms. For this, we run the algorithm with nine different random seeds and different input edge densities. The hope is that the results of the algorithm converge against the real results of Algorithm 6. This behaviour can be seen in Figure 4.5. What we can see is that the input parameter p has a significant inlfuence. If we choose this parameter to low we are far from the expected result for the simulation. However, we can see that an increase of the edge density yields a convergence against the results of Algorithm 6. This underlines that also for the triangle-removal process the intuition that the graphs should behave in the beginning like a $G(n, p)$ with the according edge density is correct. However, it also indicates that towards the later stages of the triangle-removal process a more sophisticated analyses is needed. Further, we can see that this also yields a significant improvement for the time performance. This can be seen in Table 4.2.1 where one can also see at which edge densities the results converge. We can see here that we still need a significant amount of main memory for the list of remaining triangles in the graph. However, we have a significant improvement for the time performance since we save a lot of work by starting from a $G(n, p)$. Further, this shows that the rejection process decreases the edge density not fast enough compared to the time it needs since we also use less memory than for Algorithm 6.

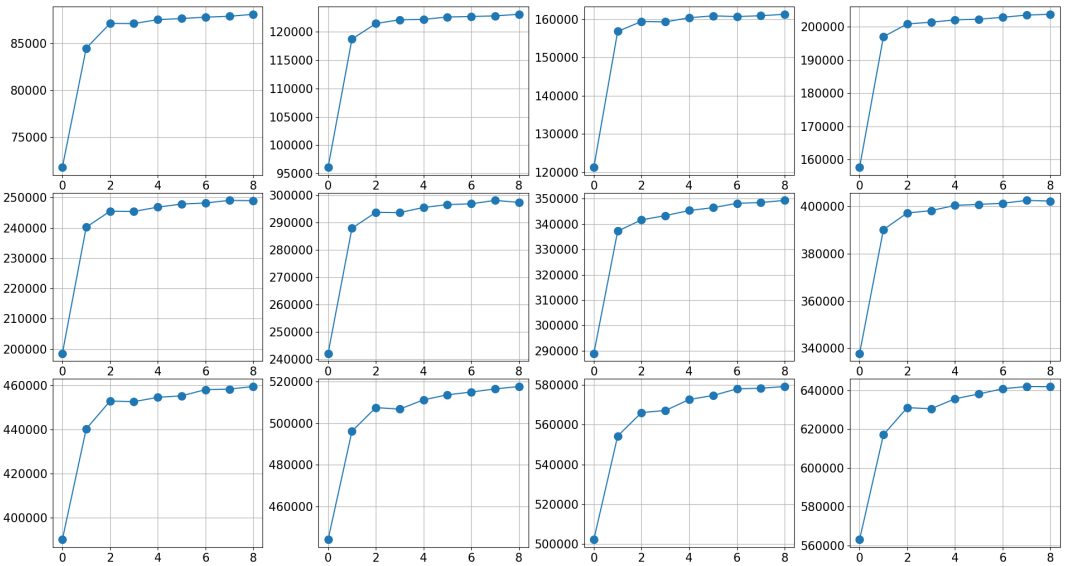


Figure 4.5.: The behaviour of Algorithm 8 for increasing edge densities. From the top left to the bottom right we show the behaviour for 4 000 vertices to 15 000 vertices in steps of 1 000. On the y-axis we see the results of the process.

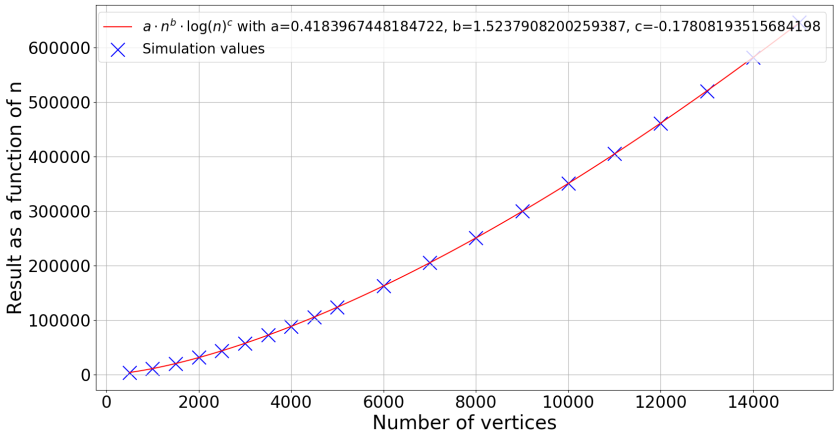


Figure 4.6.: Simulation results together with the predicted model for the model $f_1(n)$.

Models for the simulation data. Now we want to apply different models for the simulation data of Algorithm 6. Note that in order to have the most reliable results for this, we only use the data of the original process. In particular, we are interested to examine the error term for the final number of edges. For this we start with the first model

$$f_1(n) = a \cdot n^b \cdot (\log n)^c.$$

This model gives us the most parameters of freedom. The fit of this model can be seen in Figure 4.6. This gives us the values

n	ϵ_1	t[s]	m[KB]	Result	ϵ_2	t[s]	m[KB]	Result
5 000	0,067	2,59	169 204	122 818	0,072	3,10	171 088	123 101
6 000	0,055	2,91	176 592	160 911	0,06	3,60	178 868	161 286
7 000	0,053	4,29	286 132	203 504	0,058	5,28	290 424	203 746
8 000	0,055	7,04	303 192	249 080	0,06	8,85	506 040	248 981
9 000	0,055	10,32	521 732	298 169	0,06	12,96	540 580	297 438
10 000	0,05	11,58	534 732	348 514	0,055	14,27	556 368	349 359
11 000	0,052	16,92	573 608	402 479	0,057	21,80	976 252	402 195
12 000	0,05	20,24	985 292	458 409	0,055	27,14	996 084	459 573
13 000	0,045	20,93	602 916	516 516	0,05	27,07	1 009 400	517 650
14 000	0,045	25,95	1 019 052	578 263	0,05	33,55	1 034 332	579 127
15 000	0,045	32,58	1 043 780	642 034	0,05	43,41	1 859 524	641 958

Table 4.3.: Results for Algorithm 8 for the two largest edge densities.

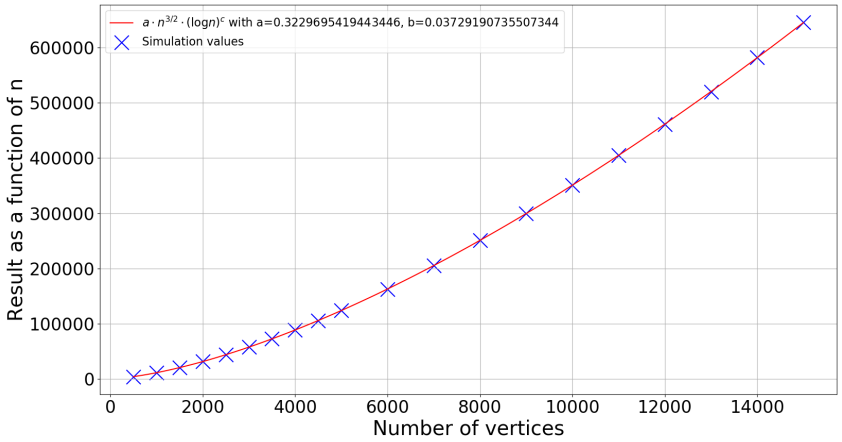


Figure 4.7.: Simulation results together with the predicted model for the model $f_2(n)$.

$$a = 0.42, \quad b = 1.52, c = -0.18$$

with a standard error of

$$S = 168.11$$

and we can see that this model is a good fit for the data. In particular, we are close to the main factor $n^{3/2}$. However, since we already know the main factor $n^{3/2}$ we apply a model

$$f_2(n) = a \cdot n^{3/2} (\log n)^c$$

where we fix this factor and focus on the error term. This can be seen in Figur 4.7. This

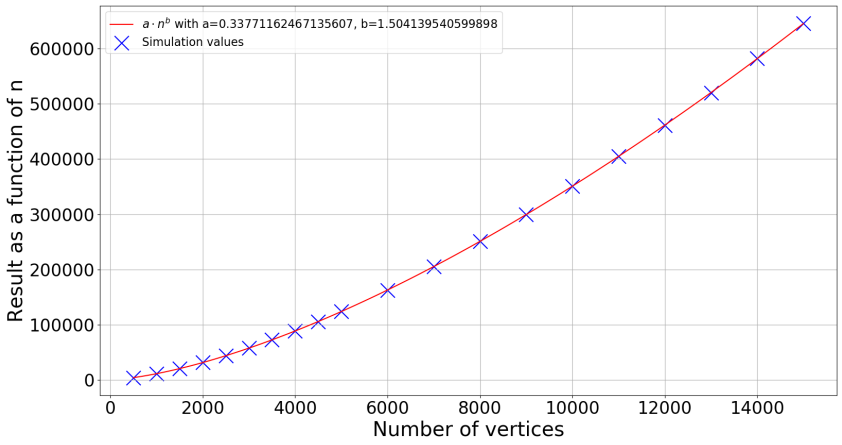


Figure 4.8.: Simulation results together with the predicted model for the model $f_3(n)$.

yields the results

$$a = 0.32, \quad c = 0.04$$

with a standard error

$$S = 181.40.$$

This also yields a good fit of the data and it indicates that there might is no logarithmic factor. In particular, this indicates that the triangle-free process and the triangle-removal process are not equivalent. This is also underlined by the model

$$f_3(n) = a \cdot n^b$$

which yields

$$a = 0.34, \quad b = 1.50$$

with an even better standard error of

$$S = 177.29$$

underlining the hypothesis that there is no logarithmic factor. In particular, we can assume that the predictions for $f_2(n)$ and $f_3(n)$ are more reliable due to a very high condition number of the covariance matrix for the model $f_1(n)$. This can indicate that the results are less reliable, see `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html`.

5. The C_4-free Process

In this chapter, we present the C_4-free process. This process is another special case of a random graph process. In particular, it is equivalently defined as the triangle-free process but the forbidden subgraph is in this case a C_4 instead of a triangle. However, to the best of our knowledge, the results that are known for this process are not as strong as for the triangle-free process. We give an introduction to the current state of the theory in the following section. Afterwards, we present our simulation algorithms for this process. Finally, we evaluate the simulation data and give suggestions for more precise results.

5.1. Theoretical Results

In this section, we want to present the latest theoretical results for the C_4-free process. Thereby, we will mainly follow the work of Picollelli [Picollelli, 2010] and Bohman et al. [Bohman and Keevash, 2009]. Our main interest is the final size of the C_4-free process which is the number of edges in the final graph. We start with our central definitions.

Definition 5.1.1. The random graph process $(G_i)_{i\in\mathbb{N}_0}$, where $G_0 = ([n], \emptyset)$ and for $i \in \mathbb{N}$ the graph G_i is obtained by selecting uniformly at random an edge from $E(K_n \setminus G_{i-1})$ which is neither already contained in G_{i-1} nor closes a C_4 in G_{i-1}, is called the C_4-free process on $n \in \mathbb{N}$ vertices.

Remark 5.1.1. As we already saw for the triangle-free process, this process becomes stationary at some point $\tau \in \mathbb{N}$ which is precisely the point if every edge in G_τ would close a C_4. We will denote this final graph as $G_{\square,n}$.

To the best of our knowledge, there is only little work which is only dedicated to the C_4-free process. However, since C_4 is strictly 2-balanced, there are still some results for the final number of edges for this process. Bollobás and Riordan [Bollobás and Riordan, 2000] examined the H-free process in this case and gave general lower bounds for the final number of edges. Additionally, they gave an upper bound which matched the lower bound within a logarithmic factor. In particular, they showed that $e(G_{\square,n}) = \Omega(n^{4/3})$ and

$e(G_{\square,n}) = O(n^{4/3}(\log n)^3)$ w.h.p. Further, Osthus and Taraz [Osthus and Taraz, 2001] gave an upper bound of $O(n^{4/3}(\log n))$ w.h.p. Bohman et al. [Bohman and Keevash, 2009] provided a new lower bound on the minimum degree of $G_{\square,n}$ which gives a new lower bound of $\Omega(n^{4/3}(\log n)^{1/3})$. Further, they conjectured that this is within a constant factor of the truth. Finally, Picollelli [Picollelli, 2010] was able to match this lower bound by showing that $e(G_{\square,n}) = O(n^{4/3}(\log n)^{1/3})$. Even though the results are not directly applicable to classical Ramsey theory as this is the case for $H = K_3$, the results of Bohman et al. [Bohman and Keevash, 2009] provided nonetheless new lower bounds for the cycle-complete Ramsey numbers $R(C_l, K_t)$ where $l \geq 4$ is fixed and t is sufficiently large. The main result we want to present in this section is the following.

Theorem 5.1.1 ([Picollelli, 2010], Corollary 1.2)**.** With high probability as $n \to \infty$ we get

$$e(G_{\square,n}) = \Theta(n^{4/3}(\log n)^{1/3}).$$

As we already saw for the triangle-free process, it turns out that following the intuition of a $G(n, m)$ is helpful for this process, too. Hence, we start by looking at this model for our basic parameters we want to follow for this process and introduce some basic definitions.

Definition 5.1.2. For every step $i \in \mathbb{N}_0$ of the C_4-free process an edge $e \in E(K_n \setminus G_{i-1})$ is called *open* if $G_{i-1} \cup \{e\}$ contains no C_4. The edge is called *closed* if $G_{i-1} \cup \{e\}$ contains a C_4.

As we already saw for the triangle-free process, the set of open edges in step $i \in \mathbb{N}$ of the process which we denote by $O(i)$ are precisely those edges which can be added to G_{i-1} without closing a C_4. The set of closed edges in step $i \in \mathbb{N}_0$ is denoted by $C(i)$ and we set $Q(i) := |O(i)|$. Further, for every $i \in \mathbb{N}_0$ in the C_4-free process we can partition the set of potential edges $E(K_n)$ into

$$E(K_n) = O(i) \cup C(i) \cup E(G_i)$$

and obviously this union is disjoint for every $i \in \mathbb{N}_0$. Obviously, the central parameter in this section is $Q(i)$ since the process ends precisely if $Q(i) = 0$. Hence, we start by looking at this parameter for the Erdös-Renyi model.

Lemma 5.1.2. Let $n \in \mathbb{N}$ and $0 < i < \binom{n}{2}$. Then for $G \sim G(n, p)$ we have

$$\mathbb{E}[\#\text{open edges in G}] = \binom{n}{2}(1-p^3)^{\binom{n-2}{2}\cdot 2} \approx \binom{n}{2} e^{-p^3 \cdot n^2},$$

where $p := \frac{i}{\binom{n}{2}}$ is the edge density of the graph $G \sim G(n,p)$.

Proof. We compute for every possible edge

$$\mathbb{P}[e = (u,v) \text{ is open}] = \mathbb{P}[A_e] = (1-p^3)^{\binom{n-2}{2}\cdot 2}, \tag{5.1}$$

where A_e is the event that there is no path from u to v of length 3. Obviously, an edge $e = (u,v)$ is open if and only if there is no path from u to v of length 3. If there is one, the edge is closed as inserting it would create a C_4. The second equality in 5.1 can be seen as follows. Any path (u,x,y,v) of length 3 has probability p^3 to occur in G and hence probability $(1-p^3)$ of not occurring in $G \sim G(n,p)$. For every pair $x \neq y$, where $\{x,y\} \cap \{u,v\} = \emptyset$, we get two possible paths, the path (u,x,y,v) and the path (u,y,x,v). Hence, there are $\binom{n-2}{2} \cdot 2$ possible paths of length 3 and we get together

$$\mathbb{P}[A_e] = (1-p^3)^{\binom{n-2}{2}\cdot 2}. \tag{5.2}$$

Using indicator variables for the event A_e for every edge $e \in E(K_n)$ yields then by linearity of expectation

$$\begin{aligned}
\mathbb{E}[\#\text{open edges in G}] &= \mathbb{E}[\sum_{e\in E(K_n)} \mathbf{1}_{\{e \text{ is open}\}}] \\
&= \sum_{e\in E(K_n)} \mathbb{E}[\mathbf{1}_{\{e \text{ is open}\}}] \\
&= \sum_{e\in E(K_n)} \mathbb{P}[\mathbf{1}_{\{e \text{ is open}\}}] \\
&= \sum_{e\in E(K_n)} \mathbb{P}[A_e] \\
&\overset{5.1}{=} \binom{n}{2}(1-p^3)^{\binom{n-2}{2}\cdot 2}
\end{aligned}$$

and hence the first statement of the lemma. Using the approximation $\binom{n}{2} \approx \frac{n^2}{2}$ we get

$$\binom{n}{2}(1-p^3)^{\binom{n-2}{2}\cdot 2} \approx \binom{n}{2} e^{-p^3 \frac{(n-2)^2}{2}\cdot 2} \approx \binom{n}{2} e^{-p^3 n^2}$$

and hence also the second statement of the lemma. □

With this lemma in mind, we would expect the number of open edges to decay at a rate of $e^{-p^3n^2}$. This will be important for the upper bound as well as for the lower bound in Theorem 5.1.1.

Upper Bound

For the upper bound, we follow the work of Picollelli [Picollelli, 2010]. The main goal of this section is to show the following theorem.

Theorem 5.1.3 ([Picollelli, 2010], Theorem 1.1)**.** There exists $\kappa > 0$ such that we have $\Delta(G_{\square,n}) \leq \kappa(n \log n)^{1/3}$ with high probability.

The overall idea for this theorem is based on the following observation. Assume we have a fixed vertex $v \in [n]$ and a certain step $i \leq \tau$. Then if there is $x \neq y$ with $\{x, y\} \cap \{v\} = \emptyset$ and $\{x, y\} \in N_{G_{\square,n}}(v)$ but $(v, x), (v, y) \notin E(G_i)$, then x and y do not have a common neighbor in G_i. With this idea, the goal is to show a bound $\Delta(G_{\square,n}) \leq \Delta(G_i) + k = O((n \log n)^{1/3})$ by showing that every set K consisting of k vertices contains two vertices $x \neq y$ such that $N(x) \cap N(y) \neq \emptyset$. This will suffice to prove Theorem 5.1.3. To achieve this, the authors track a certain type of random variables which we define below using the differential equations method. We introduce now the necessary random variables and show how they look like in the Erdös-Renyi model before we show how they can be used to prove Theorem 5.1.3.

Definition 5.1.3. We say that a set K consisting of k vertices is *covered* in G_i if there exists $x \neq y \in K$ such that $N(x) \cap N(y) \neq \emptyset$. Otherwise, we say that K is *uncovered*.

Since we would expect, if a covered k-set K has a reasonable size, that the corresponding common neighbour is likely to be in $V(G_i) \setminus K$ the idea is to look at the following.

Definition 5.1.4. Let $K \subseteq V(G_i)$ with $|K| = k$ and $0 \leq i \leq \mu(\log n)^{1/3} n^{4/3}$ for a constant $\mu \ll 1/4$. Then we define

$$X_K(i) := \left\{ ((u, v), w) \in \binom{K}{2} \times V \setminus K \ : \ (u, w), (w, v) \in O(i) \right\}$$

and

$$Y_K(i) := \Bigg\{ ((u, v), w) \in \binom{K}{2} \times V \setminus K \ : \\ |\{(u, w), (v, w) \cap O(i)\}| = |\{(u, w), (v, w) \cap E(G_i)\}| = 1 \Bigg\}.$$

We call triples in $X_K(i)$ open with respect to the set K. Triples in $Y_K(i)$ are called partial. Note for the definition of $Y_K(i)$ that $E(G_i) \cap O(i) = \emptyset$. There are some observations that can be made for the random variables $Y_K(i)$. Assume $((u, v), w) \in Y_K(i)$ and without loss of generality $(u, w) \in O(i)$ and hence by definition $(v, w) \in E(G_i)$. If then, in step $i + 1$, we choose the edge (u, w) then u and v share w as a

common neighbour and hence the set K is covered for every future step $j \geq i+1$. Hence, intuitively we can think of $Y_K(i)$ as the set which contains those edges that make K covered when chosen in step $i+1$. Further, $X_K(i)$ is then a sort of predecessor of $Y_K(i)$ because if $((u,v),w) \in X_K(i)$ and we choose, without loss of generality, (u,w) in step $i+1$, then $((u,v),w) \in Y_K(i+1)$. We can also observe that if a k-set K is uncovered, then obviously we have $|N(v) \cap K| \leq 1$ for all vertices $v \in [n]$. Following our intuition with the Erdös-Renyi model, we would expect for those random variables the following.

Lemma 5.1.4. Let $n \in \mathbb{N}$ and $0 \leq i \leq \binom{n}{2}$. Further, let $G \sim G(n,p)$, for $p := \frac{i}{\binom{n}{2}}$ and let $K \subseteq V(G)$ be a k-set. Then we have

$$\begin{aligned} \mathbb{E}[|X_K(i)|] &= \binom{k}{2}(n-k)(1-p^3)^{\binom{n-2}{2}4} \\ &\approx \frac{k^2}{2} n e^{-2p^3n^2} \end{aligned} \tag{5.3}$$

and

$$\begin{aligned} \mathbb{E}[|Y_K(i)|] &= 2\binom{k}{2}(n-k)(1-p^3)^{\binom{n-2}{2}2}p \\ &\approx k^2 n p e^{-p^3n^2}. \end{aligned} \tag{5.4}$$

Proof. We first compute $\mathbb{E}[|X_K(i)|]$. The number of pairs in K is given by $\binom{k}{2}$ since $|K| = k$. Hence, we have

$$|\binom{K}{2} \times (V \setminus K)| = \binom{k}{2}(n-k).$$

For each element $((u,v),w) \in X_K(i)$ the edges $(u,w),(v,w)$ must be open and as we saw in the proof of Lemma 5.1.2 we have for every edge $e \in E(K_n)$

$$\mathbb{P}[e \text{ is open}] = (1-p^3)^{\binom{n-2}{2}2} \tag{5.5}$$

and hence by definition of $X_K(i)$ we get for every element $((u,v),w) \in \binom{K}{2} \times V \setminus K$

$$\mathbb{P}[((u,v),w) \in X_K(i)] = \mathbb{P}[e \text{ is open}]^2 = (1-p^3)^{\binom{n-2}{2}4}. \tag{5.6}$$

Using indicator random variables for every possible element $((u,v),w) \in \binom{K}{2} \times V \setminus K$

yields

$$\begin{aligned}\mathbb{E}[X_K(i)] &= \mathbb{E}[\sum_{((u,v),w)\in\binom{K}{2}\times V\setminus K} \mathbf{1}_{\{((u,v),w)\in X_K(i)\}}] \\ &= \sum_{((u,v),w)\in\binom{K}{2}\times V\setminus K} \mathbb{E}[\mathbf{1}_{\{((u,v),w)\in X_K(i)\}}] \\ &= \sum_{((u,v),w)\in\binom{K}{2}\times V\setminus K} \mathbb{P}[((u,v),w)\in X_K(i)] \\ &\overset{5.6}{=} \binom{k}{2}(n-k)(1-p^3)^{\binom{n-2}{2}4}.\end{aligned}$$

Using our usual approximations, we get

$$\binom{k}{2}(n-k)(1-p^3)^{\binom{n-2}{2}4} \approx \frac{k^2}{2}ne^{-2p^3n^2}$$

since n dominates k and hence the first part of the lemma follows.
Now we compute $\mathbb{E}[|Y_K(i)|]$. We get with an analogous argumentation as above and by definition of $Y_K(i)$

$$\begin{aligned}\mathbb{E}[|Y_K(i)|] &= 2\binom{k}{2}(n-k)\mathbb{P}[e \text{ is open}]\mathbb{P}[e\in G] \\ &\overset{5.6}{=} 2\binom{k}{2}(n-k)(1-p^3)^{\binom{n-2}{2}2}p,\end{aligned}$$

where the additional factor of two comes from the fact that every element $((u,v),w)\in\binom{K}{2}\times V\setminus K$ represents two possible elements in $Y_K(i)$. Using the same approximations as above again, we get

$$2\binom{k}{2}(n-k)(1-p^3)^{\binom{n-2}{2}2}p \approx k^2ne^{-p^3n^2}p.$$

□

Now let $t := i/n^{4/3}$. Then using $p\approx 2i/n^2$ yields $e^{-p^3n^2}\approx e^{-8t^3}$. The main technical lemma in the work of [Picollelli, 2010] shows that Lemma 5.1.4 is up to an error term close to the truth for the C_4-free process.

Lemma 5.1.5 ([Picollelli, 2010], Lemma 2.3)**.** Let W and ϵ be a sufficiently constants such that $0\ll\epsilon\ll 1/W\ll 1/4$. With high probability, for all i, $0\le i\le m$, and K a k-set, if K is uncovered in G_i then

$$|X_K(i)| = \left(1\pm\frac{e^{W(t^3+t)}-1}{n^{3\epsilon}}\right)\left(\frac{e^{-16t^3}}{2}\pm\frac{1}{n^3\epsilon}\right)\frac{k^2}{n}$$

and

$$|Y_K(i)| = \left(1 \pm \frac{e^{W(t^3+t)} - 1}{n^{3\epsilon}}\right)\left(2te^{-8t^3} \pm \frac{1}{n^{3\epsilon}}\right) k^2 np$$

The authors proof this lemma using the differential equation technique. For details see [Picollelli, 2010], Section 3 and 4. For the proof of Theorem 5.1.3 we have to introduce some notation first. We fix constants μ, ϵ, W such that

$$0 < \mu \ll \epsilon \ll \frac{1}{V} \ll \frac{1}{4}.$$

Further, we define $p = p^* := n^{-2/3}$, $m = \mu(\log n^{1/3})n^{4/3}$ and $t_{\max} = \mu(\log n)^{1/3}$. Further, we want to choose our constants μ and ϵ sufficiently small such that

$$e^{W(t^3+t)} - 1 \leq e^{W(t_{\max}^3 + t_{\max})} - 1 = n^{W\mu^3 + \frac{W}{3}\mu} - 1$$

is at most n^ϵ and

$$e^{8t^3} \leq e^{8t_{\max}^3} = n^{8\mu^3}$$

is at most n^ϵ for $0 \leq t \leq t_{\max}$. For this, we just need to choose μ sufficiently small. Further, we want to choose ϵ sufficiently small such that

$$n^{1/8-\epsilon} \gg n^{3\epsilon}$$

and again μ sufficiently small such that

$$\frac{e^{W(t^3+t)} - 1}{n^\epsilon} = o(1)$$

uniformly in $0 \leq t \leq t_{\max}$ which can be achieved with the computations from above by choosing μ sufficiently small compared to ϵ. The only missing ingredient are now two other results from [Bohman and Keevash, 2009] which we summarize here.

Theorem 5.1.6 ([Bohman and Keevash, 2009])**.** With high probability for $0 \leq i \leq m$ we have

$$Q(i) = \left(1 \pm \frac{e^{W(t^3+t)} - 1}{n^{1/8-\epsilon}}\right)\left(e^{-8t^3} \pm \frac{1}{n^{1/8-\epsilon}}\right)\frac{n^2}{2} \tag{5.7}$$

and

$$d_{G_i}(v) = \left(1 \pm \frac{e^{W(t^3+t)} - 1}{n^{1/8-\epsilon}}\right)\left(2t \pm \frac{1}{n^{1/8-\epsilon}}\right) np. \tag{5.8}$$

The proof of Thorem 5.1.6 follows immediately by applying Theorem 2.3.2 to the trackable extension variables $2|O(i)|$ and $d_{G_i}(v)$. We have seen the the applicability of

this theorem to these variables in Lemma 2.3.3 and Lemma 2.3.4. Note that Theorem 5.1.6 implies by Equation 5.8

$$\Delta(G_i) \leq 4t_{max}np$$

with high probability.

Proof of Theorem 5.1.3: Let $\beta > 0$ be a fixed constant with

$$\beta > \frac{4}{\mu^2} \tag{5.9}$$

and let $k := \beta(n \log n)^{1/3}$. We show now that with high probability every k-set K is covered at step m. For this, let K be any uncovered k-set at step $i \leq m$. Note that e_{i+1} is chosen uniformly at random from $O(i)$. Further, each partial triple in $Y_K(i)$ contains an unique open edge. As we already explained these open edges make K covered when chosen by the process. Hence, we get

$$\mathbb{P}[K \text{ is uncovered in step i+1}] \leq 1 - \frac{|Y_K(i)|}{Q(i)}. \tag{5.10}$$

Now we look at the process for the steps $m/2 \leq i \leq m$. We want to compute the probability that some k-set K remains uncovered during all these steps. Since $m = \mu(\log n)^{1/3}n^{4/3}$ and μ is fixed we know that $m/2 \geq n^{4/3}$ for n sufficiently large. This implies that during the steps $m/2 \leq i \leq m$ we have $t \geq 1$ for n sufficiently large. Further, remember that by the assumptions on the constants for n sufficiently large and $t_{\max}/2 \leq t \leq t_{\max}$ we have

$$n^{1/8-\epsilon} \geq n^{3\epsilon}\ ,\ \frac{e^{W(t^3+t)}-1}{n^{3\epsilon}} \leq \frac{1}{3} \text{ and } \frac{1}{n^{3\epsilon}} \leq \frac{e^{-8t^3}}{2} \leq \frac{te^{-8t^3}}{2}.$$

With this in mind, we can compute with Lemma 5.1.5 and Theorem 5.1.6

$$\begin{aligned}\frac{|Y_K(i)|}{Q(i)} &\underset{5.1.6}{\overset{5.1.5}{=}} \frac{\left(1 \pm \frac{e^{W(t^3+t)}-1}{n^{3\epsilon}}\right)\left(2te^{-8t^3} \pm \frac{1}{n^{3\epsilon}}\right)k^2np}{\left(1 \pm \frac{e^{W(t^3+t)}-1}{n^{1/8-\epsilon}}\right)\left(e^{-8t^3} \pm \frac{1}{n^{1/8-\epsilon}}\right)\frac{n^2}{2}} \\ &\geq \frac{\left(1 - \frac{e^{W(t^3+t)}-1}{n^{3\epsilon}}\right)\left(2te^{-8t^3} - \frac{1}{n^{3\epsilon}}\right)k^2np}{\left(1 + \frac{e^{W(t^3+t)}-1}{n^{1/8-\epsilon}}\right)\left(e^{-8t^3} + \frac{1}{n^{1/8-\epsilon}}\right)\frac{n^2}{2}} = (*).\end{aligned} \tag{5.11}$$

This yields using $(e^{W(t^3+t)}-1)/n^{3\epsilon} \leq 1/3$ for n sufficiently large due to $(e^{W(t^3+t)}-1) \leq n^{\epsilon}$

and $1/n^{3\epsilon} \leq (te^{-8t^3})/2$ that

$$(*) = \frac{\frac{2}{3} \cdot \frac{3}{2} te^{-8t^3} \cdot k^2 np}{\left(1 + \frac{e^{W(t^3+t)}-1}{n^{1/8-\epsilon}}\right)\left(e^{-8t^3} + \frac{1}{n^{1/8-\epsilon}}\right)\frac{n^2}{2}}$$

and this implies with $(e^{W(t^3+t)} - 1)/n^{1/8-\epsilon} \leq (e^{W(t^3+t)} - 1)/n^{3\epsilon} \leq 1/3$ and $1/n^{1/8-\epsilon} \leq 1/n^{3\epsilon} \leq e^{-8t^3}/2$ that

$$(*) = \frac{\frac{2}{3} \cdot \frac{3}{2} te^{-8t^3} \cdot k^2 np}{\frac{4}{3} \cdot \frac{3}{2} e^{-8t^3} \cdot \frac{n^2}{2}} = \frac{tk^2 p}{n} \geq \frac{t_{\max} k^2 p}{2n}.$$

Hence, the probability that some k-set K remains uncovered during $m/2 \leq i \leq m$ is by Equations 5.10 and 5.11 bounded by

$$\begin{aligned}
\binom{n}{k} \cdot \left(1 - \frac{|Y_K(i)|}{Q(i)}\right)^{m/2} &\leq \binom{n}{k} \left(1 - \frac{t_{\max} k^2 p}{2n}\right)^{m/2} \\
&\leq n^k \exp\left(-\frac{t_{\max} k^2 pm}{4n}\right) \\
&= n^k \exp\left(-\frac{\mu (\log n)^{1/3} \cdot \beta^2 (n \log n)^{2/3} \cdot n^{-2/3} \cdot \mu n^{4/3} (\log n)^{1/3}}{4n}\right) \\
&= n^k \exp\left(-\frac{\mu^2 \beta^2 n^{1/3} (\log n)^{4/3}}{4}\right) = (**).
\end{aligned}$$

Further, we have

$$n^k = \exp(k \log n) = \exp(\beta n^{1/3} (\log n)^{4/3}).$$

Hence, we get

$$\begin{aligned}
(**) &= \exp\left(\frac{4\beta n^{1/3} (\log n)^{4/3} - \mu^2 \beta^2 n^{1/3} (\log n)^{4/3}}{4}\right) \\
&= \exp\left(n^{1/3} (\log n)^{4/3} \cdot \frac{(4\beta - \mu^2 \beta^2)}{4}\right).
\end{aligned}$$

This term is $o(1)$ if and only if

$$\frac{(4\beta - \mu^2 \beta^2)}{4} < 0 \iff \frac{4}{\mu^2} < \beta$$

which is exactly how we chose β, see 5.9. Hence, we see that with high probability there is no k-set K that remains uncovered at step m. Now assume $\Delta(G_{\square,n}) \geq \Delta(G_m)+k+1$ and every k-set K is covered in G_m. Then we find a vertex $v \in [n]$ and a k-set $K \subseteq [n] \setminus \{v\}$ such that $(v, x) \in E(G_{\square,n})$ for all $x \in K$ but in G_m we have $(v, x) \notin G_m$ for all $x \in K$.

But since K is covered, there is a vertex $w \in [n] \setminus \{v\}$ and vertices $x, y \in K$ such that $w \in N(x) \cap N(y)$. But this is a contradiction because then $G_{\square,n}[\{v, x, y, w\}]$ would contain a C_4. Consequently, we get

$$\Delta(G_{\square,n}) \leq \Delta(G_m) + k \leq 4\mu(n \log n)^{1/3} + \beta(n \log n)^{1/3} = \kappa(n \log n)^{1/3},$$

where $\kappa := (4\mu + \beta)$. □

Now the upper bound in Theorem 5.1.1 follows immediately with high probability by the straightforward bound

$$e(G_{\square,n}) = \frac{1}{2} \sum_{v \in V(G_{\square,n})} d_{G_{\square,n}}(v) \leq n \cdot \Delta(G_{\square,n}) \leq \kappa n^{4/3} (\log n)^{1/3}.$$

If we apply now Theorem 2.3.1 for the case $H = C_4$, we get the existence of a constant $C > 0$ such that

$$e(G_{\square,n}) \geq C n^{\frac{4}{3}} (\log n)^{\frac{1}{3}}. \tag{5.12}$$

Hence, we also proved Theorem 5.1.1.

5.2. Simulation of the C_4-free Process

In this section, we describe our approach for the algorithmic simulation of the C_4-free process. For this, we use similar techniques as in Chapter 3. However, the results for the C_4-free process are not as strong as for the triangle-free process from a theory point of view. In particular, we have seen in Section 5.1 that we know for the number of edges in the final graph $G_{\square,n}$ of the process

$$e(G_{\square,n}) = \Theta(n^{4/3} (\log n)^{1/3}) \tag{5.13}$$

with high probability but to the best of our knowledge there is no specification of a constant as in Section 3.1. However, we can use a similar intuition for the constant as for the triangle-free process and see whether our simulations can confirm that. We start by giving the necessary algorithms for the C_4-free process simulation and the intuition for the constant we want to investigate. Afterwards, we evaluate our simulations regarding various parameters.

5.2.1. Intuition for the constant

For the intuition of the constant we use again the $G(n,p)$ model. For this, we remember that we computed in this model the expected number of open edges. In particular, using $t = m/n^{4/3}$ and $p \approx 2m/n^2$, where we want to specify m, we have shown in Lemma 5.1.2 that the expected number of open edges is roughly

$$\binom{n}{2} e^{-8t^3}.$$

Now as we already discribed in Section 2.3, our intuition is particularly precise if we follow the process up to an edge density where the expected number of copies of a C_4 is equal to the amount of open edges and we have seen that this is the case for an edge density of roughly $n^{-2/3}$. This means that we follow the process up to the moment when

$$e^{-8t^3}\binom{n}{2} = n^{4/3}.$$

Solving this equation for m yields by simple calculations and estimates

$$m = \frac{1}{\sqrt[3]{12}} n^{4/3} (\log n)^{1/3}.$$

Hence, we give the following conjecture.

Conjecture 5.2.1. With high probability we have

$$e(G_{\square,n}) = \left(\frac{1}{\sqrt[3]{12}} + o(1)\right) n^{4/3} (\log n)^{1/3}$$

for the C_4-free process.

5.2.2. Simulation algorithms

Now we want to describe our algorithms we use for simulating the C_4-free process. Thereby, we follow a similar approach as for the triangle-free process in Section 3.2. Hence, we note again that the C_4-free process as stated in Section 5.1 is equivalent to the process where we choose in every step one edge $e \in E(K_n)$ uniformly at random and reject it if it is already contained in the graph or if it closes a C_4. In order to make this algorithm work we have to specify what it means for an edge $e = (u,v)$ to close a C_4. The idea is to do a modified sort of breadth-first search (*BFS*). This is based on the observation that an edge (u,v) closes a C_4 if and only if there is a path of length 3 from u to v. A convinient way to compute this is a *BFS* since we are not dealing with edge

Algorithm 9: Check whether an edge closes a C_4 in a graph G

Data: Edge $e = (u, v) \notin E(G)$
Result: `true` if e closes a C_4, `false` if not

```
1 cycleFound ← false;
2 for x ∈ N(u) do
3     for y ∈ N(x) do
4         if y ≠ u and y ≠ v then
5             Edge f ← (y, v);
6             if f ∈ E(G) then
7                 cycleFound ← true;
8                 return cycleFound;
9 return cycleFound;
```

weights. For this, we define levels of the *BFS* where level 0 contains only the source vertex itself and level $i \in \mathbb{N}$ contains every vertex which has a shortest path of length i to the source vertex. For example level 1 corresponds precisely to the neighbourhood of the source vertex. Now the algorithm to check whether an edge $e = (u, v)$ closes a C_4 does the following. First, the algorithm builds layer 1 starting from u as usual. Now when building layer 2, it checks for every vertex w whether the edge (w, v) is in the graph using a hash map for the edges for example. If this is the case, the algorithm exits the *BFS* since we found a path of length 3 from u to v which means that the edge (u, v) would close a C_4 when added to the graph. See Algorithm 9 for a detailed description. If we use now Algorithm 9, we can already introduce our rejcetion approach for the C_4-free process. Thereby, we just adapt Algorithm 1 from Section 3.2 and replace the condition for an edge to be added to the graph. For this, let $O(i)$ denote the set of open edges and $C(i)$ the set of closed edges, i.e., the set of edges which would close a C_4 in the graph G_{i-1}, after the first $i-1$ edge insertions during the C_4-free process. With this notation, we can define Algorithm 10. Note that Algorithm 9 is applied when checking the condition $e \in C(i)$ in line 4 of Algorithm 10 and the condition $e \in E(G_i)$ can be checked using a hash map for the edges. Alternatively, this could also be checked by searching for one of the vertices in the according neighbourhood. However, we see with the same arguments as for the triangle-free process that this approach is computationally not scalable. Using our intuition with the $G(n, p)$ model, we already saw in Section 5.1 in Lemma 5.1.2

$$|O(i)| \approx \binom{n}{2} e^{-p^3 n^2},$$

Algorithm 10: Rejection version of the C_4-free process

Data: Empty graph $G = (V_0 = [n], E_0 = \emptyset)$
Result: Resulting graph $G_{\square,n}$ of the C_4-free process

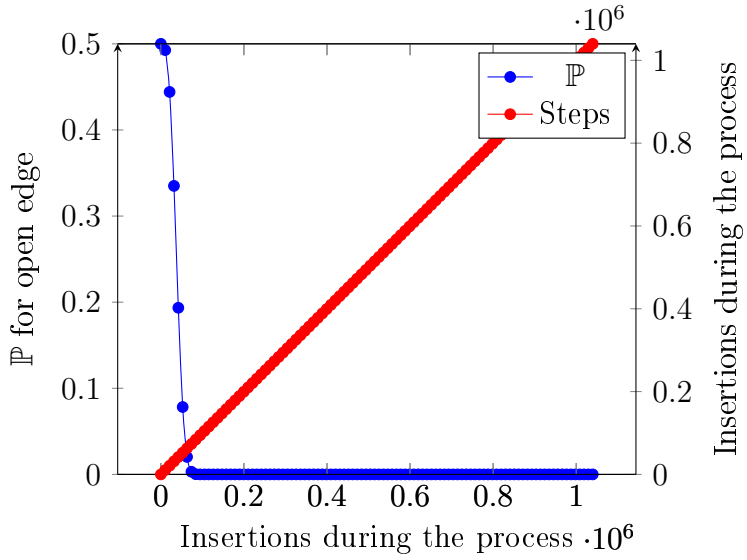
```
1 i ← 0;
2 while O(i) > 0 do
3     Edge e ← randomEdge;
4     if e ∈ E(G_i) || e ∈ C(i) then
5         reject e;
6     else
7         G_i ← G_i ∪ {e};
8         i ← i + 1;
```

where $p \approx 2i/n^2$. Consequently, we get

$$\mathbb{P}[\text{open edge gets chosen}] = \frac{|O(i)|}{n^2} \approx \frac{1}{2}e^{-p^3n^2} \approx \frac{1}{2}e^{-\frac{8i^3}{n^4}}.$$

We can see how this probability evolves in Figure 8.

Evolution of the Probability for an open edge during the process on 5000 vertices



Hence, we can see that the rejection approach is towards the end of the process not suitable. To resolve this, we present an analogous approach as for the triangle-free process. First, we note again that we can build a list of open edges at any step $i \in \mathbb{N}_0$ using Algorithm 9. For this, we first build a list of all edges that are not contained in the current graph G_i. To do so, we can use a edge hash map and iterate over every possible edge and check its value. Afterwards, we iterate over this list and add it if Algorithm 9 returns `false`. Now we can use an equivalent algorithm to Algorithm 2

Algorithm 11: List version of the C_4-free process starting at step $i \in \mathbb{N}_0$

Data: Current graph $G = (V_i = [n], E_i)$, $O(i)$
Result: Resulting graph $G_{\square,n}$ of the C_4-free process

```
1 Õ(i) ← randomShuffle(O(i));
2 for e = (u, v) ∈ Õ(i) do
3     if e closes a C_4 then
4         continue;
5     else
6         G_i ← G_i ∪ {e};
```

which we adapt for the C_4-free process. This is shown in Algorithm 11. Note that we have to check the condition in line 3 of Algorithm 11 in each iteration again because it can happen that an edge becomes closed during this process since we do not update $O(i)$ after each iteration. However, with the same arguments as for the triangle-free process its more efficient to combine both approaches. Hence, we define now the final simulation algorithm for the C_4-free process. The goal is to minimize the amount of open edges using the rejection process without having too much compuatational effort. Hence, we define again a tuning parameter $\alpha \in \mathbb{N}$ which specifies the amount of edge rejections we allow in a row before switching to the list approach of the process. This is analogously to the approach in Algorithm 3 for the triangle-free process and the approach is shown in Algorithm 12.

5.2.3. Experimental Evaluation

In this section, we want to evaluate our simulation results for the C_4-free process using similar techniques as for the triangle-free process. For this, we use only Algorithm 12 with different values for the rejection parameter $\alpha \in \mathbb{N}$ which specifies the number of edge rejections we allow in a row. The main goal of this section is to examine Conjecture 5.2.1. Note that for the C_4-free process we do not have as sophisticated results as for the triangle-free process and thus, getting an idea for the constant factor might be helpful for future research. In the rest of this section, we start by looking at some statistical properties of the random variable $e(G_{\square,n})$ in order to be able to evaluate the simulation results properly. Afterwards, we analyze the influence of the rejection parameter on the performance of the algorithm with regard to time and memory consumption. Next, we look at different models for our data that we generate using the non-linear least squares method and evaluate them with respect to precision and Conjecture 5.2.1.

Algorithm 12: C_4-free process using Algorihtm 10 and Algorithm 11

Data: Empty graph $G = ([n], \emptyset)$, tolerance α
Result: Resulting graph $G_{\square,n}$ of the C_4-free process

```
1  i ← 0;
2  steps ← 0;
3  openEdges = {};
4  while i ≤ α do
5      Edge e ← randomEdge;
6      if e ∈ E(G_steps) || e ∈ C(steps) then
7          reject e;
8          i ← i + 1;
9          if i > α then
10             break;
11     if e ∈ O(steps) then
12         G_steps ← G_steps ∪ {e};
13         steps ← steps+1;
14         i ← 0
15 for v ∈ [n] do
16     for u ∈ [n], u < v do
17         if (v, u) ∉ C(steps) and (v, u) ∉ E(G_steps) then
18             openEdges ← openEdges ∪ {(v, u)};
19 openEdges ← randomShuffle(openEdges);
20 for e ∈ openEdges do
21     if e ∉ C(steps) then
22         G_steps ← G_steps ∪ {e};
23         steps ← steps+1;
```

Sample mean	23 444,87
Sample standard deviation	20,07
Relative sample standard deviation	0,09
Sample variance	402,72

Table 5.1.: Some statistical results for the C_4-free process on 2 000 vertices with 200 repetitions.

Statistical results. We can only evaluate our simulation results correctly if we know which standard deviation we have to expect for the random variable $e(G_{\square,n})$. Hence, we repeat the C_4-free process on 2 000 vertices 200 times with different random seeds and look at some basic parameters which are shown in Table 5.2.3. We can see again that we only have a relative sample standard deviation of roughly 0.09% and hence we can expect that our models fit the data for a larger number of vertices similarly well. To

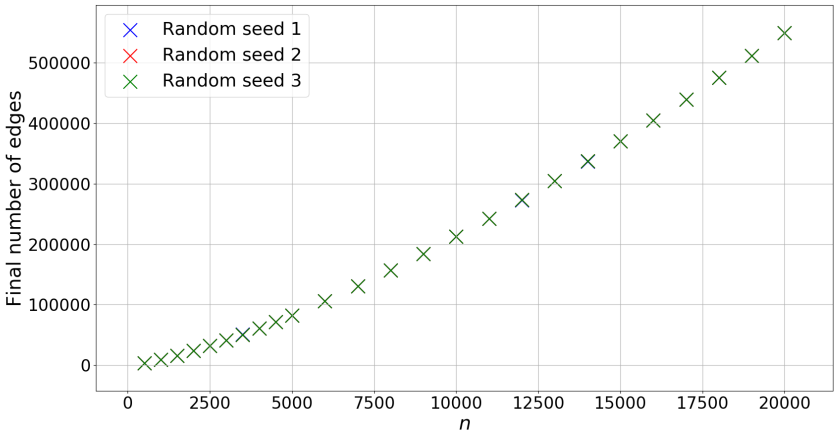


Figure 5.1.: For each number of vertices we plot three data points representing different random seeds and different values for the parameter α.

further underline this we give another plot which shows for every number of vertices n three data points, one for each random seed. Note again that we use for every repetition of the process a different parameter α since this does not influence the outcome of the algorithm, but only the time and memory performance. This plot can be seen in Figure 5.1. We can see that also for larger number of vertices the standard deviation does not seem to increase.

Influence of the parameter α. Now we want to do a similar analysis of the influence of the parameter α on the performance of the algorithm, as we did this for the triangle-free process. Note that the result quality is not influenced by this parameter. The intuition is for this process that the rejection approach gets more inefficient the further the process proceeds. However, note that the condition which we have to check to decide whether an edge is open or not is more complicated than this was the case for the triangle-free process. Hence, we see that the rejection parameter α even has a positive impact on the time performance of the algorithm. This behaviour can be seen in Table 5.2.3. What we can see in this table is that the behaviour is different from the behaviour of the triangle-free process. Even though we are able to decrease the number of open edges after the rejection phase significantly with more rejections, we see that this effect is not as big for the maximal memory consumption during the process. This is maybe due to the slightly different methodology to build the open edge list compared to the triangle-free process. The reason for this is the more complicated condition for an edge to be open and hence we want to minimize the number of times we have to check this condition. However, we can see that the number of rejections can even decrease the

n	$\alpha = 500$			$\alpha = 2\,500$			$\alpha = 5\,000$		
	$\lvert O(\alpha)\rvert$	t[s]	m[KB]	$\lvert O(\alpha)\rvert$	t[s]	m[KB]	$\lvert O(\alpha)\rvert$	t[s]	m[KB]
10 000	712 534	525,51	446 816	181 517	523,87	440 808	70 447	587,02	439 520
11 000	1 244 817	695,31	541 448	289 036	676,89	529 276	81 604	761,64	525 812
12 000	1 332 644	875,78	635 444	196 911	888,04	621 316	101 275	950,16	619 836
13 000	1 111 923	1 078,24	737 408	293 634	1 106,54	725 152	178 886	1 121,88	722 728
14 000	1 506 113	1 345,34	847 324	281 584	1 335,14	834 604	196 349	1 357,47	832 612
15 000	1 611 096	1 634,99	965 040	363 657	1 596,22	952 320	237 017	1 632,85	949 752
16 000	1 905 935	1 968,27	1 052 672	445 052	1 901,33	1 040 536	188 026	1 960,28	1 038 480
17 000	2 054 845	2 318,74	1 184 864	432 518	2 191,4	1 172 788	231 101	2 260,95	1 170 832
18 000	2 854 472	2 774,97	1 341 940	512 779	2 534,19	1 312 992	295 936	2 616,38	1 313 044
19 000	3 451 441	3 204,35	1 490 284	552 508	2 929,94	1 465 808	333 701	2 963,98	1 461 328
20 000	4 080 153	3 767,98	1 646 868	618 769	3 432,23	1 622 164	436 827	3 496,96	1 617 844

Table 5.2.: Influence of the rejection parameter α on the memory consumption and the overall time for the simulation of the C_4-free process on different numbers of vertices n. $|O(\alpha)|$ defines the number of open edges after the rejection process.

time for the simulation up to a certain point.

Models for the simulation data. In this part of the section, we want to look at different models for our simulation of the C_4-free process. Remember that we want to evaluate the results with respect to Conjecture 5.2.1 since the strongest result known for the final number of edges is

$$e(G_{\square,n}) = \Theta(n^{4/3}(\log n)^{1/3}).$$

Thus, we define

$$a^* = \frac{1}{\sqrt[3]{12}}, \quad b^* = \frac{4}{3}, \quad c^* = \frac{1}{3}.$$

First, we look at the model that tries to predict all three parmeters

$$f_1(n) := a \cdot n^{4/3} \cdot (\log n)^{1/3}$$

to see how close we are in this case. However, note that we might ignore an influence of a second order term with this model. We can see how this model fits the data in Figure 5.2 with the predicted parameters

$$a = 0.54, \quad b = 1.34, \quad c = 0.23.$$

The standard error for this fit of the data that we already introduced in Section 3.2.1

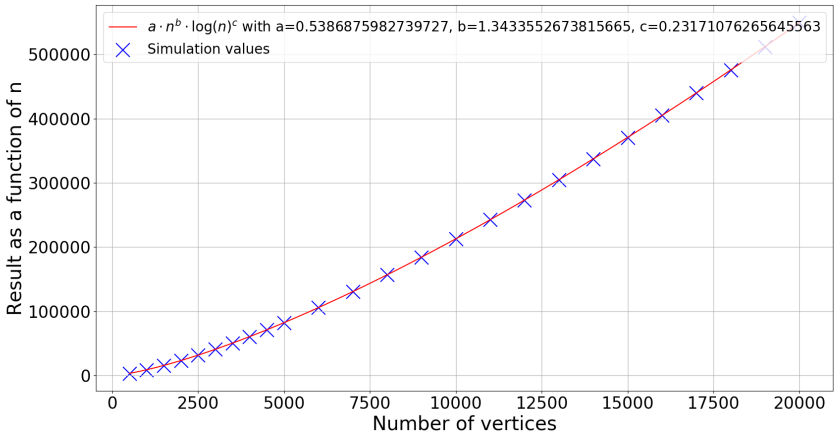


Figure 5.2.: Simulation values together with the predicted model $f_1(n)$.

gives us

$$S = 20{,}77.$$

Even though this is less than for the triangle-free process, note that this error is an absolute measure and not relative. Hence, we can not say directly that this model fits the data better than this was the case for the triangle-free process. However, we are mainly interested in a comparison of the models within one process where this measure suits well. Since we already know the exponents b and c for the C_4-free process, we are mainly interested in the constant factor a. Hence, we look at the next model

$$f_2(n) := a \cdot n^{4/3} (\log n)^{1/3}.$$

Note that this model delivered a reasonable precision for the constant factor a for the triangle-free process. A non-linear least squares method gives us for this model

$$a = 0.47$$

which is reasonable close to our conjecture a^*. Further, we get a standard error of

$$S = 81.32$$

for this model which is at least the same order of magnitude of $f_1(n)$. Hence, this model fits the data also reasonably well which can also be seen in Figure 5.3. The last model that we look at extends $f_2(n)$ by the possibility of the influence of a second order term.

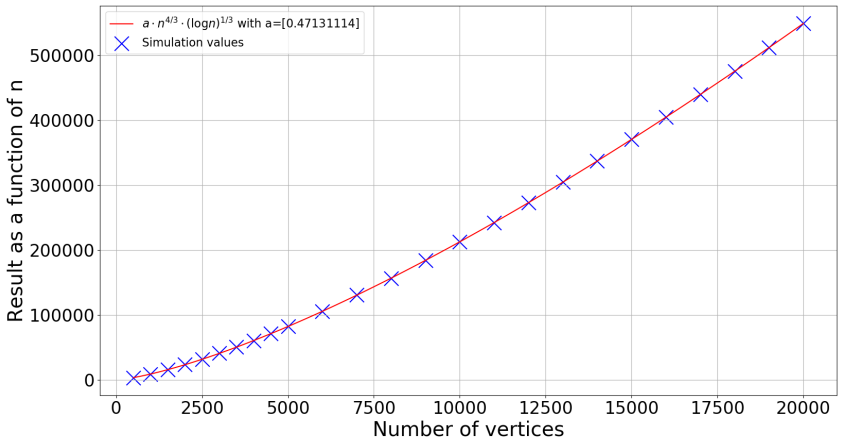


Figure 5.3.: Simulation values together with the predicted model $f_2(n)$.

This means we look at the model

$$f_3(n) := a \cdot n^{4/3}(\log n)^{1/3} + d \cdot n^{4/3}.$$

Note that this model delivered also a very good precision for the triangle-free process regarding the constant factor a as well as the standard error of the regression. For this model we get the values

$$a = 0.46, \quad d = 0.025.$$

This model deliveres a standard error of

$$S = 34.26$$

and we see that also for the C_4-free process this delivers a similar precision as the model $f_1(n)$ with respect to the fit of the data. Further, the results indicate that the influence of a second order term is much less than this was the case for the triangle-free process. However, we get even closer to our conjecture a^* with this model which, together with the standard error and the corresponding results for the triangle-free process, gives a strong hint that Conjecture 5.2.1 might be true. Further, note that the precision of the prediction of a^* is similar to the precision we achieved for the triangle-free process where we know the constant factor a^*. The fit of the data for the model $f_3(n)$ can be seen in Figure 5.4.

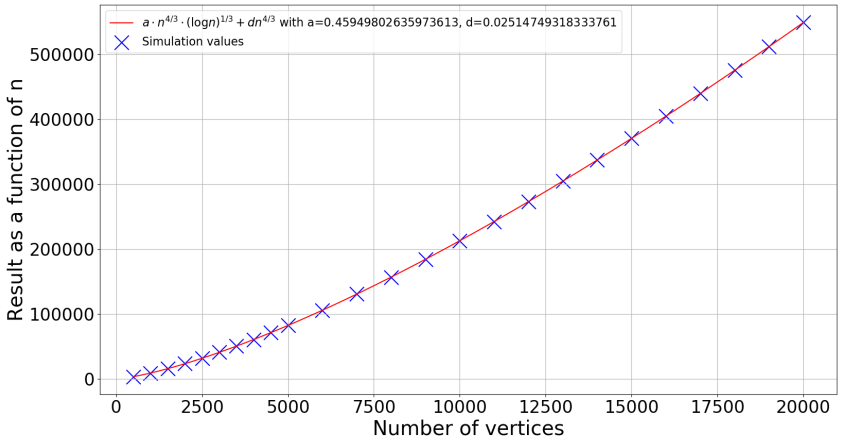


Figure 5.4.: Simulation values together with the predicted model $f_3(n)$.

6. The C_4-removal Process

In this section, we present our results for the C_4-removal process. To the best of our knowledge, it is not known when the C_4-removal process ends. Further, there is not much theory known for this process at all. Hence, our goal is to give a suggestion for the final number of edges. For this, we first give the usual definition of the C_4-removal process.

Definition 6.0.1. Let $(G_i)_{i\in\mathbb{N}}$ be the random graph process with $G_0 = K_n$ and G_i is obtained by G_{i-1} by selecting one copy of a C_4 in G_{i-1} uniformly at random and deleting its edges. Then the random graph process $(G_i)_{i\in\mathbb{N}_0}$ is called the C_4-removal process on n vertices.

Note that the process becomes stationary when there is no C_4 in the graph left. Let

$$\tau_0 := \min_{j\in\mathbb{N}_0}\{G_j \in (G_i)_{i\in\mathbb{N}_0} \text{ is } C_4\text{-free}\}.$$

Then we denote the graph G_{τ_0} as the final graph $G^r_{\square,n}$ of the process and the goal is to know how $e(G_{\tau_0})$ looks like. With a view to the triangle-removal process, we conjecture the following.

Conjecture 6.0.1. With high probability, we have

$$e(G^r_{\square,n}) = n^{\frac{4}{3}+o(1)}$$

for the number of edges in the final graph of the C_4-removal process.

The rest of this chapter is dedicated to Conjecture 6.0.1. In Section 6.1, we describe the algorithms we use for the simulation of the C_4-removal process. Thereby, we use similar techniques as for the triangle-removal process. An integral part of this is the description of how we list all possible copies of a C_4 in a graph. Afterwards, we present the results of our simulations in Section 6.1.1 and interpret them with respect to Conjecture 6.0.1.

6.1. Simulation of the C_4-removal process

In this section, we describe the algorithms for the simulation of the C_4-removal process. Thereby, we follow a similar approach as for the triangle-removal process. For this we define exact algorithms as well as heuristic algorithms. First of all, we note again that the C_4-removal process as stated in Definition 6.0.1 is equivalent to choosing four vertices uniformly at random and if they build a C_4, deleting its edges. This is repeated until there is no C_4 left. Further, the original process is also equivalent to building a list of all cycles on four vertices in a graph and then iterating over it in a random order and only delete the corresponding cycle if still all of its edges are contained in the graph. With this in mind, we look at the different algorithms.

Let $T(i)$ be the set of all C_4s in the graph G_i. Naively, we would start from a K_n and list all C_4s in a complete graph on n vertices. Afterwards, we can iterate over this list in a random order and check for every C_4 in this list whether its edges are still contained in the list. However, similar as for the triangle-removal process, this approach is not suitable since in a complete graph on n vertices we has $O(n^4)$ cycles on four vertices in the graph. This is neither in terms of running time nor in terms of memory suitable if n gets large. Another naive approach would be a similar algorithm as Algorithm 4. Thereby, we repeatedly choose four vertices uniformly at random and if they build a C_4, we remove its edges from the graph. Since the C_4-removal process becomes stationary at some point, this process will end after a finite number of steps as well. However, note that four pairwise different vertices can build three different cycles on four vertices. Let $v_1, v_2, v_3, v_4 \in V_i$ be four pairwise distinct vertices. Then the possible cycles are $(v_1, v_2, v_3, v_4), (v_1, v_2, v_4, v_3)$ and (v_1, v_3, v_2, v_4). Hence, every time we choose four vertices uniformly at random we have to choose one of the three possible cycles uniformly at random as well. This version of the process is described in Algorithm 13. However, using the same intuition as for the triangle-removal process, we can analyze how Algorithm 13 behaves over time. We define

$$p = p(i) := \frac{\binom{n}{2} - 4i}{\binom{n}{2}} \approx 1 - \frac{8i}{n^2}$$

for the edge density of the graph G_i during the process where we used $\binom{n}{2} \approx n^2/2$. Note that in every step of the process we delete 4 edges. Using the intuition that G_i should be similar to $G(n,p)$, we get

$$\mathbb{P}[\text{chosen vertices build a } C_4] \approx p^4 \approx \left(1 - \frac{8i}{n^2}\right)^4$$

Algorithm 13: Rejection version of the C_4-removal process

Data: Complete graph $G = (V_0 = [n], E_0 = E[K_n])$
Result: Resulting graph $G^r_{\square,n}$ of the C_4-removal process

```
1  i ← 0;
2  while (T(i) > 0) do
3      Vertex v_1 ← randomVertex;
4      Vertex v_2 ← randomVertex;
5      Vertex v_3 ← randomVertex;
6      Vertex v_4 ← randomVertex;
7      C(v_1, v_2, v_3, v_4) ← randomCyclePermutation;
8      if C(v_1, v_2, v_3, v_4) is C_4 in G_i then
9          delete edges of C(v_1, v_2, v_3, v_4);
10         i ← i + 1;
11     else
12         reject C(v_1, v_2, v_3, v_4);
```

and hence we see that this is even worse than for the triangle-removal process. Consequently, Algorithm 13 is not suitable for the simulation of the whole process. Thus, we want to use a similar approach as for the triangle-removal process but first we need an algorithm which lists all cycles on four vertices in a given graph G once. From a probabilistic point of view, it is not important to list every C_4 only once as long as we list every C_4 the same number of times. However, note that we can expect in a $G(n,p)$ roughly $n^4p^4/8$ cycles on four vertices and hence listing every C_4 more than once in the graph results in too much unnecessary work. Now the listing algorithm works as follows. We start by iterating over every vertex $v \in V(G)$ in the graph. Now we want to list every C_4 starting from v. We store a global vector `seenBeginnings` which has the value `true` in entry w if we already listed all cycles starting from w. This vector will be important in order to list every C_4 only once. The idea for this vector is that if we look at one vertex, we can list every cycle on four vertices containing this vertex by starting from this vertex. Hence, if we want to list further cycles, we do not have to look at cycles containing this vertex. Further, in every iteration, we store a vector `forbiddenEndings`. This vector makes sure that we really have cycles and do not use vertices multiple times. Now we run an algorithm in a *BFS* fashion to list all cycles on four vertices. First, we set the `seenBeginnings` value for v to `true`. Afterwards, we look at every vertex in $N(v)$. Those are precisely the possible second vertices for a cycle on four vertices and we set the `forbiddenEndings` value for these vertices to `true`. Since they are second vertices, they must not appear in the cycle again. Further, we only continue with one of these vertices if the `seenBeginnings` value for these vertices is `false`. Now we look at

Algorithm 14: `c4List` : List all cycles on four vertices in a given graph $G = (V, E)$

```
Data: Graph G = (V, E) with n vertices
Result: List of all cycles on four vertices in G
1  T ← ∅;
2  Vector seenBeginnings ← (n, false);
3  for v ∈ V do
4      Vector forbiddenEndings ← (n, false);
5      seenBeginnings[v] ← true;
6      for w ∈ N(v) do
7          if seenBeginnings[w] == false then
8              forbiddenEndings[w] ← true;
9              for x ∈ N(w) \ {v} do
10                 if seenBeginnings[x] == false then
11                     for y ∈ N(x) \ {v, w} do
12                         if seenBeginnings[y] == false then
13                             Edge e ← (v, y);
14                             if e ∈ E and forbiddenEndings[y] == false then
15                                 Cycle c ← (v, w, x, y);
16                                 T ← T ∪ {c};
```

the neighbourhood of the second vertex except of v itself. Those vertices are precisley the third possible vertices in a cylce. Again, we check whether they are equal to one of the first two vertices and whether they were already a beginning vertex. If this is not the case, we can use it as a third vertex of the cycle. Now we look at the neighbourhood again and if the fourth vertex is pairwise distinct with the first three vertices and it is not a forbidden ending than we found a cycle if the edge between the fourth vertex and v is contained in the graph. For this, we can use a hash map again or just search for it in the neighbourhood using our dynamic graph data structure. More details of this algorithm are shown in Algorithm 14. Now we want to prove that Algorithm 14 indeed lists all cycles on four vertices only once.

Lemma 6.1.1. Let $G = (V, E)$ be an arbitrary graph on n vertices. Then Algorithm 14 lists every C_4 in G exactly once.

Proof. Let $C = (v_1, v_2, v_3, v_4)$ be an arbitrary copy of a C_4 in G. Assume without loss of generality that $v_1 = \min\{v_1, v_2, v_3, v_4\}$. Now at some point, Algorithm 14 will look at v_1 and the other vertices of the cycle have not been seen yet. Without loss of generality, let Algorithm 14 view v_2 before v_4 when it looks at the neighbourhood of v_1. Then by

the algorithm, v_4 will not be a forbidden ending for the cycle and neither v_2, v_3 nor v_4 are seen beginnings. Hence, Algorithm 14 will find C when exploring v_1. Since C was arbitrary in G it follows that Algorithm 14 finds all cycles on four vertices in G. What remains to show is that we do not put the same cycle into the list more then once. With the same assumptions as above, we first assume that we find C again while looking at the cycles starting from v_1. However, this would only be possible by viewing v_4 first and v_2 last by definition of the algorithm. But since we viewed v_2 before v_4, v_2 will be a forbidden ending and hence the cycle (v_1, v_4, v_3, v_2) will not be added to the list. Now we assume that C is added to the list at another time. By definition of the algorithm, this can only be the case if we view one of the vertices v_2, v_3 or v_4. But if we do so, the algorithm will view v_1 at some point as part of C and since v_1 is already a seen beginning, this cycle will not be added to the list. Since those were the only possibilities for C to get added to the list, the lemma follows. □

Now, since we have an Algorithm to list all C_4s in the graph, we are able to formulate our simulation algorihtm for the C_4-removal process. The idea for this is similar as for the triangle-removal process. We simply combine the rejection process as described in Algorithm 13 and after we have a certain amount of rejections $\alpha \in \mathbb{N}_0$ in a row. We list all remaining cycles on four vertices in the current graph and iterate over it in a random order as described above. Note that α is a tuning parameter which influences the time and the memory consumption of the algorithm. This approach results in Algorithm 15 However, as we already pointed out above, the rejection phase in Algorithm 15 becomes computationally inefficient if the edge density of the graph decreases. Thus, the idea is again to use our intuition for the C_4-removal process to get faster to a lower edge density. This is again done by generting a $G(n, p)$ with only a fraction of the edges using `KaGen` as described in Section 4.2. Afterwards, we directly apply Algorithm 14 to get a list of the C_4s in the generated $G(n, p)$ and we proceed as in Algorithm 15. This heuristic is shown in Algorithm 16.

6.1.1. Experimental Evaluation

In this section, we analyze the simulation results for the C_4-removal process using the same techniques as for the triangle-removal process. As we already saw for the triangle-removal process, the simulation algorithms switch to a list of all remaining copies of a C_4 at some point. This leads to a high memory consumption if the edge density is not low enough. This turns out to be an even bigger problem for the C_4-removal process. Hence, we are able to simulate the original process only up to 3 000 vertices. The reason for that can be seen in Table 6.1.1. However, due to the low standard deviation for our

Algorithm 15: Simulation algorithm for the C_4-removal process

Data: Number of vertices n
Result: Resulting graph of the C_4-removal process $G^r_{\square,n}$.

```
1  delEdges : binom([n],2) → {true, false};
2  while i ≤ α do
3      Vertex v_1 ← randomVertex;
4      Vertex v_2 ← randomVertex;
5      Vertex v_3 ← randomVertex;
6      Vertex v_4 ← randomVertex;
7      C(v_1, v_2, v_3, v_4) ← randomCyclePermutation;
8      e_1, e_2, e_3, e_4 ←  edges of C(v_1, v_2, v_3, v_4);
9      if delEdges(e_i) = true for some i ∈ [4] then
10         reject C(v_1, v_2, v_3, v_4);
11         i ← i + 1;
12         if i > α then
13             exit rejection process;
14     else
15         delEdges(e_1) ← true;
16         delEdges(e_2) ← true;
17         delEdges(e_3) ← true;
18         delEdges(e_4) ← true;
19         i ← 0;
20 G_{τ_α} ← build remaining graph;
21 T_{τ_α} ← c4List(G_{τ_α});
22 T_{τ_α} ← randomShuffle(T_{τ_α});
23 for C_4 c = (v_1, v_2, v_3, v_4) ∈ T_{τ_α} do
24     e_1, e_2, e_3, e_4 ←  edges of C(v_1, v_2, v_3, v_4);
25     if e_1, e_2, e_3, e_4 ∈ G_{τ_α} then
26         G_{τ_α} ← G_{τ_α} − {e_1, e_2, e_3, e_4};
```

simulations we are confident that the results are still reliable.

Statistical results. As usual, we start with some statistical results for the simulation of the random variable $e(G_{\square,n})$. However, since this is the most memory intensive process, we only do 200 repetitions of the simulation on 500 vertices. This can be seen in Table 6.1.1. Even though we can see that we have a sample standard deviation of roughly two percent, we will see that this seems not to be the case if the number of vertices gets larger and we expect that our results are still similarly reliable as for the triangle-removal process. This can be seen later in this section, when we look at models that we apply to our data. In general, we expect the sample standard deviation for a

Algorithm 16: Simulate the C_4-removal process starting from a $G(n,p)$ for $n \in \mathbb{N}$ and $p \in [0,1]$.

Data: Number of vertices $n \in \mathbb{N}$, initial edge density $p_\alpha \in [0,1]$.
Result: C_4-free graph $G_{\square,p_\alpha}$ obtained by the C_4-removal process.

```
1  G(n, p_α) ← KaGen(n, p_α);
2  G_{□,p_α} ← G(n, p_α);
3  delEdges : binom([n], 2) → {true, false};
4  T_{p_α} ← c4List(G(n, p_α));
5  T_{p_α} ← randomShuffle(T_{p_α});
6  for C_4 c = {e_1, e_2, e_3, e_4} ∈ T_{p_α} do
7      if delEdges(e_1) = delEdges(e_2) = delEdges(e_3) = delEdges(e_4) = false
       then
8          G_{□,p_α} ← G_{□,p_α} − {e_1, e_2, e_3, e_4};
9          delEdges(e_1) ← true;
10         delEdges(e_2) ← true;
11         delEdges(e_3) ← true;
12         delEdges(e_4) ← true;
```

Sample mean	1 415,3
Sample standard deviation	27,81
Relative sample standard deviation	1,96
Sample variance	773,30

Table 6.1.: Some statistical results for the C_4-removal process on 500 vertices with 200 repetitions.

smaller number of vertices to be larger.

Time and memory performance. In this section, we want to analyze the time and memory performance of Algorithm 15 and Algorithm 16. Thereby, we vary the number of rejections we allow in a row α for Algorithm 15, and the initial edge density p for Algorithm 16. Note that the parameter α only influences the time and memory performance but not the solution quality. We show the behaviour for Algorithm 15 in Table 6.1.1 and Table 6.1.1. What we can see for the time is that in the beginning a larger number of rejections even improves the running time. The reason for this is probably that there are to many copies of a C_4 that we have to list. Note that by deleting the edges of one copy of a C_4 we actually destroy every copy of a C_4 which contains one of the deleted edges. However, this effect only holds for a smaller number of rejections. Afterwards, the simulation takes longer which might be due to the low probability for a random pick of a C_4. We described this probability already in the previous section.

n	$\alpha = 1\,000$	$\alpha = 5\,000$	$\alpha = 10\,000$	$\alpha = 15\,000$	$\alpha = 20\,000$
500	24,20	15,30	20,04	29,71	39,70
1 000	778,32	316,33	470,75	520,61	594,68
1 500	-	1 474,29	1 486,65	1 833,48	2 063,1
2 000	-	5 822,38	5 372,23	5 824,4	5 810,88
2 500	-	-	10 283,4	10 272,3	10 795,1
3 000	-	-	-	21 260,9	21 114,1

Table 6.2.: Influence of the rejection parameter α on the overall time for the simulation of the C_4-removal process on different numbers of vertices n.

n	$\alpha = 1\,000$	$\alpha = 5\,000$	$\alpha = 10\,000$	$\alpha = 15\,000$	$\alpha = 20\,000$
500	559 956	150 672	81 080	47 496	46 152
1 000	17 038 176	2 180 696	1 102 328	1 099 460	571 388
1 500	-	16 976 620	8 554 372	4 311 844	4 300 788
2 000	-	67 108 952	17 005 544	16 939 804	17 024 300
2 500	-	-	67 602 204	33 991 712	33 926 264
3 000	-	-	-	67 385 232	67 583 468

Table 6.3.: Influence of the rejection parameter α on the maximal memory consumption for the simulation of the C_4-removal process on different numbers of vertices n.

On the other hand, we can see that a larger number of rejections that we allow in a row significantly reduces the maximal memory consumption that the algorithm uses. In fact, for a larger number of vertices and a low number of rejections this is even to much memory for the machine that we used for the simulations. Hence, we analyze now the behaviour in this regard of Algorithm 16. This can be seen in Table 6.1.1 for the two largerst edge densities used. We can see in this table that we have significant improvements for the memory and the time consumption. Further, we can see that the increase in the edge density does not correspond to a larger result. This indicates that we have a similar convergence behaviour as for the triangle-removal process. We underline this by the model predictions that we give now.

Models for the simulation data. Now we look at different models with respect to Conjecture 6.0.1. Further, we compare for one model the predictions that we get by the original data with the predictions that we get by applying the same model to the data generated by Algorithm 16. This is used to show the convergence behaviour of this algorithm since we can not compare this with the results for the real process as we have done this for the triangle-removal process. With regard to Conjecture 6.0.1, we

n	ϵ_1	t[s]	m[KB]	Result	ϵ_2	t[s]	m[KB]	Result
5 000	0,02	99,30	553 132	30 009	0,03	468,52	2 134 632	30 078
6 000	0,01	17,72	83 348	38 129	0,015	81,61	284 824	38 355
7 000	0,01	37,34	152 040	46 600	0,015	175,72	551 144	46 739
8 000	0,01	73,36	286 400	56 130	0,015	337,89	1 081 168	56 069
9 000	0,01	131,78	290 356	65 376	0,015	650,39	2 138 340	65 357
10 000	0,01	228,38	571 596	75 658	0,015	1 187,21	2 146 096	75 583
11 000	0,01	382,35	1 085 120	85 441	0,015	1 957,06	4 251 936	85 703
12 000	0,008	258,48	559 736	96 068	0,01	606,82	1 091 080	95 688
13 000	0,008	406,03	563 580	107 213	0,01	962,93	2 147 088	107 546
14 000	0,008	576,44	1 092 652	118 211	0,01	1 523,04	2 154 892	118 464
15 000	0,008	854,57	1 098 412	129 937	0,01	2 166,41	2 161 744	129 829

Table 6.4.: Results for Algorithm 16 for the two largest edge densities.

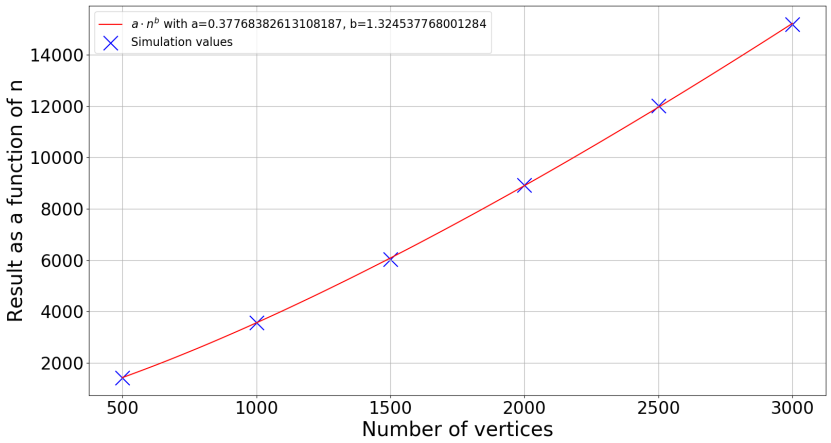


Figure 6.1.: Simulation results together with the predicted model $f_1(n)$.

first want to confirm the main factor $n^{4/3}$. For this we look at the model

$$f_1(n) = a \cdot n^b.$$

Using the data of the original process this yields

$$a = 0.38, \quad b = 1.32$$

with a standard error of

$$S = 43.91.$$

Further, we can see that this is a good fit of the data in Figure 6.1. We can see that

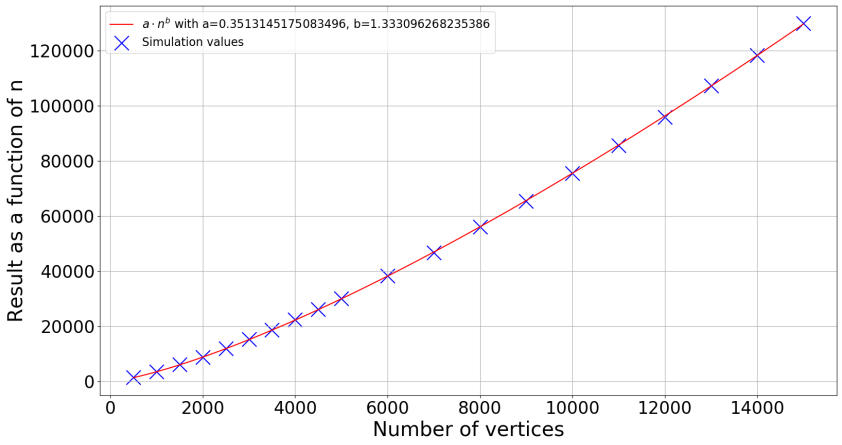


Figure 6.2.: Heuristic simulation results together with the predicted model $f_1(n)$.

this confirms the exponent in Conjecture 6.0.1. Additionally, this is also confirmed by the predictions of the same model with the heuristic data. This yields

$$a_h = 0.35, \quad b_h = 1.33$$

with a standard error

$$S = 191.07$$

and can be seen in Figure 6.2. This also indicates that our heuristic algorithm is suitable for the simulation of the C_4-free process. Finally, we want to apply the model

$$f_2(n) = a \cdot n^{4/3} \cdot (\log n)^c$$

to examine the possibility of a second factor in the first order term. This yields for the original data

$$a = 0.40, \quad c = -0.07, \quad S = 44.23.$$

This indicates that there might be no logarithmic factor for the leading term. This can be seen in Figure 6.3. Further, this is underlined by the results for the heuristic data which yields

$$a = 0.35, \quad c = -0.004, \quad S = 190.93.$$

In conclusion, we want to mention that even the constant factor a only varies in a certain range, suggesting the possibility that this can also be determined. However, we are not able to come up with a profound intuition for this.

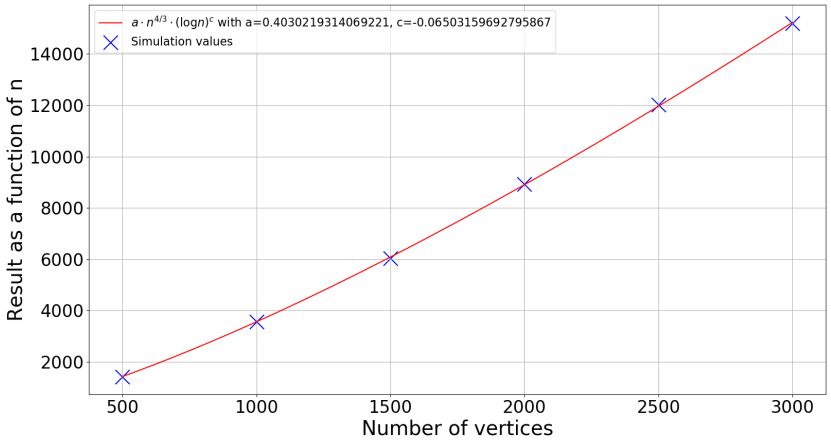


Figure 6.3.: Simulation results together with the predicted model $f_2(n)$.

7. The K_4-free Process

7.1. Theoretical Results

In this section, we want to summarize the theoretical results regarding the K_4-free process following the work of Warnke [Warnke, 2014]. In order to not go beyond the scope of this work and due to similar arguments, we go into less detail here and only provide an overview on the arguments. The K_4-free process is the random graph process where we start from $G_0 = ([n], \emptyset)$ and in every step $j \in \mathbb{N}$ the graph G_j is obtained by G_{j-1} by choosing one open edge uniformly at random. An edge is called open in this context if its insertion to G_{j-1} does not close a copy of a K_4. Also for this process the first non trivial results where given by Bollobás and Riordan [Bollobás and Riordan, 2000], and Osthus and Taraz [Osthus and Taraz, 2001]. In both cases, they proved upper and lower bounds that matched within a logarithmic factor. Further, Bohman and Keevash [Bohman and Keevash, 2009] provide a lower bound of $\Omega(n^{8/5}(\log n)^{1/5})$ which they conjecture to be tight up to a constant factor. Finally, in the work that we present in this section of Warnke [Warnke, 2014], they matched this lower bound by giving an upper bound for the maximal degree of the graph. This proves $e(G_{\boxtimes,n}) = \Theta(n^{8/5}(\log n)^{1/5})$ with high probability. As well as for the case $H = K_3$ and $H = C_4$ the main goal of this work is to provide an asymptotic for the number of edges in the final graph of the K_4-free process. Hence, the main theorem we want to present in this section is the following.

Theorem 7.1.1. With high probability as $n \to \infty$ we get

$$e(G_{\boxtimes,n}) = \Theta(n^{\frac{8}{5}}(\log n)^{\frac{1}{5}}) \tag{7.1}$$

for the final graph $G_{\boxtimes,n}$ of the K_4-free process.

Due to the similarity of the notation, we adapt the notation from Chapter 5 here. Note that the lower bound of Theorem 7.1.1 follows again with Theorem 2.3.1 for the case $H = K_4$. What remains to show is consequently the upper bound in Theorem 7.1.1, which was done by Lutz [Warnke, 2014]. Their main theorem is the following.

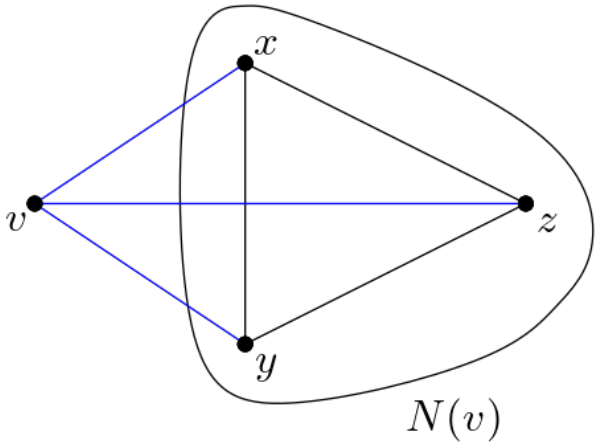


Figure 7.1.: A vertex v with a triangle in its neighbourhood $N(v)$.

Theorem 7.1.2 ([Warnke, 2014], Theorem 1.1.)**.** There exists $C > 0$ such that with high probability the maximum degree in the graph generated by the K_4-free process is at most $Cn^{3/5}(\log n)^{1/5}$.

Using again our standard estimate

$$e(G_{\boxtimes,n}) \leq n \cdot \Delta(G_{\boxtimes,n})$$

gives us with Theorem 7.1.2 the bound

$$e(G_{\boxtimes,n}) \leq Cn^{\frac{8}{5}}(\log n)^{\frac{1}{5}}$$

with high probability and hence Theorem 7.1.1 follows. Consequently, it remains to prove Theorem 7.1.2. The main idea of the proof comes from the following observation. Assume we have a K_4 in an graph G. Then this is the case if and only if there exists a vertex $u \in V(G)$ such that $G[N(u)]$ contains a triangle. This can be seen in Figure 7.1. Hence, by the definition of the K_4-free process at every step $i \in \mathbb{N}_0$ we know that there is no vertex $v \in G_i$ such that $G_i[N(v)]$ contains a triangle. The idea of the proof is now to show that after a certain amount of steps in the K_4-free process every set of vertices that is large enough already contains a triangle. Afterwards, we have to show that we can apply this to the neighbourhoods of the graph. The trick of this proof is to show that this is already the case during the time when most of the needed quantities can still be controlled with the techniques used by Bohman et al. [Bohman and Keevash, 2009] since the behaviour of the H-free process after this time is in general not well enough understood except for the triangle-free process. Hence, we start the proof by looking at the constants and functions we already introduced in Section 2.3 and deduce some results

specifically for the K_4-free process from Theorem 2.3.2. First, we define $p := n^{-2/5}$ and define our continous time variable $t := t(i) = i/n^{8/5}$. Further, let μ, ϵ, W be constants as in Section 2.3. Specifically, we have

$$W \geq 500, \quad \epsilon \leq \frac{1}{1000}, \quad 2W\mu^5 \leq \epsilon. \tag{7.2}$$

Note that there is no conflict with the specifications of the constants in Section 2.3. Further, we define

$$t_{\max} := \mu(\log n)^{1/5} \quad \text{and} \quad m := n^2 p t_{\max} = \mu n^{8/5}(\log n)^{1/5}. \tag{7.3}$$

Now we specify the functions

$$q(t) := e^{-16t^5} \quad \text{and} \quad f(t) := e^{W(t^5+t)} \tag{7.4}$$

from Section 2.3. For an intuition for $q(t)$ see Section 5.1. The intuition is transferred analogously for the K_4-free process. Further, note that using the constraints for our constants in Section 2.3 and above we can choose them such that for every $0 \leq t \leq t_{\max}$ we obtain

$$1 \geq q(t) \geq n^{-\epsilon/2} \quad \text{and} \quad 1 \leq f(t)q(t)^2 \leq f(t) \leq n^{\epsilon}. \tag{7.5}$$

For the proof of our main theorem, we have to use Theorem 2.3.2 again as described in Section 2.3 and can derive the following result.

Theorem 7.1.3 ([Warnke, 2014], Theorem 2.1, [Bohman and Keevash, 2009])**.** Set $s_e := n^{1/12-\epsilon}$. Further, let $\mathcal{G}_j$ denote the event that for every $0 \leq i \leq j$, in G_i we have $|O(i)| > 0$, and for all distinct vertices $u, v \in [n]$ we have

$$\begin{aligned} &|O(i)| = (1 \pm \frac{3f(t)}{s_e})q(t)\frac{n^2}{2}, \\ &|d_i(u)| \leq 3npt_{\max}, \\ &|N_i(u) \cap N_i(v)| \leq (\log n)np^2. \end{aligned} \tag{7.6}$$

Then the event $\mathcal{G}_m$ holds with high probability in the K_4-free process.

Now we also need the main technical result of Warnke [Warnke, 2014] in order to prove Theorem 7.1.2. For this we first need a few more parameters

$$\delta := \frac{1}{7000}, \quad \gamma := \max\left\{\frac{5}{\sqrt{\delta}\mu^{5/2}}, 150\right\}, \quad u := \gamma npt_{\max} = \gamma\mu n^{3/5}(\log n)^{1/5}. \tag{7.7}$$

Thereby, u specifies the size of the neighborhoods we want to consider.

Theorem 7.1.4 ([Warnke, 2014], Theorem 3.1.)**.** Let $\mathcal{T}_j$ denote the event that for all $n^{8/5} \leq i \leq j$, in G_i every set $U \subseteq [n]$ of size u contains at least $\delta u^3 (tp)^2 q(t)$ open pairs in $O(i)$ which would complete a copy of a triangle in U if they were added to G_i. Then $\mathcal{T}_m$ holds with high probability in the K_4-free process.

Intuitively, what Theorem 7.1.4 means is that every set $U \subseteq [n]$ which is large enough contains many open pairs which would close a triangle in U. This of course raises the probability that a K_4 is closed in the next step as we already explained in the beginning of this section. With this theorem, we have everything to prove our main result. As we already explained, the main idea of the proof is that every neighbourhood in a K_4-free graph has to be triangle-free. What we show in the following proof is that every subset of the vertices of size u contains a triangle with high probability. Consequently, we can bound the size of every neighbourhood by u and obtain an upper bound for the maximum degree in the final graph.

Proof of Theorem 7.1.2. Let $U \subseteq [n]$ be a subset of the vertices for the graph generated by the K_4-free process and let $i \leq m$ be a step in the process. Let $\mathcal{E}_{U,i}$ be the event that up to step i, the set U is triangle-free. With this let $\mathcal{E}_m$ be the event that there exists a subset $U \subseteq [n]$ such that $\mathcal{E}_{U,m}$ holds. Furthermore, for $i \leq m$ we define the event $\mathcal{H}_i := \mathcal{G}_i \cap \mathcal{T}_i$. Note that $\mathcal{H}_i$ only depends on the first i steps of the process and that the event $\mathcal{H}_{i+1}$ implies $\mathcal{H}_i$. Using the definitions of the events we can see now that the theorem follows if we can show

$$\mathbb{P}[\mathcal{E}_m \cap \mathcal{H}_m] = o(1). \tag{7.8}$$

Intuitively, what equation 7.8 means is that the probability that the process is still active $|O(m)| > 0$ and there is a set of size u which contains no triangle is $o(1)$. By Theorem 7.1.3 and Theorem 7.1.4 we already know that the event $\mathcal{H}_m$ holds with high probabilty and consequently Equation 7.8 implies $\mathbb{P}[\mathcal{E}_m] = o(1)$ which is exactly what we have to show since this implies that there is with high probability no subset of the vertices of size u which contains no triangle in step m which again implies our bound for the maximum degree by applying this statement to the neighbourhood of any vertex in the graph. For the proof of Equation 7.8 let $U \subseteq [n]$ of size u be fixed as above and define $C := \gamma\mu$. Further, we define

$$T_U(i) := \{(u,v) \in O(i) \ : \ G_i[U \cup \{(u,v)\}] \text{ contains a copy of a } K_3\} \tag{7.9}$$

to be the set of all open pairs, which would close a triangle in U. We start by calcu-

lating the probability in Equation 7.8 for one choice of U and then use an union bound argument. We get by our explanations above

$$\begin{aligned}\mathbb{P}[\mathcal{E}_{U,m} \cap \mathcal{H}_m] &= \mathbb{P}[\mathcal{E}_{U,n^{8/5}} \cap \mathcal{H}_{n^{8/5}}] \cdot \prod_{n^{8/5} \leq i \leq m-1} \mathbb{P}[\mathcal{E}_{U,i+1} \cap \mathcal{H}_{i+1} | \mathcal{E}_{U,i} \cap \mathcal{H}_i] \\ &\leq \prod_{n^{8/5} \leq i \leq m-1} \mathbb{P}[e_{i+1} \notin T_U(i) | \mathcal{E}_{U,i} \cap \mathcal{H}_i].\end{aligned} \tag{7.10}$$

For the inequalities, note that as already explained, the event $\mathcal{E}_{U,i} \cap \mathcal{H}_i$ only depends on the first i steps of the process. Hence, if we are at this point, the next edge is chosen uniformly at random from $O(i)$ and of course the event $e_{i+1} \notin T_U(i)$ given $\mathcal{E}_{U,i} \cap \mathcal{H}_i$ is weaker than $\mathcal{E}_{U,i+1} \cap \mathcal{H}_{i+1}$ given $\mathcal{E}_{U,i} \cap \mathcal{H}_i$. Now note that the event $\mathcal{G}_i$ together with our assumptions implies

$$q(t) \geq \frac{|O(i)|}{n^2} \tag{7.11}$$

for $n^2 p \leq i \leq m$. If we use now the definition of our time variable t, we get now on $\mathcal{H}_i = \mathcal{G}_i \cap \mathcal{T}_i$ using $n^{8/5} = n^2 p$ that

$$|T_U(i)| \overset{7.1.4}{\geq} \delta u^3 (tp)^2 q(t) = \delta u^3 \frac{i^2}{n^4 p^2} p^2 q(t) \geq \delta \frac{u^3 i^2}{n^6} |O(i)|. \tag{7.12}$$

Now note that

$$\mathbb{P}[e_{i+1} \text{ is not chosen from } T_U(i)] = 1 - \frac{|T_U(i)|}{|O(i)|}. \tag{7.13}$$

Hence, we get by Equality 7.10 using the standard inequality $1 - x \leq e^{-x}$ that

$$\begin{aligned}\mathbb{P}[\mathcal{E}_{U,m} \cap \mathcal{H}_m] &\leq \prod_{n^{8/5} \leq i \leq m-1} \mathbb{P}[e_{i+1} \notin T_U(i) | \mathcal{E}_{U,i} \cap \mathcal{H}_i] \\ &\leq \prod_{n^{8/5} \leq i \leq m-1} \left(1 - \frac{|T_U(i)|}{|O(i)|}\right) \\ &\leq \prod_{n^{8/5} \leq i \leq m-1} \exp\left(-\frac{|T_U(i)|}{|O(i)|}\right) \\ &= \exp\left(\sum_{n^{8/5} \leq i \leq m-1} -\frac{|T_U(i)|}{|O(i)|}\right) \\ &\leq \exp\left(-\frac{\delta u^3}{n^6} \sum_{n^{8/5} \leq i \leq m-1} i^2\right) \\ &\leq \exp\left(-\frac{\delta u^3 m^3}{4n^6}\right),\end{aligned} \tag{7.14}$$

where the last inequality follows for n large enough by simple estimates. Now note that $u = \gamma n^{3/5} t_{\max}$. Using this and the definitions of our parameters, we get using the inequalities above

$$\begin{aligned}
\mathbb{P}[\mathcal{E}_{U,m} \cap \mathcal{H}_m] &\leq \exp\left(\frac{\delta}{4} \cdot \frac{u^3 n^6 p^3 t_{\max}^3}{n^6}\right) \\
&= \exp\left(\frac{\delta}{4} u^3 p^3 t_{\max}^3\right) \\
&= \exp\left(\frac{\delta\gamma^2}{4} n^{6/5} t_{\max}^5 p^3 u\right) \\
&= \exp\left(\frac{\delta\gamma^2}{4} t_{\max}^5 u\right) \\
&= \exp\left(\frac{\delta}{4}\gamma^2 \mu^5 (\log n) u\right) = n^{-\frac{\gamma^2 \delta \mu^5}{4} u} \leq n^{-2u},
\end{aligned} \tag{7.15}$$

where the last inequality follows by the choice of γ. Now note that we have $\binom{n}{u}$ possible subsets U with our constraints. Using the simple inequality $\binom{n}{u} < n^u$, we get by a simple union bound argument over all possible subsets U that

$$\mathbb{P}[\mathcal{E}_m \cap \mathcal{H}_m] \leq \binom{n}{u} n^{-2u} = o(1), \tag{7.16}$$

by definition of u and hence Equation 7.8 follows. As already explained we can now apply this result to the neighborhoods of our graph generated by the K_4-free process. If there was a vertex with a degree higher than u, its neighbourhood would contain a triangle and hence we would have a copy of a K_4 in our graph which is a contradiction to the definition of the process. Consequently, we get

$$\Delta(G_{\boxtimes}, n) \leq C n^{3/5} (\log n)^{1/5} \tag{7.17}$$

with high probability and the theorem follows. □

As we already explained, with the proof of Theorem 7.1.2 we already proved Theorem 7.1.1.

7.2. Simulation of the K_4-free process

In this section, we present our algorithms for the K_4-free process. Thereby, we follow a similar approach as for the triangle-free process and the C_4-free process. Hence, during the presentation of the algorithms we mainly focus on the key differences between

Algorithm 17: Rejection version of the K_4-free process

Data: Empty graph $G = (V_0 = [n], E_0 = \emptyset)$
Result: Resulting graph $G_{\boxtimes,n}$ of the K_4-free process

```
1 i ← 0;
2 while (O(i) > 0) do
3     Edge e ← randomEdge;
4     if e ∈ E(G_i) || e ∈ C(i) then
5         reject e;
6     else
7         G_i ← G_i ∪ {e};
8         i ← i + 1;
```

the processes and describe the algorithm itself only shortly since the idea is the same. Further, we want to give a conjecture for a more precise statement for the number of edges in the final graph using the same intuition as for the C_4-free process. In particular, the arguments for the equivalence of the algorithm versions of the K_4-free process remain the same. For detailed arguments about this, we refer to Section 3.2. In general, our main simulation algorithm consists of two phases. We start with a rejection phase for the K_4-free process where we select in every step one edge $e \in E(K_n)$ uniformly at random and reject it if it is already contained in the current graph or is closed, i.e., it would close a K_4 in the graph. Otherwise, we add it to the graph. This algorithm can also be used as a simulation algorithm for the K_4-free process if we repeat that until there is no open edge left. This can be seen in Algorithm 17, where the notation is analogously used as in the previous chapters. However, the crucial part of this algorithm is line 4, i.e., how to determine whether an edge would close a K_4 in the graph or not. To check whether an edge is already contained in the graph, we proceed as for the triangle-free process and the C_4-free process by using a hash map for example. To decide whether an edge $e = (u, v)$ is closed or not, we first observe that the remaining two vertices of the potential K_4 have to be contained in $N(u) \cap N(v)$. Hence, we have to compute the intersection of the corresponding vertices first. Afterwards, we can already exclude that e closes a K_4 if $|N(u) \cap N(v)| < 2$ since in this case there are not enough vertices in the intersection for a potential K_4. The scenario when $|N(u) \cap N(v)| \geq 2$ is shown in Figure 7.2. However, if $|N(u) \cap N(v)| \geq 2$, we still have to check whether there is an edge between the corresponding two vertices. If we have a pair of vertices in $N(u) \cap N(v)$ which share an edge, we know that the edge (u, v) closes a copy of a K_4. This results in Algorithm 18. However, we know from our theoretical observations that the K_4-free process is particularly in the beginning similar to $G(n, p)$ with the corresponding

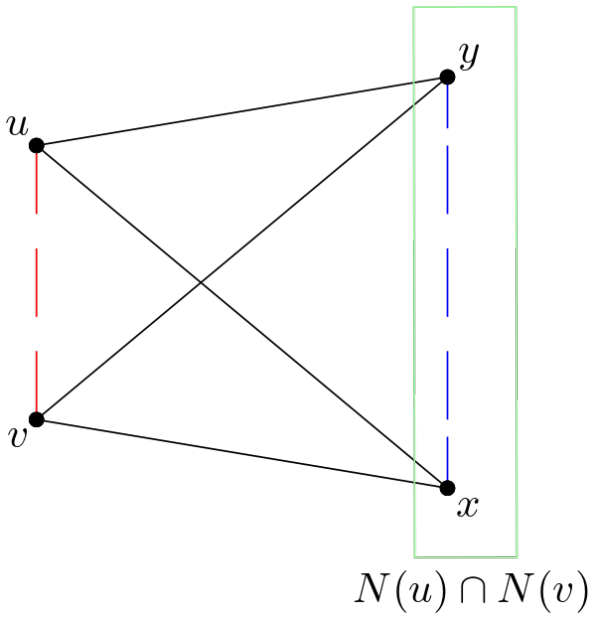


Figure 7.2.: If the blue edge is in the graph, the red edge closes a K_4.

Algorithm 18: Check whether an edge closes a K_4 in a graph G

Data: Edge $e = (u, v) \notin E(G)$
Result: `true` if e closes a K_4, `false` if not

```
if |N(u) ∩ N(v)| < 2 then
    return false;
else
    for unordered pair (x, y) ∈ N(u) ∩ N(v) do
        if edges[(x, y)] then
            return true;
    return false;
```

edge density p. Hence, we can compute, analagously as for the triangle-free and C_4-free process, that

$$\mathbb{E}[|O(i)|] = \binom{n}{2}(1-p^5)^{\binom{n-2}{2}} \approx \binom{n}{2} e^{-p^5 \frac{n^2}{2}}$$

in a $G(n,p)$. Using $p \approx 2i/n^2$ in the K_4-free process we get

$$\mathbb{P}[\text{open edge gets chosen}] = \frac{|O(i)|}{n^2} \approx \frac{1}{2} e^{-p^5 n^2/2} \approx \frac{1}{2} e^{-16\frac{i^5}{n^8}}.$$

This shows that the rejection approach is not suitable if the edge density gets sufficiently small. Note that we know from Section 7.1 that

$$e(G_{\boxtimes,n}) = \Theta(n^{8/5}(\log n)^{1/5})$$

with high probability. Thus we know that the edge density at the end of the process gets abritrarily small if n gets sufficiently large. Hence, we have to think about other approaches again and we follow the ideas for the triangle-free and the C_4-free process. This means that after a sufficient amount of rejections in a row in Algorithm 17 we generate a list of all still open edges in the current graph by checking for every edge that is not in the graph the condition that we just described. Afterwards, we iterate over the resulting list of edges in a uniform order and check whether we can add the according edge to the graph. Note that we have to check upon each insertion whether the edge is still open since we do not update the list of open edges after every insertion. This is shown in detail in Algorithm 12 which can be applied analogously to the K_4-free process using Algorithm 18.

7.2.1. Intuition for the constant

Now we want to present a similar conjecture as Conjecture 5.2.1 for the K_4-free process. Using our $G(n,p)$ intuition we have seen that

$$\mathbb{E}[|O(i)|] = \binom{n}{2}(1-p^5)^{\binom{n-2}{2}} \approx \binom{n}{2}e^{-p^5\frac{n^2}{2}}.$$

Using $t = m/n^{8/5}$ and $p \approx 2m/n^2$ we can use now the same intuition as for the C_4-free process. This means we want to follow the process up to an edge density of roughly $n^{-2/5}$ with the same arguments that we already presented in Section 2.3 and Section 5.2.1. This means we want to solve

$$e^{-16t^5}\binom{n}{2} = n^{8/5}.$$

Solving this with some estimates yields

$$m \approx \frac{1}{\sqrt[5]{40}}n^{8/5}(\log n)^{1/5}$$

and hence we conjecture the following.

Sample mean	62 745,97
Sample standard deviation	61,13
Relative sample standard deviation	0,10
Sample variance	3 736,55

Table 7.1.: Some statistical results for the K_4-free process on 1 000 vertices with 200 repetitions.

Conjecture 7.2.1. With high probability we have

$$e(G_{\boxtimes,n}) = \left(\frac{1}{\sqrt[5]{40}} + o(1)\right) n^{8/5} (\log n)^{1/5}$$

for the K_4-free process.

7.2.2. Experimental Evaluation

In this section, we present our results for the simulation of the K_4-free process with the goal to gain information about $e(G_{\boxtimes,n})$. Thereby, we proceed in the same way as for the triangle-free process and the C_4-free process. We also only use the combined algorithm of rejections and and the scanning of an open edge list like it is described in Algorithm 12. However, we already saw in Algorithm 18 that the condition we have to check before we can decide whether an edge is open or not is much more complicated than this is the case for the triangle-free process and the C_4-free process. Hence, we could not simulate the K_4-free process with the same number of vertices. In fact, we are only able to simulate this process for up to 5 000 vertices within a reasonable time.

Statistical results. We start with some statistical properties for the K_4-free process. For this, we repeat the K_4-free process on 1000 vertices 200 times. The results for this are shown in Table 7.2.2. Note that the sample standard deviation is slightly larger then for the triangle-free process but we also used a smaller number of vertices due to time reasons and we expect a larger deviation for a smaller number of vertices. However, we can see in Figure 7.3 that the standard deviation is still very low across all simulated results which forms a good basis for our further analysis.

Influence of the parameter α. Now we want to analyze the influence of the number of rejections α we allow in a row during the simulation of the K_4-free process. This is shown in Table 7.2.2. What we can see in this table is that we are able to reduce the number of open edges that we have to check after the rejection phase significantly by increasing α. However, in contrast to the C_4-free process we are not able to use this

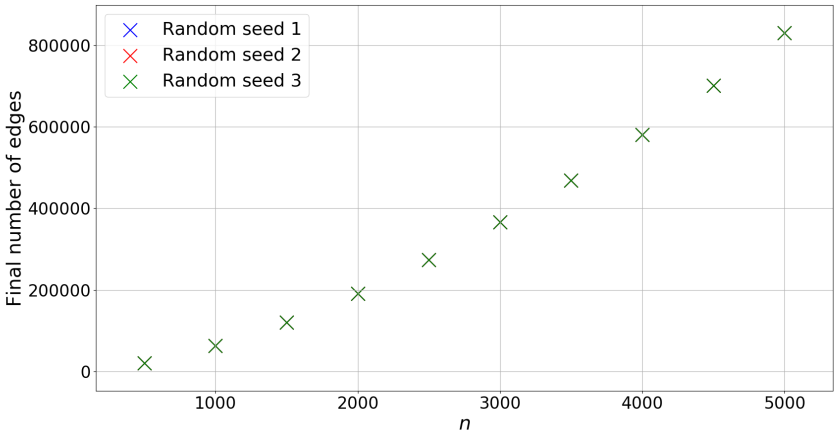


Figure 7.3.: For each number of vertices we plot three data points representing different random seeds and different values for the parameter α.

advantage with respect to the time performance of the algorithm. This is probably due to the low probability for an edge to be open during the rejection process in combination with the much more complex condition that we have to check for every edge under consideration that we describe in Algorithm 18. Further, we see that we can not use the lower number of edges after the rejection process for the maximum memory usage of our algorithm.

Models for the simulation data. In this part of the section, we analyze again different models for the simulation data which we want to analyze with respect to Conjecture 7.2.1. Note that we only know the exponents within the leading term of the final number of edges

$$e(G_{\boxtimes,n}) = \Theta(n^{8/5}(\log n)^{1/5})$$

with high probability. Hence, we define again

$$a^* = \frac{1}{\sqrt[5]{40}}, \quad b^* = \frac{8}{5}, \quad c^* = \frac{1}{5}$$

and we analyze our different models with respect to these parameters. First we look at the model

$$f_1(n) = a \cdot n^b \cdot (\log n)^c.$$

This can be seen in Figure 7.4. For this model we get the values

$$a = 0.69, \quad b = 1.56, \quad c = 0.32$$

n	$\alpha = 500$			$\alpha = 2\,500$			$\alpha = 5\,000$		
	$\lvert O(\alpha)\rvert$	t[s]	m[KB]	$\lvert O(\alpha)\rvert$	t[s]	m[KB]	$\lvert O(\alpha)\rvert$	t[s]	m[KB]
500	1 245	12,97	20 518	292	20,67	12 660	96	27,17	12 744
1 000	7 284	229,389	35 048	923	426,317	35 388	414	518,218	35 016
1 500	15 145	870,237	71 240	2 799	1 382,27	70 852	1 631	1 678,68	70 912
2 000	37 805	2 797,19	124 996	5 250	4 606,77	124 276	2 509	5 634,32	124 440
2 500	68 609	3 968,31	215 560	9 386	6 317,16	215 452	4 375	8 071,21	215 764
3 000	71 337	9 219,0	267 372	11 226	14 602,1	266 528	5 202	18 127,2	266 376
3 500	98 391	15 779,1	430 576	21 781	22 048,5	430 392	6 399	31 428,0	430 648
4 000	144 872	31 202,7	485 832	25 407	44 659,5	484 364	11 794	56 451,3	484 144
4 500	150 967	40 665,0	586 456	-	-	-	-	-	-
5 000	243 135	45 685,7	871 256	-	-	-	-	-	-

Table 7.2.: Influence of the rejection parameter α on the memory consumption and the overall time for the simulation of the K_4-free process on different numbers of vertices n. $|O(\alpha)|$ is the number of open edges after the rejection process.

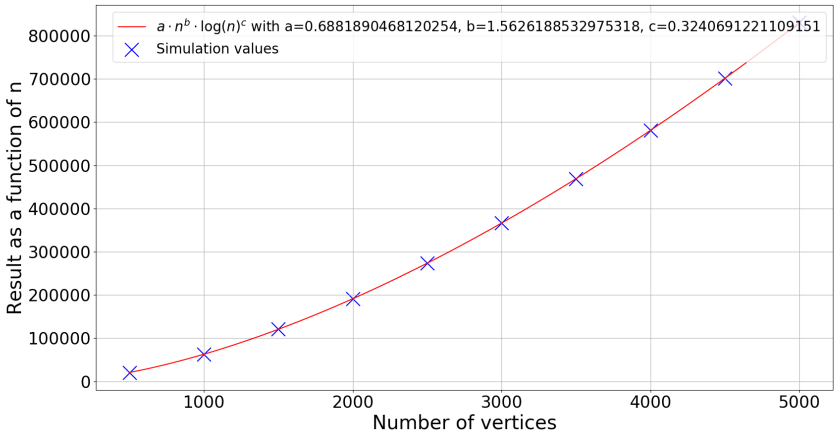


Figure 7.4.: Simulation values together with the predicted model $f_1(n)$.

and a standard error of

$$S = 69.46.$$

For this model, we see that we only get close to the prediction (a^*, b^*, c^*) for the parameter b. This can be due to effects of a second order term for example. However, since we already know the exponents b^* and c^* from the theory, we are mainly interested in the model value for a^*. This means we look at the model

$$f_2(n) = a \cdot n^{8/5} (\log n)^{1/5}$$

ignoring the possibility of a second order term for the moment. This is shown in Figure 7.5. The result for this model is

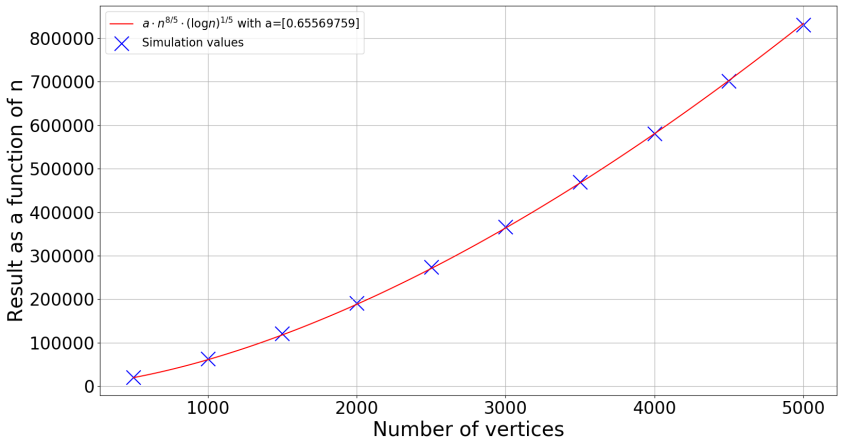


Figure 7.5.: Simulation values together with the predicted model $f_2(n)$.

$$a = 0.66$$

with an standard error of the regression of

$$S = 2767.04$$

which is much worse as for the model $f_1(n)$. In particular, we are also not within a reasonable range of our prediction a^*. To see whether this based on the influence of a second order term, we finally look at the model

$$f_3(n) = a \cdot n^{8/5}(\log n)^{1/5} + d \cdot n^{8/5}$$

since this also provided a good fit for the triangle-free process and the C_4-free process. This can be seen in Figure 7.6. For this model we get the result

$$a = 0.07, \quad b = 0.89$$

with a standard error of

$$S = 164.54$$

which is much better than for the model $f_2(n)$. This indicates that there is indeed a significant influence of a second order term which makes obviously the prediction for the constant factor a^* hard. In particular, even though this model gives us a good fit of the data we are far of from our prediction for the constant factor. However, due to the good fit of the data we do not believe that this is caused by the less data points but

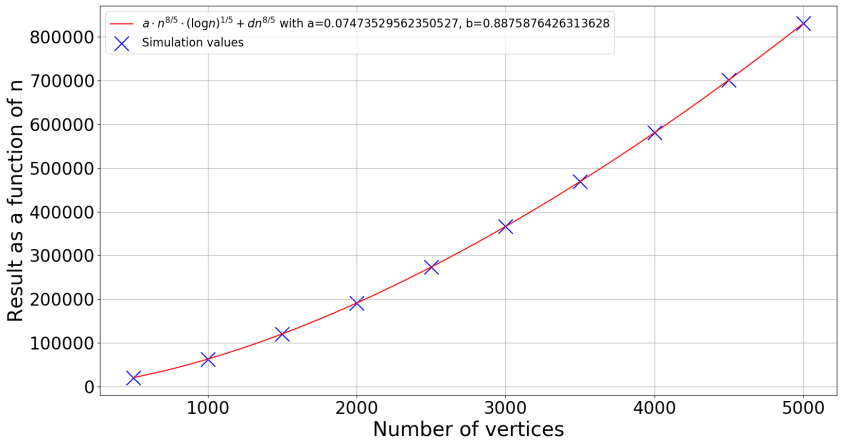


Figure 7.6.: Simulation values together with the predicted model $f_3(n)$.

rather by a significant influence of a second order term.

8. The K_4-removal Process

In this chapter, we present our results for the K_4-removal process. Similar as for the C_4-removal process to the best of our knowledge it is not known when the K_4-removal process ends in terms of the number of edges in the final graph. Tian et al. [Tian et al., 2022] showed that with high probability the number of edges in the final graph is bounded by $n^{19/20+o(1)}$. However, our experiments indicate that this bound is might not tight. We present this in detail in Section 8.1.1. We start with the usual definition of the K_4-removal process.

Definition 8.0.1. Let $(G_i)_{i\in\mathbb{N}_0}$ be the random graph process with $G_0 = K_n$ and G_i is obtained by G_{i-1} by selecting one copy of a K_4 in G_{i-1} uniformly at random and deleting its edges. The obtained random graph process is called the K_4-removal process.

Note again that this process becomes stationary in step

$$\tau_0 := \min_{j\in\mathbb{N}_0}\{G_j \in (G_i)_{i\in\mathbb{N}_0} \text{ is } K_4\text{-free}\}$$

since $E(K_n) < \infty$. We denote the graph G_{τ_0} as $G^r_{\boxtimes,n}$ and refer to it as the final graph of the K_4-removal process. Similar as for the C_4-removal process, we conjecture the following.

Conjecture 8.0.1. With high probability, we have

$$e(G^r_{\boxtimes,n}) = n^{8/5+o(1)}$$

for the number of edges in the final graph of the K_4-removal process.

The rest of this chapter is concerned with Conjecture 8.0.1. In Section 8.1, we present our algorithms for the simulation of the K_4-removal process. Thereby, we use similar techniques as for the triangle-removal and C_4-removal process and hence only describe the main ideas and the key differences. In Section 8.1.1, we present the results of the simulation algorithms and we interpret them with resprect to Conjecture 8.0.1.

8.1. Simulation of the K_4-removal process

In this section, we present our simulation algorithms for the K_4-removal process. The overall idea is here the same as for the other removal processes in this work. Hence, we will focus on the main differences in this section and for more details of the algorithms themself we refer to Section 4.2 and Section 6.1. Further, we also use the notation used in those sections. First,we have to analyze the behaviour of the rejection approach for the K_4-removal process. In order to do so, we use again our intuition that the graphs should be similar to $G(n,p)$ graphs with a corresponding edge density. This is confirmed by our experiments in Section 8.1.1. By definition of the K_4-removal process, the edge density of the graph G_i is given by

$$p = p(i) := \frac{\binom{n}{2} - 6i}{\binom{n}{2}} \approx 1 - \frac{12i}{n^2} \tag{8.1}$$

since we delete in every step of the process six edges and we start from a complete graph on n vertices. In our rejection process, we choose in every step four vertices uniformly at random and check whether they build a K_4 in the current graph. As we already explained, in the previous sections this is an equivalent approach to simulate the process. Using our $G(n,p)$ intuition this results in

$$\mathbb{P}[\text{chosen vertices build a } K_4] \approx p^6 \approx \left(1 - \frac{12i}{n^2}\right)^6.$$

However, we see here again that this approach becomes computationally inefficent fast. In particular, even faster than for the rest of the processes since more edges have to be in the graph. This behaviour can also be seen in Section 8.1.1. Hence, we now want to present the algorithm `k4List` which lists all copies of a K_4 in an arbitratry graph $G = (V, E)$ exactly once. If we have such a list, we can iterate over it a random order and delete its edges if it is still in the current graph. For details on this, see Section 4.2 for example. Now we present the algorithm `k4List`. First, we have to think about how we can achieve that our algorithm lists every copy of a K_4 exactly once. For this let $v_1^4 := \{v_1, \dots, v_4\} \subseteq V(G)$ represent a K_4 in the graph G. Then we can find an unique ordering of the vertices such that

$$v_1 < v_2 < v_3 < v_4. \tag{8.2}$$

The idea is now to iterate over every vertex of the graph G and for each $v \in V$ we list exactly those copies of a K_4 including v, where v has the role v_1 in Equation 8.2. Since

Algorithm 19: `k4List` : List all complete graphs on four vertices in a given graph $G = (V, E)$

```
Data: Graph G = (V, E) with n vertices
Result: List of all complete graphs on four vertices in G
1  T ← ∅;
2  adjList_> ← (n, ∅);
3  for v ∈ V do
4      for w ∈ N(v) do
5          if w > v then
6              adjList_>[v] ← adjList_>[v] ∪ {w};
7  for v ∈ V do
8      for w ∈ adjList_>[v] do
9          for x ∈ adjList_>[w] do
10             if (v, x) ∈ E then
11                 for y ∈ adjList_>[x] do
12                     if (v, y) ∈ E then
13                         Edge e ← (w, y);
14                         if e ∈ E then
15                             K_4 k ← (v, w, x, y);
16                             T ← T ∪ {k};
```

this ordering is unique for every copy of a K_4 in the graph, this guarantees that we list every K_4 only once. In order to use this observation, we introduce a data structure $\texttt{adjList}_>$ which stores for every vertex $v \in V$ a list of all neighbours w of v with $w > v$. Since we list no K_4 starting at v wich contains a smaller vertex, this is all information that we need. Now the algorithm works as follows. For every vertex $v \in V$ it looks at every path (v, w, x, y) of length three with $v < w < x < y$ using $\texttt{adjList}_>$. This can simply be done by iterating over the according lists. Note that we do not even have to check whether the vertices are actually greater because the data structure takes care of this by definition. But we only further look at the path if in every step the edge to vertex v is present. Afterwards, we check the missing edge for a K_4, namely (w, y). If this edge is present, we found a K_4. This is shown in Algorithm 19.

Lemma 8.1.1. Let $G = (V, E)$ be an arbitrary graph on n vertices. Then Algorithm 19 lists every K_4 in G exactly once.

Proof. Let $v_1^4 \subseteq G$ be an arbitrary copy of a K_4. Without loss of generality let

$v_1 < v_2 < v_3 < v_4$. Hence, we have

$$v_2, v_3, v_4 \in \texttt{adjList}_>[v_1],\ v_3, v_4 \in \texttt{adjList}_>[v_2],\ v_4 \in \texttt{adjList}_>[v_3].$$

Further, we have $(v_1, v_2), (v_1, v_3), (v_1, v_4) \in E$ and thus Algorithm 19 reaches line 13 with v_1, v_2, v_3, v_4 under consideration. But since we also have $(v_2, v_4) \in E$ we find v_1^4 when we explore the vertex v_1. Since the K_4 was arbitrary, it follows that we find every copy of a K_4 at least once. Now we have to show that we do not list any K_4 twice. With the notation above, assume we find v_1^4 again while looking at the K_4s starting from v_1. But this means that we have another ordering in our path and hence there is $i! = j \in \{2, 3, 4\}$ such that $v_i \geq v_j$ but v_i comes before v_j in the path. But this is a contradiction since $v_j \notin \texttt{adjList}_>[v_i]$ and hence this path can not be found. Now assume that we find v_1^4 starting from v_i with $i \in \{2, 3, 4\}$. Note that we can find a K_4 only when we start from one of the vertices which are part of the corresponding K_4. But by definition of the algorithm, we can only find a K_4 if we find a path of length three with four ascending vertices. But since $v_1 < v_i$ this can not be the case for v_1^4 and hence we also do not list v_1^4 when we examine another vertex than v_1 and the lemma follows. □

With the correctness of the algorithm `k4List` we can now define analogous algorithms to Algorithm 15 and Algorithm 16 by just transferring the algorithms to the situation of a K_4 and using the algorithm `k4List` instead of `c4List` and the explained ideas.

8.1.1. Experimental Evaluation

In this section, we present the results of our simulation for the last process. Thereby, we can observe a similar behaviour as for the C_4-removal process regarding running time and maximal memory consumption for the simulation of the original process. In addition, we find evidence that suggests that Conjecture 8.0.1 is true. The rest of this section is structured as follows. As usual, we start with some statistical results for the simulation of the original data in order to assess the reliabilty of our simulation results. Afterwards, we compare the performance of the original version of the K_4-removal process with the heuristic algorithm that we described in the previous section. Finally, we apply different models to our data to analyze Conjecture 8.0.1 and to evaluate the quality of the heuristic algorithm.

Statistical results. For the statistical results for the simulation of the random variable $e(G^r_{\boxtimes,n})$ we get similar results as for all the other processes. This can be seen in Ta-

Sample mean	29 568,06
Sample standard deviation	131,91
Relative sample standard deviation	0,45
Sample variance	17 400,84

Table 8.1.: Some statistical results for the K_4-removal process on 1 000 vertices with 200 repetitions.

n	$\alpha = 10\,000$	$\alpha = 15\,000$	$\alpha = 20\,000$	$\alpha = 100\,000$	$\alpha = 150\,000$	$\alpha = 200\,000$
500	47 132	47 156	30 276	16 196	15 872	15 988
1 000	565 488	565 700	302 908	105 528	72 772	72 856
1 500	4 276 736	2 179 408	1 130 924	344 092	343 860	212 584
2 000	8 538 588	8 538 356	4 342 396	1 193 424	1 193 204	1 193 220
2 500	33 801 752	17 023 780	17 023 636	4 438 192	2 337 064	-
3 000	67 424 552	33 869 676	33 554 480	-	-	-

Table 8.2.: Influence of the rejection parameter α on the maximal memory consumption for the simulation of the K_4-removal process on different numbers of vertices n.

ble 8.1.1 where we simulated the process on 1 000 vertices 200 times. Even though the standard deviation is larger than for the K_4-free process, it is with a sample standard deviation of less than one percent still very low. This means that we get a reliable basis for our data and this gives our data more validity. We will also see this property when we fit different models to our data.

Performance of the algorithms. For the simulation algorithm, we can observe a similar behaviour as for the C_4-removal process. This is shown in Table 8.1.1 and Table 8.1.1 where we look at the time and memory performance of the original simulation algorithm as described in the previous section for different random seeds and rejection parameters α. Note again that the solution quality is not influenced by this choice. The main difference in comparison to the C_4-removal process, which is also reflected in the results, is that we expect fewer copies of a K_4 in a $G(n, p)$ than copies of a C_4. This results in less memory consumption and even less running time for the simulation running with similar parameters like $\alpha = 15\,000$. This is even though the listing of all copies of a K_4 seems more complicated than listing all copies of a C_4. However, we see that this behaviour is limited. For a significant increase of the rejection parameter we can not efficiently lower the memory consumption. This is probably due to the low probability for a random copy of a K_4. We have already analyzed this in the previous section. However, this leads to a significant increase in running time. In particular, at some point we are not able to simulate the process with more vertices since this either

n	$\alpha = 10\,000$	$\alpha = 15\,000$	$\alpha = 20\,000$	$\alpha = 100\,000$	$\alpha = 150\,000$	$\alpha = 200\,000$
500	14,98	12,96	20,22	68,85	82,33	96,55
1 000	283,32	280,55	370,65	1 373,06	1 497,36	1 912,75
1 500	1 107,07	1 198,6	1 374,52	4 220,94	4 709,29	6 729,0
2 000	3 855,8	4 042,0	4 789,0	11 456,3	14 700,1	15 352,3
2 500	8 533,34	7 769,69	7 703,37	16 658,5	25 334,2	-
3 000	20 658,4	16 919,8	16 522,0	-	-	-

Table 8.3.: Influence of the rejection parameter α on the overall time for the simulation of the K_4-removal process on different numbers of vertices n.

n	ϵ_1	t[s]	m[KB]	Result	ϵ_2	t[s]	m[KB]	Result
500	0,52	39,86	1 062 244	9 909	0,54	47,93	1 060 888	9 778
1 000	0,32	94,38	1 065 228	29 631	0,34	128,35	1 066 024	29 420
1 500	0,30	493,09	4 218 844	56 298	0,32	685,35	4 219 588	56 302
2 000	0,27	1 194,32	4 231 044	89 068	0,29	1 717,87	8 427 232	88 766
2 500	0,22	1 289,77	4 238 384	126 843	0,24	1 995,04	8 435 620	126 774
3 000	0,22	3 157,26	8 446 948	169 487	0,24	4 902,85	16 838 544	169 446
3 500	0,17	1 860,36	4 256 056	216 099	0,19	3 274,0	8 455 080	215 914
4 000	0,17	3 627,66	4 270 448	267 567	0,19	6 338,48	8 470 808	267 773
4 500	0,16	4 810,98	8 476 536	321 635	0,17	6 523,77	8 480 092	322 760
5 000	0,16	8 118,02	8 492 248	380 794	0,17	11 011,8	16 885 060	382 010
6 000	0,12	4 792,35	4 308 584	509 017	0,15	14 423,9	16 910 228	508 918
7 000	0,12	10 167,9	8 535 224	649 806	0,15	29 332,2	33 749 736	650 437

Table 8.4.: Results for the heuristic simulation of the K_4-removal process for the two largest edge densities.

uses to much memory or takes to much time. However, we can conclude that a careful choice of this tuning parameter leads to a better performance of the overall algorithm. To overcome this problem, we look again at the heuristic approach to simulating the process, where we replace the rejection phase of the simulation with the generation of a $G(n, p)$ where p is suitably chosen. This is shown in Table 8.1.1. However, we encounter the problem that this approach does not work as well as for the C_4-removal process. What we could observe during the experiments is that we have to orientate ourselfs on a similar value for p as we describe in Section 2.3. This value is for the C_4-removal process simply lower than for the K_4-removal process. Hence, we can see that we need significantly more maximal memory consumption than for a similar number of vertices for the C_4-removal process until the results of the process converge. However, we are still able to simulate the process with a larger number of vertices and convergent results. This is further underlined when we look at the models for our data.

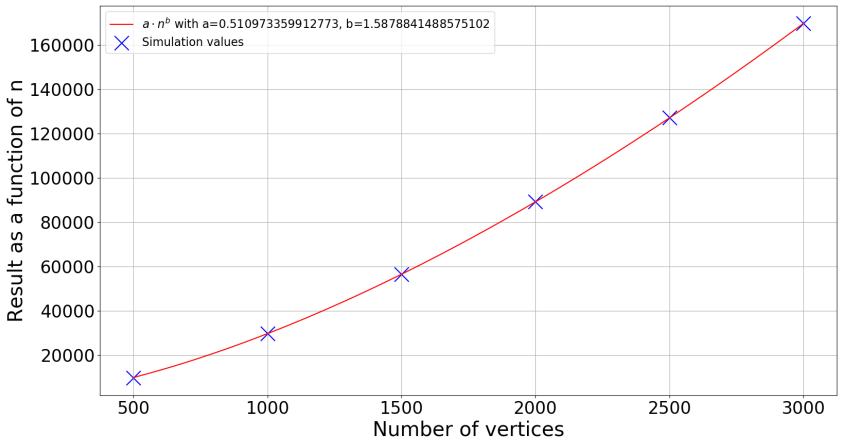


Figure 8.1.: Simulation results together with the predicted model $f_1(n)$.

Models for the simulation data. Finally, we want to examine the simulation data by fitting different models to it to analyze Conjecture 8.0.1. For this, we will use the original data as well as the data we obtain from the heuristic algorithm. First, we look at the model

$$f_1(n) := a \cdot n^b \tag{8.3}$$

to get an idea for the main factor in the leading term of $e(G^r_{\boxtimes,n})$. For the original data, we get for this model

$$a = 0.51, \quad b = 1.59, \quad S = 80.00, \tag{8.4}$$

where S is the standard error of the regression. That this model fits the data well can also be seen in Figure 8.1. Furthermore, we can see that the predicted exponent is close the exponent we suggested in Conjecture 8.0.1. If we apply this model to our heuristic data, we can see that we get similar results

$$a_h = 0.51, \quad b = 1.59, \quad S = 191.91. \tag{8.5}$$

Even though the standard error is larger, we can see that this is a good fit of the data in Figure 8.2. Further, this indicates that the heuristic approach is suitable for the simulation of the K_4-removal process since we already saw that in addition the results seem to converge. Finally, we want to examine the data with respect to the possibility of a logarithmic factor in the leading term of $e(G^r_{\boxtimes,n})$. For this, we look at the model

$$f_2(n) = a \cdot n^{8/5} \cdot (\log n)^c \tag{8.6}$$

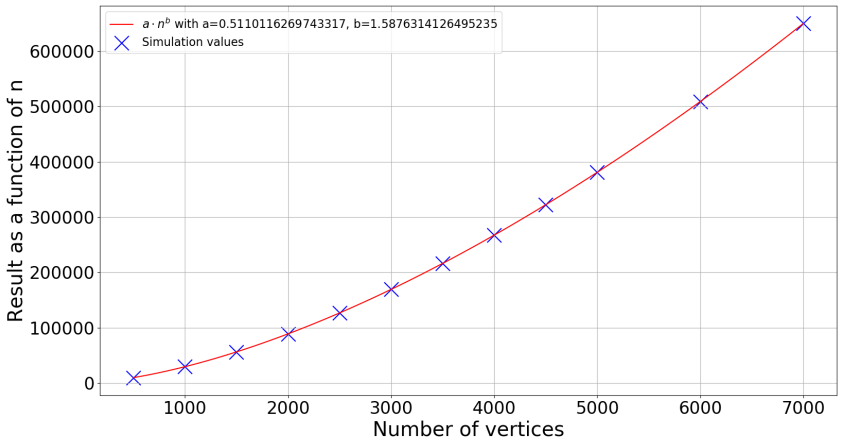


Figure 8.2.: Heuristic simulation results together with the predicted model $f_1(n)$.

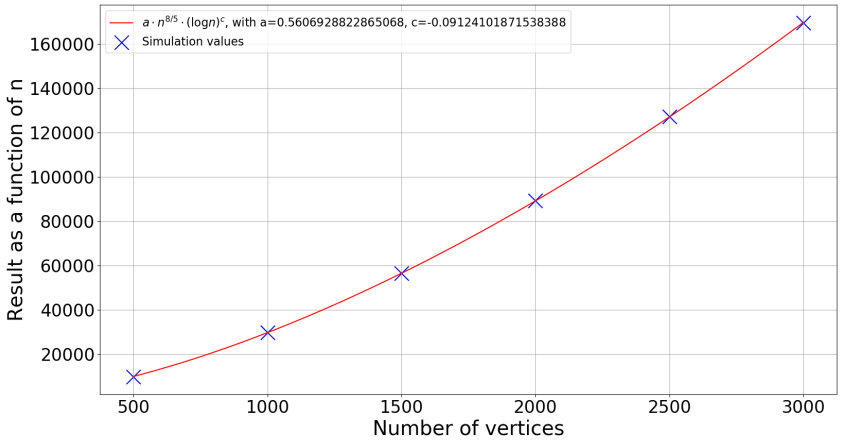


Figure 8.3.: Simulation results together with the predicted model $f_2(n)$.

assuming that we are already sure about the exponent 8/5. This yields for the original data

$$a = 0.56, \quad c = -0.09, \quad S = 89.79. \tag{8.7}$$

This can be seen in Figure 8.3 and we see that this model fits the data similarly well as the model $f_1(n)$. For the heuristic data, we get for this model the values

$$a = 0.57, \quad c = -0.10, \quad S = 203.93. \tag{8.8}$$

In conclusion, we can say that the heuristic approach is suitable for simulating the K_4-removal process and has significant advantages in terms of time and memory. Furthermore, we find evidence that confirms Conjecture 8.0.1. In addition, the data suggest

that there may not even be a logarithmic factor in the leading term of $e(G^r_{\boxtimes,n})$. Finally, the results give a strong indication that the bound that was shown in [Tian et al., 2022] is not sharp.

9. Discussion

9.1. Conclusion

In this work, we gave an extensive introduction to the theory of several random graph processes. Additionally, we developed simulation algorithms for the H-free process and the H-removal process, where $H \in \{K_3, C_4, K_4\}$, to simulate the final number of edges in the generated graph. For every process we cover in this work, we gave an algorithm which represents the original process using pseudo-randomness. Further, we gave heuristic algorithms for the H-removal process in all three cases which proved to be suitable for simulating these processes. Furthermore, they had significant advantages compared to the original processes regarding time and memory usage. Further, our data was able to confirm results for the triangle-free process that are already known from theory. This gives a strong indication that our predicted results for the C_4-free process might be true. For the K_4-free process the data did not provide similar results, as the simulation of this process was harder. For the triangle-removal process, we were able to confirm the exponent of $3/2$ that we already knew from theory and our data suggests that there might not be a logarithmic factor. This gives also a strong indication that the triangle-free and the triangle-removal process are not equivalent in any sense. For the case of $H = C_4$ and $H = K_4$, where no sharp bounds are know to the best of our knowledge, we gave strong indications for the final number of edges which are based on the precision of our predictions for the triangle-removal process and the low standard deviation in our simulations. In particular, for $H = K_4$ our results gave a strong indication that the bound in [Tian et al., 2022] is not sharp. In conclusion, we hope that these results confirm intuitions of researches in this field. Further, to the best of our knowledge this is the first work that handles these type of random graph processes with numerical simulations and we hope that this provides a value for theoreticians in this field.

9.2. Future Work

For the future, we want to further improve our simulation algorithms. Especially for the H-removal process, often the memory consumption of our approach was a bottleneck.

Since our heuristic proved to be suitable for the simulation of this process, we think that we can further refine this approach to reduce running time and memory consumption. Additionally, we would like to measure even more properties during the process to provide researchers maybe with an intuition for other parameters that need to be tracked for a proof. Moreover, we also would like to analyze the processes with even more sophisticated statistical techniques to give more reliable results. Further, we would like to apply this technique to other random graph processes as well.

A. Experiments on the independence number

Here we give some experiments on the the size of an independent set in graphs obtained by the triangle-free process (see Table A) and graphs obtained by the C_4-free process (see Table A). The results have been computed using the KaMIS repository (see `https://github.com/KarlsruheMIS/KaMIS`) with the `redumis` solver [Lamm et al., 2017]. Note that the results are not exact. Further, we give in the tables the leading term of the bounds shown by theory. For the triangle-free case this bound can be found in [Fiz Pontiveros et al., 2020], Theorem 2.12 and for the C_4-free case this is given in [Picollelli, 2010], Corollary 1.3. We repeated the experiments twice using different random seeds.

Vertices	Solution Size	$\sqrt{2} \cdot \sqrt{n \log n}$
100	24	30
500	65	79
1 000	96	118
1 500	120	148
2 000	140	174
2 500	158	198
3 000	173	219
3 500	188	239
4 000	201	258
4 500	216	275
5 000	230	292
5 500	241	308
6 000	251	323
6 500	263	338
7 000	272	352

Table A.1.: Experiments on the size of an independent set in graphs obtained by the triangle-free process.

Vertices	Solution Size	$(n \cdot \log(n))^{2/3}$
100	31	60
500	113	213
1 000	199	363
1 500	272	494
2 000	340	614
2 500	401	726
3 000	467	832
3 500	520	934
4 000	576	1 032
4 500	633	1 127
5 000	685	1 219
5 500	736	1 309
6 000	786	1 397
6 500	834	1 482
7 000	885	1 566

Table A.2.: Experiments on the size of an independent set in graphs obtained by the C_4-free process.

Bibliography

[Ajtai et al., 1980] Ajtai, M., Komlós, J., and Szemerédi, E. (1980). A note on ramsey numbers. *Journal of Combinatorial Theory, Series A*, 29(3):354–360.

[Ajtai et al., 1981] Ajtai, M., Komlós, J., and Szemerédi, E. (1981). A dense infinite sidon sequence. *European Journal of Combinatorics*, 2(1):1–11.

[Alon and Spencer, 2016] Alon, N. and Spencer, J. H. (2016). *The Probabilistic Method*. Wiley Publishing, 4th edition.

[Barabási and Albert, 1999] Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

[Batagelj and Mrvar, 2001] Batagelj, V. and Mrvar, A. (2001). A subquadratic triad census algorithm for large sparse networks with small maximum degree. *Soc. Networks*, 23(3):237–243.

[Beveridge et al., 2007] Beveridge, A., Bohman, T., Frieze, A., and Pikhurko, O. (2007). Product rule wins a competitive game. *Proceedings of The American Mathematical Society - PROC AMER MATH SOC*, 135:3061–3072.

[Bohman, 2009] Bohman, T. (2009). The triangle-free process. *Advances in Mathematics*, 221(5):1653–1677.

[Bohman et al., 2010] Bohman, T., Frieze, A., and Lubetzky, E. (2010). A note on the random greedy triangle-packing algorithm. *arXiv preprint arXiv:1004.2418*.

[Bohman et al., 2015] Bohman, T., Frieze, A., and Lubetzky, E. (2015). Random triangle removal. *Advances in Mathematics*, 280:379–438.

[Bohman et al., 2011] Bohman, T., Frieze, A. M., and Lubetzky, E. (2011). Random greedy triangle-packing beyond the 7/4 barrier. *CoRR*, abs/1108.1781.

[Bohman and Keevash, 2009] Bohman, T. and Keevash, P. (2009). The early evolution of the h-free process. *Inventiones Mathematicae*, 181.

[Bollobás, 1997] Bollobás, B. (1997). *Paul Erdős — Life and Work*, pages 1–41. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bollobás and Riordan, 2000] Bollobás, B. and Riordan, O. (2000). Constrained graph processes. *Electron. J. Comb.*, 7.

[Erdös, 1961] Erdös, P. (1961). Graph theory and probability, ii. *Canadian Journal of Mathematics*, 13.

[Erdös et al., 1960] Erdös, P., Rényi, A., et al. (1960). On the evolution of random graphs. *Publ. math. inst. hung. acad. sci*, 5(1):17–60.

[Erdös et al., 1995] Erdös, P., Suen, S., and Winkler, P. (1995). On the size of a random maximal graph. *Random Struct. Algorithms*, 6:309–318.

[Erdös and Szckeres, 1987] Erdös, P. and Szckeres, G. (1987). *A Combinatorial Problem in Geometry*, pages 49–56. Birkhäuser Boston, Boston, MA.

[Fiz Pontiveros et al., 2020] Fiz Pontiveros, G., Griffiths, S., and Morris, R. (2020). The triangle-free process and the ramsey number $r(3, k)$. *Memoirs of the American Mathematical Society*, 263.

[Funke et al., 2018] Funke, D., Lamm, S., Sanders, P., Schulz, C., Strash, D., and von Looz, M. (2018). Communication-free massively distributed graph generation. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21 – May 25, 2018*.

[Gordon et al., 1996] Gordon, D. M., Patashnik, O., Kuperberg, G., and Spencer, J. H. (1996). Asymptotically optimal covering designs. *Journal of Combinatorial Theory, Series A*, 75(2):270–280.

[Grable, 1997] Grable, D. A. (1997). On random greedy triangle packing. *the electronic journal of combinatorics*, pages R11–R11.

[Kim, 1995] Kim, J. H. (1995). The ramsey number r(3, t) has order of magnitude t2/log t. *Random Struct. Algorithms*, 7:173–208.

[Lamm et al., 2017] Lamm, S., Sanders, P., Schulz, C., Strash, D., and Werneck, R. F. (2017). Finding near-optimal independent sets at scale. *J. Heuristics*, 23(4):207–229.

[Osthus and Taraz, 2001] Osthus, D. and Taraz, A. (2001). Random maximal h-free graphs. *Random Struct. Algorithms*, 18:61–82.

[Picollelli, 2010] Picollelli, M. (2010). The final size of the c_4-free process. *Combinatorics Probability and Computing*, 20.

[Ramsey, 1987] Ramsey, F. P. (1987). *On a Problem of Formal Logic*, pages 1–24. Birkhäuser Boston, Boston, MA.

[Rödl and Thoma, 1996] Rödl, V. and Thoma, L. (1996). Asymptotic packing and the random greedy algorithm. *Random Struct. Algorithms*, 8:161–177.

[Sanders and Schulz, 2016] Sanders, P. and Schulz, C. (2016). Scalable generation of scale free graphs. *Inf. Process. Lett.*, 116(7):489–491.

[Schank and Wagner, 2005] Schank, T. and Wagner, D. (2005). Finding, counting and listing all triangles in large graphs, an experimental study. In Nikoletseas, S. E., editor, *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, volume 3503 of *Lecture Notes in Computer Science*, pages 606–609. Springer.

[Shearer, 1983] Shearer, J. B. (1983). A note on the independence number of triangle-free graphs. *Discrete Mathematics*, 46(1):83–87.

[Spencer, 1995] Spencer, J. H. (1995). Asymptotic packing via a branching process. *Random Struct. Algorithms*, 7:167–172.

[Spiess and Neumeyer, 2010] Spiess, A. N. and Neumeyer, N. (2010). An evaluation of r2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a monte carlo approach. *BMC Pharmacology*, 10:6.

[Tian et al., 2022] Tian, F., Liu, Z.-L., and Pan, X.-F. (2022). Random k_k-removal algorithm.

[Warnke, 2014] Warnke, L. (2014). When does the k4-free process stop? *Random Structures & Algorithms*, 44(3):355–397.