Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Zrandomizowane algorytmy eksploracji grafów

Jacek Mucha
Nr indeksu: 186033

Praca inżynierska napisana
pod kierunkiem
dr. hab. inż. Marka Klonowskiego

Politechnika
Wrocławska

Wrocław 2017

Spis treści

Wstęp

Od czasów odkrycia fal radiowych i głębszego zrozumienia analizy fourierowskiej, biały szum niepokoił zarówno matematyków, jak i inżynierów. Ostateczna granica wszystkich procesów fizycznych, którą astrofizycy poetycko nazwali „śmiercią cieplną Wszechświata", stan absolutnej martwoty informacyjnej, wydaje się idealną kryjówką dla tych, którzy zamierzają ukryć wiadomość. Celem stało się spreparowanie przekazu w taki sposób, aby przypominał biały szum.

Rozpatrzmy następujący problem. Przypuśćmy, że mamy system może znajdować się w określonej liczbie stanów, przy czym po każdym kroku system zmienia swój stan z określonym prawdopodobieństwem. Chcemy wiedzieć, ile kroków potrzeba wykonać, aby potencjalny obserwator śledzący nasz system nie był w stanie nic powiedzieć o stanie początkowym systemu. Modelem dla takiej sytuacji są łańcuchy Markowa, a pożądanym stanem dezinformacji - rozkłady stacjonarne łańcuchów Markowa.

Oczywiście rozkłady stacjonarne to rozkłady graniczne, w praktyce nie da się wykonać nieskończonej liczby kroków. Konieczne staje się wprowadzenie jakiejś miary odległości rozkładu procesu stochastycznego po danej liczbie kroków od rozkładu stacjonarnego oraz oszacowanie tempa zbieżności procesu o danej macierzy przejścia do rozkładu stacjonarnego. Różne podejścia są możliwe - w niniejszej pracy badać będziemy tzw. odległość wahania całkowitego. Na jej podstawie określimy tzw. czas mieszania, a więc moment, w którym rozkład procesu będzie odpowiednio „bliski" rozkładu stacjonarnego. Szczegóły techniczne zostaną omówione we wstępie teoretycznym, tutaj warto zaznaczyć, że rozważany czas mieszania jest zmienną losową. Co więcej, jest to moment stopu (czas Markowa). W literaturze znaleźć można liczne pomysłowe dolne oszacowania analityczne czasów mieszania (jako wielkości deterministycznych), badacze zgodnie jednak zaznaczają, że oszacowania te są niewystarczające i zbyt ogólne, by dało się je z powodzeniem zastosować do konkretnych łańcuchów Markowa. Podstawową literaturą była tutaj książka [**?**] oraz wykłady [**?**].

W niniejszej pracy zbadamy rozkłady czasów mieszania dla kilku konkretnych typów łańcuchów Markowa. Wydaje się, że ten temat dotychczas pomijano w badaniach, trudno wskazać w literaturze próby opisania takich rozkładów. Próbę analitycznego zbadania takich rozkładów można znaleźć w [**?**], ale tutaj definicja czasu mieszania różni się od rozważanej poniżej. Brak stosownych narzędzi analitycznych zmusza do ograniczenia metodologii do przeprowadzenia symulacji i analizy statystycznej. Mimo to uzyskane wyniki wydają się raczej niespodziewane i można mieć nadzieję, że zachęcą badaczy do dalszego zgłębiania tematu.

Czasy mieszania związane są z tzw. czasami pokrycia (*cover times*) oraz z problemem patrolowania grafu (*patrolling*). Czas pokrycia w łańcuchach Markowa to moment, w którym każdy ze stanów łańcucha został odwiedzony przynajmniej raz. Problem patrolowania w pewnym sensie rozszerza pojęcie czasu pokrycia: mając do dyspozycji n agentów poruszających się po grafie, żądamy, aby w kolejnych przedziałach czasu o jednakowej długości wszystkie wierzchołki grafu były regularnie odwiedzane przez agentów. Patrolowanie ma kluczowe znaczenie w systemach zabezpieczeń, w szczególności w sytuacjach, kiedy trzeba monitorować zasoby rozmieszczone w przestrzeni. Skuteczność naruszenia bezpieczeństwa danego zasobu jest tym większa, im dłużej żaden strażnik (agent) nie kontrolował tego zasobu. Z tego powodu poszukuje się protokołów patrolowania, które minimalizują okres czasu pomiędzy odwiedzinami wszystkich zasobów. Powyższy problem w literaturze pojawia się w licznych wariantach, rozważania na ten temat można znaleźć m.in. w [**?**], [**?**], [**?**], [**?**] oraz [**?**].

Problem patrolowania był jedną z motywacji dla badania czasów mieszania w łańcuchach Markowa. Próbę powiązania problemu patrolowania z czasami mieszania opisano w Paragrafie **??**.

Mimo, że praca ma charakter teoretyczny, to uzyskane wyniki znajdują zastosowania w teorii sieci społecznościowych. Symulacje komputerowe to domena bardziej inżynierów niż teoretyków, natomiast testy zgodności rozkładów to narzędzia bardzo popularne wśród prężnie rozwijającej się dziedziny informatyki, jaką jest *data science*.

W ramach pracy napisano symulator. Symuluje się łańcuchy Markowa i czasy mieszania. Estymuje się również wykres funkcji odległości wahania całkowitego rozkładu procesu od rozkładu stacjonarnego.

Praca ma następującą strukturę: w Rozdziale 2 przedstawiono sformułowanie rozważanych problemów, wprowadzenie teoretyczne i znane wyniki analityczne. Rozdział 3 poświęcony jest wynikom symulacji łańcuchów Markowa na klikach, grafach cyklicznych, tzw. archipelagach oraz na pewnych przykładach sieci bezskalowych. Omówienie implementacji symulatora znaleźć można z kolei w Rozdziale 4.

Analiza problemu - wstęp teoretyczny i znaczenie dla informatyki

W niniejszym rozdziale przedstawiono zarys teorii łańcuchów Markowa i czasów mieszania. W poniższym wywodzie wprowadzono definicje struktur matematycznych koniecznych do zrozumienia pojęcia czasu mieszania. Ponadto zaprezentowano znane z literatury wyniki analityczne oraz propozycje kierunku dalszych badań.

2.1 Elementy teorii łańcuchów Markowa

Definicje i fakty dotyczące ogólnej teorii łańcuchów Markowa zostały sformułowane na podstawie rozdziału 12 z książki J. Jakubowskiego i R. Sztencla [**?**]. Niech $\mathcal{S} = \{s_1,s_2,...,s_n\}$ będzie skończonym zbiorem (tzw. zbiorem stanów).

Definicja 2.1 *(Definicja 2, s.278 [**?**]) Łańcuchem Markowa nazywamy ciąg zmiennych losowych $(X_n)_{n=1}^{\infty}$ określonych na przestrzeni $(\Omega,\mathcal{F},P)$ o wartościach w $\mathcal{S}$ taki, że dla każdego $m \in \mathbb{N}$ i dowolnych $s_0,s_1,...,s_m \in \mathcal{S}$ zachodzi*

$$P(X_m = s_m|X_{m-1} = s_{m-1},...,X_1 = s_1,X_0 = s_0) = P(X_m = s_m|X_{m-1} = s_{m-1}). \tag{2.1}$$

Oznaczmy $p_{ij} = P(s_i,s_j) = P(X_{t+1} = s_j|X_t = s_i) = P(X_1 = s_j|X_0 = s_i)$ oraz $p_{ij}(t) = P(X_t = s_j|X_0 = s_i)$. P będziemy nazywać macierzą przejścia. Rozkład początkowy, tj. rozkład zmiennej X_0, będziemy oznaczać przez μ_0. Zauważmy, że rozkład zmiennej X_t, μ_t, można przedstawić jako

$$\mu_t = \mu_0 P^t. \tag{2.2}$$

Definicja 2.2 *(Definicja 1, s. 300, [**?**]) Rozkładem stacjonarnym dla łańcucha Markowa o macierzy przejścia P nazywamy wektor π taki, że $\pi = \pi P$.*

Definicja 2.3 *(s. 289, [**?**]) Łańcuch Markowa o macierzy przejścia P nazywamy nieprzywiedlnym, jeśli dla każdych dwóch stanów $s_i, s_j \in \Omega$ istnieje $t > 0$ takie, że $P^t(s_i,s_j) > 0$.*

Definicja 2.4 *(Definicja 1, s. 299, [**?**]) Okresem stanu $s_j \in \mathcal{S}$ nazywamy liczbę $o(j) = NWD\{n : p_{jj}(n) > 0\}$. Łańcuch P nazywamy nieokresowym, gdy dla każdego $s_j \in \mathcal{S}$ zachodzi $o(j) = 1$.*

Twierdzenie 2.1 *(Twierdzenie 9, s. 306, [**?**]) Jeśli łańcuch Markowa o skończonej liczbie stanów i macierzy przejścia P jest nieokresowy i nieprzywiedlny, to istnieje jedyny rozkład stacjonarny π taki, że dla dowolnego $i \in \{1,2,...,|\mathcal{S}|\}$*

$$\lim_{n\to\infty} p_{ij}(n) = \pi_j > 0. \tag{2.3}$$

Ponadto istnieją stałe $c > 0$ oraz $\gamma \in (0,1)$ takie, że

$$|p_{ij}(n) - \pi_j| < c\gamma^n. \tag{2.4}$$

Uwaga 2.1 *W nieprzywiedlnym łańcuchu Markowa wszystkie stany mają ten sam okres ([?], Twierdzenie 2, s. 299). Jeśli d jest okresem łańcucha P, to zbiór stanów $\mathcal{S}$ można rozbić na podklasy cykliczne S_1,...S_d tak, że jeśli zdefiniujemy dla s_i,$s_j \in S_m$ nową macierz przejścia $p_{ij}^m = p_{ij}(d)$, to macierz ta będzie definiować nieprzywiedlny i nieokresowy łańcuch Markowa. Odtąd będziemy w tym rozdziale implicite rozważać tylko nieokresowe łańcuchy Markowa.*

Definicja 2.5 *(s. 47, [?]) Odległością wahania całkowitego (ang. Total Variation Distance) między rozkładami prawdopodobieństwa μ i ν na (Ω,$\mathcal{F}$) nazywamy wielkość*

$$\|\mu - \nu\|_{TV} = \max_{A \subset \Omega} |\mu(A) - \nu(A)|. \tag{2.5}$$

Lemat 2.1 *(Proposition 4.2, [?])Dla miar probabilistycznych dyskretnych mamy*

$$\|\mu - \nu\|_{TV} = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|. \tag{2.6}$$

Dowód. Niech $A \subseteq \Omega$ będzie taki, że dla każdego $x \in A$ mamy $\mu(x) \geqslant \nu(x)$. Wtedy

$$\begin{aligned} \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)| &= \frac{1}{2} \left(\sum_{x \in A} (\mu(x) - \nu(x)) + \sum_{x \in A^c} (\nu(x) - \mu(x) \right) = \\ &= \frac{1}{2} (\mu(A) - \nu(A) + \nu(A^c) - \mu(A^c)) = \mu(A) - \nu(A) = \\ &= \max_{A \subset \Omega} |\mu(A) - \nu(A)|. \end{aligned} \tag{2.7}$$

♦

Twierdzenie 2.2 *(Theorem 4.9, [?]) Jeśli $|\Omega| < \infty$, to dla łańcucha Markowa o funkcji przejścia $P(\cdot,\cdot)$ z rozkładem stacjonarnym π istnieją stałe $C > 0$ i $\alpha \in (0,1)$ takie, że*

$$d(t) := \max_{s \in \Omega} \|P^t(s,\cdot) - \pi\|_{TV} \leqslant C\alpha^t. \tag{2.8}$$

Poniższe twierdzenie podaje charakteryzację stałej α z Twierdzenia **??**.

Twierdzenie 2.3 *([?], Proposition 1.1) Niech λ będzie największą wartością własną łańcucha Markowa o macierzy P ostro mniejszą niż 1. Wtedy istnieje stała $C(n)$ taka, że*

$$d(t) \leqslant C(n)\lambda^t. \tag{2.9}$$

Twierdzenie **??** wydaje się bardzo mocne, jednak fakt, że nieznana stała C zależy od liczby stanów n, skłania do poszukiwania dokładniejszych oszacowań. Zauważmy, że dla dużego n spektrum macierzy P może okazać się bardzo bogate. W szczególności λ może być bardzo bliskie 1.

Wprowadźmy jeszcze jedno oznaczenie na potrzeby dalszego wywodu,

$$\overline{d}(t) := \max_{x,y \in \Omega} \|P^t(x,\cdot) - P^t(y,\cdot)\|_{TV}. \tag{2.10}$$

Lemat 2.2 *(Exercise 4.1, [?]) Niech $\mathcal{P}$ oznacza zbiór wszystkich miar probabilistycznych na Ω. Wtedy*

$$d(t) = \sup_{\mu \in \mathcal{P}} \|\mu P^t - \pi\|_{TV}. \tag{2.11}$$

Definicja 2.6 *(4.5, [?]) Czasem mieszania (ang. Mixing Time) nazywamy wielkość*

$$t_{mix}(\epsilon) := \min\{t : d(t) \leqslant \epsilon\}. \tag{2.12}$$

Klasycznie definiowany czas mieszania jest deterministyczny. Na potrzeby niniejszej pracy wprowadźmy nieco inną definicję. Rozkładem w chwili t_0 trajektorii ω łańcucha Markowa X_t będziemy nazywali wektor $V(t) = (v_1,v_2,...,v_{|\Omega|})$, gdzie $v_i = \frac{|\{t:X_t=s_i\}|}{t_0}$. Rozkład trajektorii jest więc po prostu wektorem liczby odwiedzeń trajektorii procesu każdego ze stanów przez liczbę wszystkich wizyt (czyli $t \in \mathbb{N}$). Teraz losowy czas mieszania $\tau_{mix}(\epsilon)$ definiujemy jako

$$\tau_{mix}(\epsilon,\omega) = \min\{t > 0 : \delta(t,\omega) < \epsilon\}, \tag{2.13}$$

gdzie

$$\delta(t,\omega) = |V(t,\omega) - \pi|_{TV}.$$

Tutaj $\omega = (s_1,s_2,...) \in \Omega := \mathcal{S}^\infty$ jest zdarzeniem elementarnym opisującym trajektorię procesu. Chcielibyśmy ustalić związek między $\mathbb{E}^x\tau_{mix}(\epsilon)$, $x \in \Omega$ oraz $t_{mix}(\epsilon)$.

Fakt 2.1 *([?], wzór 4.34) Niech $n \in \mathbb{N},\ n > 0$. Wtedy*

$$d(n \cdot t_{mix}(\epsilon)) \leqslant (2\epsilon)^n. \tag{2.14}$$

Fakt 2.2 *([?], Proposition 1.5) Dla dowolnego łańcucha Markowa $\{X_i\}$ istnieje funkcja $f : \Omega \times \Lambda \mapsto \Omega$ i ciąg niezależnych zmiennych losowych o tym samym rozkładzie i wartościach w Λ, $\{Z_i\}$, takich, że łańcuch Markowa $\{X_i\}$ można przedstawić w postaci $X_t = f(X_{t-1},Z_t)$, $t \geqslant 1$.*

Definicja 2.7 *(6.2.2, [?]) Rozważmy łańcuch Markowa $\{X_i\}$ i odpowiadający mu ciąg $\{Z_i\}$ z Faktu **??**, tzn. ciąg niezależnych zmiennych losowych o tym samym rozkładzie taki, że $X_t = f(X_{t-1},Z_t)$ dla pewnej funkcji f. Moment stopu τ ciągu $\{Z_i\}$ nazywamy zrandomizowanym momentem stopu dla łańcucha $\{X_i\}$.*

Definicja 2.8 *(6.3, [?]) Niech $\{X_i\}$ będzie nieprzywiedlnym łańcuchem Markowa z rozkładem stacjonarnym π. Czasem stacjonarnym nazywamy zrandomizowany moment stopu τ, być może zależny od stanu początkowego $x \in \Omega$, taki, że X_τ ma rozkład π, to znaczy*

$$P_x(X_\tau = y) = \pi(y). \tag{2.15}$$

Definicja 2.9 *(6.4, [?]) Mocnym czasem stacjonarnym łańcucha Markowa (X_t) z rozkładem stacjonarnym π nazywamy zrandomizowany moment stopu τ, być może zależny od stanu początkowego $x \in \Omega$, taki, że*

$$P_x(\tau = t,X_\tau = y) = P_x(\tau = t)\pi(y). \tag{2.16}$$

Innymi słowy, τ jest mocnym czasem stacjonarnym, jeśli X_τ ma rozkład π oraz X_τ jest niezależna od τ.

Twierdzenie 2.4 *([?],Proposition 6.10) Niech τ będzie mocnym czasem stacjonarnym. Wtedy*

$$d(x) = \max_{x\in\Omega}\|P^t(x,\cdot) - \pi\|_{TV} \leqslant \max_{x\in\Omega} P_x(\tau > t). \tag{2.17}$$

Definicja 2.10 *(6.6, [?]) Niech (X_t) będzie łańcuchem Markowa o macierzy przejścia P i rozkładzie stacjonarnym π na Ω. Dla $t \geqslant 1$ niech σ będzie zmienną losową o rozkładzie jednostajnym na $\{0,1,...,t-1\}$. Wtedy zmienna X_σ ma rozkład*

$$v_x^t := \frac{1}{t}\sum_{s=0}^{t-1} P^s(x,\cdot). \tag{2.18}$$

Czasem mieszania Cesaro $t^*_{mix}(\epsilon)$ nazywamy najmniejsze $t \geqslant 1$ takie, że dla każdego $x \in \Omega$

$$\|v_x^t - \pi\|_{TV} \leqslant \epsilon. \tag{2.19}$$

Twierdzenie 2.5 *(Theorem 6.15, [?]) Rozważmy łańcuch Markowa o macierzy P i rozkładzie stacjonarnym π. Jeśli τ jest czasem stacjonarnym dla tego łańcucha Markowa, to*

$$t^*_{mix}(0.25) \leqslant 4\max_{x\in\Omega}\mathbb{E}^x(\tau). \tag{2.20}$$

2.1.1 Dolne oszacowania czasu mieszania

Niech (X_t) będzie nieprzywiedlnym, aperiodycznym łańcuchem Markowa. Załóżmy, że rozkład stacjonarny π jest jednostajny na Ω. Niech $\Xi(x) := |\{y : P(x,y) > 0\}|$ będzie liczbą stanów osiągalnych z x w jednym kroku i niech

$$\Delta := \max_{x\in\Omega} \Xi(x). \tag{2.21}$$

Niech Ω_t^x oznacza zbiór stanów osiągalnych ze stanu x w t krokach. Oczywiście $|\Omega_t^x| \leqslant \Delta^t$. Powyższe oznaczenia i założenia będą obowiązywały w całym podrozdziale.

Fakt 2.3 *([?], Wzór 7.2) Niech P będzie nieprzywiedlnym, niekresowym łańcuchem Markowa i niech jego rozkład stacjonarny π będzie rozkładem jednostajnym. Jeśli $\Delta^t < (1-\epsilon)|\Omega|$, to*

$$t_{mix}(\epsilon) \geqslant \frac{\log(|\Omega|(1-\epsilon))}{\log \Delta}. \tag{2.22}$$

Na łańcuch Markowa o macierzy przejścia P będziemy patrzeć jak na graf, w którym Ω jest zbiorem wierzchołków, natomiast zbiór krawędzi definiujemy jako $E = \{\{x,y\} : P(x,y) + P(y,x) > 0\}$. Średnicą łańcucha Markowa będziemy nazywać maksymalną odległość między dwoma wierzchołkami grafu $G = (\Omega,E)$.

Fakt 2.4 *(7.1.2, [?]) Niech P będzie nieprzywiedlnym, niekresowym łańcuchem Markowa o średnicy L i niech $x,y \in \Omega$ będą wierzchołkami grafu $G = (\Omega,E)$ takimi, że dystans na grafie G między x i y wynosi L. Wtedy*

- *nośniki funkcji $P^{\lfloor (L-1)/2 \rfloor}(x,\cdot)$ oraz $P^{\lfloor (L-1)/2 \rfloor}(y,\cdot)$ są rozłączne,*
- *$\overline{d}(\lfloor (L-1)/2 \rfloor) = 1$,*
- *dla każdego $\epsilon < \frac{1}{2}$ zachodzi $t_{mix}(\epsilon) \geqslant \frac{L}{2}$.*

Twierdzenie 2.6 *([?], 5.2) W przypadku błądzenia losowego po hiperkostce $\{0,1\}^n$ mamy*

$$t_{mix}(\epsilon) \geqslant \frac{1}{2} n \log n + O(n). \tag{2.23}$$

Wąskie gardła

Definicja 2.11 *(Wzór (7.4), [?]) Miarę krawędziową definiujemy jako*

$$Q(A,B) = \sum_{x\in A, y\in B} \pi(x)P(x,y). \tag{2.24}$$

$Q(A,B)$ można interpretować jako prawdopodobieństwo przejścia z A do B przy rozkładzie stacjonarnym.

Definicja 2.12 *(Wzór 7.5, [?]) Drożnością (ang. bottleneck ratio) zbioru S nazywamy stosunek*

$$\Phi(S) := \frac{Q(S,S^c)}{\pi(S)}. \tag{2.25}$$

Drożność łańcucha Markowa definiujemy jako

$$\Phi_* = \min_{S:\pi(X)\leqslant \frac{1}{2}} \Phi(S). \tag{2.26}$$

Zauważmy, że im większa drożność, tym t_{mix} będzie mniejsze.

Twierdzenie 2.7 *([?], Theorem 7.3)* $t_{mix}(1/4) \geqslant \frac{1}{4\Phi_*}$.

Czasy mieszania a czasy trafienia w zbiór

Definicja 2.13 *(1.6, [?]) Łańcuch Markowa nazywamy odwracalnym, jeśli spełnia układ równań*

$$\pi(x)P(x,y) = \pi(y)P(y,x) \ \forall x,y \in \Omega. \tag{2.27}$$

Definicja 2.14 *Czasem trafienia łańcucha Markowa (X_t) w zbiór $A \subset \Omega$ nazywamy*

$$\tau_A = \min\{t \geqslant 0 : X_t \in A\}. \tag{2.28}$$

Pierwszym czasem powrotu łańcucha (X_t), $X_0 = x$, do punktu $x \in \Omega$ nazywamy

$$\tau_x^+ = \min\{t \geqslant 1 : X_t = x\}. \tag{2.29}$$

Twierdzenie 2.8 *([?], Theorem 10.14)*

- *Dla każdego $t \geqslant 0$ oraz $x \in \Omega$ zachodzi*

$$\|P^t(x,\cdot) - \pi\|_{TV}^2 \leqslant \frac{1}{4}\left(\frac{P^{2t}(x,x)}{\pi(x)} - 1\right), \tag{2.30}$$

- *jeżeli dla każdego $x \in \Omega$ zachodzi $P(x,x) \geqslant \frac{1}{2}$, to*

$$t_{mix}(1/4) \leqslant 2 \max_{x\in\Omega} \mathbb{E}_\pi(\tau_x) + 1. \tag{2.31}$$

2.1.2 Analiza w oparciu o wartości własne

Twierdzenie 2.9 *([?], Proposition 2.1) Niech P będzie macierzą przejścia. Wtedy*

1. *Jeśli λ jest wartością własną P, to $|\lambda| \leqslant 1$.*
2. *Jeśli łańcuch P jest nieprzywiedlny, to podprzestrzeń własna związana z wartością własną $\lambda = 1$ jest jednowymiarowa i rozpięta przez wektor jednostkowy (1,1,...,1).*
3. *Jeśli P jest nieprzywiedlny i aperiodyczny, to -1 nie jest wartością własną P.*

Niech π będzie rozkładem stacjonarnym dla P. Dla funkcji f,g określonych na $\mathcal{S} = \{s_1,s_2,...,s_m\}$ definiujemy iloczyn skalarny

$$(f,g)_\pi = \sum_{i=1}^{m} f(s_i)g(s_i)\pi(s_i). \tag{2.32}$$

Definicja 2.15 *Jeśli dla każdego $x,y \in \mathcal{S}$ π spełnia*

$$\pi(x)P(x,y) = \pi(y)P(y,x) \tag{2.33}$$

i π jest rozkładem początkowym łańcucha X_t, to łańcuch X_t nazywamy odwracalnym.

Dla macierzy przejścia P odwracalnego łańcucha Markowa ustawmy jej wartości własne w porządku nierosnącym,

$$1 = \lambda_1 > \lambda_2 \geqslant ... \geqslant \lambda_{|\mathcal{S}|} \geqslant -1.$$

Niech $\lambda_* = \max\{|\lambda| : \lambda \text{ jest wartością własną } P, \lambda \neq 1\}$. Różnicę $\gamma_* = 1 - \lambda_*$ będziemy nazywać absolutnym rozstępem spektralnym (ang. *absolute spectral gap*). Dla odwracalnego łańcucha Markowa rozstęp spektralny definiujemy jako $\gamma = 1 - \lambda_2$. Czasem relaksacji t_{rel} odwracalnego łańcucha Markowa nazywamy wielkość $t_{rel} = \frac{1}{\gamma_*}$. Zachodzi twierdzenie

Twierdzenie 2.10 *(Theorems 12.3 i 12.4, [?]) Niech P będzie macierzą przejścia odwracalnego, nieprzywiedlnego łańcucha Markowa i niech $\pi_{\min} = \min\{\pi(x), x \in \mathcal{S}\}$. Wtedy*

$$t_{mix}(\epsilon) \leqslant \log\left(\frac{1}{\epsilon\pi_{\min}}\right) t_{rel}. \tag{2.34}$$

Jeśli dodatkowo P jest aperiodyczny, to

$$t_{mix}(\epsilon) \geqslant (t_{rel} - 1)\log\left(\frac{1}{2\epsilon}\right). \tag{2.35}$$

2.1.3 Czasy pokrycia

Czas pokrycia τ_{cov} łańcucha X_t to pierwszy moment, w którym każdy ze stanów został odwiedzony przynajmniej raz. Formalniej:

Definicja 2.16 *Czas pokrycia τ_{cov} to zmienna losowa taka, że:*

$$\tau_{cov} = \min\{t : \forall x \in \Omega \exists s \geqslant t : X_t = x\}. \tag{2.36}$$

Czas pokrycia definiuje się czasem w sposób deterministyczny, rozważając wartość oczekiwaną dla „pesymistycznego" rozkładu początkowego:

$$t_{cov} = \max_{x \in \Omega} \mathbb{E}^x \tau_{cov}. \tag{2.37}$$

Definicja 2.17 *Błądzeniem losowym w $\mathbb{Z}_n$ nazywamy łańcuch Markowa taki, że $\mathcal{S} = \{0,1,...,n\}$, $p_{i,i+1} = p_{i,i-1} = p_{0n} = p_{n0} = \frac{1}{2}$, oraz $p_{ij} = 0$ dla $i \neq i-1 mod n$, $i \neq i+1 mod n$.*

W poniższych rozważaniach przyda nam się następujący fakt:

Fakt 2.5 *([?], Example 11.1) Niech X_t będzie błądzeniem losowym w $\mathbb{Z}_n$. Wtedy*

$$t_{cov} = \frac{n(n-1)}{2}. \tag{2.38}$$

2.1.4 Uwagi

Czasami czasy mieszania w łańcuchach Markowa definiuje się inaczej. W artykule [?] za czas mieszania przyjmuje się wielkość

$$\tau_R = \min\{t > 0 : \max_{x,y \in \mathcal{S}} |P^t(x,y) - \pi(y)| \leqslant 1/n^3, \ x \text{ jest stanem początkowym}\}. \tag{2.39}$$

Dla tak zdefiniowanego τ_R wiadomo, że grafy o stałej liczbie stopni wierzchołków mają czasy mieszania rzędu $O(\log n)$ [?].

Przedstawione w poprzednich rozdziałach wyniki analityczne dotyczące oszacowań na czasy mieszania nie wyczerpują aktualnej wiedzy. Bardziej skomplikowane przykłady i zaawansowane techniki można znaleźć np. w [?]. W teorii czasów mieszania przedstawionych w cytowanej pozycji używa się m.in. teorii form Dirichleta, transformat Fouriera, teorii reprezentacji, martyngałów, nierówności Poincarégo i innych.

2.2 Znaczenie dla informatyki - patrolowanie

Zdefiniujemy teraz ogólny problem patrolowania, który został wspomniany we Wstępie. Załóżmy, że mamy k agentów na grafie $G = (\mathcal{S},E)$, tzn. funkcje $f_1,f_2,...,f_k$ określone na $\mathbb{N}$ o wartościach w $\mathcal{S}$. Dla zadanego $m \in \mathbb{N}$ chcemy tak dobrać $f_1,f_2,...,f_k$, aby spełniony był warunek

$$\forall i \in \mathbb{N} \ \forall s \in \mathcal{S} \ \exists t \in [im,(i+1)m) \ \exists j \in \{1,2,...,k\} : \ f_j(t) = s. \tag{2.40}$$

Dla ustalonych G,m,k taki warunek może nie być spełniony. Można zastanawiać się, jakie musi być najmniejsze k, aby dla zadanych G,m zachodził warunek (**??**) (analogicznie: jakie jest najmniejsze m, że przy ustalonych G,k warunek (**??**) jest spełniony).

Problem można także sformułować w języku teorii gier. Załóżmy, że mamy dwóch graczy: włamywacza i strażnika, oraz graf $G = (\mathcal{S},E)$. Włamywacz wybiera chwilę $\tau_{intrusion}$ i wierzchołek s i wygrywa, jeśli przez m kolejnych jednostek czasu w wierzchołku s strażnik się nie pojawi. Strażnik wybiera sposób poruszania się po grafie i potrzebuje 1 jednostkę czasu, aby przejść z jednego wierzchołka grafu do drugiego. Wygrywa, jeżeli w ciągu m jednostek czasu od $\tau_{intrusion}$ odwiedzi wierzchołek s.

Można pokusić się o powiązanie czasów mieszania z problemem patrolowania grafu. Jeśli założymy, że wyżej spacer wyżej opisanego strażnika po grafie jest błądzeniem losowym, to z punktu widzenia włamywacza dla $t > t_{mix}$ strażnik znajdzie się w chwili t w wierzchołku s z prawdopodobieństwem bliskim $\pi(s)$. W takim przypadku rozsądną strategią włamywacza będzie wybór $s' = \text{argmin}_{s \in \mathcal{S}} \pi(s)$. Wynika stąd, że dla strażnika błądzącego losowo optymalną strategią jest wybór takiej macierzy przejścia P, aby π był rozkładem jednostajnym.

W informatyce teoria patrolowania rozstrzyga nieraz jak silne zabezpieczenia należy wprowadzić, aby zabezpieczyć dane zasoby. Można wyobrazić sobie zastosowania patrollingu w programach antywirusowych, w testowaniu spójności dużych baz danych, czy też przy monitorowaniu sieci komputerowych.

Projekt systemu

3.1 Symulacje

Trudności w analitycznym ujęciu tematu czasów mieszania skłaniają do zastosowania symulacji. Symulować łańcuch Markowa możemy dwojako:

1. Określmy macierz przejścia P. Aby uzyskać rozkład procesu w n-tym kroku, wystarczy obliczyć P^n.
2. Tworzymy graf $G = (V,E)$, w którym wierzchołki reprezentują stany łańcucha, natomiast $E = \{(V_i,V_j) : p_{ij} > 0\}$. Wybieramy stan startowy zgodnie z rozkładem początkowym procesu. Jeśli stan w kroku t to V_i, to stan w kroku $t+1$, V_j, wybieramy spośród sąsiadów wierzchołka V_i z prawdopodobieństwem p_{ij}. Interesować nas będą wektory odwiedzeń trajektorii X_t każdego z wierzchołków podzielonych przez liczbę wszystkich skoków. Takie wektory losowe nazywać będziemy rozkładami trajektorii procesu.

W obu przypadkach po każdym kroku liczymy $TV distance$ i zatrzymujemy proces, jeśli tylko osiągniemy czas mieszania. W pierwszym przypadku otrzymalibyśmy dokładne, deterministyczne oszacowanie na czas mieszania, tracilibyśmy jednak informację o jego rozkładzie, natomiast w drugim przypadku symulujemy tylko pojedynczą trajektorię procesu, więc w celu wnioskowania o rozkładzie czasu mieszania konieczne będzie przeprowadzenie wielu prób Monte-Carlo. Oczywiście pierwszy algorytm jest nieporównanie szybszy od drugiego, jednak potencjalne duże uwarunkowanie zadania mnożenia macierzy oraz szybka akumulacja błędów numerycznych po każdej iteracji wyklucza go z zastosowań do celów niniejszej pracy.

Badanie τ_{mix} zamiast t_{mix} motywowane jest względami praktycznymi zarówno jak i teoretycznymi: t_{mix} jest deterministyczny i związany z deterministycznym rozkładem procesu, P^t.

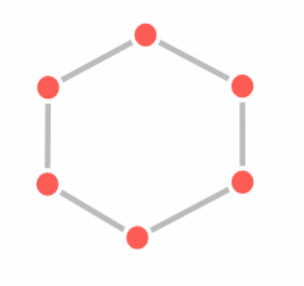
3.2 Opis algorytmów użytych w symulacjach

Algorytm $TV distance$ Przypomnijmy, że odległość wahania całkowitego wyraża się wzorem (**??**). W naszym przypadku ν to rozkład stacjonarny, który w każdym rozważanym w niniejszej pracy łańcuchu Markowa jest rozkładem jednostajnym, tj. $P(X_t = s_i) = \frac{1}{|V|}$ dla każdego $i = 1,2,...,N = |V|$.

Złożoność czasowa i pamięciowa algorytmu $TV distance$ to $O(|V|)$.

Grafy pełne Grafy pełne, jako grafy o najlepszej możliwej drożności, mają najmniejszy t_{mix}. Z tego powodu badanie rozkładu zmiennej losowej τ_{mix} za pomocą symulacji powinno okazać się najłatwiejsze.

Grafy cykliczne Błądzenie losowe po grafie cyklicznym $\mathbf{C}(n)$ można utożsamić z błądzeniem losowym w $\mathbb{Z}_n$. Jeśli n jest parzyste, to rozkład stacjonarny nie istnieje w ścisłym sensie. W takim przypadku jednak średnia liczba odwiedzin przez łańcuch Markowa w każdym ze stanów dalej będzie zbiegać do $\frac{1}{|\mathcal{S}|}$, więc jest sens mówienia o czasach mieszania.



Rysunek 3.1: Przykład grafu cyklicznego, $N = 6$.

Archipelagi Rozważmy graf $G = (V,E)$, gdzie $|V| = N$. Graf G składa się z k składowych G_1,...,G_k takich, że każdy podgraf $G_j = (V_j,E_j)$, $j = 1,...,k$, $V_j = \{v_{1,j}, v_{2,j}, ..., v_{|V_j|,j}\}$ jest kliką i $|V_j| = N/k$. Składowe G_j będziemy nazywać wyspami. Składowe G_j są połączone między sobą i krawędziami w sposób opisany następującym algorytmem:

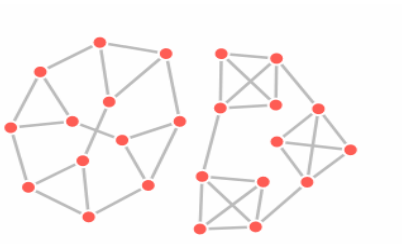
Algorithm 3.1: Sposób łączenia wysp w grafie G. Tutaj `j1` jest indeksem oznaczającym numer bieżącej wyspy, natomiast `j2` jest indeksem oznaczającym numer wyspy, z którą aktualnie łączymy wyspę `j1`. `br` + `shiftmod(N/k)` jest indeksem numeru wierzchołka obu wysp, które łączymy. Dodatkowa zmienna `shift` odpowiada za to, aby po zakończeniu dodawania krawędzi do jednej wyspy, zacząć dodawać krawędzie do kolejnych wysp z wierzchołków dotychczas nieużywanych.

```
1 initialization;
2 shift = 0;
3 for j1 in 1:k do
4     for j2 in j1:k do
5         for br in 1:i do
6             połącz V[br+shift mod(N/k),j1] i V[br+shift mod(N/k),j2];
7         shift = shift+bridges ;
```

Każdej krawędzi wychodzącej z wierzchołka V_{ij} przypisujemy prawdopodobieństwo $\frac{1}{\deg(V_{ij})}$. Rozkład stacjonarny π dla łańcucha Markowa odpowiadającego tak skonstruowanemu grafowi w symulatorze jest przybliżany poprzez wykonanie dużej liczby skoków (350000) i zliczenie liczby wizyt w każdym wierzchołku. Uzyskane rozkłady stacjonarne okazały się bliskie rozkładu stacjonarnego w sensie odległości wahania całkowitego (dla $k = 2,3,5$ odpowiednio 0.0477, 0.0273, 0.0156).

Definicja 3.1 *Grafy skonstruowane za pomocą algorytmu* **??** *będziemy nazywać* archipelagami *i oznaczać* $A = A(N,k,i)$.

Do celów symulacji przyjęliśmy $N = 120$, $i = 1$, $k \in D = \{1,2,3,4,5,6,8,10,12,20,40,60\}$,$\epsilon = 0.25$. Wykonano po $M = 1000$ prób Monte-Carlo dla każdego k.



Rysunek 3.2: Przykłady archipelagów odpowiednio: $A(12,4,1)$ oraz $A(12,3,1)$.

Model Barabási-Alberta

Definicja 3.2 *Siecią bezskalową (ang. scale free network) będziemy nazywać graf nieskierowany taki, że liczba krawędzi wychodzących z ustalonego wierzchołka k ma rozkład $P(k) \sim k^{-\gamma}$, gdzie $\gamma > 1$ jest ustaloną liczbą.*

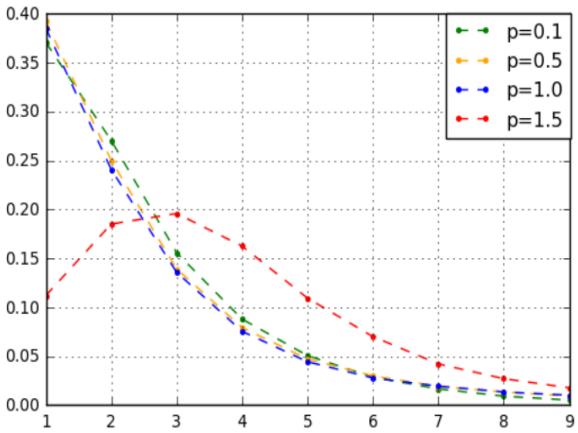
W modelu Barabási-Alberta [**?**] opracowano algorytm służący do generowania grafów losowych bezskalowych z parametrem $\gamma = 3$. Jego zaletą jest możliwość łatwego dodawania nowych węzłów (wierzchołków). Ponadto model BA ma cechę tzw. dołączania preferencyjnego (ang. *preferential attachment*), to znaczy, węzły o dużej liczbie połączeń mają większą szansę na stworzenie nowego połączenia niż węzły z małą liczbą połączeń.

Przypuśćmy, że mamy spójną sieć rozmiary n. Po dołączeniu nowego, n+1-go wierzchołka, dla każdego i = 1,2,...,n dodajemy losowo krawędź łączącą wierzchołki i oraz n+1 z prawdopodobieństwem $k_i / \sum_{j=1}^{n} k_j$, gdzie k_j jest całkowitą liczbą krawędzi wychodzących z wierzchołka j przed dodaniem n+1-go wierzchołka do sieci. Do symulacji użyliśmy zmodyfikowanego algorytmu, który zapewnia spójność grafu i pozwala modyfikować prawdopodobieństwo dołączenia krawędzi.

Algorithm 3.2: Zmodyfikowany model Barabási-Alberta

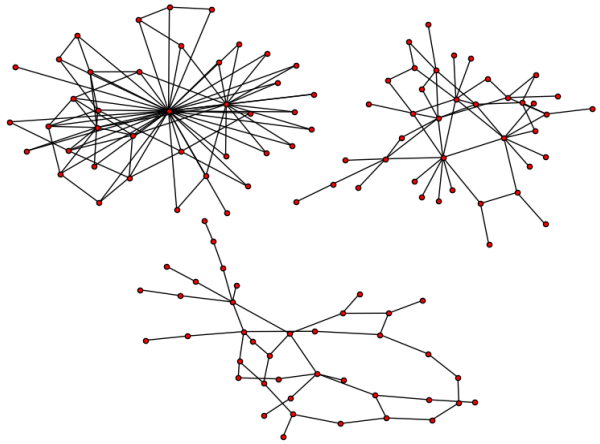
```
Data: N, p,q
add edge between vertex[0] and vertex[1] ;
for i in 2:N do
    while vertex[i] has no neighbours do
        for j in 0:i do
            with probability k_i^p / sum_{j=1}^{i-1} k_j^q add edge between vertex[i] and vertex[j]
```

Dla $p > \kappa$ może się zdarzyć, że $k_i^p / \sum_{j=1}^{i-1} k_j^\kappa > 1$ dla niektórych i i N. W takim przypadku oczywiście dołączamy krawędź. Tego typu grafy to z dużym prawdopodobieństwem gwiazdy, tj. drzewa, w których tylko jeden wierzchołek nie jest liściem.



Rysunek 3.3: Oszacowanie $P(k)$ w zmodyfikowanym modelu Barabási-Alberta dla różnych potęg licznika w grafie o 120 wierzchołkach; 1000 prób Monte-Carlo.

W przypadku zmodyfikowanego modelu Barabási-Alberta może nie istnieć jednoznacznie wyznaczony rozkład stacjonarny łańcucha Markowa i prawie nigdy nie będzie jednostajny. W takim przypadku za żozkład stacjonarny"przyjmiemy, podobnie jak w przypadku grafów cyklicznych, średnią liczbę odwiedzin przez łańcuch w każdym z wierzchołków. W praktyce oznaczać to będzie, że dla danego łańcucha wykonamy dużą liczbę skoków (500000) i za rozkład stacjonarny przyjmiemy liczbę odwiedzin w danym wierzchołku podzieloną przez liczbę wszystkich skoków. Dla grafów cyklicznych o 350 wierzchołkach uzyskane przybliżenia są odległe w sensie odległości wahania całkowitego od rozkładu stacjonarnego o mniej niż 0.003.



Rysunek 3.4: Przykładowe grafy o 40 wierzchołkach wygenerowane za pomocą zmodyfikowanego algorytmu Barabási-Alberta, $p = 1.5$, $p = 1.0$, $p = 0.1$.

3.3 Opis danych

Wygenerowane dane znajdują się w plikach `islandsMixingTime.sta`, `islandsTVdist.sta`, `circleMixingTime.sta` oraz `circleTVdist.sta`. Plik `islandsMixingTime.sta` zawiera 24 kolumny i 1000 wierszy. W 12 pierwszych kolumnach znajdują się czasy mieszania archipelagów $A(120,k,i)$, $k \in D$, natomiast pozostałe kolumny to dane zlogarytmowane. Tabela w `circleMixingTime.sta` zorganizowana jest podobnie: znajdują się w niej wyniki symulacji czasów mieszania grafów cyklicznych rozmiaru $N = 4,5,...,38$ i logarytmy tych czasów mieszania - łącznie 38 kolumn po 2500 obserwacji. Pliki `islandsTVdist.sta` oraz `circleTVdist.sta` zawierają wyniki symulacji oszacowania średnich odległości całkowitego wahania dla archipelagów i grafów cyklicznych. Kolumny numerowane są tak samo jak w plikach omawianych wyżej. W tabeli i-ty wiersz oznacza średnią arytmetyczną z odległości wahania całkowitego z $M = 1000$ (dla archipelagów) i $M = 2500$ (dla grafów cyklicznych) prób Monte-Carlo w $N+i$ momencie, gdzie N jest rozmiarem grafu ($N = 120$ dla archipelagów, $N = 3 + K$ dla grafów cyklicznych, gdzie K jest numerem kolumny).

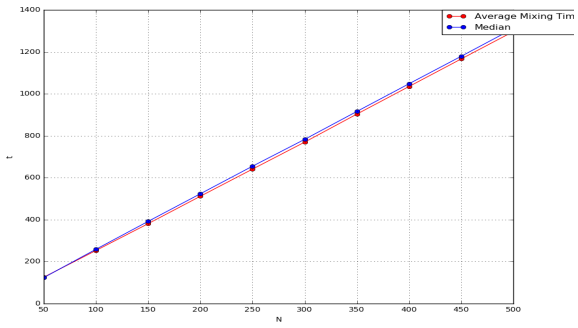
3.4 Wyniki symulacji

Symulacje miały służyć zbadaniu rozkładu zmiennej losowej τ_{mix} wybranych klas łańcuchów Markowa. Przy okazji wygenerowano dane dotyczące średniej wartości odległości wahania całkowitego dla różnych chwil $t > N$, gdzie N jest rozmiarem grafu (w języku łańcuchów Markowa: liczbą stanów łańcucha). W przypadku grafów cyklicznych (równoważnie: błądzenia losowego w $\mathbb{Z}_N$) wyniki okazały się zgodne z intuicjami i można pokusić się o ich interpretację, czy nawet wyjaśnienie w oparciu o np. znane wyniki analityczne teorii czasów pokrycia. Rezultaty dotyczące archipelagów wypadły z kolei w sposób zgoła nieoczekiwany. Stosunek liczby wysp do logarytmu z czasu mieszania dorze przybliża się funkcją liniową. Podobnie interesujące własności okazuje się mieć funkcja średniej odległości całkowitego wahania archipelagów. Poniżej zaprezentowano analizę wygenerowanych danych. Dla uproszczenia zmienną dyskretną t często będziemy traktować jako zmienną ciągłą. W szczególności wykresy funkcji $d(t)$ dla większej przejrzystości rysowane są linią ciągłą. Dla uproszczenia zapisu będziemy stosować te same oznaczenia dla funkcji $d(t)$ i dla jej rozszerzenia do funkcji ciągłej i różniczkowalnej prawie wszędzie.

3.4.1 Grafy pełne

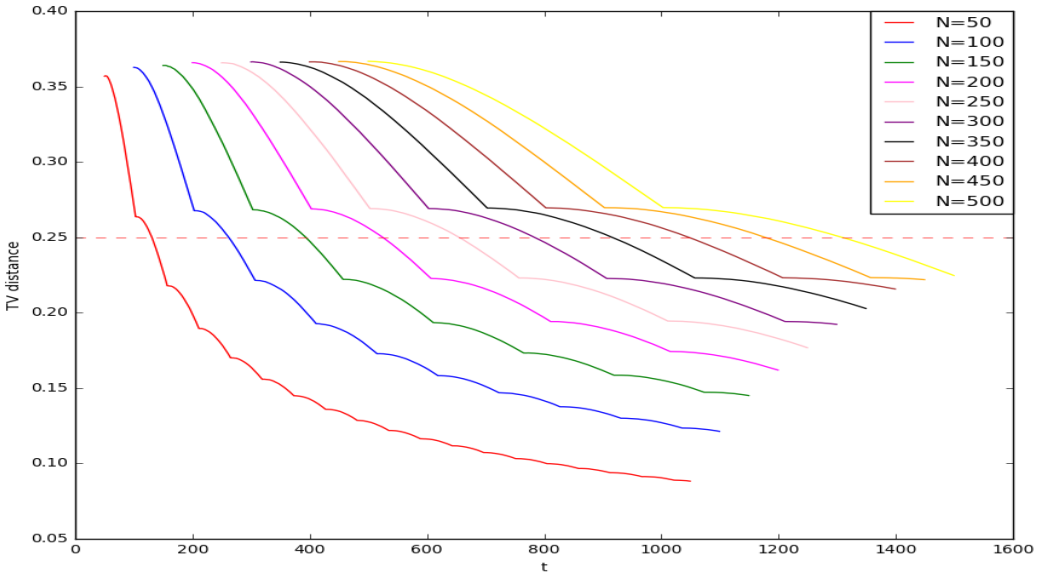
W grafach pełnych średnie czasy mieszania, $\mathbb{E}\tau_{mix}(0.25)$, rosną liniowo względem rozmiaru grafu, co można wywnioskować na podstawie Rysunku **??**. Podobnie zachowują się mediany $\tau_{mix}(0.25)$. Do wyników symulacji

dopasowano proste regresji o równaniach odpowiednio $t = 2.607N - 8.421$ dla średniej oraz $t = 632N - 4.667$ dla mediany.



Rysunek 3.5: Wykres średnich czasów mieszania przy $\epsilon = 0.25$ grafów pełnych $N = 50{,}100{,}...{,}500$.

Następnie zbadano oszacowanie średniej wartości funkcji $\delta(t)$. Jej przebieg przedstawiono na Rysunku **??**.

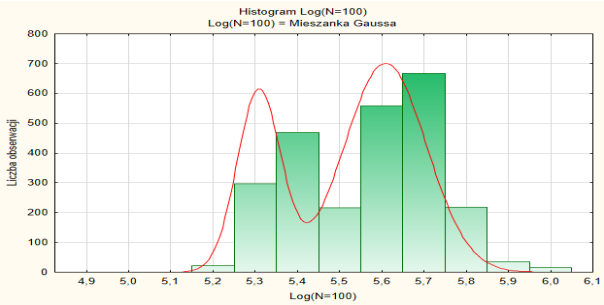


Rysunek 3.6: Wykres średniej funkcji $\delta(t)$ dla grafów pełnych $N = 50{,}100{,}...{,}500$.

Obserwacje dotyczące asymptotyki funkcji $\mathbb{E}\delta(t)$ są zgodne z Twierdzeniem **??**. Zwróćmy jednak uwagę, że nie jest to ani funkcja wypukła, ani malejąca. Co więcej, okazuje się, że $\mathbb{E}\delta(t)$ grafu pełnego o rozmiarze N ma maksima i minima lokalne oddalone od siebie o N, przy czym minima lokalne obserwowane są w punktach $t = N, 2N, 3N,...$ (por. Rysunek **??**). Taki przebieg funkcji $\mathbb{E}\delta(t)$ znajduje intuicyjne wyjaśnienie na podstawie definicji funkcji $d(t)$. Omówimy je w rozdziale dotyczącym grafów cyklicznych.

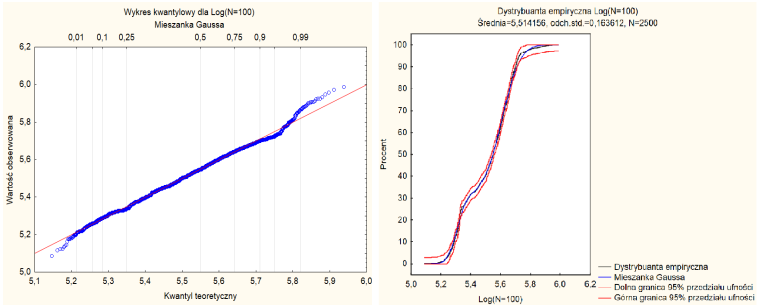
Rozkład zmiennej $\tau_{mix}(\epsilon,N)$

Po pierwsze zauważmy, że z samej definicji wynika, że zmienna losowa $\tau_{mix}(\epsilon,N)$ przyjmuje tylko wartości ściśle dodatnie i dowolnie duże. Przyjrzyjmy się statystykom podstawowym zmiennej $\log \tau_{mix}(\epsilon = 0.25, N = 100)$.



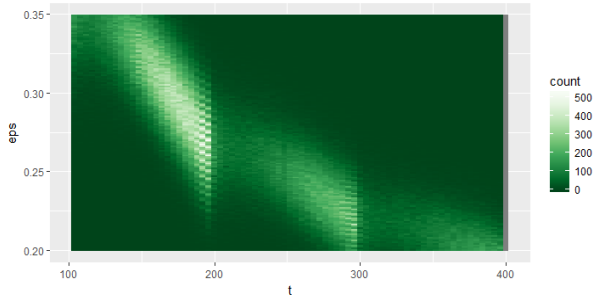
Rysunek 3.7: Histogram $\log \tau_{mix}(\epsilon = 0.25, N = 100)$ dla grafu pełnego z dopasowaną mieszanką rozkładów normalnych.

Histogram ewidentnie wskazuje na dwumodalność rozkładu. Dopasowano do niego mieszankę rozkładów normalnych $\mathcal{N}(\mu = 5.309, \sigma = 0.052)$ ze współczynnikiem mieszania 0,3155 oraz $\mathcal{N}(\mu = 5.609, \sigma = 0.097)$ ze współczynnikiem mieszania 0.6845. W celu zbadania zgodności rozkładów sporządzono wykresy dopasowania dystrybuanty oraz wykres kwantylowy.

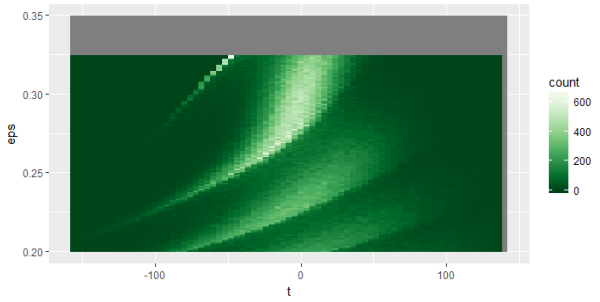


Rysunek 3.8: Wykres kwantylowy i dystrybuanta empiryczna z dopasowaną mieszanką rozkładów normalnych.

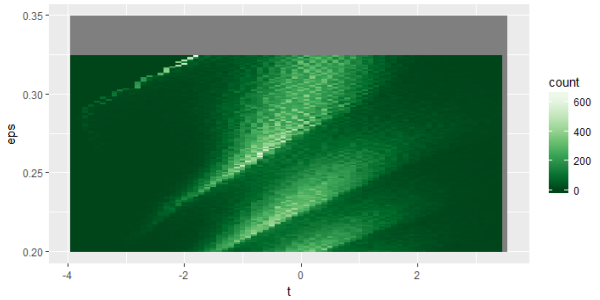
Obecność dwumodalności domaga się jakiejś interpretacji. Może ona wynikać np. z przebiegu funkcji $d(t)$, w szczególności z jej niemonotoniczności. Oznacza to, że kształt funkcji gęstości rozkładu zmiennej τ_{mix} zależałby od ϵ: wraz ze wzrostem ϵ zwiększałaby się nie tylko średnia i wariancja rozkładu, ale również jego typ. Aby zbadać tę zależność, dla grafu pełnego $N = 100$ wykonano po $M = 3000$ prób Monte-Carlo dla $\epsilon = 0.5, 0.49, 0.48, ..., 0.09$. Dla każdej otrzymanej próby sporządzono histogram. Otrzymane zależności zostały przedstawione na Rysunkach **??**, **??** oraz **??**.



Rysunek 3.9: Histogramy τ_{mix} w zależności od ϵ (oś pionowa).



Rysunek 3.10: Histogramy ilustrujące wzrost wariancji $\tau_{mix} - \overline{\tau}_{mix}$ w zależności od ϵ (oś pionowa).



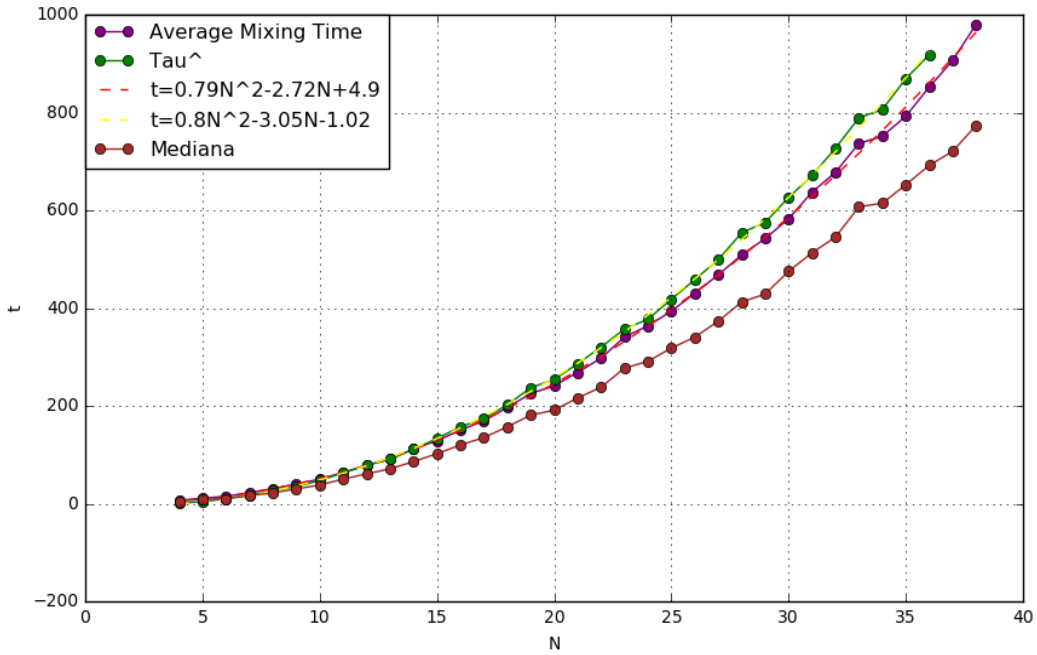
Rysunek 3.11: Histogramy ilustrujące dwumodalność rozkładu ustandaryzowanego $\frac{\tau_{mix} - \overline{\tau}_{mix}}{\mathrm{Var}\tau_{mix}}$ w zależności od ϵ (oś pionowa).

Na Rysunkach **??**, **??** i **??** wyraźnie rysuje się wielomodalność badanych rozkładów. Co więcej, mody lokują się mniej więcej w chwilach będących wielokrotnościami rozmiaru grafu N. Na Rysunku **??** znać również kształt krzywej z Wykresu **??**.

3.4.2 Grafy cykliczne

Analizę rozpoczniemy od zbadania średniego czasu mieszania i mediany czasów mieszania w grafach cyklicznych w zależności od rozmiaru grafu. Otrzymane w symulacjach wyniki zostały przedstawione na Rysunku **??**. Do punktów $\overline{\tau}_{mix}(0.25,N)$ dopasowano metodą najmniejszych kwadratów wielomian drugiego stopnia postaci $0.79N^2 - 2.72N + 4.9$. Nie zagłębiajmy się w formalną ocenę dopasowania - współczynniki są istotne, błąd standardowy niewielki, natomiast na wykresie wielomian regresji wydaje się bardzo dobrze dopasowany. Mediana dopasowuje się podobnie, jednak współczynnik przy N^2 jest nieco większy niż w przypadku średniej. Zaznaczamy tutaj, że wyestymowane stałe zależą tutaj od ϵ, którego wartość przyjęliśmy arbitralnie równą 0.25. Na podstawie symulacji i Faktu **??** można więc obronić następującą hipotezę:

Obserwacja 3.1 *W przypadku błądzenia losowego w $\mathbb{Z}_n$ średnie czasy mieszania, mediana czasów mieszania oraz czasy pokrycia mają tę samą asymptotykę, $\Theta(n^2)$.*

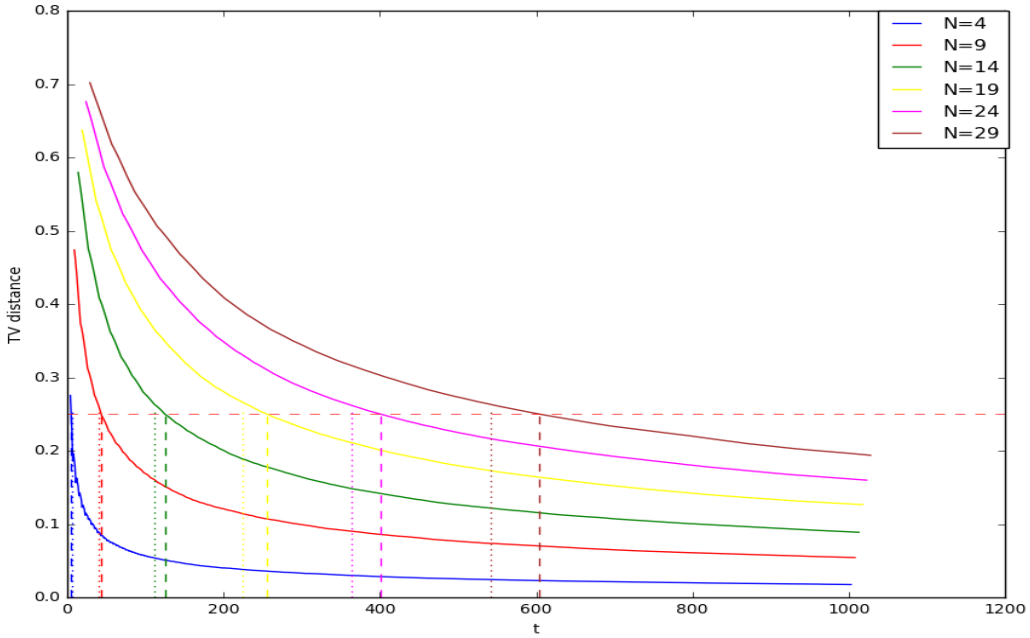


Rysunek 3.12: Wykres oszacowanych średnich czasów mieszania $\overline{\tau}_{mix}(0.25)$, oszacowań $\hat{\tau}_{mix}$ oraz median czasów mieszania, $\epsilon = 0.25$, dla grafów cyklicznych, $N = 4{,}5{,}...{,}39$

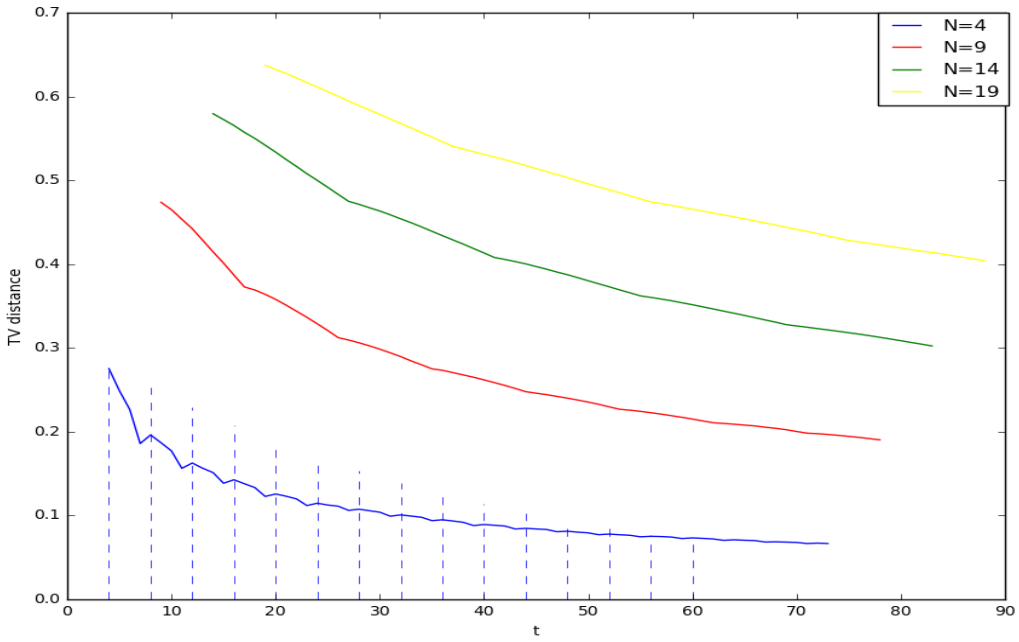
Zwróćmy uwagę na różnicę, która pojawia się przy dwóch sposobach oszacowania $\tau_{mix}(\epsilon)$:

- Generujemy M trajektorii procesu stochastycznego, dla każdej z tych trajektorii wyznaczamy $\tau^i_{mix}, i = 1{,}2{,}...{,}M$, a następnie obliczamy $\overline{\tau}_{mix} = \frac{1}{M}\sum_{i=1}^{M}\tau^i_{mix}$.
- Generujemy M trajektorii procesu stochastycznego, przy czym dla i-tej trajektorii mamy wyznaczoną funkcję odległości wahania całkowitego $d^i(t)$; następnie dla każdego t szacujemy średnią funkcję odległości wahania całkowitego $\hat{d}(t) = \frac{1}{M}\sum_{i=1}^{M} d^i(t)$. Teraz $\hat{\tau}_{mix} = \inf\{t > 0 : \hat{d}(t) < \epsilon\}$.

Z Rysunku **??** można odczytać, że różnica między $\hat{\tau}_{mix}$ i $\overline{\tau}_{mix}$ jest tym większa, im wykres funkcji odległości wahania całkowitego jest bardziej płaski. Powiązanie tej różnicy z wartością pochodnej funkcji $d(t)$ wydaje się bardziej uzasadnione niż wyjaśnianie jej bezpośrednio przez rozmiar grafu N. Powyższe estymatory τ_{mix} nie są więc takie same! Lepszym estymatorem t_{mix} wydaje się $\overline{\tau}_{mix}$, gdyż τ_{mix} wiąże się z pierwszym uderzeniem procesu $\delta(t)$ w barierę $\delta(t) = \epsilon$. Przypomnijmy, że histogramy τ_{mix} są asymetryczne i silnie prawoskośne. Z prawoskośności rozkładu wynika, że wartość oczekiwana zmiennej losowej pochodzącej z tego rozkładu jest większa niż jej mediana. Mała wartość $|d'(t)|$ w zadanym punkcie oznacza, że po uderzeniu procesu $\delta(t)$ w poziom a, proces ten z wyższym prawdopodobieństwem dłużej oscyluje jakiś wokół a niż w przypadku dużych wartości $|d'(t)|$.



Rysunek 3.13: Wykres oszacowanej funkcji odległości wahania całkowitego dla grafów cyklicznych o różnych rozmiarach N. Poziomą linią zaznaczono poziom $y = 0.25$. Pionowymi liniami przerywanymi zaznaczono odpowiednie oszacowania $\overline{\tau}_{mix}(0.25,N)$, natomiast pionowymi liniami kropkowanymi - oszacowania $\hat{\tau}_{mix}(0.25,N)$.



Rysunek 3.14: Wykres ilustrujący niemonotoniczność oszacowanej funkcji odległości wahania całkowitego $d(t-1)$ dla różnych grafów cyklicznych - powiększenie. Pionowymi liniami przerywanymi zaznaczono wielokrotności 4, czyli rozmiaru grafu $\mathbf{C}(4)$, do poziomu $y = 0.4e^{-t/10}$.

Na Rysunku **??** nie widać jeszcze niemonotoniczności, o której była mowa w sekcji dotyczącej grafów pełnych. Przypomnijmy, że zaobserwowaliśmy, iż estymowana funkcja $d(t)$ ma minima lokalne dla $t = 2N,3N,4N,....$ Zjawisko to zilustrowano na Rysunku **??**. Zauważmy, że dla błądzenia losowego w $\mathbb{Z}_4$ minima lokalne obserwowane są w punktach $t = 2 \cdot 4,3 \cdot 4,4 \cdot 4,....$ Naturalne wydaje się postawienie pytania, czy maksima lokalne funkcji $d(t)$ występują zawsze w punktach $kN+1$, $k \geqslant 2$. Jednocześnie zauważmy, że różnica $d(kN+1) - d(kN)$ maleje do zera, gdy $k \to \infty$ lub $N \to \infty$. Problem maksimów lokalnych zilustrujmy na przykładzie grafu $\mathbb{C}(3)$.

Przykład 3.1 *Rozważmy błądzenie losowe w $\mathbb{Z}_3$. W chwili $t = 2$ proces koniecznie odwiedził dwa z trzech stanów dokładnie raz, więc $d(2) = 2|\frac{1}{2} - \frac{1}{3}| + \frac{1}{3} = \frac{2}{3}$. W chwili $t = 3$ z prawdopodobieństwem $\frac{1}{3}$ proces odwiedza trzeci stan i z prawdopodobieństwem $\frac{2}{3}$ proces wraca do jednego z dwóch poprzednio odwiedzonych stanów. Mamy więc $d(3) = \frac{1}{3} \cdot 0 + \frac{2}{3} \cdot \left(\frac{1}{3} + 0 + \frac{1}{3}\right) = \frac{4}{9} < \frac{2}{3}$. Dla $t = 4$ z prawdopodobieństwem $\frac{1}{3}$ będziemy mieli konfigurację 2-2-0 i z prawdopodobieństwem $\frac{2}{3}$ konfigurację 2-1-1. Stąd $d(t) = \frac{1}{3} \cdot \frac{1}{3} + \frac{2}{3} \cdot \frac{2}{3} = \frac{5}{9} > \frac{4}{9}$, czyli minimum lokalne rzeczywiście istnieje w chwili $t = 3$.*

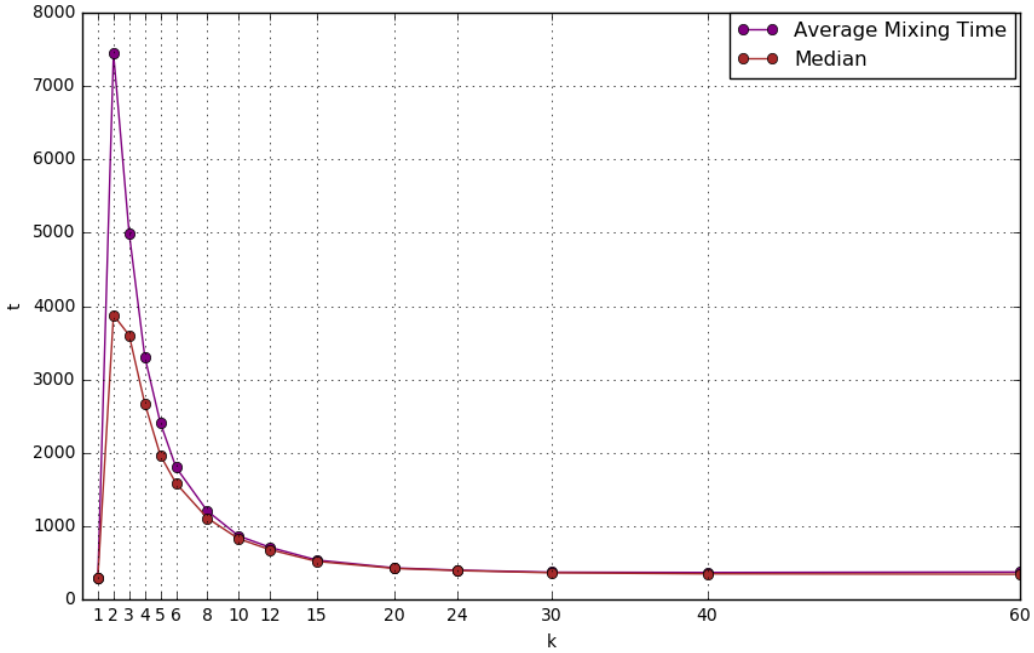
Heurystyka zagadnienia jest następująca: proces $\delta(t)$ może uderzyć w 0 wyłącznie w chwilach będących wielokrotnościami N. Tylko wówczas bowiem liczba odwiedzin w każdym ze stanów może być identyczna. Problem dla grafów pełnych można sformułować następująco.

Hipoteza 3.1 *Do $M \in \mathbb{N}$ urn wrzucono kolejno $kM + 1$ kul, $k \in \mathbb{N}$ w taki sposób, że jeśli w j-tym kroku wrzuciliśmy kulę do urny i-tej, to w następnym kroku wybieramy z równym prawdopodobieństwem urnę inną niż i-ta. W $kM - 1$-szym kroku w j-tej urnie znajduje się r_j kul, w kM-tym kroku m_j kul, natomiast w $kM + 1$-szym kroku n_j kul. Wtedy (dla odpowiednich M,k) zachodzi*

$$\mathbb{E}\left(\sum_{i=1}^{M} |\frac{r_i}{kM-1} - \frac{1}{M}|\right) \geqslant \mathbb{E}\left(\sum_{i=1}^{M} |\frac{m_i}{kM} - \frac{1}{M}|\right) \leqslant \mathbb{E}\left(\sum_{i=1}^{M} |\frac{n_i}{kM+1} - \frac{1}{M}|\right). \tag{3.1}$$

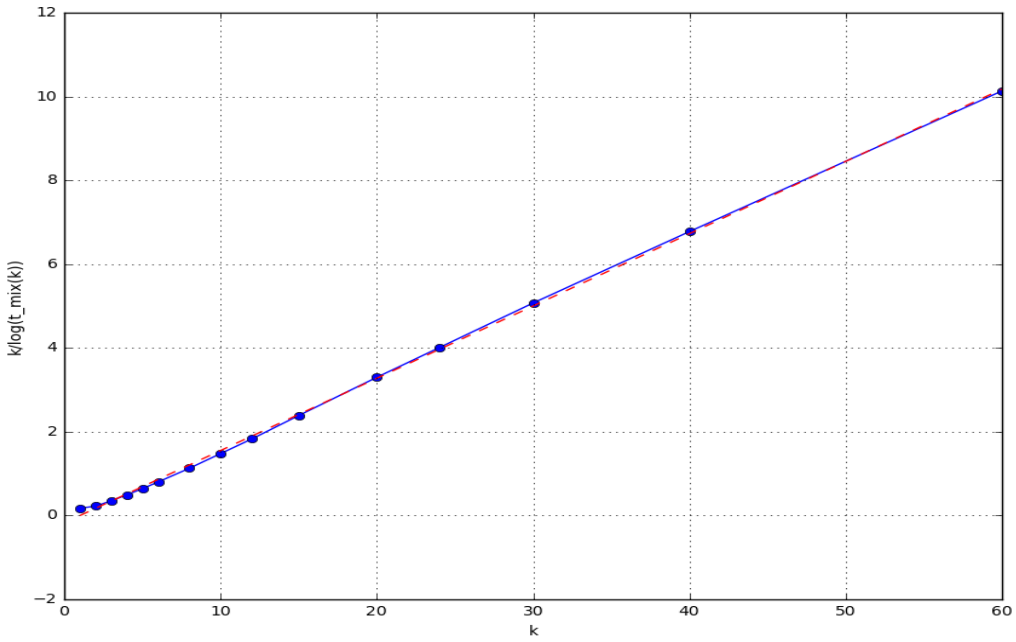
3.4.3 Archipelagi

Analizę archipelagów przeprowadzono w celu zbadania zależności czasu mieszania od drożności. Przyjrzyjmy się rozkładom średnich czasów mieszania dla $k \in \{1,2,3,4,5,6,8,10,12,15,20,24,30,40,60\}$.

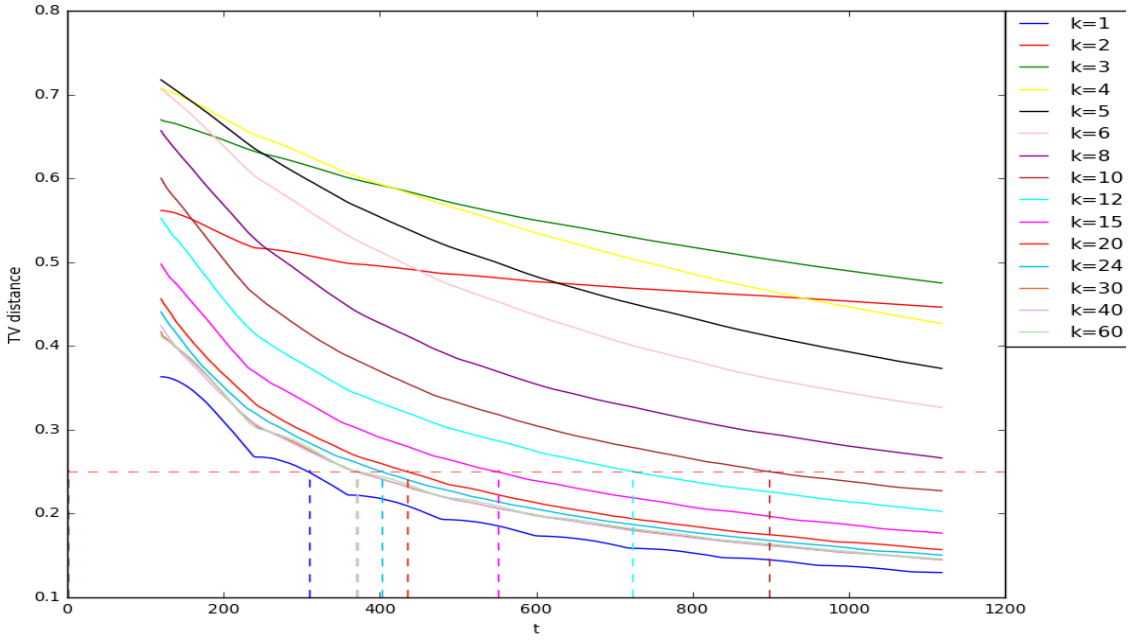


Rysunek 3.15: Wykres oszacowanych średnich czasów mieszania, $\epsilon = 0.25$, dla różnych archipelagów $N = 120$, $i = 1$

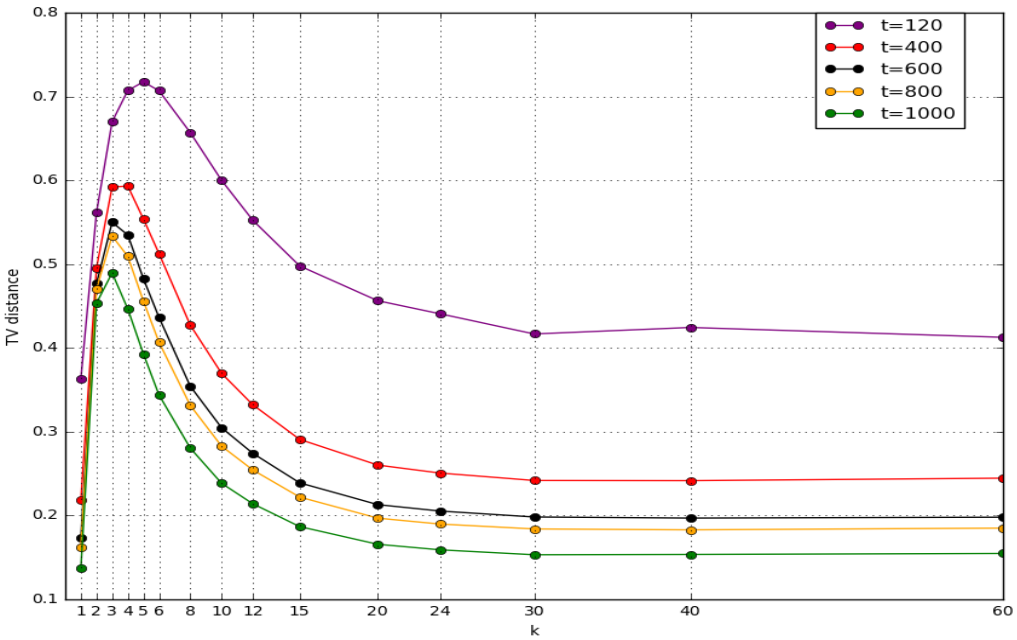
Wykres **??** pozwala porównać średnie czasy mieszania dla różnych archipelagów. Oszacowanie funckcji $h(k) = \mathbb{E}t_{mix:\epsilon=0.25}(k)$ ma kilka interesujących własności. Po pierwsze, dla $k \geqslant 2$ jest to funkcja nierosnąca. Po drugie, zgodnie z intuicjami maksimum globalne zostaje osiągnięte dla $k = 2$. Graf składa się wówczas z dwóch klik połączonych jedną krawędzią, a więc szansa przeskoczenia procesu między klikami jest najmniejsza. Po trzecie, stosunek $f(k) := \frac{k}{\log h(k)}$ jest prawie proporcjonalny. Prosta regresji $y(k) = 0.17k - 0.16$ wizualnie dopasowuje się dość dobrze do $f(k)$ (patrz Rysunek **??**).



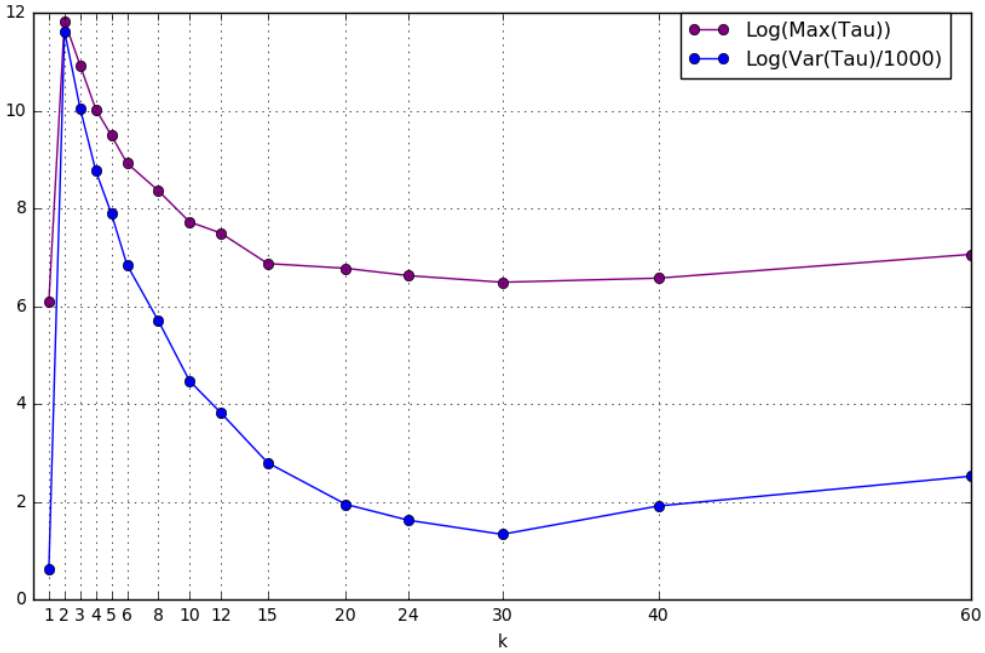
Rysunek 3.16: Wykres funkcji $f(k) = \frac{k}{\log(h(k))}$ archipelagów $N = 120$, $i = 1$ w zależności od k oraz prostej regresji, $y = 0{,}17k - 0.16$.



Rysunek 3.17: Wykres zależności średniej odległości wahania całkowitego od liczby skoków t dla archipelagów $A(120,k,1)$, gdzie $k \in \{1,2,3,4,5,6,8,10,12,15,20,24,30,40,60\}$. Pionowe odcinki poprowadzono w miejscach przecięcia wykresów odległości wahania całkowitego z prostą $y = 0.25$.



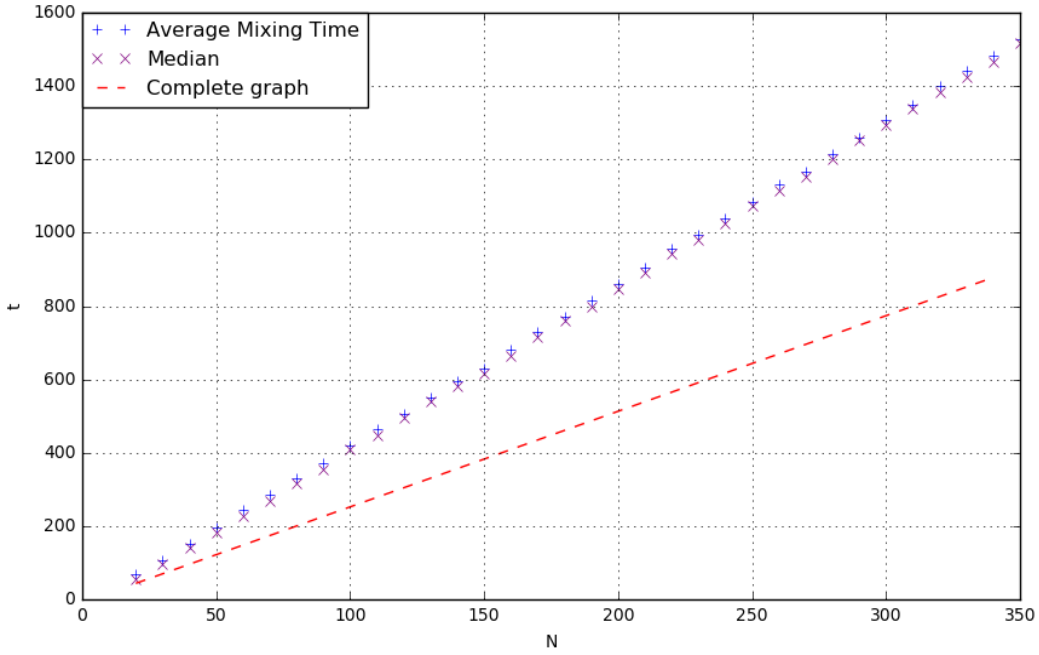
Rysunek 3.18: Porównanie średniej wartości odległości wahania całkowitego archipelagów $N = 120$, $i = 1$ w zależności od k dla trzech ustalonych chwil t.



Rysunek 3.19: Logarytmy wariancji podzielone przez 1000 i logarytmy maksimów próbkowych dla czasów mieszania na archipelagach w zależności od k, 1000 prób Monte-Carlo, $N = 120$.

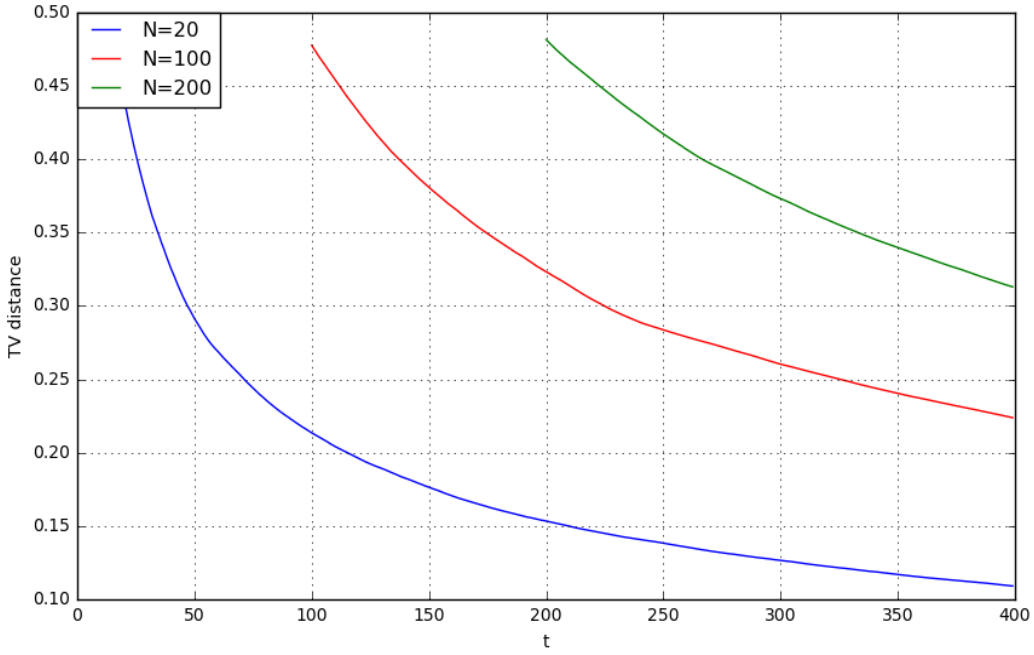
3.4.4 Model Barabási-Alberta

Badanie średniego czasu mieszania w zależności od rozmiaru klasycznej sieci Barabási-Alberta ujawniło, że zależność τ_{mix} od N dla $N \leqslant 350$ jest praktycznie liniowa (rys. **??**). Dopasowana prosta regresji spełnia równanie $y = 4.438x - 25.81$, współczynnik kierunkowy prostej jest więc wyraźnie większy niż w przypadku grafów pełnych, gdzie wynosi on ok. 2.607. Jest oczywiste, że $t_{mix}^{complete} \leqslant t_{mix}^{Barabasi} \leqslant t_{mix}^{\mathbb{Z}_N}$, gdzie $t_{mix}^{complete}$ jest czasem mieszania dla grafów pełnych, $t_{mix}^{Barabasi}$ jest czasem mieszania w rozpatrywanym modelu, a $t_{mix}^{\mathbb{Z}_N}$ jest czasem mieszania w błądzeniu losowym w $\mathbb{Z}_N$. Czas mieszania musi być zależny od średniej maksymalnej odległości między wierzchołkami, stąd zależność podkwadratową, którą otrzymaliśmy, można wyjaśnić intuicyjnie poprzez spostrzeżenie, że średnia odległość w sieciach bezskalowych z parametrem $\gamma \in (2,3)$ (u nas $\gamma = 3$) jest rzędu $\Theta(\log\log(N))$, natomiast średnica grafu rzędu $O(\log N)$ ([**?**]). Dla małych N funkcja $\log\log N$ jest prawie stała, więc model liniowy dopasowuje się bardzo dobrze. Należy jednak podkreślić, że dla bardzo dużych N na wykresie $t_{mix}^{Barabasi}(N)$ może pojawić się wyraźna wklęsłość.



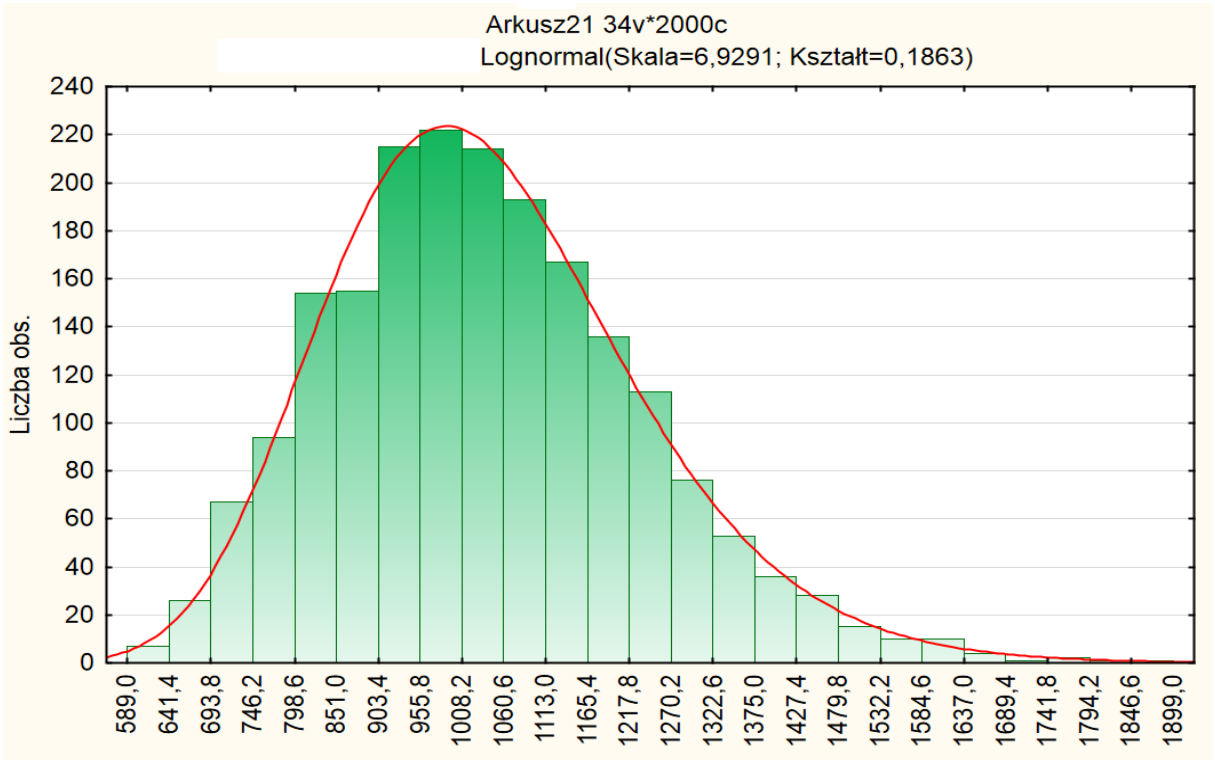
Rysunek 3.20: Oszacowania czasów mieszania $\epsilon = 0.25$ dla grafów losowych o różnych rozmiarach wygenerowanych za pomocą algorytmu Barabási-Alberta, 2000 prób Monte-Carlo. Linią przerywaną zaznaczono linię regresji dla grafów pełnych.

W porównaniu do wcześniej rozpatrywanych grafów, oszacowanie funkcji odległości wahania kwadratowego jest wygładzona (rys. **??**). Wyjaśnić to można sposobem przeprowadzania symulacji: w każdej próbie Monte-Carlo generowany jest nowy graf losowy, który posiada swój specyficzny czas mieszania, „rozkład stacjonarny" i charakterystykę TV-distance. Oszacowanie, które otrzymaliśmy, jest uśrednieniem takich specyficznych charakterystyk z wielu różnych grafów losowych o tym samym rozmiarze.

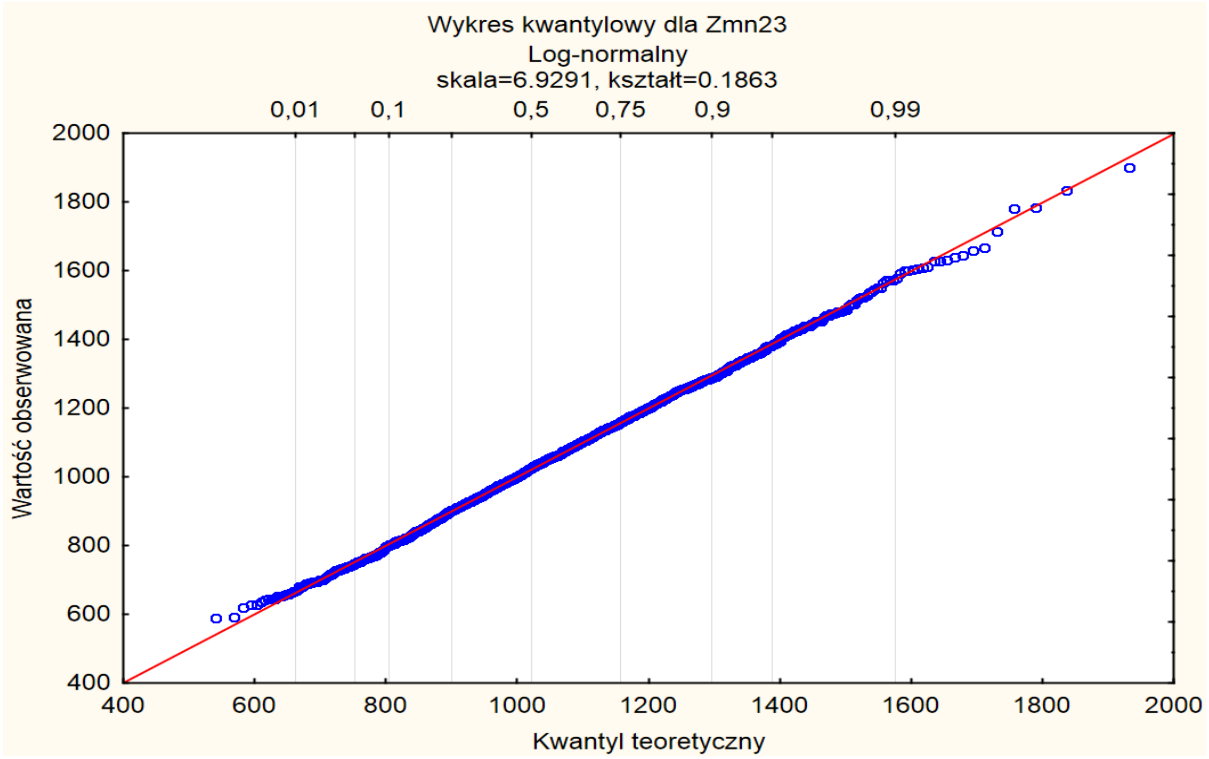


Rysunek 3.21: Oszacowanie funkcji odległości wahania całkowitego dla grafów losowych generowanych za pomocą algorytmu Barabási-Alberta, 2000 prób Monte-Carlo.

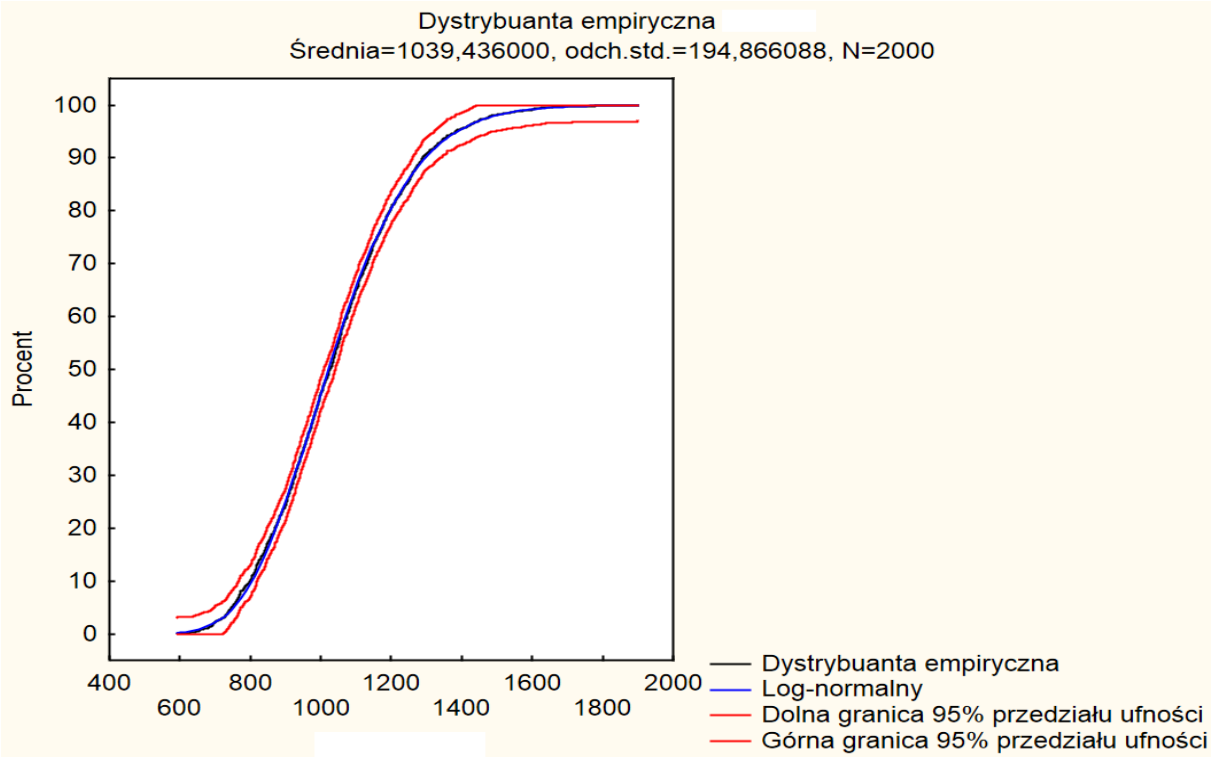
Na podstawie monotoniczności funkcji odległości wahania kwadratowego łańcucha Markowa od rozkładu stacjonarnego można postawić hipotezę, że rozkład τ_{mix} nie będzie wielomodalny. Rozkład τ_{mix} dla sieci bezskalowych $\gamma = 3$,$N = 240$ uzyskany w symulacjach niemal idealnie dopasowuje się do rozkładu log-normalnego z parametrem skali 6.9291 i parametrem kształtu 0.1863 (Rysunki **??**, **??**, **??**).



Rysunek 3.22: Histogram czasów mieszania grafów losowych o 240 wierzchołkach wygenerowanych za pomocą algorytmu Barabási-Alberta, 2000 obserwacji, z dopasowanym rozkładem log-normalnym.



Rysunek 3.23: Wykres kwantylowy czasów mieszania grafów losowych o 240 wierzchołkach wygenerowanych za pomocą algorytmu Barabási-Alberta, 2000 obserwacji, z dopasowanym rozkładem log-normalnym.



Rysunek 3.24: Dystrybuanta empiryczna czasów mieszania grafów losowych o 240 wierzchołkach wygenerowanych za pomocą algorytmu Barabási-Alberta, 2000 obserwacji, z dopasowanym rozkładem log-normalnym.

Aby zakończyć rozważania o rozkładzie τ_{mix}, przyjrzyjmy się tabeli 3.1, w której przedstawiono wyniki wybranych testów zgodności rozkładu próby z rozkładem normalnym. Na podstawie [**?**] oraz [**?**] wybrano testy Cramera-von Misesa (oparty o dystrybuantę empiryczną, czuły dla zniekształceń dla x bliskich μ, gdzie μ jest średnią w rozkładzie normalnym), Andersona-Darlinga, Shapiro-Wilka (oparty o statystyki pozycyjne, czuły na skośność i ciężkie ogony), Pearsona (oparty o statystykę χ^2, czuły na wielomodalność) i gładki test Neymana (czuły na zaburzenia masy dla x bliskich μ). W celu zbadania hipotezy o nieliniowości zależności $t_{mix}(N)$ przeprowadzono dodatkowo 60 prób Monte Carlo dla $N = 3000$ i $\epsilon = 0.25$; otrzymaną próbę oznaczmy przez $\mathtt{y}$. Przypomnijmy, że wyestymowana prosta regresji ma równanie $y = 4.438x - 25.81$, czyli, po ekstrapolacji, przy zależności liniowej oczekiwalibyśmy $t_{mix}(3000) \approx 13288.19$. Średnia z otrzymanej próby wyniosła 13472.1, natomiast mediana 13559 . Przypomnijmy, że jeśli zmienna losowa X ma rozkład log-normalny z parametrami μ i σ^2, to mediana tego rozkładu wyraża się wzorem e^{μ} oraz zmienna losowa $Y = \ln(X)$ ma rozkład normalny z parametrami μ i σ^2. Prosta regresji dopasowana do mediany rozpatrywanej próby spełnia równanie $y = 4.438358x - 39.64026$ ze współczynnikiem dopasowania modelu $R^2 > 0.999$. Na poziomie ufności 0.95 dla $N = 3000$ oczekiwana mediana powinna znaleźć się w przedziale (13236.46,13314.41). Fakt, że mediana próbkowa 13559 nie należy do tego przedziału, stanowi argument za tym, że $t_{mix}(n)$ może być funkcją ponadliniową. Przeprowadzimy teraz explicite test statystyczny, zakładając,

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Cramer-von Mises	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	0	0	1	1	0	1	1	1	0	0	0	0	0	1	1	1	1	1	0
Anderson-Darling	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	1	1	1	1	1	0
Shapiro	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0
Pearson	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	1	0	1	0	1	0	1	1	0	0	1	1	1	1	0	1
Neyman	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1	0	1	1	0	1	1	1	0	1	0	0	0	1	0	1	1	1	0

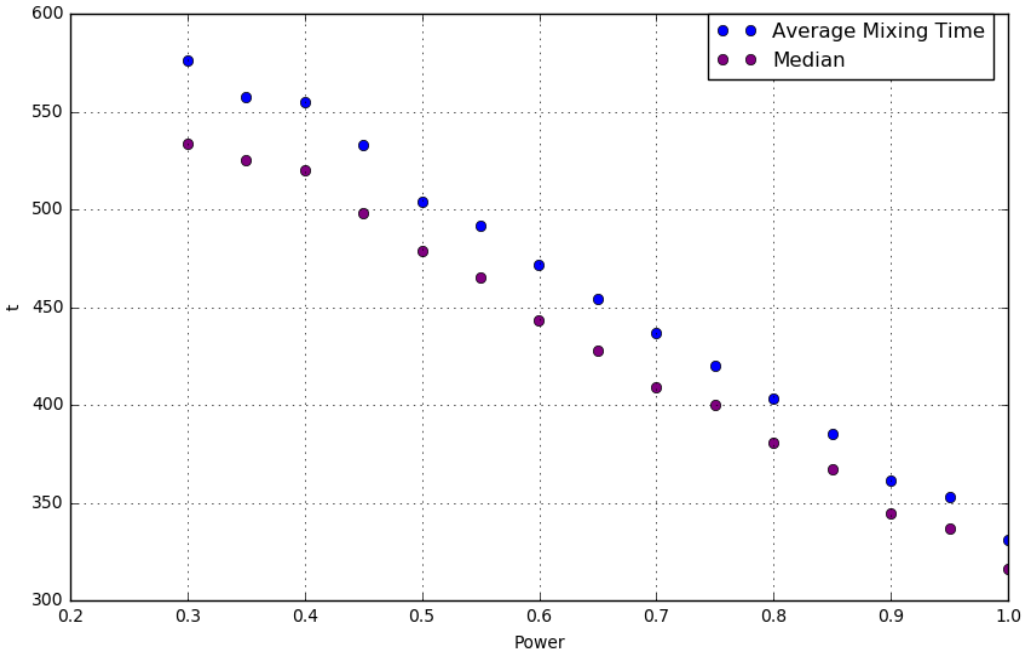
Tablica 3.1: Wyniki testów zgodności logarytmu z danych `barabasiMT.sta` z rozkładem normalnym. Nazwy kolumn oznaczają rozmiar grafu podzielony przez 10, natomiast wartości w tabeli informują o przyjęciu (1) lub odrzuceniu (0) hipotezy o normalności rozkładu na poziomie istotności 0.05.

że wszystkie `y` pochodzi z rozkładu log-normalnego. Będziemy testowali hipotezę jednostronną

$$\begin{cases} H_0: & \log \text{med}\tau_{mix}^{Barabasi}(\epsilon = 0.25, N = 3000) = \log(13275.43) \\ H_1: & \log \text{med}\tau_{mix}^{Barabasi}(\epsilon = 0.25, N = 3000) > \log(13275.43) \end{cases},$$

gdzie med$X$ oznacza medianę zmiennej losowej X. Przy teście t-Studenta dla $\alpha = 0.05$ uzyskujemy obszar krytyczny $[9.495551, \infty)$, średnią 9.507006 i p-wartość równą 0.02824. Oznacza to, że istnieją podstawy do odrzucenia hipotezy H_0. Tym samym uzyskujemy argument na rzecz hipotezy, że $t_{mix}(N)$ jest rzędu większego niż $O(N)$, czyli np. $O(N \cdot \log N)$. Dostępne moce obliczeniowe nie pozwalają jednak na wygenerowanie prób, na podstawie których dałoby się podać istotne statystycznie wnioski.

Przypomnijmy teraz algorytm **??**. Nowa krawędź między wierzchołkami i i j jest dodawana z prawdopodobieństwem $k_j^p / \sum_{n=1}^{j-1} k_n^\kappa$, gdzie k_i są aktualnymi liczbami krawędzi wychodzącymi z wierzchołka i. Dotychczas analizowaliśmy sytuację, w której $p = \kappa = 1$. Zgodnie z oczekiwaniami, im mniejsze p (przy stałym κ), tym średni czas mieszania okazuje się większy (Rysunek **??**). Wyjaśnić to można mniejszą liczbą wielokrawędziowych węzłów w grafie, co prowadzi do zwiększenia średniej średnicy grafu losowego.



Rysunek 3.25: Średnie czasy mieszania dla grafów rozmiaru $N = 80$ generowanych za pomocą algorytmu **??** z różnymi `powerNOM`

.

Implementacja symulatora

4.1 Opis wykorzystanych technologii

Przy wykonywaniu symulacji wykorzystano język Python 2 [?], w szczególności biblioteki math, matplotlib [?], networkx [?], numpy [?], pylab [?], random i scipy [?]. Do analiz statystycznych użyto języka R [?], w szczególności pakietów ddst [?], ggplot2 [?], normwhn [?], nortest [?], RColorBrewer [?], reshape2 [?] i xtable [?], oraz pakietu statystycznego Statistica 12. Języka R używano w środowisku RStudio [?], natomiast język Python był wywoływany w środowisku [?].

Język Python został wykorzystany ze względu na łatwość w implementacji grafów i narzędzia do obliczeń równoległych. Program Statistica został użyty do wstępnej analizy i obróbki wygenerowanych przez symulator danych. Pakiet R posłużył do bardziej zaawansowanych analiz i wizualizacji danych.

4.2 Opis symulatora

W niniejszej sekcji opisany zostanie projekt symulatora wcześniej omówionych łańcuchów Markowa oraz sposób przeprowadzania symulacji. Opisane funkcje znajdują się w pliku `libmix.py`.

Graf reprezentowany jest jako lista obiektów klasy `vertex` (**??**), czyli wierzchołków grafu.

```
'''Klasa reprezentująca wierzchołek grafu. Zawiera pola:
neighbours lista wierzchołków, z którymi połączony jest dany wierzchołek
name - unikalny numer wierzchołka
visits - liczba odwiedzin wierzchołka przez proces stochastyczny
Zawiera metodę: addNeighbour odpowiedzialną za dodawania nowych sąsiadów do listy neighbours'''
class vertex:
    neighbours = []
    name = 0
    visits = 0
    def addNeighbour(v):
        self.neighbours.append(v)
```

Rysunek 4.1: Klasa `vertex`

.

Graf tworzony jest poprzez tworzenie nowych obiektów klasy `vertex` i łączenie ich ze sobą krawędziami. Wierzchołek A jest połączony krawędzią z wierzchołkiem B, jeśli wierzchołek B znajduje się na liście A.`neighbours`.

4.2.1 Generowanie klik

Generowanie grafów pełnych rozmiaru N polega na stworzeniu N wierzchołków i połączeniu ich wzajemnie między sobą. Implementacja na listingu **??**.

```
'''Funkcja służąca do generowania klik rozmiaru N
Return: lista wierzchołków takich, że każdy wierzchołek posiada na swojej
liście neighbours wszystkie pozostałe wierzchołki'''
def fullGraph(N):
    vertices = []
    for i in range(N):
        vertices.append(vertex())
        vertices[i].neighbours = []
    for i in range(N):
        temp = vertices[:]
        temp.remove(vertices[i])
        vertices[i].neighbours.extend(temp)
        vertices[i].name=i
    return vertices
```

Rysunek 4.2: Funkcja `fullGraph`

.

4.2.2 Generowanie grafów cyklicznych

Generowanie grafów cyklicznych rozmiaru N polega na stworzeniu N wierzchołków i połączeniu wszystkich par wierzchołków, których $vertex.name$ to i oraz $i+1$, a także wierzchołków 1 i N. Implementacja na listingu **??**.

```
'''Funkcja generująca grafy cykliczne rozmiaru N. Tworzymy wierzchołki 1,2,3...,N,
a następnie łączymy między sobą wszystkie wierzchołki i,i+1 oraz wierzchołki 1,N.
'''
def circleGraph(N):
    vertices = []
    for i in range(N):
        vertices.append(vertex())
        vertices[i].neighbours = []
    for i in range(N):
        vertices[i].neighbours.append(vertices[(i+1)%N])
        vertices[i].neighbours.append(vertices[(i-1)%N])
        vertices[i].name=i
    return vertices
```

Rysunek 4.3: Funkcja `circleGraph`

.

4.2.3 Generowanie archipelagów

Zaprezentujemy implementację algorytmu **??** tworzenia archipelagów.

```
'''Funkcja służąca do tworzenia wysp w grafach typu archipelag.
Return: lista nOfIslands klik rozmiaru nOfVertices, które są połączone między dobą bridges krawędziami.'''
def islandGraph(nOfVertices,nOfIslands,bridges):
    islandSize=nOfVertices/nOfIslands
    islands = []
    print islandSize
    for i in range(0,nOfIslands):
        island = []
        for vert in range(0,islandSize):
            island.append(vertex())
        for vert in island: #wyspa jest grafem pełnym
            vert.neighbours = island[:]
            vert.neighbours.remove(vert)
        islands.append(island)
    #każda wyspa za pomocą mostów łączy się z innymi wyspami
    shift = 0
    for isl in range(0,nOfIslands):
        tmpIsl1 = islands[isl]
        for dest in range(isl+1,nOfIslands):
            shift = shift + 1
            tmpIsl2 = islands[dest]
            for i in range(0,bridges): #most jest jednokierunkowy
                tmpIsl1[(i+shift)%islandSize].neighbours.append(tmpIsl2[(i+shift)%islandSize])
                tmpIsl2[(i+shift)%islandSize].neighbours.append(tmpIsl1[(i+shift)%islandSize])
    return islands
```

Rysunek 4.4: Funkcja `islandGraph`

.

```
''' Funkcję przyjmująca jako argument listę wysp zwracaną przez funkcję islandGraph.
Zwraca listę wierzchołków, którą interpretuje się jako graf.
'''
def islands2graph(islands):
    graph = []
    rec = 0
    for isl in islands:
        for vert in isl:
            vert.name = rec
            rec = rec+1
            graph.append(vert)
    return graph
```

Rysunek 4.5: Funkcja `islands2graph`

.

4.2.4 Model Barabási-Alberta

Na Listingu **??** zaprezentowano realizację Algorytmu **??**.

```
''' Funkcja realizująca algorytm tworzenia grafów Barabasi-Alberta
N rozmiar grafu
'''
def barabasiGraph(N,function = 'standard',nomPower=1.0,denomPower=1.0):
    vertices = []
    for i in range(N):
        vertices.append(vertex())
        vertices[i].neighbours=[]
        vertices[i].name=i
    #inicjalizacja
    vertices[0].neighbours.append(vertices[1])
    vertices[1].neighbours.append(vertices[0])
    #dodawanie nowych krawedzi
    for i in range(2,N):
        while not vertices[i].neighbours: #dodatkowy warunek zmiejszający szansę na niespójność grafu
            for j in range(i):
                if function == 'standard':
                    deg=mt.pow(1.0*len(vertices[j].neighbours),nomPower)
                elif function == 'log':
                    deg = mt.pow(mt.log(1.0*len(vertices[j].neighbours)),nomPower)
                denominator = sum([ mt.pow(1.0*len(v.neighbours),denomPower) for v in vertices[0:i]])
                probability = 1.0*deg/denominator
                rnd = random.uniform(0,1)
                if rnd<probability:
                    vertices[i].neighbours.append(vertices[j])
                    vertices[j].neighbours.append(vertices[i])
    return vertices
```

Rysunek 4.6: Funkcja `barabasiGraph`

.

4.2.5 Obliczanie wartości odległości wahania całkowitego

Aby obliczyć wartość odległości wahania całkowitego od danego rozkładu, najpierw potrzebujemy wektora liczby odwiedzin w każdym z wierzchołków grafu. Wektor ten zwracany jest przez funkcję `graph2visits` zaimplementowaną na Listingu **??**.

```
'''Funkcja zamieniająca graf na tlistę z bieżącą liczbą odwiedzeń dla każdego wierzchołka'''
def graph2visits(graph):
    vis = []
    for vert in graph:
        vis.append(vert.visits)
    return vis
```

Rysunek 4.7: Funkcja `graph2visits`

.

```
'''Funkcja obliczająca Total Variation Distance między wektorem visits,
którego i-ty element reprezentuje liczbę odwiedzin przez proces w i-tym
wierzchołku.
'''
def TVdistanceB(visits):
    setSize = len(visits)#liczba wierzchołków
    totalMoves = sum(visits)#łączna liczba ruchów
    stationary = 1.0/float(setSize)
    distribution = [float(visits[i])/float(totalMoves) for i in range(setSize)]
    m = 0.0
    for i in xrange(setSize):
        m = m+(abs(distribution[i]-stationary))
    return m/2.0
```

Rysunek 4.8: Funkcja `TVdistanceB`

.

4.2.6 Symulowanie procesu stochastycznego

Dana jest lista obiektów klasy `vertex` połączonych między sobą w sposób opisany w poprzednich sekcjach. Z obiektów tych wybieramy w pierwszej iteracji losowy wierzchołek v. W drugiej iteracji pod v podstawiamy $v = \texttt{jump}(v)$, gdzie funkcja `jump` opisana jest na Listingu **??**.

```
'''Funkcja, która przyjmuje jako argument wierzchołek grafu i zwraca losowy wierzchołek z listy jego sąsiadów'''
def jump(vertex):
    return random.choice(vertex.neighbours)
```

Rysunek 4.9: Funkcja `jump`

.

Trajektorię procesu i czasy mieszania symuluje się za pomocą funkcji `mixingTime`, której kod znaujduje się na Listingu **??**.

```
'''Funkcja symulująca proces Markowa na grafie Graph i zwracająca czas mieszania'''
def mixingTime(graph,epsilon, resolution=1,plotRange=1000):
    N = 0 #liczba wykonanych skoków
    size = len(graph)
    vertex = graph[1] #nie ma większego znaczenia, od którego wierzchołka zaczniemy
    vertex.visits = vertex.visits + 1
    distances = [] #przechowuję tu dodatkowe statystyki - TVdist po N size,size+10,... krokach
    for i in range(size): #pierwszy przebieg pętli: co najmniej tyle, ile wynosi liczba wierzchołków
        vertex = jump(vertex) #skok do nastepnego wierzcholka
        vertex.visits = vertex.visits + 1 #zaktualizowanie liczby wizyt
        N = N+1 #aktualizacja liczby skokow
    visitsvec = graph2visits(graph)
    tvd = TVdistanceC(visitsvec)
    mixed = mixing(tvd,epsilon)
    while mixed == False: #dopóki łańcuch nie jest zmieszany, skacz, obliczaj i zapisuj zapisuj tvdistance
        if(N<size+plotRange): #tvdistance chcemy znac plotRange razy od N poczynajac
            distances.append(tvd)
        for i in range(resolution): #mniejsza dokładność, ale za to szybciej
            vertex = jump(vertex)
            vertex.visits = vertex.visits + 1
            N = N+1
        visitsvec = graph2visits(graph)
        tvd = TVdistanceC(visitsvec)
        mixed = mixing(tvd,epsilon)
    mT = N
    while N<size+plotRange: #po osiągnięciu czasu mieszania
        distances.append(tvd)
        for i in range(resolution): #mniejsza dokładność, ale za to szybciej
            vertex = jump(vertex)
            vertex.visits = vertex.visits + 1
            N = N+1
        visitsvec = graph2visits(graph)
        tvd = TVdistanceC(visitsvec)
        mixed = mixing(tvd,epsilon)
    clearGraph(graph)
    return mT,distances
```

Rysunek 4.10: Funkcja `mixingTime`

.

Funkcja `mixingTime` przyjmuje jako argumenty m.in. `graph` - graf jako listę długości N obiektów typu `vertex` i `epsilon` (typu float). Najpierw wybieramy pierwszy wierzchołek z listy (ze względu na to, że wszystkie grafy są prawie symetryczne lub generowane losowo, wybór ten nie ma znaczenia dla wyników) wykonujemy N skoków, wywołując funkcję `jump` i zwiększając wartość pola `vertex.visits` o 1 w każdym odwiedzonym wierzchołku. Następnie obliczamy wartość funkcji odległości wahania całkowitego od rozkładu

stacjonarnego (uwaga: w przypadku sieci bezskalowych trzeba najpierw wyestymować rozkład stacjonarny, podczas gdy w pozostałych przypadkach rozkładem stacjonarnym jest rozkład jednostajny) poprzez wywołanie funkcji `graph2visits` i przekazanie zwróconego wyniku do funkcji `TVdistance`. Sprawdzamy, czy wynik zwrócony przez tę ostatnią funkcję jest mniejszy niż `epsilon` i jeśli nie, to wchodzimy do pętli `while`, w której powtarzamy powyższe kroki tak długo aż odległość wahania całkowitego nie będzie mniejsza niż zadany `epsilon` i nie wykonamy minimalnej liczby skoków (parametr `plotRange`). Po każdym skoku obliczoną wartość $TV distance$ dodajemy do listy `distances`, dzięki czemu dostaniemy charakterystykę $TV distance(t)$. Zwracamy jednocześnie tę listę i obliczoną wartość τ_{mix} dla danej trajektorii procesu.

4.2.7 Symulacje Monte-Carlo

Przeprowadzenie N symulacji Monte-Carlo na grafie g w celu uzyskania oszacowania rozkładu $\tau_{mix}(\epsilon)$ i funkcji odległości wahania całkowitego na przedziale $[N,M]$ od rozkładu stacjonarnego polega na wywołaniu funkcji `monteCarlo` z Listingu **??** z parametrami `graph` $= g$, `steps` $= N$, `epsilon` $= \epsilon$ oraz `plotRange` $= M$.

```
'''Funkcja obsługująca przeprowadzanie steps prób Monte Carlo na grafie graph,
w celu oszacowania czasów mieszania dla epsilon i esymacji TVdistance na
przedziale [wielkość grafu,plotRange]
Zmienna resolution odpowiada za dokładność oszacowań - jeśli resolution=r, to
TVdistance będzie obliczana co r kroków, a nie w każdym kroku'''
def monteCarlo(graph,steps,epsilon,resolution=1,plotRange=1000):
    mTs = []
    function = [0]*plotRange
    for i in range(steps):
        clearGraph(graph)#zerujemy wizyty w grafie
        experiment = mixingTime(graph,epsilon,resolution)
        mTs.append(float(experiment[0]))
        function = [function[i] + experiment[1][i] for i in range(plotRange)]
    function = [float(x)/float(steps) for x in function]
    clearGraph(graph)
    return mTs,function
```

Rysunek 4.11: Funkcja `monteCarlo`

.

4.2.8 Przykładowe wywołania

Dane wygenerowane przez poniższe wywołanie zapisane są w plikach `daneBarabasiMT` oraz `daneBarabasiTVd`.

```
#obliczenia równoległe
def processInput(s,M,eps,nom=1.0,denom=1.0):
    result = lib.monteCarloB(s,M,eps,nom=nom,denom=denom)
    print(". mean=",np.mean(result[0]),", var=",np.var(result[0]), " median = ",np.median(result[0]))
    return result
```

```
maxRange=34
monteCarloSamples = 2000
epsilon=0.25
sizes = [10*n+20 for n in range(maxRange)]
simulationBarabasi=[0]*len(sizes)
```

```
from joblib import Parallel, delayed
import multiprocessing

num_cores = multiprocessing.cpu_count()
print num_cores
```

```
#tutaj zapisujemy wyniki dla grafów barabasi-alberta; obliczenia wykonywane są równolegle
results = Parallel(n_jobs=num_cores)(delayed(processInput)(s,monteCarloSamples,epsilon) for s in sizes)
```

Rysunek 4.12: Przykładowe wywołanie symulatora

.

Podsumowanie

Symulacje łańcuchów Markowa na klikach, grafach cyklicznych, archipelagach i sieciach bezskalowych pozwoliły na oszacowanie rozkładów zmiennych τ_{mix} i charakterystyk średniej wartości funkcji $\delta(t)$ w zależności od typu grafu, N (z pewnego zakresu), ϵ i kilku innych parametrów. W przypadku sieci bezskalowych nie ma podstaw do odrzucenia hipotezy, że τ_{mix} ma rozkład log-normalny, natomiast w pozostałych przypadkach wydaje się, że mamy do czynienia z mieszanką rozkładów log-normalnych. Do wykresu $\mathbb{E}\tau_{mix}(n)$ w przypadku grafów pełnych dobrze dopasowuje się funkcja liniowa, w przypadku grafów cyklicznych funkcja kwadratowa, natomiast w przypadku sieci bezskalowych wiele wskazuje na to, że $\mathbb{E}\tau_{mix}(n)$ rośnie nieznacznie ponadliniowo. W przypadku archipelagów stwierdzono, że nie ma prostej zależności między liczbą wysp, a τ_{mix}, czy $\delta(t)$.

Można postawić pytanie, czy $t_{mix}(N) = O(N \cdot k)$, gdzie k jest średnią odległością między wierzchołkami w grafie (lub średnicą grafu). W przypadku grafów pełnych mamy $t_{mix}(N) = O(1 \cdot N)$, zaś dla grafów cyklicznych $t_{mix}(N) = O(N \cdot N)$. Gdyby odpowiedź była twierdząca, to dla sieci bezskalowych $2 < \gamma < 3$ mielibyśmy zależność $t_{mix}(N) = O(\log N \cdot N)$.

Bibliografia

[1] R. Albert, A.-L. Barabási, *Statistical mechanics of complex networks*, 2002, Reviews of Modern Physics. 74 (1): 47–97,

[2] N. Berestycki, *Lectures on Mixing Times. A Crossroad between Probability, Analysis and Geometry*, http://www.statslab.cam.ac.uk/ beresty/teach/Mixing/mixing2.pdf, dostęp 20.03.2017,

[3] P. Biecek, *Czternaście wybranych testów normalności*, wpis na blogu www.smarterpoland.pl z 0.04.2013, dostęp 2.12.2017,

[4] P. Biecek, *Przewodnik po pakiecie R*, Oficyna Wydawnicza GiS, Wrocław 2014,

[5] P. Biecek, T. Ledwina, *ddst: Data Driven Smooth Tests* 2016, R package version 1.4, https://CRAN.R-project.org/package=ddst,

[6] H. Chuangpishit, J. Czyzowicz, L. Gasieniec, K. Georgiou, T. Jurdzinski, E. Kranakis, *Patrolling a Path Connecting a Set of Points with Unbalanced Frequencies of Visits*, arXiv preprint arXiv:1710.00466, 2017,

[7] F. Chung, L. Lu, *The average distances in random graphs with given expected degrees* , 2002, Proceedings of the National Academy of Sciences of the United States of America,

[8] A. Collins, J. Czyzowicz, L. Gasieniec, A. Kosowski, E. Kranakis, D. Krizanc, R. Martin, O. M. Ponce, *Optimal patrolling of fragmented boundaries*, Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures, 2013

[9] J. Czyżowicz, A. Kosowski, E. Kranakis, N. Taleb, *Patrolling Trees with Mobile Robots*, w: Cuppens F., Wang L., Cuppens-Boulahia N., Tawbi N., Garcia-Alfaro J. (eds) Foundations and Practice of Security. FPS 2016. Lecture Notes in Computer Science, vol 10128. Springer, Cham

[10] D. B. Dahl, *xtable: Export Tables to LaTeX or HTML*, 2016, https://CRAN.R-project.org/package=xtable,

[11] Y. Elor, A. M. Bruckstein, *Multi-A(ge)nt Graph Patrolling and Partitioning*, 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology ,

[12] L. Gąsieniec, T. Radzik, *Memory Efficient Anonymous Graph Exploration*, w: Broersma H., Erlebach T., Friedetzky T., Paulusma D. (eds) Graph-Theoretic Concepts in Computer Science. WG 2008. Lecture Notes in Computer Science, vol 5344. Springer, Berlin, Heidelberg,

[13] A. A. Hagberg, D. A. Schult, P. J. Swart, *Exploring network structure, dynamics, and function using NetworkX*, 2008, http://math.lanl.gov/ hagberg/Papers/hagberg-2008-exploring.pdf,

[14] J. D. Hunter, *Matplotlib: A 2D graphics environment*, Computing In Science & Engineering, vol. 9 nr 3, 2007,

[15] J. J. Hunter, *The Distribution of Mixing Times in Markov Chains*, arXiv:1111.0151, 2011,

[16] J. Gross, U. Ligges, *nortest: Tests for Normality*, 2015, https://CRAN.R-project.org/package=nortest,

[17] F. Pérez, B. E. Granger, *IPython: a System for Interactive Scientific Computing*, Computing in Science and Engineering, vol. 9 nr 3, 2007, http://ipython.org,

[18] J. Jakubowski, R. Sztencel, *Wstęp do teorii prawdopodobieństwa*, SCRIPT, Warszawa 2010,

[19] D.A. Levin, Y. Peres, E. L. Wilmer, *Markov Chains and Mixing Times*,

[20] E. Jones, T. Oliphant, P. Peterson e.a., *SciPy: Open source scientific tools for Python*, 2001-, http://www.scipy.org/,

[21] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org,

[22] E. Neuwirth, *RColorBrewer: ColorBrewer Palettes*, 2014, https://CRAN.R-project.org/package=RColorBrewer,

[23] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2017, https://www.R-project.org/,

[24] RStudio Team, *RStudio: Integrated Development Environment for R*, RStudio, Inc., Boston, MA, 2016, http:///www.rstudio.com,

[25] N. Taleb, *Coordinated Multi-Agents Patrolling Algorithms*, 2016,

[26] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*, Springer-Verlag New York, 2009, http://ggplot2.org,

[27] H. Wickham, *Reshaping Data with the reshape Package*, Journal of Statistical Software, vol. 21, nr 12, 2007, http://www.jstatsoft.org/v21/i12/,

[28] P. Wickham, *normwhn.test: Normality and White Noise Testing*, 2012, https://CRAN.R-project.org/package=normwhn.test